Chapter 1
Comparison-Sorting and Selecting in Totally Monotone Matrices

Noga Alon * Yossi Azar †

Abstract

An $m \times n$ matrix A is called totally monotone if for all $i_1 < i_2$ and $j_1 < j_2$, $A[i_1, j_1] > A[i_1, j_2]$ implies $A[i_2, j_1] > A[i_2, j_2]$. We consider the complexity of comparison-based selection and sorting algorithms in such matrices. Although our selection algorithm counts only comparisons its advantage on all previous work is that it can also handle selection of elements of different (and arbitrary) ranks in different rows (or even selection of elements of several ranks in each row), in time which is slightly better than that of the best known algorithm for selecting elements of the *same* rank in each row. We also determine the decision tree complexity of sorting each row of a totally monotone matrix up to a factor of at most $\log n$ by proving a quadratic lower bound and by slightly improving the upper bound. No nontrivial lower bound was previously known for this problem. In particular for the case $m = n$ we prove a tight $\Omega(n^2)$ lower bound. This bound holds for any decision-tree algorithm, and not only for a comparison-based algorithm. The lower bound is proved by an exact characterization of the bitonic totally monotone matrices, whereas our new algorithms depend on techniques from parallel comparison algorithms.

1 Introduction.

1.1 Background and previous work. Let $A = A[i, j]$ be an $m \times n$ matrix. A is called totally monotone if for all $i_1 < i_2$ and $j_1 < j_2$, $A[i_1, j_1] > A[i_1, j_2]$ implies $A[i_2, j_1] > A[i_2, j_2]$. Totally monotone matrices were introduced by Aggarwal, Klawe, Moran, Shor and Wilber [4]. These matrices arise naturally in the study of various problems in computational geometry, in the analysis of certain dynamic programming algorithms, and in other combinatorial problems related to VLSI circuit design. A wide variety of applications that use totally monotone matrices can be found in [4], [5], [9], [10] and their references.

In most of the applications the problems are reduced to a selection or sorting problem in each row in an appropriate totally monotone matrix. The basic problem considered was row maxima (or row minima), i.e., the problem of finding the maximum (or minimum) element in each row. An optimal algorithm for this problem was given in [4]. This algorithm, usually referred to as the SMAWK algorithm (see, e.g., [12]), runs in $\Theta(n)$ steps for $n \geq m$ and in $\Theta(n \log(2m/n))$ steps for $n < m$. This improves significantly the obvious $\Theta(nm)$-time algorithm that solves the row maxima problem for general matrices, and has been used in many applications.

The next natural problem considered was selecting the k'th element in each row. Kravets and Park [9] gave an algorithm which runs in $O(k(m+n))$ steps. Mansour, Park, Schieber and Sen [10] designed an algorithm which runs in time $O(m^{1/2} n \log n \log m + m \log n)$ for any k and thus yields a better complexity for the case of general k, and in particular for that of selecting the median in each row. For the typical case $m = n$ the first algorithm is better when $k \leq n^{1/2}(\log n)^2$ and the second is better for all the remaining range. Note that both algorithms require that k will be the same for all the rows, thus making a rather restrictive assumption.

Kravets and Park [9] also considered the problem of row-sorting, i.e., the problem of sorting each row in a totally monotone matrix. They designed an algorithm which runs in time $O(mn + n^2)$, improving the complexity of the trivial algorithm that sorts each row independently for the range $n = O(m \log m)$. They raised the open problem of improving their algorithm or establishing a lower bound for this problem. Note, for example, that for the case $m = n$ the SMAWK algorithm reduces the time for finding the row maxima from quadratic to linear by utilizing the special structure of a totally monotone matrix, whereas for sorting the time remains quadratic (the algorithm of [9] saves only a logarithmic factor). Observe that the fact that the size of the output of a row-sorting algorithm is $\Omega(nm)$ does not necessar-

*Department of Mathematics, Raymond and Beverly Sackler, Faculty of Exact Sciences, Tel Aviv University, Tel-Aviv, Israel, and Bellcore, Morristown, NJ, 07960, USA. Supported in part by a U.S.A.- Israeli BSF Grant and by a Bergmann Memorial Grant

†DEC Systems Research Center, 130 Lytton Ave. Palo-Alto, CA 94301. A portion of this work was done while the author was in the department of Computer Science, Stanford University, CA 94305-2140, and supported by a Weizmann fellowship and contract ONR N00014-88-K-0166.

ily provide a lower bound on the time required to sort the rows, since the output can possibly have a small representation based on the fact that totally monotone matrices are structured. Thus, it seems interesting to either improve significantly the time for row-sorting or to prove that this is impossible.

Parallel algorithms for finding the row maxima were also considered. The authors of [5] gave an algorithm that runs on a CREW PRAM in $O(\log n \log\log n)$ steps using $n/\log\log n$ processors (for the case $m = n$). A better algorithm, which runs on an EREW PRAM in $O(\log n)$ steps with n processors is given in [6].

1.2 Our results. In the present paper we consider two main problems for totally monotone matrices. The first is selecting elements of desired ranks in the rows (where the ranks may differ, and we may look for various different ranks in some rows) and the second is row-sorting.

We consider both problems mainly in the comparison model. Recall that in this model the complexity of an algorithm is determined by the number of comparisons performed, and the other steps in the computation are given for free. Such a model is realistic when the comparisons cost more than the rest of the computation. It is also interesting in the study of lower bounds. Although our algorithms are sequential, some of the techniques are based on ideas that arise in parallel comparison algorithms, and mainly these that appear in the study of approximation problems.

For the selection problem (in the special case of one required element in a row) we assume that a sequence of ranks r_i, $i = 1, \ldots m$ is given and we should find for any i the element of rank r_i in row i. We design a comparison algorithm that performs $O(nm^{1/2} \log n (\log m)^{1/2})$ comparisons for any given sequence r_i. Note that the complexity of our algorithm is slightly better than the complexity of the algorithm of [10] which is the best known algorithm for selecting elements with the same rank in each row. In fact, the improvement is more significant for m which is much bigger than n. Moreover, our algorithm has the advantage of being able to deal with distinct ranks (although it also has the disadvantage of being a comparison algorithm, i.e., only comparisons are counted in its complexity).

Our comparison algorithm can be easily parallelized to run in $O(\log m + \log\log n)$ rounds with no penalty in the number of processors, i.e., with a number of processors whose product with the above time is equal, up to a constant factor, to the above mentioned total sequential running time.

The sorting problem we consider is row-sorting, i.e., the problem of sorting each row of a given totally monotone m by n matrix. We prove tight lower and upper bounds which determine the complexity of this sorting problem up to a factor of at most $\log n$ in all cases. This settles an open problem raised in [9]. In particular, for the interesting special case $m = n$ we prove a tight lower bound of $\Omega(n^2)$.

Here is a summary of the complexity of the row-sorting problem. For $m \leq n/logn$ the best known algorithm was to sort each row independently in total running time $O(mn \log n)$. For $m > n/\log n$ the best known upper bound was $O(mn + n^2)$ as shown by [9]. No nontrivial lower bounds were known.

We first observe that one can easily design a comparison row-sorting algorithm that runs in $O(n^2 \log m)$ time, which is much smaller than the above mentioned bound when m is much bigger than n. By applying similar methods to these used in our selection algorithm we are also able to (slightly) improve the complexity for the row-sorting problem when $m \leq n$ and $n/m = 2^{o(\log n)}$. In particular, if $m = n/(\log n)^{O(1)}$ our improved algorithm replaces a logarithmic factor by a double logarithmic one. However, our main result for row-sorting is an almost tight lower bound in a general decision tree model. Specifically, we show that any algorithm that sorts each row of a totally monotone matrix requires $\Omega(\min(mn, n^2 \log(2 + m/n^2)))$ steps. In particular, for $m = n$ the $\Omega(n^2)$ bound is tight. For $m > n^{2+\epsilon}$ the $\Omega(n^2 \log n)$ bound is also tight. For all the remaining range the lower bound is of the same order of magnitude as the upper bound up to a factor of at most $\log n$. Recall that in the decision tree the algorithm is allowed at each step to branch into two possibilities according to any computation on the input (and not merely comparisons), and hence this lower bound is valid in a very general setting. We believe that similar techniques may be useful in establishing lower bounds for other sorting and searching problems dealing with totally monotone matrices.

We complete this section by the proof of the easy observation mentioned above.

Observation 1.1. *Sorting each row in a totally monotone m by n matrix can be done in $O(n^2 \log m)$ comparisons.*

Proof. Consider any two columns j_1, j_2, ($j_1 < j_2$). Let l be the minimum i for which $A[i, j_1] > A[i, j_2]$ (if such a row does not exits then define $l = m+1$). By the definition of the minimum and the definition of a totally monotone matrix, for all $i < l$, $A[i, j_1] < A[i, j_2]$ and if $i \geq l$, $A[i, j_1] > A[i, j_2]$. Hence, by a straightforward binary search one can find, for any specific pair of columns, this breakpoint l in $O(\log m)$ steps. Thus, the breakpoints for all pairs can be found in $O(n^2 \log m)$ steps. It is easy to see that the knowledge of this information yields the exact order for each row.

2 Selection.

The main result in this section is the following theorem;

THEOREM 2.1. *Let A be an m by n totally monotone matrix, and let*

$$S = \{(i_1, r_1), (i_2, r_2), \ldots, (i_s, r_s)\}$$

be a set of pairs, where $1 \le i_j \le m$ and $1 \le r_j \le n$ for all j. There exists a comparison algorithm that finds, using $T(n) = O(ns^{1/2} \log n (\log m)^{1/2})$ comparisons, the element whose rank in row number i_j is r_j, for all $1 \le j \le s$.

In particular, when $s = m$ and all the numbers i_j are distinct this is a comparison algorithm that finds, using $O(nm^{1/2} \log n (\log m)^{1/2})$ comparisons, an element of a desired rank in each row.

The proof is based on a combination of some of the techniques which have been used in parallel comparison algorithms with the reasoning in the proof of the easy Observation 1.1. The main part of the algorithm is based on comparisons performed according to the edges of appropriately chosen random graphs. These can be replaced by explicit expanders, with some increase in the total number of comparisons performed. We describe here only the version based on random graphs.

We need the following two lemmas, first proved in [3], which have been applied to various parallel comparison algorithms in [11] and in [2] as well.

LEMMA 2.1. *For every $n \ge a \ge 1$ there exists a graph $G(n, a)$ with n vertices and at most $\frac{2n^2 \log n}{a}$ edges in which any two disjoint sets of $a + 1$ vertices each are joined by an edge.*

LEMMA 2.2. *Let $G = G(n, a)$ be a graph as in Lemma 2.1, and suppose n elements are compared according to the edges of G, i.e., we associate each element with a vertex of G and compare a pair of elements iff the corresponding vertices are adjacent in G. Then, for every possible result of the comparisons, for every rank all but at most $7a \log n$ from the elements with a smaller rank will be known to be too small to have that rank. A symmetric statement holds for the elements with a bigger rank.*

Proof of Theorem 2.1 (sketch) Given a totally monotone m by n matrix A and a set S of pairs as in the theorem, let $G = G(n, a)$ be a graph as in Lemma 2.1, where a is a parameter to be chosen later. Let $\{1, 2, \ldots n\}$ be the set of vertices of G. For each edge $\{j_1, j_2\}$ of G (where $j_1 < j_2$) find, by a binary search using $\lceil \log m \rceil$ comparisons, the minimum i such that $A[i, j_1] > A[i, j_2]$. Altogether this costs $O(\frac{n^2 \log n \log m}{a})$ comparisons, after which we know the results of comparing the elements in each row of the matrix A according to the edges of G.

Claim: In each row separately, for each rank r, $1 \le r \le n$, one can find the element of rank r in the row by performing at most $O(a \log n)$ additional comparisons.

Let S be the set of elements whose rank is not known to be smaller than r nor larger than r. Denote by l the number of elements whose rank is known to be smaller than r. Clearly, the element of rank r is exactly the element of rank $r - l$ in S. Since Lemma 2.2 implies that $|S| \le 15a \log n$, we conclude that $O(a \log n)$ additional comparisons suffice to select that element. This completes the proof of the claim.

Returning to the proof of the theorem, we conclude that for each value of a it is possible to find all the required s elements by performing at most $T(n)$ comparison where

$$T(n) = O(\frac{n^2 \log n \log m}{a}) + O(sa \log n) \ .$$

In the trivial case $s \ge n^2 \log m$ we can sort, by Observation 1.1, all the rows of A in time

$$T(n) = O(n^2 \log m) \le O(ns^{1/2} \log n (\log m)^{1/2})$$

as needed. Otherwise, take

$$a = \lfloor \frac{n\sqrt{\log m}}{\sqrt{s}} \rfloor$$

and conclude that

$$T(n) = O(ns^{1/2} \log n (\log m)^{1/2}) \ .$$

This completes the proof. □

Remarks

1). Using a similar reasoning we can show that one can sort all the rows in a totally monotone m by n matrix using

$$O\left(\frac{n^2}{a} \log n \log m + mn \log(a \log n)\right)$$

comparisons, for each choice of a, $n \ge a \ge 1$. This slightly improves the trivial $O(nm \log n)$ upper bound (obtained by sorting each row separately) for values of $m \le n$ which are quite close to n. For example, for $m = n/(\log n)^{O(1)}$ this gives an algorithm that sorts the rows using $O(mn \log \log n)$ comparisons. we omit the details but mention that as shown in the next section this is tight up to the $\log \log n$ factor (even for general decision-tree algorithms).

2). The argument used in the first part of the proof, together with some of the known results on *almost sorting* algorithms ([3], [1], [8]) implies that by using only

$O(n \log n \log\log n \log m)$ comparisons one can know *almost all* the order relations between pairs of elements sharing the same row of an m by n totally monotone matrix.
We can also get an approximation algorithm for the row selection problem, i.e., find, for each pair of row and rank in the input, an element in that row whose rank is "close" to the desired rank. This yields a trade-off between the number of comparisons performed to the quality of the approximation, (which stands for the meaning of "close").
3). The algorithm can be easily parallelized. The first step can be easily done in $O(\log m)$ rounds with no penalty in the number of processors. The rest of the algorithm can be done in $O(\log\log n)$ rounds, again, with no loss in the total number of operations performed, by using the parallel selection algorithm of [7] which is based on the one of [3].

3 Comparison lower bound for row-sorting.

In this section we prove the lower bound for the problem of sorting all the rows in totally monotone matrices.

THEOREM 3.1. *Any decision tree algorithm that sorts each row in a totally monotone m by n matrix requires $\Omega(\min(mn, n^2 \log(2+m/n^2)))$ steps. In particular, for $m \le n$ the lower bound is $\Omega(mn)$, for $m \ge n$ it is $\Omega(n^2)$, and for $m \ge n^{2+\epsilon}$ it is $\Omega(n^2 \log m)$.*

Proof. We construct a large family F of $m \times n$ totally monotone matrices such that any two matrices in the family differ in the order of at least one corresponding row. Thus, any algorithm that sorts each row in the matrix should have a different output on each matrix in the family. Hence, a lower bound for any decision tree algorithm is the logarithm of the size of the family.

We start with a characterization of the bitonic totally monotone matrices, i.e., the totally monotone matrices in which each row is bitonic (=unimodal). This is done by associating with each such matrix a certain tableau $T[i,j]$ in a one-to-one manner. We note that similar tableaux are known as Young-tableaux and appear in the study of the Representations of the symmetric group, but here we are merely interested in some simple combinatorial properties of $T = T[i,j]$, described below.

1. Let l_i, $1 \le i \le m$ be integers such that $n-1 \ge l_1 \ge l_2 ... \ge l_m \ge 0$.
2. $T[i,j]$ is defined only for $i = 1,..,m,\ \ j = 1,..,l_i$.
3. For all $1 \le i \le m$ and $1 \le j \le l_i$, $T[i,j]$ is an integer and $1 \le T[i,j] \le n-1$.
4. Each row of T is a monotone increasing sequence. I.e., for all $1 \le i \le m$ and $1 \le j_1 < j_2 \le l_i$ we have $T[i,j_1] < T[i,j_2]$.
5. Each column of T is a monotone non-decreasing sequence. I.e., for all $1 \le i_1 < i_2 \le m$ and $j \le l_{j_2}$ $(\le l_{i_1})$, $T[i_1,j] \le T[i_2,j]$.

We next show how to map these tableaux into a family F of totally monotone matrices such that different tableaux would map to different order-type matrices. (Here, of course, the order-type of a matrix refers to the sequence of linear orders of its rows). To this end, construct the matrix A corresponding to the tableau T as follows. Each row of A starts with the elements in the corresponding row in the tableau T. Note that a row in T contains a subset S of the set $\{1,\ldots,n-1\}$. Put $N = \{1,...,n\}$. To complete the row we put to the right of S the number n (i.e. $A[i,l_i+1] = n$) and then the set $N - S$ where the numbers in the set appear in a decreasing order. We need the following Lemma;

LEMMA 3.1. *For any tableau T as above the construction yields a totally monotone m by n matrix. Moreover, different tableaux yield different order-type matrices.*

Proof. First note that the length of each row in the matrix is precisely n since any row consists of some permutation of the set N. Thus the construction yields, indeed, an $m \times n$ matrix. Let us show that the resulting matrix is totally monotone. It is clear that each row is bitonic. More precisely, the first $l_i + 1$ elements in row i form a monotone increasing sequence and the elements from $l_i + 1$ to the end of the row form a monotone decreasing sequence. It is also not too difficult to verify that our construction and property 5 imply that each column in the complement of the tableau is a monotone non-increasing sequence. Formally, for all $1 \le i_1 < i_2 \le m$ and $j > l_{i_1}$ $(\ge l_{i_2})$, $A[i_1,j] \ge A[i_2,j]$. This property is called property 5'.

Suppose $i_1 < i_2$ and $j_1 < j_2$. We have to show that if $A[i_1,j_1] > A[i_1,j_2]$ then $A[i_2,j_1] > A[i_2,j_2]$. We consider several possible cases. If $j_2 \le l_{i_1} + 1$ then clearly $A[i_1,j_1]$ and $A[i_1,j_2]$ are in the monotone increasing part of the row i_1 and thus $A[i_1,j_1] < A[i_1,j_2]$ and there is nothing to prove. If $j_1 \ge l_{i_2} + 1$ then clearly $A[i_2,j_1]$ and $A[i_2,j_2]$ are in the monotone decreasing part of the row i_2 and thus $A[i_2,j_1] > A[i_2,j_2]$ which also leaves nothing to prove. Thus, we can assume that

$$j_1 \le l_{i_2} \le l_{i_1} < l_{i_1} + 2 \le j_2 \ .$$

Therefore, we conclude that

$$A[i_2,j_1] \ge A[i_1,j_1] > A[i_1,j_2] \ge A[i_2,j_2]$$

where the first inequality uses property 5 of the tableau ($j_1 \leq l_{i_2}$), the middle inequality is the assumption and the last inequality takes advantage of property 5' (since $j_2 > l_{i_1}$). Hence, the matrix is totally monotone.

It is left to show that for any two different tableaux the construction yields matrices of different order types. Note that the value of each element in the tableau is in fact precisely its rank in its row. Let the sequence l_i, $i = 1, \ldots, m$ denote the *shape* of the tableau. Thus, if two different tableaux have the same shape then they must have a different value in some entry. Hence, they have different order types. On the other hand, if they differ in the shapes, then, there exists an i for which l_i in one matrix is, say, smaller than l'_i of the other. Then, the element in row i and column $l_i + 1$ has a different rank in its row in the two matrices since in the first one it has rank n, whereas in the second its rank is smaller than n. This completes the proof of the Lemma.

In fact, it is also true that any totally monotone matrix A in which each row is a bitonic sequence has the same order type as one of the matrices constructed in the above way from an appropriate tableau T. To see this, construct T as follows. Replace each element in A by its rank to obtain a matrix A'. Each row in T will be the part of the corresponding row of A' that starts from the leftmost entry in the matrix up to the element n (the maximum) excluding this element. It is easy to verify that this T has all the required properties 1 - 5 above. Since this is not essential for our results, we omit the details.

We continue the proof of Theorem 3.1 by establishing a lower bound on the size of the family F. First we observe that each row in the tableau T has at most 2^{n-1} possibilities (each row is a subset of $\{1, \ldots, n-1\}$). Since there are m such rows, the size of the family is at most 2^{mn}, so we cannot expect a better lower bound using bitonic matrices.

Consider first the case $m \leq n/2$. For this case we show that the number of such tableaux A is $2^{\Omega(nm)}$, and thus $\Omega(mn)$ steps are required in any decision-tree row-sorting algorithm. To this end we even restrict ourselves to a subfamily of the tableaux where $l_i = n/2 - i$ for $1 \leq i \leq n/2$. Moreover, we consider only the following subfamily; $T[i,j]$ will be either $2*(i+j-1)$ or $2*(i+j-1)-1$. One can easily check that any such assignment produces a legal tableau T, since all the elements are in the admissible range, each row is monotone increasing and each column is monotone increasing as well. Moreover, the number of such assignments is simply 2 to the power of the number of places in the tableau which is $2^{\Omega(mn)}$ and we are done.

For $2n^2 \geq m > n/2$ an $\Omega(n^2)$ lower bound follows from the bound for $m = n/2$.

We are left with the case $m > 2n^2$. In order to prove the $\Omega(n^2 \log(m/n^2))$ lower bound for this case we construct $k = \Theta(n^2)$ rows $R_1, ..., R_k$ with the properties described below, and consider the subfamily of all tableaux of the following type: R_1 appears x_1 times; below that R_2 appears x_2 times and so forth while $x_1 + \ldots + x_k = m$ and $x_i \geq 0$. The rows R_i will be defined in such a way that any such tableau T will be legal.

The number of tableaux that can be constructed in such a way is exactly the number of ways to partition m balls in k cells which is $\binom{m+k-1}{k-1}$. The lower bound follows from the fact that

$$\binom{m+k-1}{k-1} > ((m+k-1)/(k-1))^{k-1}$$
$$= 2^{\Omega(n^2 \log(m/n^2))} .$$

It is left to show how to define the rows $R_1, ..., R_k$. Let

$$k = 1 + (n-1) + (n-2) + ... + 1 = \binom{n}{2} + 1$$

The first row is the sequence $\{1, 2, ..., n-1\}$. Assume inductively that we have already constructed all the rows up to the row which consists of the sequence $\{i, i+1, \ldots, n-1\}$. Denote this row by S_i (and note that it is not the i'th row for $i > 1$). The next $n - i$ rows are constructed as follows. For $1 \leq j \leq n - i$ the j'th row below S_i is defined to be the row S_i without the number $n - j$. The last row in this group is exactly $S_{i+1} = \{i+1, \ldots, n-1\}$, hence we can continue inductively from S_{i+1}. Note that the last row (which is S_n) is an empty row, which is a legal row (by our definitions). It is not difficult to check that the tableau which consists of these k rows is legal, i.e, satisfies all the required properties. Moreover, omitting and duplicating rows do not affect the legality of the tableau. Thus, these rows can be used for the construction described above, completing the proof of the theorem.

4 Acknowledgement.

We would like to thank Don Coppersmith for simplifying the proof of Theorem 2.1 and Maria Klawe for helpful remarks.

References

[1] N. Alon and Y. Azar, *Sorting, approximate sorting and searching in rounds*, SIAM J. Discrete Math. 1 (1988) pp. 269–280.

[2] ———, *Parallel comparison algorithms for approximation problems*, Proc. 29th Annual IEEE Symp. on Foundations of Computer Science, White Plains, New

York, 1988, pp. 194–203. Also: Combinatorica, in press.

[3] M. Ajtai, J. Komlós, W.L. Steiger and E. Szemerédi, *Deterministic selection in* $O(\log\ \log\ n)$ *parallel time*, Proc. 18th Annual ACM Symp. on Theory of Computing, Berkeley, CA, 1986, pp. 188–195.

[4] A. Aggarwal, M. Klawe, S. Moran, P.Shor and R. Wilber, *Geometric applications of a matrix searching algorithm*, Algorithmica 2 (1987) pp. 195–208.

[5] A. Aggarwal and J. Park, *Notes on searching in multi-dimensional monotone arrays*, Proc. 29th Annual IEEE Symposium on Foundations of Computer Science 1988 pp. 497–512.

[6] M. Attallah and R. Kosaraju, *An efficient Parallel Algorithm for the row minima of totally monotone matrices*, Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 1991 pp. 394–403.

[7] Y. Azar and N. Pippenger, *Parallel selection*, Discrete Applied Math 27 (1990) pp. 49–58.

[8] B. Bollobás and G. Brightwell, *Graphs whose every transitive orientation contains almost every relation*, Israel J. Math., 59 (1987), 112–128.

[9] D. Kravets and J. Park, *Selection and sorting in totally monotone arrays*, Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 494–502.

[10] Y. Mansour, J. Park, B. Schieber and S. Sen, *Improved selection in totally monotone arrays*, May 1991.

[11] N. Pippenger, *Sorting and selecting in rounds*, SIAM J. Computing 6 (1987) pp. 1032–1038.

[12] R. Wilber, *The concave least-weight subsequence problem revisited*, Journal of Algorithms 9 (1988) pp. 418–425.