

ON COMPUTING THE LENGTH OF LONGEST INCREASING SUBSEQUENCES*

Michael L. FREDMAN

*Department of Mathematics, Massachusetts Institute of Technology,
Cambridge, Mass. 02139, USA*

Received 14 March 1974

Revised 4 June 1974

Let $S = x_1, x_2, \dots, x_n$ be a sequence of n distinct elements from a linearly ordered set. We consider the problem of determining the length of the longest increasing subsequences of S . An algorithm which performs this task is described and is shown to perform $n \log n - n \log \log n + O(n)$ comparisons in its worst case. This worst case behavior is shown to be best possible.

1. Introduction

Let $S = x_1, x_2, \dots, x_n$ be a sequence of n distinct elements from a linearly ordered set. In this paper we examine the complexity of algorithms that compute the length L of the longest increasing subsequences of S :

$$L = \max \{k : 1 \leq i_1 < i_2 < \dots < i_k \leq n \text{ and } x_{i_1} < \dots < x_{i_k} \}$$

We shall describe an algorithm which performs this task and which has a worst case running time of $O(n \log n)$. This bound is shown to be best possible for a fairly general model of computation.

Our lower bound is obtained by isolating the sorting aspects of the problem. We show that a substantial amount of ordering information about the elements of S is required before the value of L is capable of unique determination. Specifically, we consider the class of algorithms that perform comparisons, $[x_i : x_j]$, and branch in accordance with the

* Supported in part by NSF Grant GP-22796 and ONR Contract N00014-67-A0204-0063.

outcomes $x_i < x_j$ or $x_i > x_j$. We show that any such algorithm providing sufficient information to compute L must perform at least $n \log n - n \log \log n + O(n)$ comparisons in its worst case. (Logarithms are to the base two.) Furthermore, the algorithm we shall be describing never performs more than an equivalent number of comparisons, resulting in a fairly accurate approximation to the best worst case number of comparisons required. While counting comparisons provides a lower bound for the overall timing of an algorithm, there is no immediate reason to believe that this number bears a linear relationship to an upper bound. For example, even if we completely sorted S , we would still have to do further work to determine L . However, we begin by describing an algorithm whose total running time is $O(n \log n)$. Throughout this paper we interchangeably use S to denote both the sequence x_1, \dots, x_n and the set $\{x_1, \dots, x_n\}$.

2. An upper bound

We describe an algorithm due to Knuth whose mechanism amounts to computing the first row of the Young tableau associated with S (see [2, Section 5.1.4]).

We maintain a table $T(j)$ which initially has $T(1) = x_1$ and is otherwise empty. Then as j proceeds from 2 until n , we insert x_j into the table as follows. Assuming $T(k)$ is the last non-empty position of T , we compare x_j with $T(k)$. If $x_j > T(k)$, we set $T(k+1) = x_j$. Otherwise we find the least index $m \geq 1$ such that $x_j < T(m)$ and replace the current value of $T(m)$ by x_j . After all the elements of S have been processed, L turns out to be the number of non-empty positions of T . The validity of this algorithm is easily demonstrated. Assume x_1, x_2, \dots, x_j have just been processed. At this stage the values $T(1), T(2), \dots$ have the following interpretation: $T(k)$ is the least element in $\{x_1, \dots, x_j\}$ which constitutes the last term of an increasing subsequence of x_1, \dots, x_j of length k . This interpretation can be directly verified by induction on j , and the validity of the algorithm follows at once.

Because the second to the last term of an increasing subsequence of length $k+1$ comprises the last term of an increasing subsequence of length k , at any stage of the algorithm either $T(k) < T(k+1)$ or $T(k+1)$

is empty. Therefore $T(1) < T(2) < \dots$, and if we are processing x_j , we can determine its proper location in T using a binary search procedure which can be done in time $O(\log n)$. We conclude that this algorithm can be performed in time $O(n \log n)$.

Now let us carefully count the number of comparisons we might have to perform. Assuming we are inserting x_j into T and that k positions of T are non-empty, as we shall see, it is advantageous to first compare x_j with $T(k)$, and then if $x_j < T(k)$, perform $\lceil \log k \rceil$ further comparisons potentially required by the binary search; while if $x_j > T(k)$ set $T(k+1) = x_j$. We never perform more than $\log L + O(1)$ comparisons for each insertion, and each time a new position of T is filled, which occurs L times, only one comparison is performed. Therefore, no more than

$$(1) \quad (n - L) \log L + O(n)$$

comparisons can ever be required. The worst value for L is roughly $n/\log n$; in which case, (1) becomes $n \log n - n \log \log n + O(n)$. This proves the following theorem.

Theorem 2.1. *There exists an algorithm for computing L whose total running time is $O(n \log n)$, and which performs $n \log n - n \log \log n + O(n)$ comparisons in its worst case.*

3. A lower bound

We can represent the comparison aspects of an algorithm that computes L using a binary comparison tree T . (An internal node of the tree is labeled with a comparison $[x_i : x_j]$, and branching goes to the left if $x_i < x_j$, otherwise to the right.) Any one of the $n!$ possible linear orderings on S defines a path through T leading from the root and ending at an external node (or leaf). Let us assume that wasteful or redundant comparisons have been pruned or removed from T (comparisons whose outcomes are predictable on the basis of the outcomes of the previous comparisons along the path from the root), so that to each leaf \mathcal{L} of T there corresponds a partial ordering on S defined by the transitive closure of the outcomes of the comparisons along the path leading to \mathcal{L} . Furthermore, to each leaf \mathcal{L} there corresponds a non-empty set of lin-

ear orderings on S such that each linear ordering in this set defines the path through T ending at \mathcal{L} . This set, of course, is the set of linear embeddings of the partial ordering corresponding to \mathcal{L} .

At this point we can best regard L as being an attribute of the linear ordering defined on S . For an algorithm to successfully compute L , its associated comparison tree T must have the property that all of the linear orderings in the set associated with any leaf must define the same value for L . We need the following definitions and lemmas.

Let (P, \leq) be a partially ordered set. A chain is defined to be a subset of P linearly ordered by \leq . An antichain is defined to be a family of pairwise incomparable elements. Our first lemma is a statement of Dilworth's theorem. A proof is given in [1, Chapter 7].

Lemma 3.1 (Dilworth). *A finite partially ordered set (P, \leq) can be partitioned into m chains, where m is the size of its largest antichain.*

Lemma 3.2. *Let P be a finite partially ordered set and let Q be a subset of P . Any linear embedding of Q can be extended to a linear embedding of P . In other words, it is possible to linearly embed P in such a manner, that when restricted to the elements in Q , this embedding coincides with a previously given linear embedding of Q .*

Proof. Let \leq be the partial ordering on P . Extend this partial ordering by imposing upon it the linear embedding of Q . Any linear embedding of this extended partial ordering will satisfy the lemma.

Lemma 3.3. *If a linearly ordered set is partitioned into k chains, the original ordering can be algorithmically restored with at most $n \lceil \log k \rceil$ comparisons.*

Proof. This lemma is an easy consequence of the fact that two chains of size k_1 and k_2 can be merged with at most $k_1 + k_2 - 1$ comparisons.

Lemma 3.4. *Let $S(n, k)$ denote the number of linear orderings on S that define a value for L that is less than k . Then*

$$(2) \quad S(n, k) \geq n! \left(1 - \binom{n}{k}/k!\right).$$

Proof. The probability that a particular subsequence $x_{i_1}, x_{i_2}, \dots, x_{i_k}$ is increasing is $1/k!$. Since there are $\binom{n}{k}$ possible subsequences of length k , we conclude that the probability that $L \geq k$ is $\leq \binom{n}{k}/k!$. The lemma follows at once.

We are now in a position to obtain a crude lower bound on the worst case number of comparisons required to compute L . Our argument is basically information theoretic. Let A be an algorithm that computes L and let T be its associated pruned comparison tree. If T has N leaves, then T must have a path of length $\geq \log N$. Because there corresponds at least one linear embedding to each leaf of T , in at least one case A must perform at least $\log N$ comparisons. Our strategy is to show that N is large.

Theorem 3.5. *An algorithm that computes L must in its worst case perform at least $\frac{1}{2}n \log n + O(n)$ comparisons.*

Proof. Given a fixed k , to be chosen below, we derive this lower bound for algorithms that compute the answer to the more simple question, is $L \geq k$? Let A be such an algorithm and let T be its associated comparison tree. Consider those leaves of T associated with the ultimate conclusion that $L < k$. We claim that the partially ordered sets associated with these leaves have no antichains of size k . For if there were such an antichain $\{x_{i_1}, \dots, x_{i_k}\}$, we could linearly embed it so that $x_{i_r} < x_{i_s}$ if $i_r < i_s$, and by Lemma 3.2 this embedding could be extended to a linear embedding of S . But for this embedding we would have $L \geq k$, contrary to assumption. Therefore, by Lemma 3.1, the partially ordered sets associated with these leaves can be partitioned into fewer than k chains. Now consider the following enhancement A^* of A . Whenever A concludes that $L < k$, A^* continues to completely sort S , which, by Lemma 3.3, requires no more than $n \log k + O(n)$ further comparisons. Denoting by T^* the pruned comparison tree associated with A^* , clearly T^* must have at least $S(n, k)$ leaves, and therefore, must in its worst case perform at least $\log S(n, k)$ comparisons. Consequently, A must perform at least $\log S(n, k) - n \log k + O(n)$ comparisons in its worst case. Choosing $k = \lfloor 3n^{1/2} \rfloor$, by Lemma 3.4, $S(n, k) \sim n!$, and our theorem follows immediately.

The algorithm we have described earlier is shown by (1) to be capable of computing the answer to the question, is $L \geq k$?, by performing no more than $n \log k + O(n)$ comparisons. Therefore, setting $k = \lceil 3n^{1/2} \rceil$, we conclude that the bound obtained in the above proof is correct to within $O(n)$ comparisons, for this simplified task. Furthermore, for $k \leq O(n^{1/2})$, these arguments can be generalized to prove a best worst case estimate of $n \log k + O(n)$ comparisons, although a better estimate for $S(n, k)$ than that given by (2) is required, involving the enumerative theory of Young tableaux. This strongly suggests that $n \log k + O(n)$ comparisons are required for a much wider range of k , but the above lower bound proof breaks down. What we need is a stronger application of Dilworth's result.

Lemma 3.6. *Let \leq be a partial ordering defined on S . The maximum value of L associated with any linear embedding of this ordering, is equal to the minimum number of decreasing subsequences relative to \leq into which S can be partitioned.*

Proof. First, it is obvious that if we can partition S into k decreasing subsequences relative to \leq , then for no linear embedding can we have $L > k$. Hence, we have an inequality going one way.

Let \leq' be the following partial ordering on S ; $x_i \leq' x_j$ if and only if $x_i \leq x_j$ and $j \leq i$. The ordering \leq' is embedded in \leq and a chain in \leq' corresponds to a decreasing subsequence relative to \leq . If S cannot be partitioned into fewer than k decreasing subsequences relative to \leq , then relative to \leq' , there exists an antichain of k elements by Lemma 3.1. Let $x_{i_1}, x_{i_2}, \dots, x_{i_k}, i_1 < i_2 < \dots < i_k$, constitute such an antichain. If $i_r < i_s$, then because x_{i_r} and x_{i_s} are incomparable relative to \leq' , either $x_{i_r} < x_{i_s}$, or x_{i_r} and x_{i_s} are incomparable relative to \leq . Therefore, on the set $Q = \{x_{i_1}, \dots, x_{i_k}\}$, the ordering \leq'' , defined by $x_{i_r} \leq'' x_{i_s}$ if and only if $i_r \leq i_s$, is a linear embedding of \leq on Q . By Lemma 3.2, this can be extended to a linear embedding of S , and for this embedding, $L \geq k$. Hence, we have the opposite inequality, completing the proof.

Theorem 3.7. *An algorithm that computes L must in its worst case perform at least $n \log n - n \log \log n + O(n)$ comparisons.*

Proof. Choose $k \leq \frac{1}{2}n$. We define the following set Γ of linear orderings on S . Each linear ordering in Γ partitions S into k decreasing subsequences S_1, S_2, \dots, S_k , such that $x_i \in S_i$ and $x_{n-k+i} \in S_i$ for $1 \leq i \leq k$, and each element in S_i is less than each element in S_{i+1} for $1 \leq i \leq k-1$. The orderings in Γ are completely specified only by having to state to which unique set S_j the element x_j belongs, for $k < j \leq n-k$. The number of such orderings therefore is k^{n-2k} . Furthermore, given an ordering in Γ , there can be only one way to partition S into k decreasing subsequences relative to this ordering, namely these must be the subsequences S_1, \dots, S_k . To see this, try partitioning

$$x_1, x_2, \dots, x_k, x_{n-k+1}, x_{n-k+2}, \dots, x_n$$

into k decreasing subsequences. We are forced to choose the subsequences $x_1, x_{n-k+1}; x_2, x_{n-k+2};$ etc. This in turn forces the placement of the remaining elements of S .

Now let T be the pruned comparison tree associated with an algorithm that computes L . We show that no two of the orderings in Γ can be associated with the same leaf. Consider the leaf of T associated with a particular ordering A in Γ . As discussed earlier, since $L = k$ for the ordering A , all linear embeddings of the partial ordering of this leaf must define $L = k$. Hence, by Lemma 3.6, this partial ordering defines a partition of S into k decreasing subsequences. These k decreasing subsequences must be the subsequences $\{S_i\}$ that define A , since S can be partitioned into k decreasing subsequences in only one way under the ordering A . If any other ordering in Γ is associated with this leaf, it would also be consistent with the $\{S_i\}$ partition and therefore be A itself. Finally, since $|\Gamma| = k^{n-2k}$, T must have a path of length $\geq (n-2k) \log k$. Choosing $k = \lceil n/\log n \rceil$ completes our proof.

References

- [1] M. Hall, Jr., *Combinatorial Theory* (Blaisdell, Waltham, Mass., 1967).
- [2] D.E. Knuth, *The Art of Computer Programming*, Vol. 3 (Addison-Wesley, Reading, Mass., 1973).