Sorting and Selection with Imprecise Comparisons

Miklós Ajtai[1], Vitaly Feldman[1], Avinatan Hassidim[2], and Jelani Nelson[2]

[1] IBM Almaden Research Center, San Jose CA 95120, USA
[2] MIT, Cambridge MA 02139, USA

Abstract. 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 consider a simple model of the 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 δ corresponds to the *just noticeable difference unit (JND)* or *difference threshold* in the psychophysics literature, but does not require the statistical assumptions used to define this value.

In this model, the standard method of paired comparisons minimizes the errors introduced by the imprecise comparisons at the cost of $\binom{n}{2}$ comparisons. 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 tradeoff 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 close-to-optimal solutions for each of these problems.

1 Introduction

Let $x_1, \ldots, x_n$ be n elements where each x_i has an unknown value $\mathrm{val}(x_i)$. 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 $|\mathrm{val}(x_i) - \mathrm{val}(x_j)| \leq \delta$, 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, $\mathrm{val}(x_i) - \mathrm{val}(x_{i^*}) \leq k\delta$ 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 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 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 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 [20, 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 *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[3]. The idea was later generalized by Thurstone to having humans not only compare physical stimuli, but also abstract concepts [21].

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

[3] 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.

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 this scenario.

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 $\Omega(n^{1+1/(2^k-1)})$. 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 problem of sorting. For $k = 2$, we give an algorithm using $4 \cdot n^{3/2}$ comparisons. For general k, we show $O((n^{1+1/(3\cdot 2^{\lfloor k/2 \rfloor -1}-1)} + nk)\log n)$ comparisons is achievable, and we show a lower bound of $\Omega(n^{1+1/2^{k-1}})$ comparisons. When $k = O(1)$, or if only a single element of specific order needs to be selected, the $\log n$ factor disappears from our upper bound. 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 $2\log\log n$ error for sorting in $O(n \log n)$ comparisons. Standard methods using the same number of comparisons (e.g. a binary tournament tree, or Mergesort) can be shown to incur at least $\log n$ error. 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 main idea in our deterministic upper bounds for both max-finding and selection is to develop efficient algorithms for a small value of k ($k = 2$), then for larger k show how to partition elements, recursively use algorithms for smaller k, 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 [6] 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 subsampling at each level of the tournament tree. By performing a round-robin tournament on the top few tournament players together with the subsampled elements, we obtain an element of value at least $x^* - 3\delta$ with high probability.

To obtain lower bounds we translate our problems into problems on directed graphs 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.

Some of the proofs are omitted from this extended abstract and appear in the full version of the paper.

1.2 Related Work

Handling noise in binary search procedures was first considered by Rényi [18] and by Ulam [22]. An algorithm for solving Ulam's game was proposed by Rivest et. al. in [19], 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 [24] 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 [17]. 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 [11] 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 [10].

In the model where each comparison is incorrect with some probability p, Feige et al. [9] and Assaf and Upfal [2] give algorithms for several comparison problems, and [3, 15] give algorithms for binary search. We refer the reader interested in the history of faulty comparison problems to a survey of Pelc [16].

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 [23]. 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) [23], 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 [14] 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. For example, sorting in k parallel rounds $\Omega(n^{1+1/k})$ comparisons are required [5, 13], 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 [12].

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 ith element and its value, e.g. $x_i > x_j$ should be interpreted as $\text{val}(x_i) > \text{val}(x_j)$.

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 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. We say an element is a *k-max* of a set if it is k-greater than all other elements in the set, and a permutation $x_{\pi(1)}, \ldots, x_{\pi(n)}$ is *k-sorted* if $x_{\pi(i)} \geq_k x_{\pi(j)}$ for every $i > j$.

All logarithms throughout this document are base-2. For simplicity of presentation, we frequently 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 1. *Sorting according to the number of wins in a round-robin tournament yields error* 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 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.

Algorithm A_2: `// 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 set of s candidate elements and play them in a round-robin tournament. Let x have the most number of wins.
   (b) Compare x against all candidate elements and eliminate all elements that lose to x.
3. Play the final at most s candidate elements in a round-robin tournament and return the element with the most wins.

Fig. 1. The algorithm A_2 for finding a 2-max.

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

Lemma 1. *The max-finding algorithm A_2 has error 2 and makes at most $ns + n^2/s$ comparisons. In particular, the number of comparisons is at most $2n^{3/2}$ for $s = \lceil\sqrt{n}\rceil$.*

Proof. We analyze the error in two cases. If x^* is never eliminated then x^* participates in Step 3. Theorem 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 $x^* - 2$. In each step at least $(s-1)/2$ elements are eliminated implying the given bound on the total number of comparisons.

The key recursion step of our general error max-finding is the algorithm 1-Cover of Lemma 3 which is based on A_2 and the following lemma.

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

The algorithm performs a round-robin tournament and then iteratively greedily picks an element which defeats as many thus-far undefeated elements as possible. We now obtain 1-Cover by setting $s = \lceil\sqrt{n}\rceil/8$ 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 2 on the elements in the final tournament in Step 3.

Lemma 3. *There is an algorithm 1-Cover making $O(n^{3/2})$ comparisons which finds a 1-max-set of size at most $\sqrt{n}/4$ (for sufficiently large n).*

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

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

1. **return** $A_2(A'_{k-1}(x_1, x_2, \ldots, x_n))$

Algorithm A'_k: `// Returns a` $(k-1)$`-max set of size` $O(n^{2^k/(3\cdot 2^k-4)})$ `for` $k \geq 2$

1. **if** $k = 2$, return 1-Cover$(x_1, x_2, \ldots, x_n)$.
2. **else**
   (a) Equipartition the n elements into $t = b_k n^{2^{k-1}/(2^k-4/3)}$ sets $S_1, \ldots, S_t$.
   (b) Recursively call A'_{k-1} on each set S_i to recover a $(k-2)$-max set T_i.
   (c) Return the output of 1-Cover with $\cup_{i=1}^{t} T_i$ as input.

Fig. 2. The algorithm A_k for finding a k-max based on a recursive algorithm A'_k for finding a $(k-1)$-max-set. The value b_k is $(1/2) \cdot (3/4)^{k-3}$ for $k \leq 10$ and $2^{-(3/4)^{k+5}(3\cdot 2^{k-1}-4)/4}$ otherwise.

Theorem 3. *For every $3 \leq k \leq \log\log n$, there exists an algorithm A_k finds a k-max element using $O(n^{1+1/((3/4)2^k-1)})$ comparisons.*

Corollary 1. *There is a max-finding algorithm using $O(n)$ comparisons with error $\log\log n$.*

3.2 Randomized Max-Finding

We now show that randomization can significantly reduce the number of comparisons required to find an approximate maximum. We emphasize that although an adversary is not allowed to adaptively change the input values during the course of an algorithm's execution, the adversary can adaptively choose how to err when two elements are close. In particular, the classic randomized selection algorithm can take quadratic time since for an input with all equal values, the adversary can claim that the randomly chosen pivot is smaller than all other elements. Nevertheless, we show the following.

Theorem 4. *There exists a linear-time randomized algorithm which finds a 3-max with probability at least $1 - n^{-c}$ for any constant c and n large enough.*

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 was not the absolute loser in their set. We also randomly subsample a set of players 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 subsampled with high probability, or (2) x^* makes it as one of the top few tournament players with high probability. We describe the

properties of the tournament in Lemma 4. In Figure 3 we present the subroutine SampledTournament for the tournament.

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

1. Initialize $N_0 \leftarrow \{x_1, \ldots, x_n\}$, $W \leftarrow \emptyset$, and $i \leftarrow 0$.
2. **if** $|N_i| \leq n^{0.3}$, insert N_i into W and **return** W.
3. **else** randomly sample $n^{0.3}$ elements from N_i and insert them into W.
4. Randomly partition the elements in N_i into sets of size $80(c+2)$. In each set, perform a round-robin tournament to find the minimal element (the element with the fewest wins, with ties broken arbitrarily).
5. Let N_{i+1} contain all of N_i except for the minimal elements found in Step 4. That is, from each set of $80(c+2)$ elements of N_i, only one does not belong to N_{i+1}. Increment i and **goto** Step 2.

Fig. 3. The algorithm SampledTournament.

Lemma 4. SampledTournament *outputs a* W *of size* $O(n^{0.3} \log n)$ *after* $O(n)$ *comparisons such that* W *is a* 1*-max-set with probability at least* $1 - n^{-c}$.

Theorem 4 follows immediately from Lemma 4: run the algorithm SampledTournament, then return the winner of W in a round-robin tournament.

4 Sorting and Selection

Definition 1. *Element* x_j *in the set* $x_1, \ldots, x_n$ *is of* k-order i *if there exists a partition* S_1, S_2 *of* $[n]$ *with* $j \in S_1$, $|S_1| = i$, $x_\ell \leq_k x_j$ *for all* $\ell \in S_1$, *and* $x_\ell \leq_k x_{\ell'}$ *for all* $\ell \in S_1, \ell' \in S_2$. *A* k-median *is an element of* k*-order* $\lfloor n/2 \rfloor$.

Our sorting and selection algorithms are based on the following lemma.

Lemma 5. *In a round-robin tournament on* n *elements, the element with the median number of wins has at least* $(n-2)/4$ *wins and at least* $(n-2)/4$ *losses.*

We can now obtain an error-2 sorting algorithm B_2 which needs only $4 \cdot n^{3/2}$ comparisons. The idea is to modify A_2 so that the x found in Step 2(a) of Figure 1 is a pivot in the sense of Lemma 5. We then compare this x against all elements and pivot into two sets, recursively sort each, then concatenate.

Lemma 6. *There is a deterministic sorting algorithm* B_2 *with error* 2 *that requires at most* $4 \cdot n^{3/2}$ *comparisons.*

At a high level our algorithm for k-order selection is similar to the classical selection algorithm of Blum et al. [4], 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

Algorithm C_k: `// Returns an element of k-order i.`

1. **if** $k \leq 3$, sort the elements using B_2 then return the element with index i.
2. **else**
   (a) Set $k' = \lfloor k/2 \rfloor + 1$.
   (b) Equipartition the n elements into $t = b_{k'} n^{2^{k'-1}/(2^{k'}-4/3)}$ sets $S_1, \ldots, S_t$.
   (c) Recursively call C_{k-2} on each set S_i to obtain a $(k-2)$-median y_i.
   (d) Play the $y_1, \ldots, y_t$ in a round-robin tournament and let y be the element with the median number of wins.
   (e) Compare y with each of the other $n-1$ elements. If y defeats at least $(n-1)/2$ elements then let X_2 be a set of $d = (t-2)/4 \cdot ((n/t)-1)/2$ elements $(k-1)$-greater than y, and let X_1 be the set of at least $(n-1)/2 - d$ elements that y defeats which are not in X_2 (thus every $(x, x') \in (X_1 \cup \{y\}) \times X_2$ satisfies $x' \geq_k x$). If y defeats less than $(n-1)/2$ elements, X_1 and X_2 are defined symmetrically.
      i. **if** $|X_1| = i - 1$, return i.
      ii. **else if** $i \leq |X_1|$, recursively find an element of k-order i in X_1.
      iii. **else** recursively find an element of k-order $(i - |X_1| - 1)$ in X_2.

Fig. 4. The algorithm C_k. The value $b_{k'}$ is chosen as in the algorithm $A'_{k'}$ (see Figure 2).

scenario though, a good pivot must not only partition the input into nearly equal-sized chunks, but must itself be of $(k-2)$-order $c \cdot n$ for some constant $0 < c < 1$. The base case $k = 2$ can be solved by Lemma 6 since the element placed in position i of the sorted permutation is of 2-order i. Our algorithm is given in Figure 4.

Lemma 7. *For any $i \in [n]$ and $2 \leq k \leq 2 \log\log n$, the deterministic algorithm C_k finds an element of k-order i in $O(n^{1+1/(3 \cdot 2^{\lfloor k/2 \rfloor - 1} - 1)})$ comparisons.*

Theorem 5. *For any $2 \leq k \leq 2 \log\log n$, there is a deterministic sorting algorithm B_k with error k using $O((n^{1+1/(3 \cdot 2^{\lfloor k/2 \rfloor - 1} - 1)} + nk) \log n)$ comparisons. If $k = O(1)$, the number of comparisons reduces to $O(n^{1+1/(3 \cdot 2^{\lfloor k/2 \rfloor - 1} - 1)})$.*

Proof. We find a k-median using C_k, equipartition the elements into sets S_1, S_2 such that every element of S_2 is k-greater than every element of $S_1 \cup \{x\}$, recursively sort each partition, then concatenate the sorted results. The upper bound on the number of comparisons follows from the Master theorem (see [7, Theorem 4.1]), and correctness is immediate from the definition of a k-median.

5 Lower Bounds

Here we prove lower bounds against deterministic max-finding, sorting, and selection algorithms. In particular, we show that Theorem 3 and Theorem 5 achieve almost optimal trade-off between error and number of comparisons.

Lemma 8. *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. We define a comparator that decides how to answer queries online in such a way that we can 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 $y_i = \mathrm{dist}(x, x_i)$, where dist gives the length of the shortest (directed) path in the final graph G_m. If no such path exists, we set $y_i = n$. After the algorithm is done, we define $\mathrm{val}(x_i) = y_i$, We first claim that the values are consistent with the responses of the comparator. If for some pair of objects 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, $\mathrm{dist}(x, x_j) \leq \mathrm{dist}(x, x_i) + 1$, or $y_i \geq y_j - 1$. 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 $S_i = \{x_j | y_j = i\}$ 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 $\mathrm{pred}(u)$ be a node in S_{i-1} such that the edge $(\mathrm{pred}(u), u)$ is in the graph. For a node $v \in S_{i-1}$, let $S_{i,v} = \{u \in S_i \mid v = \mathrm{pred}(u)\}$. Further, let $\mathrm{d}_o(\mathrm{pred}(u), u)$ be the out-degree of $\mathrm{pred}(u)$ when the comparison between $\mathrm{pred}(u)$ and u was made (as a result of which the edge was added to G_m). Note that for any distinct nodes $u, u' \in S_{i,v}$, $\mathrm{d}_o(v, u) \neq \mathrm{d}_o(v, u')$ since the out-degree of v grows each time an edge to a node in $S_{i,v}$ is added. This implies that

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

By the definition of our comparator, for every $u \in S_i$, $d_m(u) \geq \mathrm{d}_o(\mathrm{pred}(u), u)$. 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

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

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 8 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 ith element for any $i = c \cdot n$ for constant $0 < c < 1$.

Theorem 6. *Every deterministic max-finding algorithm A with error k requires $\Omega(n^{1+1/(2^k-1)})$ comparisons.*

Theorem 7. *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.*

Theorem 6 implies the following, showing that Corollary 1 is tight.

Corollary 2. *Let A be a deterministic max-finding algorithm that makes $O(n)$ comparisons. Then A has error at least $\log\log n - O(1)$.*

References

1. G. Aggarwal, N. Ailon, F. Constantin, E. Even-Dar, J. Feldman, G. Frahling, M. R. Henzinger, S. Muthukrishnan, N. Nisan, M. Pál, M. Sandler, and A. Sidiropoulos. Theory research at Google. *SIGACT News*, 39(2):10–28, 2008.
2. S. Assaf and E. Upfal. Fault tolerant sorting networks. *SIAM J. Discrete Math*, 4(4):472–480, 1991.
3. M. Ben-Or and A. Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *FOCS*, pages 221–230, 2008.
4. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.
5. B. Bollobás and A. Thomason. Parallel sorting. *Discrete Appl. Math.*, 6:1–11, 1983.
6. R. S. Borgstrom and S. R. Kosaraju. Comparison-based search in the presence of errors. In *STOC*, pages 130–136, 1993.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.
8. H. A. David. *The Method of Paired Comparisons*. Charles Griffin & Company Limited, 2nd edition, 1988.
9. U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5), 1994.
10. I. Finocchi, F. Grandoni, and G. F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. In *ICALP*, pages 286–298, 2006.
11. I. Finocchi and G. F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *STOC*, pages 101–110, 2004.
12. W. I. Gasarch, E. Golub, and C. P. Kruskal. Constant time parallel sorting: an empirical view. *J. Comput. Syst. Sci.*, 67(1):63–91, 2003.
13. R. Häggkvist and P. Hell. Parallel sorting with constant time for comparisons. *SIAM J. Comput.*, 10(3):465–472, 1981.
14. R. Häggkvist and P. Hell. Sorting and merging in rounds. *SIAM Journal on Algebraic and Discrete Methods*, 3(4):465–473, 1982.
15. R. M. Karp and R. Kleinberg. Noisy binary search and its applications. In *SODA*, pages 881–890, 2007.
16. A. Pelc. Searching games with errors—fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.
17. B. Ravikumar, K. Ganesan, and K. B. Lakshmanan. On selecting the largest element in spite of erroneous information. In *STACS*, pages 88–99, 1987.
18. A. Rényi. On a problem in information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 6:505–516, 1962.
19. R. L. Rivest, A. R. Meyer, D. J. Kleitman, K. Winklmann, and J. Spencer. Coping with errors in binary search procedures. *J. Comput. Sys. Sci.*, 20(3):396–405, 1980.
20. S. M. Smith and G. S. Albaum. *Fundamentals of Marketing Research*. Sage Publications, Inc., first edition, 2005.
21. L. L. Thurstone. A law of comparative judgment. *Psychological Review*, 34:273–286, 1927.
22. S. M. Ulam. *Adventures of a Mathematician*. Scribner's, New York, 1976.
23. L. G. Valiant. Parallelism in comparison problems. *SIAM J. Comput.*, 4(3):348–355, 1975.
24. A. C. Yao and F. F. Yao. On fault-tolerant networks for sorting. *SIAM J. Comput.*, 14(1):120–128, 1985.