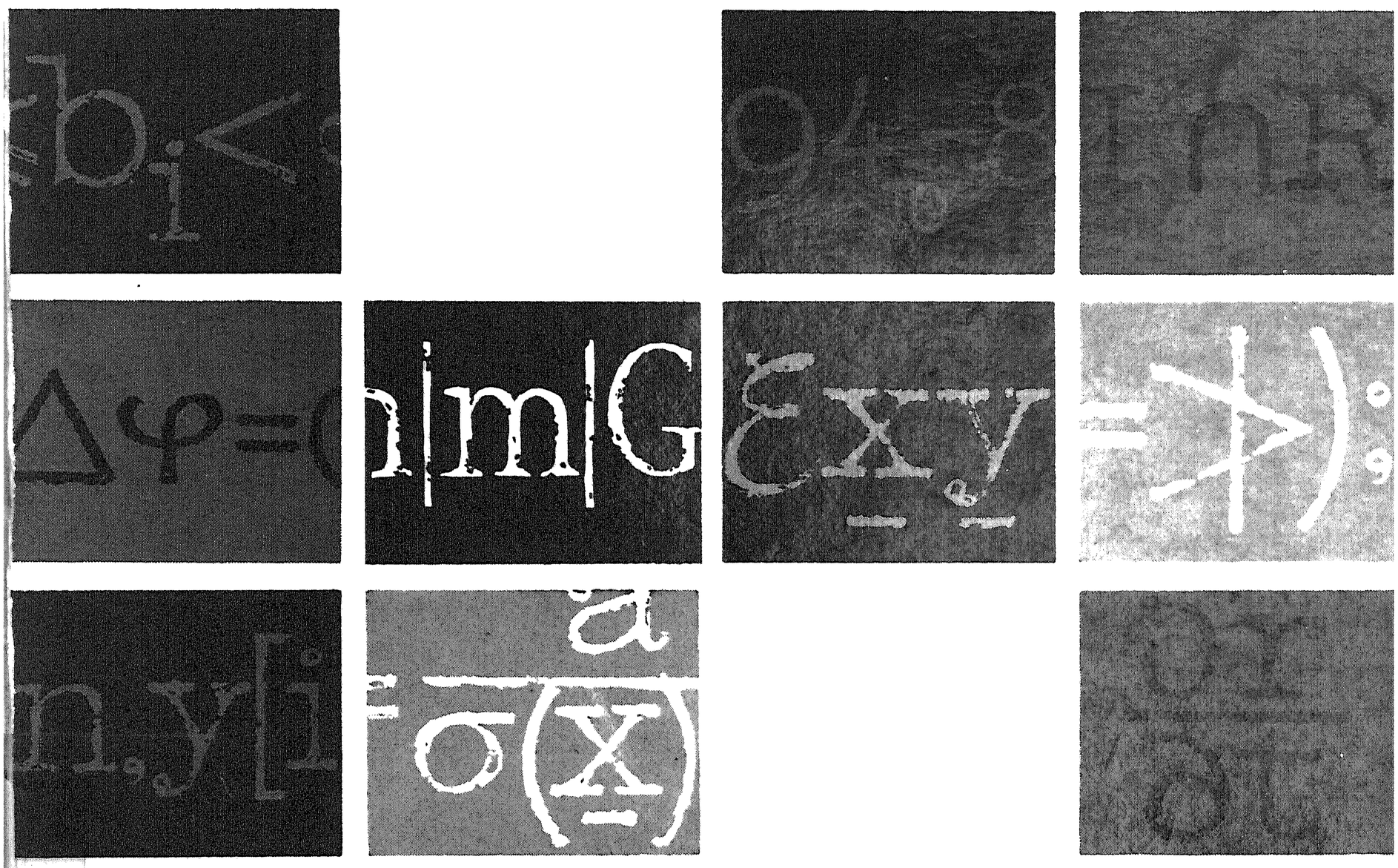
COMPUTATIONAL METHODS IN NUMBER THEORY

PART I

edited by H.W. LENSTRA, JR.
R. TIJDEMAN



MATHEMATICAL CENTRE TRACTS 154

MATHEMATICAL CENTRE TRACTS 154

COMPUTATIONAL METHODS IN NUMBER THEORY

PART I

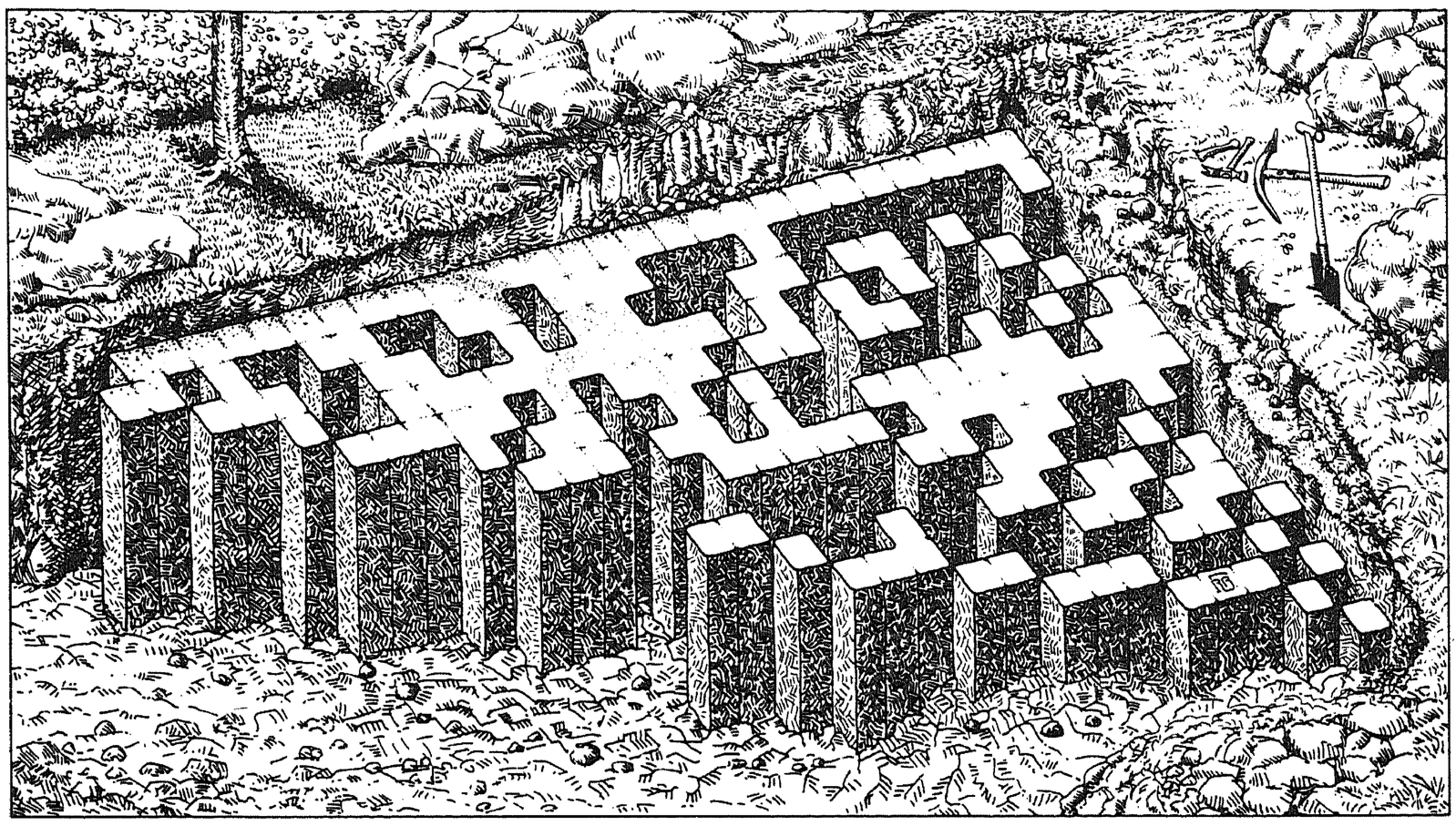
edited by

H.W. LENSTRA, JR.

R. TIJDEMAN

MATHEMATISCH CENTRUM AMSTERDAM 1982

ISBN 90 6196 248 X



CONTENTS PART I

CONTENTS PART II

PREFACE

A preliminary version of this tract appeared in 1980 under the title "Studieweek getaltheorie en computers". It contained the written versions of the lectures presented during the study week "Number theory and computers" that was held at the Mathematical Centre, September 1-5, 1980. The contents have been thoroughly revised for the present edition. We are happy to include Carl Pomerance's paper "Analysis and comparison of some integer factoring algorithms", which does not correspond to a lecture during the study week.

The editors are grateful to all those at the Mathematical Centre who have contributed to the technical realization of the tract.

H.W. Lenstra, Jr.

R. Tijdeman

ADDRESSES OF AUTHORS

R.P. BRENT : The Australian National University
Department of Computer Science
P.O. Box 4
Canberra ACT 2600
Australia

A.J. BRENTJES : Papiermolen 4
Molenwijk
2317 SV Leiden
The Netherlands

P.L. CIJSOUW : Technische Hogeschool Eindhoven
Onderafdeling der Wiskunde en Informatica
Postbus 513
5600 MB Eindhoven
The Netherlands

P. VAN EMDE BOAS : Universiteit van Amsterdam
Instituut voor Interdisciplinaire Wiskunde
Roetersstraat 15
1018 WB Amsterdam
The Netherlands

P.J. HOOGENDOORN : Stichting Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

A. KORLAAR : Technische Hogeschool Eindhoven
Onderafdeling der Wiskunde en Informatica
Postbus 513
5600 MB Eindhoven
The Netherlands

A.K. LENSTRA : Stichting Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

H.W. LENSTRA, JR. : Universiteit van Amsterdam
Mathematisch Instituut
Roetersstraat 15
1018 WB Amsterdam
The Netherlands

F.J. VAN DER LINDEN : Universiteit van Amsterdam
Mathematisch Instituut
Roetersstraat 15
1018 WB Amsterdam
The Netherlands

J. VAN DE LUNE : Stichting Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

C. POMERANCE : Department of Mathematics
University of Georgia
Athens, GA 30602
U.S.A.

H.J.J. TE RIELE : Stichting Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

R.J. SCHOOF : Rijksuniversiteit Leiden
Mathematisch Instituut
Postbus 9512
2300 RA Leiden
The Netherlands

R.J. STROEKER : Erasmus Universiteit Rotterdam
Econometrisch Instituut
Burgemeester Oudlaan 50
3062 PA Rotterdam
The Netherlands

R. TIJDEMAN : Rijksuniversiteit Leiden
Mathematisch Instituut
Postbus 9512
2300 RA Leiden
The Netherlands

J.W.M. TURK : Erasmus Universiteit Rotterdam
Econometrisch Instituut
Burgemeester Oudlaan 50
3062 PA Rotterdam
The Netherlands

M. VOORHOEVE : Philips Data Systems
Postbus 245
7300 AE Apeldoorn
The Netherlands

D.T. WINTER : Stichting Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

H. ZANTEMA : Universiteit van Amsterdam
Mathematisch Instituut
Roetersstraat 15
1018 WB Amsterdam
The Netherlands

INTRODUCTION

by

H.W. LENSTRA, JR.

This introductory lecture is devoted to a specific problem from computational number theory. The discussion will provide us with an opportunity to indicate which type of questions will be considered in the other lectures.

A classical theorem due to Fermat asserts that for every prime number p with $p \equiv 1 \bmod 4$ there exist integers x and y, unique up to order and sign, such that

$$p = x^2 + y^2.$$

For example, the prime factor $p = 1238926361552897$ of $2^{2^8} + 1$ discovered by BRENT and POLLARD [2] can be written as

$$p = 25515304^2 + 24246559^2.$$

How were these values determined? More generally, given p, how does one determine x and y in the most efficient way? That is the problem to be discussed in this lecture. Throughout p denotes a prime number that is $1 \bmod 4$.

DAVENPORT, in [5, Chapter V, Section 3], gives four methods of constructing x and y. Before we analyze their efficiency let us set ourselves a standard by first considering the trivial method. If we assume $x > y$ then $\sqrt{p/2} < x < \sqrt{p}$, so it suffices to test, for each x in this range, whether $p - x^2$ is a square. This takes time $O(p^{(1/2)+\varepsilon})$ for any $\varepsilon > 0$, the p^{ε} accounting for the arithmetic that must be done for each x; see Turk's lecture for a more precise analysis of the cost of arithmetic operations.

One of Davenport's constructions, due to Jacobsthal, is as follows. Let $(\frac{m}{p})$ denote the Legendre symbol, and choose $a,b \in \mathbb{Z}$ with $(\frac{a}{p}) = 1$, $(\frac{b}{p}) = -1$. Then the integers

$$x = \frac{1}{2}\sum_{n=1}^{p-1}\left(\frac{n(n^2-a)}{p}\right), \qquad y = \frac{1}{2}\sum_{n=1}^{p-1}\left(\frac{n(n^2-b)}{p}\right)$$

satisfy $x^2 + y^2 = p$. This can be proved by relating x and y to the number of solutions of each of the congruences

$$u^2 \equiv v^3 - av \bmod p,$$

$$u^2 \equiv v^3 - bv \bmod p,$$

see [6, Chapter 18, Theorem 5]. Using this construction for x and y in a straightforward way leads to an $O(p^{1+\varepsilon})$-algorithm, much slower than our standard.

Davenport's second construction is obtained by putting $a = -1$ in the above formula. Writing $p = 4k + 1$ and using that

$$\left(\frac{n(n^2+1)}{p}\right) \equiv (n^3+n)^{2k} \bmod p,$$

$$\sum_{n=1}^{p-1} n^i \equiv \begin{cases} 0 \bmod p & \text{if } i \not\equiv 0 \bmod p-1 \\ -1 \bmod p & \text{if } i \equiv 0 \bmod p-1 \end{cases}$$

one readily finds that

$$x \equiv -\frac{1}{2}\binom{2k}{k} \bmod p,$$

as was first proved by Gauss. Together with $|x| < \frac{1}{2}p$ this suffices to determine x and hence y. Calculating $\binom{2k}{k} \bmod p$ in the trivial way we arrive again at an $O(p^{1+\varepsilon})$-algorithm. Using the technique described in Section 4 of Pomerance's paper we can reduce this to $O(p^{(1/2)+\varepsilon})$, exactly our standard but much slower in practice. At the end of the author's lecture on primality testing it will be seen that there is a much faster way to calculate $\binom{2k}{k} \bmod p$ if arithmetic operations on ordinary integers are assumed to be doable in unit time. But the size of the numbers that appear is such that this is a very unrealistic assumption; in the terminology of the lecture by Van Emde Boas we are using the wrong *machine model*.

The third method that we discuss is basically due to Legendre. Davenport formulates it in terms of continued fractions, but here we shall use quadratic forms. Define two sequences of integers $a_0, a_1, \ldots, b_0, b_1, \ldots$

as follows:

$$a_0 = 1,$$

$$b_0 = \text{greatest odd integer} < \sqrt{p},$$

$$a_{n+1} = (b_n^2 - p)/(4a_n),$$

$$b_{n+1} \equiv -b_n \bmod 2a_{n+1}, \quad \sqrt{p} - |2a_{n+1}| < b_{n+1} < \sqrt{p}.$$

For some n it will happen that $a_{n+1} = -a_n$, and then we have $(2a_n)^2 + b_n^2 = p$. For example, for p = 73 we have

n :	0	1	2	3	4	5
a_n:	1	-6	2	-3	4	-4
b_n:	7	5	7	5	3	

$73 = (2a_4)^2 + b_4^2 = 8^2 + 3^2$.

From Schoof's lecture it will be clear that the forms $F_n = a_n X^2 + b_n XY + a_{n+1} Y^2$ are precisely the binary quadratic forms of discriminant p in the *principal cycle*. It can be shown that the length ℓ of this cycle is 2 mod 4, and that $a_{n+1} = -a_n$ occurs first for $n = (\ell-2)/4$. The known estimate $\ell = O(p^{(1/2)+\varepsilon})$ thus implies that this is again an $O(p^{(1/2)+\varepsilon})$-algorithm. But Shanks' technique of jumping through the principal cycle, explained by Schoof, improves this significantly: the desired form $F_{(\ell-2)/4}$ can be found in time $O(p^{(1/4)+\varepsilon})$, and if the generalized Riemann hypothesis is assumed even in time $O(p^{(1/5)+\varepsilon})$. In several other contributions we shall encounter algorithms in which the Riemann hypothesis plays a role. In the paper of Brent *et al.* attention is paid to numerical techniques related to the Riemann hypothesis.

The sequences (a_n), (b_n) defined above can also be used to solve the Pell equation

$$x^2 - py^2 = -4.$$

More general equations such as

$$ax^n + by^m = c$$

are considered, from different angles, in the contribution of Stroeker and Tijdeman.

In Schoof's lecture it is explained how binary quadratic forms can be used to determine the class number and the units of a quadratic field. In the lectures by Brentjes and Zantema the same questions are considered for number fields of higher degree.

The fourth method discussed by Davenport is due to Serret, and again we give a slightly different formulation, as in [3]. If $p = x^2 + y^2$ then $u = xy^{-1}$ (division mod p) satisfies $u^2 \equiv -1 \bmod p$, and up to sign it is the only such integer modulo p. Suppose now that, conversely, an integer u is given such that $u^2 \equiv -1 \bmod p$. We claim that it is easy to recover x and y. One method to do this is by calculating the greatest common divisor of p and u+i in the ring $\mathbb{Z}[i]$ of Gaussian integers. This can be done by means of the Euclidean algorithm, which is valid in this ring, and the result is

$$\gcd(p,u+i) = x + yi$$

where $x,y \in \mathbb{Z}$ are such that $x^2 + y^2 = p$.

The second method to recover x and y from u employs the Euclidean algorithm only for ordinary integers. It proceeds as follows. Calculate the gcd of p and u by means of the ordinary Euclidean algorithm, until two consecutive remainders are less than $\sqrt{p}$; then these can be taken as x and y. Example: for p = 73 we have $u^2 \equiv -1 \bmod p$ for u = 27, and the sequence of successive remainders is

$$73,27,19,8,3,\ldots$$

so that we can take x = 8, y = 3. The proof of the correctness of this algorithm depends on the symmetry appearing in the sequence of congruences

$$\begin{aligned}
0.27 &\equiv 73 \bmod 73\\
1.27 &\equiv 27 \bmod 73\\
-2.27 &\equiv 19 \bmod 73\\
3.27 &\equiv 8 \bmod 73\\
-8.27 &\equiv 3 \bmod 73\\
19.27 &\equiv 2 \bmod 73\\
-27.27 &\equiv 1 \bmod 73\\
73.27 &\equiv 0 \bmod 73.
\end{aligned}$$

This symmetry is caused by the next-to-last congruence $-u.u \equiv 1 \bmod p$.

This construction of x and y has a geometric interpretation: the pair (x,y) is a "short" vector in the two-dimensional lattice $\{(v,w) \in \mathbb{Z} \times \mathbb{Z}: v \equiv uw \bmod p\}$. For a method to find short vectors in higher dimensional lattices and an application to computational number theory we refer to [7]. The subject is closely related to diophantine approximation, as discussed in Brentjes' lecture.

How fast is the above method to construct x and y? The Euclidean algorithm takes time $O((\log p)^2)$, or in a faster version [8] only $O(\log p\ (\log\log p)^2\ \log\log\log p)$. But to this the time needed to find u should be added.

This leads to the question how the equation $u^2 \equiv -1 \bmod p$ can be solved. For the prime divisor $p = 1238926361552897$ of $2^{28}+1$ we can clearly take $u = 2^{2^7}$, and from this the values for x and y stated at the beginning can be easily computed. For general p we can take $u = (\frac{1}{2}(p-1))!$, but this formula is useless for computational purposes.

A.K. Lenstra discusses in his lecture a method to find zeros of polynomials over finite fields. Applying this to the polynomial X^2+1 over the field $\mathbb{Z}/p\mathbb{Z}$ we obtain a solution for our problem that is quite efficient in practice, but for which it is difficult to estimate the time needed in a satisfactory way.

The following method has a similar problem. Let b be the least positive integer with $(\frac{b}{p}) = -1$; then $b^{(p-1)/2} \equiv -1 \bmod p$, so we can take $u \equiv b^{(p-1)/4} \bmod p$. Using the reciprocity law for the Jacobi symbol one can calculate $(\frac{n}{p})$ in time $O((\log p)^2)$, for $0 < n < p$; perhaps this can be improved with the techniques of [8]. Further, $b^{(p-1)/4} \bmod p$ can be calculated in time $O((\log p)^{2+\varepsilon})$. Hence u can be determined in time $O(b(\log p)^2 + (\log p)^{2+\varepsilon})$; here we have $b = O(p^{1/(4\sqrt{e})+\varepsilon})$ (see [4]), and if the truth of the generalized Riemann hypothesis is assumed then $b = O((\log p)^2)$ (see [1]).

We conclude that Serret's method to solve $p = x^2 + y^2$ takes time $O(p^{1/(4\sqrt{e})+\varepsilon})$, where $1/(4\sqrt{e}) = 0.15163...$, and $O((\log p)^4)$ if the generalized Riemann hypothesis is true.

An improvement of theoretical value was recently obtained by SCHOOF [9], who showed without any unproved assumption that $p = x^2 + y^2$ can be solved in time $O((\log p)^6)$. His algorithm makes use of the elliptic curve $u^2 = v^3 - v$ (over $\mathbb{Z}/p\mathbb{Z}$) that we mentioned in connection with Jacobsthal's construction. It proceeds by investigating the action of the "Frobenius automorphism" on the ℓ-torsion points of the curve, for several small primes ℓ.

It may be expected that Schoof's algorithm is only the first of many applications of arithmetical algebraic geometry to computational number theory.

REFERENCES

[1] ANKENY, N.C., *The least quadratic non residue*, Ann. of Math. 55 (1952), 65-72.

[2] BRENT, R.P. & J.M. POLLARD, *Factorization of the eighth Fermat number*, Math. Comp. 36 (1981), 627-630.

[3] BRILLHART, J., *Note on representing a prime as a sum of two squares*, Math. Comp. 26 (1972), 1011-1013.

[4] BURGESS, D.A., *The distribution of quadratic residues and non-residues*, Mathematika 4 (1957), 106-112.

[5] DAVENPORT, H., *The higher arithmetic*, Hutchinson, London, 1968.

[6] IRELAND, K. & M. ROSEN, *A classical introduction to modern number theory*, Springer, New York, 1982.

[7] LENSTRA, A.K., H.W. LENSTRA, JR., & L. LOVÁSZ, *Factoring polynomials with rational coefficients*, Math. Annalen 261 (1982), 515-534.

[8] SCHÖNHAGE, A., *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Inform. 1 (1971), 139-144.

[9] SCHOOF, R.J., in preparation.

MACHINE MODELS, COMPUTATIONAL COMPLEXITY AND NUMBER THEORY

by

P. VAN EMDE BOAS

1. INTRODUCTION

The theory of machine models has established itself among the basic ingredients of the contemporary Computer Science curriculum. For the present occasion it is not my intention to present a rigid and formal survey of this field; such task requires about a term of teaching. Instead I like to sketch some of the basic ideas from this field, in order to convince the readers of this tract, large part of which are not Computer Scientists, that there exists a rigid mathematical formalism which enables us to discriminate between problems which are "easy" and other problems which are "hard" in a technical, well defined sense. I also like to indicate which particulars of Number Theory enable the mathematical world to retrieve various types of logical problems disguised as number theoretical ones.

The concept of effective computability ranks among the greater achievements of mathematics during the first half of this century. A typical question which has motivated the development of such a concept was Hilbert's tenth problem, asking for an "effective" decision procedure for testing whether a given Diophantine equation is solvable or not. Had the answer been positive, any rough description of an "algorithm" for deciding solvability of Diophantine equations would have satisfied the interested mathematicians, even if it would have been clear from this description that the eventual use of this algorithm for a real life instance of a Diophantine equation, though "effective", would have consumed ten times the average lifetime of its user. Proving the answer to be negative, as has been done by Matijasevic in 1970, requires that one first decides which procedures are considered to be "effective" before one can show that among these effective procedures there exist none that solve Hilberts problem.

Surprisingly the first presentations of the concept of effective computability as we know it today, did not arise in the context of computation

but rather of manipulation of formal objects. Church provided a representation of computable functions using terms in the lambda calculus. Kleene used a representation as the smallest class of functions containing the successor function, the constant function zero, and projection functions which is moreover closed under substitution, primitive recursion and unbounded minimalisation. This representation, yielding the so-called class of (general) *recursive functions*, stands closer to mathematical intuition what computation should involve; it is however far from clear that the class of functions could not be extended using other means of formal manipulation.

For a history of the creation of these formalizations, written by one of the original authors, I refer to KLEENE [20]; this survey has provoked a reply by M. DAVIS [11], from which it follows that a few more years should be added to the history of the present concept of effective computability: E. Post developed his formalization dealing with string manipulation (which gets close to our present day formal language theory) already in the early twenties of this century.

The formalizations mentioned above, all deal with effective procedures rather than with machines. In Turing's formalism, presented in 1936 [45], a machine is introduced performing the formal manipulations involved. The machine consists of a finite program which reads and prints symbols on a (potentially two-way infinite) tape; for a philosophical justification of this model as representing the "mathematician working with pencil and paper" see Turings original paper.

Many years later, after the contemporary computer had been invented, Shepherdson and Sturgis introduced the *Register machine*, which be now becomes more and more popular as a vehicle for teaching the introductary courses in Recursion and Computation theory. In its weakest form this machine is also known as the Minsky Machine [29]. Still another model is the so-called *storage modification* or *Pointer machine* which manipulates directed graph-like structures [38]. Originally those models were intended to compute on familiar structures like the natural numbers, or, essentially equivalent, strings of symbols from a finite alphabet. Over the recent years computation theory has yielded also the foundations for a theory of effective computation over other structures (see e.g. [12]), but this subject goes beyond the scope of this paper.

Examples of text books on the material covered by the first seven sections of this paper are [15] and [25].

A few words concerning terminology are in order. The present paper deals with a subject located on the borderline between recursion theory, computation theory and number theory; experience shows that some basic terminologies in these various branches of mathematics don't match, resulting in confusion for the untrained reader.

In the first place there is the choice of the domain on which functions are defined and the objects on which computations are supposed to compute. Both in the Post and in the Turing model this domain consists of *strings* over a *finite alphabet* Σ, whereas in the Kleene formalism the objects are the natural numbers (including the number zero) which set is called the set of *integers*. The register machines again compute on these integers. Now clearly it is possible to encode numbers by strings over a finite alphabet provided the alphabet has at least two different symbols. It seems irrelevant whether this encoding is based upon *unary encoding* (where the number k is encoded by a string of k+1 occurrences of the symbol 1) or whether one uses a system like *binary* or *decimal notation*, but this choice becomes relevant as soon as complexity arguments are involved. This has led to the situation that a large part of literature in recursion theory is inclined to models based upon unary notation, whereas the more recent textbooks with an emphasis on complexity theory are basically directed to binary notation, having as the basic objects strings rather than numbers. In the present paper a "switch" from numbers as the basic object to strings occurs in Section 6; this shift of attention is connected to the fact that the more abstract theory of the first five sections belongs to the area of recursion theory, whereas complexity theory takes over in the definition of the fundamental complexity classes.

Another source of confusion is the use of the words "set", "language" and "problem" for essentially the same object: a collection of strings over a finite alphabet. The term "language" arises in the area of formal language theory where one studies mechanisms for *generating*, *accepting* or *recognizing languages*. The term "problem" which has been propagated in complexity theory is inspired by the situation where the strings involved are the *encodings of instances* of some *problem* in mathematics, logic or operations research, and where the *set-recognition problem* amounts to the recognition of those instances of the given problem which are *solvable* or *feasible*.

A final trouble spot is the word "function". In recursion theory a function f does not need to be defined for each possible argument x in the set from which its arguments are drawn; such a function is called a *partial* function; it is called *total* in case the value is always defined. In the

situation where the function arises as the result of the computation of some machine the fact that f(x) is defined amounts to saying that the computation on input x *terminates* or *converges*, whereas undefinedness of f(x) corresponds to a *non-terminating* or *diverging* computation. The set of those arguments for which the function is defined is called the *domain* of the function; in general the domain is a subset of the set from which the arguments for the function are selected.

In the context of this paper a function always is supposed to be *effectively computable* by one of the machine models under discussion. So unless stated otherwise the words "function", "computable function" and "effectively computable function" mean the same. Amother phrase used for this concept is "recursive function"; originally the recursive functions are the functions computable in the Kleene formalism but the proven equivalence of all formalisms, combined with the acceptance for every day life of *Church's Thesis* (which states that the recursive functions coincide with the effectively computable ones) has turned these phrases into equivalent ones.

In the sequel we also will use the word "recursive transformation" for a recursive function in the context where the arguments of the function are not numbers for their own sake but numbers which are used as *indices of programs*; the transformation which as a mathematical object seems to be nothing but a function is supposed to "do something useful" with these programs.

2. THE TURING MACHINE MODEL

In the *Turing machine model* in its simplest form, one considers a machine which operates under control of a finite program on a two way infinite tape. On this tape symbols are written from a finite alphabet Σ; the tape consists of tape squares which are linearly ordered like the integers. During each stage of the computation the machine has visited only a finite number of tape cells, and initially only a finite number of tape cells carry information; those cells which are not yet visited and were not written on in the initial state carry a special symbol, called *blank*, which represents the absence of information. (Alternatively the tape is always finite, but extended by blank cells whenever needed).

The *finite control* of the machine consists of a *program* containing *quintuples* of the form (q,s,q',s',M), where q and q' are elements from a finite set K whose elements are called *states*, s and s' are symbols in Σ and M is a "move" which equals either R, 0 or L representing "move right",

"don't move", or "move left" respectively. The intended meaning of the above quintuple reads: "if in state g you are reading symbol s then go to state q', overwriting s by s' and move at most one square along the tape as indicated by M". Next another *instruction* (quintuple) can be executed.

Initially, the machine is started in a selected *initial state* q_0 looking at the leftmost *tapecell* of a consecutive block of (presumably nonblank) cells, representing the *input*. Subsequently instructions present in the program are executed as long as there are instructions present for the current combination of state and symbol read; if no such instruction can be found in the program the machine *halts*, and the computation is complete. If there exists such an instruction it is supposed to be unique.

The following example may illustrate the above description. Let the program of a machine consist of the following quintuples:

$(q_0,0,q_0,0,R)$,
$(q_0,1,q_0,1,R)$,
(q_0,B,q_1,B,L),
$(q_1,0,q_f,1,0)$,
$(q_1,1,q_1,0,L)$,
$(q_1,B,q_f,1,0)$.

If started on the leftmost bit of a binary number, the machine in state q_0 will look for the right end of this number; subsequently in state q_1 it replaces the maximal rightmost consecutive block of 1's by an equally sized block of zeros, replacing the zero or blank to the left of this block by a one and halting there in state q_f. If interpreted as a binary number this machine computes the successor function.

A typical computation of the machine can be represented by a sequence of so-called *instantaneous descriptions* like:

$$q_0 1011,\ 1q_0 011,\ 10q_0 11,\ 101q_0 1,\ 1011q_0 B,\ 101q_1 1B,$$

$$10q_1 10B,\ 1q_1 000B,\ 1q_f 100B.$$

How are these machines supposed to be used? It turns out that there exist three completely different ways of using the machine.

1) Given the input (expressed in the tape alphabet of the machine) the machine is started in its designated initial state reading the leftmost symbol of its input. After the machine halts (if it does so) the resulting

tape contents are read and interpreted as being the *result* of the computation. According to this interpretation the machine computed a *partial function*, i.e., the machine does not have halt on all inputs.

2) The machine is started in the same manner as described for 1) but it is now assumed that the machine halts for every input. Moreover, in the final configuration only one specific property of this configuration (for example the state in which the machine halts, or the fact whether the final contents of the tape represent zero or not) decides on whether the computation is considered to be an *accepting* or a *rejecting* one. According to this interpretation the machine recognizes a *language* of input strings, namely those inputs which yield an accepting computation.
3) Again the machine is started on the input, but in this case the language accepted by the machine consists of those input strings on which the machine halts.

Clearly, in interpretation 3 the fact that an input is rejected cannot be observed within a finite number of steps. As a consequence the class of languages accepted under interpretation 3) (called the *Recursively Enumerable* sets) is not necessarily the same as the class of languages recognized under interpretation 2) (called the *Recursive* sets); It should be clear however that the Recursive sets form a subclass of the Recursively Enumerable ones, for it is easy to modify the machine in such a way that every rejecting computation is replaced by a diverging (nonterminating) one. The converse is in general not possible as we shall see in the sequel.

3. OTHER MACHINE MODELS AND THEIR RECURSIVE EQUIVALENCE

It is easy to invent a thousand and one different ways of extending the Turing machine model. One can increase the number of tapes, separate different functions on different tapes (like a read-only, one-way input tape; a write-only, one-way output tape, and a collection of read/write two-way work tapes, as is done in what is considered at present to be the "standard model"). Again one can consider less powerful tapes, like one way infinite tapes (which possess a leftmost tape square, to the left of which the machine cannot move), or *stacks* (a one-way infinite tape where the reading head always erases the information on a tape square when moving left). One can restrict the tape alphabet to a single non-blank symbol, transforming the tape in this way to a *unary counter*. One can also consider tapes which

are more powerful by being two- or even higher dimensional (with a corresponding increase in the number of possible moves). One has introduced machines with more than one head on a tape, with possibly the additional feature that heads jump to fixed base squares in a single move or may jump to the position of other heads. Most of the above features can moreover be combined in about every conceivable or inconceivable way.

It turns out however that the above modifications don't make any difference as far as the resulting class of functions computed by this type of machines, or languages recognized or accepted by machines in such a class is concerned. The reason for this phenomenon is that all these nice features can be simulated on the standard model as well, with only the disadvantage that what is possible in a single instruction on the more fancy machine may require a time consuming subroutine on the standard model.

A similar situation arises when comparing the Turing machine model with other machine models. In the model of the *Register Machine* one considers an idealization of an ordinary computer, where both the wordlength and the address-space have become infinite: the machine contains an infinite number of *memory cells*, called *registers*, indexed by the natural numbers (including zero), each capable of containing an arbitrary natural number. Initially all memory cells, except a finite set containing the input are zero. The *program* of the register machine consists of instructions in a crude assembly code. Typical instructions are:

load constant
load a register (direct or indirect addressing allowed)
store a register (direct or indirect addressing)
unconditional jump
conditional jump (where the condition is a test on zero on the accumulator or a register)
halt (accept or reject)

Finally the machine may be equipped with more or less powerful arithmetic instructions. The Shepherdson and Sturgis model [41] did not have indirect addressing and the only arithmetic instructions allowed were the successor and predecessor on the natural numbers. The present day standard model (RAM) [37] has addition and subtraction as elementary arithmetic instructions. More powerful modelas have also multiplication and/or division, various bit-wise logical operations (where the machines are supposed to use binary

representation), and also shift instructions in their arithmetical unit [37].

Again it is a matter of programming to show that a Turing machine computation can be programmed on a register machine, by loading the contents of tape squares into registers in the disguise of small integers, and using a fixed index register to store the position of the reading head. Conversely a Turing machine can simulate a register machine by storing the contents of its registers in binary on its tape, together with their addresses. The only thing lost by these simulations is time efficiency. Also within the universe of register machines it turns out that the simplest model can simulate the more powerful instructions. It is known that a register machine with successor and predecessor as only arithmetic, which uses no more than two registers, is universal in the sense that it can simulate every other machine computation [29].

All the machine models and variants indicated above share the following properties. Each machine has a program which may be listed as a sequence of instructions. They can be written down using ordinary characters, or be encoded using binary numbers, in such a way that not only a human reader but also a machine might read and interpret and execute this program. In particular this machine may be one of the same type as the machine the program originally was intended for. This leads to the so-called *universal machine:* a machine which takes a pair, consisting of an encoded machine program i and an encoded input x, as input and simulates the computation of the machine with program i on input x in a step by step way. Consequently, the result of this computation equals the result of machine i on input x; if the machine is used for accepting or recognizing a language, so does the simulating machine. The situation is expressed by the following formula: If u represents the program of the universal machine we have $M_u((i,x)) = M_i(x)$. The = sign here means that if one side is defined then so is the other and the values are equal. The additional parentheses around the two arguments i and x indicate that the pair together is considered as a single input; in the formalization the functions computed by the machine have one argument only, and in order to include functions with more than one argument the formalization requires the presense of (recursive) pairing and projection functions $(-,-)$, p_1 and p_2 such that $p_1((x,y)) = x$, $p_2((x,y)) = y$ and $z = (p_1(z), p_2(z))$. These pairing and projection functions are crucial in case the domain is the set of integers.

There exists also a converse to the universal machine which enables us to encode programs in input data. Consider a machine M_i intended to operate

on a pair of arguments. For each fixed value of the first argument x the resulting function y $M_i((x,y))$ is an effectively computable function which is computed by a machine of the type under consideration; it turns out that one may effectively obtain (an index for) a program computing this partially parameterized function. These exists a fixed total function s (depending on the machine formalism only) such that $M_i((x,y)) = M_{s(i,x)}(y)$ for each triplet (i,x,y). This function s is called the s-m-n *function*.

On base of the above observations ROGERS [34] has introduced an *axiomatic framework* for describing machine models. It consists of an infinite list of effectively computable functions $(\phi_i)_{i\in\mathbb{N}}$, called programs having the following properties:

(0) each effectively computable function occurs somewhere in the list
(1) there exists a universal function in the list: $\phi_u(i,x)) = \phi_i(x)$
(2) there exists a s-m-n function for the list $\phi_i((x,y)) = \phi_{s(i,x)}(y)$.

In terms of this formalization one can express what it means that one machine model simulates another one. We say that model M' simulates model M'' if there exists a so-called *total recursive transformation* t, such that for every program j for model M'' one has $M'_{t(j)}(x) = M''_j(x)$, assuming that models M' and M'' operate on the same format of input (otherwise the input will have to be translated as well).

Although this yields a nice formalization of simulation between machine models this does not provide us with an interesting theory: it has been shown by Rogers that all effective enumerations are recursively isomorphic, i.e., the transformation t above can be chosen to become a bijective recursive function with recursive inverse.

Since our intuition still indicates that there are more or less usable machine models we need a more refined way of comparing models. Such tools are obtained by considering time and storage requirements of machine computations.

4. TIME AND SPACE COMPLEXITY

For each of the machine models introduced above there exists a "natural" way of ascribing to every computation an amount of *computing time* and *storage* used. In the Turing machine model, the computing time is the total number of instructions executed during the computation, whereas the space is measured by the number of tape squares visited during the computation -

the space occupied by the read-only input sometimes is not counted as space used during the computation, which makes it possible to discuss computations which use less space than the space occuppied by the input in the initial state of the computation. The above measures are meaningful for all models of Turing machines mentioned so far.

Also in the case of register machines one can measure time by the number of instructions executed and space by the number of registers used. The latter choice turns out to be less well chosen. As mentioned before a Register machine with two registers only already has universal computing power; since it is hardly believable that everything can be computed in bounded space only, one has to take into account that this universality is rooted in the fact that each register can contain an arbitrarily large number. Therefore one does not measure the number of registers used, but the sum of the base-2 logarithms of the (largest) numbers stored in such registers (or alternatively, the product of the logarithm of the largest number ever stored during the computation and the number of registers used).

One might ask whether one should not keep track of the size of the numbers when measuring time on a register machine as well. In the model one performs arithmetic on the potentially huge numbers stored in the registers, and as far as the present state of the art allows us there is no way we can perform arithmetical operations on huge numbers in constant time. In practice both models are investigated - if every arithmetical operation takes constant time one has the *uniform time* model, whereas charging an operation with the sum of the base-2 logarithms of all operands involved (including the addresses of the registers which are accessed either directly or by indirect addressing) leads to the *logarithmic* model.

For the other formalisms of computation theory, based on manipulation of formal objects corresponding time and space measures can be defined as well. Again one may ask for an axiomatic approach, and such an approach has been proposed by M. BLUM in 1966 [7]. To formalize complexity in the abstract framework of effective enumerations the list of programs $(\phi_i)_i$ is extended by another list of functions called *run-times* $(\Phi_i)_i$ which satisfies the following two axioms:

(3) each program has the same domain as its corresponding run-time:

(4) it is uniformly decidable whether a given program on given input halts in a given time or not: the predicate $\Phi_i(x) = y$ is recursive in i, x and y.

One can show that for every pair of sequences $(\Phi_i)_i$ and $(\Phi'_i)_i$, satisfying these axioms there can be constructed a total recursive function R such that for every index i the inequalities $\Phi_i(x) \leq R(x,\Phi'_i(x))$ and $\Phi'_i(x) \leq R(x,\Phi_i))$ hold for all but finitely many x. It can be inferred from this fact that all structures satisfying the axioms (0),...,(4) (which are called *Abstract Complexity Measures*) are not only recursively isomorphic but ascribe the same run-times (up to a recursive overhead function) to the same programs. So it seems that introducing complexity did not help in separating efficient and non-efficient machine models.

In order to circumvent the above trivialization one has to use a more refined way of comparing machine models. This tool is given by the concept of *Polynomial simulation*: one restricts the overhead function R, which is used to compare the run-times in two machine models to be a polynomial function in its second argument. For example it turns out that most models of Turing machines can simulate each other with *quadratic overhead*: $R(x,y) = y^2$; similarly the Turing machine model may simulate the register machine model with logarithmic time measure with quadratic overhead. It turns out moreover that by most of these simulations the space used is multiplied at most by a constant factor.

Given this more refined notion of equivalence of machine models we obtain the following picture of the presently popular machine models.

There exists a *class 0* containing the typical minimal models used in recursion theory, like the Turing machine with unary notation for input and output, the formalisms of Church and Kleene, and the register machines with successor and predecessor as only arithmetical instructions. It is easy to see that in this model in time f(x) one can compute no function larger than f(x).

Class 1, which is felt to be the "realistic" one, contains among others all Turing Machine models with binary encoding, and all register machines in logarithmic time measure. The register machine with addition and subtraction and uniform time measure also belongs to this class; due to the fact that in time f(x) no function larger than $2^{f(x)}$ can be computed it follows that the register machine in logarithmic time measure will simulate this machine with quadratic overhead. The register machine may be extended by arbitrary arithmetical operations like multiplication and division and various shifts and bitwise logical operations, provided logarithmic time measure is used, and the arithmetical operations added can be programmed in time polynomial in n for n-bit arguments.

Class 1N is obtained from class 1 by introducing a new feature: *Nondeterminism*. In nondeterministic computing devices the machine, during its computation can make *choices*. In the Turing machine model nondeterminism is obtained by relaxing the condition that for every pair of a state and a tape symbol only one instruction is present. If there are more the machine in the course of its computation may choose one of the applicable instructions. In the program for a register machine nondeterminism means that some labels in (un)conditional jumps have multiple occurrences in the program. For nondeterministic machines we have to redefine what a computation of such a device yields for a result. A nondeterministic device no longer computes a function but a relation. If used as a recognizer the machine may both have accepting and rejecting computations for a given input. By convention we say that an input is recognized, provided one of the computations accepts it. As before we assume that all possible computations halt. Finally, when used as an acceptor by agreement an input is accepted provided there exists a halting computation.

As far as computing power or recognizing or accepting power are concerned, adding nondeterminism does not add anything; the tree-like development of a nondeterministic computation can be simulated by a deterministic device searching the entire tree in a breadth-first manner. This simulation however requires exponential overhead. Consequently it is not clear whether the feature of nondeterminism can be simulated with polynomial overhead or not. The problem whether class 1 and 1N are different (as is generally believed to be the case) is one of the fundamental open problems in complexity theory. The notorious P = NP? problem is one of the incarnations of this problem.

Finally class 2 consists of machines where some form of *unbounded parallelism* is included, or where it is possible to perform complex arithmetical operations in unit time on arbitrary long arguments. The standard example of such a device is the register machine with multiplication and division in unit time. It has been established in 1981 [6] that these two instructions already suffice, and that the vector-shifts or bitwise logical operations required for the proofs published in the mid-seventies are no longer needed. Other machines in this class are the parallel Turing machines, and the alternating Turing machine. For details I refer the reader to the literature [8,14,36,37].

5. THE HALTING PROBLEM

As indicated before, it makes a difference whether some set is recognized or accepted by some computing device. Being a recursively enumerable

set is something else than being a recursive set. Such an assertion clearly needs a proof, which consists in this situation by providing an example of a set which is recursively enumerable, but not recursive. The standard set, seving this prupose is the so-called *halting problem*: the set K consisting of (the indices of) those programs which halt if run on their own encoding as input. In formula:

$$K = \{i \mid M_i(i) \text{ halts}\}.$$

That K is recursively enumerable follows from the fact that there exists a machine which on input i starts simulating M_i on input i (the machine uses the universal machine as a subroutine). This machine halts only on those inputs which belong to K, so K indeed is recursively enumerable. On the other hand it is not hard to show by contradiction that K is not recursive. Assume that the predicate $M_i(i)$ halts is decidable (i.e., the language of all elements in K is recognized by some device M_k). Combining some trivial devices with the hypothetical M_k and the universal machine one can build a machine M_d computing the following function:

$$M_d(x) := \underline{\text{if}}\ M_x(x) \text{ halts } \underline{\text{then}}\ M_x(x) + 1\ \underline{\text{else}}\ 0\ \underline{\text{fi}}$$

and taking x = d result in a contradiction; from its description one infers that $M_d(x)$ is defined for all x, so $M_d(d)$ halts and by definition equals $M_d(d) + 1$.

The non-existence of the hypothetical device M_k is the origin of all proofs for undecidability in mathematics and logic, except for a few recent ones. The idea is that any formalism which is sufficiently powerful to encode assertions of the type "$M_i(x)$ halts" as propositions in this formalism, must have an undecidable validity problem for these propositions; otherwise such a decision procedure, combined with the encoding could be used to construct a device M_k which would solve the halting problem.

Note that the above observation is shown without using any specific property of the computing devices under consideration, except for the fact that the collection of computing devices is sufficiently large to contain the "combinations" mentioned in the proof. In fact the proof does not use anything but the three axioms (0), (1) and (2), so it is a result from abstract recursion theory.

In Complexity theory a problem with a comparable status is the so-called *truncated halting problem*. Let f be a total recursive function then the f-truncated halting problem is defined by:

$$S_f := \{(i,x) \mid \text{"}M_i(x) \text{ halts within } f(x) \text{ steps"}\}$$

It is clear from the axioms of complexity theory that the set S_f is recursive. Its purpose is different - it serves as a prototype of a set which is hard to decide. The intuition is that any machine deciding membership of the pairs (i,x) in S_f should take at least f(x) steps. The actual assertions relating this complexity are less sharp, due to various technicalities in the proofs. The technicalities arise, since in the proof by contradiction that the truncated halting problem is hard, the assumption that this problem is easy is used to build some mechanism which computes something in less time than it should do. For example consider the following function:

$$d_f(x) := \underline{\text{if}}\ M_x(x) \text{ halts in less than } f(x) \text{ steps } \underline{\text{then}}\ M_x(x) + 1 \ \underline{\text{else}}\ 0\ \underline{\text{fi}}$$

The assumption that d_f is computed by M_y yields for the argument x = y the relation: $M_y(y) = M_y(y) + 1$ in case $M_y(y)$ halts within f(y) steps. Therefore $M_y(y)$ takes more than f(y) steps on input y. It is not hard to modify the above construction in such a way that the diagonalizing function d_f becomes hard at infinitly many arguments - simply replace $M_x(x)$ in the definition by $M_{p_1(x)}(x)$, where p_1 is the first projection corresponding to the pairing function. There exist alternative methods which produce diagonalizing functions which are hard for almost all inputs.

The problem is that one does not want to show that d_f is hard to compute but one wants to prove that S_f is hard to decide. The above construction can be used to prove this assertion, provided one can use the fact that the major part of the time needed for computing d_f goes into the computation of the condition (which amounts to testing whether the pair (x,x) belongs to S_f). There is no reason whay this should be the case, due to the fact that the computation also invokes the universal machine when computing $M_x(x) + 1$. This problem may be solved, by considering a partial diagonalizing function instead:

$$d'_f(x) := \underline{\text{if}}\ M_x(x) \text{ halts within } f(x) \text{ steps } \underline{\text{then}} \text{ diverge } \underline{\text{else}}\ 0\ \underline{\text{fi}}$$

Now the universal machine is no longer invoked. By assuming technical conditions on the "reasonability" of the machine model used a lower bound on the complexity of d'_f indeed proves a lower bound on the complexity of S_f.

We want to use the fact that some formalism can encode assertions of the form "(i,x) belongs to S_f" to show that deciding validity in this formalism is hard as well. It turns out to be crucial how efficient this encoding is. This leads us to the subject of *efficient reductions*. An efficient reduction from a problem A to a problem B is a total recursive function h which has the following properties:

(0) h(x) belongs to B *iff* x belongs to A
(1) h(x) is computable in polynomial time and logarithmic workspace
(2) the output h(x) is bounded by some polynomial in x.

Given an efficient reduction from A to B it is possible to derive a lower bound for the complexity of B from a lower bound for the complexity of A. Property (0) is the fundamental property of the so-called many-one reduction from recursion theory. Property (1) combines the properties known in the literature of being a *polynomial reduction* and being a *logspace reduction*. The latter condition can be shown to imply the first. Property (2) asserting that the value of the input is expanded by no more than a polynomial is needed in order to obtain good *lower bounds*: a lowerbound of f(x) for A becomes, if the input is multiplied by a factor k a bound of the shape f(x/k) for B; if the input is squared by the reduction a lower bound like $f(\sqrt{x})$ for B would be the best obtainable result.

In most applications of this technique one takes for set A a set S_f or other sets which express the halting of machine computations within a bounded amount of time or storage. Examples of this technique one discussed in Sections 7 and 8 of this paper.

For more precise results concerning this theory of efficient reductions I refer to the monograph by STOCKMEYER [44]. In this monograph the lower bounds obtained are based upon *hierarchy results* which are proved by *diagonalization techniques*. That these results can be obtained using even less powerful techniques like the Kleene form of the recursion theorem, which makes this tool applicable to machine models with less universal power than the effective enumerations has recently been established by MACHTEY & YOUNG [25,26].

6. THE FUNDAMENTAL COMPLEXITY CLASSES

In abstract complexity theory one introduces the following complexity classes:

F_g = the set of all machines ϕ_i such that $\Phi_i(x) \leq g(x)$ for all x with finitely many exceptions.

G_g = the set of all functions computed (languages recognized) by machines in F_g.

In case one uses machines in the accepting mode it only makes sense to bound the runtimes of those computations which terminate. One obtains the weak classes:

F_g^W = the set of all machines ϕ_i such that $\Phi_i(x) \leq g(x)$ for almost all x for which $\phi_i(x)$ halts.

G_g^W = the set of languages accepted by machines in F_g^W.

The above definitions are appropriate when studying complexity classes in the axiomatic framework developed by M. BLUM [7], which theory is known under the name of *Abstract Complexity Theory*. The extension of this theory to the case of the weak classes was the central theme of [47]. But this abstract theory has models which show a rather pathological behaviour, and all attempts to improve this situation be enforcing new "naturalness" axioms so far have failed. Moreover, it turns out that exactly at this point the choice of the domain mentioned in the introduction - numbers or strings - becomes relevant for choosing the proper definitions which are used in formalizing out intuitive concepts about complexity of real-life computations.

When going to concrete machine models like Turing machines, we encounter the situation that the inputs are not numbers but strings over a finite alphabet. One does not measure the size of an input by its *value* but by the *length* needed to write it down. Complexity of a problem is expressed as a function in terms of the length of the input rather than its value. There exist many inputs having the same length. Moreover, in case the devices under consideration are non-deterministic, there may exist various computations by a single device on a single input. Under those circumstances the complexity is measured using the following rules:

The *length of a computation* by a nondeterministic machine on some input is the shortest length of an accepting computation if there exists one, and undefined otherwise (recognizing and accepting modes only).

The *complexity* of some device on some input length n is the maximum of all defined lengths of computations on inputs x of length n.

Hence the fact that some input of length n is rejected does not imply that the complexity on inputs of length n become unbounded or undefined.

For space used by concrete machine models analogous definitions can be given. That one treats accepting and recognizing modes analogously does not disturb the theory too much. If one wants to enforce a reasonable time- or space-bound on all computations of a device which works in recognizing mode, one can run a "clock" in parallel with the computation which shuts off a non-accepting computation at a time at which an accepting computation should have terminated. Similarly one can "lay down" a sufficient amount of space before starting a computation, interrupting the device as soon as it wants to leave its reserved space (in this case one also should simulate a clock in order to prevent the machine from looping on a finite amount of tape).

So it is possible to define time and space complexity classes for concrete machine models which do not compute over the natural numbers as well. Still it may be the case that these classes are highly dependent on the concrete model considered. This is due to the fact that we have considered those models to be equivalent who can simulate each other with polynomial time (and constant space) overhead. This problem is eliminated by considering not individual classes but classes which are the union of increasing families of classes; these unions are formed such that they are closed under polynomial time simulation. Some examples of these fundamental classes are (n denotes always the length of the input):

LOGSPACE the set of languages accepted in space k.log(n) for some constant k by deterministic devices in class 1;

NLOGSPACE idem, for devices in class 1N;

P (= PTIME) the set of languages accepted in time $k.n^k$ for some constant k by deterministic devices in class 1;

NP (= NPTIME) idem for class 1N;

PSPACE the set of languages accepted in space $k.n^k$ for some constant k by devices in class 1;

EXPTIME the set of languages accepted in time 2^{n^k} for some constant k by devices in class 1;

NEXPTIME idem for class 1N;
etc. ...

One may ask why there is no class NPSPACE on the above list. The answer is that, with respect to space, by a result of W. SAVITCH [35] the overhead needed for eliminating nondeterminism is known to be bounded by the function x^2: a nondeterministic computation using space s(n) can be simulated deterministically in space $s(n)^2$. Hence PSPACE = NPSPACE. Clearly this result does not imply anything for the question LOGSPACE = NLOGSPACE?

We have the following chain of inclusions:

$$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME}.$$

None of the inclusions in this chain is known to be proper - they rank among the fundamental open problems in complexity theory. It is known however that some sets in the list are different; we have NLOGSPACE $\neq$ PSPACE and P $\neq$ EXPTIME and NP $\neq$ NEXPTIME.

The concept of efficient reduction, which was introduced in Section 5, needs to be redefined in case complexity is measured in terms of the length of the input, see [44].

It is possible to define the same chain of classes for other machine models as well. The strength of class 2 with respect to class 1 shows itself by the fact that it turns out that all classes are shifted by two places. So one has $\text{LOGSPACE}_2 = \text{P}_1$, $\text{NLOGSPACE}_2 = \text{NP}_1$, $\text{P}_2 = \text{NP}_2 = \text{PSPACE}_1$, etc. ... . This shift is an argument why the models in class 2 are considered unrealistic. For further details see the literature on class 2 mentioned before [6,8,14].

7. COOK'S THEOREM, AND THE EXISTENCE OF NP-COMPLETE SETS

In this and the three remaining sections of this paper we restrict ourself to the standard class (1 or 1N) of machines measuring complexity in terms of the length of the input. In particular, if the input includes representations of numbers, we assume these numbers to be written using binary or decimal notation. So the length of a number x with radix r is the integer $[^r \log x] + 1$ for $x \neq 0$ and 1 for $x = 0$.

Among the open problems in the preceding section the P = NP? problem

has drawn the most attention of researchers in complexity theory and elsewhere. This is due to the fact that intuitively the class P contains those problems which we can solve effectively, whereas the set NP contains a great number of problems which we would like to solve in practical applications; the best algorithms known for these problems in general require exponential run-times, which makes it virtually impossible to solve those problems exactly except for instances of very small size.

The property which brings some problem within the class NP is the presence of some "guess and verify" solution method. Consider for example the following simple set of linear Diophantine equations:

$$SLEQ_{\mathbb{N}} = \{(a_1,\ldots,a_k,b) \mid \Sigma x_i a_i = b \text{ is solvable with } x_i \in \mathbb{N}\}.$$

If we would have allowed solutions with $x_i \in \mathbb{Z}$ we would have an easy way of testing solvability: It would suffice to check that b is divisible by the GCD of the numbers a_i. Again if one of the a_i is negative and $b > 0$ the same test suffices, since this one negative number can be used to make all coefficients of a solution vector positive. A continued fraction algorithm will solve the problem for $k = 2$, and a recent result by H.W. LENSTRA Jr. [22] shows that the problem is solvable in polynomial time for every fixed value of k as well (the degree of the polynomial depending strongly on k!). For the general problem, however, no efficient test is known. On the other hand it is very simple to verify a proposed solution when someone hands it to you. Simply compute $\Sigma x_i a_i$ and test whether it has sum b or not. Since the guessing can be performed by a nondeterministic device this shows that the problem belongs to NP.

Assume that some problem B belongs to NP and that we have a polynomial time reduction from another problem A to B, then it follows that the problem A belongs to NP as well; a guess and verify solution for A is obtained by translating the given instance by the reduction and applying the given guess and verify method for B on the resulting output. Moreover, if some discovery in the future would make clear that the problem B actually belongs to P (i.e., the guess and verify method can be replaced by a polynomial procedure which is entirely deterministic) problem A would have been promoted from NP to P at the same time.

It would therefore be nice if we can restrict ourself to a few prototype problems in NP such that other problems may be reduced to them by polynomial translations. The situation turns out to be even much better: there

exist problems in NP to which all other problems in NP can be reduced. Problems having this property are called NP-*complete*. A problem having only the property that each problem in NP is reducible to it, but which does not necessarily belongs to NP, is called NP-*hard*.

The archetype of an NP-complete problem is the *Satisfiability* problem (named SAT in the literature) proposed by S. Cook who proved it to be NP-complete in 1971 [9]. This problem deals with formulas in the *propositional calculus*. Consider formulas whose atoms are *propositional symbols* denoting *truth values*, and which are built from these atoms using the usual *connectives* for *negation*, *disjunction*, *conjunction* and *implication*. We also consider the more restricted class of formulas in *disjunctive normal form*; these formulas are the conjunction of a number of *clauses*, each clause being a disjunction of *literals*, each literal being a variable or a negated variable. So the formula $(x_2 \underline{\text{ or }} (x_1 \underline{\text{ and not }} x_2)) \underline{\text{ imp }} x_3$ is an example of a propositional formula, which is equivalent to the following formula in disjunctive normal form: $(\bar{x}_2 \underline{\text{ or }} x_3) \underline{\text{ and }} (\bar{x}_1 \underline{\text{ or }} x_2 \underline{\text{ or }} x_3)$, where negation has been indicated by the horizontal line above an atom.

The set SAT consists of those propositional formulas in disjunctive normal form which can be given the value <u>true</u> by giving a suitable *truth-value assignment* to the variables. For example the above formula belongs to SAT since any assignment which makes x_3 <u>true</u> will make the entire formula true as well. Given a truth value assignment for the variables it is almost trivial to compute the truth value for the entire formula, hence the problem SAT belongs to NP. A deterministic procedure clearly is obtained by trying out all truth value assignments in turn, but there may exist an exponential number of them, since the number of different variables may grow linearly with the size of the formula.

That the problem SAT is actually NP-complete can be shown as follows: consider an arbitrary problem in NP, say A. Then there should exist a non-deterministic single tape Turing machine M_i which accepts inputs in A in polynomial time, say n^k. For given input x we therefore have:

> x belongs to A <u>iff</u> there exists an accepting computation of M_i on input x of length $|x|^k$,

where $|x|$ denotes the length of x.

In the proof we shall construct for each x a propositional formula h(x) which expresses the existence of this computation in the following

way: from an assignment of truth values to the variables of h(x) which makes h(x) true we can read a complete "report" of the accepting computation. Such a report consists of a rectangular array, formed by the consecutive instantaneous descriptions of the computation, as described in Section 2. This report is a rectangular table filled with symbols, satisfying a number of conditions, expressing syntactic and semantic validity, like:

(0) all rows have the same length $m \leq |x|^k$, and the number of rows N is at most $|x|^k + 1$.

(1) each row is filled with symbols from the tape alphabet of M_i, except for a single entry which represents a state from the states of M_i.

(2) the top row represents the initial configuration of the machine M_i on input x (i.e., the initial state of M_i, followed by the symbols of x, followed by $m - |x| - 1$ blanks).

(3) the final row of the table represents an accepting configuration (i.e., the state occurring in this row is an accepting state).

(4) the transitions in the table are valid transitions for M_i, i.e., a symbol in row j is repeated in the same place in row j+1 or it should occur within distance 1 from the state symbol occurrence in row j and in the latter case the transition of the three symbols in the neighbourhood of the state symbol from row j to row j+1 represents a valid transition of the Turing machine M_i.

Assume that M_i has q states and s tape symbols (including blank). Let the q+s symbols representing states and tape symbols be denoted $a_1,\ldots,a_{q+s}$. Introduce N.M.(q+s) propositional variables x_{def}, where x_{def} means "symbol a_d is written in the table on position (e,f)". Next one builds a logical formula $\tilde{h}(x)$ in terms of these variables, expressing that each entry in the table is filled with exactly one element, and that the conditions (1), (2), (3) and (4) are valid. It turns out to be possible to express this in a propositional formula of size bounded by const. $(N.M.(q+s)^2 + N.M^2.q^2 + M + q.M + N.M.s^2.(q+s)) \in O(|x|^{3k})$.

Here we have measured the size of the formula $\tilde{h}(x)$ in terms of the number of atoms. Since we in general don't have that many symbols in our alphabet we will need to introduce a large number of variables using indices; this will add another factor $\log(|x|)$ to the size of the formula. Moreover, the formula obtained in general will not be in disjunctive normal form. There exists a way to translate $\tilde{h}(x)$ into an equivalent formula h(x) in disjunctive normal form, which expands the size of the formula by at most

a constant factor (N.B. this translation process will introduce again new variables corresponding to the connectives of the original formula.)

The final formula h(x) in disjunctive normal form has a highly regular structure. Given the input x (for expressing part (2) of the conditions), the length $|x|$, the number k and the program of the Turing machine M_i, another Turing machine can write down this formula, while using no more than const. $\log(|x|)$ worktape for manipulating counters upto $|x|^k$. This translating machine can have the value k and the program of M_i encoded in its finite control; hence the only input it needs is the string x.

The above sketch of a construction establishes the result that SAT is NP-complete, since validity of the formula obtained by the translation would imply that there exists a way of filling the table with symbols, such that all the conditions are satisfied, and from this filling of the table an accepting computation of M_i on input x can be reconstructed. The reduction has moreover the property that there exists a 1-1 correspondence between truth-value assignments making the formula valid, and accepting computations, so the number of accepting computations is preserved as well.

To obtain other examples of NP-complete problems one uses the following observation: let A be an NP-complete problems and let B be some problem in NP to which A can be reduced. Then B is NP-complete as well. Cook himself was the first to use this observation; KARP [17] made the subject popular by establishing a list of about 20 problems which were NP-complete (among which there are some notorious ones like Feasibility of Integer Linear Programs, existence of Hamiltonian circuits in graphs, the travelling salesman problem formulated as a decision problem, etc. ...). For further information I refer to the extensive monograph on the subject written by GAREY and JOHNSON [13].

Needless to state that the problem $SLEQ_{\mathbb{N}}$ mentioned in the beginning of this section is NP-complete as well (see LUEKER [24]).

The techniques developed for proving problems to be NP-complete or NP-hard can be used in the context of other fundamental classes as well. So one has problems which are complete for LOGSPACE, NLOGSPACE, PSPACE, EXPTIME, etc.. In all cases the "master" problem provides for some mechanism which enables one to express the existence of accepting Turing machine computations within a given resource bound.

8. DIOPHANTINE LANGUAGE AND MACHINE MODELS

Diophantine decision problems deal with collections of Diophantine equations described by some general format, and the problem consists of deciding which equations from this set have solutions and which ones haven't. In this chapter we are interested in solutions in non-negative integers only; an integer which also can be negative always can be replaced by the difference of two non-negative ones.

Diophantine equations have the form $P(x_1,\ldots,x_n) = 0$, where P denotes some polynomial. This polynomial contains coefficients, which may be varied, creating in this way a system of equations all of the same shape. Alternatively we can include those coefficients as new variables (as long as we do not include the exponents as variables). This leads to the following definition of *Diophantine relations*:

The relation $R(y_1,\ldots,y_m)$ is Diophantine, provided there exists a polynomial P such that $R(y_1,\ldots,y_m) \equiv \exists x_1,\ldots,x_n[P(y_1,\ldots,y_m,x_1,\ldots,x_n)] = 0$.

A Diophantine relation therefore describes the generic form of a relation which can be encoded in the shape of a Diophantine decision problem. In case the relation R actually is a function from $\mathbb{N}^{m-1}$ to $\mathbb{N}$, we can call R a Diophantine function.

One might be interested in the question "which relations are Diophantine?". The surprising answer, as provided by the proof machinery behind Matijasevic' solution of Hilbert's tenth problem, is "all recursively enumerable relations". It is clear that all Diophantine relations are recursively enumerable by definition, but that the converse holds also is far from self-evident. No wonder that it took until 1970 before the "small technicalities" needed to provide a proof had been completed. For a presentation of a complete proof see [10].

Before considering the nature of these details, I first illustrate some important relations which are clearly Diophantine:

Perfect squares	$\exists x[y = x^2]$
Composite numbers	$\exists x_1,x_2[y = (x_1+2)(x_2+2)]$
$y_1 \leq y_2$	$\exists x[y_2 = y_1 + x]$
$y_1 \mid y_2$	$\exists x[y_2 = y_1.x]$
$y_1 \underline{\text{mod}}\ y_2 = y_3$	$\exists x_1,x_2[y_1 = y_3+x_1.y_2 \ \underline{\text{and}}\ 1+y_3+x_2 = y_2]$.

In the latter example we have used the connective and, which is not Diophantine itself. It is however possible to simulate the logical connectives and and or, using the following equivalences:

$$p(\ldots) = 0 \quad \underline{\text{and}} \quad q(\ldots) = 0 \quad \underline{\text{iff}} \quad p(\ldots)^2 + q(\ldots)^2 = 0$$

$$p(\ldots) = 0 \quad \underline{\text{or}} \quad q(\ldots) = 0 \quad \underline{\text{iff}} \quad p(\ldots).q(\ldots) = 0.$$

It is also clear from the definitions that the class of Diophantine predicates is closed under *existential quantification:* if $R(y_1,\ldots,y_m)$ is Diophantine then so is the relation $R'(y_1,\ldots,y_{m-1}) \equiv \exists y_m[R(y_1,\ldots,y_{m-1},y_m)]$.

It turns out that the main problem which prevents us to prove undecidability of some Diophantine predicate is the lack of some crucial tools in our toolkit. Looking at the traditional way of encoding Turing machine computations as is done in the arithmetization part of the proof that the Turing computable functions are exactly those computed by the Kleene formalism [19], one freely uses the fact that a large class of functions which are not polynomials are *primitive recursive*. (A classical, well-known subclass of the class of total recursive functions.) Consider for example the following functions:

exponentiation $y_3 = y_1^{y_2}$
factorials $y_2 = y_1!$
binomial coefficients $y_3 = \binom{y_1}{y_2}$
y_2 is the prime number with index y_1
y_3 is the exponent of prime number y_2 in the factorization of y_1.

The last predicate is crucial in one of the standard ways of encoding sequences of number by numbers.

Another tool which is lacking is the availability of bounded universal quantification; it is known that the class of primitive recursive functions is closed under this operation, but it is far from evident that this holds for the Diophantine relations.

The need for bounded universal quantification can be seen when considering how a proof might be given for Matijasevic' result. Suppose that we want to show that every recursively enumerable relation is Diophantine. For simplicity we assume we are considering recursively enumerable sets only. Let A be accepted by machine M_k. As before we have the equivalences:

x is an element of A iff
there exists an accepting computation of M_k on input x iff
there exists a table of instantaneous descriptions, representing
an accepting computation on input x.

Let the length of this accepting computation be denoted by t; the table now has dimensions at most t by t, and without loss of generality we can assume that the dimensions are exactly t by t. The entries in the table represent elements from a finite set. We can encode them by small numbers.

Contrary to the situation in the proof of the Cook theorem it is not possible to enforce the conditions (0),...,(4) on the table by listing all occurrences of these constraints; this would make the predicate depending on the length of the input and on the value of t. Moreover the number of variables grows at least as fast as t^2, whereas the number of variables in a Diophantine predicate is finite.

The trick consists of encoding the entire table in a single number. In the Matijasevic proof one uses a trick (going back to Gödel) using congruences - for suitable T and U one can make sure that $T \underline{\text{mod}}(1+kU) = a_k$ for $k = 1,...,t^2$. Here a_k denotes the entry in the table at place k. So one has encoded the entire table in the number T and one can extract individual symbols by naming their address using the additional number U. A local constraint now can be expressed using a predicate which is Diophantine involving T, U and k and other additional arguments. The fact that the constraint has to be enforced all across the table is expressed using bounded quantification over k.

It was known before 1970 that this bounded quantification could be eliminated provided one accepts conditions like $x = \binom{y}{z}$ or $x = y^z$. As a result the class of *exponential Diophantine relations* (which included exponentiation) was known to have full expressive power, and therefore was known to be undecidable [33]. Matijasevic' solution has as its basic tool the exchange of exponentiation against a relation of the form:

$x = \psi_A(k)$, representing "x is the first coordinate of the k-th solution of the Pell equation $x^2 - (A^2-1)y^2 = 1$".

The k-th solution moreover satisfies the congruence $y_k = k \underline{\text{mod}} (A-1)$. This would characterize the k-th solution had it not been the case that the resulting system $x^2 - (A^2-1)y^2 - 1 = 0$, $y = k + r(A-1)$, has other solutions

besides the intended one. These other solutions have an extremely large x-coordinate, which makes it possible to exclude them using another "Pell-like" predicate. This enables one to complete the proof.

The result that all recursively enumerable relations are Diophantine can also be interpreted in the following way. Consider the polynomial $p(y_1,\ldots,y_m,x_1,\ldots,x_n)$ which describes the predicate $R(y_1,\ldots,y_m)$ and consider the following *Diophantine machine* M_p: on input $(y_1,\ldots,y_m)$ the machine guesses non-negative integers $x_1,\ldots,x_n$, and evaluates the polynomial $P(y_1,\ldots,y_m,x_1,\ldots,x_n)$; the machine accepts if the result equals zero and rejects otherwise. Matijasevic's result shows that all recursively enumerable relations are accepted by Diophantine Machines.

These diophantine machines have been introduced by ADLEMAN & MANDERS in their FOCS conference contributions of 1975 and 1976 [1,2]. In these papers they investigate quantitative improvements over Matijasevic' result. Whereas the mathematicians and logicians were mainly interested in bringing down the number of variables needed for describing the universal recursively enumerable relation, or decreasing the degree of the polynomial used, the idea of the Diophantine machine introduces a new question: "how fast does it run?". Clearly a runtime will be an estimate of a nondeterminsitic time measure since the machines are inherently nondeterministic. It should also be clear that just counting the number of arithmetic steps used will not make sense - then the time measure would be a constant which does not depend on the input. Keeping in mind that we should use bit-complexity for the arithmetic used one obtains that the crucial measure is the size of the numbers $x_1,\ldots,x_n$ which have to be guessed in order to obtain an accepting computation. This leads to the following definition: Let $t: \mathbb{N} \to \mathbb{N}$ be a computable function. The diophantine machine M_p accepts the relation $R(y_1,\ldots,y_m)$ in time t $\underline{\text{iff}}$ ($R(y_1,\ldots,y_m)$ $\underline{\text{iff}}$ $\exists x_1,\ldots,x_n$ with $|x_i| \leq t(\Sigma|y_j|)$ and $P(y_1,\ldots,y_m, x_1,\ldots,x_n) = 0$) (as before, $|x|$ denotes the length of the binary representation of x).

Next, one should investigate how this machine model compares with the traditional ones. Using the fact that one can multiply two k-bit numbers in time $O(k^2)$ one easily obtains polynomial simulation in one direction:

$R(y_1,\ldots,y_m)$ is accepted by Diophantine machine M_p in time $t \Rightarrow$

$R(y_1,\ldots,y_m)$ is accepted by a nondeterministic Turing machine in time t^2.

In order to estimate the overhead in the other direction (that such a recursive overhead function exists follows from the abstract theory) one needs to economize on the details of the Matijasevic proof. The existing proofs yields an overhead which is doubly exponential, where the first exponentiation is due to the encoding of sequences, and the second one is needed for eliminating the large Pell solutions. Adelman and Manders show that for functions t of a particular nice type (super runtimes as they are called) a single exponential overhead suffices: for super runtimes t one has

$R(y_1,\ldots,y_m)$ is accepted by a nondeterministic Turing machine in time $t \Rightarrow$

$R(y_1,\ldots,y_m)$ is accepted by a Diophantine machine in time 2^{10t^2}.

I will not go into the details of the improvements; the interested reader may find them in [2]. It turns out that this work is based upon the ideas of Matijasevic', which were still unpublished in 1976; see however [48,49].

For the problem of deciding solvability of diophantine equations this result has the following consequences. It follows by Matijasevic' result that the problem of deciding whether some Diophantine equation is solvable is undecidable. One may consider instead the truncated variant: given a polynomial P and a number K, decide whether $P(x_1,\ldots,x_n)$ is solvable in numbers of length less than K. This truncated version clearly is decidable, but a srtaightforward brute force enumeration requires time of order 2^{nK}. Can we show that this problem indeed is that hard?

The answer to this retoric question is "no". Due to the loss of efficiency during the translation, there results an exponential gap between provable upper and lower bounds, even for the non-deterministic case. The partial result quoted below leaves open the question on how much time is required additionaly to transform the nondeterministic upper bound into a deterministic one. Moreover the formulation of the theorem is complicated by the technical details needed for the proof; in particular the bound on the size of the numbers in the solution K has to be replaced by some function depending on the size of the polynomials involved.

COROLLARY 3.4. (Diophantine Compression Theorem) in [2]: *For all recursive functions* f, *there exists a recursive function* h *such that:*

if

$$S = \{P \mid \exists x_1,\ldots,x_n \text{ with } |x_i| \leq 2^{10h^2(m(p))} \ \&\ p(x_1,\ldots,x_n) = 0\}$$

then

a) *any nondeterministic Turing machine which accepts* S *requires infinitely often more than* $h(m(p)/4) > f(p)$ *steps.*

b) *there exists a Turing machine which accepts* S *in* $2^{20h^2(m(p))}$ *steps.*

where, $p = |P|$ *is the length of (some standard encoding) of the polynomial* P, *and where* $m(x)$ *is some fixed function (independent of* f *and* h*) satisfying* $\forall x[x/c \leq m(x) \leq x]$ *for some* $c \in \mathbb{N}$.

Note that indeed in order for a complexity class like S to be a meaningful object, the bound on the size of the solutions has to grow with the size of the polynomial considered; if K and the degree and number of variables of the polynomials P are fixed, testing all possible solutions bounded in length by K becomes an "efficient" method provided the size of the coefficients of P becomes extremely large.

It is doubtful whether the polynomials referenced in the above corollary will be the ones which draw attention from the mathematical community. In fact they are not interesting at all, since they remain just translated versions of the truncated halting problem.

On the other hand the fact that the overhead for the Diophantine machine model is only slightly exponential (about the same overhead as for eliminating nondeterminism) could lead to the (science fiction) idea of a world where all problems are solved by number crunchers.

9. THE COMPLEXITY STATUS OF SOME SPECIFIC DIOPHANTINE PROBLEMS

In this section we consider some specific well-known types of diophantine problems and mention what is known about their complexity status.

Linear equations

In general one can solve linear problems and systems over the rational numbers using Gauss elimination. If one is interested in systems over the integers one can perform Gauss elimination in a modified way which preserves integrality; it is not clear however that the intermediate results will not explode in size.

SIEVEKING & VON ZUR GATHEN [42] describe a method to decide solvability over the integers of a system of linear equations with integral coefficients, which runs in polynomial time. Other examples of matrix algorithms modified in order to prevent exponential growth of the numbers occurring as intermediate results can be found in KANNAN & BACHEM [16].

As soon as one investigates solutions over $\mathbb{N}$, or equivalently considers systems of inequalities over the integers the problems become special cases of the general integer linear programming problem, which is another archetype of an NP-complete problem. The phenomenon of NP-completeness arises already for very simple types of equations of inequalities. We have encountered already the problem $SLEQ_{\mathbb{N}}$ of a simple linear equation over the natural numbers. Other simple examples are:

KNAPSACK : Is the equation $\Sigma x_i a_i = b$ solvable with $x_i = 0,1$?
PARTITION : Is the equation $\Sigma x_i a_i = 0$ solvable with $x_i = +1,-1$?
WEAK PARTITION: Does the equation $\Sigma x_i a_i = 0$ have a non-trivial solution (i.e., not all $x_i = 0$) over $x_i = -1,0,+1$?

Each of the above problems is NP-complete. For KNAPSACK and PARTITION this has been shown already by KARP [17]; NP-completeness of WEAK PARTITION has been proved by SHAMIR [40]. Fro an independent (but much more complicated) proof see [46].

On the other hand a recent result by H.W. LENSTRA JR. [22] establishes that for fixed k integer linear programming problems with no more than k variables (or with no more than k constraints, where non-negativity constraints have to be counted) is solvable in polynomial time (with the degree of the polynomial depending on k in some awful way).

Primality and factorization

These problems are dealt with extensively in other sections of this tract. Both the set of primes and its complement (the set of composite numbers) belong to NP (PRATT [30]). For primality there exists a polynomial time random procedure (see the next section for this type of algorithms), (see RABIN [32] or SOLOVAY & STRASSEN [43]); this procedure can be made deterministic, in case one believes the generalized Riemann hypothesis to be true [28]. An almost polynomial primality test has been proposed in 1980

by ADLEMAN [5]; see also [23].

Factorizing a number is considered to be a ahrd problem, though nobody has provided a proof for its hardness. A number of cryptological proposals will collapse if factorization turns out to be easy.

Quadratic Equations

In 1976 Manders & Adleman provided an example of a very simple system of quadratic equations, whose solvability problem is NP-complete [27].

the set of those nonnegative triples a, b, c such that
$ax_1^2 + bx_2 - c = 0$ is solvable over $\mathbb{N}$

For this type of problems it is clear that the "guess and verify" method is polynomial. The reduction uses a +1,-1 variant of KNAPSACK, where the choices between +1 and -1 are encoded in the fact that modulo each prime factor of b one can choose between two square roots. The problems is related to the question "does a have square root modulo b smaller than c?".

From the above one infers directly that the problem of deciding solvability of general quadratic equations is at least NP-hard. LAGARIAS has shown in 1979 [21] that it is not harder for binary quadratic equations: solvability of the equation $ax_1^2 + bx_1x_2 + cx_2^2 + dx_1 + ex_2 + f = 0$ can be verified nondeterministically in polynomial time; each equation has a "short" certificate for solvability.

Note that such a certificate, in general, cannot be provided by writing down a solution. It is known that for the anti-Pell equation:

$$x_1^2 - d_k x_2^2 = -1, \quad \text{with} \quad d_k = 5^{2k+1}$$

the smallest solution is represented by (x_1,x_2) from

$$x_1 + x_2\sqrt{5} = (2+\sqrt{5})^{5^k}$$

from which on recognizes that writing down this solution (or any other one) requires exponential space. Instead the certificates are based upon the theory of quadratic forms, and class group computations, about which theory the reader will find information in the contribution to this volume of R.J. Schoof.

For the specific case of the anti-Pell equation, the certificate moreover has the following form: one can determine nondeterministically in polynomial time a simple yes-no question whose answer is equivalent with solvability. It is also decidable whether this question is the correct one, and this decision procedure requires polynomial time. Therefore the same certificate can be used as a certificate for unsolvability as well, and consequently solvability of anti-Pess equations is a problem in NP ∩ co-NP (the class co-NP consists of those sets such that their complement, with respect to a suitable chosen superset, belongs to NP). Presently there exist only a few problems having this complexity status, which are not known to be members of P; previous candidates were the set of primes (which is polynomial for all practical purposes) and linear programming over the rationals (shown to be polynomial by the Khachijan algorithm [18]).

10. DIOPHANTINE EXAMPLES OF PROBLEMS WITH OTHER COMPLEXITY STATUS

In the previous section we have encountered the class of co-NP of sets whose complement belongs to NP. The prime example of a set in this class is the set TAUT of all tautologies in propositional calculus; to see this just observe that the tantologies are exactly the negations of the unsatisfiable formulas.

As with the other fundamental classes little is known about the precise position of the set co-NP. A weaker question than P = NP? is the question NP = co-NP? which also is open. This problem in essence asks for whether NP is closed under complementation, or whether there exists for the propositional calculus a proof system such that every tautology has a short proof within the system (a question highly relevant for researchers in artificial intelligence).

ADLEMAN & MANDERS [3] have shown the following result: Consider the class of equations $ax_1x_2 + x_2 = c$, $a,c \in \mathbb{N}$ (i.e., the problem of characterizing those pairs (a,c) such that c has some divisor x with x $\underline{\text{mod}}$ a = 1). If the solvability problem of this class belongs to co-NP then NP = co-NP.

Since P ⊆ co-NP the same conclusion follows in case the above problem would belong to P. The result shows that the simple existence of short certificates for unsolvability for these equations would have deep consequences for complexity theory.

Another class of problems, which has been inspired by RABIN's work [32] on probabilistic algorithms, is the class R. A probabilistic algorithm is a

nondeterministic algorithm, where every choice is made between two possible outcomes each having change $\frac{1}{2}$; so the control of the machine is equipped with a fair coin which it can flip in order to make choices during its computation. A problem belongs to the set R provided it is accepted by some random algorithm with probability $\frac{1}{2}$ or more. Clearly $R \subset NP$. For example, in Rabins primality test the chance of finding a "witness for non-primality" is at least $\frac{3}{4}$, so the set of non-primes belongs to R. As before we let co-R denote the class of problems whose complements belong to R (like the primes), and $R \cap$ co-R is denoted Δ_R.

One might ask whether it is possible to modify each nondeterministic algorithm in such a way that its chance of success always becomes at least $\frac{1}{2}$. This would imply NP = R. This result would follow as soon as it turns out that one of the NP-complete problems belongs to R, but in this context the reduction concept underlying the notion of NP-completeness may even be weakened. The translation performing the reduction itself may become nondeterministic, as long as it preserves membership of the problems considered and has a probability of producing an output of at least $\frac{1}{2}$. In practice nobody believes R = NP to be true, so the problems which are complete under the more general reductions are unlikely to be either in P or in R.

An example of a diophantine problem which is complete in the generalized sense, assuming the Generalized Riemann hypothesis, is the following: $ax_1x_2 + bx_2 - c = 0$, with a a power of 2, b and c non-negative.

Existence of polynomial algorithm for the solvability problem for these equations would therefore imply R = NP.

The proofs for the results mentioned in this section are interesting for number theoreticians as well. The example concerning the relation NP = co-NP? is proved using Linnik's result on primes in arithmetic progressions. The example concerning NP = R? requires additional information concerning the density of those primes and the speed with which the asymptotic density due to Dirichlet's theorem is approached. As far as I know these estimates have only been shown using Riemann Hypotheses, but the authors invoke some unidentified competent number theoretician who confirms that the required estimates are not only valid but even provable without use of unproven hypotheses. The Riemann hypothesis is used moreover in the sequel of the proof as well; one does not only need to have access to sufficiently many primes in an arithmetic progression, but one also needs to be sure that these numbers are prime, for which the authors rely on Miller's primality test, which is correct only assuming the Generalized Riemann Hypothesis.

A final example of a system of equations with an interesting complexity status is the collection

$$x_1^2 - a^2 x_2^2 = c, \quad \text{with} \quad a,c \text{ nonnegative.}$$

ADLEMAN & MANDERS [4] claim that the solvability problem for these equations is complete under an even more general class of reductions" the "lying" reduction. As a consequence, a polynomial test for deciding solvability of equations of the type above would imply the unlikely relation NP = Δ_R. The Claim having been put forward without any indication of a proof, I am unable to provide details on its background.

REFERENCES

[1] ADLEMAN, L. & K. MANDERS, *Computational complexity of decision procedures for polynomials*, Proc. IEEE Symp. Found. Comp. Sci. 16 (1975), 169-177.

[2] ADLEMAN, L. & K. MANDERS, *Diophantine Complexity*, Proc. IEEE Symp. Found. Comp. Sci. 17 (1976), 81-88.

[3] ADLEMAN, L. & K. MANDERS, *Reducibility, randomness and intractability*, Proc. ACM SIGACT Symp. Theory of Computing 9 (1977), 151-163.

[4] ADLEMAN, L. & K. MANDERS, *Reductions that lie*, Proc. IEEE Symp. Found. Comp. Sci. 20 (1979), 397-410.

[5] ADLEMAN, L., *On distinguishing prime numbers from composite numbers*, Proc. IEEE Symp. Found. Comp. Sci. 21 (1980), 387-406.

[6] BERTONI, A., G. MAURI & N. SABADINI, *A characterization of the class of functions computable in polynomial time on random access machines*, Proc. ACM SIGACT Symp. Theory on Computing 13 (1981) 168-176.

[7] BLUM, M., *A machine-independent theory of the complexity of recursive functions*, J. Assoc. Comput. Math. 14 (1967), 322-336.

[8] CHANDRA, A.K. & L.J. STOCKMEYER, *Alternation*, Proc. IEEE Symp. Found. Comp. Sci. 17 (1976), 98-108.

[9] COOK, S.A., *The complexity of theorem-proving procedures*, Proc. ACM SIGACT Symp. Theory of Computing 3 (1971), 151-158.

[10] DAVIS, M., *Hilbert's tenth problem is unsolvable*, Amer. Math. Monthly 80 (1973), 233-269.

[11] DAVIS, M., Invited lecture May 22 1981 at workshop Recursion Theoretic Aspects of Computer Science, Purdue Univ. May 19-22 1981.

[12] FENSTAD, J.E., *General Recursion Theory, An Axiomatic Approach*, Springer Berlin, etc. 1980.

[13] GAREY, M.R. & D.S. JOHNSON, *Computers and Intractability, a guide to the theory of NP-completeness*, Freeman, San Francisco 1979.

[14] HARTMANIS, J. & J. SIMON, *On the structure of feasible computations*, In: M. Rubinoff and M.C. Yovits (eds), *Advances in Computers*, Acad. Press, New York 1976, 1-43.

[15] HOPCROFT, J.E. & J.D. ULLMAN, *Introduction to automata theory, languages and computation*, Addison Wesley, Reading 1979.

[16] KANNAN, R. & A. BACHEM, *Polynomial Algorithms for computing the Smith and Hermite normal form of an integer matrix*, SIAM J. Comput. 8 (1979), 499-507.

[17] KARP, R.M., *Reducibility among combinatorial problems*, In: R.W. Miller and J.W. Thatcher (eds), *Complexity of Computer Computations*, Plenum, New York, 1972, 85-104.

[18] KHACHIJAN, L.G., *A polynomial algorithm in linear programming*, Dokl. Akad. Nauk. SSSR 244 (1979), 1093-1096. (Engl. transl. Soviet Math. Dokl. 20 (1979), 191-194.)

[19] KLEENE, S.C., *Introduction to metamathematics*, North-Holland, Amsterdam 1952.

[20] KLEENE, S.C., *Origins of recursive function theory*, Proc. IEEE Symp. Found. Comp. Sci. 20 (invited lecture) (1979), 371-382.

[21] LAGARIAS, J.C., *Succinct certificates for the solvability of binary quadratic Diophantine equations*, Proc. IEEE Symp. Math. Found. Comp. Sci. 20 (1979), 47-56.

[22] LENSTRA, H.W. JR., *Integer programming with a fixed number of variables*, Rep. MI-UVA 81-03, revised version, Apr./Nov. 1981.

[23] LENSTRA, H.W. JR., *Primality testing algorithms (after Adleman, Rumely & Williams)*, Sém. Bourbaki 576, 33^{e} année 1980/81, June 1981, Springer LNM 901 (1981), 243-257.

[24] LUEKER, G.S., *Two NP-complete problems in nonnegative integer programming*, Rep. 178 CSL Princeton Univ. 1975.

[25] MACHTEY, M. & P. YOUNG, *An introduction to the general theory of algorithms*, Theory of computation series, North-Holland, New York, 1978.

[26] MACHTEY, M. & P. YOUNG, *Remarks on recursion versus diagonalisation and exponentially difficult problems*, J. Comput. Syst. Sci. 22 (1981), 442-453.

[27] MANDERS, K.L. & L. ADLEMAN, *NP-complete decision problems for binary quadratics*, J. Comput. Syst. Sci. 16 (1978), 168-184.

[28] MILLER, G.L., *Riemann's hypothesis and tests for primality*, J. Comput. Syst. Sci. 13 (1976), 300-317.

[29] MINSKY, M., *Computation, finite and infinite machines*, Prentice-Hall, London, 1972.

[30] PRATT, V., *Every prime has a succinct certificate*, SIAM J. Comput. 4 (1975), 214-220.

[31] POMERANCE, C., *Recent developments in primality testing*, Math. Intelligencer 3 (1981), 97-105.

[32] RABIN, M.O., *Probabilistic algorithms for testing primality*, J. Number Theory 12 (1980) 128-138.

[33] ROBINSON, J., *Existential definability in arithmetic*, Trans. Amer. Math. Soc. 72 (1952), 437-449.

[34] ROGERS, H., *Gödel numberings of partial recursive functions*, J. Symb. Logic 23 (1958), 331-341.

[35] SAVITCH, W.J., *Relations between deterministic and nondeterministic tape complexities*, J. Comput. Syst. Sci. 4 (1970), 177-192.

[36] SAVITCH, W.J. & M.J. STIMSON, *Time bounded random access machines with parallel processing*, J. Assoc. Comput. Mach. 26 (1979) 103-118.

[37] SCHÖNHAGE, A., *On the power of random access machines*, Proc. Int. Coll. Automata, Languages and Programming 6, Springer LCS 71 (1979) 520-529.

[38] SCHÖNHAGE, A., *Storage modification machines*, SIAM J. Comp. 9 (1980), 490-508.

[39] SEIFERAS, J.I., M.J. FISCHER & A.R. MEYER, *Refinements of nondeterministic time and space hierarchies*, Proc. IEEE Symp. Switching Automata Theory 14 (1973), 130-136.

[40] SHAMIR, A., *On the cryptocomplexity of knapsack systems*, Proc. AMS SIGACT Symp. Theory of Computing 11 (1979), 118-129.

[41] SHEPHERDSON, J.C. & H.E. STURGIS, *Computability of recursive functions*, J. Assoc. Comput. Math. 10 (1963), 217-255.

[42] SIEVEKING, M. & J. VON ZUR GATHEN, *Weitere zum Erfüllungsproblem polynomial äquivalente kombinatorische Aufgaben*, In: E. Specker and V. Strassen (eds), *Komplexität von Entscheidungsproblemen*, Springer LCS 43 (1976), 49-71.

[43] SOLOVAY, R. & V. STRASSEN, *A fast Monte-Carlo test for primality*, SIAM J. Comput. 6 (1977), 84-85; erratum 7 (1978), 118.

[44] STOCKMEYER, L.J., *The complexity of decision problems in automata theory and logic*, Rep. MAC TR-133, MIT July 1974.

[45] TURING, A.M., *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. II 42 (1936), 230-265.

[46] VAN EMDE BOAS, P., *Another NP-complete decision problem and the complexity of computing short vectors in a lattice*, Rep. MI-UVA 81-04, April 1981.

[47] VAN EMDE BOAS, P., *Abstract Resource-Bound Classes*, Ph.D. Thesis, Sept. 1974, Math. Centre, Amsterdam.

[48] JONES, J.P., *Universal Diophantine Equation*, J. Symb. Logic 47 (1982), 549-571.

[49] MATIJASEVIC, YU V., *A new proof of the theorem on exponential Diophantine representation of enumerable sets*, J. Soviet Mathematics 14 (1980), 1475-1486.

FAST ARITHMETIC OPERATIONS ON NUMBERS AND POLYNOMIALS

by

J.W.M. TURK

In this paper we discuss some methods for computing the product or quotient of two given integers or two given polynomials as well as a method for evaluating a given polynomial at given points. The methods reduce the operations to a number of elementary operations such as the addition or multiplication of binary digits. The main feature of the methods is that they are designed to minimize the number of elementary operations asymptotically. No effort has been made to make these methods suitable for practical use.

§1. MULTIPLICATION OF TWO POLYNOMIALS. I

In this paragraph t is a non-negative integer, $n = 2^t$ and R is a commutative ring which contains 1, 2^{-1} and a primitive 2^{t+1}-th root of 1, $\zeta_{2^{t+1}}$, which is to say a zero of the 2^{t+1}-th cyclotomic polynomial $X^{2^t} + 1$.

An important tool is the concept of a *finite Fourier transform* of a sequence. Put $n = 2^t$ and let $(f_i)_{i=0}^{n-1}$ be a sequence of n elements in R. We define the sequence $(\hat{f}_i)_{i=0}^{n-1}$ in R by

$$\hat{f}_i = \sum_{j=0}^{n-1} f_j \zeta_n^{ij} \quad (\zeta_n := \zeta_{2^{t+1}}^2).$$

Also, we define the sequence $(\overset{\vee}{f}_i)_{\substack{i=0 \\ i\ \mathrm{odd}}}^{2n-1}$ in R by

$$\overset{\vee}{f}_i = \sum_{j=0}^{n-1} f_j \zeta_{2n}^{ij}.$$

The sequences $(\hat{f}_i)$ and $(\overset{\vee}{f}_i)$ are called Fourier transforms of (f_i).

Observe that the $\hat{f}_i$ are the values of the polynomial $F = \Sigma_{j=0}^{n-1} f_j X^j \in R[X]$ at the powers of ζ_n, while the $\overset{\vee}{f}_i$ are the values of F at the odd powers of ζ_{2n}.

First we show that the inverse transformations are very similar to the Fourier transformations:

For $0 \le i \le n-1$

$$(1a) \qquad f_i = n^{-1} \sum_{j=0}^{n-1} \hat{f}_j \zeta_n^{-ij} ,$$

$$(1b) \qquad f_i = n^{-1} \sum_{\substack{j=0 \\ j \text{ odd}}}^{2n-1} \overset{\vee}{f}_j \zeta_{2n}^{-ij} . \qquad (n^{-1} = (2^{-1})^t)$$

PROOF. We shall prove only (1b), as (1a) can be proven similarly. We have

$$\sum_{\substack{j=0 \\ j \text{ odd}}}^{2n-1} \overset{\vee}{f}_j \zeta_{2n}^{-ij} = \sum_{k=0}^{n-1} \overset{\vee}{f}_{2k+1} \zeta_{2n}^{-i(2k+1)}$$

$$= \sum_{k=0}^{n-1} \sum_{\ell=0}^{n-1} f_\ell \zeta_{2n}^{(2k+1)(\ell-i)}$$

$$= \sum_{\ell=0}^{n-1} f_\ell \zeta_{2n}^{\ell-i} \left(\sum_{k=0}^{n-1} \zeta_{2n}^{2(\ell-i)k} \right).$$

The inner sum equals n if $\ell \equiv i \bmod n$ and equals zero if $\ell \not\equiv i \bmod n$, which gives the assertion. □

Note that the calculation of a Fourier transform of $(f_i)_{i=0}^{n-1}$ in the straightforward manner uses n^2 additions in R and n^2 multiplications by powers of ζ_{2n}. It is an important fact that the transforms can be computed in significantly fewer operations in R.

LEMMA 1. *Let* R *and* $n = 2^t$ *be as before and let* $f_0, \ldots, f_{n-1} \in R$ *be given. The* n *elements* $\hat{f}_i = \Sigma_{j=0}^{n-1} f_j \zeta_n^{ij}$ $(0 \le i \le n-1)$ *can be computed by* nt *additions in* R *plus* nt *multiplications in* R *by powers of* ζ_{2n}. *The same holds for the* n *elements* $\overset{\vee}{f}_i = \Sigma_{j=0}^{n-1} f_j \zeta_{2n}^{ij}$ $(0 \le i \le 2n-1, \text{ i odd})$. *Given* $(\hat{f}_i)_{i=0}^{n-1}$, *or* $(\overset{\vee}{f}_i)_{i=0, i \text{ odd}}^{2n-1}$, *one can compute* $(f_i)_{i=0}^{n-1}$ *in* nt *additions in* R *plus* nt *multiplications in* R *by powers of* ζ_{2n} *plus* n *multiplications by* $n^{-1} (= (2^{-1})^t)$.

PROOF. Put $F(X) = \Sigma_{j=0}^{n-1} f_j X^j \in R[X]$. Write

$$F(X) = \sum_{j \text{ even}} f_j X^j + \sum_{j \text{ odd}} f_j X^j =: F_0(X^2) + XF_1(X^2).$$

Then the degrees of F_0 and F_1 are less than $n/2 = 2^{t-1}$ and

$$F(\zeta_n^i) = F_0(\zeta_n^{2i}) + \zeta_n^i F_1(\zeta_n^{2i}) \quad \text{for } 0 \le i \le n-1$$

while

$$\{\zeta_n^{2i}: 0 \le i \le n-1\} = \{\zeta_{n/2}^i: 0 \le i \le n/2-1\},$$

where $\zeta_{n/2} = \zeta_n^2$.

So the evaluation of F (degree < n) at the powers of ζ_n can be reduced to the evaluation of F_0 and F_1 (degree < n/2) at the powers of $\zeta_{n/2}$, at the cost of n additions in R plus n multiplications in R by powers of ζ_n. Since n is a power of 2 we can iterate this procedure until we reach constant polynomials. Adding the costs gives the first assertion of Lemma 1. Since $\{\zeta_{2n}^{2i}: 0 \le i \le 2n-1, i \text{ odd}\} = \{\zeta_n^i: 0 \le i \le n-1, i \text{ odd}\}$, the values of F at the odd powers of ζ_{2n} (i.e. the $\overset{\vee}{f}_i$) can be obtained likewise. The last assertion of Lemma 1 follows now at once from the already mentioned facts (1) about the inverse transforms. □

LEMMA 2. *Let* R *be as before. One multiplication in* $R[X]/(X^{2^t}+1)$ *can be performed by* 2^t *multiplications in* R *plus* $3t2^t$ *additions in* R *and* $3t2^t$ *multiplication in* R *by powers of* ζ_{2t+1} *plus* 2^t *multiplications in* R *by* $(2^{-1})^t$.

PROOF. Put $n = 2^t$. The assertion of the lemma is meant as follows. Let $F = \Sigma_{i=0}^{n-1} f_i X^i$ and $G = \Sigma_{i=0}^{n-1} g_i X^i$ in R[X] be given as representatives of two elements in $R[X]/(X^n+1)$ and let $H = \Sigma_{i=0}^{n-1} h_i X^i$ in R[X] be defined by $FG \equiv H \bmod (X^n+1)$. Then all h_i can be computed as stated. We have

$$\overset{\vee}{f}_i \overset{\vee}{g}_i = F(\zeta_{2n}^i)G(\zeta_{2n}^i) = H(\zeta_{2n}^i) = \overset{\vee}{h}_i, \quad \text{for } 0 \le i \le 2n-1, \ i \text{ odd}.$$

So we can compute all $\overset{\vee}{h}_i$ by n multiplications in R, given all $\overset{\vee}{f}_i$, $\overset{\vee}{g}_i$. The latter can be computed from F and G in $2t2^t$ additions in R and $2t2^t$ multiplications in R by powers of ζ_{2n} (by Lemma 1). We can then compute all h_i from the $\overset{\vee}{h}_i$, by Lemma 1 again, in $t2^t$ further additions in R and $t2^t$ further multiplications by powers of ζ_{2n} in R plus 2^t multiplications in R by n^{-1}. □

REMARK. Suppose T is a cummutative ring with $1, 2^{-1} \in T$. Let F and G in T[X] be given and of degree less than $n = 2^t$. *If* T *contains an element* $\zeta_{2^{t+2}}$ then

$FG \in T[X]$ can be computed in $2n$ multiplications in T, $6n(t+1)$ additions in T, $6n(t+1)$ multiplications by powers of ζ_{2t+2} in T plus $2n$ multiplications by $(2^{-1})^t$ in T. This follows from Lemma 2 since multiplying F and G modulo $X^{2n}+1$ clearly gives FG. In the next paragraph we shall remove the restriction $\zeta_{2t+2} \in T$ (see Theorem 1), which is important for applications.

§2. MULTIPLICATION OF TWO POLYNOMIALS. II

In this paragraph (and the next ones) T denotes a commutative ring with $1,2^{-1} \in T$. All constants implied by the O-symbols are absolute. We shall count one subtraction in T as one addition.

LEMMA 3. *Let* T *be as above and* $t \geq 2$. *One multiplication in* $T[X]/(X^{2^t}+1)$ *can be performed by* $O(2^t t)$ *multiplications in* T *plus* $O(2^t t \log t)$ *additions in* T.

PROOF. As representations for two elements in $T[X]/(X^{2^t}+1)$ we take F and G in T[X] of degree $< 2^t$. We shall compute the coefficients of $H = \Sigma_{i=0}^{2^t-1} H_i X^i$ defined as $FG \equiv H \bmod (X^{2^t}+1)$. Choose $k \in \mathbb{N}$ with $k < t$ (k shall be specified later) and write $F = \Sigma f_i X^{i2^k}$, $G = \Sigma g_i X^{i2^k}$ with f_i, g_i in T[X] of degree $< 2^k$ for $0 \leq i \leq 2^{t-k}-1$.

Step 1. Multiply $\Sigma f_i Y^i$ and $\Sigma g_i Y^i$ in $T[X][Y]/(Y^{2^{t-k}}+1)$. Let the result be $H = \Sigma_{i=0}^{2^{t-k}-1} h_i Y^i$, $h_i \in T[X]$, say.

Step 2. Substitute X^{2^k} for Y in H and reduce $\bmod (X^{2^t}+1)$. This will give $H = \Sigma_{i=0}^{2^t-1} H_i X^i$, since $(X^{2^k})^{2^{t-k}} + 1 \equiv 0 \bmod (X^{2^t}+1)$.

Note that $h_i = \Sigma_{k+\ell=i} f_k g_\ell - \Sigma_{k+\ell=i+2^{t-k}} f_k g_\ell$ for $0 \leq i < 2^{t-k}$, in particular the degrees of h_i are less than 2^{k+1}. It follows that Step 2 can be performed in at most 2^t additions in T. It also follows that the multiplication in Step 1 can actually be performed in $R[Y]/(Y^{2^{t-k}}+1)$, where $R = T[X]/(X^{2^{k+1}}+1)$. So:

One multiplication in $T[X]/(X^{2^t}+1)$ can be reduced to one multiplication in $R[Y]/(Y^{2^{t-k}}+1)$ plus 2^t additions in T, where $R = T[X]/(X^{2^{k+1}}+1)$.

We use Lemma 2 now. Note that X can take the rôle of ζ_{2k+2} in R, so if we choose k such that $2^{t-k+1} \leq 2^{k+2}$ (i.e. $k \geq (t-1)/2$) then we have an

element ζ_{2t-k+1} in R (which is a power of X). Choose $k = [t/2]$. One multiplication by a power of ζ_{2t-k+1} in R can be done by 2^{k+1} additions in T. Therefore, by Lemma 2, one multiplication in $R[Y]/(Y^{2^{t-k}}+1)$ can be done by 2^{t-k} multiplications in R plus $6(t-k)2^{t-k}.2^{k+1}$ additions in T plus $2^{t-k}.2^{k+1} = 2^{t+1}$ multiplications in T (by $(2^{-1})^{t-k}$). Taking also the 2^t additions of Step 2 into account we obtain, writing $k(t) = k+1 = [t/2] + 1$:

One multiplication in $T[X]/(X^{2^t}+1)$ can be reduced to $2^{t-k(t)+1}$ multiplications in $T[X]/(X^{2^{k(t)}}+1)$, plus at most $12t2^t$ additions in T, plus 2^{t+1} multiplications in T.

Since $k(t) < t$ for $t \geq 2$ we can iterate this reduction process until we arrive in $T[X]/(X^4+1)$, where we use any multiplication method. Denote by $M(2^t)$ and $A(2^t)$ the number of multiplications and additions in T, respectively, that this method takes to perform one multiplication in $T[X]/(X^{2^t}+1)$. We have shown that for $t \geq 3$

$$M(2^t) \leq 2^{t-k(t)+1}M(2^{k(t)}) + 2^{t+1},$$

$$A(2^t) \leq 2^{t-k(t)+1}A(2^{k(t)}) + 12t2^t.$$

Put $\alpha(t) = A(2^t)/2^t$, then $\alpha(t) \leq 2\alpha(k(t)) + 12t$. This implies $\alpha(t) = O(t \log t)$. Hence $A(2^t) = O(2^t t \log t)$. Similarly, $M(2^t) = O(2^t t)$. □

As an immediate corollary we obtain

THEOREM 1. *Let* T *be a commutative ring with* $1, 2^{-1} \in T$. *The product of two elements in* T[X] *of degree at most* n (≥ 3) *can be computed in* $M(n) = O(n \log n)$ *multiplications in* T *plus* $A(n) = O(n \log n \log\log n)$ *additions in* T.

PROOF. Let $t \in \mathbb{N}$ be minimal with $2n < 2^t$. Then $2^t \leq 4n$ and multiplication of two polynomials of degree at most n modulo $X^{2^t}+1$ is just ordinary multiplication. So the assertion follows at once from Lemma 3. □

§3. MULTIPLICATION OF TWO INTEGERS.

The usual multiplication method for two integers with n digits uses n^2 multiplications of digits and about $2n^2$ additions of digits. It is a remarkable fact, known since the early 1960's, that there exist other methods that use significantly fewer digit operations (for large n). The following relatively simple method is due to Karacuba and Toom [KNUTH].

Let f and g be two n-digit numbers (base b). First we assume that n is a power of 2. Put $m = n/2$ and write $f = f_0 + f_1 b^m$, $g = g_0 + g_1 b^m$, where f_0, f_1, g_0, g_1 are m-digit numbers. We have $fg = f_0 g_0 + (f_0 g_1 + f_1 g_0) b^m + f_1 g_1 b^{2m}$. The crucial observation is that

$$f_0 g_1 + f_1 g_0 = f_0 g_0 + f_1 g_1 - (f_0 - f_1)(g_0 - g_1).$$

So it is sufficient to perform the multiplications $m_1 = f_0 g_0$, $m_2 = f_1 g_1$, $m_3 = (f_0 - f_1)(g_0 - g_1)$, the additions/subtractions $a_1 = f_0 - f_1$, $a_2 = g_0 - g_1$, $a_3 = m_1 + m_2$, $a_4 = a_3 - m_3$, $a_5 = m_1 + a_4 b^m$, $a_6 = a_5 + m_2 b^{2m}$. Of course, multiplying by a power of b is just introducing trailing zeros. Hence, with a little effort, one can reduce the multiplication of two n-digit numbers to three multiplications of m-digit numbers and eight additions of m-digit numbers. The multiplications of m-digit numbers can be similarly reduced, until we reach 1-digit numbers (n is a power of 2). Denote by A(n) and M(n) the number of additions and multiplications of digits, respectively, that this method uses to compute the product of two n-digit numbers. We have shown that

$$A(n) \le 3A(n/2) + 4n$$

$$M(n) \le 3M(n/2),$$

while $A(1) = 0$, $M(1) = 0$. Put $\alpha(n) = A(n)/n$, then $\alpha(n) \le 3\alpha(n/2) + 4$. It follows that $\alpha(2^s) \le (3/2)^s \alpha(1) + 4((3/2)^s - 1)/(3/2 - 1) = 8((3/2)^s - 1)$. Hence $A(2^s) \le 8(3^s - 2^s) < 8.3^s$. Similarly, $M(2^s) \le 3^s$.

If n is not a power of 2, then let 2^s be the smallest power of 2 exceeding n and view f and g as 2^s-digit numbers by adding leading zeros. Then $2^{s-1} \le n$ hence $3^s \le 3n^{\lg 3}$, where $\lg 3 = \ln 3/\ln 2$. We have proven

LEMMA 4. *The method of Karacuba and Toom computes the product of two* n*-digit numbers in at most* $24n^{\lg 3}$ *additions of digits plus* $3n^{\lg 3}$ *multiplications of digits, hence in at most* $27n^{\lg 3}$ *digit operations* ($\lg 3 = (\ln 3)/(\ln 2) = 1.58\ldots$).

LEMMA 5. *Let* $t \ge 3$. *One multiplication in* $\mathbb{Z}/(2^{2^t}+1)\mathbb{Z}$ *can be performed in* $O(2^t t \log t)$ *bit operations.*

PROOF. The assertion is meant as follows. Let the binary digits of two integers f and g in $\{0,1,\ldots,2^{2^t}\}$ be given. Then the binary digits of the integer $h \in \{0,1,\ldots,2^{2^t}\}$ with $fg \equiv h \bmod(2^{2^t}+1)$ can be computed in $O(2^t t \log t)$ additions or multiplications of binary digits. The method we use is quite similar to the one used in Lemma 3, but there are some complications. Choose $k = [t/2]$ and write $f = \Sigma f_i 2^{i2^k}$, $g = \Sigma g_i 2^{i2^k}$ with $f_i, g_i \in \{0,1,\ldots,2^{2^k}-1\}$ for $0 \le i < 2^{t-k}$. Put $F = \Sigma f_i Y^i$, $G = \Sigma g_i Y^i$;

Step 1. Compute $H = \Sigma_{i=0}^{2^{t-k}-1} h_i Y^i \equiv FG \bmod(Y^{2^{t-k}} + 1)$. Then

Step 2. Evaluate H at 2^{2^k} and reduce $\bmod(2^{2^t} + 1)$.

As in Lemma 3, we have $h_i = \Sigma_{k+\ell=i} f_k g_\ell - \Sigma_{k+\ell=i+2^{t-k}} f_k g_\ell$ for $0 \le i < 2^{t-k}$. Due to the archimedean character of the usual valuation on the integers, contrasting to the non-archimedean character of the degree valuation on T[X], the equivalent of $\deg(h_i) < 2^{k+1}$ is less convenient. We have

$$-2^{t-k}(2^{2^{k+1}} + 1) \le h_i < 2^{t-k}(2^{2^{k+1}} + 1) \quad \text{for } 0 \le i < 2^{t-k}.$$

We shall compute FG in $T[Y]/(Y^{2^{t-k}} + 1)$ with $T = \mathbb{Z}/(2^{t-k+1}\mathbb{Z})$ and $T = \mathbb{Z}/(2^{2^{k+1}}+1)\mathbb{Z}$. This will give (1a)$h_i \bmod 2^{t-k+1}$ and (1b)$h_i \bmod (2^{2^{k+1}} + 1)$. By the Chinese remainder theorem we then get (1c)h_i. The final evaluation Step 2 takes only $O(2^t)$ bit operations, since the h_i have at most 3.2^k digits $(0 \le i \le 2^{t-k})$.

(1a). We shall actually compute

$$\lambda_i := \sum_{\substack{k,\ell=0 \\ k+\ell=i}}^{2^{t-k}-1} \bar{f}_k \bar{g}_\ell \qquad \text{for } 0 \le i \le 2^{t-k+1} - 1,$$

where $\bar{}$ denotes reduction mod 2^{t-k+1}. Then the $h_i \bmod 2^{t-k+1} \equiv \lambda_i - \lambda_{i+2^{t-k}}$ $(0 \le i \le 2^{t-k} - 1)$ follow by subtraction and truncation at 2^{t-k+1}. Note that

$$0 \le \lambda_i < (2^{t-k+1})^2 \sum_{k+\ell=i} 1 \le 2^{2(t-k+1)+t-k} =: 2^\sigma$$

$$\text{for } 0 \le i \le 2(2^{t-k} - 1).$$

Multiply the numbers $f^* = \Sigma_i \bar{f}_i 2^{\sigma i}$ and $g^* = \Sigma_i \bar{g}_i 2^{\sigma i}$. Since $f^* g^* = \Sigma \lambda_i 2^{\sigma i}$ and $0 \le \lambda_i < 2^\sigma$ for $0 \le i \le 2(2^{t-k}-1)$ all numbers λ_i can be read off immediately. Note that f^* and g^* have at most $\sigma(2^{t-k})$ digits. The subtractions (and

truncations) can be done in at most $\sigma(2^{t-k})$ bit operations, so, by Lemma 4, all $h_i \bmod 2^{t-k+1}$ $(0 \le i < 2^{t-k})$ can be computed in $O((t-k)2^{t-k})^{\ell g\, 3}) = o(2^t)$ bit operations.

(1b). Multiplication of FG in $T[Y]/(Y^{2^{t-k}}+1)$, $T = \mathbb{Z}/(2^{2^{k+1}}+1)\mathbb{Z}$.

We use Lemma 3. Note that 2 can take the rôle of ζ_{2k+2} in T, hence, since $k = [t/2]$, we have a ζ_{2t-k+1} in T (which is 2 or 2^2). By Lemma 2, we can compute $FG \in T[Y]/(Y^{2^{t-k}}+1)$ (i.e. all $h_i \bmod (2^{2^{k+1}}+1)$) in 2^{t-k} multiplications in T plus $O((t-k)2^{t-k})$ additions in T plus $O((t-k)2^{t-k})$ multiplications in T by powers of 2. So 2^{t-k} multiplications in $\mathbb{Z}/(2^{2^{k+1}}+1)\mathbb{Z}$ plus $O(t2^t)$ bit operations suffice.

(1c). Given all $h_i^{(1)} = h_i \bmod 2^{t-k+1} \in \{0,1,\ldots,2^{t-k+1}-1\}$ and all $h_i^{(2)} = h_i \bmod (2^{2^{k+1}}+1) \in \{0,1,\ldots,2^{2^{k+1}}\}$, the h_i can be computed in $O(2^t)$ bit operations, e.g. as follows: compute all $d_i \equiv h_i^{(1)} - h_i^{(2)} \bmod 2^{t-k+1}$, $-2^{t-k} \le d_i < 2^{t-k}$ (in $O((t-k)2^{t-k}$ bit operations) and then all $H_i = h_i^{(2)} + d_i(2^{2^{k+1}}+1)$ (in $O(2^k\, 2^{t-k})$ bit operations).

We have shown: one multiplication in $\mathbb{Z}/(2^{2^t}+1)\mathbb{Z}$ can be reduced to $2^{t-k(t)+1}$ multiplications in $\mathbb{Z}/(2^{2^{k(t)}}+1)\mathbb{Z}$ plus $O(t2^t)$ bit operations, where $k(t) = [t/2] + 1$ and $t \ge 3$. Iterating until we reach $\mathbb{Z}/(2^4+1)\mathbb{Z}$ gives a multiplication method for $\mathbb{Z}/(2^{2^t}+1)\mathbb{Z}$ which uses $O(2^t t \log t)$ bit operations for one multiplication. □

As an immediate corollary we get the following improvement (for large n) of Lemma 4.

THEOREM 2. (Schönhage/Strassen). *Two positive integers with at most* n (≥ 3) *binary digits can be multiplied in* $O(n \log n \log\log n)$ *bit operations.*

PROOF. Choose $t \in \mathbb{N}$ minimal with $2n \le 2^t$. Then $2^t < 4n$ and multiplication $\bmod(2^{2^t}+1)$ of n-digit positive integers is just ordinary multiplication. Now apply Lemma 5. □

§4. DIVISION AND EVALUATION OF POLYNOMIALS.

The following theorem shows that division of polynomials can be performed about as fast as multiplication. The functions M(-) and A(-) are the ones introduced in Theorem 1.

THEOREM 3. *Let* T *be a commutative ring with* $1, 2^{-1} \in T$. *Let* F *and* G *in* T[X] *be given of degree at most* n (≥ 3) *and assume that the inverse of the leading*

coefficient of G *is given in* T. *Let the quotient* Q *and remainder* R *(in* T[X] *) be defined by* F = QG + R, deg(R) < deg(G). *One can compute* Q *and* R *in at most* 5M(2n) = O(n log n) *multiplications in* T *plus* 5A(2n) = O(n log n loglog n) *additions in* T.

PROOF. Let n = deg(F) and m = deg(G). We may assume that m ≤ n. We have

$$\sum_{i=0}^{n} f_i X^i = \left(\sum_{i=0}^{n-m} q_i X^i\right)\left(\sum_{i=0}^{m} g_i X^i\right) + \sum_{i=0}^{m-1} r_i X^i$$

where $F = \Sigma f_i X^i$, etc. Replacing X by X^{-1} and multiplying by X^n we obtain

$$(*) \qquad \sum_{i=0}^{n} f_i X^{n-i} \equiv \left(\sum_{i=0}^{n-m} q_i X^{n-m-i}\right)\left(\sum_{i=0}^{m} g_i X^{m-i}\right) \bmod X^{n-m+1}.$$

Let $H \in T[[X]]$ (the ring of power series in X with coefficients in T) be the inverse of $G^* := \Sigma_{i=0}^{m} g_i X^{m-i}$. We shall compute H mod X^{n-m+1}. Note that H satisfies f(H) = 0, where f(P) = 1/P − G^* for $P \in T[[X]]$. We shall compute approximations φ(P) to H by Newton-iteration:

$$\phi(P) = P - f(P)/f'(P) = P(2 - G^*P) = H - G^*(P-H)^2.$$

Suppose $P \equiv H \bmod X^k$. Then $\phi(P) \equiv H \bmod X^{2k}$. Given $P = H \bmod X^k$, one can compute $\phi(P) \bmod X^{2k}$ as follows: truncate G^* at X^{2k}, multiply by −P and add 2. Then multiply by P and truncate at X^{2k}. So given a k-precision approximation to H, we can compute a (2k)-precision approximation to H by 2 multiplications of polynomials in T[X] of degree less than 2k and one very simple addition. We start the iteration with $P = g^{-1}$ (= H mod X), where g is the leading coefficient of G. Let $A^*(k)$ and $M^*(k)$ denote the number of additions and multiplications in T, respectively, that the above method takes to compute H mod X^k. Then we have shown that

$$A^*(2k) \le A^*(k) + 2A(2k),$$

$$M^*(2k) \le M^*(k) + 2M(2k)$$

where A(−) and M(−) are the functions defined in Theorem 1. Since $A^*(1) = M^*(1) = 0$ and A(x)/x and M(x)/x are increasing functions it follows that

$$A^*(2^t) \le A^*(1) + 2\sum_{s=0}^{t-1} A(2^{t-s}) \le 2A(2^t)/2^t \sum_{s=0}^{t-1} 2^{t-s} < 4A(2^t),$$

$$M^*(2^t) < 4M(2^t).$$

Choose $t \in \mathbb{N}$ such that $2^t \ge n-m+1$, t minimal. Then $2^t \le 2(n-m)$ so we can compute $H \bmod X^{n-m+1}$ in at most $4A(2(n-m))$ additions in T plus $4M(2(n-m))$ multiplications in T. Now compute $(H \bmod X^{n-m+1}) \cdot (\Sigma_{i=0}^{n} f_i X^{n-i} \bmod X^{n-m+1})$ and truncate at X^{n-m+1}. In view of (*) we obtain $\Sigma_{i=0}^{n-m} q_i X^{n-m-i}$, hence Q. Then compute QG and $R = F - QG$ in one multiplication of polynomials of degrees $n-m$ and m and one addition of polynomials of degree at most n. The total costs for computing Q and R are, therefore, at most

$$\begin{cases} 4A(2(n-m)) + A(n-m) + A(\max\{m,n-m\}) + n & \text{additions in } T \\ 4M(2(n-m)) + M(n-m) + M(\max\{m,n-m\}) & \text{multiplications in } T. \end{cases}$$

Since $\alpha(x) = A(x)/x$ and $\mu(x) = M(x)/x$ are increasing and $\alpha(2x) \ge \alpha(x) + 1/2$ we obtain the bounds of Theorem 3. □

Using the method of Theorem 3 one can show that division of two integers can be done about as fast as multiplication. See [KNUTH], Section 4.3.3.D. Combining this with Theorem 2 gives:

COROLLARY 1. *One multiplication in* $\mathbb{Z}/t\mathbb{Z}$ *can be performed in* $O(\log t \operatorname{loglog} t \operatorname{logloglog} t)$ *bit operations.*

The next theorem concerns evaluation of a given polynomial at given points.

THEOREM 4. *Let* T *be a commutative ring with* $1, 2^{-1} \in T$. *Let* $x_1, \ldots, x_n \in T$ *be given and let* $F \in T[X]$ *be given as* $F = \Sigma_{i=0}^{n} f_i X^i$ *or* $F = \Pi_{i=1}^{n} (X-y_i)$, *where* $n \ge 3$. *The elements* $F(x_1), \ldots, F(x_n)$ *can be computed in at most* $O(n(\log n)^2 \operatorname{loglog} n)$ *additions in* T *plus* $O(n(\log n)^2)$ *multiplications in* T.

PROOF. (See Borodin and Munro). Let $n \le 2^t$ with $t \in \mathbb{N}$ minimal and put $x_i = 0$ for $n < i \le 2^t$. First we compute the coefficients of the auxiliary polynomials

$$G_{i,j} = \prod_{k=(j-1)2^{i+1}}^{j\cdot 2^i} (X-x_k), \quad \text{where } 0 \le i \le t,\ 1 \le j \le 2^{t-i}.$$

Let $0 \le i \le t-1$. Given the coefficients of the 2^{t-i} polynomials $G_{i,j}$ $(1 \le j \le 2^{t-i})$ of degree 2^i, we can compute all 2^{t-i-1} polynomials $G_{i+1,j}$ $(1 \le j \le 2^{t-i-1})$ of degree 2^{i+1} by 2^{t-i-1} multiplications of two polynomials of degree 2^i. So all $G_{i,j}$ can be computed using at most $\Sigma_{i=0}^{t-1} 2^{t-i-1}A(2^i)$ additions in T plus $\Sigma_{i=0}^{t-1} 2^{t-i-1}M(2^i)$ multiplications in T. Since $A(x)/x$ is increasing we have $\Sigma_{i=0}^{t-1} 2^{t-i-1}A(2^i) \le tA(2^{t-1})$. Similarly $\Sigma_{i=0}^{t-1} 2^{t-i-1}M(2^i) \le tM(2^{t-1})$. Assume now that the coefficients of F are given. Put $F_{t,1} = F$ and define $F_{i,j}$, where $0 \le i < t$, $1 \le j \le 2^{t-i}$, inductively as the remainder of $F_{i+1,[(j+1)/2]}$ after division by $G_{i,j}$. We have, for $0 \le i \le t$, $1 \le j \le 2^{t-i}$ and $(j-1)2^i < k \le j2^i$,

$$F_{i,j}(x_k) = \begin{cases} F_{i-1,2j-1}(x_k) & \text{if } (2j-2)2^{i-1} < k \le (2j-1)2^{i-1} \\ F_{i-1,2j}(x_k) & \text{if } (2j-1)2^{i-1} < k \le 2j2^{i-1}. \end{cases}$$

Hence evaluation of $F = F_{t,1}$ at $x_1,\ldots,x_{2^t}$ can be reduced to evaluation of $F_{t-1,1}$ and $F_{t-1,2}$ at $x_1,\ldots,x_{2^{t-1}}$ and $x_{2^{t-1}+1},\ldots,x_{2^t}$, respectively. In general, evaluation of $F_{i,j}$ at x_k where $(j-1)2^i < k \le j2^i$ can be reduced to evaluation of $F_{i-1,2j-1}$ and $F_{i-1,2j}$ at x_k for $(2j-2)2^{i-1} < k \le (2j-1)2^{i-1}$ and $(2j-1)2^{i-1} < k \le 2j2^{i-1}$, respectively. This process can be iterated until we reach $F_{0,j}$ $(1 \le j \le 2^t)$, which are constants, where evaluation is costless. What are the costs of the process? Given the auxiliary polynomials $G_{i,j}$, we have only to compute, successively, the remainders $F_{i,j}$. Since $F_{i+1,j}$ is of degree at most 2^{i+1} and $G_{i,2j-1}$ and $G_{i,2j}$ are of degree 2^i it follows from the last part of the proof of Theorem 3 that $F_{i,j}$ can be computed in at most $5A(2^{i+2})$ additions in T plus $5M(2^{i+1})$ multiplications in T. So the total costs for computing all $F_{i,j}$, given all $G_{i,j}$, can be bounded by $\Sigma_{i=0}^{t-1} 5A(2^{i+1})2^{t-i-1}$ additions in T plus $\Sigma_{i=0}^{t-1} 5M(2^{i+1})2^{t-i-1}$ multiplications in T. So the total costs, including those for computing all $G_{i,j}$, are at most

$$\begin{cases} 5tA(2^t) + tA(2^{t-1}) \le 5.5tA(2^t) & \text{additions in T,} \\ 5tM(2^t) + tM(2^{t-1}) \le 5.5tM(2^t) & \text{multiplications in T.} \end{cases}$$

Suppose now that $F \in T[X]$ is not given by its coefficients but as $F(X) = \Pi_{k=1}^{n} (X-y_k)$, where $y_1,\ldots,y_n \in T$ are given.

We have shown already that the coefficients of the auxiliary polynomial

$$G_{t,1} = \prod_{k=1}^{2^t} (X-x_k) = X^{2^t-n} \prod_{k=1}^{n} (X-x_k)$$

can be computed in at most $tM(2^{t-1})$ multiplications in T plus $tA(2^{t-1})$ additions in T, given $x_1,\ldots,x_n$. So the coefficients of F(X) can be computed, given the y_k, in at most $tM(2^{t-1}) \le \frac{1}{2}\, tM(2^t)$ multiplications in T plus at most $\frac{1}{2}\, tA(2^t)$ additions in T. So in this case evaluation of F at $x_1,\ldots,x_2t$ can be done in at most $6tA(2^t)$ additions in T plus $6tM(2^t)$ multiplications in T. Since $2^t \le 2n$ the assertion of Theorem 4 follows. □

The following corollary of Theorem 4 is used in the paper of Pomerance in this volume.

COROLLARY 2. *Let* $t \in \mathbb{N}$ *be odd and let* $y_1,\ldots,y_n \in \mathbb{Z}/t\mathbb{Z}$ *be given,* $n \le t$. *Put* $F(X) = \prod_{k=1}^{n} (X-y_k)$ *and let* $x_1,\ldots,x_n \in \mathbb{Z}/t\mathbb{Z}$ *be given. Then* $F(x_1),\ldots,F(x_n)$ *can be computed in* $O(n(\log n)^2 \log t \log\log t \log\log\log t)$ *bit operations.*

PROOF. Take T = $\mathbb{Z}/t\mathbb{Z}$ in Theorem 4. One addition in T can be done in $O(\log t)$ bit operations and one multiplication in T in $O(\log t \log\log t \log\log\log t)$ bit operations, by corollary 1 (undefined logarithms should be read as 1). Hence the additions in T give rise to $O(n(\log n)^2(\log\log n)\log t)$ bit operations and the multiplications to $O(n(\log n)^2 \log t \log\log t \log\log\log t)$ bit operations. So the total costs are at most $O(n(\log n)^2 \log t \max\{\log\log n, \log\log t \log\log\log t\})$ bit operations. □

REFERENCES

KNUTH, D.E., *The Art of Computer Programming*, Vol. 2 *Seminumerical Algorithms*, Addison Wesley, Reading, Massachusetts, 1981.

BORODIN, A. & I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier, New York, 1975.

PRIMALITY TESTING

by

H.W. LENSTRA, JR.

Two fundamental problems from elementary number theory are the following:

(a) (*primality*) given an integer n > 1, how can one tell whether n is prime or composite?

(b) (*factorization*) if n is composite, how does one find a,b > 1 such that n = ab?

Many mathematicians have been fascinated by these problems throughout history. Among these are ERATOSTHENES (~-284-~-204), FIBONACCI (~1180-~1250), FERMAT (1601-1665), EULER (1707-1783), LEGENDRE (1752-1833) and GAUSS (1777-1855). Some of the fascination of the subject derives from the fact that, roughly speaking, problem (a) is 'easy' and (b) is 'difficult'. Suppose, for example, that two 80-digit numbers p and q have been proved prime; this is easily within reach of the modern techniques for dealing with (a). Suppose further, that the cleaning lady gives p and q by mistake to the garbage collector, but that the product pq is saved. How to recover p and q? It must be felt as a defeat for mathematics that, in these circumstances, the most promising approaches are searching the garbage dump and applying mnemo-hypnotic techniques. The 'numerologists' occupying themselves with (a) and (b) do not accept this defeat. They imagine all composite numbers to be created by multiplication on the zeroth day of Creation, and they make it their task to unravel the mysteries involved in this process. In this connection, it is remarkable that no clairvoyants have ever been employed to identify Mersenne primes or to factor large numbers. Such an attempt might lead to new insights, if not in numerology then in parapsychology.

"Numerology" - this condescending term was, until recently, the fashionable one for the branch of science under discussion, in spite of the famous names listed above. Nowadays, a change in this attitude is noticeable. Partly, this change is due to an increased interest in general problems of

feasibility of computations. The revival of the specific problems (a) and (b) has, in addition, been stimulated by their striking application in cryptography. For the details of this application we refer to the lecture of P.J. HOOGENDOORN [10]. Suffice it to say that, in this application, it is essential that (a) is 'easy' and that (b) is 'hard'. It is an ironic fact that the only existing evidence for the 'hardness' of (b) is the failure of generations of 'numerologists' to come up with an efficient factorization algorithm.

This lecture is devoted to a discussion of problem (a). For (b) we refer to the contributions of C. POMERANCE [23] and M. VOORHOEVE [34] to this volume.

In complexity theory, it is customary to call an algorithm *good* if its running time is bounded by a polynomial in the length of the input. For problems (a) and (b) the input is the number n, which can be specified by $[\log n/\log 2] + 1$ binary digits. Thus the length of the input has the same order of magnitude as $\log n$.

A well known algorithm for solving (a) and (b) consists of trial divisions of n by the numbers less than or equal to $\sqrt{n}$. In the worst case, this takes $\sqrt{n}$ steps, which is exponential in the length of the input. We conclude that this algorithm is not 'good'.

Before one searches for a short proof that n is prime, or for a short proof that n is composite, it is a good question to ask whether such a proof exists. In this direction, we first have the following theorem; an *arithmetic operation* is the addition, subtraction or multiplication of two integers.

THEOREM 1. *If* n *is composite, this can be proved using only* $O(1)$ *arithmetic operations. Similarly if* n *is prime.*

PROOF. For composite n, the theorem is trivial; to prove that n is composite, it suffices to write down integers $a,b > 1$ and to do the single multiplication necessary to verify that $ab = n$. Thus, in the composite case, the O-symbol is even superfluous. For prime n, the theorem is less obvious. It is an outgrowth of the negative solution of HILBERT's tenth problem [6], that there exists a polynomial in twenty-six variables

$$f \in \mathbb{Z}[\underline{A},\underline{B},\underline{C},\ldots,\underline{X},\underline{Y},\underline{Z}]$$

with the property that the set of prime numbers coincides with the set of *positive* values assumed by f if non-negative integers are substituted for A,B,...,Z. Such a polynomial, of degree 25, is explicitly given in [11]. A similar polynomial in 10 variables of degree 11281 is constructed in [17; English translation]. To prove that a positive integer n is prime it now suffices to write down twenty-six non-negative integers A,B,...,Z and to do the bounded amount of arithmetic necessary to verify that n = f(A,B,...,Z). In fact, according to [11, Theorem 5] no more than 87 arithmetic operations are needed in this verification. This proves Theorem 1.

From a practical point of view Theorem 1 has two serious defects. The first is, that it tells us that certain proofs exist, but it does not tell us how to find them. Thus, F.N. Cole's proof that $2^{67}-1$ is composite consists of the single observation that

$$2^{67}-1 = 193707721 \cdot 761838257287.$$

But it had taken him 'three years of Sundays' to find his proof, and the methods that he employed are far more interesting than the final proof itself [5,25].

With primes, the situation is slightly different. The proof that, for prime n, there exist non-negative integers A,B,...,Z such that

$$n = f(A,B,\ldots,Z)$$

is completely constructive, see [11]. But for the polynomial from [11] it is not difficult to prove that the largest of A,B,...,Z necessarily exceeds

$$n^{n^{n^{n}}}.$$

(For a much better polynomial in this respect, see [1, Theorem 3.5].) The second defect of Theorem 1 is, that it is clearly unrealistic to count an addition or multiplication of numbers of this size as a single operation. It is more realistic to count *bit* operations, which may be defined as arithmetic operations on numbers of one digit. Thus, we have:

THEOREM 2. *If* n *is composite, this can be proved using only* $O((\log n)^2)$ *bit operations.*

PROOF. It suffices to remark that the usual algorithm to multiply two numbers less than n requires no more than $O((\log n)^2)$ bit operations. This proves Theorem 2.

Using the fast multiplication routine of SCHÖNHAGE and STRASSEN [27,32] we can replace $(\log n)^2$ in Theorem 2 by $(\log n)^{1+\varepsilon}$, for any $\varepsilon > 0$, or more precisely by $O((\log n)\cdot(\log\log n)\cdot(\log\log\log n))$ (for $n > e^e$).

THEOREM 3. (PRATT [25]). *If* n *is prime, this can be proved using only* $O((\log n)^4)$ *bit operations.*

Again, using [27], we can replace $(\log n)^4$ by $(\log n)^{3+\varepsilon}$, for any $\varepsilon > 0$.

PROOF. The proof relies on the structure of the group of units

$$(\mathbb{Z}/n\mathbb{Z})^* = \{(a \bmod n): a \in \mathbb{Z},\ 0 \le a < n,\ \gcd(a,n) = 1\}$$

of the ring $\mathbb{Z}/n\mathbb{Z}$ of integers modulo n. This is a finite abelian group of order $\phi(n)$, where ϕ is the Euler function. If n is a prime number, then $(\mathbb{Z}/n\mathbb{Z})^*$ is cyclic of order n-1. Conversely, if $(\mathbb{Z}/n\mathbb{Z})^*$ has order $\ge n-1$, then n is a prime number. Thus we see that n is prime if and only if there exists $(a \bmod n) \in (\mathbb{Z}/n\mathbb{Z})^*$ of order n-1. If we assume n to be odd and write

$$(1) \qquad n-1 = \prod_{i=0}^{k} q_i,$$

$$q_0 = 2$$

$$(2) \qquad q_i \text{ prime} \quad (1 \le i \le k)$$

then $(a \bmod n)$ has order n-1 in $(\mathbb{Z}/n\mathbb{Z})^*$ if and only if

$$(3) \qquad a^{(n-1)/2} \equiv -1 \bmod n,$$

$$(4) \qquad a^{(n-1)/q_i} \not\equiv 1 \bmod n, \quad \text{for } 1 \le i \le k.$$

Therefore, to prove that n is prime, we can write down integers a, $q_0 = 2$, $q_1,\ldots,q_k$, verify that (1), (3) and (4) hold, and prove (2) recursively. This proof requires k multiplications in (1), and k+1 exponentiations $(\bmod n)$ in (3) and (4), plus what is needed for (2). So if f(n) denotes the total

number of multiplications and exponentiations in the proof, then

$$f(n) \le k + k + 1 + \sum_{i=1}^{k} f(q_i)$$

where we define $f(2) = 1$. By induction we prove that $f(n) \le 3\cdot(\log n/\log 2) - 2$. This is true for $n = 2$, and if it holds for the q_i then

$$\begin{aligned} f(n) &\le 2k + 1 + \sum_{i=1}^{k} (3(\log q_i/\log 2) - 2) \\ &= \left(\sum_{i=0}^{k} 3(\log q_i/\log 2)\right) - 2 \\ &= 3(\log(n-1)/\log 2) - 2 < 3(\log n/\log 2) - 2 \end{aligned}$$

as required.

We conclude that no more than $O(\log n)$ multiplications and exponentiations are needed. Each exponentiation in (3), (4) can be done by $O(\log n)$ squarings and multiplications mod n. Finally, each of these multiplications, squarings and multiplications mod n (or mod a number smaller than n) can be done with $O((\log n)^2)$ bit operations. The total number of bit operations is therefore $O((\log n)\cdot(\log n)\cdot(\log n)^2) = O((\log n)^4)$. This proves Theorem 3.

Theorems 2 and 3 still have the first defect of Theorem 1: one is not told how to *find* the short proof whose existence is asserted. Nevertheless, the proof we have given of Theorem 3 is not exclusively of theoretical interest, and the same ideas are actually used in computer-assisted primality proofs. To illustrate this, we begin with a particularly simple case, in which n-1 has no odd prime factors at all.

THEOREM 4. (PÉPIN, 1877). *Let* $n = 2^m + 1$, *with* $m > 1$. *Then* n *is prime* $\iff$ $3^{(n-1)/2} \equiv -1 \bmod n$.

PROOF. The implication $\Leftarrow$ follows from the proof of Theorem 3, with $a = 3$. Conversely, suppose that n is prime. Then n is not divisible by 3, since $n > 3$, so m is even. Then $n \equiv 2 \bmod 3$ and $n \equiv 1 \bmod 4$, so quadratic reciprocity gives

$$\left(\frac{3}{n}\right) = \left(\frac{n}{3}\right) = \left(\frac{2}{3}\right) = -1.$$

By Euler's theorem, $\left(\frac{3}{n}\right) \equiv 3^{(n-1)/2} \bmod n$. This proves Theorem 4.

It is known that $n = 2^m+1$ can only be prime if m is a power of 2; then n is one of the *Fermat numbers* $F_k = 2^{2^k}+1$. For $k = 0,1,2,3,4$ these numbers are actually prime, for $5 \le k \le 19$ and some other values (such as $k = 1945$) they are known to be composite. It is reasonable to conjecture that they are, in fact, all composite for $k \ge 5$. The number F_{14} has been proved composite by Pépin's test, but no factor is known. To the uninitiated reader it may seem surprising that it is possible to prove that a number is composite, without the proof yielding a factorization. This is surprising indeed; the phenomenon will be further discussed at the end of this lecture. See [36, Sec. 5], [3] and [31] for more information on the Fermat numbers.

For general n, the main difficulty of the above test is to find the complete factorization (1) of n-1. In the following variant only a partial factorization of n-1 is needed.

THEOREM 5. *Let* n *and* s *be integers satisfying*

$$n > 1, \quad s > n^{1/2}.$$

Suppose that for every prime q *dividing* s *there exists an integer* a *(depending on* q*) satisfying*

$$(5) \qquad a^{q^{m(q)}} \equiv 1 \bmod n, \quad \gcd(a^{q^{m(q)-1}} - 1, n) = 1$$

where m(q) *denotes the number of factors* q *in* s. *Then* n *is a prime number.*

PROOF. Let r be any prime dividing n and q any prime dividing s. From (5) we see that the order of (a mod r) in the group $(\mathbb{Z}/r\mathbb{Z})^*$ equals $q^{m(q)}$, so by Lagrange's theorem $q^{m(q)}$ divides r-1. Since q is arbitrary, this implies that s divides r-1, so $r > s$. The inequality $s > n^{\frac{1}{2}}$ shows that n has at most one such prime factor. Hence n is prime, as required. This proves Theorem 5.

From the proof of Theorem 5 we see that the hypotheses imply that s divides n-1. To obtain a primality test from Theorem 5, one chooses s to be the largest divisor of n-1 that one is able to factor completely. For each q, the number a is constructed as follows. Search for an integer b such that

$$b^{n-1} \equiv 1 \bmod n, \quad b^{(n-1)/q} \not\equiv 1 \bmod n,$$

and put

$$a \equiv b^{(n-1)/q^{m(q)}} \bmod n.$$

If it is difficult to find such a number b, it is unlikely that n is prime, and one should attempt to show that n is composite, using Miller's method described below. The gcd in (5) is now equal to $\gcd(b^{(n-1)/q} - 1, n)$, and it can be calculated efficiently using Euclid's algorithm. In fact, only one gcd-computation is necessary if one considers the product of the numbers $b^{(n-1)/q} - 1 \bmod n$, with q ranging over the primes dividing s.

The critical condition of Theorem 5 is the inequality $s > n^{1/2}$ that must be satisfied by the completely factored part of n-1. There are several ways to replace this condition by a weaker one. Suppose, for example, that s only satisfies

$$s > n^{1/3}.$$

From the proof of Theorem 5 we see that every prime divisor of n is 1 mod s, and the same is then true for *every* divisor. Hence, if n is composite, there exist integers x and y satisfying

$$n = (xs+1)(ys+1), \quad x > 0, \; y > 0.$$

From $n < s^3$ it follows that $xy < s$, and $(x-1)(y-1) \geq 0$ now implies that $0 < x+y \leq s$. Since $x+y \equiv (n-1)/s \bmod s$ this means that we know the value of x+y. We also know that $n = (xs+1)(ys+1)$, so x and y can now be solved from a quadratic equation. Hence, if we add the hypothesis that the solution of this equation does not give rise to a non-trivial factorization of n, we still can conclude that n is a prime number.

A second method to relieve the condition $s > n^{1/2}$ makes use of lower bounds for the unknown prime factors of n-1. For a discussion of this technique, and references to the literature, see [36, Sections 10, 11].

Later in this lecture we shall consider a third type of generalization of Theorem 5, in which the role of n-1 is played by $n^t - 1$, where t is some positive integer; see Theorem 11.

G.L. MILLER [18] introduced a different way to exploit the multiplicative structure of the integers mod n in primality tests. It leads to the following theorem, in which "GRH" denotes the generalized Riemann hypothesis,

formulated in the course of the proof.

THEOREM 6. (MILLER). *Assume the validity of* GRH. *Then there exists an algorithm, described below, that in* $O((\log n)^5)$ *steps decides whether or not* n *is prime.*

This theorem has none of the defects of Theorems 1, 2 and 3, but it has a new one: the assumption of an unproved hypothesis.

Assume that n is odd, and write $n-1 = u \cdot 2^k$, where u is odd and $k \geq 1$. Employing RABIN's terminology [26], we call an integer a a *witness* to the compositeness of n, or simply a witness for n, if the following three conditions hold:

(6) n does not divide a,

(7) $a^u \not\equiv 1 \bmod n$,

(8) $a^{u \cdot 2^i} \not\equiv -1 \bmod n$ for $i = 0,1,\ldots,k-1$.

(Others say in this situation, that n is "not a strong base a pseudoprime" ...).

Whether or not a is a witness for n depends only on a mod n; so we may restrict to $0 \leq a < n$. For a given such a, it takes only $O((\log n)^3)$ steps to check whether or not a is a witness for n, by the last paragraph of the proof of Theorem 3.

We note that witnesses are reliable: if a is a witness to the compositeness of n, then n is composite. To see this, suppose that (6), (7), (8) hold and that n is prime. By (6) and Fermat's theorem, $a^{u \cdot 2^k} = a^{n-1} \equiv 1 \bmod n$. Hence the last term in the sequence

$$a^u, a^{u \cdot 2}, \ldots, a^{u \cdot 2^k}$$

is 1 mod n, but by (7) the first term is not 1 mod n. Let $b = a^{u \cdot 2^i}$ be the last term in the sequence that is not 1 mod n. Then $0 \leq i \leq k-1$, and $b^2 \equiv 1 \bmod n$ while $b \not\equiv 1 \bmod n$. Since the integers mod n form a field, this implies that $b \equiv -1 \bmod n$, contradicting (8).

The algorithm referred to in Theorem 6 now runs as follows. We may assume that n is odd, and $n > 1$. Check whether there is a witness a for n

satisfying $0 < a < 70(\log n)^2$. If there is one, n is composite. If there is none, declare n to be prime. This algorithm clearly runs in time $O((\log n)^5)$.

To prove the correctness of the algorithm, we have to show that any composite odd n has a positive witness $a < 70(\log n)^2$, if GRH is assumed. We sketch this proof only, referring to the literature for details.

First we describe the GRH as we need it. Let n be an arbitrary positive integer, and let $\chi: (\mathbb{Z}/n\mathbb{Z})^* \to \mathbb{C}^*$ (the group of non-zero complex numbers) be a group homomorphism. We view χ as a function on $\mathbb{Z}$ by $\chi(a) = \chi(a \bmod n)$ if $\gcd(a,n) = 1$, and $\chi(a) = 0$ otherwise. Such a function on $\mathbb{Z}$ is called a *character* modulo n. The L-series associated to χ is defined by

$$L(s,\chi) = \sum_{a=1}^{\infty} \frac{\chi(a)}{a^s} .$$

If χ is non-trivial, i.e. $\chi(a) \notin \{0,1\}$ for some a, this series converges for all $s \in \mathbb{C}$ with $\mathrm{Re}(s) > 0$. We say that $L(s,\chi)$ satisfies the generalized Riemann hypothesis if $L(s,\chi) \neq 0$ for all $s \in \mathbb{C}$ with $\mathrm{Re}(s) > \frac{1}{2}$. For trivial χ, this is only meaningful if $L(s,\chi)$ has been analytically continued; to avoid this, let us simply say that $L(s,\chi)$, for trivial χ, satisfies the generalized Riemann hypothesis if and only if the classical Riemann hypothesis is true, which is equivalent to

$$\sum_{a=1}^{\infty} \frac{(-1)^a}{a^s} \neq 0 \quad \text{for all } s \in \mathbb{C} \text{ with } \tfrac{1}{2} < \mathrm{Re}(s) < 1.$$

The GRH in Theorem 6 is the conjunction of all generalized Riemann hypotheses described above.

LEMMA. (ANKENY-MONTGOMERY). *There is an absolute constant* c *with the following property. Let* χ *be a non-trivial character modulo* n, *and suppose that* $L(s,\chi)$ *satisfies the generalized Riemann hypothesis. Then there exists* $a \in \mathbb{Z}$, $0 < a < c\cdot(\log n)^2$, *such that* $\chi(a) \neq 0$ *and* $\chi(a) \neq 1$.

PROOF. See [20, Theorem 13.1], or [12, Corollary 1.3] for a version in which also the classical Riemann hypothesis is needed.

COROLLARY. *Assume* GRH, *and let* $G \neq (\mathbb{Z}/n\mathbb{Z})^*$ *be a subgroup of* $(\mathbb{Z}/n\mathbb{Z})^*$. *Then there exists* $a \in \mathbb{Z}$ *such that*

$$0 < a < c\cdot(\log n)^2, \qquad \gcd(a,n) = 1, \qquad (a \bmod n) \notin G,$$

with c *as in the lemma.*

PROOF. It suffices to apply the lemma to a non-trivial χ: $(\mathbb{Z}/n\mathbb{Z})^* \to \mathbb{C}^*$ that is trivial on G.

Let now n > 1 be composite and odd. To finish the proof of Theorem 6, with an unspecified constant c instead of 70, it suffices, by the corollary, to exhibit a proper subgroup $G \subset (\mathbb{Z}/n\mathbb{Z})^*$ containing all non-witnesses a that are not divisible by n. For this we take (cf. [33])

$$G = \{(a \bmod n) \in (\mathbb{Z}/n\mathbb{Z})^* : a^{(n-1)/2} \equiv (\tfrac{a}{n}) \bmod n\}$$

where $(\frac{a}{n})$ is the Jacobi symbol. It is a charming theorem of LEHMER [13, cf. 30] that $G \neq (\mathbb{Z}/n\mathbb{Z})^*$ for composite odd n. It is an equally charming result of SELFRIDGE [36, Theorem 17.2] that G contains all non-witnesses (mod n) not divisible by n. This finishes the proof of Theorem 6.

Using additional arguments it can be proved that the generalized Riemann hypothesis is only needed for the L-series associated to characters χ of the form $\chi(a) = (\frac{a}{d})$, where d runs over the positive integers that are 1 mod 4 and either prime or the product of two distinct primes, see [15].

The value 70 for the constant is taken from [21, Théorème 4]; here again the classical Riemann hypothesis is needed, in addition to the generalized Riemann hypotheses just described. It is reported that Weinberger (unpublished) obtained sharper results.

The idea used in the proof of Theorem 6 has two other applications. The first is a fast primality test for small numbers:

THEOREM 7. (SELFRIDGE & WAGSTAFF). *Every odd composite* n

satisfying:	*has a witness among:*
$n < 2047$	2
$n < 1373653$	2,3
$n < 2 \cdot 10^9$, $n \neq 25326001, 161304001, 960946321, 1157839381$	2,3,5
$n < 25 \cdot 10^9$, $n \neq 3215031751$	2,3,5,7

PROOF. By computer, see [24]. This proves Theorem 7.

The numbers in the left hand column are composite:

$2047 = 23 \cdot 89$,
$1373653 = 829 \cdot 1657$,
$25326001 = 2251 \cdot 11251$,
$161304001 = 7333 \cdot 21997$,
$960946321 = 11717 \cdot 82013$,
$1157839381 = 24061 \cdot 48121$,
$3215031751 = 151 \cdot 751 \cdot 28351$.

The test provided by Theorem 7 is easily implemented on a programmable pocket calculator. Thus, an HP-41C can decide the primality of an arbitrary $n < 2 \cdot 10^9$ within two minutes, using only 2, 3, 5 as possible witnesses.

The second application is based on the following theorem.

THEOREM 8. (RABIN). *Every odd composite* n *has at least* $\frac{3}{4}(n-1)$ *witnesses among* $\{1,2,\ldots,n-1\}$.

The proof is an attractive exercise in elementary number theory, in which the Carmichael numbers play a role. See [26,19]. This proves Theorem 8.

Rabin proposes the following primality test. Let m be a large integer, like 100, and choose randomly m integers $a_i \in \{1,2,\ldots,n-1\}$, $1 \leq i \leq m$. If one of these a_i is a witness for n, then n is composite. If none of the a_i is a witness for n, then either n is prime or we have extremely bad luck. By Theorem 8, this bad luck occurs in at most one out of every 4^m cases. While this method is basically incapable of yielding rigorous primality proofs, it is difficult to doubt the correctness of the answers. In any case, Rabin's method can be used to produce primes on a commercial basis: if found defective, they can easily be replaced.

If we try to remove the unproved assumption from Theorem 6 we are left with an algorithm that is no longer 'good':

THEOREM 9. (ADLEMAN, POMERANCE & RUMELY). *There is an algorithm that in* $O((\log n)^{c' \log\log\log n})$ *steps decides whether or not* n *is prime, for* $n > e^e$. *Here* c' *denotes an effectively computable constant.*

A complete proof of this theorem can be found in [2] and [16]. A probabilistic version of the algorithm, which is somewhat easier to explain, will be described below. This version of the algorithm has been implemented by H. Cohen on the CDC-Cyber 170-750 computer of the SARA Computer Centre in Amsterdam, cf. [4]. It is the only primality test in existence that can routinely handle numbers of up to 100 decimal digits, and it does so within approximately 45 seconds.

The algorithm that we shall describe can be viewed as a special case of the following primality criterion.

THEOREM 10. *Let* n > 1 *be an integer. Then* n *is prime if and only if every divisor of* n *is a power of* n.

The proof is left to the reader.

To prove that n is prime using Theorem 10 we must check that any divisor of n is a power of n, and it clearly suffices to consider only *prime* divisors of n. Below we shall see how to do this without explicitly knowing the prime divisors of n. Actually, something weaker will be done: rather than showing that a prime r dividing n is a power of n, one attempts to show that this is true for the images of r and n in certain auxiliary groups, such as the group $(\mathbb{Z}/s\mathbb{Z})^*$ for an integer s that is coprime to n.

An example of this approach is provided by Theorem 5 and its proof: in that theorem we have $n \equiv 1 \bmod s$, and the proof proceeds by showing that any prime divisor r of n satisfies $r \equiv 1 \bmod s$, i.e. is congruent to a power of n modulo s. The following theorem provides a less trivial example.

THEOREM 11. *Let* n *and* s *be positive integers, and let* A *be a commutative ring with* 1 *containing* $\mathbb{Z}/n\mathbb{Z}$ *as a subring (with the same* 1*). Suppose that there exists* $\alpha \in A$ *satisfying the following conditions:*

(9) $\quad \alpha^s = 1,$

(10) $\quad \alpha^{s/q} - 1 \in A^*$ *(the group of units of* A*) for every prime* q *dividing* s,

(11) *the polynomial* $\prod_{i=0}^{t-1} (X-\alpha^{n^i})$ *has coefficients in* $\mathbb{Z}/n\mathbb{Z}$ *for some positive integer* t.

Then every divisor r *of* n *is congruent to a power of* n *modulo* s.

PROOF. We may assume that r is *prime*. Since r is a zero divisor (or zero) in A, there exists a maximal ideal M of A with $r \in M$. Let $\bar{A}$ be the field A/M, and $\bar{\alpha} = (\alpha \bmod M) \in \bar{A}$. By (9) and (10), the order of $\bar{\alpha}$ in $\bar{A}^*$ equals s. The polynomial $\prod_{i=0}^{t-1} (X-\bar{\alpha}^{n^i})$, which has $\bar{\alpha}$ as a zero, has coefficients in the subfield $\mathbb{F}_r$ of $\bar{A}$ of cardinality r. Therefore $\bar{\alpha}^r$ is also a zero of this polynomial, so there exists $i \in \{0,1,\ldots,t-1\}$ with $\bar{\alpha}^r = \bar{\alpha}^{n^i}$, i.e.

$r \equiv n^i \bmod s$. This proves Theorem 11.

If we take $A = \mathbb{Z}/n\mathbb{Z}$ and $t = 1$, then condition (11) is trivially satisfied. It is easy to deduce Theorem 5 from Theorem 11, by choosing α equal to the product of the a's appearing in Theorem 5, taken modulo n.

The proof of Theorem 11 shows that the residue classes $(n^i \bmod s)$, $0 \le i < t$, are permuted upon multiplication by $(r \bmod s)$, for any prime r dividing n. Writing n as the product of its prime factors, we see that multiplication by $(n \bmod s)$ also permutes these residue classes, which just means that $n^t \equiv 1 \bmod s$. Hence s must be chosen to be a divisor of $n^t - 1$.

Let $t = 2$. In this case known prime factors of $n+1 = (n^2-1)/(n-1)$ can be used in addition to those of $n-1$ to build up the number s. Starting from Theorem 11 one can, for practically every primality test based on factors of $n-1$, devise a corresponding test based on factors of $n+1$. These tests are usually formulated in terms of LUCAS functions [36, Sections 12,13,14]. In the simplest case, corresponding to Pépin's Theorem 4, the number $n+1$ is a power of 2:

THEOREM 12. (LUCAS-LEHMER). *Let* $n = 2^m - 1$, *with* $m > 2$. *Define* $(e_k)_{k=1}^{\infty}$ *by* $e_1 = 4$, $e_{k+1} = e_k^2 - 2$. *Then* n *is prime if and only if* $e_{m-1} \equiv 0 \bmod n$.

PROOF. First let m be *even*. Then n is divisible by 3, and not prime. Also $e_{m-1} \equiv -1 \bmod 3$ by induction, so $e_{m-1} \not\equiv 0 \bmod n$. This proves the theorem for even m. Assume now that m is *odd*, and define

$$A = (\mathbb{Z}/n\mathbb{Z})[T]/(T^2 - \sqrt{2}T - 1),$$

where $\sqrt{2}$ denotes any element of $\mathbb{Z}/n\mathbb{Z}$ with $\sqrt{2}^2 = 2$; e.g., $\sqrt{2} = (2^{(m+1)/2} \bmod n)$. Denoting the image of T in A by α we have

$$A = \{a + b\alpha : a,b \in \mathbb{Z}/n\mathbb{Z}\}, \quad \alpha^2 = \sqrt{2}\alpha + 1.$$

Let $\beta = \sqrt{2} - \alpha = -\alpha^{-1}$ be "the" other zero of $X^2 - \sqrt{2}X - 1$ in A. From $\alpha + \beta = \sqrt{2}$ and $\alpha\beta = -1$ it follows easily by induction on k that

$$\alpha^{2^k} + \beta^{2^k} = (e_k \bmod n) \in \mathbb{Z}/n\mathbb{Z}$$

for all $k \ge 1$. Now let first n be prime. The discriminant of $X^2 - \sqrt{2}X - 1$ equals 6, and from $n \equiv 1 \bmod 3$, $n \equiv -1 \bmod 8$ and quadratic reciprocity it follows that $(\frac{6}{n}) = -1$. Hence A is a quadratic field extension of $\mathbb{F}_n$, and α

and β are conjugate over $\mathbb{F}_n$. By the theory of finite fields this implies that $\alpha^n = \beta$. Multiplying this by α we get $\alpha^{2^m} = -1$, so $(e_{m-1} \bmod n) = \alpha^{2^{m-1}} + \beta^{2^{m-1}} = \alpha^{2^{m-1}} + \alpha^{-2^{m-1}} = 0$. This proves the "only if" part. Suppose, conversely, that $(e_{m-1} \bmod n) = 0$. Then

$$\alpha^{2^m} = -1, \quad \alpha^{2^{m+1}} = 1,$$

so (9) and (10) of Theorem 11 are satisfied with $s = 2^{m+1}$. Also, $\alpha^n = \alpha^{2^m-1} = -\alpha^{-1} = \beta$, so the polynomial

$$(X-\alpha)(X-\alpha^n) = (X-\alpha)(X-\beta) = X^2 - \sqrt{2}\cdot X - 1$$

has coefficients in $\mathbb{Z}/n\mathbb{Z}$, which is condition (11) of Theorem 11 with $t = 2$. From Theorem 11 and $n^2 \equiv 1 \bmod s$ it now follows that any divisor of n is congruent to 1 or n modulo s. But $s > n$, so this means that n is prime. This proves Theorem 12.

It is known that $n = 2^m - 1$ can only be prime if m is prime: then n is one of the *Mersenne numbers* $M_p = 2^p - 1$, p prime. These are known to be prime for 27 values of p:

2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497,

and composite for all other $p < 50000$, see [29] and [31]. It is reasonable to conjecture that $\#\{m < x: 2^m-1 \text{ is prime}\}/\log x$ tends to a finite non-zero limit for $x \to \infty$. GILLIES [8] gives a probabilistic argument leading to the value $2/\log 2$ for the limit, but his reasoning is clearly in error since the same argument leads to a contradiction with the prime number theorem, cf. [9, §22.20]. The number $e^\gamma/\log 2$, where γ is Euler's constant, has been proposed as a more likely value for the limit [22]; see also [35].

If the complete factorization of $n-1$ is known then in practice it is easy to test n for primality, e.g. using Theorem 5. The same statement is true with $n-1$ replaced by $n+1$, using Theorem 11 with $t = 2$. A combination of both tests is employed in the discovery of large *twin primes*, in the following way. Let m be a large number whose complete prime factorization is known; such a number can be found by multiplying together small numbers.

Then (m+1) - 1 and (m−1) + 1 are completely factored, so we can apply an (n−1)-primality test to m+1 and an (n+1)-primality test to m−1. If both numbers turn out to be prime we have found a pair of twin primes. The largest known pair is

$$256200945 \cdot 2^{3426} \pm 1 = 2^{3426} \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 113 \cdot 151 \pm 1,$$

which have 1040 decimal digits. This pair was discovered by ATKIN and RICKERT [7].

We next discuss how Theorem 11 can for general t be used for primality testing. For A one takes a ring that if n were prime would be the field $\mathbb{F}_{n^t}$ of n^t elements. If n behaves as if it were a prime number then such a ring is in practice not difficult to construct: as in the proof of Theorem 12 one can take $A = (\mathbb{Z}/n\mathbb{Z})[T]/(f)$, where $f \in (\mathbb{Z}/n\mathbb{Z})[T]$ is a polynomial of degree t that passes a suitable irreducibility test (see [14, Sec. 5]). For s one takes the largest divisor of n^t-1 that one is able to factor completely, and for α one takes an element of A^* of order s. If n is actually prime then α is usually easy to construct, by manipulating with elements of the form $\beta^{(n^t-1)/s}$, $\beta \in A$. In this case conditions (9) and (10) are clearly satisfied, and the polynomial in (11) is a power of the irreducible polynomial of α over $\mathbb{F}_n$ so has certainly coefficients in $\mathbb{F}_n$. Suppose, conversely, that (9), (10) and (11) are found to be true. Then we cannot immediately conclude that n is prime, but we know, by Theorem 11, that any r dividing n is congruent to a power of n modulo s. If s is sufficiently large then this information can be used to finish the primality proof, in the following manner. Suppose that

$$s > n^{1/2}$$

(as in Theorem 5), and let r_i be determined by

$$n^i \equiv r_i \bmod s, \quad 0 \le r_i < s$$

for $0 \le i < t$. If n is composite then it has a non-trivial divisor r with $r \le n^{1/2} < s$, and since r is congruent to a power of n modulo s it must be equal to one of the r_i. Hence, if we verify that none of the r_i is a non-trivial divisor of n, we have proved that n is prime. A similar but somewhat more involved procedure can be followed if s satisfies the weaker

inequality $s > n^{1/3}$.

We refer to [16, Theorem (8.4)] for a more flexible version of Theorem 11, in which it is possible to vary α with q, as in Theorem 5.

For very small values of t, such as t = 2, 3, 4, 6, it is again possible to employ lower bounds for the unknown prime divisors of $n^t - 1$, cf. [36, Sections 13-16] and the references given there. It is doubtful whether such lower bounds are equally useful for the larger values of t considered below.

To analyze the above algorithm we must know how to choose t such that $s > n^{1/2}$. We need the following theorem.

THEOREM 13. (ODLYZKO-POMERANCE). *There exists an effectively computable constant* c'' *with the following property. For every integer* $n > e^e$ *there exists a positive integer* t *satisfying*

$$t < (\log n)^{c'' \operatorname{loglog} \log n}$$

t is squarefree

such that the number

$$s = \prod_{q \text{ prime},\ q-1 \text{ divides } t} q$$

satisfies

$$s > n^{1/2}.$$

PROOF. See [2, Sec. 6]. This proves Theorem 13.

Let t be as in Theorem 13; the condition that t be squarefree is irrelevant for our present purpose. If q is a prime number for which q-1 divides t, then $n^t \equiv 1 \bmod q$ by Fermat's theorem, unless q divides n. Hence, if s is as in the theorem, then s divides $n^t - 1$ provided that gcd(n,s) = 1. Also, the complete factorization of s is known, and $s > n^{1/2}$. We conclude that these values for t and s can be used in the primality test described above. The resulting algorithm has, for prime n, an expected running time that is less than $(\log n)^{c' \operatorname{loglog} \log n}$ for some constant c'. This does not yet prove Theorem 9, since we have no such bound for the worst case running time. It appears that the size of t makes the test unsuitable for practical primality testing.

The test underlying Theorem 9 is closely related to the test just described. It depends on properties of *Gaussian sums*, which we shall now consider. By ζ_m we denote a primitive m-th root of unity.

Let p and q be prime numbers not dividing n for which p divides q-1. We choose a character $\chi = \chi_{p,q}$ modulo q that has order p; i.e., $\chi: \mathbb{F}_q^* \to \langle\zeta_p\rangle$ is a surjective group homomorphism, where $\langle\zeta_p\rangle$ denotes the subgroup of $\mathbb{C}^*$ generated by ζ_p. Such a χ can be obtained by choosing a primitive root g modulo q and putting $\chi(g^i \bmod q) = \zeta_p^i$ for $i \in \mathbb{Z}$. We define the *Gaussian sum* $\tau(\chi)$ by

$$\tau(\chi) = -\sum_{x=1}^{q-1} \chi(x)\zeta_q^x.$$

This is an element of the cyclotomic ring $R = \mathbb{Z}[\zeta_p, \zeta_q]$. We have

$$\tau(\chi)^n \equiv \chi(n)^{-n} \cdot \tau(\chi^n) \bmod nR \quad \text{if n is } \textit{prime}.$$

To prove this, notice that modulo nR we have

$$\begin{aligned}\tau(\chi)^n &\equiv -\sum_{x=1}^{q-1} \chi(x)^n \cdot \zeta_q^{nx} \quad \text{(since n is prime)}\\ &= -\sum_{y=1}^{q-1} \chi(n)^{-n} \cdot \chi(y)^n \cdot \zeta_q^y \quad \text{(with } y \equiv nx \bmod q)\\ &= \chi(n)^{-n} \cdot \tau(\chi^n),\end{aligned}$$

as required. We investigate what can, conversely, be said about n if the following weaker condition is satisfied:

$$\tau(\chi)^n \equiv \eta(\chi)^{-n} \cdot \tau(\chi^n) \bmod nR \quad \text{for some } \eta(\chi) \in \langle\zeta_p\rangle. \tag{12}$$

Let σ be the automorphism of R with $\sigma(\zeta_p) = \zeta_p^n$ and $\sigma(\zeta_q) = \zeta_q$. Then (12) can be written as

$$\tau(\chi)^{n-\sigma} \equiv \eta(\chi)^{-n} \bmod nR.$$

Raising both sides to the power $\sum_{i=0}^{p-2} n^{p-2-i}\sigma^i$ we obtain:

$$\tau(\chi)^{n^{p-1}-1} \equiv \eta(\chi) \bmod nR.$$

Now let r be any prime divisor of n. Then we know that (12), with n replaced by r and $\eta(\chi)$ by $\chi(r)$, is valid, so for the same reason we have

$$\tau(\chi)^{r^{p-1}-1} \equiv \chi(r) \bmod rR.$$

Combination of the last two congruences suggests that

$$\chi(r) = \eta(\chi)^{(r^{p-1}-1)/(n^{p-1}-1)} \tag{13}$$

for any prime r dividing n. To make this meaningful we have to explain how to interpret the fractional exponent. For this we need the following hypothesis on p:

$$v_p(r^{p-1}-1) \geq v_p(n^{p-1}-1) \quad \text{for every prime } r \text{ dividing } n, \tag{14}$$

where $v_p(m)$ denotes the number of factors p in m. If (14) is satisfied we can write $(r^{p-1}-1)/(n^{p-1}-1) = a/b$, with $a,b \in \mathbb{Z}$, $b \equiv 1 \bmod p$, and the residue class of $(r^{p-1}-1)/(n^{p-1}-1) \bmod p$ is then defined to be $(a \bmod p)$; this does not depend on the choice of a and b. Since $\eta(\chi)^p = 1$ it is now meaningful to define the right hand side of (13) as $\eta(\chi)^a$.

With this interpretation it is straightforward to verify that (12) implies (13), provided that (14) is assumed. By induction on the number of prime factors one can now prove that (13) holds for *any* divisor r of n, prime or not. In particular, with r = n we obtain $\chi(n) = \eta(\chi)$, so (13) now yields

$$\chi(r) = \chi(n)^{(r^{p-1}-1)/(n^{p-1}-1)} \tag{15}$$

for any r dividing n. Again we see that every divisor of n is a power of n, if images under χ are taken.

It is a vital question how to verify hypothesis (14). Trivially, we have

$$\text{if } n^{p-1} \not\equiv 1 \bmod p^2, \text{ then (14) holds.} \tag{16}$$

In [16, Sec. 2] it is proved that

$$\text{if (12) holds with } \eta(\chi) \neq 1, \text{ then (14) is true.} \tag{17}$$

The primality test based on the preceding theory runs as follows. Let t be a positive integer having all properties listed in Theorem 13, and let s have the same meaning as in that theorem. Choose, for every pair of prime numbers p, q with q dividing s and p dividing q-1 (so p dividing t) a character $\chi = \chi_{p,q}$ as above, and check that $\chi = \chi_{p,q}$ satisfies (12); we know that this is necessary for n to be prime. Next, attempt to prove that every prime p dividing t satisfies hypothesis (14). Usually, for each p there is a q dividing s with $\eta(\chi_{p,q}) \neq 1$, and then (17) applies. If there is no such q, and (16) does not apply either, one should test (12) for characters $\chi_{p,q}$ with q a prime *not* dividing s for which p divides q-1, until an example of $\eta(\chi_{p,q}) \neq 1$ is found.

At this stage of the algorithm one knows that every $\chi_{p,q}$, with p dividing q-1 and q dividing s, satisfies (15) for each r dividing n. We claim that this implies that each r is congruent to a power of n modulo s, so that the test can be completed in the same way as the test described before Theorem 13.

To prove the claim, let r divide n, and let (i mod t) be determined by

$$i \equiv (r^{p-1}-1)/(n^{p-1}-1) \bmod p$$

(in the sense explained before) for each prime p dividing t; notice that here we use that t is squarefree. Then (15) implies that

$$\chi_{p,q}(r) = \chi_{p,q}(n^i)$$

for each pair p, q as above. For fixed q, the product of the primes p dividing q-1 equals q-1, so the characters $\chi_{p,q}$ generate the group of all characters modulo q; therefore $r \equiv n^i \bmod q$. Since this holds for all q dividing s, we conclude that $r \equiv n^i \bmod s$, as required.

The only non-deterministic part of the test is the verification of hypothesis (14). If n is composite it is conceivable that (14) is not satisfied, so that the algorithm will get stuck at this point. We refer to [2, Sec. 5] and [16, Sec. 5] for a variant that avoids hypothesis (14). It constructs an auxiliary number ν such that from a set of conditions similar to (12) it can be deduced that any divisor r of n is congruent to a power of ν, rather than a power of n, modulo s. This test is completely deterministic, and it has running time less than $(\log n)^{c' \log\log\log n}$ for $n > e^e$, where c' denotes an effectively computable constant. This concludes our sketch of

the proof of Theorem 9.

There are several ways to improve the practical performance of the test [4,16]. In the first place, the Gaussian sums can be replaced by Jacobi sums, which belong to $\mathbb{Z}[\zeta_p]$ rather than $\mathbb{Z}[\zeta_p,\zeta_q]$. Secondly, characters of prime power order rather than of prime order can be employed, so that the condition that t be squarefree can be dropped. Finally, it is possible to combine the test with the tests described earlier depending on variants of Theorem 11. However, none of these improvements reduces the running time in a theoretically significant way.

As we noted in connection with the Fermat numbers, it is surprising that we can prove that a number is composite without actually finding a factor. To analyze this situation, let us assume that we proved n composite by exhibiting an integer a for which

$$a^{n-1} \not\equiv 1 \bmod n, \quad \gcd(a,n) = 1, \tag{18}$$

and applying Fermat's theorem that (18) is impossible for prime n. To see why this gives no factorization of n we must investigate how Fermat's theorem is proved. One proof is based on the remark that the map sending i to $a\cdot i \pmod n$ is a permutation of $\{1,2,\ldots,n-1\}$, so

$$a^{n-1}\cdot(n-1)! = \prod_{i=1}^{n-1} (a\cdot i) \equiv \prod_{i=1}^{n-1} i = (n-1)! \bmod n.$$

Hence (18) implies that (n-1)! has a non-trivial gcd with n, which tells us nothing more than that n is composite. Other proofs of Fermat's theorem have similar shortcomings. The situation would be different if factorials or binomial coefficients were easy to compute modulo n. This is clear from the proof of the following charming but useless theorem, in which we also consider 'division with remainder' as an arithmetic operation.

THEOREM 14. (SHAMIR). *There is an algorithm that for every composite* n *yields a non-trivial divisor of* n, *using no more than* $O(\log n)$ *arithmetic operations.*

PROOF. We notice that n is composite if and only if $1 < \gcd(a_0!,n) < n$ for some positive integer a_0. Since gcd(a!,n) is a non-decreasing function of a, and is equal to 1, n for a = 1, n, respectively, we can determine such an a_0 by O(log n) bisections, provided that we know how to calculate gcd(a!,n).

Once we know a!, we can determine the gcd by Euclid's algorithm in O(log n) arithmetic steps. To calculate a!, we apply the formulae

$$(2b+1)! = (2b+1)\cdot(2b)!,$$

$$(2b)! = (b!)^2\cdot\binom{2b}{b}$$

O(log a) times. To calculate the binomial coefficient $\binom{2b}{b}$ needed here, we remark that $\binom{2b}{b}$ is the middle block of n binary digits in the binary expansion of $(2^n+1)^{2b}$, for $2b < n$; and the exponentiation can be done by O(log(2b)) multiplications.

This algorithm, as we described it, takes $O((\log n)^3)$ arithmetic operations. For the modifications to bring it down to O(log n) we refer to SHAMIR's paper [28]. This concludes the proof of Theorem 14.

We notice that the best known deterministic factorization algorithm, which is due to POLLARD, also depends on the calculation of factorials modulo n. This algorithm and several more practical ones are described in the papers of POMERANCE [23] and VOORHOEVE [34].

REFERENCES

[1] ADLEMAN, L. & K. MANDERS, *Diophantine complexity*, 17th Annual IEEE Symp. on Foundations of Computer Science (1976), 81-88.

[2] ADLEMAN, L.M., C. POMERANCE & R.S. RUMELY, *On distinguishing prime numbers from composite numbers*, to appear. Extended abstract: 21st Annual IEEE Symp. on Foundations of Computer Science (1980), 387-406.

[3] BRENT, R.P. & J.M. POLLARD, *Factorization of the eighth Fermat number*, Math. Comp. 36 (1981), 627-630.

[4] COHEN, H. & H.W. LENSTRA, JR., *Primality testing and Jacobi sums*, to appear. Preliminary version: H. COHEN, *Tests de primalité d'après Adleman, Rumely, Pomerance et Lenstra*, Séminaire de Théorie des Nombres, Université Scientifique et Médicale de Grenoble, June 1981.

[5] COLE, F.N., *On the factoring of large numbers*, Bull. Amer. Math. Soc. 10 (1903/4), 134-137.

[6] DAVIS, M., *Hilbert's tenth problem is unsolvable*, Amer. Math. Monthly 80 (1973), 233-269.

[7] GARDNER, M., *Mathematical games*, Scientific American 244 (2) (Feb. 1981), 14-19.

[8] GILLIES, D.B., *Three new Mersenne primes and a statistical theory*, Math. Comp. 18 (1964), 93-97.

[9] HARDY, G.H. & E.M. WRIGHT, *An introduction to the theory of numbers*, 5th ed., Oxford University Press, (1979).

[10] HOOGENDOORN, P.J., *On a secure public-key cryptosystem*, this volume.

[11] JONES, J.P., D. SATO, H. WADA & D. WIENS, *Diophantine representation of the set of prime numbers*, Amer. Math. Monthly 83 (1976), 449-464.

[12] LAGARIAS, J.C., H.L. MONTGOMERY & A.M. ODLYZKO, *A bound for the least prime ideal in the Chebotarev density theorem*, Invent. Math. 54 (1979), 271-296.

[13] LEHMER, D.H., *Strong Carmichael numbers*, J. Austral. Math. Soc. Ser. A 21 (1976), 508-510.

[14] LENSTRA, A.K., *Factorization of polynomials*, this volume.

[15] LENSTRA, H.W., JR., *Miller's primality test*, Inform. Process. Lett. 8 (1979), 86-88.

[16] LENSTRA, H.W., JR., *Primality testing algorithms (after Adleman, Rumely and Williams)*, Séminaire Bourbaki 33 (1980/1981), no. 576, pp. 243-257 in: Lecture Notes in Mathematics 901, Springer, Berlin, 1981.

[17] MATIJASEVIČ, YU.V., *Primes are nonnegative values of a polynomial in 10 variables*, Zap. Naučn. Sem. Leningrad. Otdel. Mat. Inst. Steklov (LOMI) 68 (1977), 62-82 (Russian; English translation: J. Soviet Math. 15 (1981), 33-44).

[18] MILLER, G.L., *Riemann's hypothesis and tests for primality*, J. Comput. System Sci. 13 (1976), 300-317.

[19] MONIER, L., *Evaluation and comparison of two efficient probabilistic primality testing algorithms*, Theoret. Comput. Sci. 12 (1980), 97-108.

[20] MONTGOMERY, H.L., *Topics in multiplicative number theory*, Lecture Notes in Mathematics 227, Springer-Verlag, Berlin (1971).

[21] OESTERLÉ, J., *Versions effectives du théorème de Chebotarev sous l'hypothèse de Riemann généralisée*, Journées Arithmétiques de Luminy, Astérisque 61 (1979), 165-167.

[22] POMERANCE, C., *Recent developments in primality testing*, Math. Intell. 3 (1981), 97-105.

[23] POMERANCE, C., *Analysis and comparison of some integer factoring algorithms*, this volume.

[24] POMERANCE, C., J.L. SELFRIDGE & S.S. WAGSTAFF, JR., *The pseudoprimes to* $25 \cdot 10^9$, Math. Comp. 35 (1980), 1003-1026.

[25] PRATT, V.R., *Every prime has a succinct certificate*, SIAM J. Comput. 4 (1975), 214-220.

[26] RABIN, M.O., *Probabilistic algorithm for testing primality*, J. Number Theory 12 (1980), 128-138.

[27] SCHÖNHAGE, A. & V. STRASSEN, *Schnelle Multiplikation grosser Zahlen*, Computing 7 (1971), 281-292.

[28] SHAMIR, A., *Factoring numbers in* O(log n) *arithmetic steps*, Inform. Process. Lett. 8 (1979), 28-31.

[29] SLOWINSKI, D., *Searching for the 27th Mersenne prime*, J. Recreational Math. 11 (1978/9), 258-261.

[30] SOLOVAY, R. & V. STRASSEN, *A fast Monte-Carlo test for primality*, SIAM J. Computing 6 (1977), 84-85; erratum: 7 (1978), 118.

[31] TE RIELE, H.J.J., *Perfect numbers and aliquot sequences*, this volume.

[32] TURK, J.W.M., *Fast arithmetic operations on numbers and polynomials*, this volume.

[33] VÉLU, J., *Tests for primality under the Riemann hypothesis*, SIGACT News 10 (1978), 58-59.

[34] VOORHOEVE, M., *Factorization algorithms of exponential order*, this volume.

[35] WAGSTAFF, S.S., JR., *Divisors of Mersenne numbers*, to appear.

[36] WILLIAMS, H.C., *Primality testing on a computer*, Ars Combin. 5 (1978), 127-185.

FACTORIZATION ALGORITHMS OF EXPONENTIAL ORDER

by

M. VOORHOEVE

1. INTRODUCTION

Until recently, the factorization of large integers was not considered a decent subject for mathematicians, but the advent of the Rivest-Shamir-Adleman cryptosystem (see HOOGENDOORN [4]) has increased its interest.

As POMERANCE [11] describes, the fastest algorithms known for the factorization of an integer N require a time and memory proportional to a power of L, where $L = \exp(\sqrt{(\log N \log\log N)})$. In this paper, we limit ourselves to the algorithms that are $O(N^{\alpha})$, where α is a fixed positive real number. Since the length of the input of the factorization problem is $\log N$, these are the "purely exponential" algorithms.

Although slower than the L^{α} algorithms, the N^{α} algorithms have the advantage of simplicity. Any factorization attempt of a number N should start with a search for small factors by trial division. Once the small divisors are eliminated, the remaining quotient must be tested for primality. After these tests, when it is known that N is not prime and has no small divisors, and only then it is worthwhile to unchain the heavy machinery that are the L^{α} algorithms.

Even then the simple methods can be useful, as is demonstrated by the factorization of $2^{256} + 1$ by the Pollard-rho method.

We shall treat the algorithms in order of increasing asymptotic speed and thus decreasing α.

$\alpha = 1/2$	–	Trial division
$\alpha = 1/3$	–	Sherman Lehman
	–	Rivest-Pinter
	–	Lenstra

$\alpha = 1/4$ - Pollard-rho
- Lehmer-Powers/SQUFOF
- Pollard-Strassen

$\alpha = 1/5$ - Shanks class group

We conclude with a brief evaluating section.

In the above classification and the remainder of the paper, powers of $\log N$ are neglected. For instance, a multiplication or square root extraction of a number of size proportional to N is considered as one step in the algorithm.

Note that we can limit ourselves to the problem of finding one prime factor of our number N without increasing the exponent α; this is so because N has less than $^2\log N$ prime factors.

2. TRIAL DIVISION

The algorithm consits of dividing N by the primes up to $N^{1/2}$. To save memory, one usually divides - instead of by primes - by numbers from a larger but simpler class, e.g. {2,3 and $6k \pm 1$}. The trial divisors can be performed in any order, but as a standard, one begins with the small primes, having the greatest "chance" of success. Further see KNUTH & TRABB PARDO [5].

3. SHERMAN LEHMAN

If we find two numbers A, B such that $A^2 \equiv B^2 \pmod N$, $A \not\equiv \pm B \pmod N$, then gcd(A+B,N) gives a nontrivial factor of N. This idea is combined with trial division and gives a completely deterministic $O(N^{1/3})$ algorithm. We give a brief description, also compare LEHMAN [6].

Step 1. Trial divide N up to $[N^{1/3}]$. If no factors are found, N is either prime or the product of two prime factors p, q with $N^{1/3} < p \leq q < N^{2/3}$.

Step 2. For $k = 1,2,\ldots,[N^{1/3}]$; $d = 0,1,\ldots,[N^{1/6}/4\sqrt{k} + 1]$, test whether

$$([\sqrt{4kN}] + d)^2 - 4kN$$

is a square. If so, we have found numbers A and B as described above. Since A+B and $|A-B|$ both lie between 1 and N for $N > 6$, this gives a nontrivial

factor of N and, in view of step 1, the complete factorization.

THEOREM. *If we find no factor in this way,* N *is prime.*

PROOF. Suppose $N = pq$, with $N^{1-3} < p \le q < N^{2/3}$. We assert that there exist $r,s \in \mathbb{N}$ such that $rs < N^{1/3}$ and $|ps-qr| < N^{1/3}$. With $k = rs$ we then have that

$$4kN = (ps+qr)^2 - (ps-qr)^2.$$

So $(ps+qr)^2 - 4kN$ is a square less than $N^{2/3}$. Put $d = ps + qr - [\sqrt{4kN}]$. Then

$$N^{2/3} > (ps+qr)^2 - 4kN > 2(d-1)\sqrt{4kN}.$$

So

$$d < \frac{N^{2/3}}{2\sqrt{4kN}} + 1$$

and step 2 finds the factor. □

PROOF OF THE ASSERTION. Let r_n/s_n be the n-th convergent of the continued fraction expansion of q/p. Note that $r_1 = [q/p]$, $s_1 = 1$, so that $0 < r_1 s_1 < N^{1/3}$. Take m such that

(1) $$r_m s_m < N^{1/3};$$

(2) $$r_{m+1} s_{m+1} > N^{1/3}.$$

We take $r,s = r_m,s_m$. By (1), $rs < N^{1/3}$. Furthermore we have

$$\left|\frac{r}{s} - \frac{q}{p}\right| \le \left|\frac{r}{s} - \frac{r_{m+1}}{s_{m+1}}\right| = \frac{1}{s\, s_{m+1}}.$$

Suppose $p/s_{m+1} < q/r_{m+1}$. Then

$$|rp-qs| \le \frac{p}{s_{m+1}} < \sqrt{\frac{pq}{r_{m+1}s_{m+1}}} < N^{1/3}.$$

If $p/s_{m+1} > q/r_{m+1}$, note that $\frac{s}{r}$, s_{m+1}/r_{m+1} are convergents of $\frac{p}{q}$ and proceed by interchanging p and q as well as r and s. This proves the assertion. □

4. RIVEST-PINTER

Essentially, this method consists of searching for integer points on the hyperbola $xy = N$ in the plane. By approximating the hyperbola with tangents, likely candidates are found and checked. This leads to an $O(N^{1/3})$ algorithm. However, the rigorous analysis of the computing time is incomplete. See MONIER [9], Ch.5.

5. LENSTRA

LENSTRA [8] proved the following result

THEOREM. *Let* r, s, N *be positive integers, such that*

$$s > N^{1/3}, \quad \gcd(r,s) = 1, \quad 0 < r < s.$$

Then there is a fast algorithm $(O(\log N)^{\alpha})$ *to find all divisors* d *of* N *such that*

$$d \equiv r \pmod{s}.$$

This result (of separate interest) can be applied to factoring by choosing $s = [N^{1/3}] + 1$ and applying Lenstra's algorithm for all r (mod s). This clearly gives an $O(N^{1/3})$ algorithm.

6. POLLARD-RHO

A random sequence on k points needs to have on average $O(k^{\frac{1}{2}})$ terms for a repetition to occur. For some $b,c \in \mathbb{Z}$ and a prime p we define the sequence $a_1, a_2, \ldots$ inductively by

$$a_1 = c; \quad a_k = a_{k-1}^2 + b \pmod{p}.$$

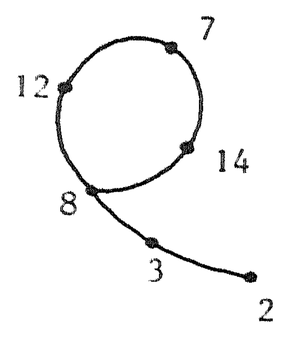
If $b \neq 0,-2$ and if c is not badly chosen, this looks like a random sequence on the $\frac{1}{2}(p+1)$ squares plus b modulo p. (If $b = 0$ we get the sequence c^{2^k}; if $b = -2$ we get a similar sequence, since $(t+t^{-1})^2 - 2 = t^2 + t^{-2}$.) Hence one expects that there are $t > s$, $t = O(p^{\frac{1}{2}})$ such that $a_t = a_s$. From now on $a_{t+1} = a_{s+1}, \ldots, a_{2t-s} = a_t = a_s, \ldots$ and the sequence becomes cyclic. So there

exists an e such that $a_e = a_{2e}$. The least such e - which is the least multiple of t - s not less than s - is called the *epact* e(p) pf p. The expected value of e(p) equals

$$E(e(p)) = (\pi^5 p/288)^{\frac{1}{2}} \sim 1.03\ p^{\frac{1}{2}}.$$

As an example, we take p = 17, b = -1, c = 2 and get

$$a_1 = 2; \quad a_2 = 3$$
$$a_3 = 8; \quad a_4 = 12$$
$$a_5 = 7; \quad a_6 = 14$$
$$a_7 = 8 = a_3$$
$$a_8 = a_4 = 12, \text{ etc.}$$



So e(17) = 4. The graphic representation of the sequence is the "rho" after which the method is named.

The algorithm consists of calculating a_{2k}, a_k modulo N and determining $\gcd(a_{2k}-a_k,N)$. If $p|N$ we find the factor p in k = e(p) steps. One step consists of three multiplications and a gcd evaluation. Occasionally, the gcd becomes N, when the prime factors of N have the same epact. Then we start with a new b. In general we find a factor p in $O(p^{\frac{1}{2}})$ steps, but there is no guaranteed maximum number of steps. It has the same advantage as trial division of finding small factors fast - but the price we have to pay is the guarantee of success.

A great success for this algorithm was the factorization of the eighth Fermat number $2^{256}+1$. In this case, one was lucky that the number has one prime factor much smaller than the other. See BRENT & POLLARD [2].

When looking at the algorithm it seems wasteful to compute the a_k's, since they have been computed earlier when the sequence a_{2k} was generated. Using a modified period-finding algorithm and some extra memory these wasteful computations can be avoided. For details, consult SEDGEWICK, SZYMANSKI & YAO [13] and BRENT [1]. Also, the gcd evaluation in each step need not be effectuated, but it can be replaced by a cumulative multiplication. The gcd is evaluated with the cumulative product after a large number of steps. Compare POLLARD [10].

7. LEHMER-POWERS / SQUFOF

We first describe a straightforward and simple method depending on continued fractions. Like in Sherman Lehman, many numbers R are generated such that $R^2 \pmod N$ is small compared to N, in the hope that $R^2 \pmod N$ is itself the square of a number T.

In this case the R's are convergents of the continued fraction expansion of $\sqrt{N}$. More exactly, the algorithm runs as follows.

1. Compute the convergents r_n/s_n of $\sqrt{N}$ and

$$d_n = r_n^2 - Ns_n^2.$$

2. If d_n is a perfect square, compute the gcd

$$(r_n + \sqrt{d_n}, N).$$

3. We are finished if this gcd lies between 1 and N.

Every second d_n is positive and $<2\sqrt{N}$, so its "probability" of being a square is $1:O(N^{1/4})$, whereas the "probability" that $r_n \not\equiv \pm\sqrt{d_n} \pmod N$ is *a priori* $\frac{1}{2}$. There is a heuristic argument that suggests that the expected number of steps of this algorithm is $0.89N^{1/4}$, for N the product of two primes. There is no guaranteed maximum number of steps, though; for N of the form $m^2 + 1$ the algorithm does not even terminate.

A refinement of the above method was suggested by LEHMER & POWERS [7]. It has developed into the Brillhart-Morrison algorithm, which has running time $O(L^{\alpha})$; see POMERANCE [11].

A second descendant is SQUFOF by D. Shanks. This algorithm is rooted in the theory of quadratic forms. It computes the numbers d_n (the same as above) as the last coefficients of reduced quadratic forms of discriminant 4N. These forms are derived from each other by reduction, starting with

$$X^2 + 2[\sqrt{N}]XY + d_1Y^2.$$

(for the reduction algorithm, see SCHOOF [12], Section 4). When a square d_n is found, a test rejects the d_n's leading to a trivial factorization, i.e. $(r_n + \sqrt{d_n}, N) = 1$ or N in the above terminology. Once a "good" square d_n is found, a form with $\sqrt{d_n}$ as last coefficient is generated and the reduction

is performed backwards. In approximately half as many steps as in the "forward" loop, the factor is found.

The strong point of SQUFOF is that all numbers arising in the computations of the two loops are less than $2\sqrt{N}$. This considerably speeds up the arithmetic. With respect to the method described above, however, this speed-up is no more than a constant factor.

8. POLLARD-STRASSEN

POMERANCE [11] mentions the following result in §4.
Let y *be a square and let* $z = \sqrt{y}$. *For every* t, *the least prime factor of* (t,y!) *can be found in*

$$O(z(\log^2 z + \operatorname{loglog} t)\log t.\operatorname{loglog} t.\operatorname{logloglog} t)$$

bit operations.

Taking $z = [N^{1/4} + 1]$, $t = N$, we have a fully deterministic and proved $O(N^{1/4})$ algorithm. In contrast with the previous two $O(N^{1/4})$ algorithms, this algorithm has mainly theoretical significance.

9. SHANKS CLASS GROUP

The algorithm is discussed *in extenso* by SCHOOF [12].

10. A COMPARISON

In this section we compare some of the above algorithms from a practical viewpoint.

TRIAL DIVISION. This is solid, simple and slow. Yet it is absolutely reliable and the small factors, having the greatest chance of success, come first. It is the fastest factorization algorithm for numbers up to 10^5 (MONIER [9], page C4).

SHERMAN LEHMAN. This algorithm has been neglected somewhat. However, like trial division it is absolutely guaranteed and it works in *parallel*. This means that all steps in the algorithm are independent of each other, which implies that they can be performed independently by different processors. With the advent of multiprocessor systems ("ultracomputers"), parallelism

in algorithms might become an important issue.

POLLARD-RHO. Very practical and simple. It resembles trial division, as small factors come first. However, no parallelism or guarantee of success. Such a guarantee though, is not so important in practice. Once the number is factorized - by whatever ramshackle method - the factorization itself is immediately proved by multiplying the factors.

SQUFOF. This algorithm is very practical as it works with half-size ($\leq 2\sqrt{N}$) numbers. This can considerably speed up the arithmetic - on a programmable pocket calculator it can factor numbers of up to 20 digits.

SHANKS. Very sophisticated: it can compete with the $O(L^{\alpha})$ methods for numbers up to 10^{40} (almost the limit of what can be reached at present) cf. MONIER [9]. Also, the algorithm admits parallelism to some extent. In a way, success is guaranteed as long as the Generalized Riemann Hypothesis is not refuted: either you factorize or you find a counter example to GRH.

11. REFERENCES

For a general overview and a more extensive bibliography, one is referred to MONIER [9] and GUY [3].

[1] BRENT, R.P., *An improved Monte Carlo factorization algorithm*, BIT 20 (1980), 176-184.

[2] BRENT, R.P. & J.M. POLLARD, *Factorization of the eighth Fermat number*, Math. Comp. 36 (1981), 627-630.

[3] GUY, R.K., *How to factor a number*, Congressus Numerantium XVI, Proc. Fifth Manitoba Conf. on Numerical Math., Winnipeg, 1976, pp. 49-89.

[4] HOOGENDOORN, P.J., *On a secure public-key cryptosystem*, this volume.

[5] KNUTH, D.E. & L. TRABB PARDO, *Analysis of a simple factorization algorithm*, Theor. Comp. Sci. 3 (1976), 321-348.

[6] LEHMAN, R.S., *Factoring large integers*, Math. Comp. 28 (1974), 637-646.

[7] LEHMER, D.H. & R.E. POWERS, *On factoring large numbers*, Bull. Amer. Math. Soc. 37 (1931), 770-776.

[8] LENSTRA, H.W. JR, *Divisors in residue classes*, in preparation.

[9] MONIER, L., *Algorithmes de factorisation d'entiers*, Thesis, Université Paris Sud, 1980.

[10] POLLARD, J.M., *A Monte Carlo method for factorization*, BIT 15 (1975), 331-334.

[11] POMERANCE, C., *Analysis and comparison of some integer factoring algorithms*, this volume.

[12] SCHOOF, R., *Quadratic fields and factorization*, this volume.

[13] SEDGEWICK, R., T.G. SZYMANSKI & A.C. YAO, *The complexity of finding cycles in periodic functions*, SIAM J. Comput. 11 (1982), 376-390.

ANALYSIS AND COMPARISON OF SOME INTEGER FACTORING ALGORITHMS

by

C. POMERANCE *)

1. INTRODUCTION

Although the problem of how to efficiently factor an integer is centuries old, in recent years an extraordinary amount of interest has been focused on the issue. There are at least two reasons. First, the problem has taken on a glamorous tinge because of its connection to the public-key cryptosystem scheme of Rivest, Shamir and Adleman (see HOOGENDOORN [12]). Second, and perhaps more fundamental, with the explosive growth of computers in society, the analysis of natural algorithmic problems, including factoring, has grown into an area of its own.

It should be pointed out that unlike with the companion subject of primality testing (see LENSTRA [19], POMERANCE [24]), there have been no recent dramatic breakthroughs concerning factoring. Nevertheless, advances have been and are being made. Many of the new ideas are based on the seminal MORRISON-BRILLHART continued fraction algorithm [21]. With this algorithm composite numbers of 50 digits with no small prime factors can be factored (although this is probably about the limit of the basic algorithm).

Although the continued fraction algorithm has worked in practice, no rigorous analysis has ever been given for its running time. However, DIXON [9] recently proposed a simplified version for which a rigorous running time analysis could be provided. Dixon's algorithm is not a serious contender as a practical method for factoring. Rather, its value lies in the respectability it brings both to the continued fraction algorithm and the heuristic argument supporting the continued fraction algorithm discussed below.

Another similar algorithm, due to SCHROEPPEL [28], ingeniously replaces time consuming trial divisions required in the continued fraction algorithm and Dixon's algorithm with a procedure based on the sieve of Eratosthenes.

---

*) Research partially supported by an NSF grant.

To my knowledge, Schroeppel's algorithm has not proved practical.

In what follows I shall be concerned with carefully analyzing the running times of these algorithms and how certain variations affect these times. Where possible, rigorous arguments are given or sketched. At other times, an argument is presented that relies on certain explicit but unproved hypotheses. There has been a certain amount of disagreement in the literature concerning the running times of the various algorithms. For example, with

$$L(n) = \exp\{(\log n \log\log n)^{\frac{1}{2}}\}$$

(all logarithms in this paper are natural logarithms), various people have suggested that the continued fraction algorithm has a running time of $L(n)^{c+o(1)}$ on input of a composite number n, with the value of c given as $\sqrt{2}$ (SCHROEPPEL [28], MONIER [20]), 2 (KNUTH [13]), and $\sqrt{3/2}$ (WUNDERLICH [34], [35] - although Wunderlich also includes in his analysis a variation which below is called Large Prime). It is my hope that the careful analysis in this paper will settle the controversy (provided the unproved hypotheses are not controversial). On the particular issue of the continued fraction algorithm, I side with Schroeppel and Monier, that is $c = \sqrt{2}$ (with or without the Large Prime variation). Unlike prior results, the running time estimates given in this paper are two-sided, not just upper bounds.

A natural variation of the continued fraction algorithm (aspects of this idea are described in MORRISON-BRILLHART [21], Remarks 4.2, 4.6 and in KNUTH [13] p.384) is what is called below the early abort strategy. I have carefully analyzed the effect of this variation with different choices of certain parameters and have come up with a particular choice of parameters that are asymptotically optimal. It is hoped that these choices can guide factorers of finite numbers.

In addition, proposed below is a new algorithm called the quadratic sieve. It belongs to the same family as the other algorithms considered and appears to be very fast. It is based on the well known fact that the consecutive values of a quadratic polynomial (in fact, any polynomial over the integers) can be factored with a sieve. The quadratic sieve algorithm is very similar to a procedure suggested long ago by KRAITCHIK [15] (also see MORRISON-BRILLHART [21], p.199). In fact it may be fair to say that the relationship of the quadratic sieve algorithm to Kraitchik's method is analogous to the relationship of the modern continued fraction algorithm to an earlier version due to LEHMER and POWERS [18]. The quadratic sieve algorithm can also be viewed as a natural outgrowth of Schroeppel's algorithm. In preliminary work by GERVER [10], it appears to be computer practical.

Each of the algorithms discussed has a running time of the form $L(n)^{\alpha+o(1)}$ where n is the (composite) number being factored, α is a constant that depends on the algorithm and L(n) is defined above. However there is an important dichotomy between Dixon's algorithm together with its variations and the other algorithms discussed. The former can be analyzed rigorously, but are probabilistic algorithms. The latter are deterministic algorithms, but as mentioned above, the analysis contains certain leaps of faith.

By a "deterministic algorithm" we shall mean one in which the input completely pre-determines everything that follows. In contrast, by a "probabilistic algorithm" we mean one in which random numbers are employed. Such an algorithm tries to take advantage of lucky breaks that cannot be guaranteed to occur, but by the law of large numbers are expected. For a probabilistic algorithm we may talk of the probability that the running time is such and so. Indeed 100 independent implementations on the same input could lead to 100 different running times and perhaps even to different outputs (for example, different factorizations of n).

When we say a deterministic factoring algorithm has a running time of $L(n)^{\alpha+o(1)}$, we mean that the running time to factor n is precisely $L(n)^{\alpha+\theta(n)}$ where $\theta(n)$ is some function that tends to 0 as $n \to \infty$ through the composites. When we say a probabilistic factoring algorithm has a running time of $L(n)^{\alpha+o(1)}$ we mean that there is a function $\theta(n)$ which tends to 0 as $n \to \infty$ through the composites such that for every $\varepsilon > 0$, there is an M such that with probability at least $1 - \varepsilon$ the running time to factor n lies between $M^{-1}L(n)^{\alpha+\theta(n)}$ and $M \cdot L(n)^{\alpha+\theta(n)}$. That is, on most implementations of the algorithm for n the running time will not be too many times greater or less than $L(n)^{\alpha+\theta(n)}$. To emphasize this difference, when speaking of a probabilistic algorithm we shall say it has an *expected* running time of $L(n)^{\alpha+o(1)}$.

With Dixon's algorithm and its variations, probability enters the picture because the algorithms involve randomly choosing some auxiliary numbers. With the other algorithms, the auxiliary numbers are deterministically produced. In the heuristic analyses for these deterministic algorithms certain explicit hypotheses are made that essentially say that the auxiliary numbers are "pseudo-random" with respect to certain properties. Unfortunately I do not presently know how to prove these hypotheses. Thus it cannot be ruled out that at almost every instantiation of the algorithm in question the running time is much less than what is claimed - or much more.

The main results are summarized in the table. The exponents given are on the base L(n). Thus, for example, Dixon's algorithm with the Pollard-

Strassen variation has expected running time $L(n)^{\sqrt{3}+o(1)}$ and space requirements of $L(n)^{\sqrt{4/3}+o(1)}$. Reiterating what was said above, only the numbers for Dixon's algorithm and its variations have rigorous proofs.

Dixon's algorithm with the Pollard-Strassen method, the early abort strategy, and Coppersmith-Winograd elimination stands as the current champion of fully proved factoring algorithms with an expected running time of $L(n)^{\sqrt{5/2}+o(1)}$. The fastest fully proved *deterministic* factoring algorithm is the Pollard-Strassen method discussed below in Section 4. The running time is $O(n^{1/4}(\log n)^3 \operatorname{loglog} n \operatorname{logloglog} n)$. Shanks has a deterministic factoring algorithm (see SCHOOF [27]) which, if the Extended Riemann Hypothesis holds, has a running time of $O(n^{1/5+\varepsilon})$ for every $\varepsilon > 0$. If the quadratic sieve algorithm with Coppersmith-Winograd elimination (discussed in Section 7) can be rigorously analyzed along our heuristic lines, it would be a deterministic algorithm with running time less than $L(n)^{1.0204}$ for all large composite n.

It should be pointed out that just because algorithm A has higher exponents than algorithm B does not necessarily make it the worse algorithm for all values of n. It may be that B is superior only for those $n > 10^{10^{10}}$ and for smaller n, A is the algorithm of choice. A little is said in Section 9 about the practical aspects of the algorithms considered.

It is suggested on a first reading that the proofs, especially those in Sections 2, 3 and 4, be skipped or skimmed. Many of the ideas presented are quite simple and can be understood even without knowing all of the intervening details.

This paper covers only a small fraction of the many types of factoring algorithms known. For alternative treatment of some of the algorithms presented here plus descriptions of many others, the reader's attention is called to BRENT and POLLARD [4], GUY [11], KNUTH [13], MONIER [20], SCHNORR [26], SCHOOF [27], and VOORHOEVE [33].

Among the many people who helped me with this paper I should like to acknowledge and thank Edward Azoff, John Brillhart, Rod Canfield, Andrew Odlyzko, Robert Rumely, Samuel Wagstaff, Hugh Williams, and especially Hendrik Lenstra.

Basic Algorithm	Variation	Elimination Method	Time Exponent		Space Exponent	
Dixon		Gauss	2		1	
	P-S	Gauss	1.732	$\sqrt{3}$	1.155	$\sqrt{4/3}$
	EAS	Gauss	1.732	$\sqrt{3}$	1.155	$\sqrt{4/3}$
	P-S & EAS	Gauss	1.604	$\sqrt{18/7}$	1.069	$\sqrt{8/7}$
	P-S & EAS	C-W	1.581	$\sqrt{5/2}$	1.265	$\sqrt{8/5}$
Morrison-Brillhart (continued fraction)		Gauss	1.414	$\sqrt{2}$	0.707	$\sqrt{1/2}$
	P-S	Gauss	1.225	$\sqrt{3/2}$	0.816	$\sqrt{2/3}$
	EAS	Gauss	1.225	$\sqrt{3/2}$	0.816	$\sqrt{2/3}$
	P-S & EAS	Gauss	1.134	$\sqrt{9/7}$	0.756	$\sqrt{4/7}$
	P-S & EAS	C-W	1.118	$\sqrt{5/4}$	0.894	$\sqrt{4/5}$
Schroeppel (linear sieve)		Gauss	1.5		1	
		C-W	1.248		1	
	Cull	Gauss	1.225	$\sqrt{3/2}$	0.816	$\sqrt{2/3}$
	Cull	C-W	1.117		0.895	
Quadratic Sieve		Gauss	1.061	$\sqrt{9/8}$	0.707	$\sqrt{1/2}$
		C-W	1.020		0.818	

Abbreviations: P-S, Pollard-Strassen method
EAS, Early abort strategy
C-W, Coppersmith-Winograd method

2. PRELIMINARIES AND A CONVENTION

The function $L(n) = \exp(\sqrt{\log n \operatorname{loglog} n})$ introduced in the previous section will often simply be denoted L. The expression $L(n)^{\alpha+o(1)}$, where α is a non-zero constant, will often be abbreviated L^{α}. With this unusual convention, seemingly absurd equations can be obtained, such as

$$2L^{\alpha} = L^{\alpha}, \quad L^{\alpha} + L^{\beta} = L^{\max\{\alpha,\beta\}}, \quad \pi(L^{\alpha}) = L^{\alpha},$$

where in the last equation π is the prime counting function. This convention serves to streamline many arguments and to have the important magnitudes

stand out. I hope it does not prove too confusing for the reader. In any event, all theorems will be stated without the convention.

A ubiquitous function from number theory that will be of use here is $\psi(x,y)$, the number of natural numbers not exceeding x free of prime factors exceeding y. We cite the following theorem from CANFIELD, ERDÖS, POMERANCE [7]:

THEOREM 2.1. *If* ε *is an arbitrary positive constant, then uniformly for* $x \geq 10$ *and* $y \geq (\log x)^{1+\varepsilon}$,

$$\psi(x,y) = x \cdot u^{-u+o(u)},$$

where $u = (\log x)/\log y$.

The assertion about uniformity means that there is a function $\theta(u)$ such that $\theta(u)/u \to 0$ as $u \to \infty$ and such that if $f(x,y)$ is defined by the equation

$$\psi(x,y) = x \cdot u^{-u+f(x,y)}$$

then $|f(x,y)| \leq \theta(u)$ for all $x \geq 10$, $y \geq (\log x)^{1+\varepsilon}$. In fact, in [7], it is shown that

$$f(x,y) = -u \frac{(\log\log u)-1}{(\log u)-1} + O\left(u \frac{(\log\log u)^2}{(\log u)^3}\right),$$

where again the error term is uniform. Actually, in [7] only the lower bound implicit in Theorem 2.1 is shown - the upper bound is found in an earlier paper of DE BRUIJN [6].

Combining Theorem 2.1 with our unusual convention, we have, for example,

$$\psi(n,L^a) = n \cdot L^{-(2a)^{-1}}.$$

In fact we shall always apply Theorem 2.1 when $y = L(x)^{a+o(1)}$, where a is a positive constant. For this restricted range of y we have a result that is more general than Theorem 2.1 that we shall occasionally need.

Let $\psi(x,y,z)$ denote the number of natural numbers not exceeding x composed solely of the primes in the interval $(z,y]$. So, for example, $\psi(x,y,1) = \psi(x,y)$.

THEOREM 2.2. *Let* $\varepsilon > 0$ *be arbitrary. If* $u = (\log x)/\log y$ *and if*

$z < y^{1-1/\log u}$, *then*

$$\psi(x,y,z) = x \cdot u^{-u+o(u)}$$

uniformly for $(\log x)^{1+\varepsilon} < y < \exp((\log x)^{1-\varepsilon})$, $x \geq 10$.

PROOF. The upper bound follows from Theorem 2.1 since $\psi(x,y,z) \leq \psi(x,y)$. We may assume x is large. Let

$$w_1 = y^{1-2/(3\log u)}, \quad w_2 = y^{1-1/(3\log u)}.$$

Then

$$\psi(x,y,z) \geq \sum_m{}' \psi(x/m,w_1,z) \tag{2.1}$$

where the sum is over those m which are the product of [u] not necessarily distinct primes in $(w_1,w_2]$. Indeed for each integer k counted by $\psi(x/m,w_1,z)$, we have $km \leq x$ and every prime in km is in $(z,w_2] \subset (z,y]$. Moreover, if $km = k'm'$ where m' is also the product of [u] primes in $(w_1,w_2]$ and k' is counted by $\psi(x/m',w_1,z)$, then $m = m'$, $k = k'$.

Note that if m is the product of [u] primes in $(w_1,w_2]$, then

$$w_1^{u-1} < m \leq w_2^u,$$

so that for large x

$$x^{1/\log u} > y \cdot x^{2/(3\log u)} > x/w_1^{u-1} > x/m \geq x/w_2^u = x^{1/(3\log u)}. \tag{2.2}$$

Let $u(m) = 1 + [(\log(x/m))/\log w_1]$. Since $\log w_1 \sim \log y$ as $x \to \infty$, we have from (2.2)

$$\frac{u}{\log u} \gtrsim u(m) \gtrsim \frac{u}{3\log u}. \tag{2.3}$$

Also let $w(m) = (x/m)^{1/u(m)}$. Then

$$w_1 > w(m) > w_1^{1-3(\log u)/u}. \tag{2.4}$$

Indeed, from (2.2) we have

$$w_1 = (x/m)^{(\log w_1)/\log(x/m)} > w(m)$$

$$\geq (x/m)^{(1+(\log(x/m))/\log w_1)^{-1}}$$

$$> \exp\{\log(x/m)\,\frac{\log w_1}{\log(x/m)}\,(1 - \frac{\log w_1}{\log(x/m)})\}$$

$$= w_1^{1-(\log w_1)/\log(x/m)}$$

$$> w_1^{1-(\log y)/\log(x/m)} \geq w_1^{1-(3\log u)/u}.$$

The product of any set of $u(m)$ distinct primes from $(z,w(m)]$ does not exceed $w(m)^{u(m)} = x/m$. Thus from the first inequality of (2.4) we have

$$\psi(x/m,w_1,z) \geq \psi(x/m,w(m),z) \geq \binom{\pi(w(m))-\pi(z)}{u(m)}.$$

From the second inequality in (2.4) and from the assumption $y \geq (\log x)^{1+\varepsilon}$, we have $w(m)/z \gtrsim \exp((1+\varepsilon)/3)$, so that for large x, we have $w(m)/z > e^{1/3} > 4/3$. Thus for large x, we have

$$\pi(w(m)) - \pi(z) > w(m)/(4\log w(m)),$$

so that

$$\psi(x/m,w_1,z) \geq (\frac{\pi(w(m))-\pi(z)}{u(m)})^{u(m)}$$

$$> (\frac{w(m)}{4u(m)\log w(m)})^{u(m)}$$

$$= \frac{x}{m}\cdot \exp\{-u(m)(\log u(m) + \log\log w(m) + \log 4)\} \tag{2.5}$$

$$> \frac{x}{m}\cdot \exp\{-u(m)(\log u(m) + \log\log y + \log 4)\}$$

$$= \frac{x}{m}\cdot \exp\{-u(m)(\log u - \log\log u + \log\log y + O(1))\}$$

from (2.3). Also, from the assumption $y < \exp((\log x)^{1-\varepsilon})$, we have $\log\log y < \varepsilon^{-1}\log u$. Thus from (2.3) and (2.5) there is a constant C that depends at most on ε such that

$$\psi(x/m,w_1,z) > \frac{x}{m}\cdot \exp(-Cu)$$

for all large x. Putting this estimate into (2.1) we have

$$\psi(x,y,z) > x \cdot e^{-Cu} \sum' m^{-1}, \tag{2.6}$$

so it remains to estimate this last sum. We have (where p represents a prime variable and we use the well-known formula for $\Sigma_{p \le X} p^{-1}$)

$$\begin{aligned}
\sum' m^{-1} &\ge \Big(\sum_{w_1 < p \le w_2} p^{-1} \Big)^{[u]}/[u]! \\
&= (\operatorname{loglog} w_2 - \operatorname{loglog} w_1 + O(e^{-\sqrt{\log y}}))^{[u]}/[u]! \\
&= (\log((1 - \tfrac{1}{3 \log u})/(1 - \tfrac{2}{3 \log u})) + O(e^{-\sqrt{\operatorname{loglog} x}}))^{[u]}/[u]! \\
&> (\tfrac{1}{4 \log u})^{u}/[u]! \\
&= \exp\{-u(\log u + \operatorname{loglog} u + O(1))\}.
\end{aligned}$$

Thus from (2.6),

$$\psi(x,y,z) \ge x \cdot \exp\{-u(\log u + \operatorname{loglog} u + O(1))\},$$

which completes the proof of the theorem.

REMARK. For $z = 1$, Theorem 2.2 reduces to Theorem 2.1 with $y < \exp((\log x)^{1-\varepsilon})$. Thus the above argument is an independent proof of the lower bound in Theorem 2.1 for this restricted set of y.

3. DIXON'S ALGORITHM

We shall only wish to apply factorization algorithms to composite numbers with no small prime factors. So in this and all subsequent sections we shall assume that n is composite and not divisible by any prime $p \le L$. If n is composite, this fact can usually be quickly ascertained with at most a few pseudoprime tests (or one could use the probabilistic compositeness tests of SOLOVAY-STRASSEN [30] or RABIN [25]). To test for all prime factors $p \le L$ by trial divisions takes L steps.

In [9], Dixon describes the following algorithm. Say we wish to find a non-trivial factorization of n. If m is an integer, denote by Q(m) the least

non-negative residue of $m^2 \bmod n$. Let $a < 1$ be a positive constant to be chosen later. We perform the following steps.

1. Randomly choose a number $m \in [1,n]$ and form $Q(m)$.
2. Try to factor $Q(m)$ by trial division with the primes $p \leq L^a$.
3. If $Q(m)$ completely factors in Step (2), store m, $Q(m)$, and its factorization.

These steps are repeated enough times until we have $\pi(L^a) + 1$ completely factored $Q(m)$'s. To each factored $Q(m)$ we associate a vector $v(m) \in (\mathbb{Z}/2\mathbb{Z})^{\pi(L^a)}$, where the i-th coordinate of $v(m)$ is 0 or 1 depending on whether the i-th prime appears an even or an odd number of times in the prime factorization of $Q(m)$. Since we have $\pi(L^a) + 1$ vectors, each with $\pi(L^a)$ coordinates, there is a linear dependency. Perform Gaussian elimination to find such a dependency. Since we are working over $\mathbb{Z}/2\mathbb{Z}$, this dependency may be expressed as

$$v(m_1) + v(m_2) + \ldots + v(m_k) = 0.$$

Thus we may compute a number X such that

$$Q(m_1)Q(m_2) \ldots Q(m_k) = X^2.$$

(Actually, we only compute the least non-negative residue $x \equiv X \bmod n$.) If y denotes the least non-negative residue of $m_1 m_2 \ldots m_k \bmod n$, then we have

$$y^2 \equiv m_1^2 m_2^2 \ldots m_k^2 \equiv Q(m_1)Q(m_2) \ldots Q(m_k) \equiv x^2 \bmod n.$$

We finally compute the greatest common factor of n and $x+y$. In [9], Dixon proves that if n is composite, then with probability at least 1/2, this last step will produce a non-trivial factor of n. Thus, say, if n is composite and the above procedure is repeated 10 times, then we are more than 99.9% certain of finding a non-trivial factorization of n.

Running time analysis

The most time consuming of steps (1), (2), (3) is step (2) which has a running time of L^a. Say we have to repeat steps (1), (2), (3) L^b times before we find $\pi(L^a) + 1$ completely factored $Q(m)$'s. Then the running time for this stage of the algorithm is L^{a+b}. (Note that at this point, we are not sure whether $b = b(n)$ approaches a positive constant or not.) The Gaussian

elimination stage of the algorithm takes L^{3a} steps. The formation of the numbers x, y and (n,x+y) takes only L^{a} steps. Thus the total running time is

$$L^{\max\{a+b,3a\}}, \tag{3.1}$$

so it remains only to find b and choose a optimally. To make a long story short, it turns out that we may choose $b = a + (2a)^{-1}$. Thus the optimal choice for a is 1/2.

It is at least heuristically clear that we should choose $b = a + (2a)^{-1}$. Indeed, one might expect that the probability that Q(m) completely factors with the primes $p \leq L^{a}$ is about $n^{-1}\psi(n,L^{a})$. Moreover, by Theorem 2.1 this quantity is $L^{-(2a)^{-1}}$. Thus to obtain L^{a} completely factored Q(m)'s, one should get $L^{a+(2a)^{-1}}$ values of m. The nice thing about Dixon's algorithm is that we are able to prove this heuristic guess.

Because of a variation of Dixon's algorithm studied in the next section we treat now a situation more general than is immediately required. In particular, to prove Theorem 3.2 below, only Theorem 2.1 is required, but instead we shall use the more general Theorem 2.2.

We introduce some notation. If T is a finite set, by $|T|$ we shall mean the cardinality of T. Let T(x,y,z) denote the set of integers counted by $\psi(x,y,z)$. That is,

$$T(x,y,z) = \{m \leq x: p|m, p \text{ prime} \Rightarrow z < p \leq y\}.$$

Thus $|T(x,y,z)| = \psi(x,y,z)$. By $T_Q(x,y,z)$ we shall mean the set of residues mod n whose squares are in T(x,y,z) and coprime to n. In symbols,

$$T_Q(x,y,z) = \{m \in [1,n]: (m,n) = 1, Q(m) \in T(x,y,z)\}.$$

Our goal is to show that in certain circumstances, $|T_Q(x,y,z)|$ and $\psi(x,y,z)$ agree up to a factor of $L^{o(1)}$. Our proof begins along the lines of Dixon's Lemma 2. What is new here is Lemma 3.2 below which depends on Theorems 2.1 and 2.2. In its place, Dixon uses a trivial estimate. By taking the middle ground of using a crude lower bound estimate for $\psi(x,y)$ in the proof of Lemma 3.2 one could obtain the upper bound estimate for the running time of Dixon's algorithm achieved by SCHNORR [26] and KNUTH [13].

LEMMA 3.1. *Suppose* $2 \le y < x \le n$ *and that all prime factors of* n *exceed* y. *Then*

$$2^{\omega(n)}\psi(x,y,z) \ge |T_Q(x,y,z)| \ge \psi(\sqrt{x},y,z)^4\Big(\sum_{T(\sqrt{x},y,z)} \tau(t)\Big)^{-2},$$

where $\omega(n)$ *denotes the number of distinct prime factors of* n *and* $\tau(t)$ *denotes the number of positive divisors of* t.

PROOF. Let Q denote the group of squares in $(\mathbb{Z}/n\mathbb{Z})^*$. Then

$$(\mathbb{Z}/n\mathbb{Z})^*/Q \simeq (\mathbb{Z}/2\mathbb{Z})^{\omega(n)}.$$

Identifying $\mathbb{Z}/2\mathbb{Z}$ with the multiplicative group $\{1,-1\}$, we have the canonical projection

$$\chi\colon (\mathbb{Z}/n\mathbb{Z})^* \to (\mathbb{Z}/2\mathbb{Z})^{\omega(n)}$$

where $\chi(t) = ((t/p_1),\ldots,(t/p_{\omega(n)}))$; $p_1,\ldots,p_{\omega(n)}$ being the distinct prime factors of n and (t/p_i) the Legendre symbol. Thus the function $\chi(t)$ describes $2^{\omega(n)}$ quadratic residue classes mod n.

If $t \in Q$, then by the above, $t = Q(m)$ for precisely $2^{\omega(n)}$ values of $m \in [1,n]$. Thus we have

$$|T_Q(x,y,z)| = 2^{\omega(n)}|Q \cap T(x,y,z)| \le 2^{\omega(n)}\psi(x,y,z), \tag{3.2}$$

which establishes the first inequality in the lemma.

Label the $2^{\omega(n)}$ quadratic residue classes of $(\mathbb{Z}/n\mathbb{Z})^*$ with the integers $i = 1,\ldots,2^{\omega(n)}$. Let T_i denote the set of $t \in T(\sqrt{x},y,z)$ that belong to the i-th class. Let $T = \cup T_i$, so that $T = T(\sqrt{x},y,z)$. Let

$$S = \sum_T \tau(t)^{-1}, \quad S_i = \sum_{T_i} \tau(t)^{-1}, \quad S' = \sum_T \tau(t).$$

By the Cauchy-Schwarz inequality, we have

$$\psi(\sqrt{x},y,z)^2 \le SS', \tag{3.3}$$

$$S^2 \le 2^{\omega(n)} \sum_{i=1}^{2^{\omega(n)}} S_i^2. \tag{3.4}$$

If $s = tt'$ for $t,t' \in T_i$ for some i, then by the multiplicativity of Legendre symbols we have $s \in Q \cap T(x,y,z)$. Let $c(s)$ denote $\Sigma\, \tau(t)^{-1}\tau(t')^{-1}$ where the sum is over all ordered pairs t, t' where $s = tt'$ and $t,t' \in T_i$ for some i. Then $c(s) > 0$ implies $s \in Q \cap T(x,y,z)$. Moreover, since $\tau(t)^{-1}\tau(t')^{-1} \le \tau(tt')^{-1}$, we have $c(s) \le 1$ for all s. Thus from (3.2), (3.3), (3.4) we have

$$\begin{aligned} |T_Q(x,y,z)| &= 2^{\omega(n)}\,|Q \cap T(x,y,z)| \\ &\ge 2^{\omega(n)} \sum_{s=1}^{n} c(s) = 2^{\omega(n)} \sum_{i=1}^{2^{\omega(n)}} S_i^2 \\ &\ge S^2 \ge \psi(\sqrt{x},y,z)^4 (S')^{-2}, \end{aligned}$$

which completes the proof of the lemma.

We shall be interested in $|T_Q(x,y,z)|$ in three situations:

(i) $x = n^c$, $y = L^\alpha$, $z = 1$,

(ii) $x = n^c$, $y = L^\alpha$, $z = L^\beta$,

(iii) $x = n^c$, $y = n^c$, $z = L^\beta$,

where $0 < c \le 1$, $0 < \beta < \alpha < 1$. In addition, recall that we are assuming n is composed solely of primes $p \ge L$. Thus the inequality $L(n)^{\omega(n)} \le n$ implies $\omega(n) \le \sqrt{(\log n)/\log\log n}$, so that

$$2^{\omega(n)} = L^{o(1)}. \tag{3.5}$$

Thus from Theorem 2.1,

$$2^{\omega(n)}\psi(n^c,L^\alpha,1) = n^c L^{-c(2\alpha)^{-1}} \tag{3.6}$$

and from Theorem 2.2,

$$2^{\omega(n)}\psi(n^c,L^\alpha,L^\beta) = n^c L^{-c(2\alpha)^{-1}}. \tag{3.7}$$

In the following lemma we consider situations (i) and (ii) at once.

LEMMA 3.2. *For* $0 < c \le 1$, $0 \le \beta < \alpha$, *we have*

$$S' \overset{\text{def}}{=\!=} \sum_{T(n^{c/2},L^\alpha,L^\beta)} \tau(t) \le n^{c/2} L(n)^{-c(4\alpha)^{-1}+o(1)}.$$

PROOF. Let $T = T(n^{c/2},L^{\alpha},L^{\beta})$, let N denote a large, temporarily fixed integer, and let $\varepsilon = c/(2N)$. Then

$$S' = \sum_{T} \tau(t) = \sum_{t\in T} \sum_{d|t} 1 = \sum_{d\in T} \sum_{\substack{d'\in T \\ d'\le n^{c/2}/d}} 1$$

$$= \sum_{d\in T} \psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$\le \sum_{i=1}^{N} \sum_{\substack{d\in T \\ n^{(i-1)\varepsilon}\le d\le n^{i\varepsilon}}} \psi(n^{c/2}/d,L^{\alpha},L^{\beta}) = \sum_{i=1}^{N} S_i' ,$$

say. We note that

$$S_1' \le \sum_{d\le n^{\varepsilon}} \psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$= \sum_{d\le n^{\varepsilon}} (n^{c/2}/d)(n^{c/2}/d)^{-1}\psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$\le \sum_{d\le n^{\varepsilon}} (n^{c/2}/d)L^{-(c/2-\varepsilon)(2\alpha)^{-1}}$$

$$= n^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}},$$

by Theorem 2.2. Also

$$S_N' = \sum_{\substack{d\in T \\ n^{c/2-\varepsilon}\le d}} \psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$\le \sum_{\substack{d\in T \\ n^{c/2-\varepsilon}\le d}} n^{c/2}/d = n^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}},$$

by Theorem 2.2 and partial summation. Finally, if $1 < i < N$, we have by the same techniques

$$S_i' = \sum_{\substack{d\in T \\ n^{(i-1)\varepsilon}\le d\le n^{i\varepsilon}}} (n^{c/2}/d)\,(n^{c/2}/d)^{-1}\psi(n^{c/2}/d, L^{\alpha}, L^{\beta})$$

$$\le \sum_{\substack{d\in T \\ n^{(i-1)\varepsilon}\le d}} (n^{c/2}/d)L^{-(c/2-i\varepsilon)(2\alpha)^{-1}}$$

$$= n^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}}.$$

Thus putting together our estimates we have

$$S' \le \sum_{i=1}^{N} S_i' \le Nn^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}}.$$

Since this result holds for each integer N, we have

$$S' \le n^{c/2}L^{-c(4\alpha)^{-1}},$$

which proves our lemma.

We can now prove

THEOREM 3.1. *If* $0 < c \le 1$, $0 < \beta < \alpha < 1$, *and* n *is composed solely of primes exceeding* L(n), *then*

$$|T_Q(n^c, L(n)^{\alpha}, 1)| = n^c L(n)^{-c(2\alpha)^{-1}+o(1)}, \tag{3.8}$$

$$|T_Q(n^c, L(n)^{\alpha}, L(n)^{\beta})| = n^c L(n)^{-c(2\alpha)^{-1}+o(1)}, \tag{3.9}$$

$$|T_Q(n^c, n^c, L(n)^{\beta})| = n^c L(n)^{o(1)}. \tag{3.10}$$

PROOF. Note that Lemma 3.1 and (3.6), (3.7) immediately give the upper bounds in (3.8), (3.9). Allowing the possibility $\beta = 0$, we have

$$\psi(n^{c/2}, L^{\alpha}, L^{\beta}) = n^{c/2}L^{-c(4\alpha)^{-1}}$$

from Theorems 2.1 and 2.2. Thus Lemmas 3.1 and 3.2 give the lower bounds in (3.8), (3.9).

For (3.10), let T_i' denote the set of prime numbers in the interval $(L^{\beta},n^{c/2}]$ which do not divide n and belong to the i-th quadratic residue class. Thus

$$\sum_{i=1}^{2^{\omega(n)}} |T_i'| \geq \pi(n^{c/2}) - \pi(L^{\beta}) - \omega(n) \sim \frac{n^{c/2}}{\log n^{c/2}},$$

so that from (3.5) there is some i with $|T_i'| \geq n^{c/2}L^{o(1)}$. But if $p_1,p_2 \in T_i'$, then $p_1p_2 \in Q \cap T(n^c,n^c,L^{\beta})$. Such a product p_1p_2 can occur in this fashion in at most one other way, namely as p_2p_1. Thus

$$|T_Q(n^c,n^c,L^{\beta})| \geq |Q \cap T(n^c,n^c,L^{\beta})| \geq \frac{1}{2}|T_i'|^2 \geq n^cL^{o(1)}.$$

This inequality immediately gives the equality (3.10).

Recall that in the implementation of Dixon's algorithm we repeat steps (1), (2), (3) L^b times, where b is chosen so that we are virtually assured of obtaining L^a values of m for which Q(m) completely factors with the primes $p \leq L^a$. If $(m,n) > 1$, then Q(m) will not completely factor with the primes $p \leq L^a$, since $a < 1$ and n is not divisible by any prime $p \leq L$. However, very few of our randomly chosen m will satisfy $(m,n) > 1$ since it is possible to show $\phi(n)/n \geq 1 - 1/L$ where ϕ denotes Euler's function. Thus we have L^b values of m with $(m,n) = 1$.

By (3.8)

$$|T_Q(n,L^a,1)| = nL^{(-2a)^{-1}}.$$

Thus given L^b random values of $m \in [1,n]$ with $(m,n) = 1$, we expect $L^{b-(2a)^{-1}}$ of these m to fall in $T_Q(n,L^a,1)$. Thus we choose $b = a + (2a)^{-1}$. By (3.1) the running time for all of Dixon's algorithm is

$$L^{\max\{2a+(2a)^{-1},3a\}}.$$

By choosing $a = 1/2$, we thus have

THEOREM 3.2. *Dixon's algorithm has expected running time* $L(n)^{2+o(1)}$. *The space required is* $L(n)^{1+o(1)}$.

PROOF. We have already shown the assertion about the running time. To see the assertion about the space, note that the most space-consuming part of the algorithm is the construction of the matrix upon which we perform the

Gaussian elimination. This matrix is $L^a \times L^a$ where we have chosen a = 1/2.

REMARKS. Dixon claims $L^{3\sqrt{2}}$ as an upper bound for the running time for his algorithm, but we have introduced some efficiencies into his proof. Both SCHNORR [26] and KNUTH [13] had already sharpened Dixon's proof to give a $L^{2\sqrt{2}}$ upper bound for the running time. As we have already remarked, the difference between their analysis and ours is that they used an elementary estimate for $\psi(x,y)$ rather than Theorem 2.1. Note that Theorem 3.2 is sharp - the expected running time is bounded above and below by L^2. (All of the running time estimates in this paper are sharp.)

4. VARIATIONS OF DIXON'S ALGORITHM

Already in MORRISON-BRILLHART [21], several variations of the continued fraction algorithm were proposed as a possible means of speeding things up. In this section we analyze three natural variations in the context of Dixon's algorithm. All three make sense for the continued fraction algorithm, but we shall only be able to give heuristic arguments there. However, with Dixon's algorithm we are able to give rigorous proofs. With our most elaborate combination of variations we are able to lower the exponent 2 in Theorem 3.2 to $\sqrt{5/2} \approx 1.581$.

Large Prime

This variation was suggested in MORRISON-BRILLHART [21] and is commonly used by people implementing the continued fraction algorithm (for example, see WUNDERLICH [36]). We now describe and analyze the Dixon algorithm analogue.

If the unfactored portion x of Q(m) after trial division up to y is less than y^2, then x is prime. Thus if Q(m) completely factors with the primes up to L^a, we usually find this out earlier before all the trial division steps have been executed. However, the majority of Q(m) do not completely factor with the primes up to L^a and for these Q(m) we must spend the full time (but see the "early abort strategy" below). Thus this idea cannot lower the asymptotic running time.

But say $x \in (L^a, L^{2a}]$. That is, after trial division up to L^a, the unfactored portion x is not 1, but $x \leq L^{2a}$. Thus Q(m) has a single prime factor in $(L^a, L^{2a}]$ and all other prime factors are in $(1, L^a]$. Since such Q(m)

are discovered with no extra work, why not use them?

At first glance it would seem that using these large primes makes the Gaussian elimination step more complex - instead of working with an $L^a \times L^a$ matrix, we have perhaps an $L^{2a} \times L^{2a}$ matrix. However, by the following device, called a "cull", there is very little extra cost at the elimination stage. First, the factored Q(m)'s are placed in order of their largest prime. We then pass through the list of corresponding 0 - 1 vectors v(m) exactly once. If v(m)'s last 1 is not already shared by a neighbouring v(m), this vector is completely deleted. If it is shared by a neighbour, the two vectors are added (mod 2), thus eliminating the large prime. (Of course, if there are k vectors with the same large prime, we add the first to each of the k - 1 other vectors.) After the cull we are left with 0 - 1 vectors where the coordinates corresponding to primes in $(L^a, L^{2a}]$ are all 0. We thus have reduced our large matrix to a matrix with only L^a columns. What makes this idea work is that each Q(m) has at most one prime in $(L^a, L^{2a}]$. If a Q(m) had more primes in this interval, the process would probably not be accomplished in one pass down the matrix.

To keep the space requirement and running time down, the cull procedure can be performed on the factored Q(m)'s themselves rather than with the vectors v(m). Since any Q(m) has at most $\log n/\log 2$ prime factors, even if we had as many as L^{2a} values of Q(m) involved in the cull, the space and running time would both be at most $cL^{2a}(\log n)^2 = L^{2a}$.

So we see we can cheaply integrate into our algorithm those Q(m) with a single prime in $(L^a, L^{2a}]$ and all other primes in $(1, L^a]$. The question is, how many new Q(m) are now included? We now show that unfortunately there are not many more.

THEOREM 4.1. *Assume that* n *is divisible by no prime up to* L(n). *Let* $0 < a < 1$ *and let* M *denote the number of* $m \in [1,n]$ *such that* Q(m) *is divisible by at most one prime, counting multiplicity, in the interval* $(L(n)^a, L(n)^{2a}]$ *and that all other prime factors of* Q(m) *are in* $(1, L(n)^a]$. *Then*
$M = nL(n)^{-(2a)^{-1}+o(1)}$.

PROOF. We clearly have $M \geq |T_Q(n, L^a, 1)|$, where the notation is defined in §3. Thus by Theorem 3.1, $M \geq n \cdot L^{-(2a)^{-1}}$. On the other hand, if $t \leq n$ is divisible by at most one prime from $(L^a, L^{2a}]$ and all other primes of t are in $(1, L^a]$, then t = Q(m) for at most $2^{\omega(n)}$ values of $m \in [1,n]$. To see this, one uses the argument at the beginning of the proof of Lemma 3.1, making separate cases for (m,n) = 1, (m,n) > 1. Thus, if p denotes a prime variable,

$$M \le 2^{\omega(n)} \sum_{L^a < p \le L^{2a}} \psi(n/p, L^a)$$

$$= 2^{\omega(n)} nL^{-(2a)^{-1}} \sum_{L^a < p \le L^{2a}} 1/p$$

$$= 2^{\omega(n)} nL^{-(2a)^{-1}} = nL^{-(2a)^{-1}},$$

where we use Theorem 2.1 and (3.5).

We remark that the result would be the same even if we allowed $Q(m)$ to have k primes in $(L^a, L^{ca}]$ for any fixed k and c. Following the proof of Theorem 3.2 but now using Theorem 4.1 rather than Theorem 3.1 gives us

THEOREM 4.2. *The expected running time for Dixon's algorithm with the Large Prime variation is* $L(n)^{2+o(1)}$. *The space required is* $L(n)^{1+o(1)}$.

We are not saying that the Large Prime variation is useless. To the contrary, those actually using the method (in the context of the continued fraction algorithm) find it very helpful. What we are saying is that the idea only affects the "o(1)" in the running time. In contrast, the remaining variations we shall discuss actually lower the exponent somewhat.

The Pollard-Strassen method

The naive way to see if $Q(m)$ completely factors with the primes up to L^a is to use trial division. This is in fact the method used in Section 3. By this method it takes L^a steps to ascertain whether $Q(m)$ so factors by this method. In [26], SCHNORR suggests using a factorization method of Pollard in place of trial division. In fact, POLLARD has two methods [22], [23] which can find the largest divisor of $Q(m)$ consisting solely of primes $p \le L^a$ in time $L^{a/2}$. One method, known as the Pollard-ρ method (so-called because a certain diagram in the description of the algorithm is in the shape of a "ρ"), is probabilistic and might cause some problems in a rigorous proof of the running time of Dixon's algorithm. Thus for our theoretical analysis it is preferable to use the second method which is deterministic and fully proved. This method is based on the fast Fourier transform. We present a simplified version due to STRASSEN [31], also employed by Schnorr.

Let y be a square and let $z = \sqrt{y}$. We shall show that for every integer

t, the least prime factor of (t,y!) can be found in $O(z(\log^2 z + \text{loglog } t)$ log t loglog t logloglog t) bit operations. Since Q(m) has at most O(log n) prime factors, by an iteration of this method we can find the largest divisor of Q(m) composed solely of primes $p \le L^a$ in time $L^{a/2}$.

The idea is to write

$$y! = \prod_{j=1}^{z} (jz)!/[(j-1)z]!$$

and to compute each (t,(jz)!/[(j-1)z]!); the first one of these which is larger than 1 isolates the least prime factor of (t,y!) in an interval of length z in which it can be quickly found. Consider the polynomial

$$f(x) = \prod_{i=0}^{z-1} (x-i).$$

Then f(jz) = (jz)!/[(j-1)z]!. Moreover by [32, Sec. 4] all of the f(jz) mod t, j = 1,...,z, can be found in a total of $O(z \log^2 z \log t \text{ loglog } t \text{ logloglog } t)$ bit operations (cf. [3, Sec. 4.5]). Since each

$$(t, f(jz) \bmod t) = (t, (jz)!/[(j-1)z]!)$$

can be found in $O(\log t (\text{loglog } t)^2 \text{ logloglog } t)$ bit operations, by [13, exercise 4.5.2.32], they all can be computed in $O(z \log t (\text{loglog } t)^2 \text{ logloglog } t)$ bit operations. Finally, when the first of these which exceeds 1 is found, it takes O(z) additional trial divisions of O(log t loglog t logloglog t) bit operations each to find the least prime in (t,f(jz) mod t). Thus there are a total of $O(z(\log^2 z + \log\log t)\log t \text{ loglog } t \text{ logloglog } t)$ bit operations, as claimed.

To ascertain the effect of using the Pollard-Strassen method in place of trial division in Dixon's algorithm, recall that if we use the primes up to L^a, then we expect to get $L^{a+(2a)^{-1}}$ values of Q(m). Previously, for each (or almost all) Q(m), we took L^a steps to see if Q(m) completely factors with the primes $p \le L^a$. With the Pollard-Strassen method, it only takes $L^{a/2}$ steps. Thus the total running time is

$$L^{\max\{3a/2+(2a)^{-1}, 3a\}}.$$

Choosing $a = \sqrt{1/3}$, we have

THEOREM 4.3. *The expected running time of Dixon's algorithm with the Pollard-Strassen method is* $L(n)^{\sqrt{3}+o(1)}$. *The space required is* $L(n)^{\sqrt{4/3}+o(1)}$.

REMARK. In [26], SCHNORR obtains an upper bound of $L^{\sqrt{6}}$ for the running time of Dixon's algorithm with the Pollard-Strassen method.

The early abort strategy

For the overwhelming majority of the m chosen in Dixon's algorithm one must spend the full allotment of time, L^a for trial division, $L^{a/2}$ for the Pollard-Strassen method, only to find that Q(m) cannot be used. If there were some way to weed out most of these bad Q(m) before too much time was invested, we could speed things up. A natural idea that many have had is to abort working with Q(m) if at some pre-chosen point it does not look too likely to be composed solely of the primes below L^a. We now make this idea precise and show that we can indeed lower the exponent in the running time.

We first consider the situation where Dixon's algorithm is altered with just one go or no go decision on a Q(m). We then generalize to k decisions for a fixed but arbitrary k. Finally, we combine the early abort strategy with the Pollard-Strassen method.

If $0 < a,c,\theta < 1$, then by Dixon's algorithm with one early abort and parameters a,c,θ we mean the following variation of Dixon's algorithm. As before we try to factor the Q(m) with the primes up to L^a. However, now after trial division to $L^{\theta a}$, if the unfactored portion of Q(m) exceeds n^{1-c}, we abort the procedure with Q(m) and get the next m.

We now ask for the probability that Q(m) will be aborted at $L^{\theta a}$ and also the probability that Q(m) will not get aborted and eventually completely factor.

THEOREM 4.4. *Let* $0 < a,c,\theta < 1$ *and assume* n *is divisible by no prime* $p \le L(n)$. *Let* M_1 *denote the number of* $m \in [1,n]$ *such that* Q(m) *has a divisor* m_1 *composed solely of primes* $p \le L(n)^{\theta a}$ *and* $Q(m)/m_1 \le n^{1-c}$. *Let* M_2 *denote the number of* m *with the same property and also* Q(m) *is composed solely of primes* $p \le L(n)^a$. *Then*

$$M_1 = n \cdot L(n)^{-c(2\theta a)^{-1}+o(1)},$$

$$M_2 = n \cdot L(n)^{-c(2\theta a)^{-1}-(1-c)(2a)^{-1}+o(1)}.$$

PROOF. Using the notation from Section 3, we evidently have

$$M_1 \geq |T_Q(n^c, L^{\theta a}, 1)||T_Q(n^{1-c}, n^{1-c}, L^{\theta a})|,$$

$$M_2 \geq |T_Q(n^c, L^{\theta a}, 1)||T_Q(n^{1-c}, L^a, L^{\theta a})|.$$

Thus from Theorem 3.1 we have the lower bounds in the theorem.

We shall find the upper bounds a bit more tedious. Our first task is to estimate for a given u the number of $m \in [1,n]$ with $Q(m) = u$. If $(u,n) = 1$, we have already seen in Section 3 that this number is either 0 or $2^{\omega(n)}$. However, if $(u,n) > 1$, there is some trouble. Suppose $q > 2$ is prime, $k > 0$, $k \geq b \geq 0$ are integers and $q^b \| u$. Then it is possible to show that

$$\sum_{\substack{m\in[1,q^k] \\ m^2\equiv u(q^k)}} 1 = \begin{cases} 2q^{b/2}, & \text{if } \left(\frac{uq^{-b}}{q}\right) = 1, \ b \text{ even}, \ b < k \\ 0, & \text{if } \left(\frac{uq^{-b}}{q}\right) = -1 \\ 0, & \text{if } b \text{ odd}, \ b < k \\ q^{[b/2]}, & \text{if } b = k. \end{cases}$$

As a consequence we have

$$(4.1) \qquad \sum_{\substack{m\in[1,n] \\ Q(m)=u}} 1 = \prod_{q^k \| n} \sum_{\substack{m\in[1,q^k] \\ m^2\equiv u(q^k)}} 1 \leq \prod_{q^k \| n} 2\sqrt{(u,q^k)} = 2^{\omega(n)}\sqrt{(u,n)}.$$

To estimate M_1 from above, note that

$$M_1 = \sum_t \sum_s \sum_{\substack{m\in[1,n] \\ Q(m)=ts}} 1$$

where t runs over $T(n^{1-c}, n^{1-c}, L^{\theta a})$ and s runs over $T(n/t, L^{\theta a}, 1)$. Furthermore, since all of n's prime factors exceed L, we have $(ts,n) = (t,n)$. Thus from (4.1) we have

$$M_1 \le 2^{\omega(n)} \sum_t \sum_s \sqrt{(t,n)} \le 2^{\omega(n)} \sum_{d|n} \sum_{\substack{d|t \\ t\le n^{1-c}}} \sum_s \sqrt{d}$$

$$= 2^{\omega(n)} \sum_{d|n} \sqrt{d} \sum_{\substack{d|t \\ t\le n^{1-c}}} \psi(n/t, L^{\theta a})$$

$$\le 2^{\omega(n)} \sum_{d|n} \sqrt{d} \sum_{\substack{d|t \\ t\le n^{1-c}}} (n/t) L^{-c(2\theta a)^{-1}}$$

$$\le 2^{\omega(n)} \sum_{d|n} \sqrt{d}\ (n/d)(\log n) L^{-c(2\theta a)^{-1}}$$

$$= nL^{-c(2\theta a)^{-1}} \sum_{d|n} 1/\sqrt{d}$$

$$\le nL^{-c(2\theta a)^{-1}} \prod_{q|n} \left(1 + \frac{1}{\sqrt{q}-1}\right)$$

$$\le nL^{-c(2\theta a)^{-1}} (1 + L^{-1/2})^{\omega(n)}$$

$$= nL^{-c(2\theta a)^{-1}},$$

where we have used Theorem 2.1 and (3.5), and where q runs over the prime factors of n (which all exceed L).

Similarly we have

$$M_2 = \sum_t \sum_s \sum_{\substack{m\in[1,n] \\ Q(m)=ts}} 1$$

where t now runs over $T(n^{1-c}, L^a, L^{\theta a})$ and s runs over $T(n/t, L^{\theta a}, 1)$ as before. Now we have $(ts,n) = 1$, so that

$$(4.2) \qquad M_2 \le 2^{\omega(n)} \sum_t \sum_s 1 = 2^{\omega(n)} \sum_t \psi(n/t, L^{\theta a}).$$

We shall find it convenient to break up this last sum. Let N be a large temporarily fixed integer and let $\varepsilon = (1-c)/N$. Let

$$T(c) = \sum_{t \in T(n^{1-c}, L^a, L^{\theta a})} \psi(n/t, L^{\theta a}).$$

Then from (4.2),

$$M_2 \le 2^{\omega(n)} T(c) = 2^{\omega(n)} + 2^{\omega(n)} \sum_{i=0}^{N-1} \{T(c+i\varepsilon) - T(c+(i+1)\varepsilon)\}$$

$$\le 2^{\omega(n)} + 2^{\omega(n)} n \sum_{i=0}^{N-1} L^{-(c+i\varepsilon)(2\theta a)^{-1}} \sum_{t > n^{1-c-(i+1)\varepsilon}}^{} 1/t$$

$$\le 2^{\omega(n)} + 2^{\omega(n)} n \sum_{i=0}^{N-1} L^{-(c+i\varepsilon)(2\theta a)^{-1}} \frac{\sum_{t \in T(n^{1-c-i\varepsilon}, L^a, 1)} 1}{(1-c-(i+1)\varepsilon)}$$

where we use Theorem 2.1. Since the exponent of the summand is least negative when $i = 0$, we have from (3.5),

$$M_2 \le nL^{-c(2\theta a)^{-1} - (1-c-\varepsilon)(2a)^{-1}}.$$

Finally letting $N \to \infty$, so that $\varepsilon \to 0$, we have our upper bound for M_2.

THEOREM 4.5. *Dixon's algorithm with one early abort and parameters* a, c, θ *has expected running time* $L(n)^{\gamma+o(1)}$ *where*

$$\gamma = \max\{a + \theta a + c(2\theta a)^{-1} + (1-c)(2a)^{-1},\ 2a + (1-c)(2a)^{-1},\ 3a\}.$$

PROOF. We shall need, as before, L^a choices of m for which Q(m) is composed solely of the primes $p \le L^a$. Every such Q(m) obtained in the algorithm has the additional property that there is a divisor m_1 of Q(m) composed solely of the primes $p \le L^{\theta a}$ and such that $Q(m)/m_1 \le n^{1-c}$. By Theorem 4.4, we thus should expect to use $L^{a+c(2\theta a)^{-1} + (1-c)(2a)^{-1}}$ values of m before L^a good values are found. For each m chosen we must at least do trial division to $L^{\theta a}$ which takes $L^{\theta a}$ steps. Thus the running time for trial division to $L^{\theta a}$ is

$$L^{a+\theta a+c(2\theta a)^{-1}+(1-c)(2a)^{-1}}. \tag{4.3}$$

Also by Theorem 4.4, the expected number of m for which we do not abort at $L^{\theta a}$ and continue trial division to L^a is $L^{a+(1-c)(2a)^{-1}}$. The time for each

such m is L^a, so the total time for trial division to L^a is

$$(4.4) \qquad L^{2a+(1-c)(2a)^{-1}}.$$

Finally the Gaussian elimination step has a running time of L^{3a}. This completes the proof of the theorem.

Thus our remaining problem is to choose the parameters a, c, θ optimally. It is not too difficult to see that the minimum expected running time is attained when the exponents in (4.3) and (4.4) are equalized. The optimal choice of parameters turns out to be

$$a = \sqrt{2/7}, \quad c = 1/7, \quad \theta = 1/2.$$

We call Dixon's algorithm with one early abort and this set of parameters simply Dixon's algorithm with one early abort. We have

THEOREM 4.6. *Dixon's algorithm with one early abort has expected running time* $L(n)^{\sqrt{7/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{8/7}+o(1)}$.

We now consider the situation for more than one early abort. Suppose

$$(4.5) \qquad 0 < a, c_i, \theta_i < 1; \quad c_1+\ldots+c_k < 1; \quad \theta_1 < \ldots < \theta_k.$$

By Dixon's algorithm with k early aborts and parameters $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$ we mean the following. As before, we try to factor each Q(m) with the primes $p \le L^a$. However, if after trial division to $L^{\theta_i a}$ the unfactored portion of Q(m) exceeds $n^{1-c_1-\ldots-c_i}$, then we abort Q(m) and get the next m. Thus for trial division to be completed to L^a, our number Q(m) must successfully pass k tests.

In an analogous fashion to Theorem 4.4 we can prove

THEOREM 4.7. *Say the procedure is to follow Dixon's algorithm with* k *early aborts and parameters* $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$. *For* $i = 1, \ldots, k$, *let* M_i *denote the number of* $m \in [1,n]$ *for which* Q(m) *is not aborted at* $L(n)^{\theta_1 a}, \ldots, L(n)^{\theta_i a}$. *Also let* M_{k+1} *denote the number of* $m \in [1,n]$ *for which not only is* Q(m) *not aborted, but* Q(m) *eventually factors completely with the primes up to* $L(n)^a$. *Then*

$$M_i = n \cdot L(n)^{-c_1(2\theta_1 a)^{-1} - \ldots - c_i(2\theta_i a)^{-1} + o(1)} \quad \textit{for } i = 1,\ldots,k,$$

$$M_{k+1} = n \cdot L(n)^{-c_1(2\theta_1 a)^{-1} - \ldots - c_k(2\theta_k a)^{-1} - (1-c_1-\ldots-c_k)(2a)^{-1} + o(1)}.$$

We remark that, as with Theorem 4.4, the proof of the lower bounds is relatively simple and follows easily from Theorem 3.1. The proof of the upper bounds is tedious but essentially relies on nothing more than Theorem 2.1.

For $i = 1,\ldots,k$ let

$$f_i = a + \theta_i a + c_i(2\theta_i a)^{-1} + \ldots + c_k(2\theta_k a)^{-1} + (1 - c_1 - \ldots - c_k)(2a)^{-1}$$

and let

$$f_{k+1} = 2a + (1 - c_1 - \ldots - c_k)(2a)^{-1}.$$

In the same fashion as Theorem 4.5 follows from Theorem 4.4 we have from Theorem 4.7,

THEOREM 4.8. *The expected running time for Dixon's algorithm with* k *early aborts and parameters* $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$ *is*

$$L(n)^{\max\{f_1,\ldots,f_{k+1},3a\}+o(1)}.$$

So we now try to choose the parameters $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$ optimally. This task, which is a bit messy, can be done by induction on k. Again, as with $k = 1$, the optimal choice occurs when the functions $f_1, \ldots, f_{k+1}$ are equalized. The optimal choice is

$$a = (3 + \frac{1}{k+1})^{-1/2}, \quad c_i = \frac{ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

We call Dixon's algorithm with k early aborts and this set of parameters simply Dixon's algorithm with k early aborts.

THEOREM 4.9. *The expected running time for Dixon's algorithm with* k *early aborts is* $L(n)^{\sqrt{3+1/(k+1)}+o(1)}$. *The space required is* $L(n)^{2/\sqrt{3+1/(k+1)}+o(1)}$.

If we now allow k to slowly tend to ∞ as $n \to \infty$, we call the resulting algorithm Dixon's algorithm with the early abort strategy.

THEOREM 4.10. *Dixon's algorithm with the early abort strategy has expected running time* $L(n)^{\sqrt{3}+o(1)}$. *The space required is* $L(n)^{\sqrt{4/3}+o(1)}$.

We thus find that the early abort strategy has the same effect on the running time of Dixon's algorithm as does the use of the Pollard-Strassen method to replace trial division. We now consider a combination of the two variations. That is, we use the Pollard-Strassen method instead of trial division and we use k optimally chosen early aborts. The running time analysis for this algorithm is very similar to the above. We simply replace the functions $f_1,\ldots,f_{k+1}$ with $g_1,\ldots,g_{k+1}$ where

$$g_i = f_i - \theta_i a/2, \quad i = 1,\ldots,k,$$

$$g_{k+1} = f_{k+1} - a/2.$$

However, if we use Gaussian elimination, which takes L^{3a} steps, we find that with an otherwise optimal choice of parameters, this step will dominate the running time. But there are asymptotically faster methods for finding a linear dependency than Gaussian elimination. For example, there is a recent algorithm of COPPERSMITH and WINOGRAD [8] which can find a linear dependency among a set of vectors in $(\mathbb{Z}/2\mathbb{Z})^K$ in time bounded by $O(K^r)$ where

$$r < 2.495548.$$

In general, we shall say that an algorithm to find a linear dependency among a set of vectors in $(\mathbb{Z}/2\mathbb{Z})^K$ has exponent r, if the running time for the algorithm is $K^{r+o(1)}$. Thus Gaussian elimination has exponent 3 since it runs in time $O(K^3)$ and not generally in time $O(K^{3-\varepsilon})$ for any $\varepsilon > 0$. It is evident that no elimination method has exponent $r < 2$. We shall say more about the problem of elimination in Section 8.

We now return to Dixon's algorithm with k early aborts and with the Pollard-Strassen method. If we use an elimination method with exponent $r > 5/2 + 1/(2k+2)$, the optimal choice of parameters is

$$a = (2r - 3 + \frac{k}{2k+2})^{-1/2}, \quad c_i = \frac{ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

If we use an elimination method with exponent $r \leq 5/2 + 1/(2k+2)$, the optimal choice of parameters is

$$a = (3 - \frac{k}{2k+2})^{-1/2}, \quad c_i = \frac{ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

We have

THEOREM 4.11. *Using an elimination method with exponent* r, *the Pollard-Strassen method, and* k *optimally chosen early aborts, Dixon's algorithm has expected running time*

$$L(n)^{r(2r-3+k/(2k+2))^{-1/2}+o(1)}, \quad \text{if } r > 5/2 + 1/(2k+2),$$

$$L(n)^{(3-k/(2k+2))^{1/2}+o(1)}, \quad \text{if } r \le 5/2 + 1/(2k+2).$$

Finally, letting k tend to ∞ slowly with n, we have

THEOREM 4.12. *The expected running time for Dixon's algorithm with the Pollard-Strassen method, the early abort strategy, and an elimination method with exponent* r *is*

$$L(n)^{r(2r-5/2)^{-1/2}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{5/2}+o(1)}, \quad \text{if } r \le 5/2.$$

The space required is

$$L(n)^{2(2r-5/2)^{-1/2}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{8/5}+o(1)}, \quad \text{if } r \le 5/2.$$

REMARK. The algorithm of Theorem 4.12 with Coppersmith-Winograd elimination stands as the fastest factorization algorithm for which there is a rigorous proof. However, we do not advocate implementing any version of Dixon's algorithm - the interest here is purely theoretical. The remaining factorization algorithms in this paper have not been rigorously proved, but the slowest among them has a heuristic running time that is less than our champion of Theorem 4.12.

5. THE CONTINUED FRACTION ALGORITHM

Unlike the other algorithms discussed in this article, the continued fraction algorithm has actually been used to factor large numbers. For example, recently WAGSTAFF (see [5]) factored

$$\frac{3^{121}-1}{11617(3^{11}-1)},$$

a 49 digit number with no small factors, using the continued fraction algorithm.

The only difference between the continued fraction algorithm and Dixon's algorithm is in the production of the quadratic residues. In Dixon's algorithm this was done by randomly choosing values of $m \in [1,n]$ and reducing $m^2 \bmod n$, obtaining $Q(m)$. In the continued fraction algorithm, the quadratic residues are produced in a deterministic fashion. Namely, if a_i/b_i is the i-th continued fraction convergent to $\sqrt{n}$, then we let the quadratic residue Q_i be defined by

$$Q_i = a_i^2 - b_i^2 n \equiv a_i^2 \bmod n.$$

In the running of the algorithm the numbers Q_i are not computed from this definition since the numbers a_i, b_i grow geometrically. Rather there is a simple iterative procedure (described in MORRISON-BRILLHART [21]) for producing the Q_i and the $a_i \bmod n$.

From the elementary inequality (for n not a square)

$$\left|\frac{a_i}{b_i} - \sqrt{n}\right| < \frac{1}{b_i b_{i+1}},$$

it is easy to see that

$$-2\sqrt{n} < Q_i < 2\sqrt{n}. \tag{5.1}$$

Note that Q_i might very well be negative, while in Dixon's algorithm the quadratic residues $Q(m)$ are always non-negative. This minor problem is treated by using the "prime" -1 in the factorization of Q_i if $Q_i < 0$. In the final elimination step of the algorithm, when we find $Q_{i_1},\ldots,Q_{i_k}$ whose product is a square, we use the "prime" -1 an even number of times - just like every other prime.

The chief advantage that the continued fraction algorithm has over Dixon's algorithm can be found in the inequality (5.1). Since $|Q_i| < 2\sqrt{n}$, it should be easier to factor Q_i using a set of small primes than it should be for a random Q(m), which is usually much larger.

The chief difficulty in the theoretical analysis of the continued fraction algorithm lies in the deterministic nature of the Q_i. How can we be sure that the Q_i behave as ordinary numbers do in the interval $[-2\sqrt{n},2\sqrt{n}]$? In fact, if the period of the continued fraction for $\sqrt{n}$ is short, then there can only be a few different values of Q_i. (This phenomenon will usually doom the continued fraction algorithm. However a very simple device can be used in this situation: use $\sqrt{\ell n}$ rather than $\sqrt{n}$ where ℓ is a small square-free integer - see [21].)

Further complicating things is that the Q_i do not behave completely like ordinary integers. First of all, they are probably not uniformly distributed in $[-2\sqrt{n},2\sqrt{n}]$. However it is likely they are at least roughly uniformly distributed. For more on this, see KNUTH [14]. Also from the equation $a_i^2 - b_i^2 n = Q_i$, we see that if p is an odd prime and $p \nmid n$, then

$$p|Q_i \text{ implies } \left(\frac{n}{p}\right) = 1.$$

Since (n/p) = 1 usually for roughly one-half of the primes $p \le L^a$, we therefore might expect Q_i to less likely factor with these primes than would a random integer in the interval $[-2\sqrt{n},2\sqrt{n}]$. However, if (n/p) = 1, then p probably has an enhanced chance of dividing Q_i. For a random integer m, the chance that $p|m$ is 1/p. But if (n/p) = 1, there is a heuristic argument that the chance $p|Q_i$ is 2/(p+1), or almost twice as much. To see this, we assume that the $p^2 - 1$ possible values of the pair $a_i \bmod p$, $b_i \bmod p$ (note that the 0, 0 pair cannot occur since $(a_i,b_i) = 1$) are equidistributed. But for each non-zero value of $b_i \bmod p$, there are precisely 2 values of $a_i \bmod p$ for which $p|a_i^2 - b_i^2 n$. Thus the chance that $p|Q_i$ should be $2(p-1)/(p^2-1) = 2/(p+1)$. For the prime p = 2, a similar analysis gives the chance 1/3 for $2|Q_i$.

In order to make a heuristic analysis of the continued fraction algorithm, we make the following

HYPOTHESIS 5.1. *There is a constant* n_0 *such that if* $n \ge n_0$, *we have the following. There is some integer* $\ell \in [1,\log^2 n]$ *such that the period for the continued fraction for* $\sqrt{\ell n}$ *is at least* $n^{1/100}$. *For this integer* ℓ *and for any* a, $1/10 < a < 1$, *the number of primes* $p \le L(n)^a$ *for which* $p \nmid \ell n$ *and*

$(\ell n/p) = 1$ is at least $\pi(L(n)^a)/3$. If a_i/b_i is the i-th convergent to $\sqrt{\ell n}$ and $Q_i = a_i^2 - b_i^2 \ell n$, then the Q_i are distributed in $[-2\sqrt{\ell n}, 2\sqrt{\ell n}]$ with respect to a certain fraction of them having all of their prime factors below some point as are all of the integers in $[-2\sqrt{\ell n}, 2\sqrt{\ell n}]$.

For simplicity, in the following we shall always assume $\ell = 1$.

As with Dixon's algorithm we trial divide each Q_i with the primes $p \le L^a$, except now we only use those $p \le L^a$ for which $(n/p) = 1$. When we have enough completely factored Q_i, as with Dixon's algorithm we use an elimination step to assemble integers x, y with $x^2 \equiv y^2 \bmod n$. It is certainly possible for $x \equiv \pm y \bmod n$ and thus for $(x+y,n)$ to fail to be a non-trivial factor of n. We now make a heuristic assumption that this unfortunate event will not occur too often. Since we need to make similar assumptions for each of the variations of the continued fraction algorithm and for the later algorithms considered we make a more general assumption.

HYPOTHESIS 5.2. *There is a constant n_1 such that if $n \ge n_1$ and if any of the algorithms in Sections 5, 6, 7 is run long enough so that $1 + [\log^2 n]$ pairs x, y are assembled with $x^2 \equiv y^2 \bmod n$, then at least one pair will satisfy $x \not\equiv \pm y \bmod n$.*

Since a random pair x, y with $x^2 \equiv y^2 \bmod n$ satisfies $x \not\equiv \pm y \bmod n$ with probability at least 1/2 (for composite n) and since $\sum_{n=1}^{\infty} (1/2)^{\log^2 n} < \infty$, the Borel-Cantelli lemma suggests that Hypothesis 5.2 is plausible. Shanks has noticed, however, that for certain n such as $2^{60} + 2^{30} - 1$, the nature of the quadratic residues Q_i can conspire to produce many trivial pairs x, y with $x^2 \equiv y^2 \bmod n$. Thus in the case of the continued fraction algorithm and its variations, we interpret Hypothesis 5.2 to mean that some non-trivial pair x, y will be found among the first $1 + [\log^2 n]$ pairs when we work with the continued fraction expansion for $\sqrt{\ell n}$ for some $\ell \le \log^2 n$. That is, if we are having very bad luck finding a non-trivial pair x, y, we discard all calculations, choose another multiplier ℓ, and start working anew. We conjecture that this drastic procedure will prohibit Shanks' observed conspiracy, but we do not advocate its use in practice.

Thus by Hypotheses 5.1 and 5.2 we need to obtain L^a completely factored Q_i's. By Theorem 2.1 we have

$$\psi(2\sqrt{n},L^a) = \sqrt{n}L^{-(4a)^{-1}},$$

so that by Hypothesis 5.1 we must generate $L^{a+(4a)^{-1}}$ values of Q_i in order to find the requisite L^a of them which are composed solely of the primes $p \leq L^a$. The time to generate a Q_i is $O(\log^2 n)$. The time used to trial divide a Q_i with the primes $p \leq L^a$ for which $(n/p) = 1$ is, by Hypothesis 5.1, L^a. The Gaussian elimination step has a running time of L^{3a}. Thus the running time for the continued fraction algorithm is

$$L^{\max\{2a+(4a)^{-1},3a\}}.$$

Thus the optimal choice for a is $1/\sqrt{8}$. We have

THEOREM 5.1. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm is* $L(n)^{\sqrt{2}+o(1)}$. *The space required is* $L(n)^{1/\sqrt{2}+o(1)}$.

REMARK. This result was already known to SCHROEPPEL [28] and MONIER [20]. However in their analysis they assumed Theorem 2.1 without proof. Using an elementary lower bound estimate for $\psi(x,y)$ that is weaker than Theorem 2.1, KNUTH [13] gives L^2 as an upper bound for the running time.

We now consider the effect of the variations discussed in Section 4 on the running time of the continued fraction algorithm. For a description of the variations, we refer the reader to Section 4.

THEOREM 5.2. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with Large Prime is* $L(n)^{\sqrt{2}+o(1)}$. *The space required is* $L(n)^{1/\sqrt{2}+o(1)}$.

REMARKS. This result follows in a fashion similar to Theorem 4.2. Although the variation does not change the exponent in the running time, it has proved useful in practice. WUNDERLICH [34], [35] claims a running time of $L^{\sqrt{3/2}}$ for this variation. The reason for the difference is that Wunderlich assumes, contrary to the spirit of Theorem 4.1, that considering Q_i with at most one prime in $(L^a,L^{2a}]$ and all other primes in $(1,L^a]$ is not much different from considering Q_i with all prime factors in $(1,L^{2a}]$.

As we have argued above, we expect to use $L^{a+(4a)^{-1}}$ values of Q_i in the implementation of the continued fraction algorithm. If we replace trial division up to L^a with the Pollard-Strassen method (see Section 4), the time spent on each Q_i is $L^{a/2}$. Thus the total running time will be

$$L^{\max\{3a/2+(4a)^{-1},3a\}}.$$

Choosing $a = 1/\sqrt{6}$, we have

THEOREM 5.3. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm augmented with the Pollard-Strassen method is* $L(n)^{\sqrt{3/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{2/3}+o(1)}$.

REMARKS. SCHNORR [26] obtains $L^{\sqrt{3}}$ as a running time upper bound for this variation. Again, the reason for this difference is that he uses a weaker form of Theorem 2.1. SCHROEPPEL [28] achieves a running time of $L^{\sqrt{3/2}}$ for the continued fraction algorithm with the Pollard-ρ method. He assumes Theorem 2.1 without proof and also the Pollard-ρ method is assumed.

We now consider the continued fraction algorithm with k early aborts and parameters a, $c_1,\ldots,c_k$, $\theta_1,\ldots,\theta_k$ (which satisfy (4.5)). By this we mean that a value of Q_j is aborted if after trial division to $L^{\theta_i a}$ the unfactored portion exceeds $n^{\frac{1}{2}(1-c_1-\ldots-c_i)}$. The analysis is the same as in Section 4 except that we now take advantage of the fact that each $|Q_j| < 2\sqrt{n}$. The functions $f_1,\ldots,f_{k+1}$ are replaced by $F_1,\ldots,F_{k+1}$ where

$$F_i = a + \theta_i a + c_i(4\theta_i a)^{-1} + \ldots + c_k(4\theta_k a)^{-1} + (1 - c_1 - \ldots - c_k)(4a)^{-1}$$

for $i = 1,\ldots,k$ and

$$F_{k+1} = 2a + (1 - c_1 - \ldots - c_k)(4a)^{-1}.$$

We have

THEOREM 5.4. *Suppose* $1/10 < a < 1$, $c_1 > 0,\ldots,c_k > 0$, $c_1 + \ldots + c_k < 1$, *and* $0 < \theta_1 < \ldots < \theta_k < 1$. *Then assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with* k *early aborts and parameters* $a,c_1,\ldots,c_k$, $\theta_1,\ldots,\theta_k$ *is* $L(n)^{\max\{F_1,\ldots,F_{k+1},3a\}+o(1)}$.

The optimal choice of parameters occurs when the F_i are all equal. This choice is

$$a = (6 + \frac{2}{k+1})^{-1/2}, \quad c_i = \frac{4ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

With this choice of parameters, we call the algorithm the continued fraction algorithm with k early aborts.

THEOREM 5.5. *Assuming Hypotheses* 5.1 *and* 5.2, *the continued fraction algorithm with* k *early aborts has running time* $L(n)^{\sqrt{3/2+1/(2k+2)}+o(1)}$. *The space required is* $L(n)^{2/\sqrt{6+2/(k+1)}+o(1)}$.

Letting $k \to \infty$ slowly with n, we call the resulting algorithm the continued fraction algorithm with the early abort strategy.

THEOREM 5.6. *Assuming Hypotheses* 5.1 *and* 5.2, *the continued fraction algorithm with the early abort strategy has running time* $L(n)^{\sqrt{3/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{2/3}+o(1)}$.

When we combine k early aborts with the Pollard-Strassen method, the analysis of the running time requires that we change the functions $F_1,\ldots,F_{k+1}$ to $G_1,\ldots,G_{k+1}$ where

$$G_i = F_i - \frac{1}{2}\theta_i a \qquad \text{for } i = 1,\ldots,k,$$

$$G_{k+1} = F_{k+1} - \frac{1}{2}a.$$

As in Section 4, the elimination method we use now plays a role. If we use an elimination method with exponent $r > 5/2 + 1/(2k+2)$, then an optimal choice of parameters is

$$a = (4r - 5 - \frac{1}{k+1})^{-1/2}, \quad c_i = \frac{2ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

If we use an elimination method with exponent $r \le 5/2 + 1/(2k+2)$, then an optimal choice of parameters is

$$a = (5 + \frac{1}{k+1})^{-1/2}, \quad c_i = \frac{2ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

We have

THEOREM 5.7. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with the Pollard-Strassen method,* k *optimally chosen early aborts, and an elimination method with exponent* r *is*

$$L(n)^{r(4r-5-1/(k+1))^{-1/2}+o(1)}, \quad \text{if } r > \frac{5}{2} + \frac{1}{2k+2},$$

$$L(n)^{(5/4+1/(4k+4))^{1/2}+o(1)}, \quad \text{if } r \le \frac{5}{2} + \frac{1}{2k+2}.$$

Finally, letting $k \to \infty$ slowly with n, we have

THEOREM 5.8. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with the Pollard-Strassen method, the early abort strategy, and an elimination method with exponent* r *is*

$$L(n)^{\sqrt{r^2/(4r-5)}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{5/4}+o(1)}, \quad \text{if } r \le 5/2.$$

The space required is

$$L(n)^{\sqrt{4/(4r-5)}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{4/5}+o(1)}, \quad \text{if } r \le 5/2.$$

6. SCHROEPPEL'S LINEAR SIEVE ALGORITHM

Several years ago SCHROEPPEL [28] advanced a variation of the continued fraction algorithm that did away with continued fractions! His idea was to find another means of producing residues near $\sqrt{n}$ that had the advantage that they could collectively be factored by a sieve, much like the sieve of Eratosthenes. Unfortunately, these residues are not specifically quadratic residues. This requirement is arranged for in the elimination step. In this section I shall describe Schroeppel's algorithm and give a heuristic running time analysis. In addition, I shall describe a variation that gives an improvement in the running time.

Let $K = [\sqrt{n}]$ and consider the function

$$S(A,B) = (K + A)(K + B) - n,$$

where A, B are integers much smaller in absolute value than K. Then S(A,B) is not many times bigger than $\sqrt{n}$. Specifically, if $\alpha \le \beta$ are positive constants and

$$|A| \le L^{\alpha}, \quad |B| \le L^{\beta}, \quad A \le B,$$

then

$$|S(A,B)| \le n^{1/2}(L^{\alpha} + L^{\beta} + 2) = n^{1/2}L^{\beta},$$

so that $|S(A,B)| \le n^{1/2+o(1)}$. We would like to find pairs A_i, B_i such that $\prod_{i=1}^{k} S(A_i,B_i)$ is a square, as in Dixon's algorithm and the continued fraction algorithm. But now, in addition, we want each distinct value of $A_1,\ldots,A_k$, $B_1,\ldots,B_k$ to be assumed an even number of times. Then $\prod_{i=1}^{k} (K+A_i)(K+B_i)$ is also a square. So if we can find such a fortuitous collection of pairs A_i, B_i, then we can find integers x, y such that $x^2 \equiv y^2 \bmod n$. Thus, as before, we have a good chance for (x+y,n) being a non-trivial factor of n.

To find such a collection of pairs A_i, B_i we try to factor the S(A,B) with the primes up to L^a by a sieve procedure described below. To each $S(A_0,B_0)$ composed solely of the primes $p \le L^a$, we associate a 0 - 1 vector with

$$V \overset{\text{def}}{=\!=} 1 + \pi(L^a) + ([L^{\alpha}] + [L^{\beta}] + 1)$$

coordinates as follows. The first coordinate is 1 if $S(A_0,B_0) < 0$ and 0 otherwise. The next $\pi(L^a)$ coordinates are given by the parity of the exponents of the primes $p \le L^a$ in the prime factorization of $S(A_0,B_0)$. The last $[L^{\alpha}] + [L^{\beta}] + 1$ coordinates are all 0, except for the A_0-th and B_0-th coordinates, which are 1 (unless $A_0 = B_0$, in which case this coordinate is 0). If we can find V+1 such vectors, they must be linearly dependent. A linear dependency corresponds to a set of pairs A_i, B_i, $i = 1,\ldots,k$ such that

$$\prod_{i=1}^{k} S(A_i,B_i) \quad \text{and} \quad \prod_{i=1}^{k} (K+A_i)(K+B_i)$$

are both squares.

If $A = A_0$ is fixed, then $S(A_0,B)$ is a linear function in the variable B. Thus for fixed A_0, the set of $S(A_0,B)$ for $A_0 \le B \le L^{\beta}$ can be factored by

a sieve. What this means is that if $p|S(A_0,B)$, then $p|S(A_0,B+kp)$ for every integer k and if p is prime and $p \nmid n$, then $p|S(A_0,B')$ implies $B' = B + kp$ for some k. Thus once we find one B with $S(A_0,B)$ divisible by p we can easily mark every $S(A_0,B)$ which is divisible by p. The time it takes to do this is essentially just the number of such multiples of p, which is $(L^{\beta} - A_0)/p$. If we repeat this procedure for each $p \le L^a$, the total time required for a fixed A_0 is

$$(L^{\beta} - A_0) \sum_{p \le L^a} 1/p \sim (L^{\beta} - A_0) \text{loglog } L^a.$$

Now summing over all $|A_0| \le L^{\alpha}$, we have the total time spent sieving is $L^{\alpha+\beta}$. That is, in time $L^{\alpha+\beta}$ we can ascertain all of the A, B such that S(A,B) is composed solely of the primes $p \le L^a$. We have ignored the problem of finding the first B such that $p|S(A_0,B)$, but this can be quickly done since it just involves solving a linear congruence mod p. In addition, we have ignored the problem of finding the exact exponent on p in the prime factorization of $S(A_0,B)$. This can be found by either sieving with the higher powers of p, by trial division by the higher powers of p on those $S(A_0,B)$ which are multiples of p, or a combination of both approaches. The running time with any of these methods does not change our above calculation of $L^{\alpha+\beta}$.

We would like to know how many completely factored values of S(A,B) we will find. We now make a reasonable assumption about the numbers S(A,B).

HYPOTHESIS 6.1. *The numbers* $|S(A,B)|$ *for* $|A| \le L(n)^{\alpha}$, $A \le B \le L(n)^{\beta}$ *are distributed with respect to a certain fraction of them having all of their prime factors below some point as are all of the integers in* $[1, n^{1/2}(L(n)^{\alpha} + L(n)^{\beta} + 2)]$.

With this hypothesis, we therefore expect that the probability that S(A,B) completely factors over the primes up to L^a is

$$\psi(n^{1/2}L^{\beta}, L^a)/(n^{1/2}L^{\beta}) = L^{-(4a)^{-1}}$$

by Theorem 2.1. Thus we expect a total of $L^{\alpha+\beta-(4a)^{-1}}$ completely factored values of S(A,B). However, we must produce $V + 1 = L^{\max\{a,\beta\}}$ completely factored S(A,B)'s. Thus our parameters a, α, β should be chosen so that

$$\alpha + \beta - (4a)^{-1} = \max\{a,\beta\}. \tag{6.1}$$

An immediate consequence of (6.1) is that

$$\alpha + \beta \geq a + (4a)^{-1} \geq 1,$$

so that $\beta \geq 1/2$ and, a fortiori,

$$\max\{a,\beta\} \geq 1/2.$$

The value 1/2 can be attained by making the choice $a = \alpha = \beta = 1/2$, which also satisfies (6.1).

For the elimination step, we must work with a matrix of size $(V+1) \times V$ where $V = L^{\max\{a,\beta\}} \geq L^{1/2}$. Thus this step has running time at least $L^{r/2} \geq L$ if we use an elimination method with exponent r. Again the choice $a = \alpha = \beta = 1/2$ allows us to attain the lower bound. We have

THEOREM 6.1. *Assuming Hypotheses* 6.1 *and* 5.2, *the running time for Schroeppel's algorithm with an elimination method with exponent* r *is* $L(n)^{r/2+o(1)}$.

REMARK. This result was also obtained by MONIER [20] who gave separate running times of L for the sieve step and $L^{3/2}$ for the elimination step (using Gaussian elimination). Monier assumed our Theorem 2.1 without proof. SCHROEPPEL [28] claims a running time of L, but he ignores the elimination step.

Of the three speed-up variations discussed in Section 4 only Large Prime is applicable with Schroeppel's algorithm. But, for the same reason as before, this variation does not lower the exponent on the running time. In any event, this variation, as are the others, is aimed at reducing the time it takes to obtain completely factored residues. In Schroeppel's algorithm, however, the roadblock is the elimination step. A successful variation would be one that can speed up elimination.

The $(V+1) \times V$ matrix that we work with is very sparse - most entries are 0. In fact, in each row at most 2 entries among the last $[L^{\alpha}] + [L^{\beta}] + 1$ are non-zero. It would be nice to take advantage of this fact in the elimination procedure, but it is not apparent how to do this. If there were just 1 non-zero entry, rather than 2, we could proceed with a cull as described in Section 4 in connection with the Large Prime variation. We now describe an alternative method of assigning a vector to a completely factored S(A,B) that will indeed let us use a cull. To each $S(A_0,B_0)$ composed solely of the primes

$p \leq L^a$ we assign a 0 - 1 vector with

$$V' \underline{\underline{\text{def}}} 1 + \pi(L^a) + (2[L^\alpha] + 1) + ([L^\alpha] + [L^\beta] + 1)$$

coordinates as follows. The first $1 + \pi(L^a)$ coordinates are as before. The next $2[L^\alpha] + 1$ coordinates are all 0 except for the A_0-th coordinate which is 1. Similarly the last $[L^\alpha] + [L^\beta] + 1$ coordinates are all 0 except for the B_0-th coordinate which is 1.

With this different method of assigning 0 - 1 vectors to factored S(A,B)'s we can use a cull. In fact in each vector, exactly one entry among the last $[L^\alpha] + [L^\beta] + 1$ entries is non-zero. Thus we first order the rows so that the last $[L^\alpha] + [L^\beta] + 1$ columns appear in echelon form. We then make one pass down the matrix. If a B-value (that is, a non-zero entry in the last $[L^\alpha] + [L^\beta] + 1$ columns) is not repeated in a neighbouring row, then this row is culled out. If it is repeated, say k times, then the first such row is subtracted from the other k - 1 rows and is then culled out. Thus in time $L^{\max\{a,\beta\}}$ we can eliminate the last $[L^\alpha] + [L^\beta] + 1$ columns from our matrix. If we now choose a, α, β so that $\beta > \max\{a,\alpha\}$, this idea could produce a net speed-up for the algorithm. (To achieve the time $L^{\max\{a,\beta\}}$ for the cull, the 0 - 1 vectors must be treated as sparse vectors as was remarked in connection with the Large Prime variation of Dixon's algorithm.)

The running time for the sieve step will still be $L^{\alpha+\beta}$. The running time for the cull is, as we have seen, L^β. The running time for the elimination step in the smaller matrix is $L^{r\cdot\max\{a,\alpha\}}$. Thus the total running time for Schroeppel's algorithm with a cull is

$$L^{\max\{\alpha+\beta, ra, r\alpha\}}. \tag{6.2}$$

However, if we insist on $V' + 1 = L^\beta$ completely factored values of S(A,B), we do not get an improvement. That is, if we still have (as in (6.1))

$$\alpha + \beta - (4a)^{-1} \geq \beta,$$

then

$$\max\{a,\alpha\} \geq \max\{a,(4a)^{-1}\} \geq 1/2. \tag{6.3}$$

Thus from (6.2), the running time would still be at least $L^{r/2}$.

But do we really need $V' + 1$ completely factored $S(A,B)$'s? After the cull procedure we actually need only $W + 1$ vectors, where

$$W \stackrel{\text{def}}{=\!=} 1 + \pi(L^a) + (2[L^\alpha] + 1).$$

So the question is now, how many completely factored $S(A,B)$'s do we need to be assured (or nearly assured) of having $W + 1$ vectors left after the cull procedure. From (6.3) we may as well assume that $\alpha < (4a)^{-1}$, for otherwise our running time is again at least $L^{r/2}$. Thus the number of factored $S(A,B)$'s is L^γ where

$$\gamma \stackrel{\text{def}}{=\!=} \alpha + \beta - (4a)^{-1} < \beta.$$

If we have L^γ objects randomly placed in L^β cells and if we cull out one object per occupied cell, how many objects do we expect to be remaining? In fact it is $\frac{1}{2}L^{2\gamma-\beta}$ as we now see.

LEMMA 6.1. *Let* y, x *be positive integers and let* S *denote the set of functions* $f: \{1,2,\ldots,y\} \to \{1,2,\ldots,x\}$. *If* $f \in S$, *let* $Y(f) = y - |\mathrm{Rng}(f)|$, *where* $\mathrm{Rng}(f)$ *denotes the image of* f. *Then with the uniform distribution on* S, *the expected value of* Y *is*

$$y - x + x(1 - x^{-1})^y$$

and the variance is

$$x(1 - x^{-1})^y - x(1 - 2x^{-1})^y + x^2\{(1 - 2x^{-1})^y - (1 - x^{-1})^{2y}\}.$$

In particular, if $y = x^\delta$ *where* $0 < \delta < 1$ *is fixed, then as* $x \to \infty$,

$$E(Y) = \frac{1}{2}x^{2\delta-1} + O(x^{\delta-1} + x^{3\delta-2}) = \sigma^2.$$

PROOF. If $f \in S$, let $X(f) = x - |\mathrm{Rng}(f)| = x - y + Y(f)$. It will be more convenient for us to compute the expectation and variance of X. Note that the variance of X is clearly the same as the variance of Y and $E(X) = x - y + E(Y)$.

Let $X_i(f) = 0$ if $i \in \mathrm{Rng}(f)$ and 1 otherwise. Then $X = X_1 + \ldots + X_x$, so that

$$E(X) = E(X_1) + \ldots + E(X_x) = xE(X_1) = x(1 - x^{-1})^y$$

and our assertion about E(Y) follows. Also

$$E(X^2) = \sum_{i=1}^{x} \sum_{j=1}^{x} E(X_i X_j).$$

Now $E(X_i^2) = E(X_i) = (1 - x^{-1})^y$. Also if $i \neq j$, then $E(X_i X_j) = (1 - 2x^{-1})^y$. Thus

$$E(X^2) = x(1 - x^{-1})^y + (x^2 - x)(1 - 2x^{-1})^y$$

and the rest of our assertions follow from some simple calculations.

Applying the lemma with $y = L^{\gamma}$, $x = L^{\beta}$, $\delta = \gamma/\beta$, we thus see that after the cull procedure we almost surely have $L^{2\gamma-\beta}$ vectors left. Thus we should choose a, α, β so that $L^{2\gamma-\beta} = W + 1 = L^{\max\{a,\alpha\}}$. That is,

$$2\alpha + \beta - (2a)^{-1} = \max\{a,\alpha\}.$$

Thus $2\alpha + \beta - (2a)^{-1} \geq \alpha$, so that

$$\alpha + \beta \geq (2a)^{-1}.$$

Hence the exponent in the running time, given by (6.2), satisfies

$$\max\{\alpha+\beta, ra, r\alpha\} \geq \max\{\alpha+\beta, ra\} \geq \max\{(2a)^{-1}, ra\} \geq \sqrt{r/2}.$$

Moreover, $\sqrt{r/2}$ can be attained by choosing

$$a = \alpha = 1/\sqrt{2r}, \quad \beta = (r-1)/\sqrt{2r}.$$

With this choice we have our other requisite as well, namely $\alpha \leq (4a)^{-1}$.

Note that the space requirement is apparently $L^{(r+2)/\sqrt{8r}}$, the size of the matrix before the cull. But, in fact, less space is required. Until now we have made the "natural" choice of fixing each A and sieving over the B's - natural because it makes sense to sieve over as long an interval as

possible. If instead we fix B and sieve over the A's then all of the factored S(A,B)'s with a given value of B are found at one time and the cull procedure can be immediately applied to this smaller set. Thus the space required is the size of the matrix after the cull, or $L^{\sqrt{2/r}}$. (This is also the size of the interval sieved.)

THEOREM 6.2. *Assuming Hypotheses* 6.1 *and* 5.2 *and that the function* f *that takes a completely factored* S(A,B) *to* B *is pseudo-random with respect to the property of Lemma* 6.1, *then the running time for Schroeppel's algorithm with a cull and with an elimination method of exponent* r *is* $L(n)^{\sqrt{r/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{2/r}+o(1)}$.

REMARK. The Large Prime variation cannot be profitably used with Schroeppel's algorithm with a cull. Indeed, this strategy requires a cull of its own and a cull on one set of data destroys the ability to perform a cull on another set of data.

7. THE QUADRATIC SIEVE ALGORITHM

The advantage of Schroeppel's linear sieve algorithm over the previous algorithms is that a sieve process is substituted for trial division or the Pollard-Strassen method. The disadvantage of Schroeppel's algorithm, however, is that the residues produced are not quadratic residues. The task of producing quadratic residues is given over to the elimination step which then enters the picture as an important time bottleneck and as a large user of space. In this section I shall describe what I call the quadratic sieve algorithm - it shares the advantage of Schroeppel's algorithm, but not the disadvantage.

Briefly, the quadratic sieve algorithm can be described as the case A = B of Schroeppel's algorithm. That is, if $K = [\sqrt{n}]$, we let

$$S(A) = (K+A)^2 - n,$$

so that S(A) = S(A,A) in the notation of Section 6. Then, quite evidently, S(A) is a quadratic residue mod n. These quadratic residues, as with the continued fraction algorithm, do not lie much above $\sqrt{n}$. In fact, if $|A| \le L^b$, then

$$|S(A)| \le \sqrt{n}(2L^b + 2) = \sqrt{n}L^b.$$

What is not so obvious is that the S(A) can be factored with a sieve, but this is nevertheless the case. In fact, the consecutive values of any polynomial can be factored by a sieve-type procedure. We describe the procedure for our particular polynomial S(A). For each prime p, we find the solutions of the congruence

$$S(A) \equiv 0 \bmod p. \tag{7.1}$$

Because of the nature of S(A), (7.1) has solutions precisely when $(n/p) = 1$, $p|n$, or $p = 2$. Say $(n/p) = 1$. Then (7.1) has two solutions $A_1(p)$, $A_2(p)$. (These solutions may be found by the probabilistic method of BERLEKAMP [2] or LEHMER [17] in time bounded by $0((\log p)^c)$ or by trial and error in time bounded by $0(p(\log p)^c)$.) Then (7.1) holds if and only if

$$A \equiv A_1(p) \bmod p \quad \text{or} \quad A \equiv A_2(p) \bmod p.$$

Thus if we have a list of consecutive values of S(A), we can quickly find all of them which are multiples of p by marking every p-th one starting at some $A_1 \equiv A_1(p) \bmod p$ and by marking every p-th one starting at some $A_2 \equiv A_2(p) \bmod p$. Thus, our sieve procedure requires two passes down the list for each p for which $(n/p) = 1$. Divisibility by higher powers of p can be decided, as in Section 6, by either trial division, sieving by higher powers of p, or by a combination approach.

Say now we work with the primes $p \le L^a$ where $1/10 < a < 1$. As in Section 3 we assume that n is divisible by no prime $p \le L$. Say we consider the S(A) for $|A| \le L^b$. The problem of deciding the correct power of 2 in each S(A) is simple - this can be done by a sieve procedure or trial division in time L^b. So say now $2 < p \le L^a$. We solve (7.1) for each such p, finding when $(n/p) = 1$ the two solutions $A_1(p)$, $A_2(p)$. The time required for this step is either L^a or L^{2a} depending on whether we use a clever or stupid method (we shall assume the latter). We then use the above-described sieve to determine for each such p and each A with $|A| \le L^b$, the correct exponent on p in the prime factorization of S(A). As in Section 6, the time required is

$$\sum_{\substack{p \le L^a \\ (n/p)=1}} L^b/p = L^b,$$

provided there are at least a few small primes p for which $(n/p) = 1$.

The question now, as it has been in all of our previous algorithms, is how many of our residues are composed solely of the primes $p \le L^a$?

HYPOTHESIS 7.1. *There is a constant* n_2, *such that if* $n \ge n_2$, *we have the following. For any* a, $1/10 < a < 1$, *the number of primes* $p \le L(n)^a$ *for which* $p \nmid n$ *and* $(n/p) = 1$ *is at least* $\pi(L(n)^a)/3$. *For any* $b > 1/10$, *the numbers* $|S(A)|$ *with* $|A| \le L(n)^b$ *are distributed with respect to a certain fraction of them having all of their prime factors below some point as are all of the integers in* $[1, \sqrt{n}(2L(n)^b + 2)]$.

With this hypothesis and Theorem 2.1, we can indeed predict how many of the L^b values of $S(A)$ are composed solely of the primes $p \le L^a$. Namely, there are $L^{b-(4a)^{-1}}$ such values.

The elimination stage of the algorithm is exactly the same as with Dixon's algorithm or the continued fraction algorithm. We thus wish to choose b so that we have L^a factored values of $S(A)$. Thus we choose $b = a + (4a)^{-1}$. If we use an elimination algorithm with exponent r, the running time for the quadratic sieve algorithm will be

$$L^{\max\{2a, a+(4a)^{-1}, ra\}}.$$

The space requirement is L^{2a} for the matrix and $L^{a+(4a)^{-1}}$ for the sieve. However the latter requirement can be eased by breaking our long interval into smaller intervals of length L^{2a}. There is a certain start-up cost for sieving a new interval, but this is $\le L^{2a}$. Thus without changing the running time we can thus get by with a space requirement of L^{2a}. We choose

$$a = 1/\sqrt{4(r-1)}$$

and obtain

THEOREM 7.1. *Assuming Hypotheses* 7.1 *and* 5.2, *the running time for the quadratic sieve algorithm using an elimination method with exponent* r *is* $L(n)^{r/\sqrt{4r-4}+o(1)}$. *The space required is* $L(n)^{1/\sqrt{r-1}+o(1)}$.

REMARKS. The function $r/\sqrt{4r-4} = 1 + O((r-2)^2)$ as $r \to 2$. Thus this exponent is not very sensitive to changes in r. In fact the running time exponent for the quadratic sieve algorithm with Gaussian elimination is only about 4% higher than the exponent with Coppersmith-Winograd elimination. There

are several possible variations of the quadratic sieve algorithm, but none of these change the exponent in the running time. Among these are the Large Prime variation discussed in Section 4, the use of a multiplier ℓ so that ℓn is a quadratic residue for more small primes than is n, and the use of the polynomials $S_i(A) = ([i\sqrt{n}] + A)^2 - i^2 n$ for various small i.

8. ELIMINATION ALGORITHMS

In Section 4 we said that an algorithm for finding a linear dependency among a set of vectors in $(\mathbb{Z}/2\mathbb{Z})^k$ has exponent r if the running time is $k^{r+o(1)}$. A more usual notion though is that of a matrix multiplication exponent. That is, an algorithm for multiplying two k×k matrices has exponent r if the number of arithmetic operations involved is $k^{r+o(1)}$. (Note that when working over $\mathbb{Z}/2\mathbb{Z}$, the number of arithmetic operations is proportional to the running time.) In this section we show that a matrix multiplication algorithm of exponent r can be used to construct an elimination algorithm of exponent r.

But first we say a word about matrix multiplication. The naive algorithm clearly has exponent 3. About 12 years ago, Schönhage introduced a matrix multiplication algorithm with exponent smaller than 3. His method is based on a tricky way of multiplying 2×2 matrices over a non-commutative ring using only 7 multiplications and a bounded number of additions. Iterating this procedure for larger matrices gives rise to an algorithm with exponent $\log 7/\log 2 \approx 2.807$. This result began a long series of better and better algorithms. The current champions are COPPERSMITH and WINOGRAD [8] who have shown the existence of an algorithm with exponent smaller than 2.495548. For the history before this development, the reader is referred to the survey of LAZARD [16].

So say now we are working with a particular matrix multiplication algorithm M that requires M(k) arithmetic operations to multiply two general k×k matrices. Suppose S is a set of k + 1 vectors each with k coordinates in $\mathbb{Z}/2\mathbb{Z}$. If $T \subset S$, let A_T denote the matrix whose rows are the vectors in T.

Suppose $m \leq k$, m is a power of 2, and A is an m×k matrix. In AHO, HOPCROFT, and ULLMAN [1], Fig. 6.4, an algorithm is described to find matrices L, U, P where L is m×m unit lower triangular, U is m×k upper triangular, and P is a k×k permutation matrix, such that

$$A = LUP,$$

provided A has rank m; if A has rank < m, the algorithm will indicate this fact as well. Moreover, if M is used for all matrix multiplications, the running time will be $O(M(k))$. By embedding A in a possibly larger matrix, we can drop the restriction that m be a power of 2.

By using this algorithm on A_T for T a proper subset of S, we can find whether the vectors in T are independent or dependent. Thus with a binary search, we can find in $O(M(k)\log k)$ arithmetic steps a subset $T_0 \subset S$ and a vector $v_0 \in S - T_0$, such that T_0 is independent but $T_0 \cup \{v_0\}$ is dependent. Thus v_0 is in the span of T_0.

Say $T_0 = \{v_1,\ldots,v_m\}$. Our problem is to find $x_1,\ldots,x_m$ in $\mathbb{Z}/2\mathbb{Z}$ such that $x_1v_1 + \ldots + x_mv_m = v_0$, that is,

$$[x_1 \ldots x_m]A_{T_0} = v_0.$$

But from the decomposition $A_{T_0} = L_0U_0P_0$, which we can find from the above algorithm, this equation is easy to solve. Since P_0 is a permutation matrix, its inverse can be found in time $O(k)$ (see [1], p.239). Moreover, since U_0 is upper triangular, a solution $y_1,\ldots,y_m$ to

$$[y_1 \ldots y_m]U_0 = v_0P_0^{-1},$$

if a solution exists (and it will in our case), can be found by a back substitution technique in $O(k^2)$ arithmetic operations. We can similarly solve

$$[x_1 \ldots x_m]L_0 = [y_1 \ldots y_m].$$

In summary, we can find the linear dependency we need for our integer factorization algorithms in $O(M(k)\log k)$ arithmetic operations. Thus if $M(k) = k^{r+o(1)}$, then the running time for the corresponding dependency finding algorithm also is $k^{r+o(1)}$.

9. PRACTICAL CONSIDERATIONS

We remarked in Section 1 that an asymptotic running time analysis is by no means the last word on an algorithm. In this section I speculate on the "real world" applicability of the various ideas in this paper.

To begin, I am doubtful that a substitute for Gaussian elimination should be used. Because we are working with matrices over $\mathbb{Z}/2\mathbb{Z}$, computers

can be programmed to handle long strings of entries as a single multi-digit object when one row is subtracted from another. Thus, although Gaussian elimination requires ck^3 steps for a k×k matrix, the coefficient c can be made fairly small. Perhaps a more important problem than speeding up the elimination step is to find a convenient way of solving the space problem. In each algorithm discussed, the space constraint is caused by the problem of working with the final matrix. Perhaps the research in fast matrix multiplication can help with this problem since these ideas involve iterative procedures where only pieces of a matrix are needed in core at any given time.

I am also doubtful that the Pollard-Strassen method should be used to replace trial division in the continued fraction algorithm. However use of the Pollard-ρ method is a distinct possibility. The asymptotic time analysis should be the same. The reason we used the Pollard-Strassen method in our discussions is because it is fully proved and deterministic.

I am optimistic about the early abort strategy either by itself or supplemented with the Pollard-ρ method. Samuel Wagstaff and I did a statistical analysis of the data computed during his 70 hour factorization of

$$\frac{3^{121}-1}{(3^{11}-1)11617}$$

using the continued fraction algorithm with Large Prime. We predict that if everything were left the same except that after trial division to 100 a value of Q_i gets aborted if the unfactored portion exceeds 10^{19}, then the running time would have been only 20 hours. In further fine tuning of the early abort strategy, Wagstaff and I are currently achieving about a 10 fold speed up over the straight continued fraction algorithm with Large Prime for numbers of about 50 decimal digits.

In [36], WUNDERLICH suggests programming the continued fraction algorithm on a parallel processor such as the English ICL-DAP. In particular he suggests that the pair $a_i \bmod n$, Q_i be computed on a sequential processor and then a large batch of Q_i's be trial divided in parallel. The assumption is that the $a_i \bmod n$, Q_i generation only accounts for a small percentage of the total running time, so that it is not too important to improve this part of the program. While this is undoubtedly true asymptotically, in practice the $a_i \bmod n$, Q_i generation does take a significant amount of time, say 5 to 10%. Moreover, if the early abort strategy is utilized, then less time is spent on average with a pair $a_i \bmod n$, Q_i, so that the time spent generating the pairs may take even 30% of the total running time. There are ways to

speed up the generation of these pairs (for example, the costly reduction of a_i modulo n can be delayed so that it is done only when $i \equiv 1,2 \bmod 100$), but it may be advantageous to find a way to produce the $a_i \bmod n$, Q_i pairs in parallel. Using an idea of SHANKS [29], it is in fact possible to cheaply jump ahead in the continued fraction expansion from Q_i to Q_j where $j \approx 2i$ when i is large. Thus, an alternative to working with an exotic parallel processor might be to simultaneously implement the continued fraction algorithm on a large bank of unextraordinary computers, each sequentially working on a different interval of consecutive terms in the continued fraction expansion.

Concerning the two sieve algorithms, I feel that the quadratic sieve is superior to the linear sieve. Even so, I am not sure that the quadratic sieve is practical. The main drawback for the quadratic sieve as compared with the continued fraction algorithm appears to be the size of the quadratic residues S(A). Most of the $|S(A)|$ are about $n^{1/2+\varepsilon}$ where $\varepsilon \to 0$ as $n \to \infty$. But for a fixed n, say about 10^{50} or 10^{60}, ε may well be as large as 1/6. Thus the quadratic residues would be near $n^{2/3}$ for such n. In comparison, the continued fraction algorithm's quadratic residues Q_i always satisfy $|Q_i| < 2n^{1/2}$ and some are much smaller. A larger magnitude for the quadratic residues of course makes it less likely they will completely factor over a set of small primes. On the other hand, an advantage of the quadratic sieve algorithm over the continued fraction algorithm is that with the former almost all of the steps can be performed with single precision numbers. Indeed, when sieving, every time there is a "hit" with a prime p (or with a prime power p^j), instead of dividing p into S(A) to produce the quotient, one can instead subtract log p from log S(A) where these logs are only singly precise. If after several subtractions, the value of the logarithm left is close to 0, then S(A) has been completely factored, while if the value is below a certain bound, then S(A) has factored but for a single large prime which can be found by division. The overwhelming majority of values of S(A) are not completely factored; for them no division is necessary. GERVER [10] has recently used the quadratic sieve algorithm to factor a 47 digit composite factor of $3^{225} - 1$ taking 70 hours on an HP3000. It remains to be seen if fine tuning the algorithm can produce better results. A final remark is that it would be very easy to simultaneously use many computers with the quadratic sieve algorithm (at least for the sieving step) since each computer can sieve over its own interval.

REFERENCES

[1] AHO, A.V., J.E. HOPCROFT & J.D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[2] BERLEKAMP, E.R., *Factoring polynomials over large finite fields*, Math. Comp. 24 (1970), 713-735.

[3] BORODIN, A. & I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier, New York, 1975.

[4] BRENT, R.P. & J.M. POLLARD, *Factorization of the eighth Fermat number*, Math. Comp. 36 (1981), 627-630.

[5] BRILLHART, J., D.H. LEHMER, J.L. SELFRIDGE, B. TUCKERMAN & S.S. WAGSTAFF, JR., *Factorizations of* $b^n \pm 1$ *up to High Powers*, to be published by the Amer. Math. Soc.

[6] BRUIJN, N.G. DE, *On the number of positive integers* $\leq x$ *and free of prime factors* $> y$, *II*, Nederl. Akad. Wet. Proc. Ser. A 69 = Indag. Math. 38 (1966), 239-247.

[7] CANFIELD, E.R., P. ERDÖS & C. POMERANCE, *On a problem of Oppenheim concerning "Factorisatio Numerorum"*, J. Number Theory, to appear.

[8] COPPERSMITH, D. & S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput. 11 (1982), 472-492.

[9] DIXON, J.D., *Asymptotically fast factorization of integers*, Math. Comp. 36 (1981), 255-260.

[10] GERVER, J.L., *Factoring large numbers with a quadratic sieve*, Math. Comp., to appear.

[11] GUY, R.K., *How to factor a number*, Proc. Fifth Manitoba Conf. Numer. Math., Utilitas, Winnipeg (1975), 49-89.

[12] HOOGENDOORN, P.J., *On a secure public-key cryptosystem*, this volume.

[13] KNUTH, D.E., *The Art of Computer Programming*, vol. 2 *Seminumerical Algorithms*, 2nd edition, Addison Wesley, Reading, Mass., 1981.

[14] KNUTH, D.E., *The distribution of continued fraction approximations*, to appear.

[15] KRAITCHIK, M., *Recherches sur la théorie des nombres*, Tome II, Gauthier-Villars, Paris, 1929.

[16] LAZARD, D., *Sur le produit de matrices*, Gazette des Math., No. 15 (Dec. 1980), 27-48.

[17] LEHMER, D.H., *Computer technology applied to the theory of numbers*, in W.J. LeVeque, ed., MAA Stud. Math. 6 (1969), 117-151.

[18] LEHMER, D.H. & R.E. POWERS, *On factoring large numbers*, Bull. Amer. Math. Soc. 37 (1931), 770-776.

[19] LENSTRA, H.W. JR., *Primality testing*, this volume.

[20] MONIER, L., *Algorithmes de factorisation d'entiers*, Thèse de 3^{me} cycle, Orsay (1980).

[21] MORRISON, M.A. & J. BRILLHART, *A method of factoring and the factorization of* F_7, Math. Comp. 29 (1975), 183-205.

[22] POLLARD, J.M., *Theorems on factorization and primality testing*, Proc. Cambridge Phil. Soc. 76 (1974), 521-528.

[23] POLLARD, J.M., *A Monte Carlo method for factorization*, BIT 15 (1975), 331-334.

[24] POMERANCE, C., *Recent developments in primality testing*, Math. Intelligencer, 3 (1981), 97-105.

[25] RABIN, M.O., *Probabilistic algorithm for primality testing*, J. Number Theory 12 (1980), 128-138.

[26] SCHNORR, C.P., *Refined analysis and improvements on some factoring algorithms*, J. Algorithms 3 (1982), 101-127. Also see the extended abstract in S. Even and O. Kariv, eds., *Automata, Languages and Programming*, Eighth Colloquium, Acre (Akko), July 1981, Lecture Notes in Computer Science #115, Springer-Verlag, 1-15.

[27] SCHOOF, R.J., *Quadratic fields and factorization*, this volume.

[28] SCHROEPPEL, R., private correspondences dated 1977 (communicated to the author by H.C. Williams).

[29] SHANKS, D., *The infrastructure of a real quadratic field and applications*, Proc. 1972 Number Theory Conf. Boulder, Colorado, 217-224.

[30] SOLOVAY, R. & V. STRASSEN, *A fast Monte-Carlo test for primality*, SIAM J. Comput. 6 (1977), 84-85; erratum: 7 (1978), 118.

[31] STRASSEN, V., *Einige Resultate über Berechnungskomplexität*, Jahresber. Deutsch. Math.-Verein 78 (1976/77), 1-8.

[32] TURK, J.W.M., *Fast arithmetic operations on numbers and polynomials*, this volume.

[33] VOORHOEVE, M., *Factorization algorithms of exponential order*, this volume.

[34] WAGSTAFF, S.S. JR. & M.C. WUNDERLICH, *A comparison of two factorization methods*, unpublished manuscript.

[35] WUNDERLICH, M.C., *A running time analysis of Brillhart's continued fraction factoring method*, in M.B. Nathanson, ed., *Number Theory Carbondale 1979*, Lecture Notes in Math. 751 (1979), 328-342.

[36] WUNDERLICH, M.C., *A report on the factorization of 2797 numbers using the continued fraction method*, unpublished manuscript.

PERFECT NUMBERS AND ALIQUOT SEQUENCES

by

H.J.J. TE RIELE

"Too much emphasis cannot be placed upon the fact that at long last the courageous arithmetician has been emancipated forever from the slavery of hand computation on Mersenne and analogous numbers by the appearance on Earth of the magnificent digital calculating assemblages."

Horace S. Uhler [61, p. 126]

1. INTRODUCTION

Perfect numbers are defined as positive integers n for which $\sigma(n) = 2n$, where $\sigma(n)$ denotes the sum of the divisors of n. By $s(n)$ we denote the sum of the *aliquot**) divisors of n, i.e., the divisors of n except n itself. Then, alternatively, we can define perfect numbers as positive integers n satisfying the equation $s(n) = n$.

One could ask the question what happens when we repeatedly apply the function $s(\cdot)$ to a given positive integer n. The resulting sequence n, $s(n)$, $s(s(n)) =: s^2(n),\ldots,s^i(n) := s(s^{i-1}(n)),\ldots$ is called the *aliquot sequence of* n. When n is a perfect number, its aliquot sequence is n,n,n,..., which can be interpreted as a *periodic* sequence with period length 1. One might then ask whether periodic sequences exist with period length *greater* than 1. At this moment, aliquot sequences are known with period lengths 1, 2 (in this case the two numbers involved form a so-called *amicable number pair*), 4, 5 and 28. Another possible behaviour of an aliquot sequence is illustrated by the following pattern which occurs frequently: ...,p, 1, 0, where p is some prime number. The sequence terminates, since $s(0)$ is not defined. A "third" behaviour occurs when the terms become so large that it takes too much computer time to compute subsequent terms.

Perfect numbers, the four smallest of which were known already to the Greeks, have played an important role in computational number theory. The fastest computers have always been used (and even been tested) to find

---

*) "aliquot" (L.) literally means: several, some.

greater perfect numbers, not in the least because the greatest known *prime* numbers are connected with the greatest known perfect numbers. This will be clarified in Section 2. The interest in aliquot sequences has stimulated to a great extent the development of fast factorization algorithms.

In the study and the computation of aliquot sequences (and in many other number-theoretical problems) two basic approaches can be discerned: a direct, exhaustive computer approach (for instance to compute *all* amicable number pairs in a certain range), and an indirect non-exhaustive approach which first involves (some) theoretical work before programming some algorithm and running it (for instance to compute amicable number pairs of a special form). In general, the second approach is more complicated than the first, although the first approach often gives hints for a justifiable second approach (for instance, the smallest amicable number pair, (220, 284), probably found by an exhaustive search, has a special structure and several other pairs of the same structure have been found later by the second approach).

In this paper we shall sketch the main developments with respect to perfect numbers and aliquot sequences, since the advent of high speed electronic computers (1950, say). Sections 2-4 are devoted, respectively, to perfect numbers, amicable number pairs, and periodic aliquot sequences of period length > 2. In Section 5 we shall deal, mainly, with non-periodic aliquot sequences, including the problem of the existence of unbounded sequences.

2. PERFECT NUMBERS

The smallest perfect number is 6. Its aliquot divisors are 1, 2 and 3, whose sum is $6 = 2\cdot 3$. The next three perfect numbers are $28 = 2^2 7$, $496 = 2^4 31$ and $8128 = 2^6 127$. One can check this by using (i) the multiplicative property of the σ-function, i.e., if $(a,b) = 1$, then $\sigma(ab) = \sigma(a)\sigma(b)$, and (ii) the formula $\sigma(p^a) = p^a + p^{a-1} + \ldots + p + 1 = (p^{a+1} - 1)/(p-1)$, for every prime p and exponent $a \in \mathbb{N}$. For instance, $\sigma(8128) = \sigma(2^6 127) = \sigma(2^6)\sigma(127) = (2^7-1)\cdot 128 = 2^7(2^7-1) = 2\cdot 8128$. These four perfect numbers were known already to the Greeks and they are of the form

$$2^{n-1}(2^n-1), \quad \text{where } 2^n - 1 \text{ is a prime.} \tag{2.1}$$

In fact, Euclid proved that *all* numbers of this form are perfect numbers. Indeed, if $N = 2^{n-1}p$, $p = 2^n - 1$ prime, then $\sigma(N) = (2^n-1)(p+1) = 2N$. Only

about 2000 years after Euclid, Euler proved a converse of this result, viz., that every *even* perfect number should have the form (2.1). A proof (by L.E. Dickson) reads as follows: let $N = 2^{n-1}m$ be an even ($n \geq 2$) perfect number, where m is odd. Then $\sigma(N) = (2^n-1)\sigma(m) = 2^n m$, so that

$$\sigma(m) = m + m/(2^n-1). \tag{2.2}$$

Since $\sigma(m)$ is an integer, $m/(2^n-1)$ must be an integer, so that 2^n-1, and hence $m/(2^n-1)$, are divisors of m. Apparently, since $\sigma(m)$ is the sum of *all* divisors of m, by (2.2) m and $m/(2^n-1)$ are the *only* divisors of m; hence $m/(2^n-1) = 1$, $m = 2^n-1$ and m is a prime number.

Now the even perfect numbers are completely characterized and one "only" needs to find prime numbers $p = 2^n - 1$ in order to find even perfect numbers. Before discussing this problem we turn to the *odd* perfect numbers. The pattern here is completely different. No odd perfect number is known, nor such a simple characterization like (2.1) for the even perfect numbers. However, many conditions are known, which an odd perfect number should satisfy. For instance, any odd perfect number should have the form $N = p^{4a+1}m^2$, where p is a prime of the form $4k+1$, $a \in N$, and m is some odd integer, not divisible by p. The proof is not difficult: let $N = \prod_{i=1}^{s} p_i^{e_i}$ be an odd perfect number, then $\sigma(N) = 2N$, $\prod_{i=1}^{s} (p_i^{e_i} + \ldots + p_i+1) = 2\prod_{i=1}^{s} p_i^{e_i}$. The right hand side has precisely one factor 2, so there must be an index $i = j$ such that $2 \| (p_j^{e_j} + \ldots + p_j+1)$, whereas all the other factors in the left hand side must be odd. It is now an easy exercise to conclude that $p_j \equiv e_j \equiv 1 \pmod 4$, and that e_i is even for $1 \leq i \leq s$, $i \neq j$.

Some other conditions which odd perfect numbers should satisfy, but which take much more ingenious reasoning and computations to derive, are the following:

- any odd perfect number should be greater than 10^{50} [27];
- any odd perfect number should have at least 8 distinct prime factors [28];
- the greatest prime factor of an odd perfect number should be greater than 300000 [14];
- the second greatest prime factor should be greater than 1000 [30];
- odd perfect numbers *not* divisible by 3 should be divisible by at least 10 distinct primes [34].

As an illustration we mention that the full proof of the 10^{50}-bound is a 83-page notebook, which was deposited by the author (Peter Hagis, Jr.) in

the UMT-file of "Mathematics of Computation". This proof has been carefully checked by TUCKERMAN [59]. BUXTON & ELMORE [11] claim a proof that any odd perfect number should exceed 10^{200}, but no details are known, nor any confirmation.

Back to the even perfect numbers. As said before, they can be found by finding prime numbers of the form 2^n-1. If $n = n_1 n_2$ is composite, then $2^n-1 = (2^{n_1}-1)(2^{n_1(n_2-1)}+2^{n_1(n_2-2)}+\ldots+2^{n_1}+1)$ is composite, hence a necessary condition for 2^n-1 to be prime is that n be prime. Numbers of the form 2^p-1, p a prime, are called *Mersenne numbers* and we shall denote them by M_p. If M_p itself is prime, it is called a *Mersenne prime*. The first four Mersenne numbers are the primes $M_2 = 3$, $M_3 = 7$, $M_5 = 31$ and $M_7 = 127$. The next Mersenne number, $M_{11} = 2047$, is composite (viz., 23·89). It is not known who found the fifth Mersenne prime, $M_{13} = 8191$, but Cataldi, in 1588, showed that M_{17} and M_{19} are also primes. A fascinating account of the search of Mersenne primes with many excursions to and unsuspected connections with other important parts of number theory can be found in the first chapter of SHANKS's book [56]. We shall only mention here the main tools which have been developed in the search for Mersenne primes.

All factors of M_p must have the form $k\cdot 2p+1$, $k \in \mathbb{N}$, and, simultaneously, the form $8r \pm 1$, $r \in \mathbb{N}$. For example, $M_{11} = 2047 = 23\cdot 89 = (22+1)(4\cdot 22+1) = (8\cdot 3-1)(8\cdot 11+1)$. Using these tools it takes only one trial division to show that M_{23} is composite, since the smallest number of the form $k\cdot 46+1$ is 47, which is $\equiv -1 \pmod 8$ and, indeed, we find $47 | M_{23}$. The next candidate is M_{29}, a nine digit number. The smallest candidate of the form $k\cdot 58+1$ which is $\pm 1 \pmod 8$ *and* which is a prime number is 233. And again we find $233 | M_{29}$. The next Mersenne number is $M_{31} = 2147483647$. There are only 84 prime divisors to be checked, which are $\leq \sqrt{M_{31}}$, and which are $\equiv -1 \pmod{62}$ and $\equiv \pm 1 \pmod 8$. In this way, Euler showed in 1772 that M_{31} is a prime number and it remained the largest known prime for over 100 years! Another tool which is useful to sieve out immediately certain composite M_p is the following: if $p \equiv 3 \pmod 4$ is a prime, and if $q = 2p+1$ is also a prime, then $q | M_p$. The smallest five M_p which can be shown to be composite with this tool are M_{11}, M_{23}, M_{83}, M_{131} and M_{179}.

In 1876, a breakthrough came with the discovery by E.A. LUCAS [39] of a very fast test for the primality of M_p, and in 1930, D.H. LEHMER [37] published the following *improved version of Lucas's test:*

M_p is prime if and only if $S_{p-1} \equiv 0 \pmod{M_p}$ where S_{p-1} is the (p-1)-th term of the sequence S_i, $i = 1,2,\ldots$, defined by $S_1 = 4$, $S_{i+1} = S_i^2-2$, $i \leq 1$.

For a proof we refer to the contribution of H.W. Lenstra, Jr., in this tract.

The important features of this test are (i) that it gives not only a necessary, but also a sufficient criterion for the primality of M_p and (ii) that the number of operations is $O(p^3)$, for $p \to \infty$, whereas, until Lucas, this number was $O(2^{p/2})$. It is this Lucas-Lehmer test by which the largest known primes have been found.

By the year 1947, near the beginning of the electronic computer era, it was found by the Lucas-Lehmer test and mechanical calculators that the 9-th to 12-th Mersenne primes are M_{61}, M_{89}, M_{107} and M_{127}, whereas the other M_p for $31 < p \leq 257$ are composite. In Table 2.1 we list the prime numbers $p \leq 44497$ for which M_p is known to be prime, with the discoverer(s), the computer and, in some cases, the computer time needed to test M_p with the Lucas-Lehmer test.

TABLE 2.1. The Mersenne primes M_p for $p \leq 44497$

p with $M_p = 2^p - 1$ prime	discoverer, computer, time in hours for some M_p
2,3,5,7,13,17,19,31, 61,89,107,127	discovered in the pre-electronic computer era
521,607,1279,2203,2281	ROBINSON [53], SWAC, 1.0 for M_{2281}
3217	RIESEL [52], BESK, 5.5 for M_{3217}
4253,4423	HURWITZ [32], IBM 7090, 0.8 for M_{4423}
9689,9941,11213	GILLIES [23], ILLIAC II, 2.3 for M_{11213}
19937	TUCKERMAN [60], IBM 360/91, 0.6 for M_{19937}
21701	NOLL & NICKEL [41], CDC CYBER 174, 7.7 for M_{21701}
23209	NOLL [41], CDC CYBER 174, 8.7 for M_{23209}
44497	SLOWINSKI [57], CRAY-1

What can we expect for the future? The cycle time of the CRAY-1 Computer is about 12 nanoseconds but very recently prospects of an even faster computer with a cycle time of 1 nanosecond (= 10^{-9} sec.) have been given by MATISOO [40]. Therefore, it is not unreasonable to expect the discovery of two or three more Mersenne primes (with the Lucas-Lehmer test) before the turn of this century. However, certain serious physical barriers come into sight with the advent of these superfast computers [40]: an electric signal can travel about 15 centimeters in one nanosecond, so that no signal path can be much longer than that if the cycle-time should be one nanosecond!

3. AMICABLE NUMBER PAIRS

In 1972, Elvin J. Lee and Joseph S. Madachy published a paper entitled "The History and Discovery of Amicable Numbers" [35]. This survey paper contains a list of over 1100 amicable pairs known at that time to Lee and Madachy. The largest pair in this list is a pair of two 25-digit numbers. *)

An amicable (number) pair was defined in the introduction as a pair of positive integers (n,m), $n \neq m$, such that $s(n) = m$ and $s(m) = n$. An alternative definition is obtained by writing $s(n) = \sigma(n) - n$ and $s(m) = \sigma(m) - m$, so that $\sigma(n) = n+m = \sigma(m)$. The smallest amicable pair is $(220,284)$. Indeed, $\sigma(220) = \sigma(2^2 \cdot 5 \cdot 11) = 7 \cdot 6 \cdot 12 = 504 = 220 + 284 = 7 \cdot 72 = \sigma(2^2 \cdot 71) = \sigma(284)$. This pair was known already to the Greeks, about 500 B.C.

Only three pairs were known before EULER [22] studied amicable pairs in a systematic way, viz., the "Greek" pair $(2^2 5 \cdot 11, 2^2 71)$, the pair $(2^4 23 \cdot 47, 2^4 1151)$ found by Fermat in 1636, and the pair $(2^7 191 \cdot 383, 2^7 73727)$ found by Descartes in 1638 . These three pairs are of the form $(2^n pq, 2^n r)$, where p, q and r are prime numbers. This suggested the general plan of four of the five methods which Euler constructed: to find amicable pairs of the form (EM,EN), where E is relatively prime to both M and N. The common factor E is *given* and M and N are the *unknowns*. Let $S := \sigma(E)$, then we must solve the equations

$$S\sigma(M) = E(M+N) = S\sigma(N).$$

Euler considered various combinations of variables in M and N. His simplest choice was $M = pq$, $N = r$, where p, q and r are distinct prime numbers. This gives

$$\begin{cases} S(p+1)(q+1) = E(pq+r) \\ (p+1)(q+1) = r+1. \end{cases}$$

Eliminating r from the first equation with the second, and rearranging, yields

$$(2E-S)pq + (E-S)(p+q) = S.$$

On multiplying both sides by $2E - S$ and adding $(E-S)^2$ to both sides, we can write the resulting left hand side as the product of two factors:

---

*) For very recent developments, cf. [63,64,65].

$$[(2E-S)p + E - S][(2E-S)q + E - S] = E^2.$$

The unknowns p and q may now be found by equating the algebraic factors on the left of this equation to all possible pairs of numerical factors of the known right hand side E^2. Thus, writing $E^2 = E_1E_2$, we find

$$p = (E_1 + S - E)/(2E - S), \quad q = (E_2 + S - E)/(2E - S).$$

There may not be integer solutions of these equations, but if solutions exist where both p and q are primes, then r can be computed from the equation $r = pq + p + q$ and tested for primality. If r is a prime and p, q and r do not divide E, then (Epq,Er) form an amicable pair. For example, take $E = 2$, then $S = \sigma(E) = 3$ and the only possibility for E_1E_2 is $E_1 = 1$, $E_2 = 4$. This gives $p = 2$, $q = 5$, $r = 17$, but since $p = E$, this gives no amicable pair. If $E = 4$, we have $S = 7$, and there are two possibilities for E_1E_2, viz., $1 \cdot 16$ and $2 \cdot 8$. The first gives $p = 4$, but the second gives $p = 5$, $q = 11$, and $r = 71$, hence the amicable pair $(2^2 5 \cdot 11, 2^2 71)$.

In general, when $E = 2^n$, $n \in \mathbb{N}$, we can choose $E_1 = 2^m$, $E_2 = 2^{2n-m}$ for $m = 1,2,\ldots,n-1$. This gives *Euler's rule:*
The numbers $2^n pq$ and $2^n r$ are amicable if the three integers $p = 2^m(2^{n-m}+1)-1$, $q = 2^n(2^{n-m}+1) - 1$ and $r = 2^{n+m}(2^{n-m} + 1)^2 - 1$ are all prime numbers for some positive integer m, satisfying $1 \le m \le n-1$.

The special case $m = n-1$ yields $p = 3 \cdot 2^{n-1} - 1$, $q = 3 \cdot 2^n - 1$, $r = 9 \cdot 2^{2n-1} - 1$, and this rule is due to THABIT-IBN-KURRA [42, pp. 98–99]. It has been checked for all $n \le 20{,}000$ [51,62], but only $n = 2,4$ and 7 actually yield amicable pairs (i.e., the "Greek" pair, and those found by Fermat*) and Descartes). Until recently, only one more pair was known from Euler's rule, viz. in case $n = 8$, $m = 1$ [36,58]. Very recently, TE RIELE [50] investigated Euler's rule for all possible values of n, m, with $1 \le m < n-1$, and $r < 10^{132}$. This search yielded (only) one new pair, viz., for $n = 40$, $m = 29$. This pair consists of two 40-digit numbers. BORHO [7] has generalized the Thabit-ibn-Kurra rule, generating many explicit formulas for amicable numbers. Some of them have produced new amicable pairs. TE RIELE [50] found three very large pairs from these generalized Thabit-ibn-Kurra rules, of 32-, 81- and 152- digit numbers, respectively. Together with the 40-digit pair mentioned above these four pairs are the largest pairs known at present.**) In his "Antrittsvorlesung"

*) But see [62].
**) Recently, BORHO [62] found two pairs with 42- resp. 43-digit numbers.

BORHO [4] gives a nice view into the kitchen where Thabit-like rules for the construction of amicable pairs are prepared.

About 85% of the known amicable pairs are of the "regular" form (EM,EN) discussed above. The second smallest amicable pair (1184,1210) = ($2^5 37$, $2 \cdot 5 \cdot 11^2$), found in 1866 by N. Paganini, a 16-year old Italian schoolboy, is an instance of what are sometimes called "exotic" pairs. These pairs are more difficult to find than regular pairs, since no Thabit-like rules for exotic pairs are known.

Many people after Euler have studied amicable pairs and computed one or more new pairs. The record is held by ELVIN J. LEE with nearly 400 pairs [35]. Like Euler, he used indeterminate analysis methods, but now programmed for computers.

Exhaustive (computer) searches for amicable pairs (n,m), $n < m$, have been conducted on a large scale only after the advent of the electronic computer. Table 3.1 gives some information about these searches.

Table 3.1. Exhaustive amicable pair searches (n,m), $n < m$

range searched, $n \le$	total # pairs in this range	# *new* pairs found in this range	searcher
6232	5	none	DICKSON [19]
15000	7	none	BROWN [10]
10^5	13	1	ROLF [54]
10^6	42	8	ALANEN, ORE & STEMPEL [1]
10^7	108	14	BRATLEY & McKAY [9]
10^8	236	56	COHEN [13]

COSTELLO [15] reports on an exhaustive search through all $n < 10^9$, but he gives no reference nor any detailed information.

The high quality of the many non-exhaustive searches, carried out before these exhaustive searches, is illustrated by the fact that only relatively few *new* pairs were found with the exhaustive searches.

A number of theoretical results concerning amicable pairs are known. For instance, KANOLD [33] proved that an amicable pair cannot be of the form $(n,m) = (p_1^{a_1} p_2^{a_2}, q_1^{b_1} q_2^{b_2})$, and the "Greek" pair ($2^2 5 \cdot 11$, $2^2 71$) shows that this result is the best possible with respect to the number of primes involved. It is not known whether or not there are infinitely many amicable

pairs. ERDÖS proved [20] that the density if the amicable pairs is zero. Very recently, POMERANCE [46,47] proved that the number of amicable numbers (i.e. numbers which belong to an amicable pair) $\leq x$ is bounded by $x \exp\{-(\log x)^{1/3}\}$, for x large enough. BORHO [5,6] has investigated amicable numbers with a given number of prime divisors. One result is that if the number of *different prime divisors* of n is prescribed and also the *number* of *divisors* of m, then only a *finite* number of amicable pairs can satisfy these requirements. Related work has been done by ARTUHOV [2,3].

No amicable pairs (n,m) are known for which n and m are relatively prime. The lesser member of such a pair must be greater than 10^{60} [29].

4. CYCLES OF LENGTH ≥ 3

As mentioned in the introduction, only a few cycles of length ≥ 3 are known. POULET [48] was the first who discovered two of such cycles: one of length 5 and one of length 28. The members of these cycles with their factorizations are given in Table 4.1. Only quite recently, cycles of length four have been found. The first one was *constructed* by BORHO [8]. Thirteen others have been found by exhaustive searches of COHEN [13], DAVID (cf. [18]) and ROOT [55]. In Table 4.2 the members of the cycles with their factorizations are given. The only cycle with the *odd* members is the one constructed by Borho.

Table 4.1. Poulet's 5- and 28-cycles

$(12496 = 2^4 11 \cdot 71,\ 14288 = 2^4 19 \cdot 47,\ 15472 = 2^4 967,\ 14536 = 2^3 23.79,\ 14264 = 2^3 1783)$

$(14316 = 2^2 3 \cdot 1193,\ 19116 = 2^2 3^4 59,\ 31704 = 2^3 3 \cdot 1321,\ 47616 = 2^9 3 \cdot 31,\ 83328 = 2^7 3 \cdot 7 \cdot 31,\ 177792 = 2^7 3 \cdot 463,\ 295488 = 2^6 3^5 19,\ 629072 = 2^4 39317,\ 589786 = 2 \cdot 294893,\ 294896 = 2^4 7 \cdot 2633,\ 358336 = 2^6 11 \cdot 509,\ 418904 = 2^3 52363,\ 366556 = 2^2 91639,\ 274924 = 2^2 13 \cdot 17 \cdot 311,\ 275444 = 2^2 13 \cdot 5297,\ 243760 = 2^4 5 \cdot 11 \cdot 277,\ 376736 = 2^5 61 \cdot 193,\ 381028 = 2^2 95257,\ 285778 = 2 \cdot 43 \cdot 3323,\ 152990 = 2 \cdot 5 \cdot 15299,\ 122410 = 2 \cdot 5 \cdot 12241,\ 97946 = 2 \cdot 48973,\ 48976 = 2^4 3061,\ 45946 = 2 \cdot 22973,\ 22976 = 2^6 359,\ 22744 = 2^3 2843,\ 19916 = 2^2 13 \cdot 383,\ 17716 = 2^2 43 \cdot 103)$

Table 4.2. The 14 4-cycles

$(1264460 = 2^2 5 \cdot 17 \cdot 3719$	$(2115324 = 2^2 3^2 67 \cdot 877$	$(2784580 = 2^2 5 \cdot 29 \cdot 4801$
$1547860 = 2^2 5 \cdot 193 \cdot 401$	$3317740 = 2^2 5 \cdot 165887$	$3265940 = 2^2 5 \cdot 61 \cdot 2677$
$1727636 = 2^2 521 \cdot 829$	$3649556 = 2^2 107 \cdot 8527$	$3707572 = 2^2 11 \cdot 84263$
$1305184 = 2^5 40787)$	$2797612 = 2^2 331 \cdot 2113)$	$3370604 = 2^2 23 \cdot 36637)$
$(4938136 = 2^3 7 \cdot 109 \cdot 809$	$(7169104 = 2^4 17 \cdot 26357$	$(18048976 = 2^4 11 \cdot 102551$
$5753864 = 2^3 23 \cdot 31271$	$7538660 = 2^2 5 \cdot 376933$	$20100368 = 2^4 919 \cdot 1367$
$5504056 = 2^3 17 \cdot 40471$	$8292568 = 2^3 59 \cdot 17569$	$18914992 = 2^4 37 \cdot 89 \cdot 359$
$5423384 = 2^3 53 \cdot 12791)$	$7520432 = 2^4 127 \cdot 3701)$	$19252208 = 2^4 1203263)$
$(18656380 = 2^2 5 \cdot 932819$	$(28158165 = 3^3 5 \cdot 7 \cdot 83 \cdot 359$	$(46722700 = 2^2 5^2 47 \cdot 9941$
$20522060 = 2^2 5 \cdot 13 \cdot 17 \cdot 4643$	$29902635 = 3^3 5 \cdot 7 \cdot 31643$	$56833172 = 2^2 11 \cdot 53 \cdot 24371$
$28630036 = 2^2 19 \cdot 449 \cdot 839$	$30853845 = 3^3 5 \cdot 11 \cdot 79 \cdot 263$	$53718220 = 2^2 5 \cdot 2685911$
$24289964 = 2^2 97 \cdot 62603)$	$29971755 = 3^3 5 \cdot 11 \cdot 20183)$	$59090084 = 2^2 43 \cdot 343547)$
$(81128632 = 2^3 13 \cdot 19 \cdot 41057$	$(174277820 = 2^2 5 \cdot 29 \cdot 487 \cdot 617$	$(209524210 = 2 \cdot 5 \cdot 7 \cdot 19 \cdot 263 \cdot 599$
$91314968 = 2^3 23 \cdot 29 \cdot 109 \cdot 157$	$205718020 = 2^2 5 \cdot 17 \cdot 43 \cdot 14071$	$246667790 = 2 \cdot 5 \cdot 17 \cdot 59 \cdot 24593$
$96389032 = 2^3 41 \cdot 71 \cdot 4139$	$262372988 = 2^2 47 \cdot 107 \cdot 13043$	$231439570 = 2 \cdot 5 \cdot 19 \cdot 23 \cdot 211 \cdot 251$
$91401368 = 2^3 149 \cdot 76679)$	$210967684 = 2^2 23 \cdot 2293127)$	$230143790 = 2 \cdot 5 \cdot 17 \cdot 499 \cdot 2713)$
$(330003580 = 2^2 5 \cdot 16500179$	$(498215416 = 2^3 19 \cdot 47 \cdot 69739$	
$363003980 = 2^2 5 \cdot 18150199$	$506040584 = 2^3 7 \cdot 233 \cdot 38783$	
$399304420 = 2^2 5 \cdot 1163 \cdot 17167$	$583014136 = 2^3 72876767$	
$440004764 = 2^2 110001191)$	$510137384 = 2^3 19 \cdot 797 \cdot 4211)$	

The construction of cycles of length ≥ 3 is very difficult, because the number of variables in the equations for these cycles is large. As far as we know, BORHO [8] is the only one who has given rules for their construction. One such rule for cycles of length 3 is the following: if ν and μ are positive integers satisfying $\mu \geq \nu$ and $2\mu + 1 \equiv 0 \pmod{\nu}$, then the four quantities

$$p = 2^\nu - 1, \quad p_1 = \frac{1}{p}[(2^{\nu+1} - 1)(2^\mu - 1) + 2^{\mu-\nu}(2^{\mu+1} - 1)],$$

$$p_2 = 2^\nu(p_1 + 1) - 1,$$

$$p_3 = 2^\nu(2^{\nu+1} - 1) + 2^{\mu+1} - 1$$

are integers. If they are prime then the numbers

$$2^{\nu}pp_1, \quad 2^{\nu}p_2 \quad \text{and} \quad 2^{\mu}p_3$$

form a cycle of length 3. Borho also gives an analogous rule for 4-cycles which requires the simultaneous primality of 7 numbers!

5. ALIQUOT SEQUENCES

CATALAN [12] has put forward the (what he calls) "empirical theorem" that all aliquot sequences tend to a limit, which equals 1 or a perfect number. He added that if the theorem is true, it would be difficult to prove it. DICKSON [19] has pointed out that aliquot sequences will sometimes run into a cycle of length ≥ 2, and he computed all aliquot sequences with starting term ≤ 1000 as far as he could with the available resources (the factor tables of D.N. Lehmer). Poulet went on with the sequence starting with 138, and he stopped after reaching the term $s^{73}(138)$.

D.H. Lehmer found this sequence to be terminating ($s^{176}(138) = 59$, $s^{177}(138) = 1$) with maximum $s^{117}(138) = 179931895322 = 2 \cdot 61 \cdot 929 \cdot 1587569$.

G.A. PAXSON [43] was probably the first who used an electronic computer to compute aliquot sequences. He pursued all sequences with starting term ≤ 3040 to completion or to a term $> 10^{10}$. After him, many others have computed aliquot sequences. One of the most enthusiastic "aliquotters" is Richard K. Guy. A survey of the present state of affairs can be found in [18] and in [17]. The Lehmers (D.H. and E.) have concentrated on the 6 sequences with starting term < 1000 which are still incompelte at this moment. The smallest is the 276-sequence. They have reached

$$s^{433}(276) = 1071000479624274560488334974030194 24$$
$$= 2^5 3 \cdot 199 \cdot C$$

where C is a number known to be composite.

Nearly all the effort in computing aliquot sequences goes into factorization and the interest in the sequences has certainly stimulated the development of better factorization methods (cf. the report of RICHARD GUY [25]).

Until recently, very few theoretical results were known about aliquot sequences. DICKSON [19] already observed that perfect numbers may "persist"

in aliquot sequences. Indeed, if $2^n p$ is a perfect number, and if $(m,2p) = 1$ with $m > 1$, then $s(2^n pm) = 2^n p(2\sigma(m)-m)$. Since $2\sigma(m)-m$ is odd, the perfect number $2^n p$ persists as long as $p \nmid 2\sigma(m) - m$. Based on these observations TE RIELE [49] constructed an aliquot sequence with at least 5092 (but probably many more) monotonically increasing terms. Soon after hearing this result, LENSTRA, Jr. suggested the falseness of the Catalan-Dickson "empirical theorem" by proving the existence of arbitrarily long monotonically increasing aliquot sequences ([38], cf. also [49]). Building on Lenstra's result, GUY [24] and ERDÖS [21] proved stronger results on increasing aliquot sequences.

In 1975 an interesting paper by GUY & SELFRIDGE [26] appeared, which introduced the concept of a "driver" of an aliquot sequence. Certain heuristic considerations were the basis for this concept: if a divisor d of n with $s(d) > d$ would exist such that d divides all succeding terms of the aliquot sequence of n, then an unbounded sequence would be found. A sufficient condition would be that each prime dividing d would divide $\sigma(d)$ to a higher power, so that, for m such that $(d,m) = 1$, it would divide $s(dm) = \sigma(d)\sigma(m) - dm$ to the same power as it divided d. However, such d unfortunately do not exist [26]. This requirement is somewhat weakened by G & S in the following definition:

a number $g = 2^a v$, $a > 0$ and v odd, is called a *guide* if $v|\sigma(2^a)$; a guide is called a *driver* if, moreover, $2^{a-1}|\sigma(v)$. For a driver, we thus have $2^{a-1}v|s(2^a v)$, and only one more factor 2 will let this driver "persist". Guy and Selfridge then prove that only the drivers are the perfect numbers and the numbers 2, $2^3 3$, $2^3 3 \cdot 5$, $2^5 3 \cdot 7$ and $2^9 3 \cdot 11 \cdot 31$ (related work may be found in [16] and [45]). G & S give some examples of driver dominated, monotonically increasing sequences, but, despite the tenacity of these drivers, none is expected to "live" forever:

				driver
			$628628 = 552:26 = 2^2 7 \cdot 11 \cdot 13 \cdot 157$	$2^2 7$
35149477	396986268	016618686	$344127020 = 552:181 = 2^2 3^2 5 \cdot 7^2 C$	
			$3985\ 297814226 = 564:83 = 2 \cdot 3 \cdot 211 \cdot 3147944561$	$2 \cdot 3$
2422	499075303	417661059	$252663526 = 564:265 = 2 \cdot 3^2 23 \cdot 89 \cdot C$	
			$11400 = 5250:3 = 2^3 3 \cdot 5^2 19$	$2^3 3 \cdot 5$
4	553462993	488753886	$439512520 = 5250:72 = 2^3 3 \cdot 5 \cdot C$	
			$8154 = 8154:0 = 2 \cdot 3^3 151$	$2 \cdot 3$
4615096	670497664	245830510	$= 8154:201 = 2 \cdot 3^6 5 \cdot 43 \cdot C$	
			$1503680 = 8904:13 = 2^6 5 \cdot 37 \cdot 127$	$2^6 \cdot 127$
3200141	507007701	992846912	$= 8904:166 = 2^6 89 \cdot 127 \cdot C$	
			$44144 = 9852:11 = 2^4 31 \cdot 89$	$2^4 31$
5149877	193773848	066488144	$= 9852:146 = 2^4 3 \cdot 11 \cdot 31 \cdot C$	

By 552:26 they mean $2^{26}(552)$. These, and several other heuristic and probabilistic arguments led Guy and Selfridge to the conjecture in [26] that most large even sequences are *unbounded*, almost diametrically against the Catalan-Dickson "empirical theorem". DEVITT [17] has continued the investigation on drivers on the basis of a statistical model. As data for this model he has computed the aliquot sequences of $10^k + 2, 10^k + 4, \ldots, 10^k + 2000$, for $k = 9$, 10, 11 and 12, until completion, or until a term exceeded the bound 10^{18}.

ACKNOWLEDGEMENT

The author is grateful to C. Pomerance for drawing his attention to a number of valuable references.

REFERENCES

[1] ALANEN, J., O.ORE & J. STEMPEL, *Systematic computation on amicable numbers*, Math. Comp., 21 (1967), pp. 242-245.

[2] ARTUHOV, M.M., *On the problem of h-fold perfect numbers*, Acta Arithm., 23 (1973), pp. 249-255.

[3] ARTUHOV, M.M., *On some problems in the theory of amicable numbers*, (Russian), ibidem, 27 (1975), pp. 281-291.

[4] BORHO, W., *Befreundete Zahlen*, Antrittsvorlesung an der Universität Bonn am 9.11.1977, 2. Auflage, Wuppertal 1979.

[5] BORHO, W., *Befreundete Zahlen mit gegebener Primteileranzahl*, Math. Ann., 209 (1974), pp. 183-193.

[6] BORHO, W., *Eine Schranke für befreundete Zahlen mit gegebener Teileranzahl*, Math. Nachr., 63 (1974), pp. 297-301.

[7] BORHO, W., *On Thabit ibn Kurrah's formula for amicable numbers*, Math. Comp., 26 (1972), pp. 571-578.

[8] BORHO, W., *Über die Fixpunkte der k-fach iterierten Teilersummenfunktion*, Mitt. Math. Geselsch. Hamburg, 9 (1969), pp. 34-48.

[9] BRATLEY, P. & J. McKAY, *More amicable numbers*, Math. Comp., 22 (1968), pp. 677-678.

[10] BROWN, B.H., *A new pair of amicable numbers*, Amer. Math. Monthly, 46 (1939), p. 345.

[11] BUXTON, M. & S. ELMORE, *An extension of lower bounds for odd perfect numbers*, Abstract, Notices Amer. Math. Soc., 23 (1976), p. A-55.

[12] CATALAN, E., *Propositions et questions diverses*, Bull. Soc. Math. France, 16 (1887/8), pp. 128-129.

[13] COHEN, H., *On amicable and sociable numbers*, Math. Comp., 24 (1970), pp. 423-429.

[14] CONDICT, J.T., *On an odd perfect number's largest prime divisor*, Senior Thesis, Middleburg College, May, 1978.

[15] COSTELLO, P.J., *Amicable pairs of Euler's first form*, J. Recr. Math., 10 (1977/8), pp. 183-189.

[16] DANDAPAT, G.G., J.L. HUNSUCKER & C. POMERANCE, *Some new results on odd perfect numbers*, Pacific J. Math., 57 (1975), pp. 359-364.

[17] DEVITT, J.S., *Aliquot sequences*, M.Sc. thesis, The Univ. of Calgary, May 1976.

[18] DEVITT, J.S., R.K. GUY & J.L. SELFRIDGE, *Third report on aliquot sequences*, Research paper no. 327, Dept. of Math. and Stat., The Univ. of Calgary, Dec. 1976.

[19] DICKSON, L.E., *Theorems and tables on the sum of the divisors of a number*, Quart. J. Math., 44 (1913), pp. 264-296.

[20] ERDÖS, P., *On amicable numbers*, Publ. Math. Debrecen, 4 (1955), pp. 108-111.

[21] ERDÖS, P., *On asymptotic properties of aliquot sequences*, Math. Comp., 30 (1976), pp. 641-645.

[22] EULER, L., *De numeris amicabilibus*, Leonhardi Euleri Opera Omnia, B.G. TEUBNER, Leipzig and Berlin, Ser. I, vol. 2, 1915, pp. 63-162.

[23] GILLIES, D.B., *Three new Mersenne primes and a statistical theory*, Math. Comp., 18 (1964), pp. 93-97.

[24] GUY, R.K., *Aliquot sequences*, pp. 111-118 in H. Zassenhaus (ed.), *Number theory and algebra*, a collection of papers honoring H. Mann, A.E. Ross and O. Taussky-Todd, New York, etc., Acad. Press, 1977.

[25] GUY, R.K., *How to factor a number*, Congressus Numerantium XII, Proc. 4th Manitoba Conf. Numer. Math., Winnipeg 1974, pp. 387-406.

[26] GUY, R.K. & J.L. SELFRIDGE, *What drives an aliquot sequence?* Math. Comp., 29 (1975), pp. 101-107; Corrigendum in: Math. Comp., 34 (1980), pp. 319-321.

[27] HAGIS Jr., P., *A lower bound for the set of odd perfect numbers*, Math. Comp., 27 (1973), pp. 951-953.

[28] HAGIS Jr., P., *Outline of a proof that every odd perfect number has at least 8 prime factors*, Math. Comp., 35 (1980), pp. 1027-1032.

[29] HAGIS Jr., P., *Lower bounds for relatively prime amicable numbers of opposite parity*, Math. Comp., 24 (1970), pp. 963-968.

[30] HAGIS Jr., P., *On the second largest prime divisor of an odd perfect number*, in: *Analytic number theory*, Proc. of a conference held at Temple University, May 12-15, 1980, to appear.

[31] HAGIS Jr., P. & W.L. McDANIEL, *On the largest prime divisor of an odd perfect number*, II, Math. Comp., 29 (1975), pp. 922-929.

[32] HURWITZ, A., *New Mersenne primes*, Math. Comp., 16 (1962), pp. 249-251.

[33] KANOLD, H.J., *Über befreundete Zahlen*, II, Math. Nachr. 10 (1953), pp. 99-111.

[34] KISHORE, M., *Odd perfect numbers not divisible by 3 are divisible by at least ten distinct primes*, Math. Comp., 31 (1977), pp. 274-279.

[35] LEE, E.J. & J.S. MADACHY, *The history and discovery of amicable numbers*, J. Recr. Math., 5 (1972), Part 1: pp. 77-93, Part 2: pp. 153-173, Part 3: pp. 231-249.

[36] LEGENDRE, A.M., *Théorie des nombres*, Vol. 2, 1830, p. 150 (4th edition, Paris, 1955, Vol. 2, pp. 148-150).

[37] LEHMER, D.H., *An extended theory of Lucas' functions*, Ann. Math., 31 (1930), pp. 419-448.

[38] LENSTRA Jr., H.W., *Problem 6064*, The Amer. Math. Monthly, 82 (1975), p. 1016 (Solution: ibidem, 84 (1977), p. 580).

[39] LUCAS, E.A., *Théorie des fonctions numériques simplement périodiques*, Amer. J. Math., 1 (1878), pp. 184-239.

[40] MATISOO, J., *The superconducting computer*, Scientific American, May 1980, pp. 38-53.

[41] NOLL, C. & L. NICKEL, *The 25th and 26th Mersenne primes*, Math. Comp., 35 (1980), pp. 1387-1390.

[42] ORE, O., *Number theory and its history*, McGraw-Hill, New York, 1948.

[43] PAXSON, G.A., *Aliquot sequences* (prelim. report), Amer. Math. Monthly, 63 (1956), p. 614.

[44] POMERANCE, C., *The second largest prime factor of and odd perfect number*, Math. Comp., 29 (1975), pp. 914-921.

[45] POMERANCE, C., *On multiply perfect numbers with a special property*, Pacific J. Math., 57 (1975), pp. 511-517.

[46] POMERANCE, C., *On the distribution of amicable numbers*, J. reine angew. Math., 293/294 (1977), pp. 217-222.

[47] POMERANCE, C., *On the distribution of amicable numbers, II*, ibidem, 325 (1981), pp. 183-188.

[48] POULET, P., *Question 4865*, l'Intermédiaire des Math., 25 (1918), pp. 100-101.

[49] TE RIELE, H.J.J., *A note on the Catalan-Dickson conjecture*, 27 (1973), pp. 189-192.

[50] TE RIELE, H.J.J., *Four large amicable pairs*, Math. Comp., 28 (1974), pp. 309-312.

[51] RIESEL, H., *Lucasian criteria for the primality of* $N = h.2^n-1$, Math. Comp., 23 (1969), pp. 869-875.

[52] RIESEL, H., *Mersenne numbers*, MTAC, 12 (1958), pp. 207-213.

[53] ROBINSON, R.M., *Mersenne and Fermat numbers*, Proc. Amer. Math. Soc., 5 (1954), pp. 842-846.

[54] ROLF, H.L., *Friendly numbers*, Amer. Math. Monthly, 72 (1965), p. 455.

[55] ROOT, M.I.T., *Artificial Intelligence Memo #239*.

[56] SHANKS, D., *Solved and unsolved problems in number theory*, second edition, Chelsea Publ. Company, New York, N.Y., 1978.

[57] SLOWINSKI, D., *Searching for the 27th Mersenne prime*, J. Recr. Math., 11 (1978/9), pp. 258-261.

[58] TCHEBYCHEFF, P.L., Jour. de Math., 16 (1851), p. 275 (Werke I, p. 90).

[59] TUCKERMAN, B., *Review of Hagis' UMT 53*, Math. Comp., 27 (1973), pp. 1005-1006.

[60] TUCKERMAN, B., *The 24th Mersenne prime*, Proc. Nat. Acad. Sci. USA, 68 (1971), pp. 2319-2320.

[61] UHLER, H.S., *A brief history of the investigations on Mersenne numbers and the latest immense primes*, Scripta Math., 18 (1952), pp. 122-131.

Added in proof:

[62] BORHO, W., *Some large primes and amicable numbers*, Math. Comp. 36 (1981), pp. 303-304.

[63] TE RIELE, H.J.J., *On generating new amicable pairs from given amicable pairs*, Preprint NW Report, Mathematical Centre, Amsterdam, Nov. 1982.

[64] TE RIELE, H.J.J., *Table of 1870 new amicable pairs generated from 1575 mother pairs*, Report NN 27/82, Mathematical Centre, Amsterdam, Oct. 1982.

[65] WOODS, D., *Construction of amicable pairs*, Abstracts of the AMS, v. 3, 1982, p.223.

ON A SECURE PUBLIC-KEY CRYPTOSYSTEM

by

P.J. HOOGENDOORN

1. INTRODUCTION

These days, electronic communication systems proliferate throughout our society. Besides offering speed, accuracy and convenience, they also confront us with serious problems of security and privacy of communication. One way to tackle these problems is to resort to *cryptosystems:* mathematical systems for encrypting, or transforming, information so that it is unintelligible and therefore useless to those who are not meant to have access to it (see [2,3]).

In this paper we will dicuss a secure *public-key cryptosystem,* the RSA (or M.I.T.) system designed by RIVEST, SHAMIR and ADLEMAN [10].

2. PUBLIC-KEY CRYPTOSYSTEM

In a public-key cryptosystem the sender and the receiver (rather than agreeing on a single secret key for the two of them) each generate two distinct keys of their own: an enciphering key, which serves to implement the system's enciphering algorithm, and a deciphering key, which serves to implement the system's deciphering algorithm. Together with the enciphering algorithm each user puts his enciphering key in a public file, whereas he keeps his deciphering key secret. The keys are related in the sense that they serve to implement inverse operations: operating on a ciphertext message first with the transformation specified by the enciphering key and then with the transformation specified by the deciphering key reproduces the message, and in some (but not all) systems applying the transformations in the reverse order also reproduces the message. The system is called *secure* if it is computationally infeasible to derive the decryption key from the encryption key.

Obviously, in a public-key cryptosystem there is no need for a secure channel for the distribution of keys. Moreover, if encryption and decryption can be performed in reverse order the system allows for the construction of an "electronic signature" that guarantees the authenticity of a message and prevents it from being forged by the receiver as well as by a third party.

Formalizing the above concepts we may think of a public-key cryptosystem as a triple $(S,\{E_K\}_{K\in S}, \{D_K\}_{K\in S})$ where S is a finite set (of keys) and $\{E_K\}_{K\in S}$, $\{D_K\}_{K\in S}$ are two families of algorithms representing transformations

$$E_K: M \to M$$

$$D_K: M \to M$$

on a finite set M (of messages), such that

(i) $\forall_S K \forall_M M: D_K(E_K(M)) = M$,

(ii) $\forall_S K \forall_M M: E_K(M)$ and $D_K(M)$ are easy to compute,

(iii) for every $K \in S$, each easily computed algorithm equivalent to D_K is computationally infeasible to derive from E_K.

This means that in fact a public-key cryptosystem is based on what is called a *trap-door one-way function*. A function f is called a one-way function if it is easy to compute f in one direction but practically infeasible to compute it in the other direction. Such a function is called trap-door if it is easy to compute the inverse function f^{-1} once certain trap-door information is known. If in fact f is a bijection then it is called a *trap-door one-way permutation*. In terms of the cryptosystem this amounts to

(iv) $\forall_S K \forall_M M: E_K(D_K(M)) = M$.

The search for trap-door one-way functions on which to base public-key cryptosystems has led to the class of NP problems (see [1,5]), problems for which at present all the algorithms that are known for finding general solutions to them call for rapidly increasing amounts of time, but once a possible solution is found it is easy to check its validity. Clearly, the NP-problems that allow for a trap-door to be built in are of particular interest, and the RSA system is based on such a problem: the problem of factoring a large number.

3. THE RSA SYSTEM

3.1. Encryption and Decryption

First, we assume that all messages are represented as integers between 0 and some large number n_0. (A long message is broken into a series of blocks, and each block is represented as such an integer.) One should think of n_0 as being a 200-digit integer. Every user of the system chooses two large "random" prime numbers p and q such that

$$n = pq \tag{1}$$

is greater than n_0. Let $\lambda = \mathrm{lcm}(p-1,q-1)$. Next, he picks a large "random" integer d such that

$$\gcd(d,\lambda) = 1, \tag{2}$$

and computes the unique integer e satisfying

$$ed \equiv 1 \pmod{\lambda}, \quad 0 < e < \lambda. \tag{3}$$

Then the user makes public the pair (e,n) as his encryption key, whereas he keeps d secret. (Note that we should properly subscript all these integers as n_A, e_A and d_A for user A, as n_B, e_B and d_B for user B, etc.).

Now, if A wants to send a secret message M to B, he looks up B's encryption key (e_B,n_B) and sends him the ciphertext C, where

$$C = E_B(M) \equiv M^{e_B} \pmod{n_B}. \tag{4}$$

Then B decrypts this chiphertext by raising it to the power d_B modulo n_B. So

$$D_B(C) \equiv C^{d_B} \pmod{n_B}. \tag{5}$$

We now prove that the encryption and decryption algorithms specified by (4) and (5), respectively, satisfy the conditions of a public-key cryptosystem.

First we note that neither encryption nor decryption increases the size of a "message"; both M and C are integers in the range 0 to n-1.

Let $\phi(n)$ be the Euler totient function denoting the number of positive integers $\leq n$ which are relatively prime to n. For prime numbers p,

$$\phi(p) = p - 1,$$

and by Euler's generalization of Fermat's theorem we have

$$M^{\phi(p)} \equiv 1 \pmod{p}$$

for all M such that $p \nmid M$. By (3)

$$M^{ed} \equiv M^{k\lambda+1} \pmod{n}$$

for some integer k. Since p-1 divides λ we have, for all M such that $p \nmid M$,

$$M^{k\lambda+1} \equiv M \pmod{p}.$$

Obviously, this congruence also holds when $M \equiv 0 \pmod{p}$, so actually it holds for all M. A similar argument for q yields

$$M^{k\lambda+1} \equiv M \pmod{q}.$$

From these last two congruences it follows that

$$M^{ed} \equiv M \pmod{n}$$

for all M, $0 \leq M < n$. Hence

$$D(E(M)) \equiv (E(M))^d \equiv (M^e)^d \equiv M^{ed} \equiv M \pmod{n}$$

$$E(D(M)) \equiv (D(M))^e \equiv (M^d)^e \equiv M^{ed} \equiv M \pmod{n}$$

for all M, $0 \leq M \leq n_0$. Therefore E and D are inverse permutations (for each user of the system).

Both E(M) and D(M) are easy to compute: computing $M^e \pmod{n}$, using the procedure "exponentiation by repeated squaring and multiplication", requires at most $2 \log_2 e$ multiplications and $2 \log_2 e$ divisions, and the same holds

for decryption (replace e by d). However, more efficient procedures are known (see [6]).

Finally, we have to show that it is computationally infeasible to find d, knowing e (and n) only. In this context RIVEST, SHAMIR and ADLEMAN [10] note that there exist no techniques to *prove* that the system is secure, the only test available is to see whether anyone can think of a way to break it; such a person is called a *cryptanalyst*. We will briefly describe some cryptanalytic approaches, and suggest that the interested reader studies the excellent paper by WILLIAMS and SCHMID [12] for details.

Factoring n

By factoring n the cryptanalyst would be able to compute $\lambda = \mathrm{lcm}(p-1, q-1)$ and thus d. However, factoring a (large) number is known to be much harder than determining whether it is prime or composite. When p and q are chosen such that they are about the same order of magnitude, having not too small a difference, and p-1, p+1, q-1, q+1 have no small prime divisors, there is no factoring algorithm known that can be used to factor n in a reasonable amount of time, i.e. to say, it will take thousands of years even on the fastest special purpose computer.

If a cryptanalyst could compute $\phi(n)$ without factoring n, he could compute d as the multiplicative inverse of e modulo $\phi(n)$. However, it is very unlikely that this approach is easier than factoring n. In fact, n can be factored using $\phi(n)$ as follows:

First, (p+q) is obtained from n and $\phi(n) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1) = n - (p+q) + 1$. Then, if $p > q$, (p-q) is the square root of $(p+q)^2 - 4n$. Finally, p is half the sum of (p+q) and (p-q).

We note that knowledge of d enables us to factor n: knowing d, one can calculate $ed - 1$, which is a multiple of λ; by MILLER [7, Lemma 5], this suffices to factor n, as follows. We have

$$u^{ed-1} \equiv 1 \pmod{n}$$

for all numbers u with $\gcd(u,n) = 1$. Let $ed - 1 = c2^t$, c odd, and consider the sequence

$$u^c, u^{c2}, u^{c2^2}, \ldots, u^{c2^t} \equiv 1 \pmod{n}.$$

If for some number u_0 with $\gcd(u_0,n) = 1$ there exists an integer i_0 with $0 \le i_0 < t$ such that

$$s :\equiv u_0^{c2^{i_0}} \not\equiv \pm 1 \pmod{n}$$

$$s^2 \equiv u_0^{c2^{i_0+1}} \equiv 1 \pmod{n},$$

then the factorization of n is given by $n = \gcd(n,s-1)\cdot\gcd(n,s+1)$. It can be shown that at least half of all $u \in \{1,2,\ldots,n-1\}$ with $\gcd(u,n) = 1$ can be used as u_0.

Decryption by iteration

Right from the start, several authors have tried to discredit the RSA system by proposing a cryptanalytic approach which has become known as "decryption by iteration" (see [4,11,12]). E.g. SIMMONS and NORRIS [11] suggest to successively re-encrypt the ciphertext C until C is again obtained: one sets C_1 to C, and computes $C_{i+1} \equiv C_i^e \pmod{n}$ until $C_{i+1} = C$. Then $C_i = M$. Clearly this method will be practical only if i turns out to be relatively small. We call i the *iteration exponent* of M; so

$$M^{e^i} \equiv M \pmod{n}. \tag{6}$$

This approach can be generalized as follows:

Let P(x) be a polynomial of small degree and let $R_x(C)$ be the residue of C^x modulo n. Then, if

$$P(x) = \sum_{i=0}^{k} a_i x^i$$

the value of

$$C^{P(x)} \equiv \prod_{i=0}^{k} R_x^i(C^{a_i}) \pmod{n}$$

can be determined fairly rapidly. Now, suppose the cryptanalyst has found a polynomial P(x) with vanishing constant term, so $P(x) = xQ(x)$, such that

$$C^{P(e)} \equiv C \equiv M^e \pmod{n},$$

then $C^{Q(e)} \equiv M \pmod{n}$, i.e. C has been decrypted without factoring n - at least, so it seems. In discussing the above idea, Williams and Schmid point out that what the cryptanalyst should really do is evaluate

$$G(P,g) = \gcd(C^{P(g)} - 1, n)$$

for various P(x) and g. Then (see [12], pp. 527-528) C can be decrypted if $G(P,g) = n$. Moreover, if $1 < G(P,g) < n$, which is less unlikely than $G(P,n) = n$, the cryptanalyst has factored n.

In the next section we will describe how one should select the key parameters in order to minimize the chance of success of the cryptanalytic attack proposed by Simmons and Norris. In this respect RIVEST [8,9] makes definite suggestions as to how the prime numbers p and q should be chosen. Moreover, it turns out that for that particular choice the chance of success of the more general decryption by iteration attack as described above is extremely small; for this we refer to [12].

From the above arguments we conclude that it will be computationally infeasible to find d knowing (e,n) only.

3.2. Selection of the key parameters

Rivest et al. suggest to choose large prime numbers p and q such that there exist large prime numbers p', p", q', q" and small integers a, a', b, b' such that

$$p = ap' + 1, \qquad q = bq' + 1,$$
$$p' = a'p'' + 1, \qquad q' = b'q'' + 1.$$

We will show that in this case there is no significant fraction of the messages that have small iteration exponents.

In order to find the iteration exponent of M we must determine:

a) the least positive integer k such that $M^k \equiv 1 \pmod{n}$

b) the least positive integer m such that $e^m \equiv 1 \pmod{k}$.

By (6), m is then the iteration exponent of M.

Let G_n be the multiplicative group of integers modulo n, and for $g \in G_n$ let $o_n(g)$ denote the order of g. We have $|G_n| = \phi(n) = abp'q'$ and $o_n(g) \mid |G_n|$ for all $g \in G_n$. Since G_n is abelian it is the direct product of cyclic prime-power order subgroups including the cyclic groups of p' and q' elements. In particular we have

$$(7) \qquad |\{g \in G_n \mid p'q' \nmid o_n(g)\}| = ab(p'+q'-1).$$

Let M be a random message ($0 \le M < n$). Since the probability that $\gcd(M,n) > 1$ is extremely small, namely $(p+q-1)/n \approx 1/p + 1/q$, we assume that $\gcd(M,n) = 1$, i.e. $M \in G_n$. Then $k = o_n(M)$, and from (7) it follows that the probability that $p'q' \nmid k$ is extremely small, namely $ab(p'+q'-1)/|G_n| \approx 1/p' + 1/q'$. Therefore we assume that $p'q' \mid k$.

Since the encoding exponent is relatively prime to p'q' (cf. (2), (3)), we have $e \in G_{p'q'}$. An argument similar to the one used above to show that $p'q' \mid k$ now applies to show that it is practically certain that $p''q'' \mid o_{p'q'}(e)$.

Since $p'q' \mid k$ we have that $o_{p'q'}(e)$ must divide m, and therefore $p''q'' \mid m$. Thus we conclude that the probability that a ciphertext $C = E(M)$ can be decrypted by successively re-encrypting C a small number of times is vanishingly small.

In addition to Rivest's suggestion at the beginning of this section how to choose p and q, it is recommended to choose them such that there exist large prime numbers p^*, q^* and small integers a^*, b^* such that

$$p = a^*p^* - 1, \quad q = b^*q^* - 1.$$

In this case, neither p+1 nor q+1 is the product of small primes, whence a frequently successful factoring technique based on p+1 (or q+1) being the product of small primes, will fail (see [12]).

3.3. Signatures

Since $D(E(M)) = E(D(M))$ for all $M \in \mathcal{M}$, every message can be "signed" as follows:

Suppose A wants to send the secret message M to B, which B is waiting for. First A decides whether $n_A < n_B$ or $n_B < n_A$. If $n_A < n_B$, A sends to B the ciphertext

$$S = E_B(D_A(M)).$$

Then B computes

$$E_A(D_B(S)) = E_A(D_A(M)) = M.$$

(If $n_B < n_A$, A sends to B the ciphertext $D_A(E_B(M))$, etc.). The reader is invited to verify that in this case B possesses a message-signature pair which is computationally infeasible to forge (by B or a third party), whereas A cannot later deny having sent this message M to B signed "S".

REFERENCES

[1] AHO, A.V., J.E. HOPCROFT & J.D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] DIFFIE, W. & M.E. HELLMAN, *New Directions in Cryptography*, IEEE Trans. on Inf. Th, Vol. IT-22, No. 6, November 1976.

[3] HELLMAN, M.E., *The Mathematics of Public-Key Cryptography*, Scientific American, August 1979.

[4] HERLESTAM, T., *Critical Remarks on Some Public-Key Cryptosystems*, BIT 18 (1978), pp. 493-496.

[5] KARP, R.H., *Reducibility among combinatorial problems*, in: *"Complexity of Computer Computation"* (R.E. Miller & J.W. Thatcher, eds), Plenum Press, New York, 1972, pp. 85-103.

[6] KNUTH, D., *The Art of Computer Programming*, Vol. II, *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass., 1981.

[7] MILLER, G.L., *Riemann's hypothesis and tests for primality*, J. Comput. System Sci. 13 (1976), pp. 300-317.

[8] RIVEST, R.L., *Remarks on a proposed cryptanalytic attack on the M.I.T. Public-Key Cryptosystem*, CRYPTOLOGIA, Vol. 2, Nr. 1, January 1978.

[9] RIVEST, R.L., *Critical remarks on "Critical Remarks on Some Public-Key Cryptosystems"* by T. Herlestam, BIT 19 (1979), pp. 274-275.

[10] RIVEST, R.L., A. SHAMIR & L. ADLEMAN, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol. 21, Nr. 2, February 1978.

[11] SIMMONS, G.J. & M.J. NORRIS, *Preliminary comments on the M.I.T. Public-Key Cryptosystem*, CRYPTOLOGIA, Vol. 1, Nr. 4, October 1977.

[12] WILLIAMS, H.C. & B. SCHMID, *Some Remarks Concerning the M.I.T. Public-Key Cryptosystem*, BIT 19 (1979), pp. 525-538.

FACTORIZATION OF POLYNOMIALS

by

A.K. LENSTRA

1. INTRODUCTION

We give an overview of the most important algorithms for factoring univariate polynomials over finite fields, over the integers, and over algebraic number fields. In the first sections we deal with problems concerning polynomials over finite fields; we present algorithms for square-free decomposition and partial factorization (Section 2), root-finding (Section 3), complete factorization (Section 4), and deciding irreducibility (Section 5). The factorization over a finite field $\mathbb{F}_q$ can be extended to the factorization over a certain ring $W_k(\mathbb{F}_q)$ containing q^k elements, for a given value of k. In Section 6 we describe several algorithms to perform this so-called *lifting* of a factorization. As a result we are able to formulate reasonably efficient algorithms for factoring univariate polynomials over the integers (section 7) and over algebraic number fields (Section 8). Finally, in Section 9, we give a very short description of the algorithms for factoring multivariate polynomials.

Throughout this paper we represent $\mathbb{Z}/p^k\mathbb{Z}$ by $\{-\lfloor(p^k-1)/2\rfloor,\ldots,-1,0,1,\ldots,\lfloor p^k/2\rfloor\}$ for any prime-power p^k, i.e. we use least absolute remainders modulo p^k. Furthermore, let $f = \sum_{i=0}^{n} a_i X^i$ be a polynomial such that $a_n \neq 0$, then the *degree* of f is denoted by $\deg(f) = n$, the *leading coefficient* of f is denoted by $\ell c(f) = a_n$, and f is called *monic* if $\ell c(f) = 1$.

We omit the proofs of most lemma's; they can be found in a standard textbook (for instance [2]) or in the literature referred to.

2. SQUARE-FREE DECOMPOSITION AND PARTIAL FACTORIZATION IN $\mathbb{F}_q[X]$

Let f be a monic polynomial of degree n in $\mathbb{F}_q[X]$, $q = p^\ell$ with p prime. The following three lemmas enable us to find the *square-free decomposition* of f, i.e. polynomials $f_i \in \mathbb{F}_q[X]$, $i = 1,\ldots,k$, such that $f = \prod_{i=1}^{k} f_i^i$, with

$\gcd(f_i,f_j) = 1,\ i \neq j.$

LEMMA 2.1.

(i) *For each* $a \in \mathbb{F}_q$ *there is a unique* $b \in \mathbb{F}_q$ *such that* $b^p = a$,
(ii) *for all* $a,b \in \mathbb{F}_q$ *we have* $(a+b)^p = a^p + b^p$,
(iii) *for each* $g \in \mathbb{F}_q[X]$ *we have* $g(X)^q = g(X^q)$. □

LEMMA 2.2. *If* $f,g \in \mathbb{F}_q[X]$ *are such that* $g^2 | f$, *then* $g | f'$. □

LEMMA 2.3. *Let* $f \in \mathbb{F}_q[X]$, *then*

$$f' = 0 \iff \exists g \in \mathbb{F}_q[X] \text{ such that } f(X) = g(X^p). \quad \square$$

If $f' = 0$ we find a polynomial $g \in \mathbb{F}_q[X]$ such that $f(X) = g(X^p)$ (Lemma 2.3). Using Lemma 2.1 we then write $f(X) = h(X)^p$, and we determine the square-free decomposition of h recursively. If $f' \neq 0$ we observe that the greatest common divisor of f and f' equals

$$\prod_{i=2,\, i\not\equiv 0 \bmod p}^{k} f_i^{i-1} \cdot \prod_{j=1}^{\lfloor k/p \rfloor} f_{jp}^{jp} = g$$

so that $f/g = \prod_{i=2,\, i\not\equiv 0 \bmod p}^{k} f_i$ and $f_1 = f/(g \cdot \gcd(g, f/g))$. Using these relations the square-free decomposition of f can be determined. For a further discussion on polynomial square-free decomposition algorithms see YUN [40].

In view of the square-free decomposition algorithm we now assume that f is *square-free*, i.e. f has no repeated factors. We present a method to obtain a *partial factorization* (or *distinct degree factorization*) of f over $\mathbb{F}_q$, i.e. polynomials $g_i \in \mathbb{F}_q[X]$, $i = 1,\ldots,n$, such that $f = \prod_{i=1}^{n} g_i$, where g_i is the product of all monic irreducible factors of degree i of f. Notice that, from this factorization we can see whether or not f is irreducible over $\mathbb{F}_q$ (see also Section 5). In general, it tells us the number of irreducible factors of f over $\mathbb{F}_q$.

LEMMA 2.4. *The polynomial* $X^{q^k} - X$ *factors over* $\mathbb{F}_q$ *into the product of all monic irreducible polynomials of degrees dividing* k. □

Lemma 2.4 enables us to determine the partial factorization (remember that f is square-free):

$$g_i = \gcd\Big(X^{q^i} - X,\ f \Big/ \prod_{j=1}^{i-1} g_j\Big), \qquad i = 1,\ldots,n.$$

In practice we can apply this method by computing $X^q \bmod f$ and $X^{q^i} \bmod f = (X^{q^{i-1}} \bmod f)^q \bmod f$ for $i = 2,\ldots,\lfloor n/2 \rfloor$. The computation of the partial factorization can therefore be done in $O(\lfloor n/2 \rfloor \cdot \log q)$ polynomial multiplications modulo f. We can improve on the number of polynomial multiplications modulo f as follows.

Let the $n \times n$ matrix Q have as its i-th row the coefficients of $X^{iq} \bmod f$ for $i = 0,\ldots,n-1$. In the sequel we identify polynomials of degree $< n$ with the row vectors formed by their coefficients.

THEOREM 2.1. *For any* $v = \sum_{i=0}^{n-1} v_i X^i \in \mathbb{F}_q[X]$ *we have* $v \cdot Q = v^q \bmod f$.

PROOF. Let $Q = (q_{ij})_{i,j=0}^{n-1}$.

$$v(X)^q \bmod f = v(X^q) \bmod f = \Big(\sum_{i=0}^{n-1} v_i X^{iq}\Big) \bmod f$$

$$= \sum_{i=0}^{n-1} v_i \sum_{k=0}^{n-1} q_{ik} X^k \bmod f$$

$$= \sum_{k=0}^{n-1} \Big(\sum_{i=0}^{n-1} v_i q_{ik}\Big) X^k \bmod f$$

$$= v \cdot Q. \quad \square$$

From Theorem 2.1 we conclude $(X^{q^{i-1}} \bmod f) \cdot Q = X^{q^i} \bmod f$, so that the $\lfloor n/2 \rfloor$ polynomials $X^{q^i} - X \bmod f$ for $i = 1,\ldots,\lfloor n/2 \rfloor$ can be computed in $O(n^3)$ finite field computations, once we have calculated the matrix Q. This can be done in $O(\log q + n - 2)$ polynomial multiplications modulo f; for large q the use of the matrix Q is therefore preferable.

BERLEKAMP [3] gives an alternative way to find the partial factorization. His method makes use of the kernel of the matrix Q-I, where I is the $n \times n$ identity matrix (see also Section 4), and can be extended to determine the complete factorization over $\mathbb{F}_q$. As this method is fairly complicated we will not discuss it here.

From the partial factorization of a square-free polynomial f we can derive the degrees of the factors of f over $\mathbb{F}_q$. If we want to know only these degrees, we can also use an algorithm developed by GUNJI and ARNON [16]; for their algorithm f has not to be square-free. Define σ_i to be the number of distinct irreducible factors of f of degree i, and ν_i as the rank of the kernel of $Q^i - I$, for $i = 1,\ldots,n$. Putting $\sigma = (\sigma_1,\ldots,\sigma_n)^T$, $\nu = (\nu_1,\ldots,\nu_n)^T$ and $A = (\gcd(i,j))_{i,j=1}^n$, Gunji and Arnon prove that $A \cdot \sigma = \nu$, and they give an explicit formula for A^{-1}, so that σ can be computed, given ν. For further details and proofs see [16].

3. ROOT-FINDING IN $\mathbb{F}_q[X]$

Let f be a monic, square-free polynomial of degree $n \geq 1$ in $\mathbb{F}_q[X]$, which splits over $\mathbb{F}_q$ into n linear factors. We present three methods to determine the n distinct roots of f, i.e. $s_1,\ldots,s_n \in \mathbb{F}_q$ such that $f = \Pi_{i=1}^{n} (X-s_i)$.

The first, rather trivial, method is to test for every element $s \in \mathbb{F}_q$ whether or not s is a root of f. For small q this method is certainly very efficient, but it is clear that we need another approach for large q.

If the characteristic of $\mathbb{F}_q$ is small, we can use the following method described by BERLEKAMP [3]. Denote by Tr(X) the *trace-polynomial* $\Sigma_{i=0}^{\ell-1} X^{p^i}$, where $q = p^{\ell}$. Observe that it gives rise to a linear map $\mathbb{F}_q \to \mathbb{Z}/p\mathbb{Z}$.

LEMMA 3.1. *For each* $\gamma \in \mathbb{F}_q$ *we have* $X^q - X = \gamma^{-1} \cdot \Pi_{s \in \mathbb{Z}/p\mathbb{Z}} (Tr(\gamma X)-s)$. □

LEMMA 3.2. *Let* f *and* v_i, $i = 1,\ldots,k$, *be monic polynomials in* $\mathbb{F}_q[X]$ *such that* $\gcd(v_i,v_j) = 1$, $i \neq j$, *and* $f \mid \Pi_{i=1}^{k} v_i$. *Then* $f = \Pi_{i=1}^{k} \gcd(f,v_i)$. □

Let $\beta \in \mathbb{F}_q$ so that $\{\beta^0,\beta^1,\ldots,\beta^{\ell-1}\}$ forms a basis for $\mathbb{F}_q$ over $\mathbb{Z}/p\mathbb{Z}$. Since f splits over $\mathbb{F}_q$ into n different linear factors, we have

$$X^q - X \equiv 0 \bmod f \quad \text{(Lemma 2.4)}.$$

Applying Lemmas 3.1 and 3.2 we find that

$$f = \prod_{s \in \mathbb{Z}/p\mathbb{Z}} \gcd(f,Tr(\beta^j X) - s), \qquad 0 \leq j < \ell.$$

If there exists an integer j', $0 \leq j' < \ell$, such that $Tr(\beta^{j'}X)$ is not congruent to a scalar modulo f, then we can find a non-trivial factorization of f just by computing $\gcd(f,Tr(\beta^{j'}X) - s)$ for every $s \in \mathbb{Z}/p\mathbb{Z}$. The roots of f can be found by applying this method recursively to the non-linear, non-trivial factors of f. Remark that we cannot use the same j' in the recursion.

We now prove that such an integer j' exists. Suppose on the contrary that for all j, $0 \leq j < \ell$, $Tr(\beta^j X)$ is congruent to a scalar modulo f,

$$\exists t_j \in \mathbb{F}_q \text{ such that } Tr(\beta^j X) \equiv t_j \bmod f, \quad 0 \leq j < \ell.$$

Let $s_1,\ldots,s_n$ be the roots of f in $\mathbb{F}_q$, then

$$\mathrm{Tr}(\beta^j s_1) = \ldots = \mathrm{Tr}(\beta^j s_n) = t_j, \quad 0 \le j < \ell,$$

and using Lemma 2.1 we have

$$\mathrm{Tr}(\beta^j (s_i - s_k)) = 0, \quad 0 \le j < \ell, \ 1 \le i < k \le n.$$

We find that for any $p_0,\ldots,p_{\ell-1} \in \mathbb{Z}/p\mathbb{Z}$

$$0 = \sum_{j=0}^{\ell-1} p_j \mathrm{Tr}(\beta^j (s_i - s_k)) = \mathrm{Tr}((\sum_{j=0}^{\ell-1} p_j \beta^j)(s_i - s_k)) \quad \text{(Lemma 2.1)}.$$

We conclude that $\mathrm{Tr}(s) = 0$, $\forall s \in \mathbb{F}_q$, because $\{\beta^0,\beta,\ldots,\beta^{\ell-1}\}$ forms a basis for $\mathbb{F}_q$ over $\mathbb{Z}/p\mathbb{Z}$ and $s_i \neq s_k$. Since $\mathrm{Tr}(X)$ has degree $p^{\ell-1}$ this is a contradiction.

Clearly this method is not very efficient for finite fields with a large characteristic; it takes at most p gcd-computations in $\mathbb{F}_q[X]$ for each trace-polynomial, and therefore, in the worst case $\Omega(p\ell)$ gcd-computations. For this reason we describe a third root-finding algorithm from RABIN [30] for which no a priori upper bound on the number of computation steps can be given, but which in practice will run quite fast.

We assume first that the characteristic p is odd. Then we know that for every non-zero $s \in \mathbb{F}_q$ either $s^r = 1$ or $s^r = -1$, where $r = (q-1)/2$. We say that non-zero elements s_1 and s_2 in $\mathbb{F}_q$ are of *different types* if $s_1^r \neq s_2^r$.

LEMMA 3.3. *Let s_1 and s_2 be two non-zero, unequal elements in $\mathbb{F}_q$, then*

$$\#\{s \in \mathbb{F}_q \mid 0 \neq (s_1+s)^r \neq (s_2+s)^r \neq 0\} = r.$$

PROOF. If $s_i + s \neq 0$ then either $(s_i + s)^r = 1$ or $(s_i + s)^r = -1$, $i = 1,2$. Therefore

$$0 \neq (s_1 + s)^r \neq (s_2 + s)^r \neq 0 \iff \left(\frac{s_1+s}{s_2+s}\right)^r = -1.$$

The equation $X^r + 1 = 0$ has exactly r distinct roots in $\mathbb{F}_q$ and every root t gives a unique $s \in \mathbb{F}_q$ because $t \neq 1$: $s = (s_1 - ts_2)/(t-1)$. □

The idea of Rabin's root-finding algorithm is as follows. Let s_1 and s_2 be two unknown, unequal, non-zero roots of f. We can try and separate the factors $(X-s_1)$ and $(X-s_2)$ of f by computing $\gcd(f, X^r - 1)$. This succeeds if and only if s_1 and s_2 are of different types. In the other case we select

at random an element $s \in \mathbb{F}_q$ and replace $f(X)$ by $f(X-s)$. According to Lemma 3.3 the roots $s_1 + s$ and $s_2 + s$ of $f(X-s)$ have a probability of at least 1/2 to be of different types. If the shift s is "lucky", that means $s_1 + s$ and $s_2 + s$ are of different types, we can separate s_1 and s_2 by computing $\gcd(f(X-s), X^r-1)$, otherwise we select another $s \in \mathbb{F}_q$ and try again.

In the case that $p = 2$, we replace the polynomial $X^r - 1$ in the above algorithm by the trace-polynomial $Tr(X)$. Again we have a probability of at least 1/2 to find a non-trivial factor of f by computing $\gcd(f(s \cdot X), Tr(X))$, where s is a randomly chosen element of $\mathbb{F}_q$. This follows from $X^q - X = Tr(X) \cdot (Tr(X) - 1)$ (Lemma 3.1) and from a reasoning similar to the proof of Lemma 3.3. Notice that at most ℓ gcd-computations in $\mathbb{F}_q[X]$ are necessary, if we choose s to run through a basis of $\mathbb{F}_q$ over $\mathbb{Z}/2\mathbb{Z}$.

Another way to deal with the case $p = 2$ is as follows. If $\ell > 1$ is even (i.e. $q \equiv 1 \bmod 3$) then we know that $\mathbb{F}_q$ contains a primitive third root of unity ζ (so $\zeta^2 + \zeta + 1 = 0$), and that for every non-zero $s \in \mathbb{F}_q$ either $s^r = 1$ or $s^r = \zeta$ or $s^r = \zeta^2 = 1 + \zeta$, where $r = (q-1)/3$. Now we have a probability of at least 2/3 to factorize f by computing $\gcd(f(X-s), X^r-1)$ and $\gcd(f(X-s), X^r - \zeta)$, for a randomly chosen $s \in \mathbb{F}_q$. Remark that ζ can be determined by looking for a root of $X^2 + X + 1$ in $\mathbb{F}_q$. If ζ has not been determined, we only compute $\gcd(f(X-s), X^r-1)$, but then we have a smaller probability of success. If ℓ is odd ($q \equiv 2 \bmod 3$) we factorize f over the quadratic extension field $\mathbb{F}_{q^2}$ of $\mathbb{F}_q$ using the above method ($q^2 \equiv 1 \bmod 3$). Since f has only linear factors over $\mathbb{F}_q$, the factors over $\mathbb{F}_{q^2}$ are also factors over $\mathbb{F}_q$.

Rabin's method is called a probabilistic method. Remark however that there is not even the slightest probability that the outcome is false; there is only a small probability that the algorithm will run for a very long time.

A root-finding algorithm similar to Rabin's is presented at the end of Section 4.

4. FACTORIZATION IN $\mathbb{F}_q[X]$

Let f be a monic, square-free polynomial of degree n in $\mathbb{F}_q[X]$. In this section we describe several methods to factorize f completely over $\mathbb{F}_q$. First we present the well-known *Berlekamp's factorization algorithm* [2,3] for a small finite field $\mathbb{F}_q$. The second method is an adaptation of Berlekamp's factorization algorithm to large finite fields using the root-finding algorithms from Section 3. Next we give a more recent algorithm

by CANTOR and ZASSENHAUS [9], and finally we discuss the case that f has equal degree factors.

The matrix Q, introduced in Section 2, appears to be very useful here, and in particular the kernel of the matrix Q - I will be of vital importance, where I is the n×n identity matrix.

THEOREM 4.1. *Let* $v \in \mathbb{F}_q[X]$, $\deg(v) < n$, *then*

$$v^q \equiv v \bmod f \iff v \cdot Q = v.$$

PROOF. Use Theorem 2.1. □

Suppose we have a polynomial $v \in \mathbb{F}_q[X]$, $0 < \deg(v) < n$, such that $v \cdot Q = v$, i.e. v is an element of the kernel of Q - I. Theorem 4.1 gives

$$f \mid v^q - v$$

and with Lemma 2.4

$$f \mid \prod_{s \in \mathbb{F}_q} (v-s).$$

We now apply Lemma 3.2 and we find

$$f = \prod_{s \in \mathbb{F}_q} \gcd(f, v-s).$$

This equation gives a non-trivial factorization of f, because $0 < \deg(v) < n$, and we see that such a polynomial $v \in \mathbb{F}_q[X]$ cannot exist if f is irreducible over $\mathbb{F}_q$. But, if f is reducible over $\mathbb{F}_q$, can we find a non-trivial v in the kernel of Q - I to factorize f in this way, and if so, how do we know whether we have found the complete factorization of f over $\mathbb{F}_q$? The next two theorems answer these questions.

THEOREM 4.2. *The rank of the kernel of* Q - I *equals the number of irreducible factors of* f *over* $\mathbb{F}_q$.

PROOF. Let $f = \prod_{i=1}^{r} g_i$ be the complete factorization of f over $\mathbb{F}_q$, with g_i pairwise relatively prime, i = 1,...,r, because f is square-free. Using the Chinese Remainder Theorem, we find that

$$\mathbb{F}_q[X]/(f) \cong \prod_{i=1}^{r} (\mathbb{F}_q[X]/(g_i)),$$

where the isomorphism is defined by

$$v \to (v \bmod g_i)_{i=1}^{r}, \quad v \in \mathbb{F}_q[X]/(f).$$

Let

$$t \to (t_i)_{i=1}^{r}, \quad t \in \mathbb{F}_q[X]/(f),$$

then

$$t^q = t \iff t_i^q = t_i, \quad i = 1,\ldots,r.$$

Since $\mathbb{F}_q[X]/(g_i)$ is a field, we have

$$t_i^q = t_i \iff t_i \in \mathbb{F}_q,$$

for $i = 1,\ldots,r$, and therefore

$$t \in \operatorname{Ker}(Q-I) \iff (t_i)_{i=1}^{r} \in (\mathbb{F}_q)^r.$$

We conclude that the rank of the kernel of $Q-I$ equals r, the number of irreducible factors of f over $\mathbb{F}_q$. □

THEOREM 4.3. *Let* f *and* g_i, $i = 1,\ldots,r$, *be as in Theorem* 4.2. *Let* $\{v_1,\ldots,v_r\}$ *form a basis of the kernel of* $Q-I$. *For every* $j \neq j'$, $1 \le j < j' \le r$, *there exist* k, $1 \le k \le r$, *and* $s \in \mathbb{F}_q$ *such that* $g_j \mid \gcd(f, v_k - s)$ *and* $g_{j'} \nmid \gcd(f, v_k - s)$.

PROOF. It is clear from the proof of Theorem 4.2 that we can find an element t in the kernel of $Q-I$ such that

$$t_j \neq t_{j'}, \quad \text{where } t \to (t_i)_{i=1}^{r}.$$

Therefore there exists an integer k, $1 \le k \le r$, such that

$$v_k \bmod g_j \neq v_k \bmod g_{j'}.$$

Let $v_k \bmod g_j = s \in \mathbb{F}_q$, then

$$g_j \mid v_k - s \quad \text{and} \quad g_j \nmid v_k - s. \quad \square$$

From these theorems it will be clear how we can factorize f completely over $\mathbb{F}_q$.

ALGORITHM 4.1. (Berlekamp's factorization algorithm). For $f \in \mathbb{F}_q[X]$, this algorithm computes the irreducible factors of f over $\mathbb{F}_q$.

(1) Determine $\{v_1,\ldots,v_r\}$, a basis of the kernel of $Q - I$, by diagonalizing $Q - I$. Take $v_1 = 1$ and $0 < \deg(v_i) < n$, $i = 2,\ldots,r$.

(2) If $r = 1$ then f is irreducible.

(3) Compute $\gcd(f, v_2 - s)$ for all $s \in \mathbb{F}_q$. Since $f = \prod_{s \in \mathbb{F}_q} \gcd(f, v_2 - s)$, and because $0 < \deg(v_2) < n$, this gives a non-trivial factorization of f. If we find r factors of f in this way we are done. In the other case we compute $\gcd(g, v_k - s)$ for all $s \in \mathbb{F}_q$, for all factors g of f discovered so far and $k = 3,\ldots,r$. Theorem 4.3 guarantees that we find all factors of f in this way.

McELIECE [22] describes an easy to program algorithm to determine a set of polynomials like $\{v_1,\ldots,v_r\}$ which separates the factors of f, but which can be computed without matrix-diagonalization (this set consists of polynomials of the form $h_i = X^i + X^{iq} + X^{iq^2} + \ldots + X^{iq^s} \bmod f$, for several values of i and s, so that $h_i^q \equiv h_i \bmod f$). A drawback of this method is that the running time strongly depends on the degrees of the factors of f, whereas Berlekamp's factorization algorithm only depends on n, q and the number of factors. On small computers however, it might be preferable to use this algorithm for factorization of large degree polynomials because there is no need for storage and diagonalization of the matrix $Q - I$. For a description of McELIECE's algorithm see [22].

The number of computation steps required for Berlekamp's factorization algorithm is $O(n^3)$ computations in $\mathbb{F}_q$ for step (1), and at most q gcd-computations in $\mathbb{F}_q[X]$ for every v_i in the basis of the kernel of $Q - I$ in step (3). If q is small this is an efficient algorithm, but if q is large it becomes rather impractical. In step (3) we compute $\gcd(f, v_2 - s)$ for all $s \in \mathbb{F}_q$, and at most r of these gcd's will not be trivial, so at least $q-r$ of the gcd-computations will be completely useless.

One way to deal with this problem is to use the root-finding algorithms from Section 3. For $v \in \mathbb{F}_q[X]$ in the kernel of $Q - I$ we define $S_v \subset \mathbb{F}_q$ as

follows:

$$S_v = \{s \in \mathbb{F}_q \mid \gcd(f,v-s) \neq 1\}.$$

It is clear that an efficient algorithm to determine S_v for a given v will give us an adaptation of Berlekamp's factorization algorithm to large finite fields. We know that

$$f = \prod_{s \in S_v} \gcd(f,v-s),$$

and therefore

$$f \mid \prod_{s \in S_v} (v-s).$$

So

$$f \mid H(v)$$

where

$$H(X) = \prod_{s \in S_v} (X-s)$$

Conversely, let $f \mid G(v)$ with $G \in \mathbb{F}_q[X]$. Substituting a common zero of f and $v - s$ for X, we then find that $G(s) = 0$, for all $s \in S_v$, so H divides G. This proves that H is the polynomial of minimal degree in $\mathbb{F}_q[X]$ for which $f \mid H(v)$. This enables us to compute H by looking for the first linear relation between the powers $v^0, v^1, v^2, \ldots$ considered modulo f. Since H has degree $\#S_v$ and $\#S_v \leq r$ we do not have to look beyond the r-th power of v. Once we have H we can determine S_v by the methods from Section 3.

The third method comes from CANTOR and ZASSENHAUS [9]. Their algorithm makes use of a similar observation as Rabin's probabilistic root-finding algorithm from Section 3. Let v be an arbitrary non-zero polynomial in the kernel of $Q - I$, i.e. v is a randomly chosen non-zero linear combination over $\mathbb{F}_q$ of the basis elements $v_1, \ldots, v_r$ of the kernel of $Q - I$. We have seen in the proof of Theorem 4.2 that

$$v \bmod g_i = t_i \in \mathbb{F}_q \quad \text{for } i = 1, \ldots, r,$$

where $f = \Pi_{i=1}^{r} g_i$ is the complete factorization of f over $\mathbb{F}_q$.

Again we assume first that q is odd, so that

$$f \mid v^q - v = v \cdot (v^{(q-1)/2} - 1) \cdot (v^{(q-1)/2} + 1).$$

Clearly $\gcd(f, v^{(q-1)/2} \pm 1) = f$ implies $\gcd(f, v^{(q-1)/2} \mp 1) = 1$ and therefore the probability that $\gcd(f, v^{(q-1)/2} - 1)$ or $\gcd(f, v^{(q-1)/2} + 1)$ will be a non-trivial factor of f equals

$$1 - \text{Prob}(\gcd(f, v^{(q-1)/2} - 1) = f) - \text{Prob}(\gcd(f, v^{(q-1)/2} + 1) = f)$$

(remark that we cannot have that both $\gcd(f, v^{(q-1)/2} - 1) = 1$ and $\gcd(f, v^{(q-1)/2} + 1) = 1$, since degree(v) < n). Now $\gcd(f, v^{(q-1)/2} - 1) = f$ if and only if $g_i \mid v^{(q-1)/2} - 1$ for $i = 1, \ldots, r$, and this is the case if and only if $t_i^{(q-1)/2} = 1$ for $i = 1, \ldots, r$. The polynomial v was randomly chosen in the kernel of Q - I, so that the probability that this occurs is $((q-1)/2q)^r$. The same reasoning holds for $\text{Prob}(\gcd(f, v^{(q-1)/2} + 1) = f)$, and therefore the probability that a non-trivial factor of f will be obtained by computing $\gcd(f, v^{(q-1)/2} - 1)$ and $\gcd(f, v^{(q-1)/2} + 1)$ equals

$$1 - 2(\frac{q-1}{2q})^r > 1 - 2(\frac{1}{2})^r \geq \frac{1}{2}$$

for all $r \geq 2$.

In the case that $p = 2$ and $q \equiv 1 \bmod 3$ we compute $\gcd(f, v^{(q-1)/3} - 1)$, $\gcd(f, v^{(q-1)/3} - \zeta)$ and $\gcd(f, v^{(q-1)/3} - 1 - \zeta)$, where ζ is a primitive third root of unity in $\mathbb{F}_q$. If $q \equiv 2 \bmod 3$ Cantor and Zassenhaus suggest to apply the method just decsribed to find the factorization of f over the quadratic extension field $\mathbb{F}_{q^2}$ of $\mathbb{F}_q$; the factorization of f over $\mathbb{F}_q$ follows by combining factors which are conjugate over $\mathbb{F}_q$. A much easier way to deal with the case $p = 2$, is to use the trace-polynomial (cf. Section 3). Let v be a randomly chosen polynomial in the kernel of Q - I, and let $\text{Tr}(v) = \Sigma_{i=0}^{\ell-1} v^{2^i}$, then $f \mid v^q - v = \text{Tr}(v) \cdot (\text{Tr}(v) - 1)$, so that $f = \gcd(f, \text{Tr}(v)) \cdot \gcd(f, \text{Tr}(v) - 1)$. It is straightforward to prove that we have a probability $1 - 2^{1-r} \geq \frac{1}{2}$ $(r \geq 2)$ to find a non-trivial factor of f by computing $\gcd(f, \text{Tr}(v))$.

In the case that all factors of f have equal degree it is not necessary to compute a basis of the kernel of Q - I to find the complete factorization. Cantor and Zassenhaus suggest the following algorithm. Let $s = n/r$ be the degree of the factors of f, and let $v \neq 0$ be an arbitrary polynomial of

degree < n over $\mathbb{F}_q$. Then clearly $X^{q^s} - X | v^{q^s} - v$, so that, if q is odd

$$f = \gcd(f,v)\cdot\gcd(f,v^{(q^s-1)/2} - 1)\cdot\gcd(f,v^{(q^s-1)/2} + 1).$$

The probability that a non-trivial factor of f will be obtained by computing these gcd's is $> 1 - 2^{1-r} \geq \frac{1}{2}$, as is easily verified [9]. We have seen above how to extend this algorithm to the case that q is even. Remark that for s = 1 we obtain another root-finding algorithm.

Another, not very easy to program method comes from RABIN [30]. All factors of f have degree s, and therefore f has its roots in $\mathbb{F}_{q^s}$. Using a root-finding algorithm from Section 3, we determine a root γ of f in $\mathbb{F}_{q^s}$. Then there exists an irreducible factor g of degree s of f over $\mathbb{F}_q$ satisfying $g(\gamma) = 0$. But also $g(\gamma)^q = g(\gamma^q) = 0$, so the roots of g in $\mathbb{F}_{q^s}$ are $\gamma, \gamma^q, \ldots \gamma^{q^{s-1}}$, and $g = \prod_{i=0}^{s-1} (X - \gamma^{q^i}) \in \mathbb{F}_q[X]$.

Several improvements on the algorithms given in this section are possible for special choices of q. See for instance CANTOR and ZASSENHAUS [9] and MOENCK [25] who gives also a detailed exposition of Berlekamp's factorization algorithm.

5. IRREDUCIBILITY IN $\mathbb{F}_q[X]$

In this section we discuss the problem of deciding whether a monic polynomial f of degree n > 1 is irreducible over $\mathbb{F}_q$. The polynomial f is square-free if and only if gcd(f,f') = 1, which can be easily verified. None of the irreducibility tests below require f to be square-free, but in the average it appears to be time saving to apply the square-free test first.

Our first irreducibility test follows directly from Lemma 2.4, as we noticed already in Section 2.

TEST 1. f is irreducible if and only if

$$\gcd(f, X^{q^i} - X) = 1, \quad \text{for } i = 1, \ldots, \lfloor n/2 \rfloor.$$

If f is reducible this test is in general quite fast; for irreducible f it becomes rather inefficient, because of the large number of gcd-computations in $\mathbb{F}_q[X]$. We can improve on this number of gcd-computations if we use the following test, which is a lemma from RABIN [30].

TEST 2. f is irreducible if and only if

(i) $f \mid X^{q^n} - X$,

(ii) $\gcd(f, X^{q^{n_i}} - X) = 1$, for all $n_i = n/k_i$, where k_i are the prime divisors of n.

Surprisingly, this test appears to be much slower than Test 1, due to the computation of Xq^n modulo f. This polynomial has to be computed if f is irreducible and in the case that the degrees of the factors of f do not divide n.

Theorem 4.2 provides us with a third method.

TEST 3. f is irreducible if and only if the rank of the kernel of Q - I equals one.

It depends on the size (and factorization) of n and q which test should be used. In the first two methods the composition of Xq^k modulo f for several values of k is the most time consuming part; the diagonalization of the matrix Q-I dominates the costs of Test 3. CALMET and LOOS [8] compared the asymptotic behaviour of Tests 2 and 3, and they show that the third test is slightly preferable. They also compared the practical behaviour of these two tests in 'Rabin's probabilistic algorithm for generating irreducible polynomials over $\mathbb{Z}/p\mathbb{Z}$':

(1) Generate a monic, random polynomial f of degree n over $\mathbb{Z}/p\mathbb{Z}$.

(2) If f is irreducible then we are ready, else go back to step (1).

They used this algorithm to generate a number of small degree irreducible polynomials and they concluded that Test 2 is slower than the test based on Theorem 4.2.

Calmet and Loos didn't pay any attention to Test 1 (with or without use of the matrix Q). To generate large degree irreducible polynomials this test (without Q) is however considerably faster than Test 3. For small degrees (and large q) the use of Q is advisable; in that case Test 1 and Test 3 are approximately equally fast. The reason is of course that in the average case one needs n choices in the above algorithm; only the last choice is irreducible. Most of the wrong (reducible) choices have a small degree factor, and this will be detected very fast by the first irreducibility test. This more than compensates for the indeed rather slow behaviour at the last choice.

A reasonable advice for large degrees therefore seems to be : use Test 1 if you expect the polynomial to be reducible, otherwise use Test 3 (and never use Test 2, certainly not if the degree n is prime).

6. FACTORIZATION IN $(W_k(\mathbb{F}_q))[X]$

In the previous sections we have seen several algorithms to factorize a polynomial over a finite field $\mathbb{F}_q$, where $q = p^\ell$. In Sections 7 and 8 it appears to be very useful to be able to release the 'modulo p' restriction of such a factorization over $\mathbb{F}_q$ to a 'modulo p^k' restriction. That is, we want to extend or *lift* a factorization over $\mathbb{F}_q$ to a factorization over a ring containing q^k elements, for a given value of $k \geq 1$. We define this ring, the so-called *truncated Witt-ring* $W_k(\mathbb{F}_q)$, as follows. To represent the elements of $\mathbb{F}_q$ we usually take a monic polynomial $G \in \mathbb{Z}[T]$ of degree ℓ, that is irreducible modulo p, so that

$$\mathbb{F}_q = \{\sum_{i=0}^{\ell-1} a_i\beta^i \mid a_i \in \mathbb{Z}/p\mathbb{Z},\ i = 0,\ldots,\ell-1\},$$

where β is a zero of $(G \bmod p)$. Then

$$W_k(\mathbb{F}_q) = \{\sum_{i=0}^{\ell-1} a_i\beta^i \mid a_i \in \mathbb{Z}/p^k\mathbb{Z},\ i = 0,\ldots,\ell-1\},$$

where β now denotes a zero of $(G \bmod p^k)$. So $W_k(\mathbb{F}_q)$ is indeed a ring with q^k elements. Remark that $W_1(\mathbb{F}_q) = \mathbb{F}_q$, $W_k(\mathbb{Z}/p\mathbb{Z}) = \mathbb{Z}/p^k\mathbb{Z}$ and that $W_k(\mathbb{F}_q)$ like $\mathbb{F}_q$ is independent of the choice of G.

Now let f be a polynomial of degree n in $(W_k(\mathbb{F}_q))[X]$ such that $\ell c(f) \neq 0$ in $\mathbb{F}_q$. Let g_1 and h_1 be polynomials in $\mathbb{F}_q[X]$ such that $f \equiv g_1 \cdot h_1$ over $\mathbb{F}_q$ and $\gcd(g_1,h_1) = 1$. In this section we present two methods to lift this factorization over $\mathbb{F}_q$ to the unique factorization over $W_k(\mathbb{F}_q)$, i.e. $g_k,h_k \in (W_k(\mathbb{F}_q))[X]$ such that $f = g_k \cdot h_k$ over $W_k(\mathbb{F}_q)$, and such that $g_k \equiv g_1$ and $h_k \equiv h_1$ over $\mathbb{F}_q$.

We need the following two algorithms; the first algorithm is the well-known extended Euclidean algorithm in $\mathbb{F}_q[X]$, the second algorithm solves the polynomial equation $\tilde{a} \cdot g_j + \tilde{b} \cdot h_j = c$ for given g_j, h_j and c in $(W_j(\mathbb{F}_q))[X]$.

ALGORITHM 6.1. (Extended Euclidean Algorithm in $\mathbb{F}_q[X]$). Given $g,h \in \mathbb{F}_q[X]$, this algorithm computes unique (up to units) $a,b,d \in \mathbb{F}_q[X]$ such that $a \cdot g + b \cdot h = d = \gcd(g,h)$ over $\mathbb{F}_q$, $\deg(a) < \deg(h) - \deg(d)$ and $\deg(b) < \deg(g) - \deg(d)$. A description of this algorithm can be found in KNUTH [18].

ALGORITHM 6.2. Given $a, b, c, g_j, h_j \in (W_j(\mathbb{F}_q))[X]$, such that $a \cdot g_j + b \cdot h_j = 1$ over $W_j(\mathbb{F}_q)$. The algorithm computes $\tilde{a}, \tilde{b} \in (W_j(\mathbb{F}_q))[X]$ such that $\tilde{a} \cdot g_j + b \cdot h_j = c$ over $W_j(\mathbb{F}_q)$ and $\deg(a) < \deg(h_j)$ as follows.

Compute $\tilde{a}, s \in (W_j(\mathbb{F}_q))[X]$ such that $a \cdot c = s \cdot h_j + \tilde{a}$ (with $\deg(\tilde{a}) < \deg(h_j)$), and take $\tilde{b} = b \cdot c + s \cdot g_j$. It is trivial to verify that $\tilde{a}$ and $\tilde{b}$ satisfy $\tilde{a} \cdot g_j + \tilde{b} \cdot h_j = c$.

The first lift-algorithm follows from the proof of the following lemma.

LEMMA 6.1. (Hensel). *Let* f, g_1, *and* h_1 *be as above. Then there exist* $g_k, h_k \in (W_k(\mathbb{F}_q))[X]$ *such that* $f = g_k \cdot h_k$ *over* $W_k(\mathbb{F}_q)$, $\deg(h_k) = \deg(h_1)$ *and* $g_k \equiv g_1$, $h_k \equiv h_1$ *over* $\mathbb{F}_q$.

PROOF. Using Algorithm 6.1 we determine $a, b \in \mathbb{F}_q[X]$ such that $a \cdot g_1 + b \cdot h_1 = 1$. It is sufficient to show how to construct $g_j, h_j \in (W_j(\mathbb{F}_q))[X]$ for $j = 2, \ldots, k$, such that $f = g_j \cdot h_j$ over $W_j(\mathbb{F}_q)$ and $g_j \equiv g_1$, $h_j \equiv h_1$ over $\mathbb{F}_q$. Suppose that g_j and h_j exist for some $j \geq 1$. Let $c \in \mathbb{F}_q[X]$ such that $f - g_j \cdot h_j = p^j \cdot c$ over $W_{j+1}(\mathbb{F}_q)$.

Compute $a, b \in \mathbb{F}_q[X]$ such that $a \cdot g_1 + b \cdot h_1 = c$ over $\mathbb{F}_q$, using Algorithm 6.2. Define $g_{j+1} = g_j + p^j \cdot \tilde{b}$ and $h_{j+1} = h_j + p^j \cdot \tilde{a}$ (notice that $\deg(h_{j+1}) = \deg(h_j)$ and that $\ell c(h_{j+1}) = \ell c(h_j)$). Now clearly $g_{j+1}, h_{j+1} \in (W_{j+1}(\mathbb{F}_q))[X]$, $g_{j+1} \equiv g_1$ and $h_{j+1} \equiv h_1$ over $\mathbb{F}_q$ and

$$\begin{aligned} g_{j+1} \cdot h_{j+1} &= (g_j + p^j \cdot \tilde{b}) \cdot (h_j + p^j \cdot \tilde{a}) \\ &= g_j \cdot h_j + p^j \cdot (g_j \cdot \tilde{a} + h_j \cdot \tilde{b}) \\ &= g_j \cdot h_j + p^j \cdot c \\ &= f \text{ over } W_{j+1}(\mathbb{F}_q). \quad \square \end{aligned}$$

It is clear that the proof of this lemma provides us with a method to lift a factorization step by step, i.e. from $\mathbb{F}_q = W_1(\mathbb{F}_q)$ to $W_2(\mathbb{F}_q)$, $W_3(\mathbb{F}_q), \ldots, W_k(\mathbb{F}_q)$. This is therefore called the linear lift-algorithm. In practice it frequently occurs that there are more than two factors to be lifted. It is of course possible to apply the above method to each factor and cofactor, but if f is monic it is possible to modify the algorithm so that an arbitrary number of relatively prime factors can be lifted simultaneously, without use of the cofactors [38].

The second method, based on a paper by ZASSENHAUS [41], is called *quadratic-lift algorithm* because it extends a factorization 'in one step' from $W_j(\mathbb{F}_q)$ to $W_{2j}(\mathbb{F}_q)$. We first present the best-known formulation of this algorithm, as it was described for instance in MUSSER [27] and YUN [39].

LEMMA 6.2. (Zassenhaus). *Let* $f \in (W_{2j}(\mathbb{F}_q))[X]$ *such that* $\ell c(f) \neq 0$ *in* $\mathbb{F}_q$. *Let* $g_j, h_j, a_j, b_j \in (W_j(\mathbb{F}_q))[X]$ *such that* $f \equiv g_j \cdot h_j$ *over* $W_j(\mathbb{F}_q)$ *and* $a_j \cdot g_j + b_j \cdot h_j = 1$ *over* $W_j(\mathbb{F}_q)$. *Then we can find* $g_{2j}, h_{2j}, a_{2j}, b_{2j} \in (W_{2j}(\mathbb{F}_q))[X]$ *in a fixed number of computation steps, such that* $f = g_{2j} \cdot h_{2j}$ *over* $W_{2j}(\mathbb{F}_q)$, $a_{2j} \cdot g_{2j} + b_{2j} \cdot h_{2j} = 1$ *over* $W_{2j}(\mathbb{F}_q)$ *and* $g_{2j} \equiv g_j$, $h_{2j} \equiv h_j$ *over* $W_j(\mathbb{F}_q)$.

PROOF. Let $c \in (W_j(\mathbb{F}_q))[X]$ such that $f - g_j \cdot h_j = p^j \cdot c$ over $W_{2j}(\mathbb{F}_q)$. Compute $\tilde{a}, \tilde{b} \in (W_j(\mathbb{F}_q))[X]$ such that $\tilde{a} \cdot g_j + \tilde{b} \cdot h_j = c$ over $W_j(\mathbb{F}_q)$, using Algorithm 6.2. Define $g_{2j} = g_j + p^j \cdot \tilde{b}$ and $h_{2j} = h_j + p^j \cdot \tilde{a}$. It is trivial to verify that g_{2j} and h_{2j} satisfy the conditions above.

Let $r \in (W_j(\mathbb{F}_q))[X]$ such that $a_j \cdot g_{2j} + b_j \cdot h_{2j} = 1 + p^j \cdot r$ over $W_{2j}(\mathbb{F}_q)$. Compute $\bar{a}, \bar{b} \in (W_j(\mathbb{F}_q))[X]$ such that $\bar{a} \cdot g_j + \bar{b} \cdot h_j = r$ over $W_j(\mathbb{F}_q)$. Define $a_{2j} = a_j - p^j \cdot \bar{a}$ and $b_{2j} = b_j - p^j \cdot \bar{b}$, then $a_{2j}, b_{2j} \in (W_{2j}(\mathbb{F}_q))[X]$ and

$$\begin{aligned} a_{2j} \cdot g_{2j} + b_{2j} \cdot h_{2j} &= (a_j - p^j \cdot \bar{a}) \cdot g_{2j} + (b_j - p^j \cdot \bar{b}) \cdot h_{2j} \\ &= a_j \cdot g_{2j} + b_j \cdot h_{2j} - p^j \cdot (\bar{a} \cdot g_{2j} + \bar{b} \cdot h_{2j}) \\ &= 1 \text{ over } W_{2j}(\mathbb{F}_q). \quad \square \end{aligned}$$

Surprisingly MIOLA and YUN [24] showed that the quadratic lift-algorithm based on the proof of this lemma is usually slower than the linear lift-algorithm. Fortunately, ZASSENHAUS [42] indicated, in response to Miola and Yun, how to interpret his own paper [41], and he presented a modified version of the quadratic lift-algorithm which is indeed substantially faster than the linear method. We give a brief description of his algorithm; for proofs and details see [42].

ALGORITHM 6.3. (Quadratic lift-algorithm). Let $f \in (W_{2k}(\mathbb{F}_q))[X]$, f monic, and $a, b, g_0, h_0 \in \mathbb{F}_q[X]$ such that $f = g_0 \cdot h_0$ and $a \cdot g_0 + b \cdot h_0 = 1$ over $\mathbb{F}_q$. The algorithm computes $g_k, h_k \in (W_{2k}(\mathbb{F}_q))[X]$ such that $f = g_k \cdot h_k$ over $W_{2k}(\mathbb{F}_q)$.

(1) $j := 0$, $v_0 := a \cdot g_0$, $w_0 := v_0 \cdot (v_0 - 1)$ modulo f.

(2) Let e_j be the largest exponent for which $p^{e_j} | w_j$. If $w_j = 0$ or $j = k$ or $e_j \geq 2^k$ then $v_k := v_j$, $j := k$ and go to step (4).

(3) $v_{j+1} := v_j + w_j - 2\cdot((v_j\cdot w_j) \text{modulo } f)$ over $W_{2e_j}(\mathbb{F}_q)$, $w_{j+1} := v_j\cdot(v_j - 1)$ modulo f (now $p^{2\cdot e_j} | w_{j+1}$), $j := j + 1$ and go back to step (2).

(4) $g_k := \gcd(f, v_k)$ over $W_{2k}(\mathbb{F}_q)$ and $h_k := f/g_k$ over $W_{2k}(\mathbb{F}_q)$ (Zassenhaus proves that this gcd exists in this case, and he shows how to compute it).

7. FACTORIZATION IN $\mathbb{Z}[X]$

Let f be a polynomial of degree n in $\mathbb{Z}[X]$. The *content* cont(f) of f is defined as the gcd of the coefficients of f, and the *primitive part* pp(f) of f is defined as f/cont(f). We assume f to be *primitive*, i.e. cont(f) = 1.

In this section we are concerned with the problem of finding the irreducible factorization of f over $\mathbb{Z}$. Clearly, a repeated factor of f is also a factor of f', so we can construct the square-free decomposition of f using an algorithm similar to the algorithm from Section 2, if we are able to perform gcd-computations in $\mathbb{Z}[X]$.

Therefore, we concentrate first on gcd-computations of polynomials over $\mathbb{Z}$. Suppose we want to compute the gcd g of two polynomials f_1 and f_2 in $\mathbb{Z}[X]$. It is a well-known fact that g is also a polynomial in $\mathbb{Z}[X]$. However, if we apply Euclid's algorithm to f_1 and f_2, we have to compute $f_{i+1} \bmod f_i$, $i = 2,3,\ldots$, until $f_{i+1} = 0$, and in general the coefficients of the polynomials f_{i+1} will not be in $\mathbb{Z}$, but in $\mathbb{Q}$. This is in many cases unacceptable, and therefore we have to find a way to remain in $\mathbb{Z}[X]$. One way to deal with this problem is to use *pseudo-division* in Euclid's algorithm, i.e.

$$f_{i+1} = (\ell c(f_i)^{\deg(f_{i-1})-\deg(f_i)+1} \cdot f_{i-1}) \bmod f_i.$$

Now f_{i+1} is certainly in $\mathbb{Z}[X]$, but in practice this so-called EPRS-algorithm (Euclidean Polynomial Remainder Sequence) is completely useless because of the exponential coefficient growth. There are several improvements on the EPRS-algorithm which keep the coefficient growth linear, or almost always linear, like Primitive PRS (PPRS), Subresultant PRS (SPRS) and Improved SPRS (ISPRS), see [5,6,7,11,12,13,18], but we will leave the PRS-algorithms and we take another approach.

THEOREM 7.1. *Let* $f_1, f_2, g \in \mathbb{Z}[X]$, $g = \gcd(f_1, f_2)$. *For all primes* p, *such that* $p \nmid \ell c(f_1)\cdot\ell c(f_2)$ *we have* $\deg(g_p) \geq \deg(g)$, *where* $g_p = \gcd(f_1, f_2)$ *over* $\mathbb{Z}/p\mathbb{Z}$, *and there are only finitely many* p *such that* $\deg(g_p) > \deg(g)$.

PROOF. The first assertion is trivial, the second assertion follows when we look at the leading coefficients of the polynomials f_{i+1} in the EPRS-algorithm. A larger degree for the gcd g_p can only occur if p divides one of these leading coefficients, and this can only happen for finitely many primes. □

Combining this theorem with the well-known Chinese Remainder Theorem, we get the so-called *modular gcd-algorithm* (MODGCD).

ALGORITHM 7.1. (MODGCD). Let $f_1, f_2 \in \mathbb{Z}[X]$, f_1 and f_2 primitive, this algorithm computes $\gcd(f_1, f_2)$ in $\mathbb{Z}[X]$.

(1) Choose a large prime p, such that $p \nmid \ell c(f_1) \cdot \ell c(f_2)$, $g := \gcd(f_1 \bmod p, f_2 \bmod p) \in (\mathbb{Z}/p\mathbb{Z})[X]$ (with g monic), product := p.

(2) Test whether or not g will lead to the gcd of f_1 and f_2 over $\mathbb{Z}$:
take $h = \ell c(f_1) \cdot g$ modulo product.
if $h \mid \ell c(f_1) \cdot f_1$ and $pp(h) \mid f_2$ then $pp(h) = \gcd(f_1, f_2)$
else go to step (3).

(3) Choose another large prime p', $p' \nmid \ell c(f_1) \cdot \ell c(f_2)$, $g_{p'} := \gcd(f_1 \bmod p', f_2 \bmod p') \in (\mathbb{Z}/p'\mathbb{Z})[X]$ (with $g_{p'}$ monic).
There are three possibilities:
- $\deg(g_{p'}) < \deg(g)$: $g := g_{p'}$, product := p', go to step (2),
- $\deg(g_{p'}) = \deg(g)$: combine g and $g_{p'}$ with the Chinese Remainder Theorem to the unique polynomial h, such that $h \equiv g \bmod$ product, $h \equiv g_{p'} \bmod p'$, and all coefficients of h are in absolute value $\leq \lfloor$product$\cdot p'/2 \rfloor$ (so h is monic).
  $g := h$, product := product$\cdot p'$, go to step (2),
- $\deg(g_{p'}) > \deg(g)$: repeat step (3).

Remark how we construct the correct leading coefficient of the gcd of f_1 and f_2 over $\mathbb{Z}$ in step (2): multiply the monic testpolynomial by $\ell c(f_1)$ modulo the product of the primes, and test whether it is a divisor of $\ell c(f_1) \cdot f_1$ over $\mathbb{Z}$. If so, the possibly non-monic primitive part is the gcd over $\mathbb{Z}$, if it divides f_2.

It follows from Theorem 7.1 that the modular gcd-algorithm terminates: after a finite (but unknown) number of steps $\lfloor$product$/2 \rfloor$ is larger than the absolute value of any coefficient of the gcd of f_1 and f_2 over $\mathbb{Z}$.

It is also possible to use the lift-algorithms from Section 6 to find the gcd of two polynomials over $\mathbb{Z}$. This is the so-called EZGCD-algorithm (Extended Zassenhaus GCD, see [24,26,39]); we will not give a description of this algorithm.

Now let us return to the factorization of f over $\mathbb{Z}$. In view of the above gcd-algorithm we assume f to be square-free. A classical method to factorize f was given by KRONECKER (see for instance VAN DER WAERDEN [32]). To find a factor g of degree $\leq m$, choose $m+1$ distinct integers $a_0,\ldots,a_m$. Then $g(a_i) \mid f(a_i)$ for $i = 0,\ldots,m$, because g is a divisor of f. Now test for all choices of divisors d_i of $f(a_i)$, for $i = 0,\ldots,m$, whether the unique polynomial g of degree $\leq m$, satisfying $g(a_i) = d_i$, $i = 0,\ldots,m$, divides f. Although several improvements are possible, this remains a very inefficient algorithm, even for reasonably small values of m. Using a similar approach, ADLEMAN and ODLYZKO [1] prove (on hypothesis) that the problems of irreducibility testing and factorization of polynomials over $\mathbb{Z}$ are polynomial time reducible to the problem of integer primality testing and factorization respectively; the practical importance of their method is questionable.

The following method, which makes use of the results from Sections 4 and 6, appears to be very useful in practice. First we present this algorithm, next we give an explanation.

ALGORITHM 7.2. (Factorization in $\mathbb{Z}[X]$). Let $f = \Sigma_{i=0}^{n} f_i X^i$ be a primitive, square-free polynomial. This algorithm computes the irreducible factors of f over $\mathbb{Z}$.

(1) Choose a prime p and factorize f completely over $\mathbb{Z}/p\mathbb{Z}$.

(2) Take k minimal such that

$$p^k \geq 2 \cdot |\ell c(f)| \cdot \binom{\lfloor n/2 \rfloor}{\lfloor n/4 \rfloor} \cdot \sqrt{\Sigma_{i=0}^{n} f_i^2} \quad , \qquad (*)$$

and lift the factorization of f over $\mathbb{Z}/p\mathbb{Z}$ to the factorization over $\mathbb{Z}/p^k\mathbb{Z}$:

$$f \equiv \ell c(f) \cdot \prod_{i=1}^{r} h_i \text{ modulo } p^k.$$

(3) Compute the polynomial $h = (\ell c(f) \cdot \Pi_{i \in S} h_i)$ modulo p^k for all subsets $S \subset \{1,\ldots,r\}$ such that $\deg(h) \leq \lfloor n/2 \rfloor$, and test whether h is a divisor of $\ell c(f) \cdot f$. If so, then pp(h) is a divisor of f over $\mathbb{Z}$.

In step (2) of this algorithm we apply one of the lift-algorithms from Section 6 to the factorization of f over $\mathbb{Z}/p\mathbb{Z}$. Therefore the factors of f modulo p have to be relatively prime, i.e. f modulo p must remain square-free. According to VAN DER WAERDEN [32] this condition is met if $p \nmid \text{discr}(f)$. Furthermore p must be chosen in such a way that $\ell c(f) \neq 0$ in $\mathbb{Z}/p\mathbb{Z}$, i.e.

$p \nmid \ell c(f)$. Since f is square-free over $\mathbb{Z}$ we have that discr(f) $\neq 0$. It follows that a prime p satisfying both conditions can be found.

In practice we can always find a small prime, so that we can apply Berlekamp's factorization algorithm to factorize f modulo p. For large n it might be preferable to compute the partial factorization of f first. This saves the diagonalization of a large matrix, even if f has only equal degree factors (use in that case the second Cantor-Zassenhaus algorithm).

The irreducible factors of f over $\mathbb{Z}$ are also factors modulo p, for every prime p, but they are not necessarily irreducible modulo p. This implies that we have to look at combinations of irreducible factors modulo p to find the irreducible factors over $\mathbb{Z}$. But the absolute values of the coefficients of these combinations are bounded from above by $\lfloor p/2 \rfloor$; this bound generally does not hold for the coefficients of the factors over $\mathbb{Z}$. Therefore we first have to lift the factorization modulo p to the factorization modulo p^k, where k is chosen in such a way that $p^k/2$ is greater than the largest possible coefficient of a factor of f over $\mathbb{Z}$. A bound on the coefficients of the factors of f over $\mathbb{Z}$ is given by MIGNOTTE [23]; let $g = \Sigma_{i=0}^{\ell} g_i X^i$ be a factor of $f = \Sigma_{i=0}^{n} f_i X^i$ over $\mathbb{Z}$, then

$$|g_i| \leq \binom{\ell}{i} \cdot \sqrt{\Sigma_{i=0}^{n} f_i^2}, \quad i = 0,\ldots,\ell.$$

Now remark that in fact we are looking for factors of $\ell c(f) \cdot f$ (due to the mechanism to restore the leading coefficient, like in Algorithm 7.1 step (2)), and that we can restrict ourselves to factors of degree $\leq \lfloor n/2 \rfloor$, so that (*) follows.

Presently there is in general no way to see which of the irreducible factors modulo p^k together form an irreducible factor over $\mathbb{Z}$, so we have to test for all possible combinations of degree $\leq \lfloor n/2 \rfloor$ whether they lead to a factor over $\mathbb{Z}$. If f is irreducible none of these division-tests will be successful; in that case the number of tests is exponential in r, the number of factors modulo p^k. KALTOFEN et al. [17] constructed a class of irreducible polynomials over $\mathbb{Z}$, for which r is linear in the degree, for any prime p; this implies that the number of combinations can become exponential in the degree. In general it is often possible to reduce the number of division tests in $\mathbb{Z}[X]$ considerably:

- first try the constant coefficient.
- factorize f modulo a number of primes. Use a prime with the smallest number of factors in step (2) and (3), and combine the information about the

degrees of the factors modulo the different primes to reduce the possibilities for the degrees of the factors over $\mathbb{Z}$. For example, if f factors modulo p_1 in 3 factors of degree 2, and f factors modulo p_2 in 2 factors of degree 3, then f is irreducible over $\mathbb{Z}$.

- lift from $\mathbb{Z}/p\mathbb{Z}$ to $\mathbb{Z}/p^k\mathbb{Z}$ for some $\tilde{k} > k$, and do not try combinations which do not satisfy the given bound.

There are two obvious ways to implement the last step of Algorithm 7.2. The first is the so-called *cardinality procedure*: take combinations of s factors at a time, for $s = 1,\ldots,\lfloor r/2 \rfloor$; the second is the *degree procedure*: take combinations of total degree s, for $s = 1,\ldots,\lfloor n/2 \rfloor$. If all partitions of n consisting of the degrees of the irreducible factors of f over $\mathbb{Z}$ are chosen to be equally likely, then COLLINS [14] gives some evidence that the mean number of combinations formed before finding an irreducible factor is dominated by n^2 if the cardinality procedure is used and that it is exponential if the degree procedure is used. The cardinality procedure is therefore preferable.

Algorithm 7.2 is a widely used algorithm to factorize polynomials over the integers, and in practice it appears to be reasonably efficient. There are some other, generally less efficient or even very inefficient algorithms which might however, in special cases, be preferable to Algorithm 7.2. For instance the multi-dimensional continued fraction algorithm of FERGUSON and FORCADE [15] (see also BRENTJES [4]) can be used to determine a factor or prove irreducibility of the polynomial f over $\mathbb{Z}$. Let $\alpha \in \mathbb{R}$ be a real root of f, then the algorithm of Ferguson and Forcade decides whether there exists a factor g of degree $\leq \ell$ of f, such that $g(\alpha) = 0$. If g exists the algorithm finds a $\mathbb{Z}$-linear relation among $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^\ell$, thus giving the coefficients of g; otherwise, it gives a lower bound for the height of such a $\mathbb{Z}$-linear relation which will prove, together with the bounds of MIGNOTTE [23], that g cannot exist (some modifications are necessary if $\alpha \in \mathbb{C}\backslash\mathbb{R}$). For small ℓ this is a reasonably efficient method; a drawback is that to implement it a high-precision floating-point arithmetic is needed.

Another way to discover factors of f was suggested by LENSTRA [19]. Let h_k be a monic irreducible factor of f modulo p^k, where p discr(f), and let $g = \Sigma_{i=0}^{\ell} g_i X^i \in \mathbb{Z}[X]$ be an irreducible factor of f over $\mathbb{Z}$ such that $h_k | g$ modulo p^k. In [19] it is proven that if $v = \Sigma_{i=0}^{m} v_i X^i \in \mathbb{Z}[X]$ is an arbitrary polynomial such that gcd(g,v) = 1 over $\mathbb{Z}$ and such that $h_k | v$ modulo p^k, then

$$p^{k\cdot \deg(h_k)} \leq \left(\sum_{i=0}^{\ell} g_i^2\right)^{m/2} \cdot \left(\sum_{i=0}^{m} v_i^2\right)^{\ell/2} \qquad (**)$$

From (**) and the Mignotte bounds it follows that, if k is chosen sufficiently large, g is the shortest-length non-zero vector in the $(\ell+1)$-dimensional lattice

$$L_k = \{q\cdot h_k + p^k\cdot r \mid q,r \in \mathbb{Z}[X],\ \deg(q) \leq \ell - \deg(h_k), \deg(r) < \deg(h_k)\} \subset \mathbb{R}^{\ell+1},$$

where the polynomials $q\cdot h_k + p^k\cdot r$ are regarded as $(\ell+1)$-dimensional vectors. The polynomial g can therefore be found by looking for the shortest-length vector in L_k, or by looking for a sufficiently orthogonal basis of L_k. L. Lovász proved that such a basis can be determined in polynomial time, so that the latter approach leads to a polynomial-time algorithm for factoring primitive polynomials over $\mathbb{Z}$. This algorithm is described in detail in [20]. In Section 8 we will see another important application of (**).

We conclude this section with a short remark on tests for irreducibility of polynomials over $\mathbb{Z}$. The test suggested above, based on the comparison of the degrees of the factors modulo p, for several values of p, was analysed by MUSSER [28]. As we have seen this method does not work for all irreducible polynomials [17] (for instance $X^4 + 1$), but Musser's test results show that the mean number of primes needed to prove the irreducibility of a randomly chosen irreducible polynomial grows very slowly with the degree, and is in fact ≤ 5 for degrees ≤ 200.

8. FACTORIZATION IN $(\mathbb{Q}(\alpha))[X]$

Let f be a monic, square-free polynomial of degree n in $(\mathbb{Q}(\alpha))[X]$, where α is a zero of an irreducible, monic polynomials F (the *minimal polynomial*) of degree m in $\mathbb{Z}[X]$. We describe in this section several methods to determine the complete factorization of f over $\mathbb{Q}(\alpha)$. We restrict ourselves to square-free polynomials; if the given polynomial is not square-free, we determine the square-free decomposition using Lemma 2.2, which is also valid for polynomials in $(\mathbb{Q}(\alpha))[X]$. To perform the necessary gcd-computations in $(\mathbb{Q}(\alpha))[X]$, we can of course use the Euclidean algorithm, but a serious drawback is that in general we will get rather large denominators. With some modifications the modular gcd-algorithm from Section 7 can

be adapted to $(\mathbb{Q}(\alpha))[X]$. We will not give a description of this algorithm.

Factorization in $(\mathbb{Q}(\alpha))[X]$ introduces the problem of choosing a denominator. The first thing we remark is that there is an integer d such that $f \in (\frac{1}{d}\mathbb{Z}[\alpha])[X]$. However, even in the case that $d = 1$ (i.e. $f \in (\mathbb{Z}[\alpha])[X]$) it is possible that the irreducible factors of f over $\mathbb{Q}(\alpha)$ have a non-trivial denominator, i.e. new denominators can occur when factoring over $\mathbb{Q}(\alpha)$. With the help of the following two lemmas we determine an integer D such that f and all monic factors of f over $\mathbb{Q}(\alpha)$ lie in $(\frac{1}{D}\mathbb{Z}[\alpha])[X]$.

LEMMA 8.1. *Let* $f \in (\frac{1}{d}R)[X]$, f *monic, and* g *and* h *monic polynomials in* $(\mathbb{Q}(\alpha))[X]$ *such that* $f = g \cdot h$, *then* $g,h \in (\frac{1}{d}R)[X]$, *where* R *is the ring of integers of* $\mathbb{Q}(\alpha)$. □

LEMMA 8.2. *Let* $b = \max\{d \in \mathbb{N} \text{ such that } d^2 \mid \text{discr}(F)\}$, *then* $\text{defect}(\alpha) \mid b$, *and therefore* $R \subset \frac{1}{b}\mathbb{Z}[\alpha]$, *where* $\text{defect}(\alpha) = \min\{d \in \mathbb{N} \text{ such that } R \subset \frac{1}{d}\mathbb{Z}[\alpha]\}$. □

If $f \in (\frac{1}{d}\mathbb{Z}[\alpha])[X]$ then certainly $f \in (\frac{1}{d}R)[X]$. Applying Lemma 8.1, we see that every monic factor of f over $\mathbb{Q}(\alpha)$ will be an element of $(\frac{1}{d}R)[X]$. But we can also find $c \in \mathbb{N}$ such that $R \subset \frac{1}{c}\mathbb{Z}[\alpha]$; we take $c = \text{defect}(\alpha)$ if $\text{defect}(\alpha)$ is known, else we use the b from Lemma 8.2. Combining these results we obtain the D we wanted to have: $D = d \cdot c$.

Having determined a value for D, let us return to the problem of finding the complete factorization of f over $\mathbb{Q}(\alpha)$. If a prime p can be found such that the minimal polynomial F remains irreducible over $\mathbb{Z}/p\mathbb{Z}$, and such that a few other, more trivial, conditions are met, the factorization algorithm is similar to Algorithm 7.2 (Factorization in $\mathbb{Z}[X]$).

ALGORITHM 8.1. (Factorization in $(\mathbb{Q}(\alpha))[X]$, minimal polynomial irreducible modulo p). Let $f = \frac{1}{d}\sum_{i=0}^{n}(\sum_{j=0}^{m-1} c_{ij}\alpha^j)X^i$ be a monic, square-free polynomial in $(\frac{1}{d}\mathbb{Z}[\alpha])[X]$, and α a zero of the minimal polynomial $F(T) = \sum_{\ell=0}^{m} F_\ell T^\ell$. This algorithm computes the monic, irreducible factors of f over $\mathbb{Q}(\alpha)$.

(1) Choose D as above.

(2) Choose a prime p and factorize f completely over $\mathbb{F}_q$, where
$\mathbb{F}_q = \{\sum_{i=0}^{m-1} a_i\alpha^i \mid a_i \in \mathbb{Z}/p\mathbb{Z},\ i = 0,\ldots,m-1\}$ (so $q = p^m$).

(3) Take $k \in \mathbb{Z}$ minimal such that

$$p^k \cdot \sqrt{|\text{discr}(F)|} \geq 2Dm!\binom{\lfloor n/2\rfloor}{\lfloor n/4\rfloor} \cdot (\sum_{\ell=0}^{m-1}|F_\ell|)^{m(m-1)/2} \cdot \cdot (\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}|c_{ij}|(\sum_{\ell=0}^{m-1}|F_\ell|)^j/d)^{\lfloor n/4\rfloor},$$

and lift the factorization of f over $\mathbb{F}_q$ to the factorization over $W_k(\mathbb{F}_q)$:
$(d^{-1} \bmod p^k)\cdot(d\cdot f) \equiv \Pi_{i=1}^{r} h_i$ over $W_k(\mathbb{F}_q)$
(remark that $\mathbb{Z}[\alpha]/(p^k) \cong W_k(\mathbb{F}_q)$).

(4) Compute the polynomial $h = \frac{1}{D}((D\cdot\Pi_{i\in S} h_i)$ over $W_k(\mathbb{F}_q))$ for all subsets $S \subset \{1,\ldots,r\}$ such that $\deg(h) \leq \lfloor n/2 \rfloor$, and test whether h is a divisor of f over $\frac{1}{D}\mathbb{Z}[\alpha]$.

This algorithm can only be applied if a prime p can be found such that (i) $p \nmid D$, (ii) the minimal polynomial F remains irreducible modulo p, and (iii) f remains square-free over $\mathbb{F}_q$. In step (2) we take a prime satisfying these conditions. An upper bound for the coefficients (in $\frac{1}{D}\mathbb{Z}[\alpha]$) of the factors of f over $\mathbb{Q}(\alpha)$ is given in WEINBERGER and ROTHSCHILD [38]; from their bound follows the value for k in step (3). In practice this value for k is almost always much too large, and therefore we often use a heuristic bound (see for instance WANG [34]). Remark that we apply the lift algorithm to the monic polynomial $(d^{-1} \bmod p^k)\cdot(d\cdot f) \in (\mathbb{Z}[\alpha])[X]$. The denominator of the trial-divisors in step (4) is constructed in the obvious way: multiply by D modulo p^k, and attach the denominator D to each of the coefficients. Now that we have seen how to handle the denominator D, we take for simplicity D = 1 in the sequel.

Difficulties arise in the case that Algorithm 8.1 cannot be applied because a suitable prime cannot be found. This means that either F is irreducible over $\mathbb{Z}/p\mathbb{Z}$ and f has repeated factors over $\mathbb{F}_q$, or F reduces modulo p. The first can only be true for a finite number of primes, so we conclude that F reduces modulo p. Since $\{\Sigma_{i=0}^{m-1} a_i\alpha^i \mid a_i \in \mathbb{Z}/p\mathbb{Z},\ i = 0,\ldots,m-1\}$ is not a finite field in this case, we cannot apply our usual factorization scheme of factoring over a finite field, extending this to the factorization over a large enough ring, and finding the true factors. There are several ways out of this problem.

The first solution comes from VAN DER WAERDEN [32] (see also WANG [34]). He reduces the problem of factoring f over $\mathbb{Q}(\alpha)$ to the problem of factoring a polynomial in two variables over the integers. As the degree of this polynomial is n·m in each variable, this is not a very efficient method, and we will not discuss it here. An improvement of Van der Waerden's algorithm was given by TRAGER [31]. Instead of the factorization of a multivariate polynomial we now get the factorization of a univariate polynomial of degree n·m over $\mathbb{Z}$. Trager's method is based on the following theorem.

THEOREM 8.1. *Let* $f \in (\mathbb{Q}(\alpha))[X]$, *such that* $\mathrm{Norm}(f) \in \mathbb{Q}[X]$ *is square-free where* $\mathrm{Norm}(f) = \prod_{i=1}^{m} f(X,\alpha_i)$ *is the product of* $f(X,\alpha_i)$ *over the* m *conjugates of* α *(so* Norm(f) *has degree* $n \cdot m$ *and is in* $\mathbb{Q}[X]$*). Let* $\prod_{i=1}^{r} g_i$ *be the complete factorization of* Norm(f) *over* $\mathbb{Q}$, *then* $\prod_{i=1}^{r} \gcd(f,g_i)$ *is the complete factorization of* f *over* $\mathbb{Q}(\alpha)$. □

To implement the factorization algorithm based on Theorem 8.1, we need, besides an algorithm to factorize in $\mathbb{Z}[X]$, an algorithm to compute the norm of f and an algorithm to transform f, if necessary, to a polynomial with a square-free norm. The norm can be calculated using an algorithm of COLLINS [13], and Trager proves that there is only a finite number of elements $s \in \mathbb{Q}$ such that $\mathrm{Norm}(f(X-s\alpha))$ is not a square-free. For further details and proofs see [31].

Another approach was proposed by WEINBERGER and ROTHSCHILD [38]. The complete factorization $F = \prod_{i=1}^{t} F_{1i}$ modulo p of the minimal polynomial over $\mathbb{Z}/p\mathbb{Z}$ defines a number (t) of finite fields. Namely, let α_{1i} denote a zero of F_{1i}, $i = 1,\ldots,t$, then we have the following t finite fields: $\mathbb{F}_{q_i} = \{\sum_{j=0}^{\deg(F_{1i})-1} a_j \alpha_{1i}^j \mid a_j \in \mathbb{Z}/p\mathbb{Z},\ j = 0,\ldots,\deg(F_{1i}) - 1\}$, $i = 1,\ldots,t$. Now we choose p such that F is square-free modulo p and such that f remains square-free over $\mathbb{F}_{q_i}$ for $i = 1,\ldots,t$ (clearly this is possible), and we take k as in step (2) of Algorithm 8.1. Then we are able to calculate the complete factorizations of f over $W_k(\mathbb{F}_{q_i}) = \{\sum_{j=0}^{\deg(F_{ki})-1} a_j \alpha_{ki}^j \mid a_j \in \mathbb{Z}/p^k\mathbb{Z}\ \ j = 0,\ldots,\deg(F_{ki}) - 1\}$, where $F = \prod_{i=1}^{t} F_{ki}$ modulo p^k and α_{ki} denotes a zero of F_{ki}, for $i = 1,\ldots,t$ (remark that $\mathbb{Z}[\alpha]/(F_{ki},p^k) \cong W_k(\mathbb{F}_{q_i})$). Let $g \in (\mathbb{Z}[\alpha])[X]$ be an irreducible factor of f over $\mathbb{Q}(\alpha)$, then g modulo p^k and F_{ki} (i.e. g reduced to $(W_k(\mathbb{F}_{q_i}))[X]$) is a combination of irreducible factors of f over $W_k(\mathbb{F}_{q_i})$, for $i = 1,\ldots,t$. Furthermore g can be reconstructed from g mod (p^k, F_{ki}), $i = 1,\ldots,t$, with the help of the Chinese Remainder Algorithm [21]. Therefore we can find the irreducible factors of f over $\mathbb{Q}(\alpha)$ by applying the Chinese Remainder Algorithm to each possible t-tuple of equal degree factors over $W_k(\mathbb{F}_{q_i})$, $i = 1,\ldots,t$. For large t this last step is the most time consuming part of the factorization algorithm. For instance, to prove the irreducibility of an irreducible polynomial over $\mathbb{Q}(\alpha)$ that factors over $\mathbb{F}_{q_i}$ in r equal degree factors for $i = 1,\ldots,t$, we must perform $\sum_{i=1}^{\lfloor r/2 \rfloor} \binom{r}{i} t \sim (2^{rt-1}/\sqrt{t})(\frac{2}{\pi r})^{(t-1)/2}$ $(r \to \infty)$ [29, p. 42] trial-divisions. But also in the case that f is irreducible over $\mathbb{Q}(\alpha)$ and we are unlucky, the number of trials can become exponential in nt. This algorithm is therefore only practically applicable if the number of factors of the minimal polynomial modulo p is

reasonably small (i.e. ≤ 3). Remark that the Weinberger and Rothschild algorithm poses the problem of an efficient implementation of the last step, minimizing the number of multiplications in $(W_k(\mathbb{F}_{q_i}))[X]$, the number of applications of the Chinese Remainder Algorithm, and with fast update facilities in the case that a factor is found.

The last method we discuss was suggested by H.W. Lenstra Jr. He made the observation that the use of the Chinese Remainder Algorithm in the above algorithm is not necessary; the factorization of f over one truncated Witt-ring $W_k(\mathbb{F}_{q_i})$ is sufficient to construct the factors of f over $\mathbb{Q}(\alpha)$, if k is chosen sufficiently large. Let F_k be an irreducible non-trivial factor of the minimal polynomial modulo p^k, and suppose that we have computed the complete (square-free) factorization of f over $W_k(\mathbb{F}_q)$, the truncated Witt-ring generated by F_k and p^k. If f has an irreducible factor $g \in (\mathbb{Z}[\alpha])[X]$ of degree ℓ over $\mathbb{Q}(\alpha)$, then clearly $g_k \equiv g \bmod (F_k, p^k) \in (W_k(\mathbb{F}_q))[X]$ is one of the (not necessarily irreducible) factors of f over $W_k(\mathbb{F}_q)$. Let $v \in \mathbb{Z}[\alpha]$ be the i-th coefficient of g (regarded as polynomial in α), for some $i \in \{0,\ldots,\ell\}$, and let $v_k = v \bmod (F_k, p^k) \in W_k(\mathbb{F}_q)$ be the i-th coefficient of g_k. Define the m-dimensional lattice $L_k = \{q \cdot F_k + p^k \cdot r \mid q,r \in \mathbb{Z}[\alpha], \deg(q) < m - \deg(F_k), \deg(r) < \deg(F_k)\} \subset \mathbb{R}^m$ (regard the polynomials $q \cdot F_k + p^k \cdot r$ as m-dimensional vectors), then clearly v and v_k are congruent modulo L_k. It is not difficult to derive from (**), Section 7, that for all $B > 0$, there is an integer $k_0 = k_0(B)$, such that all elements $\neq 0$ of L_k have length $\geq B$ for $k \geq k_0$. Since we can give an a priori upper bound for the length of v (Algorithm 8.1, step (3)), we conclude that v is the shortest-length vector congruent to v modulo L_k, if k is chosen sufficiently large. An algorithm of L. LOVÁSZ [20] enables us to construct in polynomial time a sufficiently orthogonal basis of L_k, so that we can compute v by reducing v_k modulo this basis. Experiments showed that this so-called *lattice-algorithm* is quite useful, and not only in the case of the reducing minimal polynomial; looking for a prime such that the minimal polynomial reduces and applying the lattice algorithm appeared to be faster than using Algorithm 8.1. For a complete description and some computer timings we refer to [19]. The approach from Section 7, where we viewed factors as shortest-length vectors in a certain lattice, will even lead to a polynomial-time algorithm for factoring univariate polynomials over algebraic number fields.

9. FACTORIZATION OF MULTIVARIATE POLYNOMIALS

In the previous sections we restricted ourselves to the factorization of univariate polynomials. There are several methods to extend these algorithms to multivariate polynomials [27,33,34,36,39]. We given an outline of Wang's algorithm for factoring in $\mathbb{Z}[X_1,\ldots,X_t]$; for the numerous details we refer to his paper [36].

Let $f \in \mathbb{Z}[X_1,\ldots,X_t]$ be the square-free polynomial to be factored (see [35, 37] for the square-free decomposition of multivariate polynomials). First we evaluate f in the point $(X_2 = a_2,\ldots,X_t = a_t)$ for some choice of integers $a_2,\ldots,a_t$. The resulting univariate polynomial $f(X_1,a_2,\ldots,a_t) \in \mathbb{Z}[X_1]$ is then completely factored over $\mathbb{Z}$. Next we apply a generalization of the algorithms from Section 6 to lift the factorization of $f(X_1,\ldots,X_i,a_{i+1},\ldots,a_t)$ over $\mathbb{Z}[X_1,\ldots,X_i]$ to a factorization over $\mathbb{Z}[X_1,\ldots,X_{i+1}]$, for $i = 1,\ldots,t-1$. After each application of the lift-algorithm we determine the true factors of $f(X_1,\ldots,X_{i+1},a_{i+2},\ldots,a_t)$ over $\mathbb{Z}[X_1,\ldots,X_{i+1}]$. This can be done for instance by a combinatorial search, like in step (3) of Algorithm 7.2. For $i = t-1$ this gives the complete factorization of f in $\mathbb{Z}[X_1,\ldots,X_t]$.

A completely different approach to the factorization of multivariate polynomials is given in CLAYBROOK [10] and ZIPPEL [43]. In their algorithms the sparseness of the polynomial to be factored, and of its factors, is essential.

REFERENCES

[1] ADLEMAN, L.M. & A.M. ODLYZKO, *Irreducibility testing and factorization of polynomials* (extended abstract), Proc. 22nd Annual IEEE Symp. Found. Comp. Sci., 409-418.

[2] BERLEKAMP, E.R., *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[3] BERLEKAMP, E.R., *Factoring polynomials over large finite fields*, Math. Comp. 24 (1970), 713-735.

[4] BRENTJES, A.J., *Multi-dimensional continued fraction algorithms*, Doctoral thesis, Mathematical Centre Tract 145/81 (see also this volume).

[5] BROWN, W.S., *On Euclid's algorithm and the computation of polynomial greatest common divisors*, J. ACM 18 (1971), 478-504.

[6] BROWN, W.S. & J.F. TRAUB, *On Euclid's algorithm and the theory of subresultants*, J. ACM 18 (1971), 505-514.

[7] BROWN, W.S., *The subresultant PRS algorithm*, ACM Transactions on Mathematical Software 4 (1978), 237-249.

[8] CALMET, J. & R. LOOS, *An improvement of Rabin's probabilistic algorithm for generating irreducible polynomials over* GF(p), Information Processing Letters 11 (1980), 94-95.

[9] CANTOR, D.G. & H. ZASSENHAUS, *A new algorithm for factoring polynomials over finite fields*, Math. Comp. 36 (1981), 587-592.

[10] CLAYBROOK, B.G., *A new approach to the symbolic factorization of multivariate polynomials*, Artificial Intelligence 7 (1976), 203-241.

[11] COLLINS, G.E., *Polynomial remainder sequences and determinants*, Amer. Math. Monthly 73 (1966), 708-712.

[12] COLLINS, G.E., *Subresultants and reduced polynomial remainder sequences*, J. ACM 14 (1967), 128-142.

[13] COLLINS, G.E., *The calculation of multivariate polynomial resultants*, J. ACM 18 (1971), 515-532.

[14] COLLINS, G.E., *Factoring univariate integral polynomials in polynomial average time*, Proceedings of EUROSAM '79, 317-329.

[15] FERGUSON, H.R.P. & R.W. FORCADE, *Generalization of the Euclidean algorithm for real numbers to all dimensions higher than two*, Bull. AMS 1 (1979), 912-914.

[16] GUNJI, H. & D. ARNON, *On polynomial factorization over finite fields*, Math. Comp. 36 (1981), 281-287.

[17] KALTOFEN, E. & D.R. MUSSER & B.D. SAUNDERS, *A generalized class of polynomials that are hard to factor*, Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation.

[18] KNUTH, D.E., *The art of computer programming*, Vol. 2, second edition, Addison-Wesley, Reading, Mass. 1981.

[19] LENSTRA, A.K., *Lattices and factorization of polynomials*, Mathematisch Centrum, Amsterdam, IW 190/81.

[20] LENSTRA, A.K. & H.W. LENSTRA Jr. & L. LOVÁSZ, *Factoring polynomials with rational coefficients*, Univ. of Amsterdam, Dept. of Math., Report 82-05.

[21] LIPSON, J.D., *Chinese remainder and interpolation algorithms*, Proceedings of the 1971 ACM Symposium on symbolic and algebraic manipulation, 372-391.

[22] McELIECE, R.J., *Factorization of polynomials over finite fields*, Math. Comp. 23 (1969), 861-867.

[23] MIGNOTTE, M., *An inequality about factors of polynomials*, Math. Comp. 28 (1974), 1153-1157.

[24] MIOLA, A. & D.Y.Y. YUN, *Computational aspects of Hensel-type univariate polynomial greatest common divisor algorithms*, Proceedings of EUROSAM '74 (ACM SIGSAM Bull. No. 31), 46-54.

[25] MOENCK, R.T., *On the efficiency of algorithms for polynomial factoring*, Math. Comp. 31 (1977), 235-250.

[26] MOSES, J. & D.Y.Y. YUN, *The EZGCD algorithm*, Proceedings of the ACM Annual Conference, August 1973, Atlanta, 159-166.

[27] MUSSER, D.R., *Multivariate polynomial factorization*, J. ACM 22 (1975) 291-308.

[28] MUSSER, D.R., *On the efficiency of a polynomial irreducibility test*, J. ACM 25 (1978), 271-282.

[29] PÓLYA, G. & G. SZEGÖ, *Aufgaben und Lehrsätze aus der Analysis*, (2 Vols), Springer, 1964 (3rd ed.).

[30] RABIN, M.O., *Probabilistic algorithms in finite fields*, SIAM J. Comput. 9 (1980), 273-280.

[31] TRAGER, B.M., *Algebraic factoring and rational function integration*, Proceedings of the 1976 ACM Symposium on Symbolic and algebraic computation, 219-226.

[32] VAN DER WAERDEN, B.L., *Moderne algebra*, Springer, Berlin, 1931.

[33] WANG, P.S. & L.P. ROTHSCHILD, *Factoring multivariate polynomials over the integers*, Math. Comp. 29 (1975), 935-950.

[34] WANG, P.S., *Factoring multivariate polynomials over algebraic number fields*, Math. Comp., 30 (1976), 324-336.

[35] WANG, P.S., *An efficient square-free decomposition algorithm*, ACM SIGSAM Bull. 11 (1977), 4-6.

[36] WANG, P.S., *An improved multivariate polynomial factoring algorithm*, Math. Comp. 32 (1978), 1215-1231.

[37] WANG, P.S. & B.M. TRAGER, *New algorithms for polynomial square-free decomposition over the integers*, SIAM J. Comput. 8 (1979), 300-305.

[38] WEINBERGER, P.J. & L.P. ROTHSCHILD, *Factoring polynomials over algebraic number fields*, ACM Transactions on Mathematical Software 2 (1976), 335-350.

[39] YUN, D.Y.Y., *The Hensel-lemma in algebraic manipulation*, Ph.D. Thesis, MIT, 1974.

[40] YUN, D.Y.Y., *On square-free decomposition algorithms*, Proceedings of the 1976 ACM Symposium on symbolic and algebraic computation, 26-35.

[41] ZASSENHAUS, H., *On Hensel factorization, I*, Journal of number theory 1 (1969), 291-311.

[42] ZASSENHAUS, H., *A remark in the Hensel factorization method*, Math. Comp. 32 (1978), 287-292.

[43] ZIPPEL, R., *Probabilistic algorithms for sparse polynomials*, Proceedings of EUROSAM '79, 216-226.