Sorting and Selection with Imprecise Comparisons*

Miklós Ajtai
IBM Research - Almaden
ajtai@almaden.ibm.com

Vitaly Feldman
IBM Research - Almaden
vitaly@post.harvad.edu

Avinatan Hassidim†
Bar Ilan University, Israel
avinatanh@gmail.com,

Jelani Nelson‡
Princeton University
minilek@princeton.edu

March 6, 2012

Abstract

We consider a simple model of imprecise comparisons: there exists some $\delta > 0$ such that when a subject is given two elements to compare, if the values of those elements (as perceived by the subject) differ by at least δ, then the comparison will be made correctly; when the two elements have values that are within δ, the outcome of the comparison is unpredictable. This model is inspired by both imprecision in human judgment of values and also by bounded but potentially adversarial errors in the outcomes of sporting tournaments.

Our model is closely related to a number of models commonly considered in the psychophysics literature where δ corresponds to the *just noticeable difference unit (JND)* or *difference threshold*. In experimental psychology, the method of paired comparisons was proposed as a means for ranking preferences amongst n elements of a human subject. The method requires performing all $\binom{n}{2}$ comparisons, then sorting elements according to the number of wins. The large number of comparisons is performed to counter the potentially faulty decision-making of the human subject, who acts as an imprecise comparator.

We show that in our model the method of paired comparisons has optimal accuracy, minimizing the errors introduced by the imprecise comparisons. However, it is also wasteful, as it requires all $\binom{n}{2}$. We show that the same optimal guarantees can be achieved using $4n^{3/2}$ comparisons, and we prove the optimality of our method. We then explore the general trade-off between the guarantees on the error that can be made and number of comparisons for the problems of sorting, max-finding, and selection. Our results provide strong lower bounds and close-to-optimal solutions for each of these problems.

*A preliminary version of this work containing weaker forms of some of the results has appeared in the proceedings of 36th International Colloquium on Automata, Languages and Programming (ICALP) 2009.

†Part of this work was done while the author was at Google Research, Israel.

‡Supported by NSF grant CCF-0832797. Part of this work was done while the author was at IBM Research - Almaden.

1 Introduction

Let $x_1, \ldots, x_n$ be n elements where each x_i has an unknown value $\text{val}(x_i)$. We want to find the element with the maximum value using only pairwise comparisons. However, the outcomes of comparisons are imprecise in the following sense. For some fixed $\delta > 0$, if $|\text{val}(x_i) - \text{val}(x_j)| \leq \delta$, then the result of the comparison can be either "$\geq$" or "$\leq$". Otherwise, the result of the comparison is correct. It is easy to see that in such a setting it might be impossible to find the true maximum (for example when the values of all the elements are within δ). It might however be possible to identify an approximate maximum, that is an element x_{i^*} such that for all x_i, $\text{val}(x_i) - \text{val}(x_{i^*}) \leq k\delta$ for some, preferably small, value k. In addition, our goal is to minimize the number of comparisons performed to find x_{i^*}. We refer to the minimum value k such that an algorithm's output is always guaranteed to be $k\delta$-close to the maximum as the *error* of the algorithm in this setting. Similarly, to sort the above elements with error k we need to find a permutation π such that if $\pi(i) < \pi(j)$ then $\text{val}(x_i) - \text{val}(x_j) \leq k\delta$.

A key issue that our work addresses is that in any sorting (or max-finding) algorithm, errors resulting from imprecise comparisons might accumulate, causing the final output to have high error. Consider, for example, applying the classical bubble sort algorithm to a list of elements that are originally sorted in the reverse order and where the difference between two adjacent elements is exactly δ. All the comparisons will be between elements within δ and therefore, in the worst case, the order will not be modified by the sorting, yielding error $(n-1)\delta$. Numerous other known algorithms that primarily optimize the number of comparisons can be easily shown to incur a relatively high error. As can be easily demonstrated (Theorem 3.1), performing all $\binom{n}{2}$ comparisons then sorting elements according to the number of wins, a "round-robin tournament", achieves error $k = 2$, which is lowest possible (Theorem 3.2). A natural question we ask here is whether $\binom{n}{2}$ comparisons are necessary to achieve the same error. We explore the same question for all values of k in the problems of sorting, max-finding, and general selection.

One motivation for studying this problem comes from social sciences. A common problem both in experimental psychology and sociology is to have a human subject rank preferences amongst many candidate options. It also occurs frequently in marketing research [23, Chapter 10], and in training information retrieval algorithms using human evaluators [1, Section 2.2]. The basic method to elicit preferences is to present the subject two alternatives at a time and ask which is the preferred one. The common approach to this problem today was presented by Thurstone as early as 1927, and is called the "method of paired comparisons" (see [8] for a thorough treatment). In this method, one asks the subject to give preferences for all pairwise comparisons amongst n elements. A ranked preference list is then determined by the number of "wins" each candidate element receives. A central concept in these studies introduced as far back as the 1800s by Weber and Fechner is that of the *just noticeable difference (JND)* unit or *difference threshold* δ. If two physical stimuli with intensities x, y have $|x-y| \leq \delta$, a human will not be able to reliably distinguish which intensity is greater[1]. The idea was later generalized by Thurstone to having humans not only compare physical stimuli, but also abstract concepts [24].

[1]The JND is typically defined relative to x rather than as an absolute value. This is identical to absolute difference in the logarithmic scale and hence our discussion extends to this setting.

Most previous work on the method of paired comparisons has been through the lens of statistics. In such work the JND is modeled as a random variable and the statistical properties of Thurstone's method are studied [8]. Our problem corresponds to a simplified model of this problem which does not require any statistical assumptions and is primarily from a combinatorial perspective.

Another context that captures the intuition of our model is that of designing a sporting tournament based on win/lose games. There, biases of a judge and unpredictable events can change the outcome of a game when the strengths of the players are close. Hence one cannot necessarily assume that the outcome is truly random in such a close call. It is clear that both restricting the influence of the faulty outcomes and reducing the total number of games required are important in this scenario, and hence exploring the tradeoff between the two is of interest. For convenience, in the rest of the paper we often use the terminology borrowed from sporting tournaments.

Accordingly, the problems we consider have a natural interpretation as problems on a tournament graph (that is, a complete directed graph with only one edge between any two vertices). We can view all the comparisons that were revealed by the comparator as a digraph G. The vertices of G are the n elements and it contains the directed edge (x_i, x_j) if and only if a comparison between x_i and x_j has been made, and the comparator has responded with "$x_i \geq x_j$". At any point in time the comparison graph is a subgraph of some unknown tournament graph. The problem of finding a maximum element with error k is then equivalent to finding a vertex in such a graph from which there exists a directed path of length at most k to any other vertex while minimizing the number of edges which need to be revealed (or "matches" in the context of tournaments). Similarly, sorting with error k gives an ordering of vertices such that if vertex x_i occurs after x_j in the order then there exists a directed path of length at most k from x_i to x_j. The connection to tournament graphs is made explicit in Section 5.

Finally, in a number of theoretical contexts responses are given by an imprecise oracle. For example, for weak oracles given by Lovász in the context of optimization [18] and for the statistical query oracle in learning [17] the answer of the oracle is undefined when some underlying value z is within a certain small range of the decision boundary. When z itself is the difference of two other values, say z_1 and z_2, then oracle's answer is, in a way, an imprecise comparison of z_1 and z_2. This correspondence together with our error 2 sorting algorithm was used by one of the authors to derive algorithms in the context of evolvability [10].

1.1 Our results

We first examine the simpler problem of finding only the maximum element. For this problem, we give a deterministic max-finding algorithm with error 2 using $2n^{3/2}$ comparisons. This contrasts with the method of paired comparisons, which makes $(n^2 - n)/2$ comparisons to achieve the same error. Using our algorithm recursively, we build deterministic algorithms with error k that require $O(n^{1+1/((3/4)\cdot 2^k - 1)})$ comparisons. We also give a lower bound of $\Omega(n^{1+1/(2^k - 1)})$ comparisons for the problem. The bounds are almost tight — the upper bound for our error k algorithm is less than our lower bound for error $(k-1)$ algorithms. We also give a linear-time randomized algorithm that achieves error 3 with probability at least $1 - 1/n^2$, showing that randomization greatly changes the complexity of the problem.

We then study the problems of selecting an element of a certain order and sorting. For $k = 2$, we give a deterministic algorithm that sorts using $4n^{3/2}$ comparisons (and in particular can be used for selecting an element of any order). For general k, we show that selection of an element of any order i can be achieved using $O(2^k \cdot n^{1+1/2^{k-1}})$ comparisons and sorting with error k can performed using $O(4^k \cdot n^{1+1/2^{k-1}})$ comparisons.

We give a lower bound of $\Omega(n^{1+1/2^{k-1}})$ comparisons for sorting with error k. When $k = O(1)$ our bounds are tight up to a constant factor and are at most a $\log n$ factor off for general k. Our lower bounds for selection depend on the order of the element that needs to be selected and interpolate between the lower bounds for max-finding and the lower bounds for sorting. For $k \geq 3$, our lower bound for finding the median (and also for sorting) is strictly larger than our upper bound for max-finding. For example, for $k = 3$ the lower bound for sorting is $\Omega(n^{5/4})$, whereas max-finding requires only $O(n^{6/5})$ comparisons.

Note that we achieve $\log\log n$ error for max-finding in $O(n)$ comparisons, and $\log\log n$ error for sorting in $O(n \log^2 n)$ comparisons. Standard methods using the same number (up to a $\log n$ factor) of comparisons (e.g. a single-elimination tournament tree, or Mergesort) can be shown to incur error at least $\log n$. Also, all the algorithms we give are efficient in that their running times are of the same order as the number of comparisons they make.

The basis of our deterministic algorithms for both max-finding and selection are efficient algorithms for a small value of k ($k = 2$). The algorithms for larger error k use several different ways to partition elements, then recursively use algorithms for smaller error and then combine results. Achieving nearly tight results for max-finding requires in part relaxing the problem to that of finding a small *k-max-set*, or a set which is guaranteed to contain at least one element of value at least $x^* - k\delta$, where x^* is the maximum value of an element (we interchangeably use x^* to refer to an element of maximum value as well). It turns out we can find a k-max-set in a fewer number of comparisons than the lower bound for error-k max-finding algorithms. Exploiting this allows us to develop an efficient recursive max-finding algorithm. We note a similar approach of finding a small set of "good" elements was used by Borgstrom and Kosaraju [7] in the context of noisy binary search.

For our randomized max-finding algorithm, we use a type of tournament with random seeds at each level, in combination with random sampling at each level of the tournament tree. By performing a round-robin tournament on the top few tournament players together with the sampled elements, we obtain an element of value at least $x^* - 3\delta$ with polynomially small error probability.

To obtain lower bounds we translate our problems into problems on a tournament graph in which the goal is to ensure existence of short paths from a certain node to most other nodes. Using a comparison oracle that always prefers elements that had fewer wins in previous rounds, we obtain bounds on the minimum of edges that are required to create the paths of desired length. Such bounds are then translated back into bounds on the number of comparisons required to achieve specific error guarantees for the problems we consider. We are unaware of directly comparable techniques having been used before.

1.2 Related Work

Handling noise in binary search procedures was first considered by Rényi [21] and by Ulam [25]. An algorithm for solving Ulam's game was proposed by Rivest et. al. in [22], where an adversarial comparator can err a bounded number of times. They gave an algorithm with query complexity $O(\log n)$ which succeeds if the number of adversarial errors is constant.

Yao and Yao [27] introduced the problem of sorting and of finding the maximal element in a sorting network when each comparison gate either returns the right answer or does not work at all. For finding the maximal element, they showed that it is necessary and sufficient to use $(e+1)(n-1)$ comparators when e comparators can be faulty. Ravikumar, Ganesan and Lakshmanan extended the model to arbitrary errors, showing that $O(en)$ comparisons are necessary and sufficient [20]. For sorting, Yao and Yao showed that $O(n \log n + en)$ gates are sufficient. In a different fault model, and with a different definition of a successful sort, Finocchi and Italiano [12] showed an $O(n \log n)$ time algorithm resilient to $(n \log n)^{1/3}$ faults. An improved algorithm handling $(n \log n)^{1/2}$ faults was later given by Finocchi, Grandoni and Italiano [11].

In the model where each comparison is incorrect with some probability p, Feige et al. [9] and Assaf and Upfal [3] give algorithms for several comparison problems, and [4, 16] give algorithms for binary search. We refer the reader interested in the history of faulty comparison problems to a survey of Pelc [19].

We point out that some of the bounds we obtain appear similar to those known for max-finding, selection, and sorting in parallel in Valiant's model [26]. In particular, our bounds for max-finding are close to those obtained by Valiant for the parallel analogue of the problem (with the error used in place of parallel time) [26], and our lower bound of $\Omega(n^{1+1/(2^k-1)})$ for max-finding with error k is identical to a lower (and upper) bound given by Häggkvist and Hell [15] for merging two sorted arrays each of length n using a k-round parallel algorithm. Despite these similarities in bounds, our techniques are different, and we are not aware of any deep connections. As some evidence of the difference between the problems we note that for sorting in k parallel rounds it is known that $\Omega(n^{1+1/k})$ comparisons are required [2, 6, 14], whereas in our model, for constant k, we can sort with error k in $n^{1+1/2^{\Theta(k)}}$ comparisons. For a survey on parallel sorting algorithms, the reader is referred to [13].

2 Notation

Throughout this document we let x^* denote some x_i of the maximum value (if there are several such elements, we choose one arbitrarily). Furthermore, we use x_i interchangeably to refer to the both the i^{th} element and its value, e.g. $x_i > x_j$ should be interpreted as $\mathrm{val}(x_i) > \mathrm{val}(x_j)$.

We assume $\delta = 1$ without loss of generality, since the problem with arbitrary $\delta > 0$ is equivalent to the problem with $\delta = 1$ and input values x_i/δ. We stress that the algorithm does not know δ.

We say x *defeats* y when the comparator claims that x is larger than y (and we similarly use the phrase y *loses to* x). We say x is *k-greater* than y ($x \geq_k y$) if $x \geq y - k$. The term *k-smaller* is defined analogously. A set of elements T is k-greater than a set of elements S if for every $y \in S$ there exists $x \in T$ such that $x \geq_k y$. We say an element is a *k-max* of a set if it is k-greater than all other elements in the set. If the set is not specified explicitly then we refer to the set of all input elements. A permutation $x_{\pi(1)}, \ldots, x_{\pi(n)}$ is *k-sorted* if $x_{\pi(i)} \geq_k x_{\pi(j)}$ for every $i > j$. A *k-max-set* is a subset of all elements which contains at least one element of value at least $x^* - k$.

All logarithms throughout this document are base-2. For simplicity of presentation, we occasionally omit floors and ceilings and ignore rounding errors when they have an insignificant effect on the bounds.

3 Max-Finding

In this section we give deterministic and randomized algorithms for max-finding.

3.1 Deterministic Algorithms

We start by showing that the method of paired comparisons provides an optimal error guarantee, not just for max-finding, but also for sorting.

Theorem 3.1 *Sorting according to the number of wins in a round-robin tournament has error at most* 2*.*

Proof. Let x, y be arbitrary elements with y strictly less than $x - 2$. For any z that y defeats, x also defeats z. Furthermore, x defeats y, and thus x has strictly more wins than y, implying y is placed lower in the sorted order. ■

Theorem 3.2 *No deterministic max-finding algorithm has error less than* 2*.*

Proof. Given three elements a, b, c, the comparator can claim $a > b > c > a$, making the elements indistinguishable. Without loss of generality, suppose A outputs a. Then the values could be $a = 0$, $b = 1$, $c = 2$, implying A has error 2. ■

In Figure 1 we give our error 2 algorithm for max-finding.

Lemma 3.3 *For every $s \leq n$, the max-finding algorithm* 2-`MaxFind` *has error* 2 *and makes at most $(n-s)s + (n^2 - s^2)/(s-1)$ comparisons. In particular, the number of comparisons is at most $2n^{3/2}$ for $s = \lceil \sqrt{n} \rceil$.*

Proof. We first analyze the number of comparisons. In the t^{th} iteration, the number of comparisons is at most $\binom{s}{2} + (n_t - s)$, where n_t is the number of candidate elements in round t. We now bound the number of iterations and n_t. In all but the last iteration, the total number of comparisons made in the round-robin tournament is $\binom{s}{2} = s(s-1)/2$. Thus by an averaging argument, the element which won the largest number of comparisons won at least $(s-1)/2$ times. Thus, at least $(s-1)/2$ elements are eliminated in each iteration, implying the number of iterations is at most $2(n-s)/(s-1)$ and $n_t \leq n - t(s-1)/2$. The total number of comparisons is thus at most

$$\sum_{i \leq 2(n-s)/(s-1)} [s(s-1)/2 + n - t(s-1)/2] \leq (n-s)s + (n^2 - s^2)/(s-1) \ .$$

Algorithm 2-`MaxFind`: `// Returns an element of value at least` x^*-2. `The value` $s>1$ `is a parameter which is by default` $\lceil\sqrt{n}\rceil$ `when not specified.`

1. Label all x_i as candidates.
2. **while** there are more than s candidate elements:
   (a) Pick an arbitrary subset of s of the candidate elements and play them in a round-robin tournament. Let x be the element with the largest number of wins.
   (b) Compare x against all candidate elements and eliminate all elements that lose to x.
3. Play the remaining (at most s) candidate elements in a round-robin tournament and return the element with the largest number of wins.

Figure 1: The algorithm 2-`MaxFind` for finding a 2-max.

We now analyze error in two cases. The first case is that x^* is never eliminated, and thus x^* participates in Step 3. Theorem 3.1 then ensures that the final output is of value at least $x^* - 2$. Otherwise, consider the iteration when x^* is eliminated. In this iteration, it must be the case that the x chosen in Step 2(b) has $x \geq x^* - 1$, and thus any element with value less than $x^* - 2$ was also eliminated in this iteration. In this case all future iterations only contain elements of value at least $x^* - 2$, and so again the final output has value at least $x^* - 2$. ■

The key recursion step of our general error max-finding algorithm is the algorithm 1-`Cover` (given in Lemma 3.5) which is based on 2-`MaxFind` and the following lemma.

Lemma 3.4 *There is a deterministic algorithm which makes $\binom{n}{2}$ comparisons and outputs a 1-max-set of size at most $\lceil\log n\rceil$.*

Proof. We build the output set in a greedy manner. Initialize $S = \emptyset$. At each step consider the subtournament on the vertices which are neither in S nor defeated by an element of S. An averaging argument shows that there exists an element which wins at least half its matches in this subtournament; add this element to S. At least one element in the final set S must either have value x^* or have defeated x^*, and thus S is a 1-max-set. ■

We now obtain 1-`Cover` by setting $s = \lceil 2\sqrt{n}\rceil$ in Figure 1, then returning the union of the x that were chosen in any iteration of Step 2(a), in addition to the output of Lemma 3.4 on the elements in the final tournament in Step 3.

Lemma 3.5 *There is an algorithm 1-`Cover` making at most $3 \cdot n^{3/2}$ comparisons which finds a 1-max-set of size at most $\sqrt{n}$ (for $n \geq 81$).*

Proof. Run the algorithm 2-`MaxFind` with $s = \lceil 2\sqrt{n}\rceil$. Return the set consisting of all elements that won the round-robin tournament in step 2(b) of Figure 1 in at least one iteration, in addition to a

Algorithm k-MaxFind: // Returns a k-max for $k \geq 3$

1. If $n \leq 81$ return the output of 2-MaxFind on the input elements.
2. Equipartition the n elements into sets $S_1, \ldots, S_t$ each of size $r = \max\{81, 4 \cdot n^{\frac{8}{3 \cdot 2^k - 4}}\}$
3. Call 1-Cover on each set S_i to recover a set T_i 1-greater than S_i.
4. Return $(k-1)$-MaxFind$(\cup_{i=1}^t T_i)$ // Recursion stops at 2-MaxFind.

Figure 2: The algorithm k-MaxFind for finding a k-max.

size-$\lceil \log s \rceil$ set 1-greater than the candidate elements which were left in Step 3 (using Lemma 3.4). The total size of the returned set is thus $\lceil n/s \rceil - 1 + \lceil \log s \rceil \leq \lceil \sqrt{n}/2 \rceil + \lceil \log(\lceil 2\sqrt{n} \rceil) \rceil - 1$. For $n \geq 81$, this is at most $\sqrt{n}$.

To show correctness, consider the element x^* of maximal value. Either x^* was eliminated in Step 2(c) of some iteration, in which case the element x that eliminated x^* had value at least $x^* - 1$, or x^* survived until the last iteration, in which case the set constructed via Lemma 3.4 is a 1-max-set. Finally, note that the number of comparisons is the same as the number of comparisons used by 2-MaxFind (with the same s) and therefore is less than $3n^{3/2}$. ■

We are now ready to give our main algorithm for finding a k-max.

Theorem 3.6 *For every $3 \leq k \leq \log \log n$, k-MaxFind uses $O(n^{1+1/((3/4)2^k-1)}) = O(n^{3 \cdot 2^k/(3 \cdot 2^k - 4)})$ comparisons and finds a k-max.*

Proof. We prove that for any $2 \leq k \leq \log \log n$, k-MaxFind uses at most $54 \cdot n^{3 \cdot 2^k/(3 \cdot 2^k - 4)}$ comparisons and finds a k-max by induction. First, by Lemma 3.3, it holds for 2-MaxFind.

We now prove the bound on the error. Let x be the element returned by k-MaxFind. By the inductive hypothesis, for every $y \in \cup_{i=1}^t T_i$, $x \geq_{k-1} y$. In addition, for every input element x_j there exists $y \in T_i$ for some i such that $y \geq_1 x_j$. Therefore, $x \geq_k x_j$ for every $j \in [n]$.

The total number of comparisons used by k-MaxFind are as follows.

- if $n \leq 81$ then $4 \cdot n^{3/2} \leq 36 \cdot n$. Otherwise,
- $t = n/r$ invocations of 1-Cover. By Lemma 3.5, this requires at most $3 \cdot r^{3/2} \cdot n/r = 3\sqrt{r} \cdot n$ comparisons which equals $\max\{27 \cdot n, 6 \cdot n^{\frac{3 \cdot 2^k}{3 \cdot 2^k - 4}}\}$. Note that $n^{\frac{3 \cdot 2^k}{3 \cdot 2^k - 4}} \geq n$ and therefore we can use $27 \cdot n^{\frac{3 \cdot 2^k}{3 \cdot 2^k - 4}}$ as an upper bound.
- The invocation of $(k-1)$-MaxFind on $\cup_{i=1}^t T_i$. By Lemma 3.5, the size of each T_i is at most $\sqrt{r}$. Therefore, $|\cup_{i=1}^t T_i| = \sqrt{r} \cdot n/r = n/\sqrt{r} \leq n^{\frac{3 \cdot 2^k - 8}{3 \cdot 2^k - 4}}/2$. By, the inductive assumption this

invocation requires at most

$$54 \cdot \left(n^{\frac{3\cdot 2^k-8}{3\cdot 2^k-4}}/2\right)^{1+1/((3/4)2^{k-1}-1)} \le 27 \cdot n^{\frac{3\cdot 2^k-8}{3\cdot 2^k-4}\cdot\frac{3\cdot 2^{k-1}}{3\cdot 2^{k-1}-4}} = 27 \cdot n^{\frac{3\cdot 2^k}{3\cdot 2^k-4}} .$$

Altogether the number of comparisons is at most $\max\{36 \cdot n, 54 \cdot n^{\frac{3\cdot 2^k}{3\cdot 2^k-4}}\} = 54 \cdot n^{\frac{3\cdot 2^k}{3\cdot 2^k-4}}$. ■

Corollary 3.7 *There is a max-finding algorithm using $O(n)$ comparisons with error of at most $\log\log n$.*

3.2 Randomized Max-Finding

We now show that randomization can significantly reduce the number of comparisons required to find an approximate maximum. Our algorithm operates correctly even if the adversary can adaptively choose how to err when two elements are close (though we stress that the adversary may not change input values over the course of an execution). In particular, the classic randomized selection algorithm can take quadratic time in this adversarial model since for an input with all equal values, the adversary can claim that the randomly chosen pivot is smaller than all other elements. Nevertheless, even in this strong adversarial model, we show the following.

Theorem 3.8 *For any integer $c \ge 1$, there exists a randomized algorithm which given any n elements finds a 3-max of the set with probability at least $1 - n^{-c}$ using at most $(s-1)n + ((c+1)/(C\ln 2))^2/2 \cdot n^{2/3}\ln^4 n = O(n)$ comparisons, where s, C are as defined in Figure 3.*

Taking $c > 1$, and using the fact that the error of our algorithm can never be more than $n-1$, this gives an algorithm which finds an element with expected value at least $x^* - 4$. The high-level idea of the algorithm is as follows. We randomly equipartition the elements into constant-sized sets. In each set we play a round-robin tournament and advance everyone who won more than 3/4 of its comparisons. As we will prove, the element with the median number of wins can win at most 3/4 of its comparisons and hence no more than half of the elements advance. We also randomly sample a set of elements at each level of the tournament tree. We show that either (1) at some round of the tournament there is an abundance of elements with value at least $x^* - 1$, in which case at least one such element is sampled with high probability, or (2) x^* advances as one of the top few tournament elements with high probability. Figure 3 presents the subroutine `SampledTournament` for the algorithm.

We now proceed to the analysis of our algorithm. First we show that the element with the median number of wins (or the element of order $\lceil n/2 \rceil$ when sorted in increasing order by number of wins) must incur a significant number of losses. We in fact show that it must also incur a significant number of wins, but we will not need this latter fact until presenting our selection and sorting algorithms.

Lemma 3.9 *In a round-robin tournament on n elements, the element with the median number of wins has at least m wins and at least m losses for $m = \lceil(\lceil n/2 \rceil - 1)/2\rceil \ge \lceil n/5 \rceil$.*

Algorithm `SampledTournament`: `// For constant` c `and` n `sufficiently large, returns a 1-max-set with probability at least` $1 - n^{-c}$.

1. Initialize $N_0 \leftarrow \{x_1, \ldots, x_n\}$, $W \leftarrow \emptyset$, $s \leftarrow 15(c+2)+1$, $C \leftarrow 4^4/(5e)^5$, and $i \leftarrow 0$, where e is Euler's number.
2. **if** $|N_i| \leq ((c+1)/C)n^{1/3} \ln n$, insert N_i into W and **return** W.
3. **else** insert a random subset of $((c+1)/C)n^{1/3} \ln n$ elements from N_i into W.
4. Randomly partition the elements in N_i into sets of size s. In each set, perform a round-robin tournament.
5. Let N_{i+1} contain all elements of N_i which had strictly fewer than $(s-2)/4$ losses in their round-robin tournament in Step 4. Increment i and **goto** Step 2.

Figure 3: The algorithm `SampledTournament`.

Proof. Sort the elements by their total number of wins and let x' be the median according to the number of wins. Let ℓ denote the number of wins of x'. Assume that n is even. Then the total number of wins for all the elements is at most $n/2 \cdot \ell + n/2 \cdot (n-1) - \binom{n/2}{2}$ since there are $n/2$ elements with at most ℓ wins and the total number of wins by the $n/2$ elements that have more wins than x' is at most $n/2 \cdot (n-1) - \binom{n/2}{2}$. But the total number of wins is exactly $\binom{n}{2}$ and therefore we obtain that $\ell \cdot n/2 \geq \binom{n/2}{2}$, or $\ell \geq (n-2)/4$. The bound on the number of losses is obtained in the same way. If n is odd then this argument gives a bound of $(\lceil n/2 \rceil - 1)/2$. ■

Lemma 3.10 *Given any integer $c \geq 1$, `SampledTournament` outputs a W of size at most $(c+1)/(C \ln 2) \cdot n^{1/3} \ln^2 n$ after at most $(s-1)n$ comparisons such that W is a 1-max-set with probability at least $1 - n^{-c}$.*

Proof. By Lemma 3.9 the element with the median number of wins has at least $(s-2)/4$ losses during each round-robin tournament in Step 4, and thus the fraction of elements that survive from one iteration to the next in Step 5 is at most $1/2$. Therefore, the number of iterations is at most $\lceil \log_2 n \rceil$. In each iteration we sample $((c+1)/C) \cdot n^{1/3} \ln n$ elements, and thus the size of the output W is as claimed. Also, the total number of comparisons is at most $\sum_{i=0}^{\infty} \binom{s}{2} \cdot (n/2^i)/s = (s-1)n$, where $n/2^i$ is an upper bound on $|N_i|$.

We now show that W is a 1-max-set with probability at least $1 - n^{-c}$. We say that an iteration i is *good* if either x^* advances to N_{i+1}, or W contains a 1-max at the end of round i. We then show that for any iteration i, conditioned on the event that iterations $1, \ldots, i-1$ were good we have that i is good with probability $1 - 1/n^{c+1}$. The lemma would then follow by a union bound over all iterations i.

Now consider an iteration i where $1, \ldots, i-1$ were good. Then either W already contains a 1-max, in which case i is good, or W does not contain a 1-max but $x^* \in N_i$. Let us focus on this latter case. Define $n_i = |N_i|$. Let $\mathcal{Q}_i$ be the event that the number of elements in N_i with value at least $x^* - 1$ is at least $C\alpha n_i$ for $\alpha = n^{-1/3}$. We show a dichotomy: either $\mathcal{Q}_i$ holds, in which case x^* is sampled into W with probability $1 - 1/n^{c+1}$, or $\mathcal{Q}_i$ does not hold, in which case $x^* \in N_{i+1}$ with probability $1 - n^{c+1}$.

To show the dichotomy, let us first assume $\mathcal{Q}_i$ holds. Then, the probability we do not sample a 1-max into W is at most

$$\left(1 - \frac{C\alpha n_i}{n_i}\right)^{((c+1)/C)n^{1/3}\ln n} \leq \left(1 - \frac{C}{n^{1/3}}\right)^{((c+1)/C)n^{1/3}\ln n} \leq e^{-(c+1)\ln n} = \frac{1}{n^{c+1}}.$$

Now let us assume that $\mathcal{Q}_i$ does not hold. Then for x^* to not advance to N_{i+1} we must have that at least $s/5$ 1-maxes were placed into the same set as x^* in iteration i. Using that $(a/b)^b \leq \binom{a}{b} \leq (ea/b)^b$, this happens with probability at most

$$\begin{aligned}
\frac{\binom{C\alpha n_i}{s/5} \cdot \binom{n_i}{4s/5}}{\binom{n_i}{s-1}} &< n_i \cdot \frac{\binom{C\alpha n_i}{s/5} \cdot \binom{n_i}{4s/5}}{\binom{n_i}{s}} \\
&\leq n_i \cdot \frac{e^s \cdot (5C\alpha n_i/s)^{s/5} \cdot (5Cn_i/(4s))^{4s/5}}{(n_i/s)^s} \\
&\leq n_i \cdot (e \cdot C^{1/5} \cdot 5^{1/5} \cdot (5/4)^{4/5} \cdot n^{-1/15})^s \\
&= n^{-s/15+1},
\end{aligned}$$

which is at most $n^{-(c+1)}$ by our choice of s and C. ■

Proof (of Theorem 3.8). Run the algorithm in Figure 3. By Lemma 3.10, the output W is 1-max-set with probability at least $1 - n^{-c}$. Conditioned on this event, a 2-max of W is thus a 3-max of the entire original input. A 2-max of W can be found via a round-robin tournament by Theorem 3.1 using $\binom{|W|}{2} < |W|^2/2$ comparisons. The total number of comparisons is thus the sum of comparisons made in Figure 3, and in the final round robin tournament, which gives the bound claimed in the theorem statement. ■

4 Sorting and Selection

We now consider the problems of sorting and selection. We first present an algorithm 2-`Sort` which sorts with error 2 using $O(n^{3/2})$ comparisons (and, in particular can be used for selection with error 2). We then describe the selection and sorting algorithms for general error k. We start by formally defining what is meant by selecting an element of certain order with error.

Definition 4.1 *Element x_j in the set $X = \{x_1, \ldots, x_n\}$ is of k-*order *i if there exists a partition S_1, S_2 of $X \setminus \{x_j\}$ such that $|S_1| = i - 1$, and $S_1 \cup \{x_j\} \leq_k S_2 \cup \{x_j\}$. A k-*median *is an element of k-order $\lceil n/2 \rceil$.*

Our error 2 sorting algorithm is based on modifying 2-`MaxFind` so that the x found in Step 2(a) of Figure 1 is used as a pivot. We then compare this x against all elements and pivot into two sets, recursively sort each, then concatenate.

Theorem 4.2 *There is a deterministic sorting algorithm* 2-`Sort` *with error* 2 *that requires at most* $4 \cdot n^{3/2}$ *comparisons.*

Proof. The algorithm 2-`Sort` is based on the algorithm 2-`MaxFind` described in Figure 1. If $n \le 64$ then we just perform a round-robin tournament on the elements (since $4 \cdot 64^{3/2} > \binom{64}{2}$). Otherwise let $s(n) = \sqrt{2n}$. We choose some s elements and perform a round-robin tournament on them. Now let x be an element with the median number of wins. We compare x to all elements and let S_1 be the set of elements that lost to x and S_2 be the set of all elements that defeated x. Then we recursively sort S_1 and S_2 then output sorted S_1, then x, and then sorted S_2. Note that any $(y, y') \in (S_1 \cup \{x\}) \times S_2$ satisfies $y' \ge_2 y$ since $y' \ge_1 x$ and $x \ge_1 y$. Correctness follows by induction from Theorem 3.1.

Let $g(n)$ be the worst-case number of comparisons required by this algorithm. We claim that $g(n) \le 4 \cdot n^{3/2}$. By the definition, this holds for $n \le 64$. Now let $t = |S_2|$. The number of comparisons used in a recursive call is at most $s(s-1)/2 + n - s + g(n-t-1) + g(t)$. By our inductive assumption this is at most $s(s-1)/2 + n - s + 4((n-t-1)^{3/2} + t^{3/2})$. Without loss of generality we can assume that $t \le (n-1)/2$ and then obtain that this function is maximized when t is the smallest possible (via a straightforward analysis of the derivative). By Lemma 3.9, x has at least $(s-2)/4$ wins and $(s-2)/4$ losses and therefore $t \ge (s-2)/4$. Now we observe that $(n-t-1)^{3/2} \le n^{3/2} - \frac{3}{2}\sqrt{n}(t+1)$ (to verify it is sufficient to square both sides and use the fact that $t \le n$). Hence

$$\begin{aligned} g(n) &\le s^2/2 + n - 3s/2 + 4(n^{3/2} - \frac{3}{2}\sqrt{n}(t+1) + t^{3/2}) \\ &\le 4 \cdot n^{3/2} + 2n - 2\sqrt{2n} - \frac{3\sqrt{n}(\sqrt{2n}+2)}{2} + n^{3/4} \\ &= 4 \cdot n^{3/2} - (3/\sqrt{2} - 2)n - (3 + 3/\sqrt{2})\sqrt{n} + n^{3/4} < 4 \cdot n^{3/2}. \end{aligned}$$

The last line of the equation follows from the following application of the inequality of arithmetic and geometric means

$$(3/\sqrt{2} - 2)n + (3 + 3/\sqrt{2})\sqrt{n} \ge 2\sqrt{(3/\sqrt{2} - 2)(3 + 3/\sqrt{2})n^{3/4}} > n^{3/4} \ .$$

This finishes the proof of the induction step. ■

At a high level our algorithm for k-order selection is similar to the classical selection algorithm of Blum et al. [5], in that in each step we try to find a pivot that allows us to recurse on a problem of geometrically decreasing size. In our scenario though, a good pivot must have an additional property which we now define.

Definition 4.3 *Element x_j in the set $X = \{x_1, \ldots, x_n\}$ is a k-pivot for m elements if there exist disjoint sets $S_w \subset X \setminus \{x_j\}$* (winning set) *and $S_l \subset X \setminus \{x_j\}$* (losing set)*, such that $|S_w| = |S_l| = m$, $x_\ell \le_k x_j$ for all $\ell \in S_l$, and $x_\ell \ge_k x_j$ for all $\ell \in S_w$.*

In order to use an element as a pivot in our algorithm it must be a $(k-1)$-pivot. We construct a necessary pivot via a recursive algorithm that given n elements and a number k constructs a k-pivot for at least $n/(5 \cdot 2^{k-1})$ elements. For $k = 1$, Lemma 3.9 gives the desired algorithm. The general algorithm is effectively a recursive application of Lemma 3.9 and is described below. Our algorithm for selecting an element of k-order i is in Figure 5.

We claim that algorithm k-`Select` finds an element of k-order i using at most $O(2^k \cdot n^{1+1/2^{k-1}})$ comparisons. To prove this, we first analyze the algorithm k-`Pivot`.

Lemma 4.4 *For any $1 \leq k \leq \log\log n$, given n elements the deterministic algorithm k-`Pivot` finds a k-pivot for $m \geq n/(5 \cdot 2^{k-1})$ and corresponding losing set S_l and winning set S_w using at most $9 \cdot n^{1+1/(2^k-1)} + c_n$ comparisons, where $c_n = \min\{\binom{n}{2}, \binom{215}{2}\}$.*

Proof. We prove that for any $1 \leq k \leq \log\log n$, k-`Pivot` uses at most $9 \cdot n^{1+1/(2^k-1)} + c_n$ comparisons and finds a k-pivot for $m \geq n/(5 \cdot 2^{k-1})$ elements by induction. First, if $k = 1$ or $n \leq 215$ then by Lemma 3.9, it holds for 1-`Pivot` since $m \geq n/5$ and the total number of comparisons $n(n-1)/2 \leq 9 \cdot n^{1+1/(2^k-1)} + c_n$.

We now prove the bound on the error in the general case when $k \geq 2$ and $n \geq 216$. Let y be the element returned by k-`Pivot`. By the inductive hypothesis, for every $v \in S_l$, if $v \in S'_l$ then $v \leq_{k-1} y$. Otherwise, when $v \in S_{i,l}$ for some $y_i \in S'_l$ we get that $v \leq_1 y_i$ and $y_i \leq_{k-1} y$. This implies that in both cases $v \leq_k y$. Similarly, for every $v \in S_w$, $v \geq_k y$.

Next we prove the bound on m. The algorithm performs a round-robin on s elements until at most $s-1$ elements are left and each such round eliminates exactly $2s'+1$ elements. Therefore the number of rounds t is at least $\geq \lceil (n-s+1)/(2s'+1) \rceil$. By the inductive hypothesis,

$$m' = |S'_l| = |S'_w| \geq t/(5 \cdot 2^{k-2}) \geq \frac{n-s+1}{5 \cdot 2^{k-2} \cdot (2s'+1)} \ .$$

Now,

$$m = m' \cdot (s'+1) \geq \frac{n-s+1}{2s'+1} \cdot \frac{s'+1}{5 \cdot 2^{k-2}} = \frac{n}{5 \cdot 2^{k-1}} + \frac{n-2(s-1)(s'+1)}{5 \cdot 2^{k-1} \cdot (2s'+1)} \ .$$

If $s \leq 19$ then $n \geq 216$ implies that $n-2(s-1)(s'+1) \geq 0$. Otherwise (for $s \geq 20$), $k \geq 2$ implies that $s \leq \lceil 3 \cdot n^{1/3} \rceil$ and therefore, $n \geq ((s-1)/3)^3$. By definition of s', $s' = \lceil (\lceil s/2 \rceil - 1)/2 \rceil \leq (s+1)/4$. Therefore, for $s \geq 20$,

$$n - 2(s-1)(s'+1) \geq \left(\frac{s-1}{3}\right)^3 - \frac{s^2+4s+5}{2} > 0.$$

Hence $m = |S_l| = |S_w| \geq \frac{n}{5 \cdot 2^{k-1}}$.

Finally, we prove the bound on the total number of comparisons used by k-`Pivot` when $k \geq 2$ and $n \geq 216$ as follows:

- $t \leq \lfloor n/(2s'+1) \rfloor$ invocations of round-robin on s elements. By definition $2s'+1 \geq s/2$ and therefore this step requires at most $\lfloor 2n/s \rfloor s(s-1)/2 \leq n(s-1) \leq 3 \cdot n^{1+1/(2^k-1)}$ comparisons.

Algorithm k-`Pivot`: `// Given a set of` $n \geq 3$ `elements` X`, returns a` k`-pivot for` $m \geq n/(5 \cdot 2^{k-1})$ `elements and corresponding losing and winning sets (as described in Definition 4.3).`

1. **if** $k = 1$ or $n \leq 215$ `// Base case`
   (a) Perform a round-robin tournament on X.
   (b) Let y be the element with the median number of wins.
   (c) Let m be the smaller of the number of wins and the number of losses of y.
   (d) Let S_l be a set of m elements that lost to y and S_w be a set of m elements that defeated y.
2. **else**
   (a) Set $s \leftarrow \left\lceil 3 \cdot n^{\frac{1}{2^k - 1}} \right\rceil$ and set $s' \leftarrow \lceil (\lceil s/2 \rceil - 1)/2 \rceil$.
   (b) Initialize $T \leftarrow X$, $i \leftarrow 0$.
   (c) **while** $|T| \geq s$
      i. $i \leftarrow i + 1$.
      ii. Let X_i be a set of s arbitrary elements from T.
      iii. Perform a round-robin tournament on X_i.
      iv. Set y_i to be the element with the median number of wins.
      v. Set $S_{i,l}$ to be a set of any s' elements that lost to y and $S_{i,w}$ a set of s' elements that defeated y_i.
      vi. Update $T \leftarrow T \setminus (S_{i,l} \cup S_{i,w} \cup \{y_i\})$.
   (d) Set $t \leftarrow i$ and $Y \leftarrow \{y_1, y_2, \ldots, y_t\}$.
   (e) Recursively call $(k-1)$-`Pivot` on Y and let y, S'_l and S'_w be the $(k-1)$-pivot and the sets returned.
   (f) Set $S_l \leftarrow S'_l \cup \bigcup_{y_i \in S'_l} S_{i,l}$ and $S_w \leftarrow S'_w \cup \bigcup_{y_i \in S'_w} S_{i,w}$.
3. Return y, S_l and S_w.

Figure 4: The algorithm k-`Pivot` for finding a k-pivot for at least $n/(5 \cdot 2^{k-1})$ elements

Algorithm k-`Select`: `// Given a set` X `of` $n \geq 3$ `elements, returns an element of` k`-order` i `in` X

1. **if** $k \leq 2$ or $n \leq 8$, sort elements using 2-`Sort` then return the element with index i.
2. **else**
   (a) Set $s \leftarrow \left\lfloor n^{1-2^{1-k}} \right\rfloor$ and let T be a set of any s elements from X.
   (b) Call $(k-1)$-`Pivot` on T and let y, S_l and S_w be the pivot and the sets returned.
   (c) Compare y with each of the other $n-1$ elements.
   (d) **if** y defeats at least $(n-1)/2$ elements
      i. Set X_1 to be the set of all elements that y defeats and are not in S_w.
      ii. Set $X_2 = X \setminus (X_1 \cup \{y\})$.
   (e) **else** // the symmetric case
      i. Set X_2 to be the set of all elements that y lost to and are not in S_l.
      ii. Set $X_1 = X \setminus (X_2 \cup \{y\})$.
   (f) **if** $|X_1| = i - 1$ return y.
   (g) **else if** $i \leq |X_1|$, call k-`Select` recursively to find an element of k-order i in X_1.
   (h) **else** call k-`Select` recursively to find an element of k-order $(i - |X_1| - 1)$ in X_2.

Figure 5: The algorithm k-`Select`.

- The invocation of $(k-1)$-Pivot on $t \le \lfloor 2n/s \rfloor$ elements. By our inductive hypothesis, this requires $9t^{1+1/(2^{k-1}-1)} + c_t \le 9(2 \cdot n^{1-1/(2^k-1)}/3)^{1+1/(2^{k-1}-1)} + c_n \le 6 \cdot n^{1+1/(2^k-1)} + c_n$.

Altogether the number of comparisons is at most $9 \cdot n^{1+1/(2^k-1)} + c_n$, as was claimed. ∎

We are now ready to state and prove our bounds for k-Select formally.

Theorem 4.5 *For any $2 \le k \le \log\log n$, there is a deterministic algorithm k-Select which, given n elements and $i \in [n]$, finds an element of k-order i using at most $25 \cdot 2^{k-1} n^{1+2^{1-k}} + 5 \cdot 2^{2k-3} n^{2^{1-k}} c_n$ comparisons, where $c_n = \min\{\binom{n}{2}, \binom{215}{2}\}$ (as defined in Lemma 4.4).*

Proof. We prove the claim by induction on n. For $n \le 12$ we use 2-Sort which sorts with error 2. An element i in such a sorting has 2-order i. Also, according to Theorem 4.2, the algorithm uses at most $4n^{3/2} \le 25 \cdot 2^{k-1} \cdot n^{1+2^{1-k}}$ comparisons.

We now consider the general case when $n \ge 9$ and $k \ge 3$. If y defeats at least $(n-1)/2$ elements then for every element $z_1 \in X_1$, $z_1 \le_1 y$ and, by the properties of the $(k-1)$-pivot for every element z_2 in X_2, $y \le_{k-1} z_2$. In particular, $z_1 \le_k z_2$. Now, if $|X_1| = i-1$ then X_1 and X_2 form a partition of $X \setminus \{y\}$ showing that y is an element of k-order i. If $|X_1| < i-1$ then let y' be the k-order i element in X_1 returned by the recursive call to k-Select. There exists a partition of X_1 into sets S_1' and S_2' showing that y' is an element of k-order i in X_1. We set $S_1 = S_1'$ and $S_2 = S_2' \cup \{y\} \cup X_2$. For every $z_1 \in S_1$, $z_1 \le_k y$ and for every $z_2 \in X_2$, $z_1 \le_k z_2$. Hence, y is an element of k-order i. The case when $|X_1| > i$ and the symmetric case when y lost to at least $(n-1)/2$ are analogous. This proves that k-Select returns an element of k-order i.

Finally, in the general case, the number of comparisons k-Select uses is as follows.

- Call to $(k-1)$-Pivot on $s = \left\lfloor n^{1-2^{1-k}} \right\rfloor \ge \lfloor \sqrt{n} \rfloor \ge 3$ elements. Lemma 4.4 implies that this step uses at most $9s^{1+1/(2^{k-1}-1)} + c_s \le 9n + c_n$ comparisons.

- Comparison of the pivot with all the elements uses at most $n-1$ comparisons.

- The invocation of k-Select on one of X_1 and X_2. By the definition of pivot, $|S_w| = |S_l| \le (s-1)/2 \le (n-2)/4$. Hence we observe that if y defeated at least $(n-1)/2$ elements then $|X_1| \ge (n-1)/2 - |S_w| \ge (n-2)/4 \ge |S_l|$. And by the definition of X_2, $|X_2| \ge |S_w|$. Similarly, if y lost to at least $(n-1)/2$ elements then $|X_1| \ge |S_l|$ and $|X_2| \ge |S_w|$. Assume, without loss of generality, that $|X_1| \ge |X_2|$ and let $\alpha = (|X_2|+1)/n$. By the properties of the $(k-1)$-pivot,

$$|X_2| \ge s/(5 \cdot 2^{k-2}) \ge \left\lfloor n^{1-2^{1-k}} \right\rfloor / (5 \cdot 2^{k-2}) \ge n^{1-2^{1-k}}/(5 \cdot 2^{k-2}) - 1,$$

or $\alpha \ge n^{-2^{1-k}}/(5 \cdot 2^{k-2})$. The number of comparisons is maximized when k-Select is executed on X_1 which has size $n - |X_2| - 1 = n - \alpha n$ and, by our inductive hypothesis, this step can be done using N comparisons for

$$
\begin{aligned}
N \leq & 25 \cdot 2^{k-1} (n-\alpha n)^{1+2^{1-k}} + 5 \cdot 2^{2k-3} \cdot (n-\alpha n)^{2^{1-k}} \cdot c_n \\
\leq & 25 \cdot 2^{k-1} \cdot n^{1+2^{1-k}} (1-\alpha)^{1+2^{1-k}} + 5 \cdot 2^{2k-3} \cdot n^{2^{1-k}} \cdot (1-\alpha)^{2^{1-k}} \cdot c_n \\
\leq & 25 \cdot 2^{k-1} \cdot n^{1+2^{1-k}} (1-\alpha) + 5 \cdot 2^{2k-3} \cdot n^{2^{1-k}} \cdot (1-2^{1-k} \cdot \alpha) \cdot c_n \\
= & 25 \cdot 2^{k-1} \cdot n^{1+2^{1-k}} + 5 \cdot 2^{2k-3} \cdot n^{2^{1-k}} c_n - 10 \cdot n - c_n
\end{aligned}
$$

Therefore altogether the number of comparisons is at most $25 \cdot 2^{k-1} \cdot n^{1+2^{1-k}} + 5 \cdot 2^{2k-3} \cdot n^{2^{1-k}} \cdot c_n$.
■

We now show that with a small change our selection algorithm can be used to produce a complete sorting with error k. Namely, instead of running recursively on one of the subsets X_1 and X_2, we recursively sort each partition, then concatenate the sorted results in the order k-`Sort`$(X_1), y, k$-`Sort`(X_2). As expected, in the base case (when $n \leq 8$) we just output the result of 2-`Sort`. We claim that the resulting algorithm, to which we refer to as k-`Sort`, has the following bounds on the number of comparisons.

Theorem 4.6 *For any $2 \leq k \leq \log\log n$, there is a deterministic algorithm k-`Sort` which given a set X of n elements, sorts the elements of X with error k and uses at most $7 \cdot 2^{2k} \cdot n^{1+2^{1-k}} + n \cdot c_n$ comparisons, where $c_n = \min\{\binom{n}{2}, \binom{215}{2}\}$ (as defined in Lemma 4.4).*

Proof. As before, we prove the claim by induction on n. For $n \leq 8$ we use 2-`Sort` which produces sorting with error 2 and gives a suitable bound on the number of comparisons.

We now consider the general case ($n \geq 9$ and $k \geq 3$). As in the case of selection, it is easy to see that the algorithm sorts X with error k. However the bound on the number of comparisons is different from the one we gave for k-`Select` since now the algorithm is called recursively on both X_1 and X_2. As before, the call to $(k-1)$-`Pivot` on $s = \left\lfloor n^{1-2^{1-k}} \right\rfloor$ elements uses at most $9s^{1+1/(2^{k-1}-1)} + c_n \leq 9n + c_n$ comparisons and there are at most $n-1$ comparisons of the pivot with all the other elements. We again assume, without loss of generality, that $|X_1| \geq |X_2|$ and denote $\alpha = (|X_2|+1)/n$. By our inductive hypothesis, the number of comparisons N used for the recursive calls is bounded by

$$
\begin{aligned}
N &\leq 7 \cdot 2^{2k} \left((n-\alpha n)^{1+2^{1-k}} + (\alpha n)^{1+2^{1-k}} \right) + c_n((n-\alpha n) + (\alpha n - 1)) \\
&= 7 \cdot 2^{2k} \cdot n^{1+2^{1-k}} \left((1-\alpha)^{1+2^{1-k}} + \alpha^{1+2^{1-k}} \right) + (n-1)c_n \ . \qquad (1)
\end{aligned}
$$

By differentiating the expression $(1-\alpha)^{1+2^{1-k}} + \alpha^{1+2^{1-k}}$ as a function of α we obtain that it is monotonically decreasing in the interval $[0, 1/2]$ and hence its minimum is attained when α is the smallest. Therefore we can use the lower bound $\alpha \geq n^{-2^{1-k}}/(5 \cdot 2^{k-2})$ to conclude that

$$\begin{aligned}(1-\alpha)^{1+2^{1-k}} + \alpha^{1+2^{1-k}} &\leq (1-\alpha)(1-\alpha)^{2^{1-k}} + \alpha \leq (1-\alpha)(1-2^{1-k}\cdot\alpha) + \alpha \\ &= 1-(1-\alpha)\cdot 2^{1-k}\alpha \leq 1 - \frac{9}{10}\cdot 2^{1-k} n^{-2^{1-k}}/(5\cdot 2^{k-2}) \\ &= 1 - \frac{9}{25}\cdot 2^{2-2k} n^{-2^{1-k}}.\end{aligned}$$

By substituting this bound into equation (1) we obtain that

$$N \leq 7\cdot 2^{2k}\cdot n^{1+2^{1-k}}\left(1 - \frac{9}{25} 2^{2-2k} n^{-2^{1-k}}\right) + (n-1)c_n = 7\cdot 2^{2k}\cdot n^{1+2^{1-k}} + n\cdot c_n - 10n - c_n \ .$$

Therefore altogether the number of comparisons used by k-Sort is at most $7\cdot 2^{2k}\cdot n^{1+2^{1-k}} + n\cdot c_n$. ■

An immediate corollary of Theorem 4.6 is that it is possible to achieve error of no more than $\log\log n$ in close to optimal time.

Corollary 4.7 *There is a sorting algorithm using $O(n\log^2 n)$ comparisons with error of at most $\log\log n$.*

5 Lower Bounds

Here we prove lower bounds against deterministic max-finding, sorting, and selection algorithms. In particular, we show that Theorem 3.6, Theorem 4.5 and Theorem 4.6 achieve almost optimal trade-off between error and number of comparisons.

Our proof is based on the analysis of the comparison graph, or the directed graph on all elements in which an edge (x_i, x_j) is present whenever a comparison between x_i and x_j was made and its imprecise outcome was "$x_i \geq x_j$". We show that one can only conclude that $x_i \geq_k x_j$ if this graph has a path of length at most k from x_i to x_j. The existence of short paths from an element to numerous other elements (such as when the element is a k-max) is only possible when there are many vertices with large out-degree. Following this intuition we define an oracle that when comparing two elements always responds that the one with the smaller out-degree is larger than the one with the larger out-degree. Such an oracle will ensure that a large number of comparisons needs to be made in order to obtain a sufficient number of vertices with high out-degree. We also show that the responses of the oracle can be seen as derived from actual values defined using the resulting comparison graph.

Lemma 5.1 *Suppose a deterministic algorithm A upon given n elements guarantees that after m comparisons it can list r elements, each of which is guaranteed to be k-greater than at least q elements. Then $m = \Omega(\max\{q^{1+1/(2^k-1)}, q\cdot r^{1/(2^{k-1})}\})$.*

Proof. To create a worst case input we first define a strategy for the comparator and later choose values for the elements which are consistent with the given answers, while maximizing the error of the algorithm.

Let G_t be the comparison graph at time t. That is, G_t is a digraph whose vertices are the x_i and which contains the directed edge (x_i, x_j) if and only if before time t a comparison between x_i and x_j has been made, and the comparator has responded with "$x_i \geq x_j$". We denote the out-degree of x_i in G_t by $d_t(x_i)$. Assume that at time t the algorithm wants to compare some x_i and x_j. If $d_t(x_i) \geq d_t(x_j)$ then the comparator responds with "$x_j \geq x_i$", and it responds with "$x_i \geq x_j$" otherwise. (The response is arbitrary when $d_t(x_i) = d_t(x_j)$.) Let x be an element that is declared by A to be k-greater than at least q elements.

Let $\ell_i = \text{dist}(x, x_i)$, where dist gives the length of the shortest (directed) path in the final graph G_m. If no such path exists, we set $\ell_i = n$. After the algorithm is done, we define $\text{val}(x_i) = \ell_i$. We first claim that the values are consistent with the responses of the comparator. If for some pair of elements x_i, x_j the comparator has responded with "$x_i \geq x_j$", then G_m contains edge (x_i, x_j). This implies that for any x, $\text{dist}(x, x_j) \leq \text{dist}(x, x_i) + 1$, or $\ell_i \geq \ell_j - 1$. Therefore the answer "$x_i \geq x_j$" is consistent with the given values.

Consider the nodes x_i that x can reach via a path of length at most k. These are exactly the elements k-smaller than x, and thus there must be at least q of them. For $i \leq k$ let $S_i = \{x_j \mid \ell_j = i\}$ and $s_i = |S_i|$. We claim that for every $i \in [k]$, $m \geq s_i^2/(2s_{i-1}) - s_i/2$. For a node $u \in S_i$, let $\text{pred}(u)$ be a node in S_{i-1} such that the edge $(\text{pred}(u), u)$ is in the graph. For a node $v \in S_{i-1}$, let $S_{i,v} = \{u \in S_i \mid v = \text{pred}(u)\}$. Further, let $d_{\text{out}}(\text{pred}(u), u)$ be the out-degree of $\text{pred}(u)$ when the comparison between $\text{pred}(u)$ and u was made (as a result of which the edge was added to G_m). Note that for any distinct nodes $u, u' \in S_{i,v}$, $d_{\text{out}}(v, u) \neq d_{\text{out}}(v, u')$ since the out-degree of v grows each time an edge to a node in $S_{i,v}$ is added. This implies that

$$\sum_{u \in S_{i,v}} d_{\text{out}}(v, u) \geq \sum_{d \leq |S_{i,v}|-1} d = |S_{i,v}|(|S_{i,v}| - 1)/2 \ .$$

By the definition of our comparator, for every $u \in S_i$, $d_m(u) \geq d_{\text{out}}(\text{pred}(u), u)$. This implies that

$$m \geq \sum_{v \in S_{i-1}} \sum_{u \in S_{i,v}} d_m(u) \geq \sum_{v \in S_{i-1}} \frac{|S_{i,v}|(|S_{i,v}| - 1)}{2} = \frac{\sum_{v \in S_{i-1}} |S_{i,v}|^2 - |S_i|}{2} \ .$$

Using the inequality between the quadratic and arithmetic means,

$$\sum_{v \in S_{i-1}} |S_{i,v}|^2 \geq \left(\sum_{v \in S_{i-1}} |S_{i,v}| \right)^2 / |S_{i-1}| = s_i^2 / s_{i-1}.$$

This implies that $m \geq \frac{s_i^2}{2s_{i-1}} - \frac{s_i}{2}$.

We can therefore conclude that $s_i \leq \sqrt{(2m + s_i)s_{i-1}} \leq \sqrt{3ms_{i-1}}$ since $s_i \leq n \leq m$. By applying this inequality and using the fact that $s_0 = 1$ we obtain that $s_1^2/3 \leq m$ and $s_i \leq 3m \cdot (3m/s_1)^{2^{-(i-1)}}$ for $i > 1$. Since $\sum_{i \leq k} s_i \geq q + 1$, we thus find that $q \leq 12 \cdot m \cdot (3m/s_1)^{2^{-(k-1)}}$. This holds since either

1. $(3m/s_1)^{2^{-(k-1)}} > 1/2$ and then $12 \cdot m \cdot (3m/s_1)^{2^{-(k-1)}} \geq 6m > n$, or
2. $(3m/s_1)^{-2^{-(k-1)}} \leq 1/2$ and then

$$(3m/s_1)^{-2^{-i+1}}/(3m/s_1)^{-2^{-i}} = (3m/s_1)^{-2^{-i}} \leq (3m/s_1)^{-2^{-(k-1)}} \leq 1/2$$

for $i \leq k-1$, where the penultimate inequality holds since $s_1 < 3m$. In this case

$$\begin{aligned} q - s_1 \leq \sum_{i=2}^{k} s_i \leq \sum_{i=2}^{k} (3m)(3m/s_1)^{-2^{-(i-1)}} &\leq \sum_{i \leq k} 2^{i-k}(3m)(3m/s_1)^{-2^{-(k-1)}} \\ &< 2(3m)^{1-2^{-(k-1)}} s_1^{2^{-(k-1)}} \end{aligned}$$

If $s_1 \geq q/2$, then $m = \Omega(q^2)$ since $m \geq s_1^2/3$. Otherwise we have that

$$m \geq (q/4)^{1/(1-2^{-(k-1)})}/(3s_1^{1/(2^{(k-1)}-1)}),$$

implying

$$m = \Omega(\max\{s_1^2, q^{1/(1-2^{-(k-1)})}/s_1^{1/(2^{(k-1)}-1)}\}) = \Omega(q^{1+1/(2^k-1)}),$$

where the final equality can be seen by making the two terms in the max equal.

Also, note that the choice of x amongst the r elements of the theorem statement was arbitrary, and that s_1 is just the out-degree of x. Let $s_{\min}$ be the minimum out-degree amongst the r elements. Then we trivially have $m \geq r \cdot s_{\min}$. Thus, if $s_{\min} \geq q/2$ then $m \geq qr/2$, and otherwise

$$m = \Omega(\max\{r \cdot s_{\min}, q^{1/(1-2^{-(k-1)})}/s_{\min}^{1/(2^{(k-1)}-1)}\}) = \Omega(q \cdot r^{1/(2^{k-1})})$$

where the final equality is again seen by making the two terms in the max equal. ■

From Lemma 5.1 we immediately obtain a lower bound for max-finding by setting $r = 1$, $q = n-1$, and for median-finding and sorting by setting $r = q = n/2$. In general, the sorting lower bound holds for k-order selection of the i^{th} element for any $i = c \cdot n$ for constant $0 < c < 1$.

Theorem 5.2 *Every deterministic max-finding algorithm A with error k requires $\Omega(n^{1+1/(2^k-1)})$ comparisons.*

Theorem 5.2 implies that k-`Sort` and k-`Select` are optimal up to a constant factor for any constant k.

Theorem 5.3 *Every deterministic algorithm A which k-sorts n elements, or finds an element of k-order i for $i = c \cdot n$ with $0 < c < 1$ a constant, requires $\Omega(n^{1+1/2^{k-1}})$ comparisons.*

In addition we obtain that Corollary 3.7 is essentially tight.

Corollary 5.4 *Let A be a deterministic max-finding algorithm that makes $O(n)$ comparisons. Then A has error at least $\log\log n - O(1)$.*

6 Conclusions

We defined a simple and natural model of imprecision in a result of a comparison. The model is inspired by both imprecision in human judgement of values and also by bounded but potentially adversarial errors in sporting tournaments. Our results show that there exist algorithms that are robust to imprecision in comparisons while using substantially fewer comparisons than the naïve methods.

We note that in most of the results substantially tighter constants can be obtained in the bounds using small modifications of the algorithms, more careful counting and optimization for small values of k and n. This would yield algorithms that improve significantly on the naïve approach even for small values of n. We made only a modest effort to improve the constants to make the presentation of the main ideas clearer.

While our lower bounds show that many of the algorithms we give are essentially optimal a number of interesting and natural problems are left open.

1. What is the complexity of deterministic maximum finding with error 2? `2-MaxFind` uses $O(n^{3/2})$ comparisons whereas our lower bound is $\Omega(n^{4/3})$ comparisons. Resolving the case of $k = 2$ is likely to lead to closing of the gap for larger error k.

2. Can error 2 be achieved by a randomized algorithm using $O(n)$ comparisons? `SampledTournament` only guarantees error 3.

3. We have not addressed the complexity of randomized sorting with error k.

References

[1] Gagan Aggarwal, Nir Ailon, Florin Constantin, Eyal Even-Dar, Jon Feldman, Gereon Frahling, Monika R. Henzinger, S. Muthukrishnan, Noam Nisan, Martin Pál, Mark Sandler, and Anastasios Sidiropoulos. Theory research at Google. *SIGACT News*, 39(2):10–28, 2008.

[2] Noga Alon and Yossi Azar. Sorting, approximate sorting, and searching in rounds. *SIAM J. Discrete Math*, 1(3):269–280, 1988.

[3] Shay Assaf and Eli Upfal. Fault tolerant sorting networks. *SIAM J. Discrete Math*, 4(4):472–480, 1991.

[4] Michael Ben-Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 221–230, 2008.

[5] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[6] Béla Bollobás and Andrew Thomason. Parallel sorting. *Discrete Appl. Math.*, 6:1–11, 1983.

[7] Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 130–136, 1993.

[8] H. A. David. *The Method of Paired Comparisons*. Charles Griffin & Company Limited, 2nd edition, 1988.

[9] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5), 1994.

[10] Vitaly Feldman. Robustness of evolvability. In *Proceedings of COLT*, pages 277–292, 2009.

[11] Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. *Theor. Comput. Sci.*, 410:4457–4470, 2009.

[12] Irene Finocchi and Giuseppe F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 101–110, 2004.

[13] William I. Gasarch, Evan Golub, and Clyde P. Kruskal. Constant time parallel sorting: an empirical view. *J. Comput. Syst. Sci.*, 67(1):63–91, 2003.

[14] Roland Häggkvist and Pavol Hell. Parallel sorting with constant time for comparisons. *SIAM J. Comput.*, 10(3):465–472, 1981.

[15] Roland Häggkvist and Pavol Hell. Sorting and merging in rounds. *SIAM Journal on Algebraic and Discrete Methods*, 3(4):465–473, 1982.

[16] Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *SODA*, pages 881–890, 2007.

[17] Michael Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998.

[18] László Lovász. *An Algorithmic Theory of Numbers, Graphs, and Convexity*. CBMS-NSF Regional Conference Series in Applied Mathematics 50, SIAM, 1986.

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

[20] Bala Ravikumar, K. Ganesan, and K. B. Lakshmanan. On selecting the largest element in spite of erroneous information. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 88–99, 1987.

[21] Alfréd Rényi. On a problem in information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 6:505–516, 1962.

[22] Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. Coping with errors in binary search procedures. *J. Comput. Sys. Sci.*, 20(3):396–405, 1980.

[23] Scott M. Smith and Gerald S. Albaum. *Fundamentals of Marketing Research*. Sage Publications, Inc., first edition, 2005.

[24] Louis Leon Thurstone. A law of comparative judgment. *Psychological Review*, 34:273–286, 1927.

[25] Stanisław Marcin Ulam. *Adventures of a Mathematician*. Scribner's, New York, 1976.

[26] Leslie G. Valiant. Parallelism in comparison problems. *SIAM J. Comput.*, 4(3):348–355, 1975.

[27] Andrew Chi-Chih Yao and Frances Foong Yao. On fault-tolerant networks for sorting. *SIAM J. Comput.*, 14(1):120–128, 1985.