Average-Case Analysis of Algorithms Using Kolmogorov Complexity*

Tao Jiang†
McMaster University

Ming Li‡
University of Waterloo

Paul Vitányi§
CWI and University of Amsterdam

Abstract

Analyzing the average-case complexity of algorithms is a very practical but very difficult problem in computer science. In the past few years, we have demonstrated that Kolmogorov complexity is an important tool for analyzing the average-case complexity of algorithms. We have developed the incompressibility method [7]. In this paper, we use several simple examples to further demonstrate the power and simplicity of such method. We prove bounds on the average-case number of stacks (queues) required for sorting sequential or parallel Queueusort or Stacksort.

1 Introduction

We obtain some results on the average number of stacks or queues (sequential or parallel) required for sorting under the uniform distribution on input permutations. These problems have been studied before by Knuth [6] and Tarjan [10] for the worst case. This is a further application of the incompressibility method on sorting algorithms following [2].

The goal of the paper is not to obtain these bounds, but to demonstrate a new methodology of analyzing the average-case complexity of algorithms via these problems.

Kolmogorov complexity and the Incompressibility Method: The technical tool to obtain our results is the incompressibility method. This method is especially suited for the average case analysis of algorithms and machine models, whereas average-case analysis is usually more difficult than worst-case analysis using more traditional methods. A survey of the use of the incompressibility method is [7] Chapter 6, and recent work is [1]. A recent

*A preliminary version of this work was presented in part at the *26th International Colloquium on Automata, Languages, and Programming (ICALP99)*, Prague, Czech Republic, July 1999.

†Supported in part by the NSERC Research Grant OGP0046613 and a CITO grant. Address: Department of Computing and Software, McMaster University, Hamilton, Ont L8S 4K1, Canada. Email: jiang@cas.mcmaster.ca

‡Supported in part by the NSERC Research Grant OGP0046506, a CITO grant, and the Steacie Fellowship. Address: Department of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada. E-mail: mli@math.uwaterloo.ca

§Partially supported by the European Union through NeuroCOLT II ESPRIT Working Group. Address: CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Email: paulv@cwi.nl

application is a general nontrivial lower bound on the average-case complexity of multipass Shellsort, [2], a problem that seen had remarkably little progress in the last forty years [6].

Informally, the Kolmogorov complexity $C(x)$ of a binary string x is the length of the shortest binary program (for a fixed reference universal machine) that prints x as its only output and then halts [5]. A string x is *incompressible* if $C(x)$ is at least $|x|$, the approximate length of a program that simply includes all of x literally. Similarly, the *conditional* Kolmogorov complexity of x with respect to y, denoted by $C(x|y)$, is the length of the shortest program that, *w*ith extra information y, prints x. And a string x is incompressible *r*elative to y if $C(x|y)$ is large in the appropriate sense. For details see [7]. Here we use that, both absolutely and relative to any fixed string y, there are incompressible strings of every length, and that *most* strings are nearly incompressible, by *any* standard. [1] Another easy one is that significantly long subwords of an incompressible string are themselves nearly incompressible by *any* standard, even relative to the rest of the string. [2] In the sequel we use the following easy facts (sometimes only implicitly).

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

LEMMA 2 *If A is a finite set, then for every y, every element $x \in A$ has complexity $C(x|A,y) \leq \log |A| + O(1)$.*

The first lemma is proved by simple counting. The second lemma holds since x can be described by first describing A in $O(1)$ bits and then giving the index of x in the enumeration order of A.

2 Average Case of Bubble Sort: An Example

Generally speaking, the difficulty of analyzing the average-case complexity comes from the fact that one has to analyze the time complexity for all inputs of each length and then compute the average. This is a formidable task.

Using the incompressibility method, we choose just one input — a representative input. Via Kolmogorov complexity, we can show that the time complexity of this input is in fact the average-case complexity of all inputs of the algorithm on this length. Finding such a "representative input" is impossible, but we know it exists and this is sufficient.

The mothodology is best described via examples. We present the following average-case analysis of Bubble Sort.

EXAMPLE 1 A description and average-case analysis of Bubble Sort can be found in [6]. It is well-known that Bubble Sort uses $\Theta(n^2)$ comparisons/exchanges on the average. We

[1] By a simple counting argument one can show that whereas some strings can be enormously compressed, like strings of the form $11\ldots1$, the majority of strings can hardly be compressed at all. For every 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 $C(x) \geq n$. Similarly, for every length n and any binary string y, there is a binary string x of length n such that $C(x|y) \geq n$.

[2] Strings that are incompressible 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 [7].

present a very simple proof of this fact. The number of exchanges is obviously at most n^2, so we only have to consider the lower bound. In Bubble Sort we make at most $n-1$ passes from left to right over the permutation to be sorted and move the largest element we have currently found right by exchanges. For a permutation π of the elements $1, \ldots, n$, we can describe the total number of exchanges by $M := \sum_{i=1}^{n-1} m_i$ where m_i is the initial distance of element $n-i$ to its final position. Note that in every pass more than one element may "bubble" right but that means simply that in the future passes of the sorting process an equal number of exchanges will be saved for the element to reach its final position. That is, every element executes a number of exchanges going right that equals precisely the initial distance between its start position to its final position. A simple analysis as in [2] shows that $\log(M/n) \geq \log n + O(1)$ for a random permutation. As before this holds for an overwhelming fraction of all permutations, and hence gives us an $\Omega(n^2)$ lower bound on the expected number of comparisons/exchanges. $\diamond$

3 Sorting with Queues and Stacks

Knuth [6] and Tarjan [10] have studied the problem of sorting using a network of queues or stacks. In particular, the main variant of the problem is: assuming the stacks or queues are arranged sequentially as shown in Figure 1 or in parallel as shown in Figure 2, then how many stacks or queues are needed to sort n elements with comparisons only. The input sequence is scanned from left to right and the elements follow the arrows to go to the next stack or queue or output. We will concentrate on the average-case analyses of the above two main variants, although our technique in general apply to arbitrary acyclic networks of stacks and queues as studied in [10].

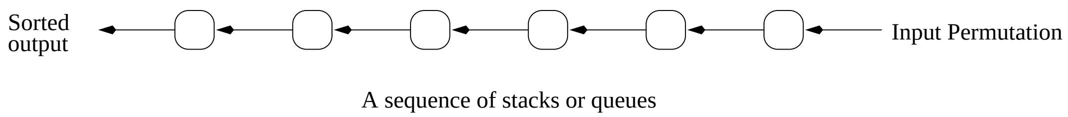

Figure 1: Six stacks/queues arranged in sequential order

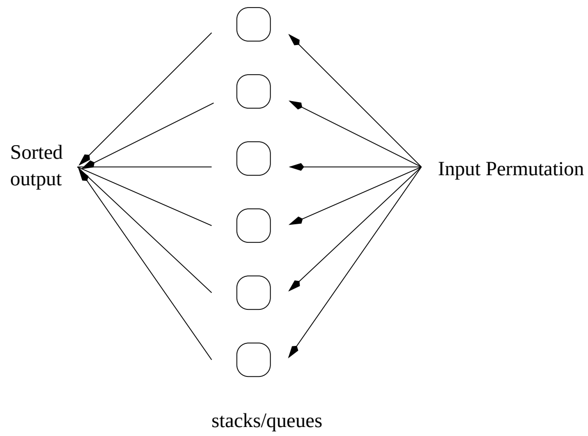

Figure 2: Six stacks/queues arranged in parallel order

3.1 Sorting with Sequential Stacks

The sequential stack sorting problem is in [6] exercise 5.2.4-20. We have k stacks numbered $S_0, \ldots, S_{k-1}$. The input is a permutation π of the elements $1, \ldots, n$. Initially we push the elements of π on S_0 at most one at a time in the order in which they appear in π. At every step we can pop a stack (the popped elements will move left in Figure 1) or push an incoming element on a stack. The question is how many stack are needed for sorting π. It is known that $k = \log n$ stacks suffice, and $\frac{1}{2} \log n$ stacks are necessary in the worst-case [6, 10]. Here we prove that the same lower bound also holds on the average with a very simple incompressibility argument.

THEOREM 1 *On the average, at least $\frac{1}{2} \log n$ stacks are needed for sequential stack sort.*

PROOF. Fix a random permutation π such that

$$C(\pi | n, P) \geq \log n! - \log n = n \log n - O(\log n),$$

where P is an encoding program to be specified in the following.

Assume that k stacks are sufficient to sort π. We now encode such a sorting process. For every stack, exactly n elements pass through it. Hence we need perform precisely n pushes and n pops on every stack. Encode a push as 0 and a pop as 1. It is easy to prove that different permutations must have different push/pop sequences on at least one stack. Thus with $2kn$ bits, we can completely specify the input permutation π. Then, as before,

$$2kn \geq \log n! - \log n = n \log n - O(\log n).$$

Hence, approximately $k \geq \frac{1}{2} \log n$ for the random permutation π.

Since most (a $(1 - 1/n)$th fraction) permutations are incompressible, we calculate the average-case lower bound as:

$$\frac{1}{2} \log n \cdot \frac{n-1}{n} + 1 \cdot \frac{1}{n} \approx \frac{1}{2} \log n.$$

□

3.2 Sorting with Parallel Stacks

Clearly, the input sequence $2, 3, 4, \ldots, n, 1$ requires $n - 1$ parallel stacks to sort. Hence the worst-case complexity of sorting with parallel stacks, as shown in Figure 2, is $n-1$. However, most sequences do not need this many stacks to sort in parallel arrangement. The next two theorems show that on the average, $\Theta(\sqrt{n})$ stacks are both necessary and sufficient. Observe that the result is actually implied by the connection between sorting with parallel stacks and *longest increasing subsequences* given in [10] and the bounds on the length of longest increasing subsequences of random permutations given in [4, 8, 3]. However, the proofs in [4, 8, 3] use deep results from probability theory (such as Kingman's ergodic theorem) and are quite sophisticated. Here we give simple proofs using incompressibility arguments.

THEOREM 2 *On the average, the number of parallel stacks needed to sort n elements is $O(\sqrt{n})$.*

Proof. Consider an incompressible permutation π satisfying

$$C(\pi|n) \geq \log n! - \log n. \tag{1}$$

We use the following trivial algorithm (which is described in [10]) to sort π with stacks in the parallel arrangement as shown in Figure 2. Assume that the stacks are named $S_0, S_1, \ldots$ and the input sequence is denoted as $x_1, \ldots, x_n$.

Algorithm Parallel-Stack-Sort

1. For $i = 1$ to n do

   Scan the stacks from left to right, and push x_i on the the first stack S_j whose top element is larger than x_i. If such a stack doesn't exist, put x_i on the first empty stack.

2. Pop the stacks in the ascending order of their top elements.

We claim that algorithm Parallel-Stack-Sort uses $O(\sqrt{n})$ stacks on the permutation π. First, we observe that if the algorithm uses m stacks on π then we can identify an increasing subsequence of π of length m as in [10]. This can be done by a trivial backtracing starting from the top element of the last stack. Then we argue that π cannot have an increasing subsequence of length longer than $e\sqrt{n}$, where e is the natural constant, since it is compressible by at most $\log n$ bits.

Suppose that σ is a longest increasing subsequence of π and $m = |\sigma|$ is the length of σ. Then we can encode π by specifying:

1. a description of this encoding scheme in $O(1)$ bits;

2. the number m in $\log m$ bits;

3. the combination σ in $\log \binom{n}{m}$ bits;

4. the locations of the elements of σ in π in at most $\log \binom{n}{m}$ bits; and

5. the remaining π with the elements of σ deleted in $\log(n-m)!$ bits.

This takes a total of

$$\log(n-m)! + 2\log\frac{n!}{m!(n-m)!} + \log m + O(1) + 2\log\log m$$

bits, where the last item $\log\log m$ is needed for self-delimiting encoding of m. Using Stirling approximation and the fact that $\sqrt{n} \leq m = o(n)$, we the above expression is upper bounded by:

$$\begin{aligned}
& \log n! + \log\frac{(n/e)^n}{(m/e)^{2m}((n-m)/e)^{n-m}} + O(\log n) \\
\approx\ & \log n! + m\log\frac{n}{m^2} + (n-m)\log\frac{n}{n-m} + m\log e + O(\log n) \\
\approx\ & \log n! + m\log\frac{n}{m^2} + 2m\log e + O(\log n)
\end{aligned}$$

This description length must exceed the complexity of the permutation which is lower-bounded in (1). This requires that (approximately) $m \leq e\sqrt{n} = O(\sqrt{n})$.

This yields an average complexity of Parallel-Stack-Sort of:

$$O(\sqrt{n}) \cdot \frac{n-1}{n} + n \cdot \frac{1}{n} = O(\sqrt{n}).$$

$\square$

THEOREM 3 *On the average, the number of parallel stacks required to sort a permutation is $\Omega(\sqrt{n})$.*

PROOF. Let A be a sorting algorithm using parallel stacks. Fix a random permutation π with $C(\pi|n, P) \geq \log n! - \log n$, where P is the program to do the encoding discussed in the following. Suppose that A uses T parallel stacks to sort π. This sorting process involves a sequence of moves, and we can encode this sequence of moves by a sequence of the following terms:

- push to stack i,
- pop stack j,

where the element to be pushed is the next unprocessed element from the input sequence and the popped element is written as the next output element. Each of these term requires $\log T$ bits. In total, we use precisely $2n$ terms since every element has to be pushed once and popped once. Such a sequence is unique for every permutation.

Thus we have a description of an input sequence in $2n \log T$ bits, which must exceed $C(\pi|n, P) \geq n \log n - O(\log n)$. It follows that $T \geq \sqrt{n} = \Omega(\sqrt{n})$.

This yields the average-case complexity of A:

$$\Omega(\sqrt{n}) \cdot \frac{n-1}{n} + 1 \cdot \frac{1}{n} = \Omega(\sqrt{n}).$$

$\square$

3.3 Sorting with Parallel Queues

It is easy to see that sorting cannot be done with a sequence of queues. So we consider the complexity of sorting with parallel queues. It turns out that all the result in the previous subsection also hold for queues.

As noticed in [10], the worst-case complexity of sorting with parallel queues is n since the input sequence $n, n-1, \ldots, 1$ requires n queues to sort. We show in the next two theorems that on the average, $\Theta(\sqrt{n})$ queues are both necessary and sufficient. Again, the result is implied by the connection between sorting with parallel queues and longest *decreasing* subsequences given in [10] and the bounds in [4, 8, 3] (with sophisticated proofs). Our proofs are almost trivial given the proofs in the previous subsection.

THEOREM 4 *On the average, the number of parallel queues needed to sort n elements is upper bounded by $O(\sqrt{n})$.*

PROOF. The proof is very similar to the proof of Theorem 2. We use a slightly modified greedy algorithm as described in [10]:

Algorithm Parallel-Queue-Sort

1. For $i = 1$ to n do

   Scan the queues from left to right, and append x_i on the the first queue whose rear element is smaller than x_i. If such a queue doesn't exist, put x_i on the first empty queue.

2. Delete the front elements of the queues in the ascending order.

Again, we claim that algorithm Parallel-Queue-Sort uses $O(\sqrt{n})$ queues on any permutation π that cannot be compressed by more than $\log n$ bits. We first observe that if the algorithm uses m queues on π then a decreasing subsequence of π of length m can be identified, and we then argue that π cannot have a decreasing subsequence of length longer than $e\sqrt{n}$, in a way analogous to the argument in the proof of Theorem 2.

□

THEOREM 5 *On the average, the number of parallel queues required to sort a permutation is $\Omega(\sqrt{n})$.*

PROOF. The proof is the same as the one for Theorem 3 except that we should replace "push" with "enqueue" and "pop" with "dequeue". □

4 Open Questions

The incompressibility method is a good tool to analyzing the average-case complexity of sorting algorithms. Simplicity has been our goal. Examples of such average-case analyses of some other algorithms are given in [1]. This methodology and applications can be easily taught to undergraduate students.

The average-case performance of Shellsort [2] has been one of the most fundamental and interesting open problems in the area of algorithm analysis. The simple average-case analysis of Bubble Sort, stack-sort and queue-sort are further examples to demonstrate the generality and simplicity of our technique in analyzing sorting algorithms in general. We believe the method can be widely applied for analyzing the average case complexity of many other problems. Some specific open questions are:

1. Tighten the average-case lower bound for Shellsort. The bound in [2] is not tight for $p = 2$ passes.

2. For sorting with sequential stacks, can we close the gap between $\log n$ upper bound and the $\frac{1}{2}\log n$ lower bound?

References

[1] H. Buhrman, T. Jiang, M. Li, and P. Vitányi, New applications of the incompressibility method, *ICALP'99*, July 11-15, 1999, Prague. Also in *Theoretical Computer Science*, 235(1999)

[2] T. Jiang, M. Li, and P. Vitányi, A lower bound on the average-case complexity of Shellsort, *J. Assoc. Comp. Mach.*, To appear. Also in *ICALP'99*, July 11-15, 1999, Prague.

[3] S.V. Kerov and A.M. Versik, Asymptotics of the Plancherel measure on symmetric group and the limiting form of the Young tableaux, *Soviet Math. Dokl.* 18 (1977), 527-531.

[4] J.F.C. Kingman, The ergodic theory of subadditive stochastic processes, *Ann. Probab.* 1 (1973), 883-909.

[5] A.N. Kolmogorov, Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1:1(1965), 1–7.

[6] D.E. Knuth, *The Art of Computer Programming, Vol.3: Sorting and Searching*, Addison-Wesley, 1973 (1st Edition), 1998 (2nd Edition).

[7] M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, New York, 2nd Edition, 1997.

[8] B.F. Logan and L.A. Shepp, A variational problem for random Young tableaux, *Advances in Math.* 26 (1977), 206-222.

[9] D.L. Shell, A high-speed sorting procedure, *Commun. ACM*, 2:7(1959), 30–32.

[10] R.E. Tarjan, Sorting using networks of queues and stacks, *Journal of the ACM*, 19(1972), 341–346.