On the longest increasing subsequence of a circular list[*]

M. H. Albert[†] M. D. Atkinson[†] Doron Nussbaum[‡]
Jörg-Rüdiger Sack[‡] Nicola Santoro[‡]

Abstract

The longest increasing circular subsequence (LICS) of a list is considered. A Monte-Carlo algorithm to compute it is given which has worst case execution time $O(n^{3/2} \log n)$ and storage requirement $O(n)$. It is proved that the expected length $\mu(n)$ of the LICS satisfies $\lim_{n\to\infty} \frac{\mu(n)}{2\sqrt{n}} = 1$. Numerical experiments with the algorithm suggest that $|\mu(n) - 2\sqrt{n}| = O(n^{1/6})$.

1 Introduction

The properties of the longest increasing subsequence (LIS) of a finite sequence of numbers have inspired a number of research areas in mathematics and computer science over many decades. As long ago as 1935 Erdös and Szekeres showed that every sequence of length n has an increasing subsequence or a decreasing subsequence of length about $\sqrt{n}$. It follows immediately that the expected length of an LIS in a random permutation of length n is at least $\frac{1}{2}\sqrt{n}$. That result was the first in a series of investigations (see [3] for a survey) that culminated in the seminal paper [5] which obtained the complete limiting distribution of the length of an LIS in a permutation of length n chosen uniformly at random. An important step in this research was taken by Baer and Brock [4] who correctly estimated the expected length to be $2\sqrt{n}$ by computer simulation. Their work ties in with another aspect of the LIS problem: to find efficient algorithms for computing the LIS. That problem was solved by Schensted by a now classical textbook algorithm (see e.g., [7, 10, 11, 13]) for computing an LIS in time $O(n \log n)$ based on dynamic programming, and that algorithm in turn has

[†]Department of Computer Science, University of Otago, Dunedin, New Zealand.
[‡]School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario K1S5B6, Canada.
[*]Research supported in part by NSERC and SUN Microsystems.

connections to the study of Young tableaux. Computing the LIS has recently gained some practical importance since it is used in the MUMmer system [8] for aligning whole genomes. Fredman [9] has shown that the dynamic programming algorithm is optimal under the comparison model.

In this paper we study a variant of the LIS problem. We shall regard the given sequence as a circular structure. In other words, we shall allow the LIS to wrap around if necessary. The longest increasing circular sequence (LICS) is defined as the longest increasing subsequence when wrap-around is permitted. The LICS is never more than twice as long as the LIS but examples such as $4, 5, 6, 1, 2, 3$ show that this bound can be attained. Throughout the paper we interpret "increasing" in the non-strict sense. Strictly increasing sequences can be found by a minor variation.

Although the LICS problem seems to be a natural extension of the LIS problem we know of only one paper that bears upon it. In [2] a (Las Vegas) randomized algorithm was developed to compute the LIS in all windows of a given span. If applied to a sequence of the form XX (a sequence X concatenated with itself) then, with a window span $n = |X|$, it can compute the LICS of X in worst case time $O(n^2)$ and expected time $O(n^{3/2})$.

The contribution of the present paper is both practical and theoretical. Specifically, we give a Monte-Carlo randomized algorithm for the LICS problem whose worst case run-time is $O(n^{3/2} \log n)$ with tiny error probability, and storage requirement $O(n)$ which is simple to program. The algorithm depends on a result (Proposition 1) that gives a connection between the LIS and LICS which appears interesting in its own right. Then we prove that the expected length of the LICS is asymptotic to $2\sqrt{n}$ which, since this is true also for the LIS, is somewhat surpising. Finally we report on some numerical evidence that suggests an even tighter result.

The LICS problem can be thought of as a special case of a class of permutation problems that was introduced in [1]. In this broader framework one is given a fixed set $\mathcal{A}$ of permutations and some input permutation σ of length n; the task is to compute the longest subsequence of σ that is order isomorphic to one of the permutations in $\mathcal{A}$. The ordinary LIS problem is the special case that $\mathcal{A}$ is the set of identity permutations; the LICS problem is the case that $\mathcal{A}$ is the set of all permutations $k+1, k+2, \ldots, m, 1, 2, \ldots, k$ for some k, m. As noted in [1], there are only few classes $\mathcal{A}$ for which an $O(n \log n)$ algorithm is known.

Besides the fact that circular lists are a natural extension of linear lists another reason for seeking a solution to the LICS problem is that genomes of bacteria are (considered to be) circular [6]; so circular problems do arise naturally.

2 Review of the LIS algorithm

An important ingredient of our approach is the standard dynamic programming algorithm to find an LIS. To make the paper self-contained and to clarify what can be gleaned from this algorithm we review its operation.

Suppose we are given a sequence $X = x_1 x_2 \cdots x_n$. We scan the sequence term by term and maintain at every step certain positions $t_1, t_2, \ldots t_r$. The term x_{t_k} is the value of the least possible ending term in an increasing subsequence of length k in the prefix of the sequence that has been scanned to this point. Initially we have $r = 0$ (and $t_0 = 0, x_0 = -\infty$, by convention) indicating that, before the sequence is examined, no increasing subsequences have been identified. Notice that we shall necessarily have

$$x_{t_1} \leq x_{t_2} \leq \ldots \leq x_{t_r}$$

since, if $x_{t_{k-1}} > x_{t_k}$, the terminator x_{t_k} of an increasing subsequence of length k will be preceded by the penultimate term of that subsequence, and that term will be a smaller terminator than $x_{t_{k-1}}$ of an increasing subsequence of length $k-1$).

When we inspect the term $y = x_i$ (which we do for values $i = 1, 2, \ldots, n$ in turn) we have to update the positions $t_1, t_2, \ldots, t_r$. To do this, we locate (using binary search), the index s for which

$$x_{t_{s-1}} \leq y < x_{t_s}$$

If such an index is found then we know that y extends an increasing sequence of length $s-1$ ending at position t_{s-1} and that this new sequence of length s has a smaller terminator than x_{t_s}; thus we redefine t_s to be i. The only situation where s cannot be located is the case $x_{t_r} \leq y$; this implies that, for the first time, we have encountered an increasing subsequence of length $r+1$, so we set t_{r+1} equal to i and increment r. We also define back pointers b_i by setting $b_i = t_{s-1}$ (or, in the latter case, $b_i = t_{r-1}$). The back pointers record how x_i was established as the final term of an increasing sequence of length s.

When the entire sequence X has been inspected the final value of r is the length of the LIS. We can then reconstruct (in reverse) the LIS itself by following back pointers from position t_r. Indeed, by recording the value of t_r for each of $i = 1, 2, \ldots, n$, we can reconstruct an LIS in any initial segment of X.

Clearly this algorithm takes time $O(n \log n)$. Notice that the algorithm can equally be used to construct a longest decreasing subsequence. It can also operate from right to left if desired.

We note some technical properties of the LIS algorithm. Let $t_1^{(i)}, t_2^{(i)}, \ldots$ denote the values of the variables $t_1, t_2 \ldots$ at the point that x_i has just been processed. The position i itself will occur among $t_1^{(i)}, t_2^{(i)}, \ldots$ and, of course, all the other positions of this set will be less than i. We define s_i by

$$t^{(i)}_{s_i} = i$$

Thus s_i is the length of the increasing subsequence that x_i created (either for the first time or by having a smaller final term than previously known increasing subsequences of length s_i). Clearly

$$t^{(i)}_j \leq t^{(i+1)}_j$$

It is convenient to think of the positions

$$t^{(i)}_j, t^{(i+1)}_j, t^{(i+2)}_j \dots$$

as being the positions of X which "improve" increasing subsequences of length j (in the sense that they define final positions with decreasing values).

Lemma 1 *Let i, j be two positions of X with $i < j$ and $x_i \leq x_j$. Let $j \leq q$ and $s_j = s_q$. Then*

1. *$x_j \geq x_q$ (with strict inequality if $j < q$) and*
2. *there exists an index p with $i \leq p < q$, $s_i = s_p$, and $x_p \leq x_q$*

Proof: The hypotheses $s_j = s_q$ and $j \leq q$ tell us that both x_j and x_q improved increasing subsequences of length s_j and that x_j precedes x_q. If x_q is a strictly later improvement (that is, $j < q$) we have $x_j > x_q$.

Put $p = t^{(q)}_{s_i}$. In other words, just after x_q was processed, p was the position where the current "best" increasing subsequence of length s_i ended. Therefore both x_i and x_p, when they were processed, each created improved increasing subsequences of length s_i; that is to say, $s_i = s_p$. Furthermore, as $x_i \leq x_j$, x_j improved an even longer subsequence than the one improved by x_i; that is, $s_i < s_j$.

Since q is the unique maximal position of $\{t^{(q)}_1, t^{(q)}_2, \dots\}$ we have $p < q$ (with strict inequality since $s_p < s_q$).

Finally, as the values of the sequence X at the positions $\{t^{(q)}_1, t^{(q)}_2, \dots\}$ are non-decreasing we have $x_p \leq x_q$. ∎

Corollary 1 *Let $Y = x_{u_1} x_{u_2} \cdots x_{u_m}$ be an increasing subsequence of X. Then there exists an increasing subsequence $Z = x_{v_1} x_{v_2} \cdots x_{v_m}$ such that*

1. *$u_1 \leq v_1$,*
2. *$x_{u_m} \geq x_{v_m}$,*

3. $v_m \in \{t_1^{(n)}, t_2^{(n)}, \ldots\}$

Proof: Let $d = s_{u_m}$ and let $t_d^{(n)} = v_m$. In other words, when x_{u_m} was processed it improved an increasing sequence of length d and this was improved for a final time when x_{v_m} was processed. So, with $j = u_m$ and $q = v_m$ we have $j \leq q$ and $s_j = s_q$. Also, if $i = u_{m-1}$ then $i < j$ and $x_i \leq x_j$. In other words the hypotheses of the previous lemma hold.

The lemma tells us that $x_{u_m} \geq x_{v_m}$ and that there is an index $p = v_{m-1}$ with $u_{m-1} \leq v_{m-1} < v_m$, that $s_i = s_p$ (i.e. $x_{v_{m-1}}$ improves the sequence that $x_{u_{m-1}}$ improved), and that $x_{v_{m-1}} \leq x_{v_m}$.

We can now repeat the argument and successively construct the sequence Z with the appropriate properties. ■

3 The algorithm for the LICS

We first discuss a deterministic algorithm for computing the LICS of a sequence X whose worst case time is still no better than $O(n^2 \log n)$ (although its expected time is $O(n^{3/2} \log n)$). Then we give the Monte-Carlo variant that, with vanishing error probability, has worst case time $O(n^{3/2} \log n)$.

The algorithm is based on two ideas. The first is an algorithm A_1 for finding the LICS *that contains a specific term of* X. The second is an algorithm A_2 for identifying a set of candidates for terms of X which are part of some LICS. The standard LIS algorithm is used in both methods. The LICS itself is computed by applying A_1 to each of the candidates found by A_2 and selecting the longest.

Suppose we are given a term of X and wish to construct a LICS containing this term. For convenience we rotate X so that the given term is x_n. Let X_L and X_U be (respectively) the subsequences of X whose terms x_k satisfy $x_k \leq x_n$ and $x_k \geq x_n$. We run the LIS algorithm twice. First we run it in the left to right version (as described above) on the subsequence X_U; this is done by processing X from left to right and ignoring terms less than x_n. For each k we record the length v_k of the LIS of $X_U \cap x_1 x_2 \cdots x_k$.

In the second run of the LIS we process X from right to left searching for sequences that *decrease from right to left*, and we consider only terms of (the reverse of) X_L. This allows us to find longest increasing subsequences in all suffixes of X_L. For each k we record the length u_k of the LIS of $X_L \cap x_{n-k+1} x_{n-k+2} \cdots x_n$.

An LICS containing x_n will have an initial part of length u_{n-k} in the segment $x_{k+1} \cdots x_n$ and a final part of length v_k in the segment $x_1 \cdots x_k$. Thus we can identify the LICS containing x_n by choosing k to maximize $u_{n-k} + v_k$ and then use the output of the two runs of the LICS above to find the two constituents of the LICS.

Clearly A_1 runs in time $O(n \log n)$. However, for A_1 to correctly identify the LICS we must give it a term of X that lies within some LICS. Algorithm A_2 finds candidates for such terms and is very simple: we just run the ordinary LIS algorithm and note the final values of $t_1, \ldots, t_r$; these are the candidate positions. The following result justifies this process.

Proposition 1 *If $t_1, \ldots, t_r$ is the output of the ordinary LIS then there is an LICS that passes through one of these positions.*

Proof: Consider any LICS. It has some initial subsequence $Y = x_{u_1} x_{u_2} \cdots x_{u_m}$ and then continues cyclically from the beginning of X as a subsequence W say. By Corollary 1 there exists an increasing subsequence Z that begins no earlier than Y, has final position one of $t_1, \ldots, t_r$, and whose final term is no larger than the final term of Y. It follows that ZW is a LICS with the stated property. ■

Suppose that A_2 computed m candidate terms through which the LICS passed. Then we have to run A_1 a total of m times and so we shall find the LICS in time $O(mn \log n)$. However, m is the length of an LIS of X and its expected value is known to be close to $2\sqrt{n}$ which means that our LICS algorithm has expected execution time $O(n^{3/2} \log n)$.

The Monte-Carlo variant simply modifies this last step. If A_2 has computed m candidates then, if $m \leq 3\sqrt{n}$, we proceed as above for a total execution time $O(n^{3/2} \log n)$. However, if $m > 3\sqrt{n}$, we proceed differently and exploit that the length of the LICS is at least equal to m. We sample values from X uniformly at random and, for each one, compute the longest increasing circular sequence that passes through it. The probability p that a randomly sampled element of X belongs to a LICS is at least m/n. Hence the expected number of elements of X that have to be sampled before one is found in the LICS is at most n/m.

Indeed, suppose we want to ensure that the probability of failing to find the LICS is bounded above by ϵ. Then we require k trials where $(1-p)^k < \epsilon$. Taking logarithms and using $\ln(1-p) < -p$ we find that k should exceed $-\ln(\epsilon)/p$. Hence, if $m > 3\sqrt{n}$, it suffices to choose $k > -\sqrt{n} \ln(\epsilon)/3$.

4 The expected length of the LICS

In this section we discuss bounds on the expected length of the LICS and report on some numerical experiments to estimate the mean μ and variance σ^2 of the length of the LICS. For the LIS itself these statistics are known. From [5] we have

$$\mu(LIS) = 2n^{1/2} - \gamma n^{1/6} + o(n^{1/6})$$

and

$$\sigma(LIS) = \delta n^{1/6} + o(n^{1/6})$$

where $\gamma = 1.711$ and $\delta = 0.902$.

The following result shows that, asymptotically, the length of the LICS is the same as that of the LIS. This may seem somewhat surprising but, as the proof below indicates, it is essentially because of the tight concentration of the LIS distribution around its mean.

Theorem 1

$$\lim_{n\to\infty} \frac{\mathbf{E}\, LICS_n}{2\sqrt{n}} = 1.$$

Proof: The permutations of length n fall into $(n-1)!$ classes, the permutations of each class being cyclic rotations of one another. In each class, choose a permutation whose LICS is actually a LIS and let M_n be the set of all such representatives. Let C be such a class and θ its representative. Then

$$\sum_{\pi\in C} |LICS(\pi)| = n\, |LIS(\theta)|$$

Thus:

$$\sum_{\pi\in S_n} |LICS(\pi)| = n \sum_{\theta\in M_n} |LIS(\theta)|\,.$$

A consequence of formula (1.8) of [5] is that for some constants b and c, any $0 < \epsilon < 5/6$, and sufficiently large n:

$$\mathbf{Pr}\left(LIS_n > 2\sqrt{n} + n^{1/6+\epsilon}\right) \le b\exp(-cn^{3\epsilon/5}).$$

Let U_n be the set of those $\theta \in M_n$ whose LIS has length greater than $2\sqrt{n} + n^{1/6+\epsilon}$, and let $D_n = M_n \backslash U_n$. In all of S_n there are fewer than $n!b\exp(-cn^{3\epsilon/5})$ permutations whose LIS has length greater than $2\sqrt{n}+n^{1/6+\epsilon}$. So in particular:

$$\sum_{\theta\in U_n} |LIS(\theta)| < n(n!)b\exp(-cn^{3\epsilon/5})$$

since n is an upper bound on $LIS(\theta)$ for all θ. Also, since $|D_n| \le (n-1)!$,

$$\sum_{\theta\in D_n} |LIS(\theta)| < (n-1)!\left(2\sqrt{n} + n^{1/6+\epsilon}\right)$$

Using these estimates we obtain

$$\begin{aligned}
\mathbf{E}\,LICS_n &= \frac{1}{n!}\sum_{\pi\in S_n}|LICS(\pi)| \\
&= \frac{1}{(n-1)!}\sum_{\theta\in M_n}|LIS(\theta)| \\
&= \frac{1}{(n-1)!}\sum_{\theta\in U_n}|LIS(\theta)| + \frac{1}{(n-1)!}\sum_{\theta\in D_n}|LIS(\theta)| \\
&< bn^2\exp(-cn^{3\epsilon/5}) + 2\sqrt{n} + n^{1/6+\epsilon} \\
&= 2\sqrt{n} + o(\sqrt{n}).
\end{aligned}$$

Therefore

$$\lim_{n\to\infty}\frac{\mathbf{E}\,LICS_n}{2\sqrt{n}} \leq 1.$$

The reverse inequality is trivial as the LICS is at least as long as the LIS. ■

The argument in this proof encourages the conjecture that the expected length of the LICS may be very close to $2\sqrt{n} + n^{1/6}$.

For each of various lengths up to 20000 we have generated 100000 random sequences, found the LICS in each, and computed the mean length and variance. We did not use the sampling technique even when m exceeded $3\sqrt{n}$ and so we can be sure that the means in our samples are correct. We found a very close agreement between the mean and $2\sqrt{n}$ although we are doubtful that they agree to within a constant. The following subset of our results for the mean and variance are typical.

n	10000	12000	14000	16000	18000	20000
μ	200.145	219.325	236.959	253.376	268.794	283.371
$\mu - 2\sqrt{n}$	0.0094	-0.0097	0.0099	0.0101	0.0103	0.0104
σ^2	8.899	9.341	9.776	10.227	10.552	10.880

Although the values of σ^2 for this sample do not provide sufficient evidence for a definitive conjecture, they are not plainly inconsistent with $\sigma^2 = cn^{1/3} + o(n^{1/3})$.

References

[1] M. H. Albert, R .A. Aldred, M. D. Atkinson, H. van Ditmarsch, B. Handley, C. C. Handley, J. Opatrny: Longest subsequences in permutations, Australian J. Combinatorics 28 (2003), 225-238.

[2] M. H. Albert, A. Golynski, A. M. Hamel, A. López-Ortiz, S. R. Rao, M. A. Safari: Longest increasing subsequences in sliding windows, Theoretical Computer Science 321 (2004) 405–414.

[3] D. Aldous, P. Diaconis: Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem, Bull. Amer. Math. Soc. 36, pp. 413-432, 1999.

[4] R. M. Baer and P. Brock: Natural sorting over permutation spaces, Math. Comp., 22, (1968), 385-510.

[5] J. Baik, P. Deift and K. Johansson: On the Distribution of the Length of the Longest Increasing Subsequence of Random Permutations, J. Amer. Math. Soc., 12, (1999) 1119-1178.

[6] R. Charlebois, 1999. Organization of the Prokaryotic Genome. ASM Press, Washington, D.C.http://www.life.uiuc.edu/micro/316/topics/chroms-genes-prots/chromosomes.html

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest: *Introduction to algorithms*, MIT Press, McGraw-Hil Book Company (New York), 1990.

[8] A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Paterson, O. White, S.L. Salzberg: Alignment of whole genomes, Nucleic Acis Research 27, pp. 2369-2376, 1999.

[9] M.L. Fredman: On computing the length of longest increasing subsequences, Discrete Math. 11, pp. 29-35, 1975.

[10] D. Gries: *The Science of Programming*, Springer Verlag (Heidelberg, New York), 1981.

[11] U. Manber: *Introduction to algorithms*, Addison-Wesley (Reading, Mass.), 1989.

[12] C. Schensted: Longest increasing and decreasing subsequences, Canad. J. Math 13, pp. 179-191, 1961.

[13] J.M. Steele: *Probability Theory and Optimization*, SIAM, 1997.