Centrum voor Wiskunde en Informatica

Centre for Mathematics and Computer Science

M.J. Coster

Some algorithms on addition chains and their complexity

Computer Science/Department of Algorithmics & Architecture Report CS-R9024 June

1990

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

M.J. Coster

Some algorithms on addition chains and their complexity

Computer Science/Department of Algorithmics & Architecture Report CS-R9024 June

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

SOME ALGORITHMS ON ADDITION CHAINS AND THEIR COMPLEXITY

Matthijs J. Coster
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

ABSTRACT.

We consider in this report 12 algorithms, for calculating addition chains, addition sequences and vector addition chains. Some of these algorithms are new. We compare these algorithms on two criteria namely the length of the chains/sequences and the memory usage. To do this we use two new techniques. The first new technique involves a kind of addition chain graph, which is an important tool for studying the memory use in more detail. The second new technique concerns a way to split up the chains and sequences into subsets that can be studied independently. Using number theoretical tools (especially from ergodic theory) we get some new results on upper bounds and average values for the lengths of chains and sequences. Finally we compare these algorithms using some tables. The algorithms are described in detail in this report and an example is included for most of the algorithms.

Key Words & Phrases: Addition Chain, Addition Sequence, Vector Addition Chain, Graph, Algorithm, Fast Exponentiation, RSA, Space- and Time- Complexity, ergodic theory.
1985 AMS Mathematics Subject Classification: 65V05, 68M20.
Computing Reviews code: F21: Number theoretic computation; G4: Algorithm analysis and efficiency.

1. INTRODUCTION

Much research is concerned with speeding up RSA calculations. A lot of work has been aimed at increasing the speed of multiplications. More recently, further improvements have been achieved by applying so called *addition chains*. In RSA applications an exponentiation is done by some modular multiplications using addition chains. For each element in the addition chain a multiplication is done. Hence RSA becomes faster if the addition chain is shorter.

For instance, new RSA-applications (cf. [Fia89] and [CBHMS90]) can take advantage of a generalization called *vector addition chains*. Using this feature the RSA application becomes significantly faster. Olivos gave in [Oli81] a description how such vector addition chains can be constructed. This construction was implemented by

Bos in [Bos90]. Addition chains also have applications in exponentiations of discrete log based systems [ElG85], exponentiations in GF(2^n) (cf. [AMV89]) and factoring algorithms (cf. [Len86], [Wil82]).

This report considers several algorithms for addition chains and vector addition chains. Included are the Binary Algorithm, the Window Algorithm (cf. [Knu81]), another window algorithm (cf. [BoCo89]), the Batch-RSA Algorithm (cf. [Fia89]), the Continued Fraction Algorithm (cf. [B^{3}D89]), the Generalized Continued Fraction Algorithm (cf. [Bos90]), some other algorithms based on the continued fraction algorithm (cf. [B^{3}89]), and the Generalized Window Algorithm (cf. [Yao76] and [Str64]). Apart from these algorithms we consider some new algorithms.

This report consists of 13 sections. In Section 2 we define the notation used in this paper and in Section 3, we give a review of the literature. The goal of this report is to compare these algorithms on applicability in RSA, elliptic curves, etc. We distinguish two criteria. We will consider space complexity and time complexity.

Space complexity will be discussed in Section 5 and Section 6. In Section 5 we will introduce a new way to draw an addition chain graph which is nearly related to the addition chain graph in Knuth, but it has some advantages. Firstly the space complexity can be seen directly and secondly there remains place in the graph to put some additional information, (e.g. the operations which are used in the computer). We prove in this section that the only algorithm which uses 1 memory location is the Binary Algorithm. It is well known that this algorithm does not produce the shortest chains in general. Another result in this section is that the Generalized Continued Fraction Algorithm (cf. [Bos90]) uses only n memory locations to produce a vector addition chain of width n and this is optimal. In Section 6, we calculate the number of memory locations used by the other algorithms.

Time complexity will be discussed in Section 7 and Section 8. We denote for a set of positive integers $\{a_1, \ldots, a_n\}$ by $l_\Theta(a_1, \ldots, a_n)$ the length of the addition sequence produced by Algorithm Θ. Suppose that $a_1 \leq \ldots \leq a_n$. We want to express approximations of $l_\Theta(a_1, \ldots, a_n)$ in terms of n and a_n. A good measure for $l_\Theta(a_1, \ldots, a_n)$ is the number $\rho_\Theta(a_1, \ldots, a_n)$ which is defined by

$$\rho_\Theta(a_1, \ldots, a_n) = \frac{l_\Theta(a_1, \ldots, a_n)}{\log_2 a_n}. \tag{1}$$

In Section 7 we are interested in the upper bound of those numbers $\rho_\Theta(a_1, \ldots, a_n)$. In some of the applications the construction of a doubling is much easier than the construction of other additions. We denote by $l^{\square}_\Theta(a_1, \ldots, a_n)$ the number of doublings in the addition sequence produced by Algorithm Θ and by $l^{+}_\Theta(a_1, \ldots, a_n)$ the number of other additions. We have the identity

$$l_\Theta(a_1, \ldots, a_n) = l^\square_\Theta(a_1, \ldots, a_n) + l^+_\Theta(a_1, \ldots, a_n). \tag{2}$$

In a similar way we define

$$\rho^\square_\Theta(a_1, \ldots, a_n) = l^\square_\Theta(a_1, \ldots, a_n)/\log_2 a_n \tag{3a}$$

and

$$\rho^+_\Theta(a_1, \ldots, a_n) = l^+_\Theta(a_1, \ldots, a_n)/\log_2 a_n. \tag{3b}$$

We define the upper bound $\hat{\rho}_\Theta(\alpha, n)$ of $\rho_\Theta(a_1, \ldots, a_n)$ by

$$\hat{\rho}_\Theta(\alpha,n) = \max_{(a_1, \ldots, a_n)} \frac{l_\Theta(a_1, \ldots, a_n)}{\log_2 a_n}, \tag{4}$$

where the n-tuples $(a_1, \ldots, a_n)$ are taken such that $\alpha \le \log_2 a_n < \alpha + 1$. We will also consider the numbers $\hat{\rho}^\square_\Theta(\alpha, n)$ and $\hat{\rho}^+_\Theta(\alpha, n)$.

In Section 8 we consider the average value for $\rho_\Theta(a_1, \ldots, a_n)$. We define $\bar{\rho}_\Theta(\alpha, n)$ by

$$\bar{\rho}_\Theta(\alpha,n) = \frac{1}{\Sigma} \cdot \sum_{(a_1, \ldots, a_n)} \frac{l_\Theta(a_1, \ldots, a_n)}{\log_2 a_n}, \tag{5}$$

where the sum is taken over all n-tuples $(a_1, \ldots, a_n)$ for which $\alpha \le \log_2 a_n < \alpha + 1$ and Σ is the number of those n-tuples $(a_1, \ldots, a_n)$. Some of the values $\bar{\rho}_\Theta(\alpha, n)$ are calculated using some ergodic theoretical tools. We calculate also the values of $\bar{\rho}^\square_\Theta(\alpha, n)$ and $\bar{\rho}^+_\Theta(\alpha, n)$. In general an arbitrary n-tuples $(a_1, \ldots, a_n)$ satisfies

$$l_\Theta(a_1, \ldots, a_n) \approx \bar{\rho}_\Theta(\alpha, n) \cdot \log_2 a_n.$$

Therefore $\bar{\rho}_\Theta(\alpha, n)$ is a good measure to compare more algorithms. The upper bound $\hat{\rho}_\Theta(\alpha, n)$ is important for calculating running-time approximations in the applications.

In Section 9 we will compare the different algorithms. We make three tables for those comparisons. One table is for the case of the asymptotical values of a_n, a second table is for the case that we consider addition chains with a bounded ($\log a \approx 512$) and a third table is for the case that we consider vector addition chains of width 32 ($n = 32$) with $\log a_{32} \approx 80$.

In Section 10 we give some topics for further research and some open problems which are of large interest for developing the theory of addition chains, addition sequences and vector addition chains.

After the references in Section 11, Section 12 contains an appendix with concrete examples of the algorithms. The last section consists of one table which should be

included in Section 7.3.2, but we improved the readibility by moving this table to the end of the paper.

2. DEFINITIONS AND NOTATION

We use the following notation:

$\mathbf{a}$	a vector
$\mathbf{f}_i$	unit vector $[0, \ldots, 0, 1, 0, \ldots, 0]$
$[x]$	largest integer which does not exceed x.
$\lceil x \rceil$	smallest integer which is at least x.
$]x[$	nearest integer to x.
$\nu(n)$	the number of ones that occur in the binary representation of n
$\log n$	$\log_2 n$
$\lambda(n)$	$[\log_2 n]$
$\ln n$	$\log_e n$
$\#I$	the number of elements of set I.

An *addition chain* for a positive integer a is the set of positive integers $\{b_0, \ldots, b_l\}$ with the following properties:

(*i*) $b_0 = 1$,
(*ii*) for all $1 \le k \le l$ there exist i, j such that $b_k = b_i + b_j$, for $0 \le i, j < k$,
(*iii*) $b_l = a$.

We call the index l the *length* of the addition chain. We denote a shortest addition chain by $L(a)$ and its length by $l(a)$. We denote the subset of $L(a)$ of numbers b for which $b/2$ in $L(a)$ by $L^{\square}(a)$ (the subset of doubles), we denote $L^{+}(a) = L(a) \backslash L^{\square}(a)$ (the subset of other additions). An example of a shortest addition chain for the number 23 is

$$L(23) = \{1, 2, 3, 5, 10, 20, 23\},$$

which has length 6. We have $L^{\square}(23) = \{2, 10, 20\}$, $L^{+}(23) = \{3, 5, 23\}$.

An *addition sequence* for a set of positive integers $\{a_1, \ldots, a_n\}$ is a set of positive integers $\{b_0, \ldots, b_l\}$ with the following properties:

(*i*) $b_0 = 1$,
(*ii*) for all $1 \le k \le l$ there exist i, j such that $b_k = b_i + b_j$, for $0 \le i, j < k$,
(*iii*) for $1 \le m \le n$: $a_m \in \{b_0, \ldots, b_l\}$.

We call n the *width* of the addition sequence and l the *length*. We assume in the rest of the text that $a_1 \le \ldots \le a_n$. We denote the smallest set $\{b_0, \ldots, b_l\}$ by $L(a_1, \ldots, a_n)$. We denote by $L^{\square}(a_1, \ldots, a_n)$ the largest subset of

$L(a_1, \ldots, a_n)$ which consists of doubles ($b/2$ is an element of $L(a_1, \ldots, a_n)$) and $L^+(a_1, \ldots, a_n) = L(a_1, \ldots, a_n) \backslash L^\square(a_1, \ldots, a_n)$ (the subset of other additions). An addition chain is an addition sequence of width 1. An example of an addition sequence is

$$L(6,59) = \{1, 2, 3, 6, 7, 14, 28, 56, 59\},$$

which has width 2 and length 8, $L^\square(6,59) = \{2, 6, 14, 28, 56\}$ and $L^+(6,59) = \{3, 7, 59\}$.

A *vector addition chain* for a vector $\boldsymbol{a}=[a_1, \ldots, a_n]$ is a set of vectors $\{\mathbf{b}_{1-n}, \ldots, \mathbf{b}_l\}$ with the following properties:

(*i*) $\mathbf{b}_{i-n}$ is a unit vector $\mathbf{f}_i$ for $1 \le i \le n$,
(*ii*) for all $1 \le k \le l$ there exist i, j such that $\mathbf{b}_k = \mathbf{b}_i + \mathbf{b}_j$, for $1-n \le i, j < k$.
(*iii*) $\mathbf{b}_l = \mathbf{a}$.

We call n the *width* of the vector addition chain and l the *length*. In order to calculate $\mathbf{x}^{\mathbf{a}} = x_1^{a_1} \ldots x_n^{a_n}$ from $\mathbf{x} = [x_1, \ldots, x_n]$, we construct a vector addition chain $\{\mathbf{b}_{1-n}, \ldots, \mathbf{b}_l\}$ for $\mathbf{a}= [a_1, \ldots, a_n]$. In a similar way as above we define $L[\mathbf{a}]$, $L^\square[\mathbf{a}]$ and $L^+[\mathbf{a}]$. An example of a vector addition chain is

$$L[6,59] = \{[1, 0], [0, 1], [0, 2], [0, 4], [0, 8], [1, 8],$$
$$[2, 16], [2, 17], [4, 34], [6, 51], [6, 59]\},$$

which has width 2 and length 9 and $L^\square[6,59] = \{[0, 2], [0, 4], [0, 8], [2, 16], [4, 34]\}$ and $L^+[6,59] = \{[1, 8], [2, 17], [6, 51], [6, 59]\}$. The calculation of X^6Y^{59} from X and Y can be done via

$$\{X, Y, Y^2, Y^4, Y^8, XY^8, X^2Y^{16}, X^2Y^{17}, X^4Y^{34}, X^6Y^{51}, X^6Y^{59}\}.$$

For more detailed definitions we refer to [Knu81].

We use also the following maps and functions:

$L[\mathbf{a}]$	a shortest vector addition chain for $[\mathbf{a}]$.
$L^\square[\mathbf{a}]$	the subset of vectors $\mathbf{b}$ in $L[\mathbf{a}]$ for which $\mathbf{b}/2$ is in $L[\mathbf{a}]$.
$L^+[\mathbf{a}]$	$L[\mathbf{a}] \backslash L^\square[\mathbf{a}]$.
$L^{(n+k)}[\mathbf{a}]$	a shortest vector addition chain containing $a_1, a_2, \ldots, a_n$ using only $n+k$ memory locations ($k \ge 0$).
$l(a_1, a_2, \ldots, a_n)$	length of a shortest addition sequence containing $a_1, a_2, \ldots, a_n$.
$l^\square(a_1, a_2, \ldots, a_n)$	$\#L^\square(a_1, a_2, \ldots, a_n)$.

$l^+(a_1, a_2, \ldots, a_n)$	$\#L^+(a_1, a_2, \ldots, a_n)$.
$l[\mathbf{a}]$	length of a shortest vector addition chain for $[a_1, a_2, \ldots, a_n]$.
$l^{\square}[\mathbf{a}]$	$\#L^{\square}[\mathbf{a}]$.
$l^+[\mathbf{a}]$	$\#L^+[\mathbf{a}]$.
$I(a_1, a_2, \ldots, a_n)$	is a map discussed in Section 4,
$T(a_1, a_2, \ldots, a_n)$	is a map discussed in Section 4, The maps I and T satisfy $\max(T(a_1, a_2, \ldots, a_n)) < a_n$ and $L(a_1, a_2, \ldots, a_n) = L(T(a_1, a_2, \ldots, a_n)) \cup I(a_1, a_2, \ldots, a_n)$.
$\rho(a_1, a_2, \ldots, a_n)$	$l(a_1, a_2, \ldots, a_n)/\log a_n$.
$\rho^{\square}(a_1, a_2, \ldots, a_n)$	$l^{\square}(a_1, a_2, \ldots, a_n)/\log a_n$.
$\rho^+(a_1, a_2, \ldots, a_n)$	$l^+(a_1, a_2, \ldots, a_n)/\log a_n$.
μ	$2^{1/\rho}$.
μ_k	μ^{-k}.
$\hat{l}(\alpha, n)$	$\max\{l(a_1, \ldots, a_n)\}$ such that $\lambda(\max\{a_1, \ldots, a_n\}) = \alpha$, abbreviated as $\hat{l}$.
$\overline{l}(\alpha, n)$	mean value of $l(a_1, a_2, \ldots, a_n)$ such that $\lambda(\max\{a_1, \ldots, a_n\}) = \alpha$, $\overline{l}$ for short.
$\hat{\rho}(\alpha, n)$	$\max\{\rho(a_1, a_2, \ldots, a_n)\}$ such that $\lambda(a_n) = \alpha$. We abbreviate this notation to $\hat{\rho}$.
$\overline{\rho}(\alpha, n)$	average value of $\rho(a_1, a_2, \ldots, a_n)$ such that $\lambda(a_n) = \alpha$. We abbreviate this notation to $\overline{\rho}$.

We will consider some algorithms. We denote the algorithms by an index. We distinguish the following algorithms.

$L_0(a)$	the addition chain for a constructed by the Binary Algorithm.
$L_{\mathrm{MOA}}(a)$, $L_{\mathrm{MOB}}(a)$	the addition chains for a constructed by the Algorithms of Morain and Olivos.
$L_{\mathrm{K}(k)}(a)$	the addition chain for a constructed by the k-Window Algorithm described in [Knu81].

$L_{W(n)}(a)$	the addition chain for a constructed by the Large Window Algorithm described in [BoCo89]; n expresses the number of windows.
$L_S(a)$	the addition sequence which is based on the Addition-Subtraction Algorithm $L_S(a, b)$.
$L_{II}(a, b)$	the addition sequence containing a, b which can be constructed by Algorithm II.
$L_S(a, b)$	the addition sequence containing a, b which can be constructed by an Addition-Subtraction Algorithm.
$L_{III}(a, b, c)$	the addition sequence containing a, b, c which can be constructed by Algorithm III.
$L_I(a_1, a_2, \ldots, a_n)$	the addition sequence containing $a_1, a_2, \ldots, a_n$ constructed by Algorithm I (cf. [Bos89]).
$L_F(a_1, a_2, \ldots, a_n)$	the addition sequence containing $a_1, a_2, \ldots, a_n$ constructed by the generalized Algorithm of Fiat (cf. [Fia89]).
$L_{Y(k)}(a_1, a_2, \ldots, a_n)$	the addition sequence containing $a_1, a_2, \ldots, a_n$ constructed by the Algorithm of Yao (cf. [Yao76]). Here k expresses the width of the windows.
$L_{St(k)}(a_1, a_2, \ldots, a_n)$	the addition sequence containing $a_1, a_2, \ldots, a_n$ constructed by the Algorithm of Straus (cf. [Str64]). Here k expresses the width of the windows.
$L_B[a_1, a_2, \ldots, a_n]$	the vector addition chain for $[a_1, a_2, \ldots, a_n]$ constructed by the Batch-RSA Algorithm of Fiat (cf. [Fia89]).
$L_{B,I}[a_1, a_2, \ldots, a_n]$	the vector addition chain for $[a_1, a_2, \ldots, a_n]$ constructed by a combination of the Batch-RSA Algorithm in which Algorithm I is applied.

Related to these algorithms we have some functions and maps. The most important functions related to algorithm Θ (Θ = 0, K(k), W(n), S, I, II, III, F, Y(k), St(k), B) are:

$l_\Theta(a_1, \ldots, a_n)$	the length of the addition sequence $L_\Theta(a_1, \ldots, a_n)$.
$l_\Theta[a_1, \ldots, a_n]$	the length of the vector addition chain $L_\Theta[a_1, \ldots, a_n]$.
$m_\Theta(a_1, \ldots, a_n)$	the number of memory locations needed for constructing the

	related addition sequence $L_\Theta(a_1, \ldots, a_n)$.
$m_\Theta[a_1, \ldots, a_n]$	the number of memory locations needed for constructing the related vector addition chain $L_\Theta[a_1, \ldots, a_n]$.
$\hat{\rho}_\Theta(\alpha, n)$ $\bar{\rho}_\Theta(\alpha, n)$ $\hat{\rho}^{\square}{}_\Theta(\alpha, n)$ $\bar{\rho}^{\square}{}_\Theta(\alpha, n)$ $\hat{\rho}^{+}{}_\Theta(\alpha, n)$ $\bar{\rho}^{+}{}_\Theta(\alpha, n)$	the definitions are clear.

For two algorithms Θ_1, Θ_2 we consider by Algorithm $L_{\Theta_1,\Theta_2}(a_1, \ldots, a_n)$ the algorithm produced by Algorithm Θ_1 in which Algorithm Θ_2 is used for producing sub-chains or -sequences. Algorithm $L_{B,I}[a_1, a_2, \ldots, a_n]$ is an example of such an combination of two algorithms. In the sequel we will consider other examples.

Algorithm Θ is called *well conditioned for constructing sequences* if there exist numbers M_n such that $m_\Theta(a_1, \ldots, a_n) \leq M_n$ for each n-tuple $(a_1, \ldots, a_n)$. Otherwise the algorithm is called *ill conditioned for constructing sequences.* Algorithm Θ is called *well conditioned for constructing vector addition chains* if there exist numbers M_n such that $m_\Theta[a_1, \ldots, a_n] \leq M_n$ for each n-tuple $(a_1, \ldots, a_n)$. Otherwise the algorithm is called *ill conditioned for constructing vector addition chains*. If no confusion is possible we abbreviate this to *well conditioned* and *ill conditioned.*

For each addition sequence algorithm we define the *reverse of the algorithm* as the algorithm which constructs the *reverse vector addition chain.* This will be explained in Section 5. The sets $L_\Theta(a_1, \ldots, a_n)$ and $L_\Theta[a_1, \ldots, a_n]$ denote the addition sequence and vector addition chain constructed by Algorithm Θ. By the theorem of Olivos [Oli81] we have that $l_\Theta[a_1, \ldots, a_n] = l_\Theta(a_1, \ldots, a_n)+n-1$. There does not exist any relation between $m_\Theta(a_1, \ldots, a_n)$ and $m_\Theta[a_1, \ldots, a_n]$. While Algorithm Θ is well conditioned for constructing addition sequences, it can be ill conditioned for constructing vector addition chains. By modifying small parts of Algorithm Θ it is possible to make Algorithm Θ well conditioned for constructing vector addition chains. In order to distinguish the unmodified and modified algorithm we denote the algorithms by $L_{\Theta\alpha}$ and $L_{\Theta\beta}$. But if no confusion is possible we omit α and β.

In the case that we consider an addition chain algorithm we also distinguish $L_\Theta(a)$ and $L_\Theta[a]$. We denote by $L_\Theta[a]$ *reverse chain* constructed by the reverse of the algorithm.

3. A BRIEF REVIEW OF THE LITERATURE

Much is known about bounds for addition chains. For instance, it is known that

$$\log a + \log \nu(a) - 2.13 \leq l(a). \tag{6}$$

This bound is from [Sch75]. [Bra39] gives an upper bound of

$$l(a) \leq \log a + \log a / \log \log a + o(\log a / \log \log a). \tag{7}$$

This bound is theoretical. In practise we can't get this bound for larger numbers. The *k-window*-method described in [Knu69] and [BoCo89] gives a worse upper bound:

$$l(a) \leq \log a + \tfrac{1}{k} \cdot \log a + 2^{k-1} - k - 1, \tag{8}$$

which is optimal if $k^2 \cdot 2^k \approx 2 \cdot \log a / \ln 2$. Less is known about addition sequences and vector addition chains. Straus gives in [Str64] an upper bound for vector addition chains:

$$l(a_1, \ldots, a_n) \leq \log a_n + \tfrac{1}{k} \cdot \log a_n + 2^{nk} - n - 1, \tag{9a}$$

which is optimal if $k^2 \cdot 2^{nk} \approx \log a_n / (n \cdot \ln 2)$. In [Yao76] Yao gives an upper bound for addition sequences:

$$l(a_1, \ldots, a_n) \leq \log a_n + \tfrac{n}{k} \cdot \log a_n + n \cdot 2^k - k - 1, \tag{9b}$$

which is optimal if $k^2 \cdot 2^k \approx \log a_n / \ln 2$. Here a_n is the largest number in the sequence. Olivos proved in [Oli81] that there exists a relation between the length of a vector addition chain and the length of the associated addition sequence. This relation has the following form: *the length of a vector addition chain is equal to the length of the associated addition sequence plus the width of the addition sequence minus one.* In formula we have

$$l[a_1, \ldots, a_n] = l(a_1, \ldots, a_n) + n - 1. \tag{10}$$

4. AN OVERVIEW OF THE ALGORITHMS

Before we deal with the algorithms, we will first introduce some functions which are good tools to describe most of the algorithms, and which will play an important role in the rest of this paper. Using these functions we will describe $L(a_1, \ldots, a_n)$ on induction. Suppose that $L(a'_1, \ldots, a'_n)$ was calculated for all n-tuples $(a'_1, \ldots, a'_n)$ with $0 \leq a'_1 \leq a'_2 \leq \ldots \leq a'_n < a_n$. For a special choice of $a'_1, \ldots, a'_n$ we have $L(a_1, \ldots, a_n) = L(a'_1, \ldots, a'_n) \cup \{c_1, \ldots, c_p\}$.

10 Some algorithms on addition chains and their complexity.

We introduce for Algorithm Θ and for $0 \leq a_1 \leq \ldots \leq a_n$ the mapping $T_\Theta(a_1, \ldots, a_n) = (a'_1, \ldots, a'_n)$ with the properties

(*i*) $0 \leq a'_1 \leq a'_2 \leq \ldots \leq a'_n < a_n$,

(*ii*) $a_i = \sum \tau_{ij} a'_j$, for some non-negative integers τ_{ij},

and the mapping $I_\Theta(a_1, \ldots, a_n) = \{c_1, \ldots, c_p\}$. The elements of $I_\Theta(a_1, \ldots, a_n)$ will belong to $L_\Theta(a_1, \ldots, a_n)$. Now we construct $L_\Theta(a_1, \ldots, a_n)$ on induction:

$$L_\Theta(a_1, \ldots, a_n) = L_\Theta(T_\Theta(a_1, \ldots, a_n)) \cup I_\Theta(a_1, \ldots, a_n). \tag{9}$$

4.1 The Binary Algorithm. $L_0(a)$, (see [Knu81] pp. 441-442).

This algorithm is well-known. For the sake of completeness we will describe this algorithm. In fact we consider two algorithms which can be derived from each other using the reverse method (cf. [Knu81], pp. 460-462). We write $a = \Sigma\, 2^{r_j}$, the binary representation. For each r_j we need besides the doubling an extra multiplication.

The first algorithm ($L_{0\alpha}$) reads:

$$T_{0\alpha}(a) = [a/2] \text{ and } I_{0\alpha}(a) = \begin{cases} \{a\} & \text{if } a \text{ is even,} \\ \{a, a-1\} & \text{if } a \text{ is odd.} \end{cases}$$

We get for $a > 2$ by the inductive construction:

$$L_{0\alpha}(a) = L_{0\alpha}(T_{0\alpha}(a)) \cup I_{0\alpha}(a).$$

We have, e.g., $L_{0\alpha}(23) = L_{0\alpha}(11) \cup \{22, 23\} = L_{0\alpha}(5) \cup \{10, 11, 22, 23\} = L_{0\alpha}(2) \cup \{4, 5, 10, 11, 22, 23\} = \{1, 2, 4, 5, 10, 11, 22, 23\}$.

The second algorithm ($L_{0\beta}$) reads:

$$L_{0\beta}(a) = \{1, 2, 4, \ldots, 2^{\lambda(a)} = 2^{r_1}, 2^{r_1} + 2^{r_2}, \ldots, \Sigma\, 2^{r_j} = a\}.$$

This algorithm has the disadvantage that it is ill conditioned as will explained in Section 5. Nevertheless we mention this algorithm since the reverse chain (Algorithm $L_{0\beta}[a]$) is well conditioned.
We have for instance $L_{0\beta}(23) = \{1, 2, 4, 8, 16, 20, 22, 23\}$, and $L_{0\beta}[23] = L_{0\alpha}(23)$. Verifying this will be left to the reader (after reading Section 5).

4.2 The Subtraction Algorithm of Morain and Olivos.

$L_{MO}(a)$, (see [MO89]).

This algorithm is worthfull in the case that subtractions are as expensive as additions. An *addition-subtraction* chain for a positive integer a is a set of positive integers $\{b_0, \ldots, b_l\}$ which satisfies the properties of an addition chain, but for which in property (*ii*) b_k can be written as $b_k = \pm b_i + \pm b_j$.

We write a number in binary representation. Let $a = \sum_{i=0}^{s} a_i 2^i$. Let $a_{-1} = a_{-2} = a_{s+1} = a_{s+2} = 0$. The algorithms are described in [MO89] completely. We give here only an abstract.

For the first algorithm we consider the blocks consisting of n 1-'s for $n > 2$. These can be replaced by 1 followed up by $n-1$ zeroes and a '-1'. Instead of n additions of 1 we get one addition and one subtraction. Algorithm L_{MOA} reads:

Put (1, 0, 0) in (M, M^+, M^-)
For i in {0 ... s}
    If $a_i = 0$: Put $2M$ in M
    If $a_i = 1$: If $a_{i-1} = 0$: If $a_{i+1} = 0$: Put $M^+ + M$ in M^+
                  If $a_{i+1} = 1$: Put $M^- + M$ in M^-
                  Put $2M$ in M
                  If $a_{i-1} = 1$: Put $2M$ in M
                  If $a_{i+1} = 0$: Put $M^+ + M$ in M^+
Put $M^+ - M^-$ in *result*

In the second algorithm Morain and Olivos also consider isolated 0's. The addition which was the result of the block of 1's before the 0 can be combined with the subtraction which has to be done in the block of 1's after the 0. The end of a block is marked by 00 or 010. We introduce the boolean *sb* (subtraction-block) which will indicate wether we are in such a block of 1's and isolated 0's. Algorithm L_{MOB} reads:

Put (1, 0, 0) in (M, M^+, M^-)
Put FALSE in *sb*
For i in {0 ... s}
    If *sb* = FALSE If $a_i = 1$: If $a_{i+1} = 0$: Put $M^+ + M$ in M^+
                                  If $a_{i+1} = 1$: Put $M^- + M$ in M^-
                                  Put TRUE in *sb*
    If *sb* = TRUE If $a_i = 0$: If $a_{i+1} = 0$: Put $M^+ + M$ in M^+

Put FALSE in *sb*
If $a_{i+1} = 1$: If $a_{i+2} = 0$: Put $M^+ + M$ in M^+
Put FALSE in *sb*
If $a_{i+2} = 1$: Put $M^- + M$ in M^-

Put $2M$ in M
Put $M^+ - M^-$ in *result*

An example of this algorithm will be given in Section 12.1.

4.3 THE SMALL WINDOW ALGORITHM. $L_{K(k)}(a)$, (see [Knu81] p. 451).

We write a number in binary representation and split it in pieces (*windows*), in such a way that each window contains either only zeroes or the window represents an odd number smaller then 2^k. We distinguish again two algorithms. The first algorithm is well conditioned. The second algorithm is ill conditioned but its reverse (which is the first algorithm) is well conditioned. In fact $L_0(a) = L_{K(1)}(a)$.

The first algorithm ($L_{K(k)\alpha}$) reads:

$$T_{K(k)\alpha}(a) = \begin{cases} a/2 & \text{if } a \text{ is even,} \\ [a/2^k], a - 2^k \cdot [a/2^k] & \text{if } a \text{ is odd.} \end{cases}$$

and

$$I_{K(k)\alpha}(a) = \begin{cases} \{a\} & \text{if } a \text{ is even,} \\ \{2 \cdot [a/2^k], 4 \cdot [a/2^k], \ldots, 2^k \cdot [a/2^k], a\} & \text{if } a \text{ is odd.} \end{cases}$$

We get

$$L_{K(k)\alpha}(a) = \{1, 2, 3, 5, 7, \ldots, 2^k - 1, a\} \qquad \text{if } a < 2^k$$

$$L_{K(k)\alpha}(a) = L_{K(k)\alpha}(T_{K(k)\alpha}(a)) \cup I_{K(k)\alpha}(a) \qquad \text{if } a \geq 2^k.$$

We have for instance $L_{K(2)\alpha}(15) = \{1, 2, 3, 6, 12, 15\}$. A more detailed example can be found in Section 12.2.

For the second algorithm we need the following "window"-representation of a, namely $a = \Sigma\, w_j 2^{r_j}$, where $0 < w_j < 2^k$ and w_j odd. We define, for odd indices i, $g_i = \sum_{w_j = i} 2^{r_j}$. Then we get

$$L_{K(k)\beta}(a) = \{1, 2, 4, \ldots, 2^{\lambda(a)}, \ldots, g_1, \ldots, g_{2^k-1}, g_{2^k-1} + g_{2^k-3},$$

$$\ldots, \sum_{j=1}^{2^{k-1}-1} g_{2j+1}, \ldots, \sum_{j=1}^{2^{k-1}-1} jg_{2j+1}, \sum_{j=1}^{2^{k-1}-1} 2jg_{2j+1}, \sum_{j=1}^{2^{k-1}} (2j-1)g_{2j-1} = a\}.$$

We have for instance $L_{K(2)\beta}(15) = \{1, 2, 4, 5, 10, 15\}$. A more detailed example of this algorithm can be found in Section 12.2.

Improvement. Let w_0 be the most significant k-window. If $\lambda(w_0) < k-1$ then we make an improvement by choosing the first window $w_0' = 2^h w_0$, where $h = k-\lambda(w_0)$. We have $w_0' = (2^k-1) + (w_0' - 2^k+1)$. These elements are both elements of $\{1, 3, 5, 7, \ldots, 2^k-1\}$.

4.4 The Large Window Algorithm. $L_{W(n)}(a)$, (see [BoCo89]).

In this algorithm n denotes the number of windows and we denote by k the size of the windows. We have $k \approx \lceil (\log a)/n \rceil$. The windows are such large that the length of the chain would be such large that the algorithm should be uninteresting, if all the numbers $1, 2, 3, 5, 7, \ldots, 2^k-1$ were in the chain. On the other hand we do not need all those numbers. Only a few are necessary, say $w_1, \ldots, w_n$. The first part of the addition chain is an addition sequence $L_\Theta(w_1, \ldots, w_n)$. There are different ways to choose the windows. We prefer to choose the windows in such a way that

(*i*) The number of windows is as small as possible,
(*ii*) The most significant window represents the largest number,
(*iii*) $\prod w_i$ is minimal.

In [BoCo89] a method is described which finds such windows and which is linear in $\log a$. In Section 12.3 we give an example of this algorithm.

4.5 Algorithm II. $L_{II}(a, b)$.

The reason why we treat this algorithm is the fact that for this algorithm we can calculate quite easily the upper bound of the length of the sequence. The proof of this upper bound will give a good insight how such upper bounds can be calculated. Nevertheless since this algorithm is ill conditioned for constructing vector addition chains, it will not find its way into practice. The algorithm can be given by a description of $T_{II}(a, b)$ and $I_{II}(a, b)$. The Algorithm reads

$$L_{II}(a, b) = \{1, 2, 3, 4, 5, 6, 7\} \qquad \text{if } b < 8$$

$$L_{II}(a, b) = L_{II}(T_{II}(a, b)) \cup I_{II}(a, b) \qquad \text{if } b \geq 8.$$

For a/b in $[0, 1)$ we distinguish the following cases:

① If $0 \le a/b < 0.3$ then $I_{II}(a, b)=\{2[b/4], 4[b/4], b\}$
and $T_{II}(a, b)=(a, [b/4])$,

② If $0.3 \le a/b < 0.7$ then $I_{II}(a, b)=\{b\}$ and $T_{II}(a, b)=(a, b-a)$,

③ If $0.7 \le a/b < 0.78$ then $I_{II}(a, b)=\{2(b-a), a, b\}$
and $T_{II}(a, b)=(3a-2b, b-a)$,

④ If $0.78 \le a/b < 0.8$ then $I_{II}(a, b)=\{2(b-a), 3(b-a), a, b\}$
and $T_{II}(a, b)=(4a-3b, b-a)$,

⑤ If $0.8 \le a/b < 0.84$ then $I_{II}(a, b)=\{2(b-a), 4(b-a), a, b\}$
and $T_{II}(a, b)=(5a-4b, b-a)$,

⑥ If $0.84 \le a/b < 1$ then $I_{II}(a, b)=\{2[a/8], 4[a/8], 8[a/8], a, b\}$
and $T_{II}(a, b)=(b-a, [a/8])$.

4.6 THE ADDITION-SUBTRACTION ALGORITHM. $L_S(a)$ and $L_S(a, b)$.

Assuming that subtractions are as expensive as additions (e.g. elliptic curves, see [MO88]) we can consider algorithms in which appear subtractions. Such an algorithm is for instance the modified continued fraction algorithm. We get the following algorithm for width 2. $T_S(a, b) = (|b-ra|, a)$ and $I_S(a, b) = \{2a, \dots, ra, b\}$, where $\{2, \dots, r\}$ is an addition chain for r constructed by any algorithm and $r = b/a-[b/a+1/2]$. Finally we get $L_S(a, b) = L_S(T_S(a, b)) \cup I_S(a, b)$. Subtraction is needed each time that $b < ra$.
In the sequel we use two algorithms $L_S(a)$ which differ in the choice of the algorithm for constructing the addition chain for r. We call these algorithms $L_{S,0}(a)$ if the addition chain for r is construct by the Binary Algorithm and $L_{S,K(2)}(a)$ if the addition chain for r is construct by the Small Window Algorithm. The algorithm $L_S(a)$ is deduced from $L_S(w_1, w_2)$, using algorithm $L_{W(2)}(a)$ (in fact this is Algorithm $L_{W(2),S}(a)$).

4.7 ALGORITHM III. $L_{III}(a, b, c)$.

This algorithm is comparable to Algorithm II. It is a straightforward calculation to find a upper bound, and the algorithm is ill conditioned. The algorithm can be given by describing of $T_{III}(a, b, c)$ and $I_{III}(a, b, c)$. We have

$$L_{III}(a, b, c) = \{1, 2, 3, 4, 5, 6, 7\} \qquad \text{if } c < 8$$

$$L_{III}(a, b, c) = L_{III}(T_{III}(a, b, c)) \cup I_{III}(a, b, c) \qquad \text{if } c \ge 8.$$

Assuming that $a \le b \le c$, we distinguish the following cases:

① $I_{III}(a, b, c)=\{c\}$ and $T_{III}(a, b, c)=(a, c-b, b)$,
② $I_{III}(a, b, c)=\{2[c/2], c\}$ and $T_{III}(a, b, c)=(a, b, [c/2])$,
③ $I_{III}(a, b, c)=\{b, c\}$ and $T_{III}(a, b, c)=(a, c-b, b-a)$,
④ $I_{III}(a, b, c)=\{2[b/2], b, c\}$ and $T_{III}(a, b, c)=(a, [b/2], c-b)$,
⑤ $I_{III}(a, b, c)=\{2[a/2], a, b, c\}$ and $T_{III}(a, b, c)=([a/2], b-a, c-b)$,
⑥ $I_{III}(a, b, c)=\{2[a/4], 4[a/4], a, b, c\}$ and $T_{III}(a, b, c)=([a/4], b-a, c-b)$.

The numbers ① ... ⑥ correspond to the 6 areas in Figure 1. The exact areas can be obtained from the author.

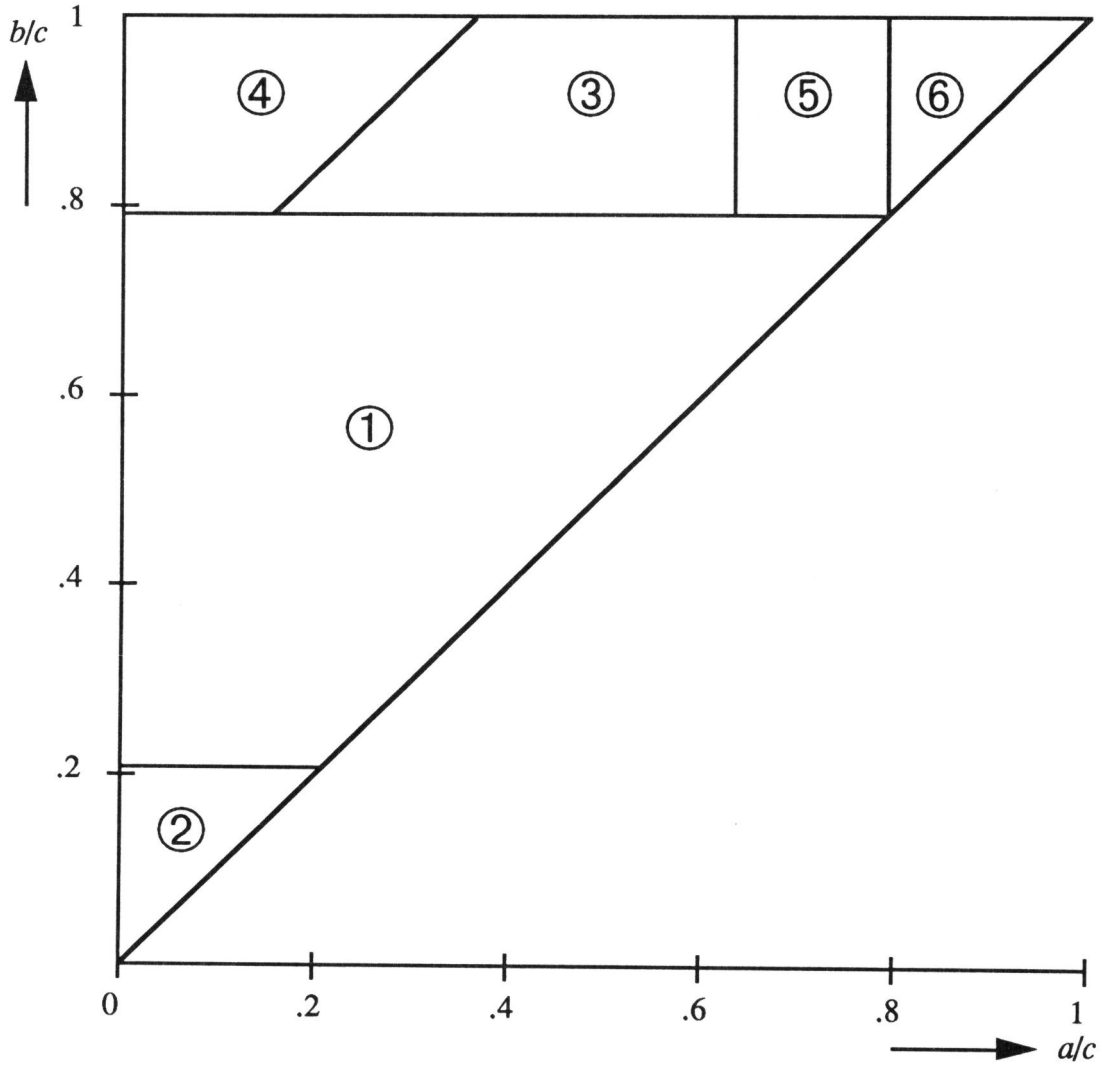

Figure 1 shows the area-division of Algorithm III.

4.8 ALGORITHM I. $L_I(a_1, \ldots, a_n)$ and $L_I[a_1, \ldots, a_n]$, (see [Bos90]).

This algorithm is a generalization of the continued fraction algorithm described in [B^{3}D89]. In that paper only sequences of width 2 were obtained. This algorithm is its generalization. We distinguish two algorithms. The first algorithm is an addition sequence algorithm, the second algorithm is the vector addition chain algorithm which was considered in [Bos90].

The first algorithm reads: Put $[a_n / a_{n-1}] = r$. Then $T_{I\alpha}(a_1, \ldots, a_n) = (a_n - ra_{n-1}, a_1, \ldots, a_{n-1})$ and $I_{I\alpha}(a_1, \ldots, a_n) = \{2a_{n-1}, \ldots, ra_{n-1}, a_n\}$, where $\{2, \ldots, r\}$ is a chain for r constructed by any algorithm. To get the algorithm in [Bos90] we have to choose the Binary Algorithm. In fact this is Algorithm $I_{I,0\alpha}(a_1, \ldots, a_n)$, (see Section 4.5).

We get

$$L_{I\alpha}(a_1, \ldots, a_n) = \{1, 2\} \qquad \text{if } a_n \leq 2,$$

$$L_{I\alpha}(a_1, \ldots, a_n) = \qquad (10)$$

$$L_{I\alpha}(T_{I\alpha}(a_1, \ldots, a_n)) \cup I_{I\alpha}(a_1, \ldots, a_n) \qquad \text{if } a_n > 2.$$

For the second algorithm, put $[a_n / a_{n-1}] = r = 2^{i_1} + \ldots + 2^{i_k}$, with $i_1 < i_2 < \ldots < i_k$. Then $I_{I\beta}(a_1, \ldots, a_n) = \{2a_{n-1}, 4a_{n-1}, \ldots, 2^{i_k}a_{n-1}, a_n - (2^{i_1} + \ldots + 2^{i_k-1})a_{n-1}, a_n - (2^{i_1} + \ldots + 2^{i_k-2})a_{n-1}, \ldots, a_n - 2^{i_1}a_{n-1}, a_n\}$ and $T_{I\beta}(a_1, \ldots, a_n) = (a_n - ra_{n-1}, a_1, \ldots, a_{n-1})$. We get the same construction of $L_{I\beta}$ as in (10). A nice property of Algorithm Iβ is that the algorithm is well conditioned for constructing vector addition chains, and more than that: *the number of memory locations used in the algorithm is equal to the width of the vector addition chain.* We use the Binary Algorithm 0β for constructing sets $I_{I\beta}(a_1, \ldots, a_n)$. If we use other algorithms to construct this set then probably the number of memory locations will be larger. Comparable to Section 4.5, we will consider in the sequel the algorithms $I_{I,0\beta}(a_1, \ldots, a_n)$ and $I_{I,K(2)\beta}(a_1, \ldots, a_n)$.

An example of this algorithm will be given in Section 12.4.

Note. It is possible to make some improvements to the algorithm without increasing the memory usage. For instance the following modification implies in some cases a smaller sequence: if $a_n = a_i + a_j$ for some i and $j < n$ then: $I_I(a_1, \ldots, a_n) = \{a_n\}$ and $T_I(a_1, \ldots, a_n) = (0, a_1, \ldots, a_{n-1})$. A disadvantage is that the algorithm is slower, since each step will contain more comparisons. We will return to this point

at the end of Section 5.

4.9 THE ALGORITHM OF FIAT. $L_F(a_1, \ldots, a_n)$, (see [Fia89]).

In [FIA89] Fiat mentions in the final notes an addition chain algorithm for the width 2. Here we will discuss the algorithm in its general state. We need some definitions first (see [Fia89]). We define $a \cap b$ as the bit-wise AND-operation, a' as the bit-wise NOT-operation. For the sequence $L_F(a, b)$ we consider c = $a \cap b$, $d = a \cap b'$ and $e = a' \cap b$. We get $a = c + d$ and $b = c + e$. Finally we get $L_F(a, b) = \{1, 2, \ldots, 2^{\lambda(b)}, \ldots, c, \ldots, d, \ldots, e, a, b\}$. In the general case we construct the numbers c_k for $0 \leq k \leq 2^n-2$ as follows. Let $[\kappa_1 \ldots \kappa_n]$ be the binary representation of k. Let $c_k = \bigcap_{i=1}^{n} a_i^{(\kappa_i)}$, where $a^{(0)} = a$ and $a^{(1)} = a'$. (i.e. $c_0 = \bigcap_{i=1}^{n} a_i$, $c_1 = \bigcap_{i=1}^{n-1} a_i \cap a_n'$ and $c_{2^n-2} = a_1 \cap \bigcap_{i=2}^{n} a_i'$). We get $a_i = \sum_{\substack{j=1 \\ j=[\kappa_1 \ldots \kappa_n] \\ \kappa_i = 0}}^{2^{n-1}} c_j$. We get $L_F(a_1, \ldots, a_n) = \{1, 2, \ldots, 2^{\lambda(a_n)}, \ldots, c_0, \ldots, c_{2^n-2}, \ldots, a_1, \ldots, a_n\}$.

An example of this algorithm can be found in Section 12.5.

4.10 THE GENERALIZED WINDOW ALGORITHMS.

The following two algorithms are two examples of addition sequence algorithms based on division into windows. The Algorithm of Yao is better for small values of $\lambda(a_n)$, while the Algorithm of Straus is better for larger values. For both algorithms we split **a** into windows $\mathbf{w}_j$. Let $D = 2^k$.We construct window coordinates w_{ij} for the coordinates a_i of **a** by defining

$$a_i = \sum_{j=0}^{[\lambda(a_i)/k]} w_{ij}D^j, \text{ where } 0 \le w_{ij} < D \text{ and } 1 \le i \le n.$$

We define the window vectors $\mathbf{w}_j$ by $\mathbf{w}_j = [w_{1j}, \ldots, w_{nj}]$ for $0 \le j \le [\lambda(a_n)/k]$. Hence $\mathbf{a} = \sum \mathbf{w}_j \cdot D^j$.

4.10.1 THE ALGORITHM OF STRAUS. $L_{St(k)}[a_1, \ldots, a_n]$, (see [Str64]).

Using this algorithm, Straus was able to find an upper bound for vector addition chains. (see Section 3). In fact the algorithm is a generalization of $L_{K(k)}$. We first construct the set $V = \{\mathbf{v}_j\}$ of all vectors $\mathbf{v}_j = [v_{1j}, \ldots, v_{nj}]$ with $0 \le v_{ij} < D$. Notice that the vectors $\mathbf{w}_j$ belong to the set V. We construct the vector addition chain by

$$L_{St(k)}[a_1, \ldots, a_n] = \{\mathbf{v}_1, \ldots, \mathbf{v}_{2^{kn}}, 2\cdot\mathbf{w}_{[\lambda(a_n)/k]}, \ldots, 2^k\cdot\mathbf{w}_{[\lambda(a_n)/k]},$$
$$2^k\cdot\mathbf{w}_{[\lambda(a_n)/k]} + \mathbf{w}_{[\lambda(a_n)/k]-1}, 2^{k+1}\cdot\mathbf{w}_{[\lambda(a_n)/k]} + 2\cdot\mathbf{w}_{[\lambda(a_n)/k]-1}, \ldots, \mathbf{a}\}.$$

In Section 12.6.1 we give an example of this algorithm.

Note. As a natural extension of the Window Algorithm of Straus we consider the Generalized Large Window Algorithm. This algorithm is comparable to the Large Window Algorithm. First we calculate an optimal *vector addition sequence* for the window vectors $\{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$. Then we construct a vector addition chain as above. We do not consider vector addition sequences in this paper. We did not find a reference in the literature how vector addition sequences can be constructed optimally.

4.10.2 THE ALGORITHM OF YAO. $L_{Y(k)}[a_1, \ldots, a_n]$, (see [Yao76]).

This algorithm differs to the Algorithm of Straus only in the construction of the set V. Nevertheless we give the algorithm since the number of memory locations is smaller. We consider two algorithms one algorithm which construct an addition sequence and its reverse which construct a vector addition chain.

Let $S_{il} = \sum_{w_{ij}=l} D^j$, for $1 \le i \le n$, $1 \le l < D$. Hence $\sum_{l=1}^{2^k-1} l \cdot S_{il} = a_i$. The algorithm reads:

$$L_{Y(k)}(a_1, \ldots, a_n) = \{1, 2, 2^2, \ldots, 2^{k\cdot[\lambda(a_n)/k]}, \ldots, S_{1,1}, \ldots, S_{1,2}, \ldots, S_{1,2^k-1},$$
$$S_{1,2^k-1} + S_{1,2^k-2}, \ldots, S_{1,2^k-1} + \ldots + S_{1,1}, 2\cdot S_{1,2^k-1} + \ldots + 2\cdot S_{1,2} + S_{1,1},$$
$$3\cdot S_{1,2^k-1} + \ldots + 3\cdot S_{1,2} + 2\cdot S_{1,2} + S_{1,1}, \ldots, \Sigma j\cdot S_{1,j} = a_1, \ldots, a_n\}.$$

The reverse algorithm reads

$$L_{Y(k)}[a_1, \ldots, a_n] = \{\mathbf{f}_1, 2\cdot\mathbf{f}_1, \ldots, (2^k-1)\cdot\mathbf{f}_1, \ldots, \mathbf{f}_n, 2\cdot\mathbf{f}_n, \ldots, (2^k-1)\cdot\mathbf{f}_n,$$
$$[w_{10}, w_{20}, 0, \ldots, 0], \ldots, \mathbf{w}_0, \ldots, \mathbf{w}_{[\lambda(a_n)/k]}, 2\cdot\mathbf{w}_{[\lambda(a_n)/k]}, \ldots, 2^k\cdot\mathbf{w}_{[\lambda(a_n)/k]},$$
$$2^k\cdot\mathbf{w}_{[\lambda(a_n)/k]} + \mathbf{w}_{[\lambda(a_n)/k]-1}, 2^{k+1}\cdot\mathbf{w}_{[\lambda(a_n)/k]} + 2\cdot\mathbf{w}_{[\lambda(a_n)/k]-1}, \ldots, \mathbf{a}\}.$$

In Section 12.6.2 we give an example of this algorithm.

4.11 THE BATCH-RSA ALGORITHM. $L_B[a_1, \ldots, a_n]$ (see [Fia89]).

This algorithm has only a special application. Namely $a_1, \ldots, a_n$ have to be of the form E/e_i where $E = \prod e_i$. In [CBMHS90] we have that situation. The precise description of the algorithm can be found in [Fia89]. This algorithm constructs a vector addition chain. In Section 12.7 an example have been given of this algorithm.
Fiat considers a tree. Each node in this tree represents an intermediate result which either has been built up from two other results, lower in the tree, or was one of the original inputs. We suppose first that $n = 2^h$. We denote with t the level in the tree where the algorithm runs. At the beginning we have t=0 and at the end t=h. At each level t, the algorithm consists of 2^{h-t} intermediate results. Each couple intermediate results on one level of the tree will be combined in order to get 2^{h-t-1} intermediate results at level t+1.

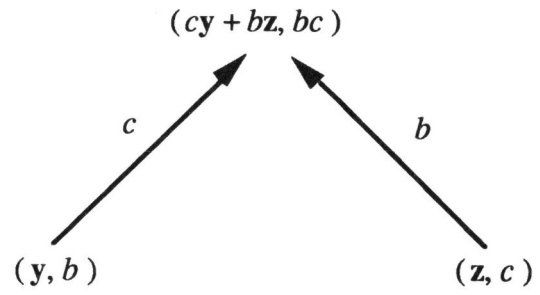

Figure 2, "crossing over"

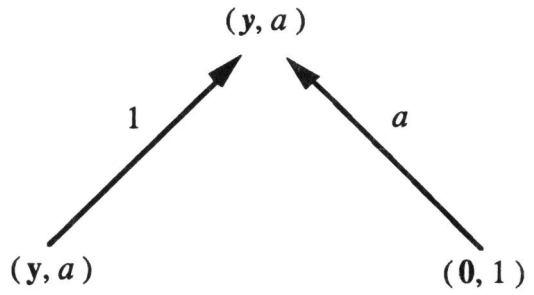

Figure 3, "crossing over with (0, 1)"

The basic idea used in this algorithm is the "*crossing over*", which has been drawn in Figure 2 above. In one memory location has been put a combination of an intermediate result and a special chosen scalar. During the crossing over is one pair calculated from two pairs as has been shown in Figure 2. The algorithm consists of two parts:

(*i*) *Initialization.*
Put the pair $(\mathbf{f}_i, e_i)$ in memory location m_i for $1 \le i \le n$.
Put 0 in t.

(*ii*) *Reduction*
Repeat (1) divide the 2^{h-t} pairs $(\mathbf{x}, a)$ into couples of pairs $\{(\mathbf{y}, b), (\mathbf{z}, c)\}$.
(2) run the procedure "crossing over" for all the 2^{h-t-1} couples of pairs.

(3) Put t+1 in t.

Until $t=h$.

Suppose that $n \neq 2^h$ for any h. Then let $h = \lceil \log n \rceil$ and fill the empty $2^h - n$ locations with the pairs (**0**, 1) (see Figure 3).

Remark. This algorithm can be made much faster by using Algorithm I for running the crossing-over procedure. We denote this algorithm by $L_{\mathrm{BI}}[a_1, \ldots, a_n]$. An example of this algorithm will be given in Section 12.7.

5. Space Complexity.

We suppose that the producer of the chain/sequence is an *addition-chain-machine* with an external memory. This addition-machine has only one memory location (*the accumulator*) and is able to double the number in this accumulator or to add a number to it from the external memory. Figure 4 shows the situation.

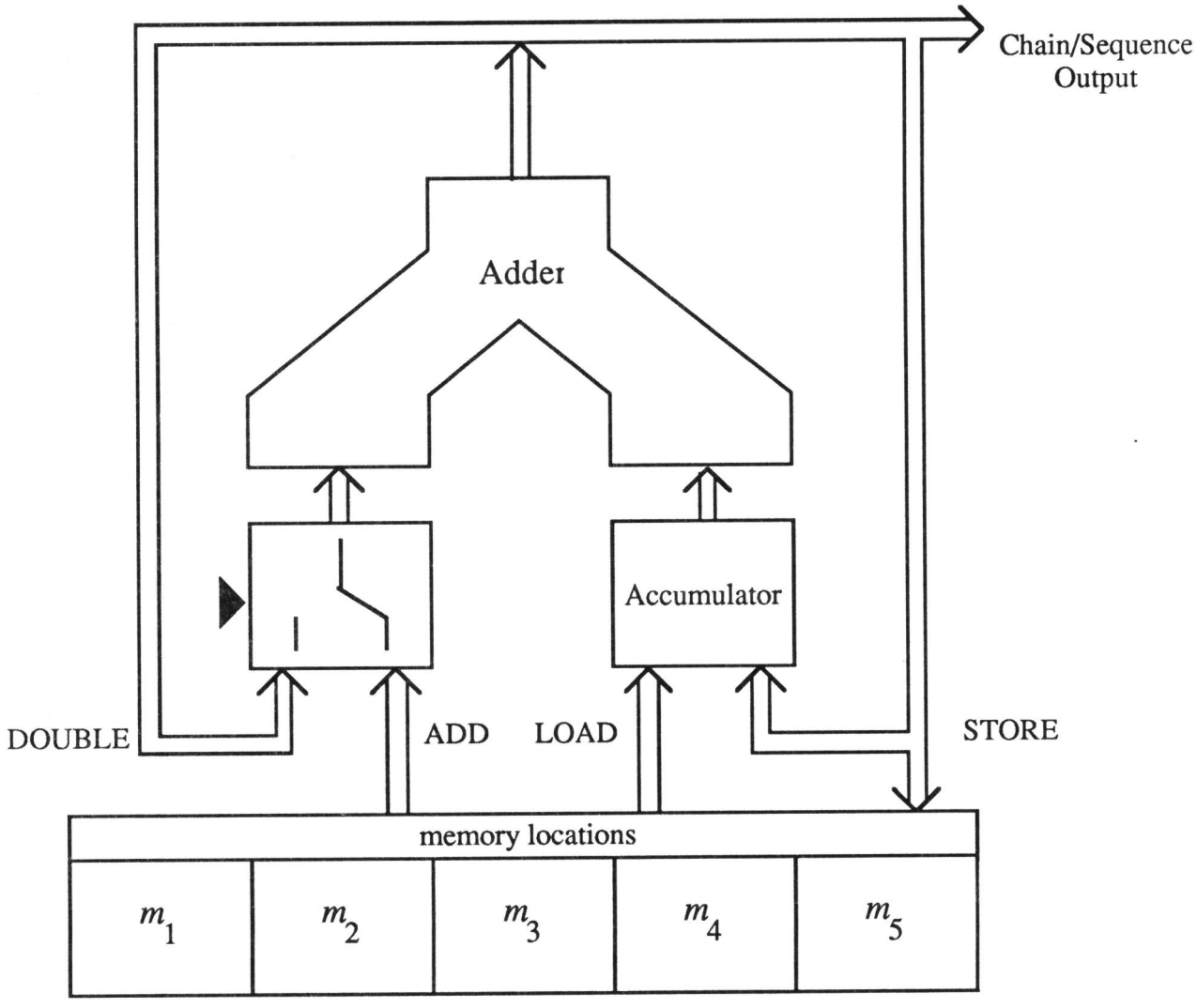

Figure 4. The addition-chain-machine.

In the case of an addition chain or sequence, the initialization consists of putting "1" in memory location m_1. In the case of a vector addition chain of width n we put the vectors $\mathbf{f}_n, \ldots, \mathbf{f}_1$ in memory locations $m_1, \ldots, m_n$, respectively. We distinguish the following operations

LD k: the number/vector in memory location m_k is put in the accumulator
STO k: the number/vector in the accumulator is put in memory location m_k
ADD k: the number/vector in memory location m_k is added to the number/vector in the accumulator
DBL: the number/vector in the accumulator is doubled.

With a list of these operations we are able to reconstruct the chain or sequence any time. But this list is also the main tool to calculate efficiently an RSA signature or a point $[n]P$ on an elliptic curve, etc.

Suppose we want to calculate ${x_1}^{a_1} \ldots {x_n}^{a_n}$ from $x_1, \ldots, x_n$ and $a_1, \ldots, a_n$ using an RSA-chip. An RSA-chip is only able to square or to multiply two numbers modulo a given modulus. We put $x_n, \ldots, x_1$ in memory locations $m_1, \ldots, m_n$ respectively. Then we execute the list of operations which was constructed by the addition-machine during the calculation of $L_\Theta[a_1, \ldots, a_n]$. The operations "DBL" and "ADD" must be replaced by "SQR" (squaring) and "MLT" (multiplication) respectively.
We give an example. If we calculate $L[6,59]$, then we get the following list of operations: LD 1, DBL, DBL, DBL, ADD 1, ADD 2, STO 2, ADD 1, STO 1, DBL, DBL, ADD 1, ADD 2 (this will be shown in Figure 8) If we want to calculate X^6Y^{59} from X and Y, this can be done via the intermediate results. [1]

LD 1	SQR	SQR	SQR	MLT 1	MLT 2	STO 2	MLT 1	STO 1	SQR	SQR	MLT 1	MLT 2
Y	Y^2	Y^4	Y^8	Y^9	XY^9	—	XY^{10}	—	X^2Y^{20}	X^4Y^{40}	X^5Y^{50}	X^6Y^{59}

Of course an important question is to keep the number of memory locations low. This section and the following section are concerned with memory use. In this section we give a general overview on memory use and we consider the case of vector addition chains in more detail. In Section 6 we give upper bounds on the memory requirements for all algorithms. It is clear that the definitions of well- and ill-conditioned algorithms are important in order to know whether an algorithm can be used in cases with large numbers.

We will first consider the case of the addition chain. We start with an example. To get the chain {1, 2, 4, 5, 9, 18, 23} we need two memory locations. The list of operations reads

---

[1] In fact the number of calculations can decrease with one if X^6Y^{59} is calculated in another way. This has been shown in Section 2.

LD 1,	DBL,	DBL,	STO 2,	ADD 1,	STO 1,	ADD 2,	DBL,	ADD 1.
1	2	4	—	5	—	9	18	23

In this example it is impossible to reduce the number of memory locations to 1 without increasing the length of the chain. This will be the subject of Lemma 1A.

Lemma 1A. *For a number a we get, using only one memory location,*

$l^{(1)}(a) = \min_{a=r_1 \cdots r_k} \left\{ \sum_{j=1}^{k} l_0(r_j) \right\}$, *where the minimum is taken over all possibilities of writing a as a product of positive integers.*

Proof. We initialize by putting 1 in memory location m_1. The first operation is LD 1. Then we get some DBL- and ADD-operations. If we get the operation STO 1 then the number r in the accumulator will be stored in memory location m_1. This number r is constructed by Algorithm 0α. After some more DBL- and ADD-operations we have a number s which will be stored in memory location m_1. Notice that s is an r-fold. □

Remark. If we use 2 memory locations then the length of the chain is in general much smaller.

In the case that we want to know the number of memory locations of a chain or a sequence in general this can be found by a straight on calculation. The situation differs in the case of vector addition chains.

Before starting the theory of memory usage by vector addition chain algorithms, we will first introduce a new notation. In fact our notation is comparable to the addition chain graph notation in [Knu81], pp. 460-462. Our notation is very useful to describe the relation between vector addition chains and addition sequences. Let $L(a_1, \ldots, a_n)=\{b_0, \ldots, b_l\}$ be an addition sequence with $1 = b_0 < b_1 < \cdots < b_l$. We write $b_0, \ldots, b_l$ in a column, such that b_l is at the top and b_0 is at the bottom. We draw a vertical arrow between b_{i+1} and b_i if this step is a star-step (i.e. $b_{i+1} = b_i + b_k$ for some $0 \leq k \leq i$). We draw a couple of vertical arrows if $b_{i+1} = 2b_i$. We call the vertical line $b_l, \cdots, b_0$ (the vertical arrows included) the *spine* of the graph. The arrows at the right side of a number b_i are called *out-arrows* and describe how b_i can be written as a sum of b_{i-1} and b_{t_1}. The arrows at the left side of a number b_i are called *in-arrows* and point from the numbers b_{s_j} (higher in the spine). The numbers b_{s_j} satisfy the equation $b_{s_j} = b_{s_j-1} + b_i$. Hence each in-arrow at b_i corresponds to an out-arrow at b_{s_j}. In scheme we have an example of a part of an addition chain graph and a corresponding counter example.

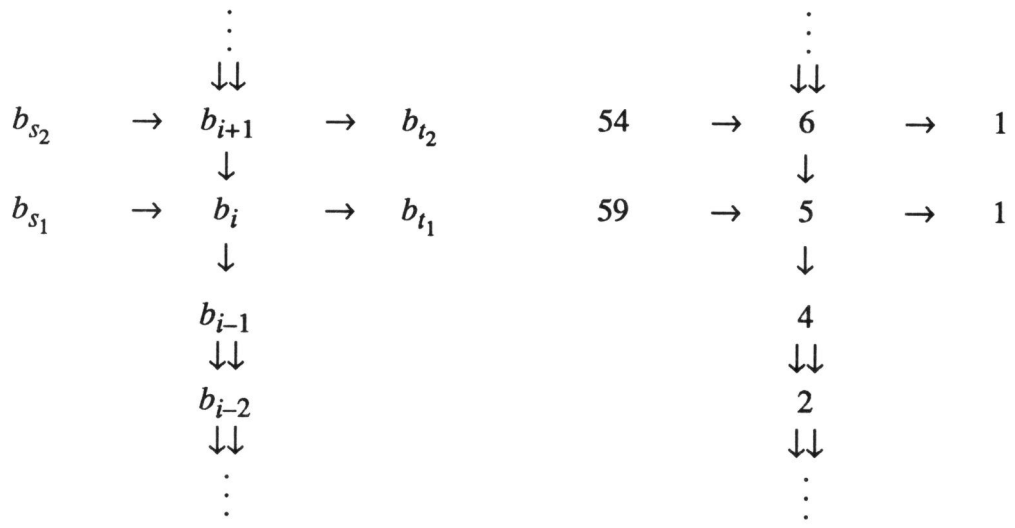

Figure 5. The example at the right is a part of Figure 8.

As Knuth we will remove the numbers b_i which do not appear in sums $b_j = b_i + b_k$ except the star-step sum. (In terms of arrows: the numbers b_i to which only one arrow is pointing.) In the following scheme we have that $c_j = c_{t_1} + 2\,c_{j-1}$, and c_j appears in the sum $c_{s_1} = \ldots + c_j$.

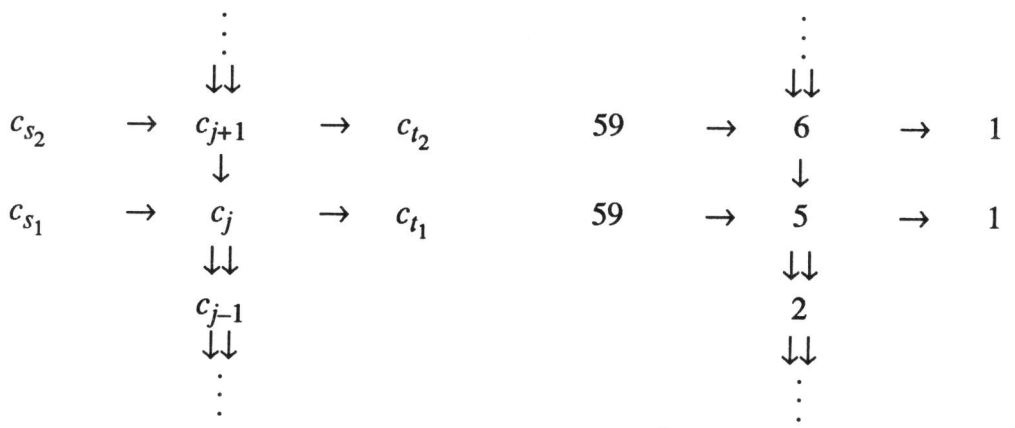

Figure 6. The example at the right is a part of Figure 8.

In order to construct the vector addition chain from the addition sequence we need the following theorem

The Reversal Theorem of N. Pippenger. *Let* $1 = c_0 < c_1 < \ldots < c_t = a$ *be a set of numbers which are written in spine-form with the in- and out-arrows. Turn all the arrows in the opposite direction, replace a by* 1 *and remove the other numbers* c_i. *Fill the spine with numbers* c_i' *such that all these numbers* c_i' $(1 \le i \le t)$ *are the sum of the numbers to which the arrows point. Then the largest element in the spine will be a.*

Proof. See [Knu81, p. 466, ex. 39].

We mentioned before that the reverse of an addition chain is an addition chain (the *reverse chain*). The reverse chain of this second reverse chain is the original chain. We can also consider the reverse of an addition sequence. In this case we consider in fact n addition chains which lead to the n different elements in the argument of the addition sequence. Here is n the width of the addition sequence. We reverse each of the n addition chains. We conceive the n addition chains as one vector addition chain of width n. We call the reverse of an addition sequence a *reverse vector chain.*

The algorithm described in [Bos90] is in fact a combination of Algorithm I and an algorithm which produces a reverse vector chain. The algorithm has the advantage that these two algorithms work simultaneously.

Figure 7 and Figure 8 are two examples of the algorithm. We calculate in both cases a vector addition chain for [6, 59]. We assume that the base $\{\mathbf{f}_1, \mathbf{f}_2\}$ is not saved during the algorithm. We put $\mathbf{f}_1$ in memory location m_2 and $\mathbf{f}_2$ in memory location m_1. The number of memory locations which is needed is 4 in the first example (Figure 7) and 2 in the second (Figure 8).

addition chain graph	operations		vector addition chain	counter
$\mathbf{f}_2 \rightarrow 59 \rightarrow 1$	LD 1		[0, 1]	
↓↓	DBL		[0, 2]	**1**
29 → 1		STO 3		
↓↓	DBL		[0, 4]	**2**
14				
↓↓	DBL		[0, 8]	**3**
7 → 1		STO 4		
↓				
$\mathbf{f}_1 \rightarrow 6 \rightarrow 2$	ADD 2	STO 2	[1, 8]	**4**
↓↓	DBL		[2, 16]	**5**
6 → 2	ADD 2		[3, 24]	**6**
↓↓	DBL		[6, 48]	**7**
59 →	ADD 1		[6, 49]	**8**
29 →	ADD 3		[6, 51]	**9**
7 → 1	ADD 4		[6, 59]	**10**

Figure 7, example of calculating L[6,59] in an inefficient way.

addition chain graph	operations		vector addition chain	counter
f_2 → 59 → 5	LD 1		[0, 1]	
→ 6				
↓↓	DBL		[0, 2]	**1**
24				
↓↓	DBL		[0, 4]	**2**
12				
↓↓	DBL		[0, 8]	**3**
59 →	ADD 1		[0, 9]	**4**
f_1 → 6 → 1	ADD 2	STO 2	[1, 9]	**5**
↓				
59 → 5 → 1	ADD 1	STO 1	[1, 10]	**6**
↓↓	DBL		[2, 20]	**7**
2				
↓↓	DBL		[4, 40]	**8**
5 →	ADD 1		[5, 50]	**9**
6 → 1	ADD 2		[6, 59]	**10**

Figure 8, example of calculating $L[6,59]$ in an efficient way.

The number of memory locations depends on the number and place of the in- and out-arrows. One or more out-arrows from one number in the spine imply the usage of one memory location. We distinguish two kinds of (out)-arrows. *Long-arrows* are arrows which point from a number in the spine to the smallest number at the right side of the spine. We denote these arrows with "→". In Figure 8 all except the arrow from 59 to 6 are long-out-arrows. Besides those arrows we have *short-arrows*. These are the out-arrows from a number to all but the smallest number. We denote these arrows with "→". An in-arrow implies that a memory location becomes free if and only if this arrow is a long-arrow.

At the moment that the algorithm starts, all memory locations are occupied. The amount of memory locations will increase if at a vertex of the spine a long-arrows starts and no long-arrow enters. The amount of memory locations will decrease if at a vertex of the spine a long-arrow enters and there are no out-arrows. In Figure 7 has been shown how the total number of arrows increases enormously, since most of the storages are not combined. It is possible to hold the number of memory locations constant. In Figure 8 such an algorithm has been shown.

Notice that in general the short and long arrows are not invariant if we replace an addition chain (addition sequence) by its reverse.

Algorithm I uses only n locations. This is the subject of the following theorem.

Theorem 2. $m_I(a_1, \ldots, a_n) = n$.

Before we will prove this theorem, we will first prove a simple lemma which plays a central role in the proof of the theorem.

Lemma 1B. *For a number a we get, using only one memory location,*

$l^{(1)}[a] = \min_{a=r_1 \ldots r_k} \left\{ \sum_{j=1}^{k} l_0(r_j) \right\}$, *where the minimum has to be taken over all possibilities writing a as a product of some positive integers.*

Proof. Consider the following figure.

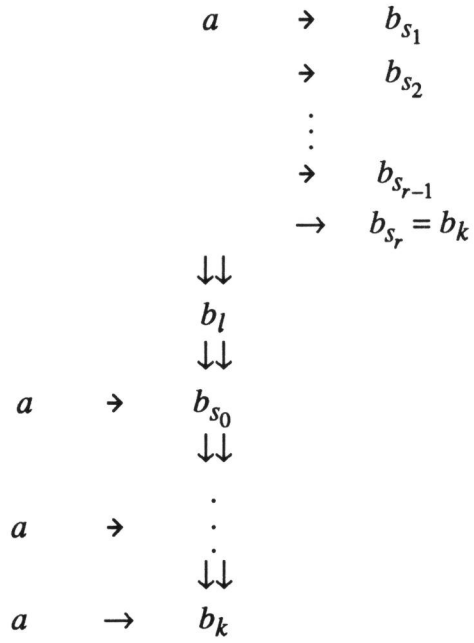

Figure 9

The crucial part of the proof is that since the arrow from a to b_k is a long-arrow, there will be no more out-arrows between a and b_k. Therefore the spine between a and b_k will contain only doublings. Hence b_{s_j} is of the form $b_{s_j}=b_k \cdot 2^{t_j}$. Finally we find that a is of the form $a = b_k \cdot \sum_{j=1}^{r} 2^{t_j}$, which implies that b_k divides a. □

Proof of Theorem 2. We will proof this theorem on induction to n and a_n. The proof of the theorem in the case that $n = 1$ is given in Lemma 1B. By the induction assumption we suppose that we proved the theorem for $n-1$ and for n-tuples $(b_1, \ldots, b_n)$ with $b_n < a_n$. If $a_1 = 0$ then we have in fact a $n-1$-tuple for which we proved the theorem. Now we consider $\left[\frac{a_n}{a_{n-1}}\right] = r = r_0 + 2r_1 + \ldots + 2^k r_k$. Let $\{r_{i_1}, \ldots, r_{i_s}\}$ be the set of bits in r equal to 1. Then Algorithm I predicts the following addition chain graph.

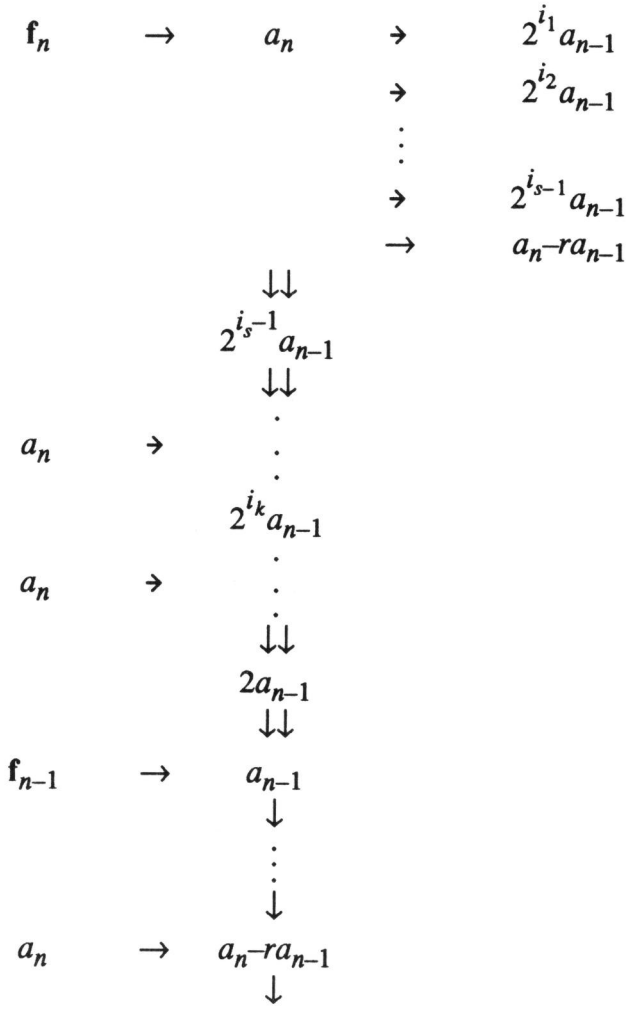

Figure 10

By the induction assumption we proved the theorem in the case of the n-tuple $(a_n - ra_{n-1}, a_1, \ldots, a_{n-1})$. Starting the algorithm for the n-tuple $(a_1, \ldots, a_n)$ we get one memory location, since an in-arrow points to a_n. The long arrow points to $a_n - ra_{n-1}$. Therefore we cannot store numbers in the memory locations between a_n and a_{n-1}. This implies that in the part of the spine between a_n and a_{n-1} there will be no out-arrows. The situation is comparable to the situation in Lemma 1B. □

Some generalizations of Algorithm I. The following generalizations do not influence the number of memory locations.

① If a long-in-arrow points to a_n and a_n is even it is possible to half a_n, especially if $a_n \geq 2a_{n-1}$.

② Instead of a_{n-1} we can choose a_{n-i} in Figure 10, if $2a_{n-i} > a_{n-1}$.

③ Instead of spine element $a_n - ra_{n-1}$ we can choose numbers b of the form

$b = a_n - \left(ra_{n-i} + \sum_{a_{j_k} \neq a_{n-i}} a_{j_k} \right)$, where $a_{n-i} > a_{n-1}/2$ as in generalization ②. In this case it is necessary to choose b such that $b < a_{j_k}$, for all k.

④ As was mentioned before insert the following statement: if $a_n = a_i + a_j$ for some i and $j < n$ then: $I(a_1, \ldots, a_n) = \{a_n\}$ and $T(a_1, \ldots, a_n) = (0, a_1, \ldots, a_{n-1})$.

⑤ Instead of the set $L_0(r)$ we can take $L^{(1)}$ as was described in Lemma 1B.

6. THE AMOUNT OF MEMORY USED BY THE ALGORITHMS.

We saw in Section 5 that $m_I[a_1, \ldots, a_n] = n$. In this section we will consider the other algorithms and the number of memory locations if those algorithms are used.

Lemma 3. *We have the following amounts of memory*:

(*i*)	①	$m_{0\alpha}(a) = 1$,	②	$m_{0\beta}(a) = \max(\nu(a)-1, 1)$,
	③	$m_{0\alpha}[a] = \max(\nu(a)-1, 1)$,	④	$m_{0\beta}[a] = 1$.
(*ii*)		$m_{\mathrm{MOA}}(a) = m_{\mathrm{MOB}}(a) = 2$		
(*iii*)	①	$m_{\mathrm{K}(k)\alpha}(a) = 2^{k-1}$	②	$m_{\mathrm{K}(k)\beta}(a) = 2^{k-1}+1$
	③	$m_{\mathrm{K}(k)\alpha}[a] = 2^{k-1}+1$	④	$m_{\mathrm{K}(k)\beta}[a] = 2^{k-1}$
(*iv*)		$m_{\mathrm{W}(n)}(a) = n$		
(*v*)		$m_{\mathrm{S}}(a)=2$		
(*vi*)	①	$m_{\mathrm{II}}(a, b) = 9$	②	$L_{\mathrm{II}}[a, b]$ is ill conditioned
(*vii*)	①	$m_{\mathrm{S}}(a, b) = 2$	②	$m_{\mathrm{S}}[a, b] = 2$
(*viii*)	①	$m_{\mathrm{III}}(a, b, c) = 6$	②	$L_{\mathrm{III}}[a, b]$ is ill conditioned
(*ix*)	①	$m_{\mathrm{I},0\alpha}(a_1, \ldots, a_n) = n$	②	$m_{\mathrm{I},0\beta}[a_1, \ldots, a_n] = n$
	③	$m_{\mathrm{I},\mathrm{K}(2)\alpha}[a_1, \ldots, a_n] = n+1$	④	$m_{\mathrm{I},\mathrm{K}(2)\beta}[a_1, \ldots, a_n] = n+1$
(*x*)		$m_{\mathrm{F}}(a_1, \ldots, a_n) = 2^n-1$		
(*xi*)		$m_{\mathrm{St}(k)}[a_1, \ldots, a_n] = [\lambda(a_n)/k] + (2^{nk}-1)$ (hence ill conditioned)		
(*xii*)	①	$m_{\mathrm{Y}(k)}(a_1, \ldots, a_n) = n(2^k-1)$		
	②	$m_{\mathrm{Y}(k)}[a_1, \ldots, a_n] = [\lambda(a_n)/k] + n(2^k-1)$ (hence ill conditioned)		
(*xiii*)	①	$m_{\mathrm{B}}[a_1, \ldots, a_n] = n$	②	$m_{\mathrm{B,I}}[a_1, \ldots, a_n] = n$

Proof.

(*i*) ① can be found in Lemma 1A and ④ in Lemma 1B. $m_{0\beta}(a) = \nu(a)-1$ since the fact that if $a = \sum 2^{r_i}$ then each term 2^{r_i} except of the largest term must be stored. For ③ holds exact the same argument as for ② since both chains are

identical.

(*ii*) We need only memory locations for M^+ and M^-.

(*iii*) ① We have to store $\{1, 3, 5, \ldots, 2^k-1\}$. ② In Algorithm K($k$)β we need 2^{k-1} memory locations for the numbers $g_1, \ldots, g_{2^k-1}$ (see Section 4.3). Besides these 2^{k-1} memory locations we need 1 memory location for doubling 2^r. Since K(k)α and K(k)β are their reverses ③ follows from ② and ④ follows from ①.

(*iv*) We need n memory locations for the n windows w_i. The sequence which constructs these n windows do not need more memory (see Algorithm I).

(*v*) The algorithm is based on the combination of the Addition-Subtraction Algorithm and Algorithm W(2). Both algorithms do not use more than 2 memory locations.

(*vi*) ① We need 7 memory locations for the numbers 1, ... , 7. These numbers appear in the cases ① and ⑥. Besides these 7 memory locations we need 2 more for calculating a and b from smaller numbers a' and b' in the other cases. ② Almost each time that we pass step ① or step ⑥, a long arrow is required to one of the numbers 1, ... , 7 which implies an extra memory location.

(*vii*) The construction using the mappings I_S and T_S is identical to the case of Algorithm I. The proof of ① and ② is comparable to the proof of ① and ② in (*viii*). We have to warn that the choice of the algorithm which produces the chain $\{1, 2, \ldots, r\}$ must satisfy the same conditions as described in Section 4.8.

(*viii*) We have the same situation as in (*v*).

(*ix*) The first result can easily be verified using Lemma 1A. The second result was proved in Theorem 2. We need one memory location more if we use the 2-Window Algorithm.

(*x*) We need 2^n-1 memory locations for the numbers c_k.

(*xi*) We need the memory locations to store the vectors $\mathbf{v}_j$ for $0 \le v_{ij} \le 2^k-1$, $1 \le i \le n$, but we can omit the $\mathbf{0}$-vector.

(*xii*) ① We need $n(2^k-1)$ memory locations to build up and store the numbers S_{il}. ② We need the memory locations to store the vectors $l \cdot \mathbf{f}_i$ for $1 \le l \le 2^k-1$, $1 \le i \le n$ and $\mathbf{w}_j$ for $0 \le j \le [\lambda(a_n)/k]$.

(*xiii*) We need the n memory locations for the n vectors $\mathbf{f}_i$. At each level the number of memory locations is reduced. □

7. UPPER BOUNDS $\hat{\rho}_\Theta$.

In this Section we consider upper bounds for numbers $\hat{\rho}_\Theta(\alpha, n)$ which appear in

$$\hat{\rho}_\Theta(\alpha, n) = \max_{\substack{a_1 \dots a_n \\ \lambda(a_n)=\alpha}} \left\{ \frac{l(a_1, \dots, a_n)}{\log a_n} \right\}.$$

In the rest of this report we will abbreviate this notation to $\hat{\rho}_\Theta$. Besides this upper bound we consider other upper bounds, namely $\hat{\rho}^{\square}{}_\Theta(\alpha, n)$ and $\hat{\rho}^{+}{}_\Theta(\alpha, n)$, which were defined in Section 2 and which have to do with the number of doubles and the other additions respectively.

We will calculate these upper bounds for almost all algorithms. We conjecture an upper bound for Algorithm I. As mentioned in the introduction of Section 4, the mappings $T(a_1, \dots, a_n)$ and $I(a_1, \dots, a_n)$ play a main role in this report. We can construct $L(a_1, \dots, a_n)$ by the recursive formula

$$L_\Theta(a_1, \dots, a_n) = L_\Theta(T_\Theta(a_1, \dots, a_n)) \cup I_\Theta(a_1, \dots, a_n). \tag{11}$$

Let a'_n be $\max(T_\Theta(a_1, \dots, a_n))$. Suppose that $l_\Theta(T_\Theta(a_1, \dots, a_n)) \leq \hat{\rho}_\Theta \log a'_n$. Then $I_\Theta(a_1, \dots, a_n)$ must satisfy

$$\#I_\Theta(a_1, \dots, a_n) \leq \hat{\rho}_\Theta \log (a_n/a'_n) \tag{13}$$

We introduce numbers μ and μ_k by

$$\begin{aligned} \mu &= 2^{1/\hat{\rho}}, \\ \mu_k &= \mu^{-k}. \end{aligned} \tag{14}$$

Inequality (13) can be rewritten as

$$\begin{aligned} \#I(a_1, \dots, a_n) &= k, \\ a'_n &\leq a_n \cdot \mu_k. \end{aligned} \tag{15}$$

7.1 Upper bounds for the Small Window Algorithm and the Algorithm of Morain and Olivos.

Worst cases for L_{MOA} and L_{MOB} are the numbers with binary representation 11011...11011 and 10101...10101 respectively. It is not difficult to find that $l_{MOA}(a) \leq \frac{5}{3}\lambda(a)$, $l^{+}_{MOA}(a) \leq \frac{2}{3}\lambda(a)$, $l_{MOB}(a) \leq \frac{3}{2}\lambda(a)$ and $l^{+}_{MOB}(a) \leq \frac{1}{2}\lambda(a)$.

Notice in the case of the Small Window Algorithm that the length of both $L_{K(k)\alpha}$ and $L_{K(k)\beta}$ are the same. Therefore we consider here only $L_{K(k)\alpha}$. We will consider below the maximal length of the addition chain. We divide the elements of the chain into three subsets. (cf. BoCo89):

For $k > 1$ we get:

		# elements
(*i*)	the elements 2, 3, 5, 7, ... , 2^k-1,	2^{k-1}
(*ii*)	the numbers n for which $n/2$ is in the chain or the number which is the sum of two numbers of (*i*)	$\lambda(a)-k+1$
(*iii*)	numbers which are the sum of a number of (*i*) and a number of (*ii*)	$[\lambda(a) / k]$
	TOTAL:	$l_{K(k)}(a) \leq [(1 + 1/k)\lambda(a)] + 2^{k-1}-k + 1$

We find $l^{\square}_{K(k)}(a) \leq \lambda(a)-k+1$ and $l^{+}_{K(k)}(a) \leq [1/k\lambda(a)] + 2^{k-1}$. The length is minimal if $k^2 \cdot 2^k \approx 2 \cdot \log a / \ln 2$. Hence we find for each $\varepsilon > 0$ a number x such that for each $\alpha > x$ and we have $\hat{\rho}_{K(k)}(\alpha, 1) = 1 + 1/k + \varepsilon$. As a special case of the Small Window Algorithm we have for $k = 1$ the Binary Algorithm. We leave to the reader that $l_0(a) \leq 2\,\lambda(a)$, $l^{\square}{}_0(a) \leq \lambda(a)$ and $l^{+}{}_0(a) \leq \lambda(a)$ (cf. [Knu81]). Hence $\hat{\rho}_0(\alpha, 1) = 2$, $\rho^{\square}{}_0(\alpha, 1) = 1$ and $\hat{\rho}^{+}{}_0(\alpha, 1) \leq 1$. We will use later in this report the upper bounds $l_{K(2)}(a) \leq \frac{3}{2} \cdot \lambda(a) + 1$ and $l^{+}_{K(2)}(a) \leq \frac{1}{2} \cdot \lambda(a) + 1$.

7.2 Upper bounds for Algorithms I, II, III and W(*n*).

We will discuss first the Algorithms II and III, and then we will conjecture an upper bound for Algorithm I.

Theorem 4. *For Algorithm* II, *we have the inequality*

$$l_{II}(a, b, c) \leq 1.94\cdots \log c + 1.$$

Hence $\hat{\rho}_{II}(\alpha, 2) = 1.94\cdots + \varepsilon$.

Remark. The algorithm is a bit arbitrary. It is possible to improve Algorithm II, and to replace the value $\hat{\rho}_{II}=1.94\cdots$ by a lower value. But for lower values of $\hat{\rho}_{II}$, Algorithm II is more complicated, especially if $\hat{\rho}_{II} \leq 1.9$.

Proof. The proof is by induction. We assume that $L_{II}(a, b)$ contains the numbers $1, 2, \ldots, 7$. Now b (in the case that $a/b< 3$ or $a/b> 0.84$) can be written as the sum of $8[b/8]$ and (eventually) a smaller number in the addition sequence.

The theorem can be verified for $b < 8$. For $b \geq 8$ we use the following induction step. We search for a number pair (a', b') which satisfies

(i) $L_{II}(a, b)=L_{II}(a', b')\cup\{c_1, \ldots, c_k\}$,

(ii) $k \leq \hat{\rho} \log (b/b')$.

If we find for each pair (a, b) a pair (a', b') which satisfies (i) and (ii) then we have proved the theorem since

$$l_{II}(a,b) = l_{II}(a', b') + k \leq \hat{\rho} \log b' + 4 + \hat{\rho} \log (b/b').$$

In Algorithm II the pair $(a', b') = T_{II}(a,b)$ and $\{c_1, \ldots, c_k\} = I_{II}(a, b)$. We need only to prove property (ii) in the 6 cases of Algorithm II. To do this we use the numbers μ and μ_k introduced in (14). Property (ii) can be rewritten as

(ii') $\max(a', b') \leq b\mu_k$.

Suppose that $\#I_{II}(a, b) = k$ and

$$\begin{cases} a' = \sigma_1 a - \tau_1 b & \text{and} \quad b' = \tau_2 b - \sigma_2 a \quad \text{if } \sigma_1 > 0, \\ a' = \tau_2 b - \sigma_2 a & \text{and} \quad b' = \sigma_1 a - \tau_1 b \quad \text{if } \sigma_2 > 0. \end{cases}$$

then the pair (a', b') satisfies (ii') if

$$\max((\tau_2-\mu_k)/\sigma_2, \tau_1/\sigma_1) \leq a/b \leq \min((\tau_2/\sigma_2, (\tau_1+\mu_k)/\sigma_1). \tag{16}$$

$\hat{\rho} = 1.94\cdots$ implies that $\mu=10/7$. The cases ①, ..., ⑥ satisfy for this μ inequality (16), which follows from the following table.

Table I

case	k	σ_1	τ_1	σ_2	τ_2	$\leq a/b \leq$	
①	3	1	0	0	1/4	0	0.343
②	1	1	0	1	1	0.3	0.7
③	3	3	2	1	1	0.666⋯	0.781
④	4	4	3	1	1	0.7599	0.810025
⑤	4	5	4	1	1	0.8	0.84802
⑥	5	1/8	0	1	1	0.83193	1

□

We have a comparable bound for n=3.

Theorem 5. *We have the inequality*

$$l_{\mathrm{III}}(a, b, c) \leq 3 \log c + 1.$$

Hence $\hat{\rho}_{\mathrm{III}}(\alpha, 3) = 3 + \varepsilon$.

Proof. The proof is comparable to the proof of Theorem 4. We prove the theorem by induction. The theorem holds for all 3-tuples (a, b, c) with $c \leq 4$. For a 3-tuple (a, b, c) with $c > 4$ we search for a 3-tuple (a', b', c') which satisfies

(*i*) $L_{\mathrm{III}}(a, b, c) = L_{\mathrm{III}}(a', b', c') \cup \{d_1, \ldots, d_k\}$,
(*ii*) $k \leq 3 \log (c/c')$.

As in the proof of Theorem 4 property (*ii*) can be rewritten in the form

(*ii'*) $\max(a', b', c') \leq c\mu_k$.

$\hat{\rho} = 3$ implies that $\mu = \sqrt[3]{2}$ (hence $\mu_1 = 2^{-1/3} = 0.7937$; $\mu_2 = 2^{-2/3} = 0.6300$; $\mu_4 = 2^{-4/3} = 0.3969$; $\mu_5 = 2^{-5/3} = 0.3150$). The cases ①, … , ⑥ cover the simplex $\{a, b, c \mid 0 \leq a \leq b \leq c\}$. To see this we divide by c and consider the 3-gon $\{\frac{a}{c}, \frac{b}{c} \mid 0 \leq \frac{a}{c} \leq \frac{b}{c} \leq 1\}$. Each case has some restrictions on $\frac{a}{c}$ and $\frac{b}{c}$. These restrictions are tabled below. The reader verifies that the 6 areas cover the 3-gon mentioned above.

Table II

case	k	conditions on $\frac{a}{c}$ and $\frac{b}{c}$.	
①	1	$1-\mu_1 \le \frac{b}{c} \le \mu_1$	-
②	2	$1 \le 2\mu_2$; $\frac{b}{c} \le \mu_2$	$\frac{a}{c} \le \mu_2$
③	2	$1-\mu_2 \le \frac{b}{c}$	$\frac{b}{c} - \frac{a}{c} \le 2\mu_2$
④	3	$1-\mu_2 \le \frac{b}{c} \le 2\mu_2$	$\frac{a}{c} \le \mu_2$
⑤	4	$1-\mu_4 \le \frac{b}{c}$	$\frac{a}{c} \le 2\mu_4$
⑥	5	$1-\mu_5 \le \frac{b}{c}$	$\frac{a}{c} \le 4\mu_5$

□

We will now give some arguments for a conjecture about an upper bound for $l_I(a_1, \ldots, a_n)$. For large n in almost all steps r will be equal to 1. Therefore we make a fair approximation if we assume that in all steps $T_I(a_1, \ldots, a_n) = (a_0, a_1, \ldots, a_{n-1})$, where $a_0 = a_n - a_{n-1}$. We are interested in an upper bound for ρ and therefore in a lower bound for the average quotient $\mu = a_n/a_{n-1}$. As described in Theorem 4 we have

$$\hat{\rho} \ge 1 / \log \mu$$

Assume that $a_k/a_{k-1} = \mu$ is constant for $k = 1, \ldots, n$. In that case we get $a_k = \mu^k a_0$ and especially

$$a_{n-1} = \mu^{n-1} a_0$$

$$a_n = \mu^n a_0$$

Subtracting these two equations leads to

$$\mu^n = \mu^{n-1} + 1.$$

μ can be approximated by

$$\mu = 1 + \ln n / n.$$

From μ we can approximate $\hat{\rho}$ which leads to the following conjecture.

Conjecture 6. *We conjecture that*

$$l_I(a_1, \ldots, a_n) < \hat{\rho} \log a_n, \text{ where } \hat{\rho} \approx 1 + n / \log n .$$

We verified the conjecture for different cases, $2 \le n \le 100$ and $10^3 \le a_n \le 10^{100}$, and

in all cases with $a_n \geq 2^n$ the conjecture seems to hold.

7.2.1 Application on the Large Window Algorithm.

Using the conjecture concerning the upper bound of Algorithm I we are able to find a conjecture for the upper bound for the Large Window Algorithm. As for the Small Window Algorithm, we divide the elements in three subsets. Notice that $w_n \approx \lambda(a)/n$.

		# elements
(i)	the elements $L_I(w_1, \ldots, w_n)$	$(1+n/\log n)\log(\lambda(a)/n)$
(ii)	the numbers n for which $n/2$ is in the chain	$(1-1/n)\lambda(a)$
(iii)	numbers which are the sum of a number of (i) and a number of (ii)	$n-1$
	TOTAL:	$l_{W(n)}(a) \leq (1 + 1/\log n)\lambda(a) + n-1$

7.3 Upper bounds for $L_I(a,b)$, $L_S(a,b)$, $L_{W(2)}(a)$ and $L_S(a)$.

7.3.1 An Upper bounds for $L_I(a,b)$ and $L_{W(2)}(a)$.

Theorem 7. ① *Using Algorithm* I *in combination with the Binary Algorithm, we get for sequences of width* 2 *the upper bound*

$$l_{I,0}(a, b) \leq 2 \log b.$$

② *This upper bound is sharp.*

Proof. ① The proof is comparable to the proof of the upper bound of Algorithm II. The claim can be checked for pairs (a, b) with $b < 8$. For $b \geq 8$ we prove that for each pair (a, b) that there exists a pair (a', b') such that

(i) $L_I(a, b)=L_I(a', b') \cup \{d_1, \ldots, d_k\}$,
(ii) $k \leq 2 \log (b/b')$.

We distinguish the following cases:

(1) $0 \leq a/b \leq 0.5$
(2) $0.5 < a/b \leq 0.7$
(3) $0.7 < a/b < 1$

(1) If $a/b \leq 0.5$ then we have $T_I(a, b) = (b-ra, a)$ and $I_I(a, b) = \{2a, \ldots, ra, b\}$ where $r > 1$. Here the set $\{1, 2, \ldots, r\}$ is constructed by the Binary Algorithm. Property (ii) can be written in the form

(ii') $\#I_I(a, b) \leq 2 \log (b/b')$.

Since $\#I_I(a, b) = l_0(r) + 1$ and $b' = a$ we get the inequality

$$l_0(r) + 1 \leq 2 \log (b/a).$$

In Lemma 8 the proof has been given that $l_0(r) + 1 < 2 \cdot \log r$ for $r > 1$, which implies the inequality.

(2) If $0.5 < a/b \leq 0.7$ then the proof is exact the same as in case (2) of Theorem 4.

(3) If $a/b > 0.7$ then we have $T_I^2(a,b) = ((r+1)a - rb, b-a)$ and $I_I(a, b) \cup I_I(T_I(a, b)) = \{2(b-a), \ldots, r(b-a), a, b\}$, in this set $\{1, 2, \ldots, r\}$ is the addition chain constructed by the Binary Algorithm. In this case we have to prove that $l_0(r) + 2 \leq 2 \log (b/(b-a))$. Notice that $b/(b-a) \geq r+1$. In Lemma 8 the proof has been given that $l_0(r) + 2 \leq 2 \cdot \log (r+1)$, which implies the inequality. We get an equality if $r = 2^h - 1$.

② For proving the fact that this upper bound cannot be improved, we consider the quadratic number $x = [0, \overline{1, r}] = \frac{1}{2}\left(\sqrt{r^2+4r} - r\right)$. Lemma 9 predicts that for $r = 2^h - 1$ we have $\hat{\rho} = \dfrac{2h}{\log \frac{1}{2}\left(\sqrt{r^2+4r} + r + 2\right)}$. If $h \to \infty$ then $\hat{\rho} \to 2$. □

Lemma 8. *We have the following inequalities*

(i) $l_0(r) + 1 < 2 \cdot \log r$ *for* $r \geq 2$,

(ii) $l_0(r) + 2 \leq 2 \cdot \log (r+1)$ *for* $r \geq 1$.

Proof. These inequalities can be proved by induction on r. For $r = 2$ and 3 the first inequality satisfies, for $r = 1$ the second inequality satisfies. Since $l_0(2r) = l_0(r) + 1$ and $l_0(2r + 1) = l_0(r) + 2$ the inequalities follow for larger values of r. □

Corollary 9. We find comparable to Section 7.2.1 for Algorithm W(2) the upper bound $l_{W(2),I}(a) \leq \frac{3}{2} \cdot \log a + 1$. (This is the same result as found in Section 7.1.)

If we do not restrict ourselves to use the Binary Algorithm to produce an addition chain for r, but we allow ourselves to use a more efficient algorithm to produce this chain, then we find a lower upper bound. Before we give the theorem we need a definition and a technical lemma.

Definition. We define $S_r = \begin{pmatrix} -r & 1 \\ 1 & 0 \end{pmatrix}$. Using this definition it is easy to describe $T_I(a, b)$. Namely if $\left[\frac{b}{a}\right] = r$ then $T_I(a,b) = \left[S_r \begin{pmatrix} a \\ b \end{pmatrix}\right]^T$.

Lemma 10. *(i) Let* $H_0 = [0, \alpha_1, \ldots, \alpha_n]$ *and* $H_1 = [0, \alpha_1, \ldots, \alpha_n+1]$. *The theory of continued fractions predicts that if n is even then* $H_1 < H_0$ *else* $H_0 < H_1$. *Let H be the interval* $[H_1, H_0]$ *or* $[H_0, H_1]$ *depending on n. Let* $a/b \in H$. *Then*

$$l_I(a,b) \le \frac{k}{\log|\sigma_1|} \cdot \log\left(\frac{b}{b'}\right) + l_I(a',b'), \tag{17}$$

where $\mathbb{T}\begin{pmatrix}a\\b\end{pmatrix} = \begin{pmatrix}a'\\b'\end{pmatrix}, \mathbb{T} = S_{\alpha_n} \cdot S_{\alpha_{n-1}} \ldots S_{\alpha_1} = \begin{pmatrix}\sigma_1 & \tau_1\\ \sigma_2 & \tau_2\end{pmatrix}$ *and* $k = \sum_{i=1}^{n} (l(\alpha_i)+1)$.

(ii) If $\alpha_{n+1} = r > 0$ *then we have*

$$l_I(a,b) \le \frac{k}{\log(|\sigma_1| + \frac{1}{r+1} \cdot |\sigma_2|)} \cdot \log\left(\frac{b}{b'}\right) + l_I(a',b'). \tag{18}$$

Proof. *(i)* Let $\frac{a}{b} = [0, \alpha_1, \ldots, \alpha_n, \alpha_{n+1}, \ldots, \alpha_{n+k}]$ with $k \ge 0$. Then $\frac{a'}{b'} = [\alpha_{n+1}, \ldots, \alpha_{n+k}]$. Hence $a'/b' \in [0, 1)$. We have $l_I(a, b) = l_I(a', b') + k$ where $k = \sum (l(\alpha_i)+1)$. Now we want to find an expression for b/b'. It is sufficient to prove that $b/b' \ge |\sigma_1|$. This can be seen as follows. Since $\det S_r = -1$ we have $\det \mathbb{T} = (-1)^n$. Hence $\mathbb{T}^{-1} = (-1)^n \cdot \begin{pmatrix}\tau_2 & -\tau_1\\ -\sigma_2 & \sigma_1\end{pmatrix}$. Notice that $(-1)^n \cdot \sigma_1 \ge 0$ and $-(-1)^n \cdot \sigma_2 \ge 0$. Finally we find

$$b = (-1)^n \cdot (-\sigma_2 \cdot a' + \sigma_1 \cdot b') = (|\sigma_2| \cdot a' + |\sigma_1| \cdot b') \ge |\sigma_1| \cdot b'. \tag{19}$$

(ii) The condition $\alpha_{n+1} = r > 0$ implies that $\frac{1}{r+1} < \frac{a'}{b'} \le \frac{1}{r}$. Now apply (19). □

Now we have the tools for proving the following theorem. The upper bound in the theorem seems not to be much better then the bound given in Theorem 7. However this bound is only an indication. Using a computer the upper bound can be improved.

Algorithm K(2) which appears in the theorem below is only an example. We leave the proof to the reader to replace K(2) by another algorithm.

Theorem 11. $l_{I,K(2)}(a, b) \le 1.96 \cdot \log b$.

Proof. The proof is comparable to the proof of Theorem 7. We proof the theorem on induction. We can verify the theorem for small values of a, b ($a \le b \le 16$). For larger values of a and b we apply Lemma 10(*i*). We will prove for all values a/b in $[0,1)$ that there exists a pair (a', b') such that the quotient $\frac{k}{\log|\sigma_1|}$ satisfies the inequality

$$\frac{k}{\log|\sigma_1|} \leq \hat{\rho}. \tag{20}$$

Using the theory written in the introduction of Section 7 the theorem can be derived. We have only to verify the cases for which inequality (20) holds. These cases can be derived from the table below. Using $l_{K(2)}(r) \leq \frac{3}{2} \cdot \lambda(r) + 1$ (see Section 7.1) we find that this inequality is true for $r > 2^{\frac{9}{4}(g_1+1)}$. (Here is $g_1 = k - l_{K(2)}(r)$.)For the numbers $r < 2^{\frac{9}{4}(g_1+1)}$ we have to verify the inequality by a calculator.

Table III

cont. fraction	k	$\lvert\sigma_1\rvert$	r such that $\frac{k}{\log\lvert\sigma_1\rvert} > \hat{\rho}$.
$[0, r, \ldots]$	$l(r)+1$	r	1, 2
$[0, 1, r, \ldots]$	$l(r)+2$	$r+1$	1, 3, 7
$[0, 2, r, \ldots]$	$l(r)+3$	$2r+1$	
$[0, 1, 1, r, \ldots]$	$l(r)+3$	$2r+1$	
$[0, 1, 3, r, \ldots]$	$l(r)+5$	$4r+1$	1
$[0, 1, 7, r, \ldots]$	$l(r)+7$	$8r+1$	1
$[0, 1, 3, 1, r, \ldots]$	$l(r)+6$	$5r+4$	
$[0, 1, 7, 1, r, \ldots]$	$l(r)+6$	$9r+8$	

□

Improvements on Theorem 11. The number $\hat{\rho}_{I,K(2)} = 1.96$ of the theorem can be improved. But there are restrictions. Table IV shows that $\hat{\rho}_{I,K(2)} > 1.904\ldots$. But in practice the proof for numbers $\hat{\rho}_{I,K(2)}$ between 1.904 and 1.96 is rather hard. Before we give Table IV we will give a technical lemma which is useful to verify the results of Table IV.

Lemma 12. *Let* $x = [0, \overline{\alpha_1, \alpha_2, \ldots, \alpha_n}]$. *Let* $\left\{\frac{P_s}{Q_s}\right\}_{s=1}^{\infty}$ *be the set of convergents of* x *with* $\frac{P_0}{Q_0} = 0$. *Let* $S_r = \begin{pmatrix} -r & 1 \\ 1 & 0 \end{pmatrix}$ *and* $\mathbb{T} = S_{\alpha_n} \cdot S_{\alpha_{n-1}} \ldots S_{\alpha_1} = \begin{pmatrix} \sigma_1 & \tau_1 \\ \sigma_2 & \tau_2 \end{pmatrix}$. *Then we have* $\lim_{s\to\infty} l_I(P_s, Q_s) = \hat{\rho} \cdot \log Q_s$ *where* $\hat{\rho} = \frac{\sum_{i=1}^{n}(l(\alpha_i)+1)}{-\log\theta}$ *an* $\theta = \frac{1}{2}\left(\sigma_1 + \tau_2 \pm \sqrt{(\sigma_1+\tau_2)^2 - 4(-1)^n}\right)$. *Here the* $\pm$*-sign must be chosen such that* $0 < \theta < 1$.

Proof. Notice that $\det \mathbb{T} = (-1)^n$. x is a quadratic number which satisfies

$\mathbb{T}\begin{pmatrix} x \\ 1 \end{pmatrix} = \xi\begin{pmatrix} x \\ 1 \end{pmatrix}$ such that $\xi = \frac{1}{2}\left(\sigma_1 + \tau_2 \pm \sqrt{(\sigma_1 + \tau_2)^2 - 4(-1)^n}\right)$. (See [Per29]). The value of x follows easily, but is of no interest for our purpose. Let a/b be the $tn+v$–th convergent of x. Let $\mathbb{T}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a' \\ b' \end{pmatrix}$. Then a'/b' is the $(t-1)n+v$-th convergent of x. We construct $L_I(a, b)$ as follows. Let $q_0 = P_v$, $q_1 = Q_v$ and let $q_{k+1} = \alpha_k q_k + q_{k-1}$. Then we have $a = q_{tn+v}$, $b = q_{tn+v+1}$, $a' = q_{(t-1)n+v}$, $b' = q_{(t-1)n+v+1}$. (See [Kra90], [Per29]). Notice that $l_I(q_k, q_{k+1}) =$

$l_I(q_{k-1}, q_k) + l(\alpha_k) + 1$. Hence $l_I(a, b) = l_I(a', b') + \sum_{i=1}^{n}(l(\alpha_i)+1)$. In order to prove the theorem we have to prove that $\sum_{i=1}^{n}(l(\alpha_i)+1) = l_I(a, b) - l_I(a', b')$ $= \hat{\rho}.\log(b/b')$. Hence we must show that $b'/b \to \theta$ if $t \to \infty$. This is clear since $a/b \to x$ if $t \to \infty$, hence b'/b converges to ξ and $\theta = \xi$. □

Note. The number ξ is in fact the number μ_k of formula (14).

In Table IV we give some related coefficients ρ, in the cases that n=1 and n=2. This table can be interpreted in the following way. Let (a, b) be an integer pair such that a / b is a convergent of x.

Table IV.

cont. fract. (x)	alg. expr. (x)	num. value (x)	alg. expr. (ρ)	num. value (ρ)
$[0,\overline{1}]$	$\frac{1}{2}(\sqrt{5}-1)$	0.62···	$\frac{1}{\log(\sqrt{5}/2+1/2)}$	1.44···
$[0,\overline{2}]$	$\sqrt{2}-1$	0.41···	$\frac{2}{\log(\sqrt{2}+1)}$	1.57···
$[0,\overline{3}]$	$\frac{1}{2}(\sqrt{13}-3)$	0.30···	$\frac{3}{\log(\sqrt{13}/2+3/2)}$	1.74···
$[0,\overline{4}]$	$\sqrt{5}-2$	0.24···	$\frac{1}{\log(\sqrt{5}/2+1/2)}$	1.44···
$[0,\overline{5}]$	$\frac{1}{2}(\sqrt{29}-5)$	0.19···	$\frac{4}{\log(\sqrt{29}/2+5/2)}$	1.68···
$[0,\overline{6}]$	$\sqrt{10}-3$	0.16···	$\frac{4}{\log(\sqrt{10}+3)}$	1.52···
$[0,\overline{7}]$	$\frac{1}{2}(\sqrt{53}-7)$	0.14···	$\frac{5}{\log(\sqrt{53}/2+7/2)}$	1.76···
$[0,\overline{11}]$	$\frac{1}{2}(5\sqrt{5}-11)$	0.09···	$\frac{6}{5\log(\sqrt{5}/2+1/2)}$	1.72···
$[0,\overline{1,2}]$	$\sqrt{3}-1$	0.73···	$\frac{3}{\log(\sqrt{3}+2)}$	1.58···
$[0,\overline{1,3}]$	$\frac{1}{2}(\sqrt{21}-3)$	0.79···	$\frac{4}{\log(\sqrt{21}/2+5/2)}$	1.77···
$[0,\overline{1,4}]$	$2\sqrt{2}-2$	0.83···	$\frac{2}{\log(\sqrt{2}+1)}$	1.57···
$[0,\overline{1,5}]$	$\frac{1}{2}(3\sqrt{5}-5)$	0.85···	$\frac{5}{4\log(\sqrt{5}/2+1/2)}$	1.80···
$[0,\overline{1,6}]$	$\sqrt{15}-3$	0.87···	$\frac{5}{\log(\sqrt{15}+4)}$	1.68···
$[0,\overline{1,7}]$	$\frac{1}{2}(\sqrt{77}-7)$	0.89···	$\frac{6}{\log(\sqrt{77}/2+9/2)}$	1.904···
$[0,\overline{1,11}]$	$\frac{1}{2}(\sqrt{165}-11)$	0.92···	$\frac{7}{\log(\sqrt{165}/2+13/2)}$	1.896···
$[0,\overline{2,7}]$	$\frac{1}{2}(\sqrt{63}-7)$	0.47···	$\frac{7}{\log(\sqrt{63}+8)}$	1.75···

Note. From Table IV we deduce that $\hat{\rho}^{+}_{I} \geq 1.44\cdots$, which is the case if a is a convergent of $x = [0,\overline{1}] = \frac{1}{2}(\sqrt{5}-1)$. We conjecture $\hat{\rho}^{+}_{I} = 1.44\cdots$. As a consequence we find that $\hat{\rho}^{+}_{W(2),I} \geq 0.72\cdots$.

7.3.2 UPPER BOUNDS FOR $L_S(a,b)$ AND $L_S(a)$.

In the case that subtractions are allowed, we can decrease the upper bound. This bound will be given in Theorem 13. We need some definitions and a technical lemma before.

Definition. We consider the generalized continued fractions:

$$[\delta,\varepsilon_1\alpha_1,\varepsilon_2\alpha_2, \dots] = \delta + \cfrac{\varepsilon_1}{\alpha_1 + \cfrac{\varepsilon_2}{\alpha_2 + \dots}},$$

where $\delta = 0$ or 1, $\varepsilon_i = \pm 1$ and α_i are positive integers (cf. [Kra90], [Jag85]). Each $\varepsilon_i < 0$ corresponds to a subtraction. We define S_{-r} by $S_{-r} = \begin{pmatrix} r & -1 \\ 1 & 0 \end{pmatrix}$. Now we can define $T_S(a, b)$ (see Section 4.6) by

$$T_S(a,b) = \begin{cases} \left[S_r \begin{pmatrix} a \\ b \end{pmatrix}\right]^T & \text{if } r \le \frac{b}{a} \le r + \frac{1}{2}, \\ \left[S_{-r} \begin{pmatrix} a \\ b \end{pmatrix}\right]^T & \text{if } r - \frac{1}{2} < \frac{b}{a} < r. \end{cases}$$

Lemma 13. *We have the following identities*:

(*i*) $[\dots, \alpha, \beta, \dots] = [\dots, \alpha+1, -1, \beta-1, \dots]$ for $\beta \ge 2$,
(*ii*) $[\dots, \alpha, 1, \beta, \dots] = [\dots, \alpha+1, -\beta-1, \dots]$,
(*iii*) $[\dots, \alpha, 1, \beta, 1, \gamma, \dots] = [\dots, \alpha+1, -\beta-2, -\gamma-1, \dots]$.

Proof. The identities are a consequence of the matrix products (cf. [Kra90, p. 79])

$$S_\beta \cdot S_\alpha = S_{\beta-1} \cdot S_{-1} \cdot S_{\alpha+1},$$
$$S_\beta \cdot S_1 \cdot S_\alpha = S_{-\beta-1} \cdot S_{\alpha+1},$$
$$S_\gamma \cdot S_1 \cdot S_\beta \cdot S_1 \cdot S_\alpha = S_{-\gamma-1} \cdot S_{-\beta-2} \cdot S_{\alpha+1}. \quad \square$$

Theorem 14. (*i*) $l_{S,0}(a, b) \le 2 \log b$.
(*ii*) $l_{S,K(2)}(a, b) \le 1.86 \log b$.
Proof. (*i*) This proof is analogue to the proof of Theorem 7.
(*ii*) This proof is comparable to the proof of Theorem 10. We need to distinguish 58 cases. The interested reader can consider Table X in the appendix in Section 13. We use the identities of Lemma 13 (*i*) in the cases that $(\alpha, \beta) = (7, 2), (7, 3), (7, 5), (7, 7), (11, 2), (11, 3)$ and $(11, 7)$ and the identity of Lemma 13 (*ii*) in the cases that $(\alpha, \beta) = (3, 3), (3, 7), (3, 11), (5, 3), (5, 5), (5, 7), (5, 11), (7, 1), (7, 2), (7, 3), (7, 5), (7, 7), (7, 11), (7, 13), (7, 19), (11, 1), (11, 2), (11, 3), (11, 5), (11, 7), (11, 11), (11, 13)$ and $(11, 19)$. $\square$

The upper bound $\hat{\rho} = 1.86$ is just an indication. We can improve this bound. As shown in Table IV and Table V, we will probably be able to find improvements for $\hat{\rho} > 1.74$ ($x = [0, \overline{3}]$).

Table V. Lower upper bounds using the Subtraction Algorithm.

cont. frac.	equiv. expr.	improvement	alg. expr.	num. value
$[0,\overline{7}]$	$[0,\overline{8,-1,6}]$	$\frac{9}{10}$	$\frac{9}{10}\cdot\frac{5}{\log(\sqrt{53}/2+7/2)}$	1.54⋯
$[0,\overline{11}]$	$[0,\overline{12,-1,10}]$	$\frac{11}{12}$	$\frac{11}{12}\cdot\frac{6}{5\log(\sqrt{5}/2+1/2)}$	1.57⋯
$[0,\overline{1,3}]$	$[0,\overline{1,4,-4}]$	$\frac{7}{8}$	$\frac{7}{8}\cdot\frac{4}{\log(\sqrt{21}/2+5/2)}$	1.55⋯
$[0,\overline{1,5}]$	$[0,\overline{1,6,-6}]$	$\frac{9}{10}$	$\frac{9}{10}\cdot\frac{5}{4\log(\sqrt{5}/2+1/2)}$	1.62⋯
$[0,\overline{1,6}]$	$[0,1,4,\overline{-8}]$	$\frac{4}{5}$	$\frac{4}{5}\cdot\frac{5}{\log(\sqrt{15}+4)}$	1.34⋯
$[0,\overline{1,7}]$	$[0,\overline{1,8,-8}]$	$\frac{3}{4}$	$\frac{3}{4}\cdot\frac{6}{\log(\sqrt{77}/2+9/2)}$	1.55⋯
$[0,\overline{1,11}]$	$[0,\overline{1,12,-12}]$	$\frac{11}{14}$	$\frac{11}{14}\cdot\frac{7}{\log(\sqrt{165}/2+13/2)}$	1.62⋯
$[0,\overline{2,7}]$	$[0,2,\overline{8,-1,1}]$	$\frac{6}{7}$	$\frac{6}{7}\cdot\frac{7}{\log(\sqrt{63}+8)}$	1.50⋯

Corollary 15. We have the upper bound $l_S(a) \leq 1.43 \log a + 1$.
Proof. See Corollary 9.

7.4 The upper bound for the Batch-RSA Algorithm.

Before starting this section we will define E' and give an approximation for this number in the case that this algorithm will often be used. Suppose $e_1 > e_2 > \ldots > e_n$. Let $F(t) = \prod \max(b_i, c_i)$, where the product is taken over all 2^{h-t} crossing-over pairs at level t. Notice that $F(t)$ consists of a product of $\lceil \frac{1}{2}(n+1) \rceil$ factors e_i. Let $E' = \max_{1 \leq t \leq h} F(t)$. We approximate E' by $E' = \prod_{i=1}^{[\frac{1}{2}(n+1)]} e_i$. We want to express E' in terms of E. In general this is impossible. Therefore we make the following assumption.

Assumption 1. $\log E' = \frac{1}{2} \log E + \delta n$, where $\frac{1}{2} < \delta < 1$.

Assumption 2. $\log E \leq \frac{n}{n-1} \log a_n$.

Motivation for the assumptions. In one of the applications of the algorithm (see [CAB...90]) the numbers e_i are prime numbers. Suppose that $e_i = p_{\frac{3}{2}n-i+1}$, the $(\frac{3}{2}n - i + 1)^{\text{-th}}$ prime number. If we make the raw approximation $e_i \approx (\frac{3}{2}n - i + 1)\log(\frac{3}{2}n - i + 1)$ then we find

$$\log E \approx \sum_{i=1}^{n} \log(\tfrac{1}{2}n+i) + \sum_{i=1}^{n} \log\log(\tfrac{1}{2}n+i) \approx \tfrac{3}{2}n\cdot\log\tfrac{3}{2}\,n - \tfrac{1}{2}n\cdot\log\tfrac{1}{2}n$$

and

$$\log E' \approx \sum_{i=\frac{1}{2}(n+1)}^{n} \log(\tfrac{1}{2}n+i) + \sum_{i=\frac{1}{2}(n+1)}^{n} \log\log(\tfrac{1}{2}n+i) \approx \tfrac{3}{2}n\cdot\log\tfrac{3}{2}\,n - n\cdot\log n.$$

Hence $\log E' - \frac{1}{2}\log E \approx \frac{3}{4}n\cdot\log\frac{3}{2} + \frac{1}{4}n\cdot\log\frac{1}{2} = \delta n$.

For the second assumption observe that $e_n < E^{1/n}$. Hence $a_n = E/e_n > E^{(n-1)/n}$. □

Lemma 16. We have under the assumption given above

(*i*) $l_B[a_1, \ldots, a_n] \le \frac{3}{2}\cdot\frac{n}{n-1}\cdot\lceil\log n\rceil\cdot\log a_n + 2^{\lceil\log n\rceil}$,

(*ii*) $l_{B,I}[a_1, \ldots, a_n] \le \frac{n}{n-1}\cdot\lceil\log n\rceil\cdot\log a_n + 2n\cdot\lceil\log n\rceil + 2^{\lceil\log n\rceil}$.

Proof. (*i*) We assume that $n = 2^h$. We need h steps. During step t we have to do 2^{h-t} "crossing-overs". For each crossing-over we need a chain for both scalars b and c (see Figure 2). In step t we need chains of lengths $l(b)$, $l(c)$ and we need 1 element to sum $b\mathbf{z}$ and $c\mathbf{y}$. Using the remark of Section 7.1 we can approximate $l(b) \le \frac{3}{2}\cdot\log(b)$. Finally we have

$$l_B[a_1, \ldots, a_n] \le \sum_{t=1}^{h} (\tfrac{3}{2}\log E + 2^{h-t}) = \tfrac{3}{2}\lceil\log n\rceil\cdot\log E + 2^{\lceil\log n\rceil}.$$

With similar technics as written in Section 7.1 we find for $b \ge 2^{512}$ that $l(b) \le 1.2\cdot\log b$. The second assumption yields Lemma 16 (*i*).

(*ii*) Instead of constructing two addition chains for b and c, we construct a vector addition chain for $[b, c]$. This can be done with 2 log c elements. We make a fair approximation by saying that during each step t we need $\sum l[b, c] \approx 2\log E'$ elements where E' was defined above. Using the approximation given in the assumption above, we get

$$l_{B,I}[a_1, \ldots, a_n] \le \sum_{t=1}^{h} (2\log E' + 2^{h-t}) \le \lceil\log n\rceil\cdot(\log E + 2n) + 2^{\lceil\log n\rceil}.$$ □

Note. Using the conjecture at the end of Section 7.3.1 we find that $\hat{\rho}^{+}_{B,K(2)} = \frac{1}{2}\cdot\frac{n}{n-1}\cdot\lceil\log n\rceil$ and $\hat{\rho}^{+}_{B,I} \ge 0.72\cdot\lceil\log n\rceil$.

7.5 The upper bounds for the algorithms of Straus and Yao.

We will first consider the Algorithm of Straus. As in Section 7.1 we divide the elements of the chain into three subsets. The first subset contains the set V. But we have to omit the vectors $\mathbf{f}_i$ and the $\mathbf{0}$-vector.

		# elements
(*i*)	the set V of vectors $\mathbf{v}_j$	$(2^{nk}-n-1)$
(*ii*)	the vectors $\mathbf{u}$ for which $\mathbf{u}/2$ is in the chain	$k\cdot[\lambda(a_n)/k]$
(*iii*)	vectors which are the sum of a vector of (*i*) and a vector of (*ii*)	$[\lambda(a_n)/k]$
	TOTAL:	$l_{St(k)}[a_1, \dots, a_n] \le (k+1)[\lambda(a_n)/k] + 2^{nk}-n-1$

We find $l^+_{St(k)}[a_1, \dots, a_n] \le [\lambda(a_n)/k] + 2^{nk}-2n-1$.

In the case of Yao's Algorithm, we only consider the upper bound of the length of a vector addition chain.

		# elements
(*i*)①	the vectors $l\cdot\mathbf{f}_i$,	$n(2^k-2)$
(*i*)②	the vectors $\mathbf{w}_j$	$(n-1)([\lambda(a_n)/k]+1)$
(*ii*)	the vectors $\mathbf{u}$ for which $\mathbf{u}/2$ is in the chain	$k\cdot[\lambda(a_n)/k]$
(*iii*)	vectors which are the sum of a vector of (*i*) and a vector of (*ii*)	$[\lambda(a_n)/k]$
	TOTAL:	$l_{Y(k)}[a_1, \dots, a_n] \le (n+k)[\lambda(a_n)/k] + n2^k-1-n$

We have $l^+_{Y(k)}[a_1, \dots, a_n] \le n[\lambda(a_n)/k] + n2^k-1-2n$. We leave to the reader the proof that $l_{Y(k)}(a_1, \dots, a_n) = l_{Y(k)}[a_1, \dots, a_n] - n+1$.

7.6 The upper bound for the algorithm of Fiat.

For Fiat's Algorithm we have the following calculation of the length of the addition sequence.

		# elements
(*i*)	chain with the numbers $1, \dots, 2^{\lambda(a_n)}$	$\lambda(a_n)$
(*ii*)	chain to $c_1, \dots, c_{2^n-1}$	$\lambda(a_n) + 1 - (2^n-1)$
(*iii*)	chains to $a_1, \dots, a_n$	$2^{n+1}-2n-2$
	TOTAL:	$l_F[a_1, \dots, a_n] \le 2\lambda(a_n) + 2^n - 2n$

Ad (*iii*): In principle we need $n(2^n-1)$ elements. But this number can be reduced using some addition-"tricks".

8. AVERAGE VALUES $\bar{\rho}_\Theta$.

Let $\bar{\rho}_\Theta(\alpha,n)$ be defined by

$$\bar{\rho}_\Theta(\alpha,n) = \frac{1}{\Sigma} \cdot \sum_{(a_1, \dots, a_n)} \frac{l_\Theta(a_1, \dots, a_n)}{\log_2 a_n}, \tag{5}$$

where the sum is taken over all n-tuples $(a_1, \dots, a_n)$ with $\lambda(a_n) = \alpha$ and Σ is the number of those n-tuples $(a_1, \dots, a_n)$. We call $\bar{\rho}_\Theta(\alpha,n)$ the *average value* of sequences/chains of width n . We will abbreviate this notation to $\bar{\rho}_\Theta$. We will now study approximations of the form

$$l(a_1, \dots, a_n) \approx \bar{\rho}_\Theta \log a_n.$$

We will also consider the average values $\bar{\rho}^{\square}{}_\Theta(\alpha, n)$ and $\bar{\rho}^{+}{}_\Theta(\alpha, n)$.

8.1. ERGODIC THEORETICAL BACKGROUND

We will now give an approximation of the average value $\bar{\rho}$ in the case that we use Algorithm I for n=2 .We have the following theorem.

Theorem 17. *We have the following average value using the Continued Fraction Algorithm*

$$\bar{\rho}_{I,0}(\alpha,2) = 1.60810\cdots$$

and

$$\bar{\rho}^{+}{}_{I,0}(\alpha,2) = 0.87586\cdots.$$

Hence $l_{I,0}(a, b) \approx 1.60810\cdots \log a_n$ and $l^{+}{}_{I,0}(a, b) \approx \bar{\rho}^{+}{}_{I,0} \cdot \log a_n$.

Proof. Let L_I, I_I,T_I, be as defined in Section 4.8. Hence for positive integers a, b we can express the addition sequence in terms of I_I and T_I by

$$L_I(a,b) = \bigcup_{k=1}^{n(a,b)} I_I T_I^k(a,b).$$

Here is $n(a,b)$ the number such that $T^{n(a,b)}(a,b) = (0,1)$. Instead of the pair (a,b) we will consider the quotient a / b. We consider the mapping

$\tilde{T}_I : [0,1) \longrightarrow [0,1)$ which is defined by $\tilde{T}_I(x) = \frac{1}{x} - \left\lfloor \frac{1}{x} \right\rfloor$. We have

$\tilde{T}_I(\frac{a}{b}) = \frac{b}{a} - r = \frac{b-ar}{a} = \frac{a'}{b'}$. Notice that $b = \prod_{k=0}^{n(a,b)} 1/\tilde{T}_I^{k}(a/b)$ and

$l_I(a,b) = \sum_{k=0}^{n(a,b)} \left(l_0([\tilde{T}_I^k(a/b)]) + 1 \right)$. We calculate $\bar{\rho}$ by

$$\bar{\rho} = \frac{l_I(a,b)}{\log(b)} = \frac{\sum_{k=1}^{n(a,b)} l_0([1/\tilde{T}_I^k(a/b)]) + 1}{\log \prod_{k=1}^{n(a,b)} 1/\tilde{T}_I^k(a/b)} = \lim_{n\to\infty} \frac{\frac{1}{n}\sum_{k=1}^{n} l_0([1/\tilde{T}_I^k(x)]) + 1}{-\frac{1}{n}\sum_{k=1}^{n} \ln \tilde{T}_I^k(x)/\ln 2}$$

In the last step we put the number x instead of a / b. Ergodic theory shows how the numerator and denominator can be calculated. In general Ergodic theory says that under certain conditions we have

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} f(\tilde{T}_I^k(x)) = \frac{1}{\ln 2} \int_0^1 \frac{f(x)}{1+x} dx.$$

The denominator is well known. We have

$$\frac{1}{n} \sum_{k=1}^{n} \ln \tilde{T}_I^k(x) = \frac{1}{\ln 2} \int_0^1 \frac{\ln(x)}{1+x} dx = -\frac{\pi^2}{12 \ln 2}.$$

(cf. [Knu81] or [Bil65]). We get for the numerator a less pleasant expression:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} l_0([1/\tilde{T}_I^k(x)]) + 1 = \frac{1}{\ln 2} \int_0^1 \frac{l_0([1/x]) + 1}{1+x} dx =$$

$$\int_{\frac{1}{n+1}}^{\frac{1}{n}} \frac{l_0([1/x])}{1+x} dy = 1 + \frac{1}{\ln 2} \sum_{n=1}^{\infty} l_0(n) \left(\ln\left(1+\tfrac{1}{n}\right) - \ln\left(1+\tfrac{1}{n+1}\right) \right).$$

Using the identity

$$\sum_{n=1}^{\infty} f(n)(g(n) - g(n+1)) = f(1)g(1) + \sum_{n=2}^{\infty} g(n)(f(n) - f(n-1)) \qquad (21)$$

we can evaluate the sum in the expression above in the following way:

$$\sum_{n=1}^{\infty} l_0(n) \left(\ln\left(1+\tfrac{1}{n}\right) - \ln\left(1+\tfrac{1}{n+1}\right) \right) = \sum_{n=2}^{\infty} (l_0(n) - l_0(n-1)) \ln\left(\frac{n+1}{n}\right).$$

We define two functions $p(n)$ and $q_k(n)$ as follows.

$$p(n) = \begin{cases} 1 \text{ if } n = 2^m,\ m \geq 1, \\ 0 \text{ else}. \end{cases}$$

and

$$q_k(n) = \begin{cases} 1 \text{ if } n = (2m+1)2^k,\ m \geq 1, \\ -1 \text{ if } n = (2m+2)2^k,\ m \geq 1, \\ 0 \text{ else}. \end{cases}$$

We can express the difference $l_0(n) - l_0(n\text{-}1)$ as

$l_0(n) - l_0(n-1) = p(n) + \sum_{k=0}^{\lambda(n)} q_k(n)$. A calculation gives

$$\sum_{n=2}^{\infty} (l_0(n) - l_0(n-1)) \ln\left(\frac{n+1}{n}\right) = \sum_{n=2}^{\infty} \left(p(n) + \sum_{k=0}^{\lambda(n)} q_k(n)\right) \ln\left(\frac{n+1}{n}\right)$$

$$= \sum_{k=1}^{\infty} \ln\left(\frac{2^k+1}{2^k}\right) + \sum_{k=0}^{\infty} \sum_{m=1}^{\infty} \left\{ \ln\left(\frac{(2m+1)2^k+1}{(2m+1)2^k}\right) - \ln\left(\frac{(2m+2)2^k+1}{(2m+2)2^k}\right) \right\}$$

$$= \ln\left\{ \frac{1}{2} \prod_{k=0}^{\infty} \left((1+2^{-k}) \prod_{m=1}^{\infty} \frac{\left(m+\frac{1}{2}+2^{-k-1}\right)(m+1)}{\left(m+1+2^{-k-1}\right)\left(m+\frac{1}{2}\right)} \right) \right\}$$

$$= \ln\left\{ \frac{1}{2} \prod_{k=0}^{\infty} \left((1+2^{-k}) \frac{\Gamma(\frac{3}{2})\Gamma(2+2^{-k-1})}{\Gamma(2)\Gamma(\frac{3}{2}+2^{-k-1})} \right) \right\}$$

$$= \ln\left\{ \frac{1}{2} \prod_{k=0}^{\infty} \left((1+2^{-k-1}) \frac{\Gamma(\frac{1}{2})\Gamma(1+2^{-k-1})}{\Gamma(\frac{1}{2}+2^{-k-1})} \right) \right\}$$

Here we will use the duplication formula for the Γ-function (cf. [AS70], p..256). In particular we have for $z = 1/2 + 2^{-k-1}$:

$$2^{2^{-k}} \Gamma\left(\frac{1}{2}+2^{-k-1}\right) \Gamma(1+2^{-k-1}) = \Gamma(1+2^{-k}) \Gamma\left(\frac{1}{2}\right).$$

Using this result we get

$$\frac{1}{2} \prod_{k=0}^{\infty} \left((1+2^{-k-1}) \frac{\Gamma(\frac{1}{2})\Gamma(1+2^{-k-1})}{\Gamma(\frac{1}{2}+2^{-k-1})} \right) = 2 \prod_{k=0}^{\infty} \left((1+2^{-k-1}) \frac{\Gamma^2(1+2^{-k-1})}{\Gamma(1+2^{-k})} \right) = 2 \prod_{k=1}^{\infty} \Gamma(2+2^{-k}).$$

Another result expresses a Γ-function in a sum of ζ-functions (cf [AS70], p. 256).

$$\ln \Gamma(2+z) = z(1-\gamma) + \sum_{m=2}^{\infty} \frac{(-1)^m}{m} (\zeta(m) - 1) z^m.$$

where $\gamma = 0.57\cdots$ is Euler's constant. Using this result we get

$$\ln\left\{ \frac{1}{2} \prod_{k=0}^{\infty} \left((1+2^{-k-1}) \frac{\Gamma(\frac{1}{2})\Gamma(1+2^{-k-1})}{\Gamma(\frac{1}{2}+2^{-k-1})} \right) \right\} = \ln 2 + 1 - \gamma + \sum_{m=2}^{\infty} \frac{(-1)^m (\zeta(m) - 1)}{m(2^m - 1)}.$$

Finally we find for $\bar{\rho}$ the expression

$$\bar{\rho} = \frac{1+\frac{1}{\ln 2}\sum_{n=1}^{\infty}(l_0(n)-l_0(n-1))\ln\left(\frac{n+1}{n}\right)}{\pi^2/12\,(\ln 2)^2} = \frac{12\ln 2}{\pi^2}\left\{1-\gamma+\ln 4+\sum_{m=2}^{\infty}\frac{(-1)^m(\zeta(m)-1)}{m(2^m-1)}\right\} = 1.60810\ldots.$$

We evaluate the numerator of equation (22) as follows. Since $\bar{\rho}_I(\alpha, 2) = \bar{\rho}^{\square}{}_I(\alpha, 2) + \bar{\rho}^{+}{}_I(\alpha, 2)$, we will calculate $\bar{\rho}^{\square}{}_I(\alpha, 2)$. Notice that $\bar{l}^{\square}{}_0(a) = [\log a]$. Hence

$$\bar{\rho}^{\square} = \frac{l^{\square}{}_I(a,b)}{\log(b)} = \lim_{n\to\infty}\frac{\frac{1}{n}\sum_{k=1}^{n}[\log(1/\tilde{T}_I^k(x))]}{-\frac{1}{n}\sum_{k=1}^{n}\ln\tilde{T}_I^k(x)/\ln 2}.$$

For the numerator we have

$$\lim_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n}[\log(1/\tilde{T}_I^k(x))] = \frac{1}{\ln 2}\int_0^1\frac{[\log(1/x)]}{1+x}dx = \frac{1}{\ln 2}\sum_{n=0}^{\infty}\int_{2^{-n-1}}^{2^{-n}}\frac{n}{1+x}dx =$$

$$= \frac{1}{\ln 2}\sum_{n=0}^{\infty}n(\ln(1+2^{-n})-\ln(1+2^{-n-1})) = \frac{1}{\ln 2}\sum_{n=1}^{\infty}\ln(1+2^{-n}).$$

Using identity of formula (21) we have

$$\sum_{n=1}^{\infty}\ln(1+2^{-n}) = \sum_{n=1}^{\infty}\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k\cdot 2^k} = \sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k\cdot(2^k-1)}.$$

Finally we get

$$\bar{\rho}^{+} = \bar{\rho}-\bar{\rho}^{\square} = \frac{12\ln 2}{\pi^2}\left\{1-\gamma+\ln 4+\sum_{m=2}^{\infty}\frac{(-1)^m(\zeta(m)-1)}{m(2^m-1)}-\sum_{m=1}^{\infty}\frac{(-1)^{m+1}}{m\cdot(2^m-1)}\right\} =$$

$$\frac{12\ln 2}{\pi^2}\left\{\ln 4-\gamma+\sum_{m=2}^{\infty}\frac{(-1)^m\zeta(m)}{m(2^m-1)}\right\} = 0.87586\ldots. \quad \square$$

Next, in this part of the section we determine the average value $\bar{\rho}_{S,0}$ (see Section 4.6). We assume that the computer is able to insert new elements in the chain/sequence by subtracting. We define for $x \in \mathbb{R}$: $]x[=[x+\frac{1}{2}]$. Our Theorem is a bit more general because we assume that one subtraction can be done with the costs of s additions. The final expression for $\bar{\rho}_{S,0}$ is an expression in which s appears. If s=1 then we get the value of $\bar{\rho}_{S,0}$ which expresses that the costs of additions and subtractions are equal.

Theorem 18. *The average value* $\bar{\rho}_{S,0}$ *is* $\bar{\rho}_{S,0} = 1.44902\cdots + 0.17861\cdots\, s$ *where* s *expresses the number of additions which can be done in the same time as one subtraction. (If* s=1 *we get* $\bar{\rho}_{S,0} = 1.62763\cdots$).

Proof. Let L_S, I_S, T_S, be as defined in Section 4.6. We can describe the addition

sequence which produces a and b by

$$L_S(a,b) = \bigcup_{k=1}^{n(a,b)} I_S T_S^{\,k}(a,b).$$

Here is $n(a,b)$ the number such that $T_S^{\,n(a,b)}(a,b) = (0,1)$. Instead of the pair (a,b) we will consider the quotient a / b. We consider the mapping

$\tilde{T}_S : [0,\frac{1}{2}) \longrightarrow [0,\frac{1}{2})$ which is defined by $\tilde{T}_S(x) = \left| \frac{1}{x} - \left]\frac{1}{x}\right[\right|$. We have

$\tilde{T}_S(\frac{a}{b}) = \left|\frac{b}{a} - r\right| = \left|\frac{b-ra}{a}\right| = \frac{a'}{b'}$. Notice that $b = \prod_{k=0}^{n(a,b)} 1/\tilde{T}_S^{\,k}(a/b)$ and

$l_S(a,b) = \sum_{k=0}^{n(a,b)} \left(l_0\left(\left] 1/\tilde{T}_S^{\,k}(a/b) \right[\right) + \sigma(\tilde{T}_S^{\,k}(a/b)) \right)$. Here we have

$\sigma(x) = \begin{cases} 1 & \text{if } \frac{1}{x} - \left]\frac{1}{x}\right[\geq 0 \\ s & \text{if } \frac{1}{x} - \left]\frac{1}{x}\right[< 0 \end{cases}$, where s is the number of subtractions. We calculate $\overline{\rho}$ by

$$\overline{\rho} = \frac{l_S(a,b)}{\log(b)} = \frac{\sum_{k=0}^{n(a,b)} l_0\left(\left] 1/\tilde{T}_S^{\,k}(a/b) \right[\right) + \sigma(\tilde{T}_S^{\,k}(a/b))}{\log \prod_{k=0}^{n(a,b)} 1/\tilde{T}_S^{\,k}(a/b)}$$

$$= \lim_{n\to\infty} \frac{\frac{1}{n+1}\sum_{k=0}^{n} l_0\left(\left] 1/\tilde{T}_S^{\,k}(x) \right[\right) + \sigma(\tilde{T}_S^{\,k}(x))}{-\frac{1}{n+1}\sum_{k=0}^{n} \ln \tilde{T}_S^{\,k}(x)/\ln 2}.$$

In the last step we put the number x instead of a / b. The ergodic theory predicts how the numerator and denominator can be calculated. In general the ergodic theory says that under some circumstances we have

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} f(\tilde{T}_S^{\,k}(x)) = \frac{1}{\ln \tau} \int_0^{\frac{1}{2}} \left\{ \frac{f(x)}{\tau^2 - x} + \frac{f(x)}{\tau + x} \right\} dx.$$

Especially the denominator is well known. We have

$$\frac{1}{n} \sum_{k=1}^{n} \ln \tilde{T}_S^{\,k}(x) = \frac{1}{\ln \tau} \int_0^{\frac{1}{2}} \left\{ \frac{\ln(x)}{\tau^2 - x} + \frac{\ln(x)}{\tau + x} \right\} dx = -\frac{\pi^2}{12 \ln \tau}.$$

(cf. [Rie79], [Jag85] or [Kra90]). We get for the numerator two expressions. The first expression can be calculated in the following way:

$$\frac{1}{n}\sum_{k=1}^{n}\sigma\left(\tilde{T}_S^{\,k}(x)\right)=\frac{1}{n}\sum_{k=0}^{n-1}\sigma\left(\tilde{T}_S^{\,k}(x)\right)=\frac{1}{\ln\tau}\int_0^{\frac{1}{2}}\left\{\frac{\sigma(\tilde{T}_S^{\,k}(x))}{\tau^2-x}+\frac{\sigma(\tilde{T}_S^{\,k}(x))}{\tau+x}\right\}dx$$

$$=\frac{1}{\ln\tau}\int_0^{\frac{1}{2}}\left\{\frac{s}{\tau^2-x}+\frac{1}{\tau+x}\right\}dx=\frac{1}{\ln\tau}\left(\ln\left(\frac{\tau^2}{2}\right)+s\cdot\ln\left(\frac{2}{\tau}\right)\right).$$

The other term is less pleasant:

$$\frac{1}{n}\sum_{k=0}^{n}l_0\left(\left\lfloor 1/\tilde{T}_S^{\,k}(x)\right\rfloor\right)=\frac{1}{\ln\tau}\int_0^{\frac{1}{2}}\left\{\frac{l_0\left(\lfloor 1/x\rfloor\right)}{\tau^2-x}+\frac{l_0\left(\lfloor 1/x\rfloor\right)}{\tau+x}\right\}dx$$

$$=\frac{1}{\ln\tau}\left(\int_{\frac{2}{5}}^{\frac{1}{2}}\left\{\frac{1}{\tau^2-x}+\frac{1}{\tau+x}\right\}dx+\sum_{n=3}^{\infty}l_0(n)\int_{m(n)}^{M(n)}\left\{\frac{1}{\tau^2-x}+\frac{1}{\tau+x}\right\}dx\right)$$

$$=\frac{1}{\ln\tau}\left(\int_0^{\frac{1}{2}}\left\{\frac{1}{\tau^2-x}+\frac{1}{\tau+x}\right\}dx+\sum_{n=3}^{\infty}(l_0(n)-l_0(n-1))\int_0^{M(n)}\left\{\frac{1}{\tau^2-x}+\frac{1}{\tau+x}\right\}dx\right)$$

$$=1+\frac{1}{\ln\tau}\sum_{n=3}^{\infty}(l_0(n)-l_0(n-1))\cdot\ln\left(\frac{n+g}{n+g-1}\right).$$

Here is $g=\frac{1}{2\tau^3}$, $m(n)=\frac{1}{n+\frac{1}{2}}$ and $M(n)=\frac{1}{n-\frac{1}{2}}$. We define the functions $p(n)$ and $q_k(n)$ as in section 1 and we express the difference $l_0(n)-l_0(n\text{-}1)$ as $l_0(n)-l_0(n-1)=p(n)+\sum_{k=0}^{\lambda(n)}q_k(n)$. A calculation gives

$$\sum_{n=3}^{\infty}(l_0(n)-l_0(n-1))\ln\left(\frac{n+g}{n+g-1}\right)$$

$$=\sum_{n=3}^{\infty}\left(p(n)+\sum_{k=0}^{\lambda(n)}q_k(n)\right)\ln\left(\frac{n+g}{n+g-1}\right)$$

$$=\sum_{k=2}^{\infty}\ln\left(\frac{2^k+g}{2^k+g-1}\right)+\sum_{k=0}^{\infty}\sum_{m=1}^{\infty}\left\{\ln\left(\frac{(2m+1)2^k+g}{(2m+1)2^k+g-1}\right)-\ln\left(\frac{(2m+2)2^k+g}{(2m+2)2^k+g-1}\right)\right\}$$

$$=\ln\left\{\prod_{k=2}^{\infty}\left(\frac{2^k+g}{2^k+g-1}\right)\prod_{k=1}^{\infty}\prod_{m=1}^{\infty}\frac{\left(m+\frac{1}{2}+g\cdot 2^{-k}\right)\left(m+1+(g-1)\cdot 2^{-k}\right)}{\left(m+\frac{1}{2}+(g-1)\cdot 2^{-k}\right)\left(m+1+g\cdot 2^{-k}\right)}\right\}$$

$$= \ln\left\{\prod_{k=1}^{\infty}\left(\frac{1+g\cdot 2^{-k-1}}{1+(g-1)\cdot 2^{-k-1}}\cdot\frac{\Gamma(\frac{3}{2}+(g-1)\cdot 2^{-k})\Gamma(2+g\cdot 2^{-k})}{\Gamma(\frac{3}{2}+g\cdot 2^{-k})\Gamma(2+(g-1)\cdot 2^{-k})}\right)\right\}.$$

Here we use the duplication formula for the Γ-function (cf. [AS70], p. 256). In particular we have, for $z = \frac{3}{2} - g2^{-k}$,

$$2^{2-g\cdot 2^{-k+1}}\cdot\Gamma(\tfrac{3}{2}+g\cdot 2^{-k})\cdot\Gamma(2+g\cdot 2^{-k}) = \Gamma(3+g\cdot 2^{-k+1})\Gamma(\tfrac{1}{2}).$$

Using this result we get

$$\prod_{k=1}^{\infty}\left(\frac{1+g\cdot 2^{-k-1}}{1+(g-1)\cdot 2^{-k-1}}\cdot\frac{\Gamma(\frac{3}{2}+(g-1)\cdot 2^{-k})\Gamma(2+g\cdot 2^{-k})}{\Gamma(\frac{3}{2}+g\cdot 2^{-k})\Gamma(2+(g-1)\cdot 2^{-k})}\right)$$

$$= 4\cdot\frac{1+\frac{1}{2}(g-1)}{1+\frac{1}{2}g}\prod_{k=1}^{\infty}\left(\frac{\Gamma(2+(g-1)\cdot 2^{-k+1})}{\Gamma(2+g\cdot 2^{-k+1})}\cdot\left\{\frac{\Gamma(2+g\cdot 2^{-k})}{\Gamma(2+(g-1)\cdot 2^{-k})}\right\}^2\right)$$

$$= \frac{4(1+g)}{2+g}\cdot\frac{\Gamma(1+g)}{\Gamma(2+g)}\cdot\prod_{k=1}^{\infty}\left(\frac{\Gamma(2+g\cdot 2^{-k}}{\Gamma(2+(g-1)\cdot 2^{-k}}\right) = \frac{4}{2+g}\cdot\prod_{k=1}^{\infty}\left(\frac{\Gamma(2+g\cdot 2^{-k}}{\Gamma(2+(g-1)\cdot 2^{-k}}\right).$$

We express the Γ-function in a sum of ζ-functions (cf [AS70], p. 256). We get

$$\ln\left\{\prod_{k=1}^{\infty}\frac{\Gamma(2+g\cdot 2^{-k}}{\Gamma(2+(g-1)\cdot 2^{-k}}\right\} = 1-\gamma+\sum_{m=2}^{\infty}\frac{(\zeta(m)-1)}{m}\cdot\frac{(-g)^m-(1-g)^m}{2^m-1}.$$

And finally we find for $\bar{\rho}$ the expression

$$\bar{\rho} = \frac{1}{\pi^2/(12\ln 2\cdot\ln\tau)}\cdot\left\{1+\frac{1}{\ln\tau}\left[\sum_{n=3}^{\infty}(l_0(n)-l_0(n-1))\cdot\ln\left(\frac{n+g}{n+g-1}\right)+\ln\left(\frac{\tau^2}{2}\right)+s\cdot\ln\left(\frac{2}{\tau}\right)\right]\right\}$$

$$= \frac{12\ln 2}{\pi^2}\left\{\ln\tau+1-\gamma+\ln\left(\frac{4}{2+g}\right)+\sum_{m=2}^{\infty}\frac{(\zeta(m)-1)}{m}\cdot\frac{(-g)^m-(1-g)^m}{2^m-1}+\ln\left(\frac{\tau^2}{2}\right)+\ln\left(\frac{2}{\tau}\right)s\right\}$$

$$= \frac{12\ln 2}{\pi^2}\left\{1-\gamma+\ln 4+\sum_{m=2}^{\infty}\frac{(\zeta(m)-1)}{m}\cdot\frac{(-g)^m-(1-g)^m}{2^m-1}+\ln\left(\frac{2}{\tau}\right)s\right\}$$

$$= 1.44902\ldots+0.17861\ldots\cdot s. \quad \square$$

At the moment of writing this paper we were able to apply the ergodic theory on $L_I(a_1, \ldots, a_n)$ and to calculate the average value $\bar{\rho}_I(\alpha, n)$. The reader who is interested has to inform the author.

8.2 APPLICATIONS OF THESE AVERAGE VALUES TO $L_{W(2)}(a)$ AND $L_S(a)$.

The way to find these average values is comparable to the upper bounds mentioned in Corollary 9 and 13. We mention only the results.

Corollary 19. We have $\overline{l}_{W(2)}(\alpha, n) = \overline{\rho}_{W(2)} \cdot \alpha$ and $\overline{l}_S(\alpha, n) = \overline{\rho}_S \cdot \alpha$, where $\overline{\rho}_{W(2)} = 1.30405\cdots$ and $\overline{\rho}_S = 1.31382\cdots$. For the number of additions we have $\rho_{W(2)} = 0.43793\cdots$.

8.3 OTHER AVERAGE VALUES.

This section contains the average value calculations of the Small Window Algorithm, the Generalized Window Algorithm of Yao and the Batch-RSA Algorithm. We do not have an apart subsection for the Generalized Algorithm of Fiat, since the calculations in the general case are very hard and the generalization can only be applied in a reasonable way for sequences and chains which have a width smaller than 6. In [Fia89] Fiat gives the average value $l_F(a, b) \approx 1.75 \log b$.

8.3.1 UPPER BOUNDS FOR THE SMALL WINDOW ALGORITHM AND THE ALGORITHM OF MORAIN AND OLIVOS.

In the case of the Subtraction Algorithms of Morain and Olivos the average values can be calculated using Markov Chains (cf. [MO89]). We give the results:

$l_{MOA}(a) \approx \frac{11}{8}\lambda(a)$, $l^+_{MOA}(a) \approx \frac{3}{8}\lambda(a)$, $l_{MOB}(a) \approx \frac{4}{3}\lambda(a)$ and $l^+_{MOB}(a) \approx \frac{1}{3}\lambda(a)$.

The average length of a chain produced by the Small Window Algorithm is in general a bit smaller than the upper bound predicts. As in Section 7.1 we consider 3 subsets. Only the number of elements in subset (*iii*) will decrease in general. This number corresponds to the number of windows (which is at most $[\lambda(a_n)/k] + 1$). We expect after each window v zeroes with probability 2^{-k}, if $k > 0$ and 0 zeroes with probability $\frac{1}{2}$. On average we have 1 zero after each window. Hence the number of windows is $[\lambda(a_n)/(k+1)] + 1$. Hence we have the following scheme for $k > 1$.

		# elements
(*i*)	the elements 2, 3, 5, 7, ... , 2^k-1,	2^{k-1}
(*ii*)	the numbers n for which $n/2$ is in the chain or the number which is the sum of two numbers of (*i*)	$\lambda(a)-k+1$
(*iii*)	numbers which are the sum of a number of (*i*) and a number of (*ii*)	$[\lambda(a) / (k+1)]$
	TOTAL:	$l_{K(k)}(a) \approx [(1 + 1/(k+1))\lambda(a)] + 2^{k-1}-k + 1$

Hence $l^{+}{}_{K(k)}(a) \approx [\ 1/(k+1)\lambda(a)] + 2^{k-1}-k$. For the Binary Algorithm we find $l_0(a) \approx \frac{3}{2}\ \lambda(a)$ and $l_0(a) \approx \frac{1}{2}\ \lambda(a)$ (cf. [Knu81]).

8.3.2 The upper bound for the Batch-RSA Algorithm.

Lemma 20. *We have under the assumption given in Section* 7.4

(*i*) $l_{B,K(2)}[a_1, \dots, a_n] \approx \frac{4}{3} \cdot \frac{n}{n-1} \cdot \log a_n \cdot \log n + n.$

(*ii*) $l_{B,I}[a_1, \dots, a_n] \approx 0.8\cdots \frac{n}{n-1} \log n \log a_n + 1.6\cdots \delta n \log n + n$, *for a number* $\frac{1}{2} < \delta < 1$.

Proof.(*i*) We get in each step by approximation that the number of calculations is $\sum l_{K(2)}(a_i) \approx \frac{4}{3} \log E$. Using the fact that the number of steps is about $\log n$, we get

$$l_{B,K(2)}[a_1, \dots, a_n] \leq \frac{4}{3} \cdot \frac{n}{n-1} \log n \log E + n.$$

Now apply the approximation given in the assumption of Section 7.4.

(*ii*) We use the average value found in Theorem 17. Hence we get in step t by approximation $\sum l_I[b,c] \approx 1.608\cdots \log E'$ where E' is as defined in Section 7.4. Using the approximation given in the assumption of that section, we get

$$l_{B,I}[a_1, \dots, a_n] \leq 0.8\cdots \log n \log E + n(1 + 1.6\cdots \delta \log n + 1\ ). \square$$

Note. We have $\overline{\rho}^{+}{}_{B}[\alpha, n] = \frac{1}{3} \log n$ and $\overline{\rho}^{+}{}_{B,I}[\alpha, n] = 0.43793\cdots \cdot \log n$.

8.3.3 The Algorithm of Straus and Yao.

In the case of the Generalized Window Algorithm of Yao it is more complicated. There is a possibility that $l \cdot \mathbf{f}_i$ does not appear in a window or that $w_{ij} = 0$. Both situations decrease the length of the vector addition chain.

① $l\cdot\mathbf{f}_i$ does not appear in the windows w_{ij} (i.e. $w_{ij} \neq l$ for $0 \leq j \leq [\lambda(a_n)/k]$). The probability that this happens is $(1-2^{-k})^{[\lambda(a_n)/k]+1}$. Hence the probability that l appears is $1 - (1-2^{-k})^{[\lambda(a_n)/k]+1}$.

② $w_{ij} = 0$. The probability that $w_{ij} \neq 0$ is $1 - 2^{-k}$.

Using these modifications we have the following scheme

		# elements
(i)①	the vectors $l\cdot\mathbf{f}_i$,	$n(2^k-2)\cdot\left(1-(1-2^{-k})^{\lambda(a_n)/k+1}\right)$
(i)②	the vectors $\mathbf{w}_j$	$(n-1)(\lambda(a_n)/k+\frac{1}{2})\cdot(1-2^{-k})$
(ii)	the vectors $\mathbf{u}$ for which $\mathbf{u}/2$ is in the chain	$\lambda(a_n) - \frac{1}{2}k$
(iii)	vectors which are the sum of a vector of (i) and a vector of (ii)	$\lambda(a_n)/k - \frac{1}{2}$
	TOTAL:	$l_{Y(k)}[a_1, \ldots, a_n]$

We use an approximation for k which was mentioned in Section 3, namely

$$\lambda(a_n) \approx k^2\cdot 2^k\cdot \ln 2. \tag{22}$$

Using this approximation for k we find for large values of a_n for expression (i)①

$$\#\text{ vectors } l\cdot\mathbf{f}_i \approx n(2^k-3), \tag{23}$$

since $(1-2^{-k})^{\lambda(a_n)/k} \approx (1-2^{-k})^{\ln 2\cdot k\cdot 2^k} \approx e^{-\ln 2\cdot k} = 2^{-k}$. Approximation (23) follows since $(1-2^{-k})(2^k-2) \approx 2^k-3$. Using these approximations we find

		# elements
(i)①	the vectors $l\cdot\mathbf{f}_i$,	$n(2^k-3)$
(i)②	the vectors $\mathbf{w}_j$	$(n-1)(\lambda(a_n)/k+\frac{1}{2}-k\cdot\ln 2)$
(ii)	the vectors $\mathbf{u}$ for which $\mathbf{u}/2$ is in the chain	$\lambda(a_n) - \frac{1}{2}k$
(iii)	vectors which are the sum of a vector of (i) and a vector of (ii)	$\lambda(a_n)/k - \frac{1}{2}$
	TOTAL:	$l_{Y(k)}[a_1, \ldots, a_n] \approx (1+\frac{n}{k})\lambda(a_n)+n2^k-\frac{5}{2}n-kn\cdot\ln 2$

We did not make the calculations in the case of Straus' Algorithm.

9. Comparison of the Algorithms.

9.1 The Tables.

In the following tables we will compare the algorithms. We distinguish 6 columns. The first three columns give the name of the algorithm, the place in the text where we defined the algorithm and the notation of the algorithm respectively. The fourth column gives the number of memory locations used by the algorithm ($m_\Theta(a_1, \ldots, a_n)$). Table VI gives the values for m_Θ, $\hat{\rho}_\Theta$ and $\bar{\rho}_\Theta$ in the asymptotical case (i.e. $a_n \to \infty$). In Table VII we compare also $\hat{\rho}^+_\Theta$ and $\bar{\rho}^+_\Theta$. Table VIII gives the values for m_Θ, $\hat{l}_\Theta(512, 1)$ and a average value $\bar{l}_\Theta(512, 1)$. Table IX gives those values in the case of vector addition chains of width 32, with $\log a_{32} \approx 80$. In Table IX we did not consider the Algorithms of Straus and Fiat, since their memory usage was too large for normal applications. (In theory $> 4 \cdot 10^9$.)

Table VI, the asymptotical case

Name	Ref.	Notation	# mem. loc.	$\hat{\rho}_\Theta$	$\bar{\rho}_\Theta$
Binary	4.1	$L_0(a)$	1	2	1.5
Morain-Olivos	4.2	$L_{\mathrm{MO_A}}(a)$	2	1.6666⋯	1.375
Morain-Olivos	4.2	$L_{\mathrm{MO_B}}(a)$	2	1.5	1.3333⋯
Small Window	4.3	$L_{\mathrm{K}(2)}(a)$	2	1.5	1.3333⋯
Large Window	4.4	$L_{\mathrm{W}(2)}(a)$	2	1.5	1.3040⋯
Small Window	4.3	$L_{\mathrm{K}(k)}(a)$	2^{k-1}	$1+\frac{1}{k}$	$1+\frac{1}{k+1}$
Large Window	4.4	$L_{\mathrm{W}(k)}(a)$	n	$1+\frac{1}{\log n}$ †	?
Continued Fractions	4.8	$L_{\mathrm{I},0}(a, b)$	2	2	1.6080⋯
Continued Fractions	4.8	$L_{\mathrm{I}}(a, b)$	3	1.96	?
Fiat	4.9	$L_{\mathrm{F}}(a, b)$	3	2	1.75
Subtraction	4.6	$L_{\mathrm{S},0}(a, b)$	2	2	1.6273⋯
Subtraction	4.6	$L_{\mathrm{S,K}(2)}(a, b)$	3	1.86	?
Algorithm II	4.5	$L_{\mathrm{II}}(a, b)$	9	1.94⋯	1.58⋯ ‡
Algorithm III	4.7	$L_{\mathrm{III}}(a, b, c)$	10	3	?
Algorithm I	4.8	$L_{\mathrm{I}}(a_1, \dots, a_n)$	n	$1+\frac{n}{\log n}$ †	?
Fiat	4.9	$L_{\mathrm{F}}(a_1, \dots, a_n)$	2^n-1	2	$2-2^{-n}$
Straus	4.10.1	$L_{\mathrm{St}(k)}(a_1, \dots, a_n)$	$(2^{nk}-1)$	$1+\frac{1}{k}$	$1+\frac{1}{k}$
Yao	4.10.2	$L_{\mathrm{Y}(k)}(a_1, \dots, a_n)$	$n(2^k-1)$	$1+\frac{n}{k}$	$1+\frac{n}{k}$
Batch-RSA ❶	4.11	$L_{\mathrm{B,K}(2)}[a_1, \dots, a_n]$	n	$\frac{3n}{2(n-1)} \cdot \lceil \log n \rceil$	$\frac{4n}{3(n-1)} \cdot \log n$
B + I ❶	4.11	$L_{\mathrm{B,I}}[a_1, \dots, a_n]$	n	$\frac{n}{n-1} \cdot \lceil \log n \rceil$	$\frac{n \cdot 0.8040\dots}{n-1} \cdot \log n$

†: This result is given as a conjecture

‡: This result is determined from experiments

❶: This algorithm holds only in special cases

Table VII, the additions

Algorithm	$\hat{\rho}_{\Theta}$	$\bar{\rho}_{\Theta}$	$\hat{\rho}^{+}_{\Theta}$	$\bar{\rho}^{+}_{\Theta}$
$L_0(a)$	2	1.5	1	0.5
$L_{\mathrm{MOA}}(a)$	1.6666⋯	1.375	0.6666⋯	0.375
$L_{\mathrm{MOB}}(a)$	1.5	1.3333⋯	0.5	0.3333⋯
$L_{\mathrm{K}(2)}(a)$	1.5	1.3333⋯	0.5	0.3333⋯
$L_{\mathrm{W}(2)}(a)$	1.5	1.3040⋯	0.72⋯ †	0.4379⋯
$L_{\mathrm{K}(k)}(a)$	$1+\frac{1}{k}$	$1+\frac{1}{k+1}$	$\frac{1}{k}$	$\frac{1}{k+1}$
$L_{\mathrm{W}(k)}(a)$	$1+\frac{1}{\log n}$ †	?	?	?
$L_{\mathrm{I},0}(a, b)$	2	1.6080⋯	1.44⋯ †	0.8758⋯
$L_{\mathrm{I}}(a, b)$	1.96	?	1.44⋯ †	?
$L_{\mathrm{F}}(a, b)$	2	1.75	1	0.75
$L_{\mathrm{S},0}(a, b)$	2	1.6273⋯	?	?
$L_{\mathrm{S,K}(2)}(a, b)$	1.86	?	?	?
$L_{\mathrm{II}}(a, b)$	1.94⋯	1.58⋯ ‡	?	?
$L_{\mathrm{III}}(a, b, c)$	3	?	?	?
$L_{\mathrm{I}}(a_1, \dots, a_n)$	$1+\frac{n}{\log n}$ †	?	?	?
$L_{\mathrm{F}}(a_1, \dots, a_n)$	2	$2-2^{-n}$	1	$1-2^{-n}$
$L_{\mathrm{St}(k)}(a_1, \dots, a_n)$	$1+\frac{1}{k}$	$1+\frac{1}{k}$	$\frac{1}{k}$	$\frac{1}{k}$
$L_{\mathrm{Y}(k)}(a_1, \dots, a_n)$	$1+\frac{n}{k}$	$1+\frac{n}{k}$	$\frac{n}{k}$	$\frac{n}{k}$
$L_{\mathrm{B,K}(2)}[a_1, \dots, a_n]$ ❶	$\frac{3n}{2(n-1)}\cdot\lceil\log n\rceil$	$\frac{4n}{3(n-1)}\cdot\log n$	$\frac{n}{2(n-1)}\cdot\lceil\log n\rceil$	$\frac{n}{3(n-1)}\cdot\log n$
$L_{\mathrm{B,I}}[a_1, \dots, a_n]$ ❶	$\frac{n}{n-1}\cdot\lceil\log n\rceil$	$\frac{n\cdot 0.8040\dots}{n-1}\cdot\log n$	$\frac{n\cdot 0.72}{(n-1)}\cdot\lceil\log n\rceil$ †	$\frac{n\cdot 0.44}{(n-1)}\cdot\log n$

†: This result is given as a conjecture

‡: This result is determined from experiments

❶: This algorithm holds only in special cases

Table VIII, the case $\lambda(a) = 512$.

Name	Ref.	Notation	# mem. loc.	$\hat{l}_{\Theta}(512, 1)$	$\overline{l}_{\Theta}(512, 1)$
Binary	4.1	$L_0(a)$	1	1024	768
Morain-Olivos	4.2	$L_{MOA}(a)$	2	854	704
Morain-Olivos	4.2	$L_{MOB}(a)$	2	769	683
Small Window	4.3	$L_{K(2)}(a)$	2	769	683
Large Window	4.4	$L_{W(2)}(a)$	2	768	668
Small Window	4.3	$L_{K(5)}(a)$	16	626	609
Small Window	4.3	$L_{K(6)}(a)$	32	624	614
Large Window	4.4	$L_{W(16)}(a)$	32	645 †	?
Bos-Coster	[BoCo89]	-	32	610 ‡	605 ‡

†: This result is given as a conjecture

‡: This result is obtainned from experiments

Table IX. Vector addition chains in the case that n=32 and $\log a_n = 80$.

Name	Ref.	Notation	# mem. loc.	$\hat{l}_{\Theta}[80,32]$	$\overline{l}_{\Theta}[80,32]$
Algorithm I	4.8	$L_I(a_1, \ldots, a_{32})$	32	592 †	?
Yao	4.10.2	$L_{Y(3)}[a_1, \ldots, a_{32}]$	224	1133	1030
Yao	4.10.2	$L_{Y(4)}[a_1, \ldots, a_{32}]$	480	1199	1026
Batch-RSA ❶	4.11	$L_B[a_1, \ldots, a_{32}]$	32	652	590 ‡
B + I ❶	4.11	$L_{B,I}[a_1, \ldots, a_{32}]$	32	605	458

†: This result is given as a conjecture

‡: This result depends on the choice of δ (see Section 7.4)

❶: This algorithm holds only in special cases

9.2 Comparison of the Small and Large Window Algorithm.

In approximation both algorithms have the same properties: they use the same amount of memory , while the lengths of the chains are equal. This can be seen when we put $n = 2^k$.

9.3 Comparison of Algorithm I and Yao.

Table IX compares for several values of width n which algorithm produces the shortest vector addition chain given $\lambda(a_n)$. It is clear that for small numbers $\lambda(a_n)$ Algorithm I is better and for large numbers $\lambda(a_n)$ Yao's Algorithm wins. In the third column is written for which number of bits of a_n the two algorithms are comparable. The second column indicates the number of memory locations for Algorithm I. The fifth column gives the optimal window size for Yao's Algorithm. The last column indicates the number of memory locations for Yao's Algorithm.

Table X

n	$m_I(a_1, \ldots, a_n)$	equal	k	$m_{Y(k)}(a_1, \ldots, a_n)$
2	2	≈ 40	3	27
4	4	≈ 40	3	41
8	8	≈ 175	4	164
16	16	≈ 625	5	621

10. Further work and open problems.

Finally we finish with some open problems.

① How many memory locations are neede at least to produce a shortest chain $L(a)$, a shortest sequence $L(a_1, \ldots, a_n)$ and a shortest vector addition chain $L[a_1, \ldots, a_n]$?

② What is the theoretical asympthotical upper bound and average for $l(a_1, \ldots, a_n)/\log a_n$ given the restriction on the number of memory locations?

③ Which is the smallest pair (a, b) such that $l(a, b) < l^{(m)}(a,b)$ where m indicates that only m memory locations were used?

④ It would be interesting to find a proof of Conjecture 6.

⑤ As was mentioned in Section 8.1, we made recently progression in calculating $\bar{\rho}_I(\alpha, n)$ for arbitrary numbers α and n. These values are important to compare Algorithm I with other algorithms. Another interesting thing is to compare this average value with the upper bound.

⑥ In this report was suggested that the upper bounds $\hat{\rho}_{I,K(2)}(\alpha, 2)$ and $\hat{\rho}_{S,K(2)}(\alpha, 2)$ in Theorem 11 and Theorem 14 respectively could be decreased. What are the lower bounds of these upper bounds ?

⑦ Is it possible to improve the value of $\rho_S(\alpha, 2)$ which was found in Section 8.1 using the identities of Lemma 13.

⑧ Are there other algorithms which use less memory and produce shorter chains ?

ACKNOWLEDGEMENT

I am greatly indebted to Lambert Meertens for the work he investigated in this report. I greatfully thank Jurjen Bos, for our discussions and David Chaum, Eugène van Heyst and John Tromp for reading some parts of the manuscript. Furthermore I want to thank Cor Kraaikamp, Prof. H. Jager and Dr. H. C. P. Berbee for their ideas concerning the ergodic theory.

11. REFERENCES

[AS70] M. Abramowitz and I.A. Stegun: *Handbook of mathematical functions,* Dover Publications, Inc. New York, Ninth printing 1970.

[AMV90] G.B. Agnew, R.C. Mullin and S.A. Vanstone: *Arithmetic Operations in* GF(2^n), to appear in J. of Cryptology.

[B^{3}89] F. Bergeron, J. Berstel and S. Brlek: *A unifying approach to the generalisation of addition chains*, in preparation, 1989.

[B^{3}D89] F. Bergeron, J. Berstel, S. Brlek, C. Duboc: *Addition Chains Using Continued Fractions*, Journal of Algorithms **10**, (1989), 403-412.

[Bel63] R. Bellman: *Advanced problem 5125*, Amer. Math. Monthly **70** (1963), 765.

[Bil65] P. Billingsley, *Ergodic theory and information,* Wiley, New York, 1965, pp. 40-50.

[BO89] F. Bergeron, J. Olivos: *Vectorial addition chains using Euclid's algorithm*, Prel. version, Univ. Québec à Montréal, 1989.

[BoCo89] J.N.E. Bos and M.J. Coster, *Addition chain heuristics*, Proceedings Crypto '89, to be published.

[Bos90] J.N.E. Bos, On-the-fly generation of vector addition chains, to appear, 1990.

[Bra39] A. Brauer: *On addition chains*, Bull. Am. Math. Soc. **45** (1939), 736-739.

[BrCa89] S. Brlek and P. Castéran: *The addition chain control structure*, in preparation, 1989.

[CAB...] D. Chaum, H. van Antwerpen, H. den Boer, J.N.E. Bos, M.J. Coster, E. van Heyst, ... : *Efficient cash checks*, to appear, 1990.

[CBHMS] D. Chaum, H. den Boer, E. van Heyst, S. Mjølsnes and A. Steenbeek: *Efficient offline electronic checks*, Eurocrypt '89, to appear in Lecture Notes in Computer Science.

[Cot73] A. Cottrell: *A lower bound for the Scholz-Brauer problem*, Notices AMS, Abst. #73T-A200, **20** (1973), A-476.

[DL80] D. Dobkin and R. J. Lipton: *Addition chain methods for the evaluation of specific polynomials*, Siam J. Comput. **9** (1980), 121-125.

[DLS81] P. Downey, B. Leony and R. Sethi: *Computing sequences with addition chains*, Siam Journ. Comput. **3** (1981), 638-696.

[ElG] T. ElGamal: *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. Inf. Th. **31** (1985).

[Erd60] P. Erdös: *Remarks on number theory III, on addition chains*, Acta Arith. **6** (1960), 77-81.

[Fia89] A. Fiat: *Batch RSA*, Abstracts Crypto '89, to be published.

[GO89] R. Gouet and J. Olivos: *Asymptotic probabilistic analysis of a method for addition subtraction chains*, in preparation, 1989.

[GS78] A. A. Gioia and M. V. Subbarao: *The Scholz-Brauer problem in addition chains II*, Proc. eighth Manitoba conference on numerical math. and computing, (1978),251-274.

[Jag85] H. Jager: *Metrical results for the nearest integer continued fraction*, Indag. Math. **47** (1985), 417-427.

[Knu81] D. E. Knuth: *The art of computer programming*, Vol. 2, Seminumerical algorithms, Addison-Wesley, Reading, Mass., (1969), second edition 1981, pp. 339-364 and pp. 441-466.

[Kra90] C. Kraaikamp:*Metric and arithmetic results for continued fraction expansions*, Thesis, Amsterdam, 1990.

[Lee77] J. van Leeuwen: *An extension of Hansen's theorem for star chains*, J. Reine Angew. Math. **295** (1977), 203-207.

[Len86] H. W. Lenstra, jr.: *Factoring integers with elliptic curves*, Report 86-18, Universiteit van Amsterdam, 1986.

[MO88] F. Morain, J. Olivos: *Speeding up the computations on an elliptic curve using addition-subtraction chains*, preprint, INRIA, 1988.

[Oli81] J. Olivos: *On Vectorial Addition Chains*, J. of Algorithms **2** (1981), 13-21.

[Per29] O. Perron, *Die Lehre von den Kettenbrüchen*, Chelsea, New York 1929.

[Rie79] G.J. Rieger: *Mischung und Ergodizität bei Kettenbrüchen nach nächsten Ganzen*, J. reine angew. Math. **310** (1979), 171-181.

[Sch75] A. Schönhage: *A lower bound on the length of addition chains*, Theoret. Comput. Sci. **1** (1975), 1-12.

[Str64] E.G. Straus: *Addition chains of vectors*, Amer. Math. Monthly **71** (1964), 806-808 (problem 5125).

[TC87] Y. H. Tsai and Y. H. Chin: *A study of some addition chain problems*, Intern. J. Comp. Math. **22** (1987), 117-134.

[Thu73a] E. G. Thurber: *The Scholz-Brauer problem on addition chains*, Pacific J. Math. **49** (1973), 229-242.

[Thu73b] E. G. Thurber: *On addition chains $l(mn) \leq l(n)-b$ and lower bounds for $c(r)$*, Duke Math. J. **40** (1973), 907-913.

[Thu76] E. G. Thurber: *Addition chains and solutions of $l(2n) = l(n)$ and $l(2^n-1) = n+l(n)-1$*, Discr. Math. **16** (1976), 279-289.

[Veg75] E. Vegh: *A note on addition chains*, J. Comb. Th. (A) **19** (1975), 117-118.

[Vol85] H. Volger: *Some results on addition/subtraction chains*, Inf. Proc. Lett. **20** (1985), 155-160.

[VZ89] M. Veldhorst and H. Zantema: *Average length of addition chains*, personal communication, 1989.

[Wil82] H. C. Williams, *A p+1 method of factoring*, Math. Comp. **39** (1982), 225-234

[Yao76] A. Yao, *On the evaluation of powers*, Siam J. Comput. **5**, (1976).

[Zan89] H. Zantema: *Minimizing sums of addition chains*, Univ. of Utrecht, Report RUU-CS-89-15, 1989.

12. APPENDIX 1: EXAMPLES.

For some of the algorithms we give an extended example. We consider the vector addition chain for [385, 455, 715, 1001]. These numbers do appear in an cryptological application, (in the calculation of $X_1^{1/13}X_2^{1/11}X_3^{1/7}X_4^{1/5}$). We copied an example of the Small- and Large- window Algorithm from [BoCo89]. We consider the number $a =$ 26235947428953663183223 with binary representation 10110001110010000000⋯ ⋯11101001010011101010000001011110000011111001100101110111.

12.1. THE SUBTRACTION ALGORITHM OF MORAIN AND OLIVOS.

We have

$a=$ 10110001110010000001110100101001110101000000101111000001111100110010111011 1

$M^+=$ 11000010000010000010000100101010000101000000110000000010000001000001100000000

$M^-=$ 10000010000000000010000000000010000000000000001000000000100010000000100 1

Hence

		#elements
(*i*)	the numbers *n* for which *n*/2 is in the chain	74
(*ii*)	additions for M^+	16
(*iii*)	additions for M^-	8
(*iv*)	subtraction	1
	TOTAL:	99

12.2. THE SMALL WINDOW ALGORITHM.

We consider windows of length 3:

101 10001110010000001110100101001110101000000101 11100000111 11001100101 110111

5 1 7 1 3 1 5 7 5 5 7 7 3 3 5 3 7

We distinguish, as in Section 7.1, 3 types of elements in the addition chain

		#elements
(*i*)	the elements 2, 3, 5 and 7.	4
(*ii*)	the numbers n for which $n/2$ is in the chain	72
(*iii*)	numbers which are the sum of a number of (*i*) and a number of (*ii*)	16
	TOTAL:	92

12.3. The Large Window Algorithm.

We consider this algorithm with $n = 6$. (In [BoCo89] we did not consider the number of windows, but we considered the length of the largest window). However the length of the sequence for the six windows is equal to the length of the sequence in [BoCo89], the sequence itself differs, since this sequence was produced by Algorithm I, instead of Makechain Algorithm described in [BoCo89]. The algorithm works as follows:

```
10110001110010000001110100101001110101000000101111000001111100110010111011 1
          5689              933      117          47           499        375
```

The addition sequence yielding the first part is:

```
1 2 3 5 6 7 14 21 23 24 47 52 59 65 117 234 351 375 434 499
  933 1866 3732 3823 5689
```

We get the chain as in the Small Window Algorithm. In this case we find the length by:

(*i*)	length of the sequence for the 6 windows.	24
(*ii*)	the numbers n for which $n/2$ is in the chain	62
(*iii*)	numbers which are the sum of a number of (*i*) and a number of (*ii*)	5
	TOTAL:	91

This result is better than the result of the Small Window Algorithm.

12.4. Algorithm I.

On the following page of this report we give an application of Algorithm I on the vector [385, 455, 715, 1001]. We find $l_I[385, 455, 715, 1001] = 22$.

addition chain graph					operations		vector addition chain	counter
f_4	→	1001	→	286	LD 1		[0, 0, 0, 1]	
		↓						
f_3	→	715	→	260	ADD 2		[0, 0, 1, 1]	**1**
		↓						
f_2	→	455	→	70	ADD 3		[0, 1, 1, 1]	**2**
		↓						
f_1	→	385	→	99	ADD 4		[1, 1, 1, 1]	**3**
		↓						
1001	→	286	→	26	ADD 1	STO 1	[1, 1, 1, 2]	**4**
		↓						
715	→	260	→	62	ADD 2	STO 2	[1, 1, 2, 3]	**5**
		↓↓			DBL		[2, 2, 4, 6]	**6**
385	→	99	→	29	ADD 4	STO 4	[3, 3, 5, 7]	**7**
		↓						
455	→	70	→	8	ADD 3	STO 3	[3, 4, 6, 8]	**8**
		↓						
260	→	62	→	4	ADD 2	STO 2	[4, 5, 8, 11]	**9**
		↓↓			DBL		[8, 10, 16, 22]	**10**
99	→	29	→	3	ADD 4	STO 4	[11, 13, 21, 29]	**11**
		↓						
286	→	26	→	8	ADD 1		[12, 14, 22, 31]	**12**
			→	2	ADD 1	STO 1		
		↓↓			DBL		[24, 28, 44, 62]	**13**
70	→				ADD 3		[27, 32, 50, 70]	**14**
26	→	8			ADD 1		[39, 46, 72, 101]	**15**
		↓↓			DBL		[78, 92,144, 202]	**16**
62	→	4	→	1	ADD 2	STO 2	[82, 97,152, 213]	**17**
		↓						
29	→	3	→	1	ADD 4	STO 4	[93,110,173, 242]	**18**
		↓						
26	→	2			ADD 1		[105,124,195, 273]	**19**
		↓↓			DBL		[210,248, 390 ,546]	**20**
4	→				ADD 2		[292,345, 542 ,759]	**21**
3	→	1			ADD 4		[385,455, 715,1001]	**22**

Figure 12

12.5. The Generalized Algorithm of Fiat.

We consider as in the Algorithm of Yao the binary representation of 385, 455, 715 and 1001. We define the numbers $c_0, \ldots, c_{14}$ by

$$c_0=129,\ c_2=256,\ c_8=64,\ c_9=2,\ c_{11}=4,\ c_{12}=520,\ c_{14}=32.$$

The other numbers c_i are zero. We find the relations

$$\begin{aligned} 385 &= c_0+ c_2 \\ 455 &= c_0+ c_2+ c_8+ c_9+ c_{11} \\ 715 &= c_0+ c_8+ c_9+ c_{12} \\ 1001 &= c_0+ c_2+ c_8+ c_{12}+ c_{14} \end{aligned}$$

We get the following sequence:

{1, 2, 4, 8, 16, 32, 64, 128, 129, 256, 385, 449, 453, 455,
512, 520, 584, 586, 715, 969, 1001}.

We find the following reverse vector addition chain for [385, 455, 715, 1001]:

L_F[385, 455, 715, 1001] = {[0, 0, 1, 1], [0, 1, 0, 1], [1, 1, 0, 1], [1, 1, 1, 2], [1, 1, 2, 3], [1, 1, 1, 1], [2, 2, 3, 4], [3, 3, 5, 7], [6, 6, 10, 14], [6, 6, 11, 14], [6, 7, 11, 15], [12, 14, 22, 30], [12, 14, 22, 31], [24, 28, 44, 62], [48, 56, 88, 124], [48, 56, 89, 125], [96, 112, 178, 250], [96, 113, 178, 250], [192, 226, 356, 500], [192, 226, 357, 500], [192, 227, 357, 500], [384, 454, 714, 1000], [385, 455, 715, 1001]}

and l_F[385, 455, 715, 1001] = 23 and m_F[385, 455, 715, 1001] = 7.

12.6. The Window Algorithms of Yao and Straus.

We apply the Generalized Window Algorithm of Straus on the same example. We choose k = 1. We write the elements of the argument binary. We have

385	=		1	1	0	0	0	0	0	1
455	=		1	1	1	0	0	0	1	1	1
715	=	1	0	1	1	0	0	1	0	1	1
1001	=	1	1	1	1	1	0	1	0	0	1

For the set $\{\mathbf{w}_j\}$ which is in V, we get:

V {[0, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 1], [0, 1, 1, 1], [0, 1, 1, 0], [0, 1, 0, 1]}.

We have the following vector addition chain:

$L_{St(1)}$[385, 455, 715, 1001] = {[0, 0, 1, 1], [0, 1, 1, 0], [0, 1, 1, 1], [1, 1, 1, 1], [0, 1, 0, 1], [1, 1, 0, 1], [0, 0, 2, 2], [1, 1, 2, 3], [2, 2, 4, 6], [3, 3, 5, 7], [6, 6, 10, 14], [6, 7, 11, 15], [12, 14, 22, 30], [12, 14, 22, 31], [24, 28, 44, 62], [48, 56, 88, 124], [48, 56, 89, 125], [96, 112, 178, 250], [96, 113, 178, 250], [192, 226, 356, 500], [192, 226, 357, 500], [192, 227, 357, 500], [384, 454, 714, 1000], [385, 455, 715, 1001]}

and $l_{St(1)}$[385, 455, 715, 1001] = 23.

For Yao's Algorithm, we choose k = 2. We write the elements of the argument binary. We have

```
385 =   1  10  00  00  01
455 =   1  11  00  01  11
715 =  10  11  00  10  11
1001 = 11  11  10  10  01
```

For the vectors $\mathbf{w}_j$ we find:

$\mathbf{w}_0$=[1, 3, 3, 1], $\mathbf{w}_1$=[0, 1, 2, 2], $\mathbf{w}_2$=[0, 0, 0, 2], $\mathbf{w}_3$=[2, 3, 3, 3] and $\mathbf{w}_4$=[1, 1, 2, 3].

We have the following vector addition chain:

$L_{Y(2)}$[385, 455, 715, 1001] = {[2, 0, 0, 0], [0, 2, 0, 0], [0, 3, 0, 0], [0, 0, 2, 0], [0, 0, 3, 0], [0, 0, 0, 2], [0, 0, 0, 3], [1, 3, 0, 0], [1, 3, 3, 0], [1, 3, 3, 1], [0, 1, 2, 0], [0, 1, 2, 2], [2, 3, 0, 0], [2, 3, 3, 0], [2, 3, 3, 3], [1, 1, 0, 0], [1, 1, 2, 0], [1, 1, 2, 3], [2, 2, 4, 6], [4, 4, 8, 12], [6, 7, 11, 15], [12, 14, 22, 30], [24, 28, 44, 60], [24, 28, 44, 62], [48, 56, 88, 124], [96, 112, 176, 248], [96, 113, 178, 250], [192, 226, 356, 500], [384, 452, 712, 1000], [385, 455, 715, 1001]}

and $l_{Y(2)}$[385, 455, 715, 1001] = 30.

12.7. The Batch-RSA Algorithm.

We apply this algorithm on the vector [385, 455, 715, 1001] as mentioned above. We get

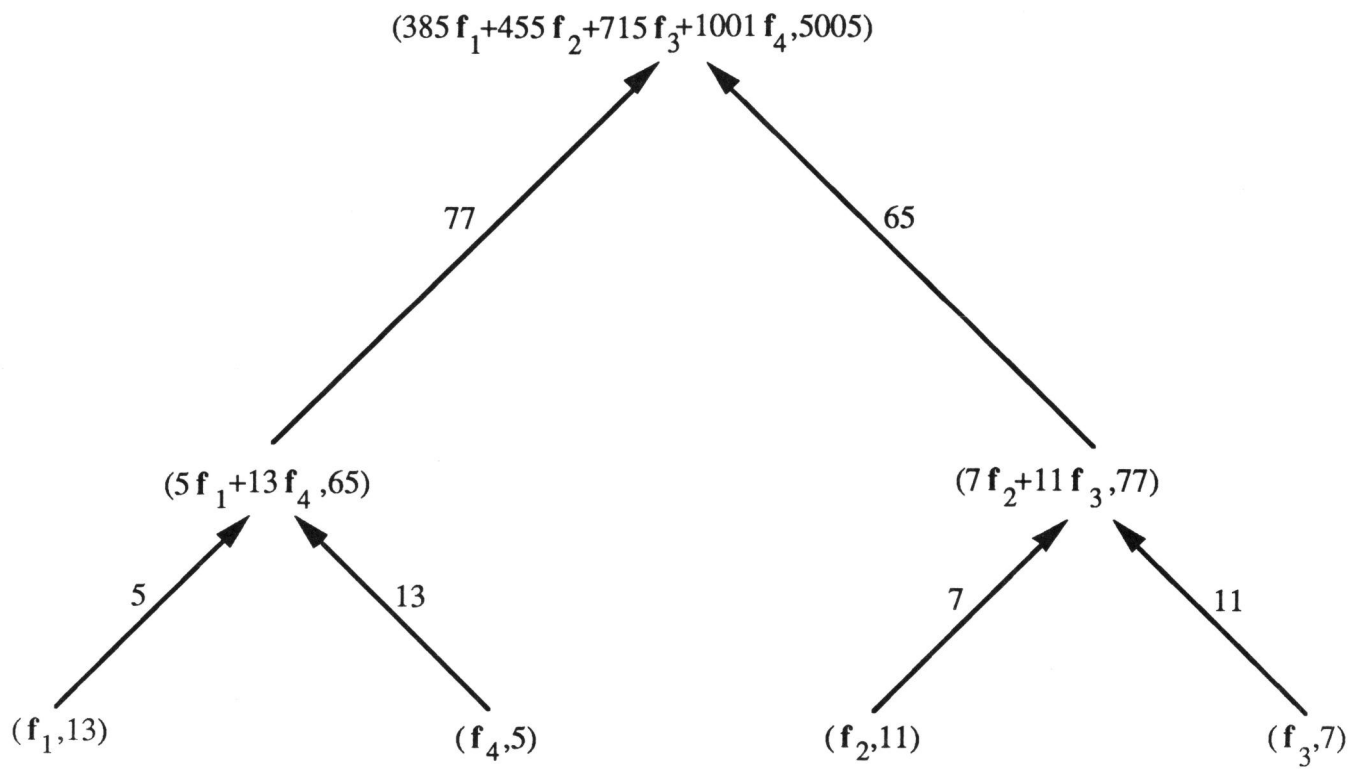

Figure 11.

Using Algorithm I we make vector addition chains for $L_I[5, 13]$, $L_I[7, 11]$ and $L_I[65, 77]$. We get $l_I[5, 13] = 6$, $l_I[7, 11] = 6$ and $l_I[65, 77] = 11$. Hence $l_B[385, 455, 715, 1001] = 23$. For the sake of completeness we give also

$L_{B,I}[385, 455, 715, 1001] = \{[0, 0, 0, 2], [1, 0, 0, 2], [1, 0, 0, 3], [2, 0, 0, 5], [4, 0, 0, 10], [5, 0, 0, 13], [0, 1, 1, 0], [0, 1, 2, 0], [0, 2, 3, 0], [0, 4, 6, 0], [0, 6, 9, 0], [0, 7, 11, 0], [5, 7, 11, 13], [10, 14, 22, 26], [20, 28, 44, 52], [25, 35, 55, 65], [30, 35, 55, 78], [60, 70, 110, 156], [65, 77, 121, 169], [130, 154, 242, 338], [260, 308, 484, 676], [325, 385, 605, 845], [385, 455, 715, 1001]\}$.

13. Appendix 2: List of all Cases in the proof of Theorem 14.

In this section we give a list which is a part of the proof of Theorem 14.

Table XI

cont. fraction	impr. cont. fraction	k	$\lvert\sigma_1\rvert$	fails for $r =$
$[0, r, \ldots]$		$l(r) + 1$	r	1, 2, 3
$[0, 1, r, \ldots]$		$l(r) + 2$	$r + 1$	1, 2, 3, 5, 7,11
$[0, 2, r, \ldots]$		$l(r) + 3$	$2r + 1$	1
$[0, 3, r, \ldots]$		$l(r) + 4$	$3r + 1$	1
$[0, 1, 1, r, \ldots]$		$l(r) + 3$	$2r + 1$	1
$[0, 1, 2, r, \ldots]$		$l(r) + 4$	$3r + 1$	1
$[0, 1, 3, r, \ldots]$		$l(r) + 5$	$4r + 1$	1, 2, 3
$[0, 1, 5, r, \ldots]$		$l(r) + 6$	$6r + 1$	1, 2, 3
$[0, 1, 7, r, \ldots]$		$l(r) + 7$	$8r + 1$	1, 2, 3, 5, 7
$[0, 1, 11, r, \ldots]$		$l(r) + 8$	$12r + 1$	1, 2, 3, 7
$[0, 2, 1, r, \ldots]$		$l(r) + 4$	$3r + 2$	
$[0, 3, 1, r, \ldots]$		$l(r) + 5$	$4r + 3$	
$[0, 1, 1, 1, r, \ldots]$		$l(r) + 4$	$3r + 2$	
$[0, 1, 2, 1, r, \ldots]$		$l(r) + 5$	$4r + 3$	
$[0, 1, 3, 1, r, \ldots]$		$l(r) + 6$	$5r + 4$	1, 3, 7, 11
$[0, 1, 3, 2, r, \ldots]$		$l(r) + 7$	$9r + 4$	1
$[0, 1, 3, 3, r, \ldots]$		$l(r) + 8$	$13r + 4$	1
$[0, 1, 5, 1, r, \ldots]$		$l(r) + 7$	$7r + 6$	1, 3, 5, 7, 11
$[0, 1, 5, 2, r, \ldots]$		$l(r) + 8$	$13r + 6$	1
$[0, 1, 5, 3, r, \ldots]$		$l(r) + 9$	$19r + 6$	1
$[0, 1, 7, 1, r, \ldots]$		$l(r) + 8$	$9r + 8$	1,2,3,5,7,11,13,19
$[0, 1, 7, 2, r, \ldots]$	$[0, 1, 8, -1, 1, r, \ldots]$	$l(r) + 8$	$17r + 8$	
$[0, 1, 7, 3, r, \ldots]$	$[0, 1, 8, -1, 2, r, \ldots]$	$l(r) + 9$	$25r + 8$	
$[0, 1, 7, 5, r, \ldots]$	$[0, 1, 8, -1, 4, r, \ldots]$	$l(r) + 10$	$41r + 8$	
$[0, 1, 7, 7, r, \ldots]$	$[0, 1, 8, -1, 6, r, \ldots]$	$l(r) + 11$	$57r + 8$	
$[0, 1, 11, 1, r, \ldots]$		$l(r) + 8$	$13r + 12$	1,2,3,5,7,11,13,19
$[0, 1, 11, 2, r, \ldots]$	$[0, 1, 12, -1, 1, r, \ldots]$	$l(r) + 9$	$25r + 12$	
$[0, 1, 11, 3, r, \ldots]$	$[0, 1, 12, -1, 2, r, \ldots]$	$l(r) + 10$	$37r + 12$	
$[0, 1, 11, 7, r, \ldots]$	$[0, 1, 12, -1, 6, r, \ldots]$	$l(r) + 12$	$85r + 12$	
$[0, 1, 3, 1, 1, r, \ldots]$		$l(r) + 7$	$9r + 5$	
$[0, 1, 3, 1, 3, r, \ldots]$	$[0, 1, 4, -4, r, \ldots]$	$l(r) + 8$	$19r + 5$	
$[0, 1, 3, 1, 7, r, \ldots]$	$[0, 1, 4, -8, r, \ldots]$	$l(r) + 9$	$39r + 5$	
$[0, 1, 3, 1, 11, r, \ldots]$	$[0, 1, 4, -12, r, \ldots]$	$l(r) + 10$	$59r + 5$	

Table XI (continued)

cont. fraction	impr. cont. fraction	k	$\lvert\sigma_1\rvert$
[0, 1, 3, 2, 1, r, ...]		$l(r) + 8$	$13r + 9$
[0, 1, 3, 3, 1, r, ...]		$l(r) + 9$	$17r + 13$
[0, 1, 5, 1, 1, r, ...]		$l(r) + 8$	$13r + 7$
[0, 1, 5, 1, 3, r, ...]	[0, 1, 6, –4, r, ...]	$l(r) + 9$	$27r + 7$
[0, 1, 5, 1, 5, r, ...]	[0, 1, 6, –6, r, ...]	$l(r) + 10$	$41r + 7$
[0, 1, 5, 1, 7, r, ...]	[0, 1, 6, –8, r, ...]	$l(r) + 10$	$55r + 7$
[0, 1, 5, 1, 11, r, ...]	[0, 1, 6, –12, r, ...]	$l(r) + 11$	$83r + 7$
[0, 1, 5, 2, 1, r, ...]		$l(r) + 9$	$19r + 13$
[0, 1, 5, 3, 1, r, ...]		$l(r) + 10$	$25r + 19$
[0, 1, 7, 1, 1, r, ...]	[0, 1, 8, –2, r, ...]	$l(r) + 7$	$17r + 9$
[0, 1, 7, 1, 2, r, ...]	[0, 1, 8, –3, r, ...]	$l(r) + 8$	$26r + 9$
[0, 1, 7, 1, 3, r, ...]	[0, 1, 8, –4, r, ...]	$l(r) + 8$	$35r + 9$
[0, 1, 7, 1, 5, r, ...]	[0, 1, 8, –6, r, ...]	$l(r) + 9$	$53r + 9$
[0, 1, 7, 1, 7, r, ...]	[0, 1, 8, –8, r, ...]	$l(r) + 9$	$71r + 9$
[0, 1, 7, 1, 11, r, ...]	[0, 1, 8, –12, r, ...]	$l(r) + 10$	$107r + 9$
[0, 1, 7, 1, 13, r, ...]	[0, 1, 8, –14, r, ...]	$l(r) + 11$	$125r + 9$
[0, 1, 7, 1, 19, r, ...]	[0, 1, 8, –20, r, ...]	$l(r) + 11$	$179r + 9$
[0, 1, 11, 1, 1, r, ...]	[0, 1, 12, –2, r, ...]	$l(r) + 8$	$25r + 13$
[0, 1, 11, 1, 2, r, ...]	[0, 1, 12, –3, r, ...]	$l(r) + 9$	$38r + 13$
[0, 1, 11, 1, 3, r, ...]	[0, 1, 12, –4, r, ...]	$l(r) + 9$	$51r + 13$
[0, 1, 11, 1, 5, r, ...]	[0, 1, 12, –6, r, ...]	$l(r) + 10$	$77r + 13$
[0, 1, 11, 1, 7, r, ...]	[0, 1, 12, –8, r, ...]	$l(r) + 10$	$103r + 13$
[0, 1, 11, 1, 11, r, ...]	[0, 1, 12, –12, r, ...]	$l(r) + 11$	$155r + 13$
[0, 1, 11, 1, 13, r, ...]	[0, 1, 12, –14, r, ...]	$l(r) + 12$	$181r + 13$
[0, 1, 11, 1, 19, r, ...]	[0, 1, 12, –20, r, ...]	$l(r) + 12$	$259r + 13$