ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Том 101

ANNUAL OF SOFIA UNIVERSITY „ST. KLIMENT OHRIDSKI“

FACULTY OF MATHEMATICS AND INFORMATICS
Volume 101

ASYMPTOTICALLY FASTEST SORTING ALGORITHM FOR ALMOST SORTED ARRAYS

STEFAN GERDJIKOV

The *patience sorting algorithm* was introduced by Mellows. If a given array has n elements and can be considered as a shuffle of m already sorted arrays, then the patience algorithm sorts the original array in $O(n \log m)$ time. In the current paper we show that this upper bound is worst-case optimal even if the minimum value of the parameter m is known in advance.

Keywords: Patience sorting algorithm, worst-case optimality, increasing subsequences
2000 Math. Subject Classification: 68W40

1. INTRODUCTION

We consider the problem of sorting a sequence of n distinct numbers. Although this problem is well studied and optimal $O(n \log n)$ worst-case and average-case algorithms have been developed [7], there is no exact estimate of the complexity of these algorithms with respect to the disorder in the sequence.

In the current paper we consider the *patience sorting algorithm* introduced by Mellows, [8, 9, 1, 2]. Essentially, this approach of sorting real numbers first splits the given array into a minimal number of increasingly sorted subarrays and afterwards merges the resulting arrays. Using a result of Fredman, [5], it can be easily shown that this algorithm runs in $O(n \log m)$-time for every sequence of size n that contains no decreasing subsequence of size $m + 1$. Note that m is not previously known to the algorithm. However, even if an upper bound for m is known in advance no better worst-case algorithm exists as we prove in Section 3.

The rest of this paper is organized as follows. In Section 2 we outline the patience sorting algorithm in details, prove its correctness and its time-complexity. In Section 3 we argue that the algorithm is worst-case optimal and in Section 4 we conclude.

2. PATIENCE ALGORITHM DESCRIPTION

In this section we assume that $a_1, \ldots, a_n$ is a sequence of distinct numbers that is to be sorted in *increasing* order. To this end we describe an $O(n \log m)$-time algorithm where m is the size of the longest *decreasing* subsequence, i.e. m is maximal natural number with the property :

$$\text{there are } \; i(1) < i(2) < \cdots < i(m), \text{ such that } \; a_{i(1)} > a_{i(2)} > \cdots > a_{i(m)}.$$

The patience algorithm consists of two steps, [8, 9]:

1. Split $a_1, \ldots a_n$ into minimum number of increasing subsequences:
$$\{a_{1,1} < \cdots < a_{1,k_1}\}, \ldots, \{a_{M,1} < \cdots < a_{M,k_M}\}.$$

2. Merge the resulting subsequences into an increasing array:
$$a_{\pi(1)} < a_{\pi(2)} < \cdots < a_{\pi(n)}.$$

Both these steps can be performed in time $O(n \log M)$ and using Dilworth's Theorem [4, 6] it is not difficult to see that $M = m$, which implies the result.

In the sequel we first prove that $M = m$ and then we briefly explain how to efficiently perform each of the two steps of the algorithm.

Given a sequence $a_1, \ldots, a_n$, we introduce a partial ordering $\prec$ on the set $\{1, 2 \ldots, n\}$ in the following way:

$$i \prec j \iff i < j \text{ and } a_i < a_j.$$

With this notation it is obvious that the following are equivalent:

- $i(1) \prec i(2) \cdots \prec i(k)$;
- $(a_{i(1)}, a_{i(2)}, \ldots, a_{i(k)})$ is an increasing subsequence of $\{a_j\}_{j=1}^n$.

Thus each chain in $(\{1, \ldots n\}, \prec)$ corresponds to an increasing subsequence in $a_1, \ldots a_n$ and vice versa.

On the other hand, there is a similar relationship between the antichains in $(\{1, \ldots, n\}, \prec)$ and the decreasing subsequences of $a_1, \ldots a_n$. Specifically, we consider an antichain $\{i(1), \ldots i(k)\}$, i.e. $i(j)$, $i(l)$ are incomparable with respect to $\prec$. We can assume that $i(1) < i(2) < \cdots < i(k)$. Now consider a pair $i(j) < i(l)$: since $i(j) \not\prec i(l)$, we deduce that $a_{i(j)} \not< a_{i(l)}$. Furthermore, $i(j) \neq i(l)$

and the members of the sequence a are all distinct numbers, which implies that $a_{i(j)} > a_{i(l)}$. Thus we have established that for every antichain $\{i(1), \dots i(k)\}$ such that $i(1) < i(2) < \cdots < i(k)$,

$$a_{i(1)} > a_{i(2)} > \cdots > a_{i(k)} \text{ is a decreasing subsequence of } \{a_j\}_{j=1}^n.$$

Conversely, if $a_{i(1)} > a_{i(2)} > \cdots > a_{i(k)}$ is a decreasing subsequence, then $i(j) < i(l)$ implies $a_{i(j)} > a_{i(l)}$, i.e. $i(j)$ and $i(l)$ are incomparable with respect to $\prec$ and consequently determine an antichain.

Now the Dilworth's Theorem [4, 6] implies the following lemma:

Lemma 1. *Let m be the maximal length of a decreasing subsequence of $a_1, \dots, a_n$ and let M be the minimal number of increasing subsequences of $a_1, \dots, a_n$ in which $a_1, \dots, a_n$ can be partitioned. Then $M = m$.*

Proof. By the discussion above, m is the size of a maximal antichain in $(\{1, 2, \dots, n\}, \prec)$ and M is the minimum covering of $(\{1, 2, \dots, n\}, \prec)$ with $\prec$-chains. Therefore, since $\prec$ is a partial ordering, Dilworth's Theorem [4, 6] implies $m = M$. $\square$

Next we briefly describe the first part of the algorithm – determining the least number of increasing subsequences that cover $a_1, \dots, a_n$. We basically follow the ideas presented in [5, 3]. The algorithm processes the elements a_i in increasing order of i. At each step i we keep a set of lists $L_1, \dots, L_{m_i}$, such that $L_1, \dots L_{m_i}$ form a minimum $\prec$-chain covering of the set $\{1, 2, \dots, i\}$ and additionally for each $k \in L_{j+1}$ we keep a witness $w(k) \in L_j$ such that

$$w(k) < k \text{ and } w(k) \not\prec k,$$

which is equivalent to $w(k) < k$ and $a_{w(k)} > a_k$. Moreover, we maintain an array of the last elements $l[s] \in L_s$. Note that $a_{l[s+1]} < a_{w(l[s+1])} \leq a_{l(s)}$. The first inequality follows by the definition of the witnesses and the second follows by the fact that $w(l[s+1]) \preceq l[s]$ according to the definition of $l[s]$.

Now we describe how to maintain these invariants from step i to step $i + 1$.

1. Find the least s, such that $a_{l[s]} < a_{i+1}$.
2. If such an s does not exist, set $s = m_i + 1$, create a new list L_{m_i+1} and set $m_{i+1} = m_i + 1$, otherwise set $m_{i+1} = m_i$.
3. Insert $i + 1$ into L_s and set $l[s] = i + 1$.
4. If $s > 1$ set $w(i + 1) = l[s - 1]$.

Note that $i+1 > j$ for each $j \in \cup_{k=1}^{m_i} L_k$. Therefore $i+1 > l[s]$, and since $a_{l[s]} < a_{i+1}$, we obtain that $l[s] \prec i+1$. However, $l[s]$ is the maximal element of the list L_s, which implies that $L_s \cup \{i + 1\}$ is again a chain with maximal element $i + 1$. Next note that if $s > 1$, the choice of s implies that $a_{l[s-1]} > a_{i+1}$. Since $i+1 > l[s-1]$, we can safely define the witness of $i+1$ as $w(i+1) = l[s-1]$, as it is done in step 4. Finally, we argue that L_j is again a minimum covering of $\{1, 2, \dots, i+1\}$ with $\prec$-chains. This is clear in the case $m_{i+1} = m_i$, i.e. if $s \leq m_i$. Assume that $s = m_i + 1$, then we can consider the sequence $\{w^k(i+1) \,|\, 0 \leq k \leq m_i\}$. Since $i+1 \in L_{m_i+1}$, the definition of the witness implies that $w^k(i+1) \in L_{m_i+1-k}$. Moreover, we have that $w^k(i+1) > w^{k+1}(i+1)$ and $a_{w^k(i+1)} < a_{w^{k+1}(i+1)}$. Therefore, the set $\{w^k(i+1) \,|\, 0 \leq k \leq m_i\}$ is an anti-chain of size $m_i + 1$ in $(\{1, 2, \dots, i+1\}, \prec)$. Now by Dilworth's Theorem [4, 6] each covering with chains of $\{1, \dots i+1\}$ contains at least $m_i + 1$ elements, and therefore $m_{i+1} = m_i + 1$.

This shows that the above algorithm determines a minimum covering with increasing subsequences. Next we prove the main result of this section:

Theorem 1. *There is an $O(n \log m)$-time algorithm that sorts an arbitrary sequence of distinct numbers $a_1, \dots, a_n$ which contains no decreasing subsequence of length more than m.*

Proof. From the discussion above we know that the above algorithm provides a minimum covering with increasing subsequences. Now we consider its efficiency. Each of the steps 2, 3 and 4 can be performed in $O(1)$ time and step 1 can be performed in $O(\log m_i)$-time by binary searching the array $l[s]$ (recall that $a_{l[s]} > a_{l[s+1]}$). Since $m_i \leq m$ and we have n iterations in total, we obtain $O(n \log m)$-time algorithm to compute an optimal covering of $a_1, \dots a_n$ with increasing subsequences.

Now, since $L_1, \dots L_m$ are sorted in increasing order, we can easily merge them in $O(n \log m)$-time. One way to achieve this is to group the lists in pairs and merge the lists in every single pair. Each such step needs $O(n)$ time and reduces the number of lists twice. Thus in $O(\log m)$ iterations we end up with a single sorted list. Since we spend $O(n)$ time per iteration, the time bound follows.

Another possibility is to maintain a binary heap with up to m elements, each element corresponding to the least element of a list L_s which is still not sorted. At each step we extract the minimal element e from the heap and add it to the sorted output list (at the back). Next, if $e \in L_s$, we insert in the heap the next element of L_s. Clearly, we have $O(n)$ operations insert and extract minimal element from a heap with $O(m)$ elements. Therefore, each such operation can be performed in $O(\log m)$-time and the total time complexity results in $O(n \log m)$. $\square$

3. OPTIMALITY

In this section we show that each algorithm which sorts correctly in increasing order a sequence of distinct numbers $a_1, \dots, a_n$ needs to perform $\Theta(n \log m)$ comparisons where m is the length of the longest decreasing subsequence of $a_1, \dots, a_n$.

This would imply that the algorithm we described in the preceding section is worst-case optimal. The approach we use is similar to that in [5].

To this end we first show that there are $e^{\Theta(n \log m)}$ permutations $a_{\pi(1)}, \ldots, a_{\pi(n)}$ which contain no decreasing subsequence of length more than m.

Lemma 2. *Let $a_1 < a_2 < \cdots < a_n$ be distinct numbers and let $\Pi(m)$ be the set of permutations $\pi \in S_n$ such that $a_{\pi(1)}, \ldots, a_{\pi(n)}$ contains no decreasing subsequence of length greater than m. Then*

$$|\Pi(m)| \geq \frac{m^n}{m!}\,.$$

Proof. We count the permutations $\pi \in S_n$ with the property that there exist integers m' and $k_1, \ldots k_{m'}$, $k_{m'+1}$ such that:

$$m' \leq m \text{ and } 1 = k_1 < k_2 < \ldots k_{m'} < k_{m'+1} = n+1$$
$$\forall j (a_{\pi(k_j)} < a_{\pi(k_j+1)} < \cdots < a_{\pi(k_{j+1}-1)})$$
$$\forall i \leq m'(a_{\pi(k_i)} > a_{\pi(k_{i+1})}).$$

In fact, $\{k_j, k_j+1, \ldots, k_{j+1}-1\}_{j=1}^{m'}$ define m' chains in $(\{1, 2, \ldots n\}, \prec)$,where $\prec$ is defined with respect to the sequence $a_{\pi(1)}, \ldots, a_{\pi(n)}$. Consequently, by the discussion in the previous section, there is no decreasing subsequence of length more than m' in $a_{\pi(1)}, \ldots, a_{\pi(n)}$. On the other hand, the elements $a_{\pi(k_j)}$ witness for such a decreasing sequence. Therefore, each such permutation π belongs to the set $\Pi(m)$.

All such permutations π can be generated in the following way:

- assign each element $i \in \{1, 2, \ldots, n\}$ to exactly one of m sets B_j for $j \leq m$.
- discard all empty sets B_j.
- sort each $B_j \neq \emptyset$ in increasing order. In this fashion for each set B_j we obtain an increasing sequence b_j.
- arrange the sequences b_j's in decreasing order of their first elements. In this way we obtain the sequence $\pi(1), \ldots, \pi(n)$.

Clearly, each permutation obtained in this way can be uniquely decomposed into the increasing sequences b_j's which witness that $\pi \in \Pi(m)$. Next observe that different families of sets $\{B_1, \ldots, B_m\}$ and $\{B'_1, \ldots B'_m\}$ determine different permutations π and π'. Indeed, if it were the case that $\pi = \pi'$, then these permutations would determine the same sequence of increasing sequences $b_1 = b'_1$, $b_2 = b'_2, \ldots$, $b_{m'} = b'_{m'}$. Since each sequence b_j uniquely determines the set B_j, we conclude that $B_j = B'_j$ and since $\{B_1, \ldots, B_m\}$ and $\{B'_1, \ldots, B'_m\}$ define a partition of $\{1, 2, \ldots, n\}$, we obtain that $\{B_1, \ldots, B_m\} = \{B'_1, \ldots, B'_m\}$.

Therefore, it suffices to bound from below the number of all different families $\{B_1, \dots, B_m\}$. It is easy to count that the assignment in the first part of the construction can be done in m^n different ways. Since each family $\{B_1, \dots, B_m\}$ can be generated by at most $m!$ permutations of the sets B_j, we obtain that the number of all different families $\{B_1, \dots, B_m\}$ is at least $\frac{m^n}{m!}$ and therefore there are at least $\frac{m^n}{m!}$ permutations such that $a_{\pi(1)} \dots, a_{\pi(n)}$ contains no decreasing subsequence of length more than m. Therefore $|\Pi(m)| \geq \frac{m^n}{m!}$. □

Corollary 1. *The number of permutations of a sequence $a_1, \dots, a_n$ of distinct numbers that contain no decreasing subsequence of length more than m is $e^{\Omega(n \log m)}$.*

Proof. We consider first the case $m \leq \frac{n}{2}$. According to Lemma 2, the number of permutations $\Pi(m)$ that contain no decreasing subsequence of length more than m is

$$|\Pi(m)| \geq \frac{m^n}{m!}.$$

By Stirling's formula, $m! = \sqrt{2\pi m}\, m^m\, e^{-m+o(1)}$. Hence, $m! = e^{m(\log m + O(1))}$. Therefore,

$$|\Pi(m)| \geq \frac{m^n}{e^{m(\log m + O(1))}} \geq e^{(n-m)\log m + O(m)} \geq e^{\frac{n}{2}\log m + O(m)} = e^{\Theta(n \log m)},$$

since $m \leq \frac{n}{2}$.

In the case $m > \frac{n}{2}$ we have $\log \frac{n}{2} \leq \log m \leq \log \frac{n}{2} + 1$. Now we use that

$$\Pi\left(\frac{n}{2}\right) \subseteq \Pi(m).$$

By the discussion above we obtain that

$$\left|\Pi\left(\frac{n}{2}\right)\right| = e^{\Omega(n \log \frac{n}{2})},$$

and since $\Omega(n \log \frac{n}{2}) = \Omega(n \log m)$ for $\frac{n}{2} \leq m \leq n$,it is easy to see that

$$|\Pi(m)| \geq \left|\Pi\left(\frac{n}{2}\right)\right| = e^{\Omega(n \log m)},$$

and the result follows in this case either. □

Corollary 2. *Each algorithm which correctly sorts in increasing order each sequence $a_1, \dots, a_n$ of distinct numbers which contains no decreasing subsequence of length more than m, has worst-case time-complexity $\Omega(n \log m)$.*

Proof. By Corollary 1, there are $e^{\Omega(n \log m)}$ different permutations of $a_1, \dots, a_n$ that the algorithm has to be capable to distinguish. Now, if the algorithm performs $o(n \log m)$ comparisons on each such instance, we can assign each such permutation to a leaf of a binary decision tree of height $o(n \log m)$. However, each such tree has $e^{o(n \log m)}$ leaves and therefore two different permutations will be assigned to the same leaf of the tree. Consequently, the algorithm will be unable to distinguish between them. $\square$

As a corollary we obtain the following result:

Theorem 2. *The $O(n \log m)$-time sorting algorithm described in Section 2 is worst-case optimal.*

Proof. By Theorem 1 we have the correctness and the $O(n \log m)$ bound for the algorithm. On the other hand, Corollary 2 implies that any other algorithm that solves this problem is worst-case $\Omega(n \log m)$. $\square$

4. CONCLUSION

We have studied the problem of sorting a sequence of n distinct numbers with respect to the size m of the longest decreasing subsequence that it contains. We described an $O(n \log m)$-time algorithm that solves this problem without any assumptions on m and we showed that this time-complexity is worst-case optimal even under the assumption that an upper bound for m is known in advance.

ACKNOWLEDGEMENTS. The author is grateful to Professor Tinko Tinchev for encouraging him to present formally his observation. Thanks are due to the anonymous referee, who pointed to some previous works on patience sorting.

The research on this problem was supported by Contract 136/2010 of Sofia University and the project European Structural Fonds for Human Resources 2007-2013 BG051PO001-3.3.04/27.

5. REFERENCES

1. Aldous, D., P. Diaconis: Longest increasing subsequences: from patience sorting to the baik-deift-johansson theorem. *Bull. Amer. Math. Soc.*, **36**, no. 4, 1999, 413–432.
2. Burstein, A., I. Lankham: Combinatorics of patience sorting piles. http://arXiv:math/0506358, 2005.
3. Colrmen, T. H., C. E. Leiserson, R. L. Rivest: Introduction to Algorithms. Cambridge, MA: MIT Press, 5th edition, 1991.
4. Dilworth, R. P.: A decomposition theorem for partially ordered sets. *Ann. of Math*, **51**, 1950, 161–166.
5. Fredman, M. L.: On computing the length of the longest increasing subsequence. *Discrete Math.*, **11**, 1975, 29–35.

6. Gessel, I., G. C. Rota : Classic papers in combinatorics. Birkhäuser Boston, Inc. Boston, MA, 1987.
7. Knuth, D. E.: The Art of Computer Programming, Vol. 3, Sorting and Searching. Addison-Wesley, 1975.
8. Mallows, C. L.: Problem 62-2, patience sorting. *SIAM Review*, **5**, 1963, 375–376.
9. Mallows, C. L.: Patience sorting. *Bull. Inst. Math. Appl.*, **9**, 1973, 216–224.

Received on December 28, 2010
In revised form on July 17, 2014

Stefan Gerdjikov
Faculty of Mathematics and Informatics
"St. Kl. Ohridski" University of Sofia
5, J. Bourchier blvd., BG-1164 Sofia
BULGARIA
e-mail: st_gerdjikov@abv.bg