CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE 2012, Article 02, pages 1–10
http://cjtcs.cs.uchicago.edu/

Minimum and maximum against k lies*

Michael Hoffmann Jiří Matoušek† Yoshio Okamoto‡
Philipp Zumstein

Received: November 26, 2010; revised: June 20, 2012; published: July 16, 2012.

Abstract: A neat 1972 result of Pohl asserts that $\lceil 3n/2 \rceil - 2$ comparisons are sufficient, and also necessary in the worst case, for finding both the minimum and the maximum of an n-element totally ordered set. The set is accessed via an oracle for pairwise comparisons. More recently, the problem has been studied in the context of the Rényi–Ulam liar games, where the oracle may give up to k false answers. For large k, an upper bound due to Aigner shows that $(k+\mathrm{O}(\sqrt{k}))n$ comparisons suffice. We improve on this by providing an algorithm with at most $(k+1+C)n+\mathrm{O}(k^3)$ comparisons for some constant C. A recent result of Pálvölgyi provides a lower bound of the form $(k+1+0.5)n-D_k$, so our upper bound for the coefficient of n is tight up to the value of C.

Key words and phrases: computing the minimum and maximum, computation against lies, number of comparisons, liar games, computation in the presence of errors

1 Introduction

We consider an n-element set X with an unknown total ordering $\leq$. The ordering can be accessed via an oracle that, given two elements $x, y \in X$, tells us whether $x < y$ or $x > y$. It is easily seen that the minimum element of X can be found using $n-1$ comparisons. This is optimal in the sense that $n-2$ comparisons are not enough to find the minimum element in the worst case.

One of the nice little surprises in computer science is that if we want to find *both* the minimum and the maximum, we can do significantly *better* than finding the minimum and the maximum separately.

*A preliminary version appeared in *Proc. 12th Scandinavian Symposium and Workshops on Algorithm Theory*, Bergen (Lecture Notes in Computer Science 6139), Springer, 2010, pages 139–149.

†Supported by the ERC Advanced Grant No. 267165.

‡Supported by Global COE Program "Computationism as a Foundation for the Sciences" at Tokyo Institute of Technology, and Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

DOI: 10.486/cjtcs.2012.002

Pohl [9] proved that $\lceil 3n/2 \rceil - 2$ is the optimal number of comparisons for this problem ($n \geq 2$). The algorithm first partitions the elements of X into pairs and makes a comparison in each pair. The minimum can then be found among the "losers" of these comparisons, while the maximum is found among the "winners."[1]

Here we consider the problem of determining both the minimum and the maximum in the case where the oracle is not completely reliable: it may sometimes give a false answer, but only at most k times during the whole computation, where k is a given parameter.

We refer to this model as *computation against k lies*. Let us stress that we admit repeating the same query to the oracle several times, and each false answer counts as a lie. This seems to be the most sensible definition—if repeated queries were not allowed, or if the oracle could always give the wrong answer to a particular query, then the minimum cannot be determined.

So, for example, if we repeat a given query $2k+1$ times, we always get the correct answer by majority vote. Thus, we can simulate any algorithm with a reliable oracle, asking every question $2k+1$ times, but for the problems considered here, this is not a very efficient way, as we will see.

The problem of finding both the minimum and the maximum against k lies was investigated by Aigner [1], who proved that $(k+\mathrm{O}(\sqrt{k}))n$ comparisons always suffice.[2] We improve on this as follows.

Theorem 1.1. *There is an algorithm that finds both the minimum and the maximum among n elements against k lies using at most* $(k+1+C)n+\mathrm{O}(k^3)$ *comparisons, where C is a constant.*

Our proof yields the constant C reasonably small (below 10, say, at least if k is assumed to be sufficiently large), but we do not try to optimize it.

Lower bounds. The best known lower bounds for the number of comparisons necessary to determine both the minimum and the maximum against k lies have the form $(k+1+c_k)n - D_k$, where D_k depends only on k and the c_k are as follows.

- $c_0 = 0.5$, and this is the best possible. This is the result of Pohl [9] for a truthful oracle mentioned above.
- $c_1 = \frac{23}{32} = 0.71875$, and this is again tight. This follows from a recent work by Gerbner, Pálvölgyi, Patkós, and Wiener [4] who determined the optimum number of comparisons for $k = 1$ up to a small additive constant: it lies between $\lceil \frac{87}{32}n \rceil - 3$ and $\lceil \frac{87}{32}n \rceil + 12$. This proves a conjecture of Aigner [1].
- Aigner [1] proved $c_k = \Omega(2^{-5k/4})$ for all k. After a preliminary version of the present paper was published, Pálvölgyi [7] found an elegant simple argument showing that $c_k \geq 0.5$ for every k.

The optimal constant $c_1 = \frac{23}{32}$ indicates that obtaining precise answers for $k > 1$ may be difficult.

[1] Another optimal algorithm partitions the elements of X into two groups of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, recursively finds the minima and the maxima of both groups, and outputs the smaller of the minima and the larger of the maxima. The worst-case number $t(n)$ of comparisons satisfies $t(n) \leq t(\lceil n/2 \rceil) + t(\lfloor n/2 \rfloor) + 2$, which yields $t(n) \leq \lceil 3n/2 \rceil - 2$. The non-uniqueness of an optimal algorithm can be regarded as an indication that the problem is not too simple.

[2] Here and in the sequel, $\mathrm{O}(.)$ and $\Omega(.)$ hide only absolute constants, independent of both n and k.

Related work. The problem of determining the minimum alone against k lies was resolved by Ravikumar, Ganesan, and Lakshmanan [10], who proved that finding the minimum against k lies can be performed by using at most $(k+1)n-1$ comparisons, and this is optimal in the worst case.

The problem considered in this paper belongs to the area of *searching problems against lies* and, in a wider context, it is an example of "computation in the presence of errors." This field has a rich history and beautiful results. A prototype problem, still far from completely solved, is the *Rényi–Ulam liar game* from the 1960s, where one wants to determine an unknown integer x between 1 and n, an oracle provides comparisons of x with specified numbers, and it may give at most k false answers. We refer to the surveys by Pelc [8] and by Deppe [2] for more information.

2 A simple algorithm

Before proving Theorem 1.1, we explain a simpler algorithm, which illustrates the main ideas but yields a weaker bound. We begin with formulating a generic algorithm, with some steps left unspecified. Both the simple algorithm in this section and an improved algorithm in the next sections are instances of the generic algorithm.

The generic algorithm

1. For a suitable integer parameter $s = s(k)$, we arbitrarily partition the considered n-element set X into n/s groups $X_1, \ldots, X_{n/s}$ of size s each.[a]
2. In each group X_i, we find the minimum m_i and the maximum M_i. The method for doing this is left unspecified in the generic algorithm.
3. We find the minimum of $\{m_1, \ldots, m_{n/s}\}$ against k lies, and independently, we find the maximum of $\{M_1, M_2, \ldots, M_{n/s}\}$ against k lies.

[a] If n is not divisible by s, we can form an extra group smaller than s and treat it separately, say—we will not bore the reader with the details.

The correctness of the generic algorithm is clear, provided that Step 2 is implemented correctly. Eventually, we set $s := k$ in the simple and in the improved algorithm. However, we keep s as a separate parameter, because the choice $s := k$ is in a sense accidental.

In the simple algorithm we implement Step 2 as follows.

Step 2 in the simple algorithm

2.1. (Sorting.) We sort the elements of X_i by an asymptotically optimal sorting algorithm, say mergesort, using $O(s \log s)$ comparisons, and ignoring the possibility of lies. Thus, we obtain an ordering $x_1, x_2, \ldots, x_s$ of the elements of X_i such that *if* all queries during the sorting have been answered correctly, *then* $x_1 < x_2 < \cdots < x_s$. If there was at least one false answer, we make no assumptions, except that the sorting algorithm does not crash and outputs some ordering.

2.2. (Verifying the minimum and maximum.) For each $j = 2, 3, \ldots, s$, we query the oracle $k+1$ times with the pair x_{j-1}, x_j. If any of these queries returns the answer $x_{j-1} > x_j$, we *restart*: We go back to Step 2.1 and repeat the computation for the group X_i from scratch. Otherwise, if all the answers are $x_{j-1} < x_j$, we proceed with the next step.

2.3. We set $m_i := x_1$ and $M_i := x_s$.

Lemma 2.1 (Correctness)**.** *The simple algorithm always correctly computes the minimum and the maximum against k lies.*

Proof. We note that once the processing of the group X_i in the above algorithm reaches Step 2.3, then $m_i = x_1$ has to be the minimum. Indeed, for every other element x_j, $j \geq 2$, the oracle has answered $k+1$ times that $x_j > x_{j-1}$, and hence x_j cannot be the minimum. Similarly, M_i has to be the maximum, and thus the algorithm is always correct. □

Actually, at Step 2.3 we can be sure that $x_1, \ldots, x_s$ is the sorted order of X_i, but in the improved algorithm in the next section the situation will be more subtle. The next lemma shows, that the simple algorithm already provides an improvement of Aigner's bound of $(k + O(\sqrt{k}))n$.

Lemma 2.2 (Complexity)**.** *The number of comparisons of the simple algorithm for $s = k$ on an n-element set is $(k + O(\log k))n + O(k^3)$.*

Proof. For processing the group X_i in Step 2, we need $O(s \log s) + (k+1)(s-1) = k^2 + O(k \log k)$ comparisons, provided that no restart is required. But since restarts may occur only if the oracle lies at least once, and the total number of lies is at most k, there are no more than k restarts for all groups together. These restarts may account for at most $k(k^2 + O(k \log k)) = O(k^3)$ comparisons. Thus, the total number of comparisons in Step 2 is $\frac{n}{s}(k^2 + O(k \log k)) + O(k^3) = (k + O(\log k))n + O(k^3)$.

As we mentioned in the introduction, the minimum (or maximum) of an n-element set against k lies can be found using $(k+1)n - 1$ comparisons, and so Step 3 needs no more than $2(k+1)(n/s) = O(n)$ comparisons. (We do not really need the optimal algorithm for finding the minimum; any $O((k+1)n)$ algorithm would do.) The claimed bound on the total number of comparisons follows. □

3 The improved algorithm: Proof of Theorem 1.1

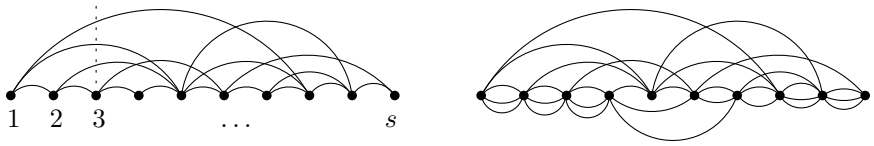
In order to certify that x_1 is indeed the minimum of X_i, we want that for every x_j, $j \neq 1$, the oracle declares x_j larger than some other element $k+1$ times. (In the simple algorithm, these $k+1$ comparisons were all made with x_{j-1}, but any other smaller elements will do.) This in itself requires $(k+1)(s-1)$ queries per group, or $(k+1)(n-n/s)$ in total, which is already close to our target upper bound in Theorem 1.1 (we note that s has to be at least of order k, for otherwise, Step 3 of the generic algorithm would be too costly).

Similarly, every x_j, $j \neq s$, should be compared with larger elements $k+1$ times, which again needs $(k+1)(n-n/s)$ comparisons, so all but $O(n)$ comparisons in the whole algorithm should better be used for *both* of these purposes.

In the simple algorithm, the comparisons used for sorting the groups in Step 2.1 are, in this sense, wasted. The remedy is to use most of them also for verifying the minimum and maximum in Step 2.2. For example, if the sorting algorithm has already made comparisons of x_{17} with 23 larger elements, in the verification step it suffices to compare x_{17} with $k+1-23$ larger elements.

One immediate problem with this appears if the sorting algorithm compares x_{17} with some $b > k+1$ larger elements, the extra $b-(k+1)$ comparisons are wasted. However, for us, this will not be an issue, because we will have $s = k$, and thus each element can be compared to at most $k-1$ others (assuming, as we may, that the sorting algorithm does not repeat any comparison).

Another problem is somewhat more subtle. In order to explain it, let us represent the comparisons made in the sorting algorithm by edges of an ordered graph. The vertices are $1,2,\ldots,s$, representing the elements $x_1,\ldots,x_s$ of X_i in sorted order, and the edges correspond to the comparisons made during the sorting, see the figure below on the left.



In the verification step, we need to make additional comparisons so that every x_j, $j \neq 1$, has at least $k+1$ comparisons with smaller elements and every x_j, $j \neq s$, has at least $k+1$ comparisons with larger elements. This corresponds to adding suitable extra edges in the graph, as in the right drawing above (where $k = 2$, and the added edges are drawn on the bottom side).

As the picture illustrates, sometimes we cannot avoid comparing some element with *more* than $k+1$ larger ones or $k+1$ smaller ones (and thus some of the comparisons will be "half-wasted"). For example, no matter how we add the extra edges, the elements x_1, x_2, x_3 together must participate in at least 3 half-wasted comparisons. Indeed, x_2 and x_3 together require 6 comparisons to the left (i.e. with a smaller element). These comparisons can be "provided" only by x_1 and x_2, which together want only 6 comparisons to the right—but 3 of these comparisons to the right were already made with elements larger than x_3 (these are the arcs intersecting the dotted vertical line in the picture).

The next lemma shows that this kind of argument is the only source of wasted comparisons. For an ordered multigraph H on the vertex set $\{1,2,\ldots,s\}$ as above, let us define $t(H)$, the *thickness* of H, as $\max\{t(j) : j = 2,3,\ldots,s-1\}$, where $t(j) := |\{\{a,b\} \in E(H) : a < j < b\}|$ is the number of edges going "over" the vertex j.

Lemma 3.1. *Let H be an undirected multigraph without loops on $\{1,2,\dots,s\}$ such that for every vertex $j = 1,2,\dots,s$,*

$$d_H^{\text{left}}(j) := |\{\{i,j\} \in E(H) : i < j\}| \le k+1\,,$$
$$d_H^{\text{right}}(j) := |\{\{i,j\} \in E(H) : i > j\}| \le k+1\,.$$

Then H can be extended to a multigraph $\overline{H}$ by adding edges, so that

(i) *every vertex $j \neq 1$ has at least $k+1$ left neighbors and every vertex $j \neq s$ has at least $k+1$ right neighbors; and*

(ii) *the total number of edges in $\overline{H}$ is at most $(k+1)(s-1)+t(H)$.*

The proof is a network flow argument and therefore constructive. We postpone it to the end of this section.

For a comparison-based sorting algorithm $\mathcal{A}$, we define the *thickness* $t_{\mathcal{A}}(s)$ in the natural way: It is the maximum, over all s-element input sequences, of the thickness $t(H)$ of the corresponding ordered graph H (the vertices of H are ordered as in the output of the algorithm and each comparison contributes to an edge between its corresponding vertices). As the above lemma shows, the number of comparisons used for the sorting but not for the verification can be bounded by the thickness of the sorting algorithm.

Lemma 3.2. *There exists a (deterministic) sorting algorithm $\mathcal{A}$ with thickness $t_{\mathcal{A}}(s) = \mathrm{O}(s)$.*

Proof. The algorithm is based on Quicksort, but in order to control the thickness, we want to partition the elements into two groups of equal size in each recursive step.

We thus begin with computing the median of the given elements. This can be done using $\mathrm{O}(s)$ comparisons (see, e.g., Knuth [6]; the current best deterministic algorithm due to Dor and Zwick [3] uses no more than $2.95s + \mathrm{o}(s)$ comparisons). These algorithms also divide the remaining elements into two groups, those smaller than the median and those larger than the median. To obtain a sorting algorithm, we simply recurse on each of these groups.

The thickness of this algorithm obeys the recursion $t_{\mathcal{A}}(s) \le \mathrm{O}(s) + t_{\mathcal{A}}(\lfloor s/2 \rfloor))$, and thus it is bounded by $\mathrm{O}(s)$. □

We are going to use the algorithm $\mathcal{A}$ from the lemma in the setting where some of the answers of the oracle may be wrong. Then the median selection algorithm is not guaranteed to partition the current set into two groups of the same size. However, we can check if the groups have the right size, and in case they don't, we restart the computation (similar to the simple algorithm).

Now we can describe the improved algorithm, again by specifying Step 2 of the generic algorithm.

Step 2 in the improved algorithm

2.1′. (Sorting.) We sort the elements of X_i by the algorithm $\mathcal{A}$ with thickness $O(s)$ as in Lemma 3.2. If an inconsistency is detected (as discussed above), we restart the computation for the group X_i from scratch.

2.2′. (Verifying the minimum and maximum.) We create the ordered graph H corresponding to the comparisons made by $\mathcal{A}$, and we extend it to a multigraph $\overline{H}$ according to Lemma 3.1. We perform the comparisons corresponding to the added edges. If we encounter an inconsistency, then we restart: We go back to Step 2.1′ and repeat the computation for the group X_i from scratch. Otherwise, we proceed with the next step.

2.3′. We set $m_i := x_1$ and $M_i := x_s$.

Proof of Theorem 1.1. The correctness of the improved algorithm follows in the same way as for the simple algorithm. In Step 2.2′, the oracle has declared every element x_j, $j \neq 1$, larger than some other element $k+1$ times, and so x_j cannot be the minimum. A similar argument applies for the maximum.

It remains to bound the number of comparisons. From the discussion above, the number of comparisons is at most $((k+1)(s-1)+t_{\mathcal{A}}(s))(\frac{n}{s}+k)+2(k+1)\frac{n}{s}$, with $t_{\mathcal{A}}(s)=O(s)$. For $s=k$, we thus get that the number of comparisons at most $(k+1+C)n+O(k^3)$ for some constant C, as claimed. □

Proof of Lemma 3.1. We will proceed in two steps. First, we construct a multigraph H^* from H by adding a maximum number of (multi)edges such that the left and right degree of every vertex are still bounded above by $k+1$. Second, we extend H^* to $\overline{H}$ by adding an appropriate number of edges to each vertex so that condition (i) holds.

For an ordered multigraph H' on $\{1,2,\ldots,s\}$ with left and right degrees upper bounded by $k+1$, let us define the *defect* $\Delta(H')$ as

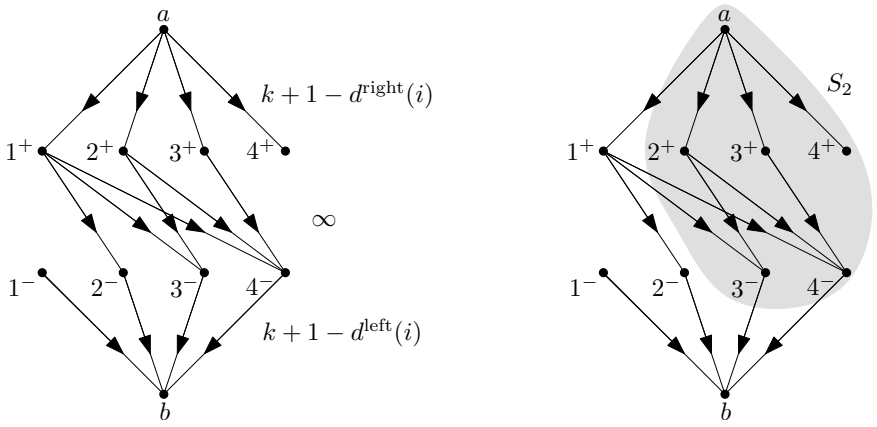
$$\Delta(H') := \sum_{j=1}^{s-1}(k+1-d^{\text{right}}_{H'}(j)) + \sum_{j=2}^{s}(k+1-d^{\text{left}}_{H'}(j))\,.$$

We have $\Delta(H') = 2(k+1)(s-1)-2e(H')$, where $e(H')$ is the number of edges of H'.

By a network flow argument, we will show that by adding suitable $m^* := (k+1)(s-1)-e(H)-t(H)$ edges to H, one can obtain a multigraph H^* in which all left and right degrees are still bounded by $k+1$ and such that $\Delta(H^*) = 2t(H)$. The desired graph $\overline{H}$ as in the lemma will then be obtained by adding $\Delta(H^*)$ more edges: For example, for every vertex $j \geq 2$ of H^* with $d^{\text{left}}_{H^*}(j) < k+1$, we add $k+1-d^{\text{left}}_{H^*}(j)$ edges connecting j to 1, and similarly we fix the deficient right degrees by adding edges going to the vertex s.

It remains to construct H^* as above. To this end, we define an auxiliary directed graph G, where each directed edge e is also assigned an integral capacity $c(e)$; see Figure 1(a).

The vertex set of G consists of a vertex j^- for every $j \in \{1,2,\ldots,s\}$, a vertex j^+ for every $j \in \{1,2,\ldots,s\}$, and two special vertices a and b. There is a directed edge in G from a to every vertex j^+



(a) The graph G with capacities. (b) The cut S_2 in G.

Figure 1: The directed graph G constructed in the proof of Lemma 3.1.

and the capacity of this edge is $k+1-d_H^{\text{right}}(j)$. Similarly, there is a directed edge in G from every vertex j^- to b, and the capacity of this edge is $k+1-d_H^{\text{left}}(j)$. Moreover, for every i,j with $1 \le i < j \le s$, we put the directed edge (i^+, j^-) in G, and the capacity of this edge is ∞ (i.e., a sufficiently large number).

We will check that there is an integral a-b flow in G with value m^* in G. By the max-flow min-cut theorem [5], it suffices to show that every a-b cut in G has capacity at least m^* and there is an a-b cut in G with capacity m^*.

Let $S \subseteq V(G)$ be a minimum a-b cut. Let i be the smallest integer such that $i^+ \in S$. Since the minimum cut cannot use an edge of unbounded capacity, we have $j^- \in S$ for all $j > i$.

We may assume without loss of generality that $j^+ \in S$ for all $j > i$ and $j^- \notin S$ for all $j \le i$ (the capacity of the cut does not decrease by doing otherwise). Therefore it suffices to consider a-b cuts of the form

$$S_i := \{a\} \cup \{x^+ : x \ge i\} \cup \{x^- : x > i\}$$

for $i = 1, \dots, s$. The capacity of S_i, see Figure 1(b), equals

$$\sum_{j<i} c(a, j^+) + \sum_{j>i} c(j^-, b) = (s-1)(k+1) - \sum_{j<i} d_H^{\text{right}}(j) - \sum_{j>i} d_H^{\text{left}}(j).$$

Now let us look at the quantity $\sum_{j<i} d^{\text{right}}(j) + \sum_{j>i} d^{\text{left}}(j)$, and see how much an edge $\{j, j'\}$ $(j < j')$ of H contributes to it: For $j < i < j'$, the contribution is 2, while all other edges contribute 1. Hence the capacity of the cut S_i is $(k+1)(s-1) - e(H) - t(i)$, and the minimum capacity of an a-b-cut is $(k+1)(s-1) - e(H) - t(H) = m^*$ as required.

Thus, there is an integral flow f with value m^* as announced above. We now select the edges to be added to H as follows: For every directed edge (i^+, j^-) of G, we add $f(i^+, j^-)$ copies of the edge $\{i, j\}$, which yields the multigraph H^*. The number of added edges is m^*, the value of the flow f, and the capacity constraints guarantee that all left and right degrees in H^* are bounded by $k+1$. Moreover, the defect of H^* is at most $2t(H)$. □

Acknowledgments

We thank Döm Pálvölgyi for bringing the problem investigated in this paper to our attention. This work was started at the 7th Gremo Workshop on Open Problems, Hof de Planis, Stels in Switzerland, July 6–10, 2009. We also thank the participants of the workshop for the inspiring atmosphere, and anonymous referees for numerous useful comments concerning the presentation.

References

[1] Martin Aigner. Finding the maximum and the minimum. *Discrete Appl. Math.*, 74(1):1–12, 1997. 2

[2] Christian Deppe. Coding with feedback and searching with lies. In Imre Csiszár, Gyula O. H. Katona, and Gábor Tardos, editors, *Entropy, Search, Complexity*, volume 16 of *Bolyai Soc. Math. Studies*. Springer-Verlag, 2007. 3

[3] Dorit Dor and Uri Zwick. Selecting the median. *SIAM J. Comput.*, 28:1722–1758, 1999. 6

[4] Dániel Gerbner, Dömötör Pálvölgyi, Balázs Patkós, and Gábor Wiener. Finding the maximum and minimum elements with one lie. *Discrete Appl. Math.*, 158(9):988–995, 2010. 2

[5] Lester Randolph Ford Jr. and Delbert Ray Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956. 8

[6] Donald E. Knuth. *The Art of Computer Programming 3: Sorting and Searching*. Addison-Wesley, 1973. 6

[7] Dömötör Pálvölgyi. Lower bounds for finding the maximum and minimum elements with k lies. *Acta Univ. Sapirntiae, Inform.*, 3:224–229, 2011. arXiv:1111.3288 [cs.DM]. 2

[8] Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoret. Comput. Sci.*, 270(1–2):71–109, 2002. 3

[9] Ira Pohl. A sorting problem and its complexity. *Commun. ACM*, 15:462–464, 1972. 2

[10] Bala Ravikumar, K. Ganesan, and Kadathur B. Lakshmanan. On selecting the largest element in spite of erroneous information. In *Proc. 4th Sympos. Theoret. Aspects Comput. Sci.*, volume 247 of *Lecture Notes Comput. Sci.*, pages 88–99. Springer-Verlag, 1987. 3

AUTHORS

Michael Hoffmann
Institute of Theoretical Computer Science,
ETH Zürich, Switzerland
hoffmann@inf.ethz.ch

Jiří Matoušek
Department of Applied Mathematics,
Charles University, Prague, Czech Republic and
Institute of Theoretical Computer Science,
ETH Zürich, Switzerland
matousek@kam.mff.cuni.cz

Yoshio Okamoto
Department of Communication Engineering and Informatics,
University of Electro-Communications, Japan
okamotoy@uec.ac.jp

Philipp Zumstein
University Library
University of Trier, Germany
zumstein@uni-trier.de