Some Examples of Average-case Analysis by the Incompressibility Method

Tao Jiang, Ming Li, and Paul Vitányi

Summary. The incompressibility method is an elementary yet powerful proof technique. It has been used successfully in many areas, including average-case analysis of algorithms [14]. In this expository paper, we include several new simple average-case analyses to further demonstrate the utility and elegance of the method.

1 Introduction

The incompressibility of individual random objects yields a simple but powerful proof technique, namely *the incompressibility method.* This method is a versatile tool that can be used to prove lower bounds on computational problems, to obtain combinatorial properties of concrete objects, and to analyze the average-case complexity of algorithms. Since the early 1980's, the incompressibility method has been successfully used to solve many well-known questions that had been open for a long time and to supply new simplified proofs for known results. A comprehensive survey can be found in [14].

In this short expository paper, we use four simple examples of diverse topics to further demonstrate how easy the incompressibility method can be used to obtain (upper and lower) bounds which are useful in the domain of average-case analysis. The topics covered in this paper are well-known ones such as sorting, matrix multiplication, longest common subsequences, and majority finding. The proofs that we choose to include are not difficult ones and all the results are known before. However, our new proofs are much simpler than the old ones and are easy to understand. More such new proofs are contained in [5,13].

To make the paper self-contained, we give an overview of Kolmogorov complexity and the incompressibility method in the next section. We then consider the four diverse problems, namely sorting, boolean matrix multiplication, longest common subsequences, and majority finding, in four separate sections.

2 Kolmogorov Complexity and the Incompressibility Method

We use the following notation. Let x be a finite binary string. Then $l(x)$ denotes the *length* (number of bits) of x. In particular, $l(\epsilon) = 0$ where ϵ denotes the *empty word.*

We can map $\{0,1\}^*$ one-to-one onto the natural numbers by associating each string with its index in the length-increasing lexicographical ordering

$$(\epsilon, 0), (0,1), (1,2), (00,3), (01,4), (10,5), (11,6), \ldots. \tag{1}$$

This way we have a binary representation for the natural numbers that is different from the standard binary representation. It is convenient not to distinguish between the first and second element of the same pair, and call them "string" or "number" arbitrarily. As an example, we have $l(7) = 00$. Let $x, y, \in \mathcal{N}$, where $\mathcal{N}$ denotes the natural numbers. Let $T_0, T_1, \ldots$ be a standard enumeration of all Turing machines. Let $\langle \cdot, \cdot \rangle$ be a standard one-one mapping from $\mathcal{N} \times \mathcal{N}$ to $\mathcal{N}$, for technical reasons chosen so that $l(\langle x, y \rangle) = l(y) + O(l(x))$.

Informally, the Kolmogorov complexity, [15], of x is the length of the *shortest* effective description of x. That is, the *Kolmogorov complexity* $C(x)$ of a finite string x is simply the length of the shortest program, say in FORTRAN (or in Turing machine codes) encoded in binary, which prints x without any input. A similar definition holds conditionally, in the sense that $C(x|y)$ is the length of the shortest binary program which computes x on input y. Kolmogorov complexity is absolute in the sense of being independent of the programming language, up to a fixed additional constant term which depends on the programming language but not on x. We now fix one canonical programming language once and for all as reference and thereby $C()$. For the theory and applications, as well as history, see [14]. A formal definition is as follows:

Definition 1. Let U be an appropriate universal Turing machine such that

$$U(\langle \langle i, p \rangle, y \rangle) = T_i(\langle p, y \rangle)$$

for all i and $\langle p, y \rangle$. The *conditional Kolmogorov complexity* of x given y is

$$C(x|y) = \min_{p \in \{0,1\}^*} \{l(p) : U(\langle p, y \rangle) = x\}.$$

The unconditional Kolmogorov complexity of x is defined as $C(x) := C(x|\epsilon)$.

By a simple counting argument one can show that whereas some strings can be enormously compressed, the majority of strings can hardly be compressed at all. For each n there are 2^n binary strings of length n, but only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ possible shorter descriptions. Therefore, there is at least one binary string x of length n such that $C(x) \geq n$. We call such strings *incompressible.*

Definition 2. For each constant c we say a string x is *c-incompressible* if $C(x) \geq l(x) - c$.

Strings that are incompressible (say, c-incompressible with small c) are patternless, since a pattern could be used to reduce the description length. Intuitively, we think of such patternless sequences as being random, and we use "random sequence" synonymously with "incompressible sequence." It is possible to give a rigorous formalization of the intuitive notion of a random sequence as a sequence that passes all effective tests for randomness, see for example [14].

How many strings of length n are c-incompressible? By the same counting argument we find that the number of strings of length n that are c-incompressible is at least $2^n - 2^{n-c} + 1$. Hence there is at least one 0-incompressible string of length n, at least one-half of all strings of length n are 1-incompressible, at least three-fourths of all strings of length n are 2-incompressible, ..., and at least the $(1 - 1/2^c)$th part of all 2^n strings of length n are c-incompressible. This means that for each constant $c \geq 1$ the majority of all strings of length n (with $n > c$) is c-incompressible. We generalize this to the following simple but extremely useful *Incompressibility Lemma.*

Lemma 1. *Let c be a positive integer. For each fixed y, every set A of cardinality m has at least $m(1-2^{-c})+1$ elements x with $C(x|A,y) \geq \lfloor \log m \rfloor - c$.*

Proof. By simple counting. □

Definition 3. A *prefix set*, or prefix-free code, or prefix code, is a set of strings such that no member is a prefix of any other member. A prefix set which is the domain of a partial recursive function (set of halting programs for a Turing machine) is a special type of prefix code called a *self-delimiting* code because there is an effective procedure which reading left-to-right determines where a code word ends without reading past the last symbol. A one-to-one function with a range that is a self-delimiting code will also be called a self-delimiting code.

A simple self-delimiting code we use throughout is obtained by reserving one symbol, say 0, as a stop sign and encoding a natural number x as 1^x0. We can prefix an object with its length and iterate this idea to obtain ever shorter codes:

$$E_i(x) = \begin{cases} 1^x0 & \text{for } i = 0, \\ E_{i-1}(l(x))x & \text{for } i > 0. \end{cases} \tag{2}$$

Thus, $E_1(x) = 1^{l(x)}0x$ and has length $l(E_1(x)) = 2l(x)+1$; $E_2(x) = E_1(l(x))x = 1^{l(l(x))}0x$ and has length $l(E_2(x)) = l(x)+2l(l(x))+1$. We have for example

$$l(E_3(x)) \leq l(x) + \log l(x) + 2\log\log l(x) + 1.$$

Define the pairing function

$$\langle x, y \rangle = E_2(x)y \tag{3}$$

with inverses $\langle\cdot\rangle_1, \langle\cdot\rangle_2$. This can be iterated to $\langle\langle\cdot,\cdot\rangle,\cdot\rangle$.

The Incompressibility Method. In a typical proof using the incompressibility method, one first chooses an individually random object from the class under discussion. This object is effectively incompressible. The argument invariably says that if a desired property does not hold, then the object can be compressed. This yields the required contradiction. Then, since most objects are random, the desired property usually holds on the average.

3 Lower Bound for Sorting

We begin this paper with a very simple incompressibility proof for a well-known lower bound on comparison based sorting.

Theorem 1. *Any comparison based sorting algorithm requires* $\log n!$ *comparisons to sort an array of* n *elements.*

Proof. Let A be any comparison based sorting algorithm. Consider permutation I of $\{1, \ldots, n\}$ such that

$$C(I|A, P) \geq \log n!$$

where P is a fixed program to be defined. Suppose A sorts I in m comparisons. We can describe I by recording the binary outcomes of the m comparisons, which requires a total of m bits. Let P be such a program converting m to I (given A). Thus,

$$m \geq C(I|A, P) \geq \log n!$$

Hence, $m \geq \log n!$. □

The above proof in fact also implies a lower bound of $\log n! - 2\log n$ on the average number of comparisons required for sorting.

Corollary 1. *Any comparison based sorting algorithm requires* $\log n! - 2\log n$ *comparisons to sort an array of* n *elements, on the average.*

Proof. In the above proof, let I have Kolmogorov complexity:

$$C(I|A, P) \geq \log n! - \log n$$

Then we obtain that the (arbitrary) algorithm A requires

$$m \geq \log n! - \log n$$

comparisons on the permutation I. It follows from the Incompressibility Lemma that on the average, A requires at least

$$\frac{n-1}{n} \cdot (\log n! - \log n) + \frac{1}{n} \cdot (n-1) \geq \log n! - 2\log n$$

comparisons. □

4 Average Time for Boolean Matrix Multiplication

We begin with a simple (almost trivial) illustration of average-case analysis using the incompressibility method. Consider the well-known problem of multiplying two $n \times n$ boolean matrices $A = (a_{i,j})$ and $B = (b_{i,j})$. Efficient algorithms for this problem have always been a very popular topic in the theoretical computer science literature due to the wide range of applications of boolean matrix multiplication. The best worst-case time complexity obtained so far is $O(n^{2.376})$ due to Coppersmith and Winograd [7]. In 1973, O'Neil and O'Neil devised a simple algorithm described below which runs in $O(n^3)$ time in the worst case but achieves an average time complexity of $O(n^2)$ [16].

Algorithm QuickMultiply(A, B)

1. Let $C = (c_{i,j})$ denote the result of multiplying A and B.
2. For $i := 1$ to n do
3. Let $j_1 < \cdots < j_m$ be the indices such that $a_{i,j_k} = 1$, $1 \leq k \leq m$.
4. For $j := 1$ to n do
5. Search the list $b_{j_1,j}, \ldots, b_{j_m,j}$ sequentially for a bit 1.
6. Set $c_{i,j} = 1$ if a bit 1 is found, or $c_{i,j} = 0$ otherwise.

An analysis of the average-case time complexity of QuickMultiply is given in [16] using simple probabilitistic arguments. Here we give an analysis using the incompressibility method, to illustrate some basic ideas.

Theorem 2. *Suppose that the elements of A and B are drawn uniformly and independently. Algorithm QuickMultiply runs in $O(n^2)$ time on the average.*

Proof. Let n be a sufficiently large integer. Observe that the average time of QuickMultiply is trivially bounded between $O(n^2)$ and $O(n^3)$. By the Incompressibility Lemma, out of the 2^{2n^2} pairs of $n \times n$ boolean matrices, at least $(n-1)2^{2n^2}/n$ of them are $\log n$-incompressible. Hence, it suffices to consider $\log n$-incompressible boolean matrices.

Take a $\log n$-incompressible binary string x of length $2n^2$, and form two $n \times n$ boolean matrices A and B straightforwardly so that the first half of x corresponds to the row-major listing of the elements of A and the second half of x corresponds to the row-major listing of the elements of B. We show that QuickMultiply spends $O(n^2)$ time on A and B.

Consider an arbitrary i, where $1 \leq i \leq n$. It suffices to show that the n sequential searches done in Steps 4 – 6 of QuickMultiply take a total of $O(n)$ time. By the statistical results on various blocks in incompressible strings given in Section 2.6 of [14], we know that at least $n/2 - O(\sqrt{n \log n})$ of these searches find a 1 in the first step, at least $n/4 - O(\sqrt{n \log n})$ searches find a 1 in two steps, at least $n/8 - O(\sqrt{n \log n})$ searches find a 1 in three steps, and so on. Moreover, we claim that none of these searches take more than $4 \log n$ steps. To see this, suppose that for some j, $1 \leq j \leq n$, $b_{j_1,j} = \cdots = b_{j_{4 \log n},j} = 0$. Then we can encode x by listing the following items in a self-delimiting manner:

1. A description of the above discussion.
2. The value of i.
3. The value of j.
4. All bits of x except the bits $b_{j_1,j},\ldots,b_{j_{4\log n},j}$.

This encoding takes at most

$$O(1)+2\log n+2n^2-4\log n+O(\log\log n)<2n^2-\log n$$

bits for sufficiently large n, which contradicts the assumption that x is $\log n$-incompressible.

Hence, the n searches take at most a total of

$$\begin{aligned}&(\sum_{k=1}^{\log n}(n/2^k-O(\sqrt{n\log n}))\cdot k)+(\log n)\cdot O(\sqrt{n\log n})\cdot(4\log n)\\&<(\sum_{k=1}^{\log n}kn/2^k+O(\log^2 n\sqrt{n\log n})\\&=O(n)+O(\log^2 n\sqrt{n\log n})\\&=O(n)\end{aligned}$$

steps. This completes the proof. □

5 Expected Length of a Longest Common Subsequence

For two sequences (*i.e.* strings) $s=s_1\ldots s_m$ and $t=t_1\ldots t_n$, we say that s is a *subsequence* of t if for some $i_1<\ldots<i_m$, $s_j=t_{i_j}$. A *longest common subsequence* (LCS) of sequences s and t is a longest possible sequence u that is a subsequence of both s and t. For simplicity, we will only consider binary sequences over the alphabet $\Sigma=\{0,1\}$.

Let n be an arbitrary positive integer and consider two random strings s and t that are drawn independently from the uniformly distributed space of all binary string of length n. We are interested in the expected length of an LCS of s and t. Tight bounds on the expected LCS length for two random sequences is a well-known open question in statistics [17,19]. After a series of papers, the best result to date is that the length is between $0.762n$ and $0.838n$ [6,8–10]. The proofs are based on intricate probablistic and counting arguments. Here we give simple proofs of some nontrivial upper and lower bounds using the incompressibility method.

Theorem 3. *The expected length of an LCS of two random sequences of length n is at most $0.867n+o(n)$.*

Proof. Let n be a sufficiently large integer. Observe that the expected length of an LCS of two random sequences of length n is trivially bounded between

$n/2$ and n. By the Incompressibility Lemma, out of the 2^{2n} pairs of binary sequences of length n, at least $(n-1)2^{2n}/n$ of them are $\log n$-incompressible. Hence, it suffices to consider $\log n$-incompressible sequences.

Take a $\log n$-incompressible string x of length $2n$, and let s and t be the first and second halves of x respectively. Suppose that string u is an LCS of s and t. In order to relate the Kolmogorov complexity of s and t to the length of u, we re-encode the strings s and t using the string u as follows. (The idea was first introduced in [12].)

We first describe how to re-encode s. Let the LCS $u = u_1u_2\cdots u_m$, where $m = l(u)$. We align the bits of u with the corresponding bits of s greedily from left to right, and rewrite s as follows:

$$s = \alpha_1 u_1 \alpha_2 u_2 \cdots \alpha_m u_m s'.$$

Here α_1 is the longest prefix of s containing no u_1, α_2 is the longest substring of s following the bit u_1 containing no u_2, and so on, and s' is the remaining part of s after the bit u_m. Thus α_i does not contain bit u_i, for $i = 1, \ldots, m$. In other words, each α_i is a unary string consisting of the bit complementary to u_i. We re-encode s as string:

$$s(u) = 0^{l(\alpha_1)}10^{l(\alpha_2)}1\cdots 0^{l(\alpha_m)}1s'.$$

Clearly, given u we can uniquely decode the encoding $s(u)$ to obtain s.

Similarly, the string t can be rewritten as

$$t = \beta_1 u_1 \beta_2 u_2 \cdots \beta_m u_m t',$$

where each β_i is a unary string consisting of the bit complementary to u_i, and we re-encode t as string:

$$t(u) = 0^{l(\beta_1)}10^{l(\beta_2)}1\cdots 0^{l(\beta_m)}1t'.$$

Hence, the string x can be described by the following information in the self-delimiting form:

1. A description of the above discussion.
2. The LCS u.
3. The new encodings $s(u)$ and $t(u)$ of s and t.

Now we estimate the Kolmogorov complexity of the above description of x. Items 1 and 2 take $m + O(1)$ bits. Since $s(u)$ contains at least m 1's, it is easy to see by simple counting and Stirling approximation (see *e.g.* [14]) that

$$\begin{aligned} C(s(u)) &\le \log \sum_{i=m}^{n} \binom{n}{i} + O(1) \\ &\le \log \left[\frac{n}{2}\binom{n}{m}\right] + O(1) \\ &\le \log n + \log \binom{n}{m} + O(1) \\ &\le 2\log n + n\log n - m\log m - (n-m)\log(n-m) + O(1) \end{aligned}$$

The second step in the above derivation follows from the trivial fact that $m \geq n/2$. Similarly, we have

$$C(t(u)) \leq 2\log n + n\log n - m\log m - (n-m)\log(n-m) + O(1)$$

Hence, the above description requires a total size of

$$O(\log n) + m + 2n\log n - 2m\log m - 2(n-m)\log(n-m).$$

Let $p = n/m$. Since $C(x) \geq 2n - \log n$, we have

$$\begin{aligned} 2n - \log n &\leq O(\log n) + m + 2n\log n - 2m\log m - 2(n-m)\log(n-m) \\ &= O(\log n) + pn - 2np\log p - 2n(1-p)\log(1-p) \end{aligned}$$

Dividing both sides of the inequality by n, we obtain

$$2 \leq o(1) + p - 2p\log p - 2(1-p)\log(1-p)$$

Solving the inequality numerically we get $p \leq 0.867 - o(1)$. □

Next we prove that the expected length of an LCS of two random sequences of length n is at least $0.66666n - O(\sqrt{n\log n})$. To prove the lower bound, we will need the following greedy algorithm for computing common subsequences (not necessarily the longest ones).

Algorithm Zero-Major($s = s_1 \cdots s_n, t = t_1 \cdots t_n$)

1. Let $u := \epsilon$ be the empty string.
2. Set $i := 1$ and $j := 1$;
3. **Repeat** steps 4–6 until $i > n$ or $j > n$:
4. **If** $s_i = t_j$ **then begin** append bit s_i to string u; $i := i+1$; $j := j+1$ **end**
5. **Elseif** $s_i = 0$ **then** $j := j+1$
6. **Else** $i := i+1$
7. Return string u.

Theorem 4. *The expected length of an LCS of two random sequences of length n is at least* $0.66666n - O(\sqrt{n\log n})$.

Proof. Again, let n be a sufficiently large integer, and take a $\log n$-incompressible string x of length $2n$. Let s and t be the first and second halves of x respectively. It suffices to show that the above algorithm Zero-Major produces a common subsequence u of length at least $0.66666n - O(\sqrt{n\log n})$ for strings s and t.

The idea is to encode s and t (and thus x) using information from the computation of Zero-Major on strings s and t. We consider the comparisons made by Zero-Major in the order that they were made, and create a pair of strings y and z as follows. For each comparison (s_i, t_j) of two complementary

bits, we simply append a 1 to y. For each comparison (s_i, t_j) of two identical bits, append a bit 0 to the string y. Furthermore, if this comparison of identical bits is preceded by a comparison $(s_{i'}, t_{j'})$ of two complementary bits, we then append a bit 0 to the string z if $i' = i - 1$ and a bit 1 if $j' = j - 1$. When one string (s or t) is exhausted by the comparisons, we append the remaining part (call this w) of the other string to z.

As an example of the encoding, consider strings $s = 1001101$ and $t = 0110100$. Algorithm Zero-Major produces a common subsequence 0010. The following figure depicts the comparisons made by Zero-Major, where a "*" indicates a mismatch and a "|" indicates a match.

```
s =            10  01101
comparisons    *|**||*|*
t =             01101 0 0
```

Following the above encoding scheme, we obtain $y = 101100101$ and $z = 01100$.

It is easy to see that the strings y and z uniquely encode s and t and, moreover, $l(y)+l(z) = 2n$. Since $C(yz) \geq C(x)-2\log n \geq 2n-3\log n-O(1)$, and $C(z) \leq l(z) + O(1)$, we have

$$C(y) \geq l(y) - 3\log n - O(1)$$

Similarly, we can obtain

$$C(z) \geq l(z) - 3\log n - O(1)$$

and

$$C(w) \geq l(w) - 3\log n - O(1)$$

where w is the string appended to z at the end of the above encoding.

Now let us estimate the length of the common subsequence u produced by Zero-Major on strings s and t. Let $\#zeroes(s)$ and $\#zeroes(t)$ be the number of 0's contained in s and t respectively. Clearly, u contains $\min\{\#zeroes(s), \#zeroes(t)\}$ 0's. From [14] (page 159), since both s and t are $\log n$-incompressible, we know

$$n/2 - O(\sqrt{n\log n}) \leq \#zeroes(s) \leq n/2 + O(\sqrt{n\log n})$$

$$n/2 - O(\sqrt{n\log n}) \leq \#zeroes(t) \leq n/2 + O(\sqrt{n\log n})$$

Hence, the string w has at most $O(\sqrt{n\log n})$ 0's. Combining with the fact that $C(w) \geq l(w) - 3\log n - O(1)$ and the above mentioned result in [14], we claim

$$l(w) \leq O(\sqrt{n\log n}).$$

Since $l(z) - l(w) = l(u)$, we have a lower bound on $l(u)$:

$$l(u) \geq l(z) - O(\sqrt{n\log n}).$$

On the other hand, since every bit 0 in the string y corresponds to a unique bit in the common subsequence u, we have $l(u) \geq \#zeroes(y)$. Since $C(y) \geq l(y) - 2\log n - O(1)$,

$$l(u) \geq \#zeroes(y) \geq l(y)/2 - O(\sqrt{n \log n}).$$

Hence,

$$3l(u) \geq l(y) + l(z) - O(\sqrt{n \log n}) \geq 2n - O(\sqrt{n \log n}).$$

That is,

$$l(u) \geq 2n/3 - O(\sqrt{n \log n}) \approx 0.66666n - O(\sqrt{n \log n}).$$

□

Our above upper and lower bounds are not as tight as the ones in [6,8–10]. Recently, Baeza-Yates and Navarro improved our analysis and obtained a slightly better upper of 0.860 [4]. It will be interesting to know if stronger bounds can be obtained using the incompressibility method by more clever encoding schemes.

6 Average Complexity of Finding the Majority

Let $x = x_1 \cdots x_n$ be a binary string. The *majority bit* (or simply, the *majority*) of x is the bit (0 or 1) that appears more than $\lfloor n/2 \rfloor$ times in x. The majority problem is that, given a binary string x, determine the majority of x. When x has no majority, we must report so.

The time complexity for finding the majority has been well studied in the literature (see, *e.g.* [1–3,11,18]). It is known that, in the worst case, $n - \nu(n)$ bit comparisons are necessary and sufficient [2,18], where $\nu(n)$ is the number of occurrences of bit 1 in the binary representation of number n. Recently, Alonso, Reingold and Schott [3] studied the average complexity of finding the majority assuming the uniform probability distribution model. Using quite sophisticated arguments based on decision trees, they showed that on the average finding the majority requires at most $2n/3 - \sqrt{8n/9\pi} + O(\log n)$ comparisons and at least $2n/3 - \sqrt{8n/9\pi} + \Theta(1)$ comparisons.

In this section, we consider the average complexity of finding the majority and prove an upper bound tight up to the first major term, using simple incompressibility arguments.

The following standard tournament algorithm is needed.

Algorithm Tournament($x = x_1 \cdots x_n$)

1. If $n = 1$ then return x_1 as the majority.
2. Elseif $n = 2$ then

3. If $x_1 = x_2$ then return x_1 as the majority.
4. Else return "no majority".
5. Elseif $n = 3$ then
6. If $x_1 = x_2$ then return x_1 as the majority.
7. Else return x_3 as the majority.
8. Let $y = \epsilon$.
9. For $i := 1$ to $\lfloor n/2 \rfloor$ do
10. If $x_{2i-1} = x_{2i}$ then append the bit x_{2i} to y.
11. If n is odd and $\lfloor n/2 \rfloor$ is even then append the bit x_n to y.
12. Call Tournament(y).

Theorem 5. *On the average, algorithm Tournament requires at most* $2n/3 + O(\sqrt{n})$ *comparisons.*

Proof. Let n be a sufficiently large number. Again, since algorithm Tournament makes at most n comparisons on any string of length n, by the Incompressibility Lemma, it suffices to consider running time of Tournament on δ-incompressible strings, where $\delta \leq \log n$. Consider an arbitrary $\delta \leq \log n$ and let $x = x_1 \cdots x_n$ be a fixed δ-incompressible binary string. For any integer $m \leq n$, let $\sigma(m)$ denote the maximum number of comparisons required by algorithm Tournament on any δ-incompressible string of length m.

We know from [14] that among the $\lfloor n/2 \rfloor$ bit pairs $(x_1, x_2), \ldots, (x_{2\lfloor n/2 \rfloor - 1}, x_{2\lfloor n/2 \rfloor})$ that are compared in step 10 of Tournament, there are at least $n/4 - O(\sqrt{n\delta})$ pairs consisting of complementary bits. Clearly, the new string y obtained at the end of step 11 should satisfy

$$C(y) \geq l(y) - \delta - O(1)$$

Hence, we have the following recurrence relation for $\sigma(m)$:

$$\sigma(m) \leq \lfloor m/2 \rfloor + \sigma(m/4 + O(\sqrt{m\delta}))$$

By straightforward expansion, we obtain that

$$\begin{aligned}
\sigma(n) &\leq \lfloor n/2 \rfloor + \sigma(n/4 + O(\sqrt{n\delta})) \\
&\leq n/2 + \sigma(n/4 + O(\sqrt{n\delta})) \\
&\leq n/2 + (n/8 + O(\sqrt{n\delta})/2) + \sigma(n/16 + O(\sqrt{n\delta})/4 + O(\sqrt{(n\delta)/4})) \\
&= n/2 + (n/8 + O(\sqrt{n\delta})/2) + \sigma(n/16 + (3/4) \cdot O(\sqrt{n\delta})) \\
&\leq \cdots \\
&\leq 2n/3 + O(\sqrt{n\delta})
\end{aligned}$$

Using the Incompressibility Lemma, we can calculate the average complexity of algorithm Tournament as:

$$\sum_{\delta=1}^{\log n} \frac{1}{2^\delta}(2n/3 + O(\sqrt{n\delta})) + \frac{1}{n}n = 2n/3 + O(\sqrt{n})$$

□

References

1. L. Alexanderson, L.F. Klosinski and L.C. Larson, *The William Lowell Putnam Mathematical Competition, Problems and Solutions: 1965-1984*, Mathematical Association of America, Washington, DC, 1985.
2. L. Alonso, E. Reingold and R. Schott, Determining the majority, *Information Processing Letters* 47, 1993, pp. 253-255.
3. L. Alonso, E. Reingold and R. Schott, The average-case complexity of determining the majority, *SIAM Journal on Computing* 26-1, 1997, pp. 1-14.
4. R. Baeza-Yates and G. Navarro, Bounding the expected length of longest common subsequences and forests, *Manuscript*, 1997.
5. H. Buhrman, T. Jiang, M. Li, and P.M.B. Vitányi, New applications of the incompressibility method: Part II, accepted to *Theoret. Comput. Sci.* (Special Issue for "Int'l Conf. Theoret. Comput. Sci.", Hong Kong, April, 1998).
6. V. Chvátal and D. Sankoff. Longest common subsequences of two random sequences. *J. Appl. Probab.* 12, 1975, 306-315.
7. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Proc. of 19th ACM Symp. on Theory of Computing*, 1987, pp. 1-6.
8. V. Dančík and M. Paterson, Upper bounds for the expected length of a longest common subsequence of two binary sequences, *Proc. 11th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 775, Springer, pp. 669-678, Caen, France, 1994.
9. J.G. Deken, Some limit results for longest common subsequences, *Discrete Mathematics* 26, 1979, pp. 17-31.
10. J.G. Deken, Probabilistic behavior of longest-common-subsequence length, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* (D. Sankoff and J. Kruskall, Eds.) , Addison-Wesley, Reading, MA., 1983, pp. 359-362.
11. D.H. Greene and D.E. Knuth, *Mathematics for the Analysis of Algorithms*, 3rd ed., Birkhäuser, Boston, MA, 1990.
12. T. Jiang and M. Li, On the approximation of shortest common supersequences and longest common subsequences, *SIAM Journal on Computing* 24-5, 1122-1139, 1995.
13. T. Jiang, M. Li, and P.M.B. Vitányi, New applications of the incompressibility method, accepted to *The Computer Journal*, 1998.
14. M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, New York, 2nd Edition, 1997.
15. A.N. Kolmogorov, Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1(1):1-7, 1965.
16. P. O'Neil and E. O'Neil. A fast expected time algorithm for boolean matrix multiplication and transitive closure. *Information and Control* 22, 1973, pp. 132-138.
17. M. Paterson and V. Dančík, Longest common subsequences, *Proc. 19th International Symposium on Mathematical Foundations of Computer Science*, LNCS 841, Springer, pp. 127-142, Kosice, Slovakia, 1994.
18. M.E. Saks and M. Werman, On computing majority by comparisons, *Combinatorica* 11, 1991, pp. 383-387.
19. D. Sankoff and J. Kruskall (Eds.) *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA., 1983.