A Nearly Optimal Algorithm for Approximate Minimum Selection with Unreliable Comparisons∗

Stefano Leucci1 and Chih-Hung Liu2

1 Department of Computer Science, ETH Zürich, Switzerland. stefano.leucci@inf.ethz.ch
2 Department of Computer Science, ETH Zürich, Switzerland. chih-hung.liu@inf.ethz.ch

Abstract
We consider the approximate minimum selection problem in presence of independent random comparison faults. This problem asks to select one of the smallest k elements in a linearly-ordered collection of n elements by only performing unreliable pairwise comparisons: whenever two elements are compared, there is constant probability that the wrong answer is returned.

We design a randomized algorithm that solves this problem with high probability (w.h.p.) for the whole range of values of k using $O(\log n \cdot \left(\frac{n}{k} + \log \log \log n\right))$ expected time. Then, we prove that the expected running time of any algorithm that succeeds w.h.p. must be $\Omega\left(\frac{n}{k} \log n\right)$, thus implying that our algorithm is optimal, in expectation, for almost all values of k (and it is optimal up to triple-log factors for $k = \omega\left(\frac{n}{\log \log \log n}\right)$). These results are quite surprising in the sense that for $k$ between $\Omega(\log n)$ and $c \cdot n$, for any constant $c < 1$, the expected running time must still be $\Omega\left(\frac{n}{k} \log n\right)$ even in absence of comparison faults. Informally speaking, we show how to deal with comparison errors without any substantial complexity penalty w.r.t. the fault-free case. Moreover, we prove that as soon as $k = O\left(\frac{n}{\log \log n}\right)$, it is possible to achieve a worst-case running time of $O\left(\frac{n}{k} \log n\right)$.

1998 ACM Subject Classification F.2.2 Sorting and searching

Keywords and phrases approximate minimum; comparison errors; unreliable comparisons

1 Introduction

In an ideal world, computational tasks are always carried out reliably, i.e., every operation performed by an algorithm behaves exactly as intended. Practical architectures, however, are error-prone and even basic operations can sometimes return the wrong results, especially when large-scale systems are involved. When dealing with these spurious results, the first instinct is that of trying to detect and correct the errors as they manifest, so that the problems of interest can then be solved using classical (non fault-tolerant) algorithms. An alternative approach deliberately allows errors to interfere with the execution of an algorithm, in the hope that the computed solution will still be good, at least in an approximate sense. This begs the question: is it possible to devise algorithms that cope with faults by design and return probably good solutions?

We investigate this question by considering a generalization of the fundamental problem of finding the minimum element in a totally-ordered set: in the fault-tolerant approximate minimum selection problem (FT-Min(k) for short) we wish to return one of the smallest k elements in a collection of size n using only unreliable pairwise comparisons, i.e., comparisons

∗ Research supported by SNF (project number 200021_165524).
in which the result can sometimes be incorrect due to errors. This allows, for example, to find a representative in the top percentile of the input set, or to obtain a good estimate of the minimum from a set of noisy observations.

In this paper, we provide both upper and lower bounds on the running time of any (possibly randomized) algorithm that solved FT-Min\((k)\) with high probability (w.h.p.).\(^1\) Our algorithms are nearly optimal in the sense that it is not possible to solve FT-Min\((k)\) w.h.p. using an asymptotically lower expected number of comparisons, up to a \(\log\log\log n\)-factor for a small range of the parameter \(k\).

Our results find application in any setting that is subject to random comparison errors (e.g., due to communication interferences, alpha particles, charge collection, cosmic rays\([4,29]\), or energy-efficient architectures where the energy consumed by the computation can be substantially reduced if a small percentage of faulty results is allowed\([2,8,9,25]\), or in which performing accurate comparisons is too resource-consuming (think, e.g., of the elements as references to remotely stored records) while approximate comparisons can be carried out much quicker.

Before presenting our results in more detail, let us briefly discuss the error model we use.

### 1.1 The Comparison Model

We consider independent random comparison faults: a simple and natural error model in which there exists a true strict ordering relation among the \(n\) input elements, but algorithms are only allowed to gather information about such a relation via unreliable comparisons between pairs of elements. The outcome of a comparison involving two distinct elements \(x\) and \(y\) can be either “\(<\)” or “\(>\)” to signify that \(x\) is reported to be “smaller than” or “larger than” \(y\), respectively. Most of the times the outcome of a comparison will correspond to the true relative order of the compared elements, but there is a constant probability \(p < \frac{1}{2}\) that the wrong result will be reported instead. An algorithm can compare the same pair of elements more than once. When this happens the outcome of each comparison is chosen independently of the previous results. In a similar way, comparisons involving different pairs of elements are also assumed to be independent.

The above error model was first considered in the 80s and 90s when the related problems of finding the minimum, selecting the \(k\)th smallest element, and of sorting a sequence have been studied\([13,26,27]\). The best solutions are due to Feige et al.\([13]\), who provided optimal Monte Carlo algorithms having a success probability of \(1 - q\) and requiring \(O\left(n \log \frac{1}{q}\right)\), \(O\left(n \log \frac{\min(k,n-k)}{q}\right)\) and \(O\left(n \log \frac{n}{q}\right)\) time, respectively. In the sequel we will invoke the minimum finding algorithm of \([13]\)—which we name \(\text{FindMin}\)—as a subroutine. We therefore find convenient to summarize its performances in the following:

\begin{itemize}
  \item \textbf{Theorem 1 \([13]\).} Given a set \(S\) of \(n\) elements and a parameter \(q \in (0, 1/2]\), Algorithm \(\text{FindMin}\) returns, in \(O\left(n \log \frac{1}{q}\right)\) time, the minimum of \(S\) with a probability of at least \(1 - q\).
\end{itemize}

### 1.2 Our Contributions

We develop a randomized algorithm that solves FT-Min\((k)\) with high probability and in \(O\left(\frac{k}{\pi} \log n + (\log n) \log \log \log n\right)\) expected time for the whole range of values of \(k \in [1, n - 1]\). Moreover, we prove that any algorithm that solves FT-Min\((k)\) w.h.p. requires \(\Omega\left(\frac{k}{\pi} \log n\right)\)

\(^1\) We use the term with high probability to refer to probabilities of at least \(1 - \frac{1}{n}\).
comparisons in expectation, and thus the expected running time of our algorithm is optimal as soon as $k = O\left(\frac{n}{\log \log n}\right)$. These results are quite surprising since for $k = \omega(\log n)$ the expected running time must still be $\Omega\left(\frac{n}{k} \log n\right)$ even in absence of comparison faults (indeed, any random subset of $O\left(\frac{n}{k} \log n\right)$ elements does not contain any of smallest $k$ elements with a probability larger than $\frac{1}{n}$). In other words, comparison errors almost do not increase the computational complexity of the approximate minimum selection problem. In addition, as soon as $k = O\left(\frac{n}{\log \log n}\right)$, we can solve FT-Min($k$) w.h.p. in the optimal worst-case running time of $O\left(\frac{n}{k} \log n\right)$.

Another way to evaluate algorithms for FT-Min($k$) is to consider the range of values of $k$ that they are able to handle, w.h.p., within a given limit $T$ on their running time. For example, if $T = O(n)$, a natural $O\left(\frac{n}{k} \log^2 n\right)$-time algorithm that executes $\text{FindMin}$ with $q = O\left(\frac{1}{k}\right)$ on a randomly chosen subset of $O\left(\frac{n}{k} \log n\right)$ elements only works for $k = \Omega(\log^2 n)$, while our algorithm works for any $k = \Omega(\log n)$, thus exhibiting a quadratic difference in w.r.t. the smallest achievable $k$. More importantly, when $T$ is $o(\log^2 n)$, the natural algorithm cannot provide any bound on the rank of the returned element w.h.p., while our algorithm yields an asymptotically optimal upper bound of $O\left(\frac{n}{k} \log n\right)$ as long as $T = \Omega((\log n) \log \log \log n)$.

We obtain the nearly optimal running time in four stages. First, we design an $O\left(\frac{n}{k} \log n\right)$-time reduction that transforms the problem of solving FT-Min($k$) w.h.p. into the problem of solving FT-Min($\frac{3}{4}n$) with exponentially high probability (w.e.h.p), i.e., with a probability of at least $1 - c^{-n}$ for some constant $c > 1$. This reduction proves that if FT-Min($\frac{3}{4}n$) can be solved w.e.h.p. in $T(n)$ time, then FT-Min($k$) can be solved w.h.p. in $O\left(\frac{n}{k} \log n\right) + T(\Theta(\log n))$ time. Since $\text{FindMin}$ solves FT-Min($\frac{3}{4}n$) in $O(n^2)$ time w.e.h.p., this already matches the optimal running time for $k = O\left(\frac{n}{\log n}\right)$.

To extend the range of $k$, we develop an algorithm for FT-Min($\frac{3}{4}n$) that is reminiscent of knockout-style tournaments and requires only $O(n \log n)$ time, thus reducing the time required to solve FT-Min($k$) to $O\left(\frac{n}{k} \log n + (\log n) \log \log n\right)$ in the worst-case, which is optimal for $k = O\left(\frac{n}{\log \log n}\right)$.

To further improve the running time, we first consider a seemingly simpler fault-tolerant retrieval problem that we name FT-Retrieval($k$): given a collection of $n$ elements (e.g., web-pages), $k$ of which are relevant, and an oracle $\mathcal{O}$ that quickly determines whether an element is relevant with a probability of error of at most $p$, we wish to locate one of the relevant elements. Notice that, in this problem, comparisons between elements are no longer allowed but, rather, an algorithm (e.g., a search engine) can only gather information on the elements through queries to $\mathcal{O}$. FT-Retrieval($k$) can be solved w.h.p. by using a multi-phase process: elements advance from one phase to the next by passing tests with exponentially decreasing error probabilities. This method requires $O\left(\frac{n}{k} \log n\right)$ worst-case time, and this is optimal since our lower bound for FT-Min($k$) also applies to the retrieval problem. Interestingly, this is a special case of the multi-armed bandit problem (MABP)\footnote{12}: given a set of $n$ arms, each with an unknown stochastic reward, MABP asks to find an approximation of the arm with the highest expected reward (as we discuss in Section 3).

Finally, we combine the previous two techniques to solve FT-Min($\frac{3}{4}n$) w.e.h.p. in expected $O(n \log n)$ time: we first use a truncated version of our knockout-tournament to pre-select a suitable set of $O\left(\frac{n}{\log n}\right)$ elements, and then we use (a modified version of) the multi-phase process on these elements. Thanks to our reduction, this results in an algorithm for solving FT-Min($k$) w.h.p. in $O\left(\frac{n}{k} \log n + (\log n) \log \log n\right)$ expected time.

All probabilistic techniques we employ are simple, nonetheless we believe that their combination to achieve the nearly-optimal running time is not straightforward. To some extent, we show how to consolidate simple probabilistic techniques to deal with independent
comparison faults. One remaining open problem is that of obtaining the optimal worst-case running time for \( k = \omega \left( \frac{n}{\log \log n} \right) \). This would provide an answer to the following question: can comparison faults be handled within the same (asymptotic) time as retrieval faults, even for sub-logarithmic running times?

### 1.3 Other Related Works

The problem of finding the exact minimum of a collection of elements using unreliable comparisons had already received attention back in 1987 when Ravikumar et al. [28] considered the variant in which only up to \( f \) comparisons can fail and they proved that \( \Theta(fn) \) comparisons are needed in the worst case. Notice that, in our case, \( f = \Omega \left( \frac{n}{k} \right) \) in expectation since a \((1/k)\)-fraction of the elements must be compared and each comparison fails with constant probability. In [1], Aigner considered a prefix-bounded probability of error \( p < \frac{1}{2} \): at any point during the execution of an algorithm, at most a \( p \)-fraction of the past comparisons can have failed. Here, the situation significantly worsens as up to \( \Theta \left( \frac{1}{1-p} \right)^n \) comparisons might be necessary to find the minimum (and this is tight). Moreover, if the fraction of erroneous comparisons is globally bounded by \( \rho \), and \( \rho = \Omega \left( \frac{1}{n} \right) \), then Aigner also proved that no algorithm can succeed with certainty [1]. The landscape improves when we assume that errors occur independently at random: in addition to the already-cited \( O(n \log \frac{1}{p}) \)-time algorithm by Feige et al. [13] (see Section 1.1), a recent paper by Braverman et al. [6] also considered the round complexity and the number of comparisons needed by partition and selection algorithms. The results in [6] imply that, for constant error probabilities, \( \Theta(n \log n) \) comparisons are needed by any algorithm that selects the minimum w.h.p.

Recently, Chen et al. [10] focused on computing the smallest \( k \) elements given \( r \) independent noisy comparisons between each pairs of elements. For this problem, in a more general error model, they provide a tight algorithm that requires at most \( O(\sqrt{n \text{polylog} n}) \) times as many samples as the best possible algorithm that achieves the same success probability.

If we turn our attention to the related problem of sorting with faults, then \( \Omega(n \log n + fn) \) comparisons are needed to correctly sort \( n \) elements when up to \( f \) comparisons can return the wrong answer, and this is tight [3,20,22]. In the prefix-bounded model, the result in [1] on selecting the minimum also implies that \( \left( \frac{1}{1-p} \right)^{O(n \log n)} \) comparisons are sufficient for sorting, while a lower bound of \( \Omega \left( \left( \frac{1}{1-p} \right)^n \right) \) holds even for the easier problem of checking whether the input elements are already sorted [5]. The problem of sorting when faults are permanent (or, equivalently, when a pair of elements can only be compared once) has also been extensively studied and it exhibits connections to both the rank aggregation problem and to the minimum feedback arc set [6,7,14–16,18,19,21,23]. For more related problems on the aforementioned and other fault models, we refer the interested reader to [27] for a survey and to [11] for a monograph.

Finally, we briefly discuss the fault-free case. For the approximate minimum selection problem, a simple sampling strategy allows to find, w.h.p., one of the smallest \( k \) elements in \( O(\min \{n, n \log n \}) \) time.

### 1.4 Paper Organization

In Section 2 we give some preliminary remarks and we outline a simple strategy to reduce the error probability. In Section 3 we prove our lower bounds, while Section 4 and Section 5 are devoted to our reduction and to the knock-out tournament method, respectively. Finally, in Section 6 and Section 7 we describe our (nearly)-optimal algorithms for \( \text{FT-Retrieval}(k) \) and \( \text{FT-Min}(k) \). Most of the proofs are moved to the Appendix.
2 Preliminaries

We will often draw elements from the input set into one or more (multi)sets using sampling with replacement, i.e., we allow multiple copies of an element to appear in the same multiset. We will then perform comparisons among the elements of these multisets as if they were all distinct: when two copies of the same element are compared, we break the tie using any arbitrary (but consistent) ordering among the copies.

According to our error model, comparison (resp. query) faults happen independently at random with probability at most $p < \frac{1}{2}$. This probability can be reduced by using a simple majority strategy, as shown in the following:

Lemma 2. Let $x$ and $y$ be two distinct elements. For any fixed error probability $p \in [0, \frac{1}{2})$ there exists a constant $c_p \in \mathbb{N}$ such that the strategy that compares $x$ and $y$ (resp. queries $x$) $2c_p \cdot t + 1$ times and returns the majority result is correct with probability at least $1 - e^{-t}$.

3 Lower Bounds

In this section we derive a lower bound of $\Omega\left(\frac{n}{k} \log n\right)$ to the expected number of queries (and hence to the running time) of any algorithm that solves FT-Retrieval($k$) w.h.p. and we show that this implies an analogous lower bound on the number of comparison for FT-Min($k$). Our proof can be seen as a generalization of the lower bound of [13] for computing the or-function of a set of bits. The high-level idea is that of constructing a set of instances containing exactly $k$ relevant elements in such a way that the (non-)relevance of most of the elements is preserved among instances, yet the relevant elements are well spread. Intuitively, any fault-tolerant algorithm must distinguish between those similar input instances, and since two instances might appear to be the same due to errors, the algorithm needs to perform enough queries to achieve the desired success probability. The main technical difficulty is that algorithms can be adaptive, i.e., they can select which element to query as a function of the previous outcomes.

Theorem 3. The expected number of queries of any (possibly randomized, possibly adaptive) algorithm that solves FT-Retrieval($k$) w.h.p., for $1 \leq k \leq c \cdot n$ and any constant $c < 1$, is $\Omega\left(\frac{n}{k} \log n\right)$.

Proof. Let $A$ be an algorithm that returns (on every input instance) one of the $k$ relevant elements in a set $S$ of $n$ elements with probability at least $1 - \frac{1}{n}$. We consider a binary decision tree $T$ associated with $A$ (see Figure. 1 (b)): each internal vertex of $T$ is either a random-choice vertex or a query vertex; each of the two children of a random-choice vertex corresponds to the outcome of a coin flip. A query vertex, say $v$, is associated with an index $j \in \{1, \ldots, n\}$ and represents a query operation on the $j^{th}$ element in $A$’s input. The two children of $v$ correspond to the two different outcomes of the operation. Each of the leaves of $T$ is associated with the index of the element of $S$ returned by $A$ (notice that there might be multiple leaves associated with the same position, and thus to the same element). There is a one-to-one correspondence between the possible executions of $A$ and the set of root-to-leaf paths in $T$, hence the maximum number of query vertices in such a path is a lower bound on the worst-case number of queries of $A$. Similarly, the average number $\overline{h}$ of query vertices traversed by an execution of $A$, on a given instance, is a lower bound on the expected number of queries of $A$ (on that instance). In what follows we will focus on lower-bounding $\overline{h}$.

We start by defining a class $\mathcal{I} = \{I_1, \ldots, I_n\}$ of $n$ instances: the first instance $I_1$ consists of $k$ relevant elements followed by $n - k$ non-relevant elements, while $I_i$ for $i \geq 2$ is obtained
by shifting the elements of $I_1$ by $i - 1$ positions to the right, in a modular fashion (see Figure 1(a)). More precisely, the $j$th element of the $I_i$ is a relevant iff $(j - i) \mod n < k$. We call $A(I_i)$ the random variable representing the path in $T$ corresponding to an execution of $A$ with input $I_i$, and let $\overline{h}$ be the average number of query vertices traversed by $A(I_1)$.

Let $x$ be a leaf in $T$ and let $P_x$ be the path from the root of $T$ to $x$. Let $\langle u_1, u_2, \ldots \rangle$ be the query vertices traversed by $P$ where $u_j$ queries the element in position $x_j$ in the input instance (see Figure 1(b), and notice that $x_j$'s are not necessarily distinct).

Let $B_i = \{1, \ldots, k\} \triangle \{1+(i-1) \mod n, 1+(i \mod n), 1+(i+1 \mod n), \ldots, 1+(i+k-2) \mod n\}$ (where $\triangle$ denotes the symmetric difference between two sets) and notice that, for $i > 1$, we have that if $x_j$ does not belong to $B_i$ then the element in position $x_j$ has the same relevance in both $I_i$ and $I_1$. It follows that, once both $A(I_j)$ and $A(I_1)$ reach vertex $u_j$, they have the same probability of continuing towards the next vertex in $P_x$. On the contrary, if $x_j$ is in $B_i$, then the element in position $x_j$ has a different relevance in $I_i$ and $I_1$, and we say that $u_j$ is a bad query vertex for $I_i$. In this case, we have that the aforementioned probabilities differ by a factor of either $p/(1 - p)$ or $(1 - p)/p$. Notice also that random-choice vertices in $P_x$ do not affect the relative probabilities of the events "$A(I_j) = P_x$" and "$A(I_1) = P_x$".

Let $\gamma(i, x)$ be the number of bad query vertices for $I_i$ in the path $P_x$ and let $i \geq 1$. From the previous discussion: $\Pr(A(I_i) = P_x) \geq \left(\frac{p}{1-p}\right)^{\gamma(i,x)} \Pr(A(I_1) = P_x)$ where $p/(1-p) < 1$ since $p < \frac{1}{2}$.

Let $L$ be the set of leaves in $T$ and let $L_i$ be the set of leaves of $T$ that correspond to non-relevant elements in $I_i$. Since $A$ succeeds with probability at least $1 - \frac{1}{n}$, we must have: $\sum_{i=1}^n \sum_{x \in L_i} P(A(I_i) = P_x) \leq \sum_{i=1}^n \frac{1}{n} = 1$. Hence, defining $H_x = \{i : x \in L_i\}$, we have:

$$1 \geq \sum_{i=1}^n \sum_{x \in L_i} P(A(I_i) = P_x) \geq \sum_{i=1}^n \sum_{x \in L_i} \left(\frac{p}{1-p}\right)^{\gamma(i,x)} \Pr(A(I_1) = P_x)$$

$$= \sum_{x \in L \cap H_x} \left(\frac{p}{1-p}\right)^{\gamma(i,x)} \Pr(A(I_1) = P_x) = \sum_{x \in L} \Pr(A(I_1) = P_x) \sum_{i \in H_x} \left(\frac{p}{1-p}\right)^{\gamma(i,x)}.$$ \hspace{1cm} (1)

Notice now that each element is relevant in exactly $k$ instances in $\mathcal{I}$ and, since $A(I_i)$ succeed iff it reaches a leaf associated with a relevant element, we have that each leaf $x \in L$ allows $A$ to succeed in exactly $k$ of the instances in $\mathcal{I}$, i.e., $|H_x| = n - k$. Fix a leaf $x \in L$, let $h(x)$ be
the number of query vertices in \( P_x \), and consider a query vertex \( u_j \in P_x \) (corresponding to a query on the element in position \( x_j \)); index \( x_j \) appears in exactly \( k \) sets \( B_i \) and hence \( u_j \) is a bad query vertex for at most \( k \) instances \( I_i \in \mathcal{I} \). As there are at most \( h(x) \) query vertices in \( P_x \), we conclude that \( \sum_{i \in H_x} \gamma(i, x) \leq \sum_{i=1}^n \gamma(i, x) \leq kh(x) \). Since \( \varphi(\gamma(i, x)) = \left( \frac{p}{1-p} \right)^{\gamma(i, x)} \) is a convex function, by Jensen’s inequality \( \frac{1}{|H_x|} \sum_{i \in H_x} \varphi(\gamma(i, x)) \geq \varphi \left( \frac{1}{|H_x|} \sum_{i \in H_x} \gamma(i, x) \right) \) and, using the fact that \( \varphi(\cdot) \) is monotonically decreasing:

\[
\sum_{i \in H_x} \left( \frac{p}{1-p} \right)^{\gamma(i, x)} \geq |H_x| \left( \frac{p}{1-p} \right)^{kh(x)} = (n-k) \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}}. \tag{2}
\]

Let \( \phi(h(x)) = \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}} \) and \( \alpha_x = \Pr(A(I_1) = x) \). Using Jensen’s inequality once again and the fact that \( \sum_{x \in L} \alpha_x = 1 \) we have \( \sum_{x \in L} \alpha_x \phi(h(x)) \geq \phi \left( \sum_{x \in L} \alpha_x h(x) \right) \). The above, together with the equality \( \overline{h} = \sum_{x \in L} h(x) P(A(I_1) = P_x) \), allows us to write:

\[
\sum_{x \in L} \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}} \Pr(A(I_1) = P_x) \geq \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}} \sum_{x \in L} h(x) \Pr(A(I_1) = P_x) = \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}}. \tag{3}
\]

Combining the inequalities \([1] \) to \([3] \) and using the fact that \( k \leq c \cdot n \):

\[
1 \geq \sum_{i=1}^{n} \sum_{x \in L_i} \Pr(A(I_1) = P_x) \geq \sum_{x \in L} \Pr(A(I_1) = P_x) \sum_{i \in H_x} \left( \frac{p}{1-p} \right)^{\gamma(i, x)} \\
\geq (n-k) \sum_{x \in L} \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}} P(A(I_1) = P_x) \geq (n-k) \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}} \geq n(1-c) \left( \frac{p}{1-p} \right)^{\frac{kh(x)}{\gamma}}. 
\]

And hence \( \frac{kh(x)}{(1-c)n} \geq \log \frac{\overline{h}}{\frac{1}{(1-c)n}} \), which implies \( \overline{h} \geq \frac{(1-c)n}{\gamma} \cdot \frac{\log n - \log \frac{1}{(1-c)n}}{k} = \Omega \left( \frac{n \log n}{k} \right) \).

Finally, we remark that our choice of \( I_1 \) in the definition of \( \overline{h} \) is arbitrary and, by symmetry, the same lower bound also holds for the running time of \( A \) on any instance \( I_i \in \mathcal{I} \).

Using the above theorem, we can easily extend our lower bound to \( \text{FT-Min}(k) \):

\begin{unstated} \vspace{-1em}

\begin{corollary}
The expected number of comparisons of any (possibly randomized, possibly adaptive) algorithm that solves \( \text{FT-Min}(k) \) w.h.p., for \( 1 \leq k \leq c \cdot n \) and any constant \( c < 1 \), is \( \Omega \left( \frac{n}{k} \log n \right) \).
\end{corollary}

\end{unstated}

\section{Reduction}

In this section we reduce the problem of solving \( \text{FT-Min}(k) \) w.h.p. to the problem of solving \( \text{FT-Min}(\frac{k}{4}n) \) w.e.h.p. We say that an element \( x \) is small if it is one of the smallest \( k \) elements of \( S \), otherwise we say that \( x \) is large. The reduction selects a set \( S^* \) of size \( m \) containing at least \( \frac{k}{4}n \) small elements. For convenience we let \( m \) be the smallest power of 2 that is at least \( \gamma \log n \), where \( \gamma > 0 \) is parameter that will be chosen later.\footnote{Throughout the rest of the paper we will use \( \log \) for base-two logarithms and \( \ln \) for natural logarithms.} We construct \( S^* \) as follows:

\begin{itemize}
\item Create \( m \) sets by independently sampling, with replacement, \( 3 \frac{n}{k} \) elements per set from \( S \).
\end{itemize}
Run FindMin with error probability \( q = \frac{1}{m} \) on each of the sets. Let \( S^* = \{x_1, \ldots, x_m\} \) be the collection of the returned elements, where \( x_i \) is the element selected from the \( i \)-th set.

Using Theorem 1, Lemma 2 and the Chernoff bound, we are able to prove the following:

**Lemma 5.** The probability that less than \( \frac{3}{4}m \) elements in \( S^* \) are small is at most \( e^{-\frac{m}{4\ln 2}} \).

We are now ready to show the consequence of the above reduction:

**Lemma 6.** Let \( A \) be an algorithm that solves FT-Min(\( \frac{3}{4}n \)) with a probability of success of at least \( 1 - \epsilon^{-n} \), for some constant \( \epsilon > 1 \), and let \( T(n) \) be its running time. For any \( k \), there exists an algorithm that solves FT-Min(k) w.h.p. in \( O\left(\frac{n}{k}\log n\right) + T(\Theta(\log n)) \) time.

**Proof.** We first choose \( \gamma = \max\{600, \frac{2}{\ln \epsilon}\} \) and we compute the set \( S^* \) according to our reduction. Then we run \( A \) on \( S^* \) and answer with element it returns. The first step of the reduction can be easily implemented in \( O\left(\frac{mn}{\epsilon \ln n}\right) = O\left(\frac{n}{\epsilon} \log n\right) \) time, and since each of the \( m = O(\log n) \) executions of FindMin requires time \( O\left(\frac{n}{\epsilon} \log \frac{1}{\delta}\right) = O\left(\frac{n}{\epsilon}\right) \) (see Theorem 1 and recall that \( q = 1/10 \)), the total time spent so far is \( O\left(\frac{n}{\epsilon} \log n\right) \). Moreover, \( m = \Theta(\log n) \) implying that \( A \) takes time \( T(m) = T(\Theta(\log n)) \). Since \( \gamma \geq 600 \), by Lemma 5 the probability that less than \( (3/4)m \) elements in \( S^* \) are small is at most \( e^{-\frac{m}{4\ln 2}} < \epsilon^{\frac{1}{2} \ln n} < \frac{1}{n^2} \).

Moreover, as \( m \geq \gamma \log n \), the probability that \( A \) returns one of the smallest \( \frac{3}{4}m \) elements in \( S^* \) is at least \( 1 - \epsilon^{-\gamma \log n} = 1 - 2^{-(\log n) \gamma \log n} = 1 - n^{-\gamma \log \epsilon} \geq 1 - \frac{1}{n^2}. \) The claim follows by using the union bound on the previous two probabilities.

It is not hard to see that, if we choose algorithm \( A \) in Lemma 6 to be FindMin with \( q = 2^{-n} \), then we have \( T(n) = O(n \log \frac{1}{\delta}) = n^2 \). By Lemma 6, this already solves FT-Min(k) in \( O\left(\frac{n}{k} \log n + \log^2 n\right) \) time w.h.p., which matches our lower bound of \( \Omega\left(\frac{n}{k} \log n\right) \) for \( k = O\left(\frac{n}{\log n}\right) \). Nevertheless, the major difficulty in solving FT-Min(k) lies in the case \( k = \omega\left(\frac{n}{\log n}\right) \). In Section 5 and Section 7, we will design two algorithms that solve FT-Min(\( \frac{3}{4}n \)) w.e.h.p. and respectively achieve \( T(n) = O(n \log n) \) in the worst case and \( T(n) = O(n) \) in expectation, thus improving the time required to solve FT-Min(k) to \( O\left(\frac{n}{k} \log n + (\log n) \log \log n\right) \) in the worst case and \( O\left(\frac{n}{k} \log n\right) \) in expectation.

We close this section by pointing out that a similar reduction strategy also works for the related problem FT-Retrieval(k):

**Lemma 7.** Let \( A \) be an algorithm that solves FT-Retrieval(\( \frac{3}{4}n \)) with probability at least \( 1 - \epsilon^{-n} \), for some constant \( \epsilon > 1 \), and let \( T(n) \) be its running time. For any \( k \), there exists an algorithm that solves FT-Retrieval(k) w.h.p. in \( O\left(\frac{n}{k} \log n\right) + T(\Theta(\log n)) \) time.

## 5 Knock-Out Tournament

In this section we design an algorithm that solves FT-Min(\( \frac{3}{4}n \)) w.e.h.p using \( O(n \log n) \) comparisons in the worst case. For the sake of simplicity we assume that \( n \) is a power of 2.

Our algorithm simulates a knockout tournament and works in \( \log n \) rounds: in the beginning we construct a set \( S_n \) by sampling with replacement \( n \) elements from the input set \( S \), then in the generic \( i \)-th round we match together \( \frac{n}{2^i} \) pairs of elements selected without replacement from the set \( S_{\frac{n}{2^i}} \), and we add the match winners to a new set \( S_{\frac{n}{2^{i+1}}} \). After the \( (\log n) \)-th round we are left with a set \( S_1 \) containing a single element: this is the winner of the tournament, i.e., it is the element returned by our algorithm.

A match between elements \( x \) and \( y \) in the \( i \)-th round consists of \( 2c_p \left[ 2^i \right] + 5 \) comparisons using the majority strategy of Lemma 2 i.e., the winner of the match is the element that
is reported to be smaller by the majority of the comparisons (here $c_p$ is the constant of Lemma \ref{lem:prob_error} corresponding to an error probability of $p$). The following lemma bounds the success probability of our algorithm:

\begin{lemma}
Consider a tournament among $n$ elements, where $n$ is a power of 2. The probability that the winner of the tournament is a small element is at least $1 - 2^{-\Omega(n)}$.
\end{lemma}

\begin{proof}
We prove the claim by induction on $n$ by upper bounding the complementary probability. If $n = 1$, then there exists only one element $x \in S_n$, which is trivially the winner and, by our choice of $S_n$, we have $\Pr(x \text{ is large}) \leq 1 - \frac{1}{2} = 2^{-2}$.

Now, let $n > 2$ be a power of 2 and suppose that the claim holds for tournaments of $n/2$ elements. We prove that it must also hold for tournaments of $n$ elements. Let $x$ be the winner of the tournament. Since $n \geq 2$, $x$ must be the winner of a match between two elements $x_1, x_2 \in S_{n/2}$ which, in turn, must be the winners of two (independent) sub-tournaments involving $\frac{n}{2}$ elements each.

For $x$ to be large either (i) $x_1$ and $x_2$ are both large, which happens with probability at most $2^{-2}\theta^{-2}$ (by inductive hypothesis), or (ii) exactly one of $x_1$ and $x_2$ is large and it wins the match. The probability that exactly one of $x_1, x_2$ is large can be upper-bounded by $2 \cdot 2^{-\frac{n}{2} - 1} (1 - 2^{-\frac{n}{2} - 1})$, and hence we focus on the probability that, in a match between a large and a small element, the large element wins. Since $x_1$ and $x_2$ are compared at least $2c_p\left(\frac{n}{2}\right) + 5$ times we know, by Lemma \ref{lem:prob_error} that this probability must be smaller than $2^{-\frac{n}{2} - 2}$. Putting it all together, we have:

$$
\Pr(x \text{ is large}) \leq 2 \cdot 2^{-\frac{n}{2} - 2} + 2 \cdot 2^{-\frac{n}{2} - 1} (1 - 2^{-\frac{n}{2} - 1}) 2^{-\frac{n}{2} - 2} < 2^{-n - 2} + 2^{-n - 2} = 2^{-n - 1} \quad \Box
$$

The above lemma already results in algorithm with error probability exponentially small in $n$ and $O(n \log n)$ running time, as shown by the following:

\begin{lemma}
Simulating the tournament requires $O(n \log n)$ time.
\end{lemma}

In particular, combining Lemmas \ref{lem:prob} and \ref{lem:count} with our reduction (see Lemma \ref{lem:count}) we can immediately obtain an algorithm for FT-Min$(k)$ which is optimal for $k = O(\log \log n)$:

\begin{theorem}
FT-Min$(k)$ can be solved w.h.p. in $O\left(\frac{n}{k} \log n + (\log n) \log \log n\right)$ worst-case time, w.h.p.
\end{theorem}

We conclude this section by providing a corollary that will be useful in Section \ref{sec:multi}:

\begin{corollary}
After the $i$th iteration, $\frac{n}{2^i}$ elements are selected into $S_{\frac{n}{2^i}}$ in $O(n \cdot i)$ worst-case time. Each of the elements is small, independently, with probability at least $1 - 2^{-2^{i+1}}$.
\end{corollary}

\section{Solving FT-Retrieval $(k)$}

In this section we argue that FT-Retrieval$(k)$ can be solved in optimal worst-case time, w.h.p. Thanks to our reduction of Section \ref{sec:multi} (see Lemma \ref{lem:count}), it suffices to solve FT-Retrieval$(\frac{n}{k})$ w.e.h.p. in $O(n)$ time. Recall that, in this problem, comparisons between elements are no longer allowed, rather, we can only gather information on whether an element is relevant by querying an oracle $O$, which answers correctly with a probability of at least $1 - p$. It turns out that FT-Retrieval$(\frac{n}{k})$ can be seen as a special case of the multi-armed bandit problem (MABP for short, see, e.g., \cite{12}): in our instances we know that the stochastic reward of each arm is either $p$ or $1 - p$, and that there are at least $\frac{n}{k}$ arms that maximize the expected reward. This allows to reduce the required running time needed
to find w.h.e.p. the best arm from $\Omega(n^2 \log n)$ (which would result from using a algorithm form the general MABP problem) to $O(n)$, e.g., by using the algorithm in [17]. We now describe a variant of this algorithm, which will then further modify in the following section in order to obtain our results for FT-MIN.

The algorithm works as follows: we consider the input elements one at a time and we subject each of them to up to $1 + \log n$ phases of tests. In particular, an element starts from phase 1 and it advances from phase $i$ to phase $i + 1$ by passing a test consisting of $2^{i-1} \cdot 6cp + 1$ queries to $O$ (where $c_p$ is the constant of Lemma 2). For this test to be successful the majority of the queries must regard the element as relevant. If the element fails the test, then it is immediately discarded, the next input element is considered, and the process repeats from phase 1. We return the first element to pass the test of the $(1 + \log n)^{th}$ phase.

In such a process, a non-relevant element is likely to be discarded after a constant number of phases, while a relevant element is either discarded quickly or it is likely to pass all the tests. In the end we return a relevant element in $O(n)$ time with probability at least $1 - \frac{2}{n}$.

For the sake of completeness, we formalize this intuition in Appendix E, while here will only state our main result pertaining FT-RETRIEVAL, which is a direct consequence of the previous discussion and of Lemma 7.

\begin{theorem}
FT-RETRIEVAL can be solved in $O(\frac{n}{\log n})$ time with high probability.
\end{theorem}

7 Solving FT-MIN(k) in Optimal Expected Time

In this section we solve FT-MIN(k) w.h.p. in $O(\frac{n}{\log n} + (\log n) \log \log n)$ expected time. We achieve this by designing an algorithm that requires $O(n \log n)$ expected time to solve FT-MIN($\frac{n}{4}$) w.e.h.p. The latter algorithm is inspired by the multi-phase process of Section 6. The key intuition is simple: if we are able design an oracle $O$ that can detect, in constant time, whether an element is small with a probability of error bounded away from $\frac{1}{2}$, we can then simulate the multi-phase process for FT-RETRIEVAL($\frac{n}{4}$) by considering small elements to be relevant. Unfortunately, designing such an oracle is problematic since it needs to distinguish between elements of ranks $\frac{3}{4} n \pm \Theta(1)$. As a first attempt to get around this problem, we could relax the requirements on $O$ and only require it to provide accurate answers for a constant fraction of the elements (namely, the smallest and the largest ones). While it is now possible to run the multi-phase process, this weaker guarantee on $O$ negatively impacts its running time since elements with inaccurate answers are problematic, i.e., they are likely to get discarded after a non-constant number of tests. To archive the desired liner-time complexity, we combine the knockout tournament of Section 5 and the multi-phase process into a two-stage algorithm. In the first stage we apply the knockout tournament to pre-select $\frac{n}{\log n}$ elements, most of which are likely to be small. In the second stage, we design two different oracles $O_1$ and $O_2$, both of which only provide weak guarantees. $O_1$ is used to conduct a preliminary test on the $\frac{n}{\log n}$ elements to prune (most of) the problematic elements for $O_2$, which is then used to actually simulate (a modified version of) the multi-phase process of Section 6. We separately study the correctness and time complexity of these two stages.

7.1 Pre-Selection

We run the knockout tournament of Section 5 with $\alpha = 2^{0.9}$ up to the $(\log \log n)^{th}$ iteration. By Lemma 8 and Corollary 11 the time needed is $O(n \log \log n)$ and the number of surviving elements is $m = \frac{n}{\log n}$, each of which is independently small with probability at least
$1 - 2^{-2^m} = \frac{1}{2^m}$. As a consequence, at least three quarters of the selected elements are good w.e.h.p., as shown by the following:

**Lemma 13.** It is possible to select in $O(n \log \log n)$ worst-case time a set of $\frac{n}{4\log n}$ elements containing at least $\frac{3n}{4\log n}$ small elements with probability at least $1 - 2^{-\frac{n}{2^m}}$.

**Proof.** For $j = 1, \ldots, n/\log n$, let $X_j$ be an indicator random variable that is 1 iff the $j$th selected element is not small. Let $X = \sum_{j=1}^{n/\log n} X_j$ be the number of selected elements that are not small. We have $\Pr(X_j = 1) \leq 1/n$ and we need to bound $\Pr \left( X > \frac{n}{4\log n} \right)$.

Since $X$ is stochastically smaller or equal to $Y = \sum_{j=1}^{n/\log n} Y_j$, where $Y_j$s are i.i.d. Bernoulli random variables of parameter $p = \frac{1}{n}$, we have that $\mathbb{E}[Y] = np = \frac{n}{\log n} \cdot \frac{1}{n} = \frac{1}{\log n}$ and we can use the following Chernoff bound for $\delta > e$ (whose proof is shown in Appendix [F.1]):

$$\Pr(Y > \delta \cdot \mathbb{E}[Y]) \leq \exp \left( -\frac{\delta^2 \mathbb{E}[Y] \cdot \ln(\delta/e)}{2} \right).$$

Indeed, choosing $\delta = \frac{1}{4}$, and using the identity

$$\ln x \geq \frac{x}{4} \log x \forall x \geq 1,$$

we obtain:

$$\Pr \left( X > \frac{n}{4\log n} \right) \leq \Pr \left( Y > \frac{n}{4\log n} \right) = \Pr \left( Y > \delta \cdot \mathbb{E}[Y] \right) \leq 2^{-\frac{n}{4\log n} \cdot \frac{1}{n} \cdot \ln 4},$$

where the last inequality holds for sufficiently large values of $n$. \hfill \Box

### 7.2 Modified Multi-Phase Process

Let $S'$ be the set of $\frac{n}{\log n}$ elements after the pre-selection stage, we call $S'_\rho$ the set containing the smallest $\lceil \rho \cdot \frac{n}{\log n} \rceil$ elements of $S'$, and we let $S'_p = S' \setminus S'_\rho$. We use two oracles $O_1, O_2$ that can be queried with an element $x \in S'$, answer in constant time, and satisfy the following conditions: $O_1$ reports an element $x$ to be relevant with probability at least $1 - p_1$ if $x \in S'_{1/6}$ and at most $p_1$ if $x \in S'_{1/3}$; $O_2$ reports an element $x$ to be relevant with probability at least $1 - p_2$ if $x \in S'_{1/3}$, and at most $p_2$ if $x \in S'_{1/4}$. Here $p_1$ and $p_2$ are absolute constants in $[0, \frac{1}{2}]$. Notice that, unlike in Section 6, these oracles provide no guarantees for elements $x \in S'_{1/3} \setminus S'_{1/6}$ and $x \in S'_{1/4} \setminus S'_{1/3}$, respectively. It is easy to see that $O_1$ and $O_2$ can be implemented by comparing $x$ with other randomly selected elements from $S'$ (as we discuss in Appendix [C]). Let $c_{p_1}$ and $c_{p_2}$ be the constants of Lemma 2 for $p = p_1$ and $p = p_2$, respectively. We modify the process of Section 6 as follows:

- Whenever a new element is considered, we first conduct a preliminary test consisting of $8 \cdot c_{p_1} \left\lfloor \ln n \right\rfloor + 1$ queries to $O_1$. If the test is passed, i.e., if the majority of the queries report the element to be relevant, we proceed with the regular tests (see the next item), otherwise we immediately discard the element and we move to the next one;

- The $i$th test now consists of $2 \cdot \left\lfloor 2^i \ln n \right\rfloor c_{p_2} + 1$ queries to $O_2$;

- We lower the number of phases to $\eta = 1 + \left\lceil \log \frac{n}{\log n} \right\rceil$ (instead of $1 + \log n$).

We start by studying the success probability of this modified process.

**Lemma 14.** An element in $S'_{1/6}$ passes the preliminary test and all the following tests with probability at least $1 - \frac{1}{n^2}$.

**Proof.** By Lemma 2 and by the definition of $O_1$, the probability that an element in $S'_{1/6}$ passes the preliminary test is at least $1 - e^{-4\ln n} = 1 - n^{-4}$. Moreover, Lemma 2 also implies
that the probability that such an element passes the \( i \)th test is at least \( 1 - e^{-2i \ln n} > 1 - \frac{1}{n^2} \).

As a consequence, an element in \( S'_{1/6} \) passes all the tests with probability at least:

\[
\prod_{i=1}^{\eta} \left( 1 - \frac{1}{n^2} \right) \geq 1 - \sum_{i=1}^{\eta} \left( \frac{1}{n^{2i}} \right)^2 \geq 1 - \left( \sum_{i=1}^{\eta} \frac{1}{n^2} \right)^2 = 1 - \left( \frac{1 - \frac{1}{n}}{n} \right)^2 = 1 - \frac{4}{n^2},
\]

where we used Weierstrass product inequality, the identity \( \sum_{i=0}^{\infty} \frac{1}{n^i} = \frac{1}{1 - 1/n} \). The claim follows by using the union bound on the complementary probabilities.

Lemma 15. An element in \( S'_{3/4} \) passes the preliminary test and all the following tests with probability at most \( 2^{-n} \).

Proof. If the process does not return an element in \( S'_{3/4} \) then the following condition is true: (i) no element in \( S'_{1/6} \) passes the preliminary test and all the tests, or (ii) one element from \( S'_{3/4} \) passes the pre-test and all the tests. By Lemma 14 the probability that an element \( S'_{1/6} \) passes the preliminary test and all the following tests is at least \( 1 - \frac{3}{n^2} \). Therefore, for sufficiently large values of \( m \), the probability of (i) is at most: \( \left( \frac{5}{n^2} \right)^{\ln n} \leq 2^{\frac{\log 3}{\log n}} \frac{1}{n} = 2^{-\frac{2}{3} + o(n)} \leq 2^{-\frac{2}{3}} \).

Moreover, by Lemma 15 the probability of (ii) is most \( \frac{n}{4} \cdot 2^{-n} < 2^{-\frac{2}{3}} \). The claim follows, as soon as \( n \geq 25 \), by using the union bound.

We now provide an upper bound on the expected running time of our modified process:

Lemma 17. The expected number of queries to \( O_1 \) and \( O_2 \) of our modified process is \( O(n) \).

Proof. The preliminary test is performed on all the elements in \( S' \) and it consists of \( 8 \cdot c_{p_1} \cdot [\ln n] + 1 \) queries to \( O_1 \) per element. Hence, the number of queries of this step is \( O(\frac{n}{\log n} \cdot [\ln n]) = O(n) \), and we focus on upper bounding the number of queries performed during the second step of the process. Consider an element in \( x \in S'_{1/3} \): by Lemma 2 the probability that it passes the first \( i - 1 \) tests, and fails the \( i \)th test is at most \( e^{-2i \ln n} = n^{-2i} \).

In this situation, the total number of queries on \( x \) is at most \( \sum_{j=1}^{i} (2 \cdot 2^i \ln n) \cdot c_{p_2} + 1 \leq i + 2^i \ln n \cdot c_{p_2} \sum_{j=1}^{i} 2^j < 2^{i+3} \ln n \cdot c_{p_2} \), hence the expected number of queries on \( x \) is at most:

\[
\sum_{i=1}^{\eta} 2^{i+3} \ln n \cdot c_{p_2} \cdot n^{-2i} \leq 16 \ln n \cdot c_{p_2} \sum_{i=1}^{\eta} n^{-2i+1} \leq 16 \ln n \cdot c_{p_2} \sum_{i=0}^{\infty} 2^{-i} = 32 \ln n \cdot c_{p_2}.
\]

If \( x \in S'_{1/3} \), the probability that \( x \) passes the pre-test is at most \( 1/n^4 \) and, since the maximum number of queries performing on \( x \) during the following tests is at most \( \sum_{i=1}^{\eta} 2^{i+3} [\ln n] \cdot c_{p_2} \leq 32 [\ln n] \cdot c_{p_2} \cdot \frac{n}{\log n} = O(n) \), we have that, for sufficiently large values of \( n \), the expected number of queries on \( x \) is at most \( 1 \).

It remains to bound number of queries of an element that passes all the tests: once again this can be at most \( \sum_{i=1}^{\eta} 2^{i+3} [\ln n] \cdot c_{p_2} = O(n) \).

We can finally bound the expected number of queries performed during the whole process as follows: \( |S'_{1/3}| \cdot 32 [\ln n] \cdot c_{p_2} + |S'_{1/3}| \cdot 1 + O(n) \leq \frac{n}{\log n} 32 [\ln n] \cdot c_{p_2} + O(n) = O(n) \).

Theorem 18. FT-Min \( (\frac{1}{4} n) \) can be solved in \( O(n \log \log n) \) expected time w.e.h.p.

Theorem 18 and Lemma 19 together immediately imply the main positive result of this paper:

Theorem 19. FT-Min \( (k) \) can be solved in \( O(\frac{n}{k} \log n + (\log n) \log \log \log n) \) expected time w.h.p.
**Acknowledgments.** The authors wish to thank Tomáš Gavenčiak, Barbara Geissmann, Paolo Penna, and Peter Widmayer for insightful discussions, and an anonymous reviewer of a former version of this paper for pointing out the connection between FT-Retrieval($k$) and the multi-armed bandit problem.

**References**

1. Martin Aigner. Finding the maximum and minimum. *Discrete Applied Mathematics*, 74(1):1–12, 1997.
2. Gary Anthes. Inexact design: beyond fault-tolerance. *Commun. ACM*, 56(4):18–20, 2013.
3. A. Bagchi. On sorting in the presence of erroneous information. *Inf. Process. Lett.*, 43(4):213–215, 1992.
4. Robert C Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and materials reliability*, 5(3):305–316, 2005.
5. Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC’93)*, pages 130–136, 1993.
6. Mark Braverman, Jiemin Mao, and S. Matthew Weinberg. Parallel algorithms for select and partition with noisy comparisons. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, (STOC’16)*, Cambridge, MA, USA, June 18-21, 2016, pages 851–862, 2016.
7. Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA’08)*, pages 268–276, 2008.
8. S Cheemalavagu, Pinar Korkmaz, and Krishna V Palem. Ultra low-energy computing via probabilistic algorithms and devices: CMOS device primitives and the energy-probability relationship. In *Proc. of The 2004 International Conference on Solid State Devices and Materials*, pages 402–403, 2004.
9. Suresh Cheemalavagu, Pinar Korkmaz, Krishna V Palem, Bilge ES Akgul, and Lakshmi N Chakrapani. A probabilistic CMOS switch and its realization by exploiting noise. In *IFIP International Conference on VLSI*, pages 535–541, 2005.
10. Xi Chen, Sivakanth Gopi, Jiemin Mao, and Jon Schneider. Competitive analysis of the top-$K$ ranking problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, Barcelona, Spain, Hotel Porta Fira, January 16-19, pages 1245–1264, 2017.
11. Ferdinando Cicerone. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013.
12. Eyal Even-Dar, Shie Mannor, and Yishay Mansour. PAC bounds for multi-armed bandit and markov decision processes. In *Computational Learning Theory, 15th Annual Conference on Computational Learning Theory, COLT 2002*, Sydney, Australia, July 8-10, 2002, *Proceedings*, pages 255–270, 2002.
13. Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.
14. Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with recent comparison errors. In *Proceeding of the 28th International Symposium on Algorithms and Computation (ISAAC’17)*, pages 38:1–38:12, 2017.
15. Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna, editors. *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS’16)*, volume To appear. of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
Barbara Geissmann, Matúš Mihalák, and Peter Widmayer. Recurring comparison faults: Sorting and finding the minimum. In Proceedings of the 20th International Symposium on Fundamentals of Computation Theory (FCT’15), pages 227–239, 2015.

Ofer Grossman and Dana Moshkovitz. Amplification and derandomization without slowdown. In IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA, pages 770–779, 2016.

Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC’07), pages 95–103, 2007.

Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant Algorithms. In Proceedings of the 19th Annual European Symposium on Algorithms, pages 736—747, 2011.

K. B. Lakshmanan, Bala Ravikumar, and K. Ganesan. Coping with erroneous information while sorting. IEEE Trans. Computers, 40(9):1081–1084, 1991.

Frank Thomson Leighton and Yuan Ma. Tight bounds on the size of fault-tolerant merging and sorting networks with destructive faults. SIAM J. Comput., 29(1):258–273, 1999.

Philip M. Long. Sorting and searching with a faulty comparison oracle. Technical report, Santa Cruz, CA, USA, 1992.

Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Sorting noisy data with partial information. In Proceeding of the 4th Innovations in Theoretical Computer Science conference, (ITCS’13), pages 515–528, 2013.

Michael Mitzenmacher and Eli Upfal. Probability and computing - randomized algorithms and probabilistic analysis. Cambridge University Press, 2005.

Krishna V. Palem and Lingamneni Avinash. Ten years of building broken chips: The physics and engineering of inexact computing. ACM Trans. Embedded Comput. Syst., 12(2s):87:1–87:23, 2013.

Andrzej Pelc. Searching with known error probability. Theor. Comput. Sci., 63(2):185–202, 1989.

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

Bala Ravikumar, K. Ganesan, and K. B. Lakshmanan. On selecting the largest element in spite of erroneous information. In Proceedings of 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS’87), pages 88–99, 1987.

Tezzaron Semiconductor. Soft errors in electronic memory – a white paper, 2004.
A Omitted Proofs from Section 2

A.1 Proof of Lemma 2

Proof. Suppose, w.l.o.g., that \( x < y \). Let \( X_i \in \{0, 1\} \) be an indicator random variable that is 1 iff the \( i^{th} \) comparison (resp. query) succeeds. Let \( X = \sum_{i=1}^{2c_p t + 1} X_i \). Since the \( X_i \)s are independent Bernoulli random variables of parameter \( 1 - p \), \( X \) is a binomial random variable of parameters \( \frac{2c_p t + 1}{1 - p} \) and \( 1 - p \), and hence \( \mathbb{E}[X] = 2\eta(1 - p) = (1 - p)(2c_p t + 1) \). Moreover, since \( p < 1/2 \) we know that \( 2(1 - p) > 1 \) and hence we can use the Chernoff bound \( \Pr(X \leq (1 - \delta)\mathbb{E}[X]) \leq \exp\left(-\frac{\delta^2 \mathbb{E}[X]}{2}\right) \) to upper bound to the probability of failure of the majority strategy. Indeed:

\[
\Pr(X \leq \eta) = \Pr\left(X \leq \frac{1}{2(1 - p)\mathbb{E}[X]}\right) \leq \exp\left(-\frac{(2(1 - p) - 1)^2}{8(1 - p)^2}2\eta(1 - p)\right)
\]

\[
= \exp\left(-\frac{(1 - 2p)^2}{4(1 - p)}\eta\right) < \exp\left(-c_p(1 - 2p)^2\right),
\]

which satisfies claim once we choose \( c_p = \left\lceil \frac{4(1 - p)}{(1 - 2p)^2} \right\rceil \). ◼

B Omitted Proofs from Section 3

B.1 Proof of Corollary 4

Proof. We show how any algorithm \( A \) that solves \( \text{FT-Min}(k) \) can also be used to solve \( \text{FT-Retrieval}(k) \) using a number of comparisons that matches, up to constant factors, the number of queries performed by \( A \). This implies that the lower bound of Theorem 3 can be directly translated into a lower bound \( \text{FT-Min}(k) \). For the sake of simplicity we assume that \( p < \frac{1}{4} \) as otherwise we can replace each query with \( 4c_p + 1 \) queries and select the majority result, as shown in Lemma 2.

Given an instance of \( \text{FT-Retrieval}(k) \) we fix an arbitrary order among the elements in \( S \) and we simulate the execution of \( A \) on \( S \) with a probability of comparison errors of \( 2p < \frac{1}{2} \). This is done as follows: each time a comparison between two elements \( x \) and \( y \) is to be performed, we query \( x \) and \( y \) instead. If the answers to the two queries are equal we choose the result of the comparison in accordance to our arbitrary order. Otherwise, we report the element whose answer to the query was “relevant” as being the smaller between \( x \) and \( y \). It is easy to see that (i) in absence of errors, this strategy consistently reports the \( k \) relevant elements in \( S \) as being the \( k \) smallest elements, and (ii) when simulating a comparison there is a probability of at least \( 1 - 2p < \frac{1}{2} \) that the answers to the two queries are correct. The above discussion implies that the element returned by \( A \) must be a relevant w.h.p., hence the claim. ◼

C Omitted Proofs from Section 4

C.1 Proof of Lemma 5

Proof. Since the \( i^{th} \) set contains \( 2^{\frac{n}{i}} \) elements and each of them is small independently with a probability of \( \frac{k}{n} \), the probability that no element in the \( i^{th} \) set is small is upper bounded by

\[
\left(1 - \frac{k}{n}\right)^{2^{\frac{n}{i}}} = \left(1 - \frac{1}{i}\right)^{3^i} \leq e^{-3} < \frac{1}{20}.
\]
where we let $t$ be $\frac{2}{6}$ and we used the inequality $(1 + \frac{1}{t})^t < e^{-1}$ for $t \geq 1$. In other words, for every $i$, the event “the $i$th set contains a small element” has probability at least $1 - \frac{3}{240}$. Moreover, by our choice of $q$, the probability that $\text{FindMin}$ returns the correct minimum of the $i$th set is at least $1 - \frac{1}{15}$. Clearly, if both the previous events happen, $x_i$ must be a small element and, by the union bound, the complementary probability can be at most $\frac{1}{20} + \frac{1}{10} \leq \frac{1}{5}$.

Let $X_i$ be an indicator random variable that is 1 iff $x_i$ is a good element so that

$$X = \sum_{i=1}^{m} X_i$$

is exactly the number of small elements in $S^*$. Since the $x_i$s are independently good with a probability of at least $\frac{2}{6}$, the variable $X$ is stochastically larger than a Binomial random variable of parameters $m$ and $\frac{2}{6}$. As a consequence $\mathbb{E}[X] \geq \frac{5}{6}m \geq \frac{5}{6}n \gamma \log n$ and, by using Chernoff bound [24, Theorem 4.2 (2)], we obtain:

$$\Pr \left( X \leq \frac{3}{4}m \right) = \Pr \left( X \leq \frac{9}{10} \mathbb{E}[X] \right) \leq e^{-\frac{1}{3} \left( \frac{9}{10} \right)^2 \cdot \frac{5}{6} n \gamma \log n} = e^{-\frac{3}{40} \gamma n \log n}.$$  

\[\blacktriangleright\]

C.2 Proof of Lemma 7

\textbf{Proof.} We construct $m$ sets $S_1, \ldots, S_m$ by sampling, with replacement $\frac{2}{5}$ elements per set from $S$, where $m$ is the first power of two larger than $\max\{600, \frac{\log n}{\log c} \}$ elements. Then, we consider these sets one at a time, and for each set $S_i$, we select one element $x_i \in S_i$ as follows: we simulate an execution of $\text{FindMin}$ with $q = \frac{1}{15}$ on $S_i$ by replacing each comparison between two elements $x, y \in S_i$ with $6c_0 + 1$ queries on $x$ and on $y$. We consider the majority result among the queries for $x$ (resp. $y$) and, if the two majority results differ we report the element with a majority of “relevant” answers as being the smaller of the two. If the majority results coincide we break the tie in an arbitrary, but consistent, way. This process consistently simulates the results we would get if the set of elements had an intrinsic order in which any relevant element is smaller than any non-relevant element. Moreover, by Lemma 2 and by the union bound, the comparison results are wrong with a probability of at most $2e^{-3}$ and hence $\text{FindMin}$ returns a relevant element with probability at least $1 - (\frac{1}{15} + \frac{2}{5}) \geq \frac{5}{6}$.

Let $S^* = \{x_1, \ldots, x_m\}$ and let $X_i$ be an indicator random variable that is 1 iff $x_i$ is a relevant element. We have that $X = \sum_{i=1}^{m} X_i$ is the number of relevant elements in $S^*$. Since the $x_i$s are independently relevant with a probability of at least $\frac{2}{5}$, by the same argument used in the proof of Lemma 7 we have: $\Pr \left( X \leq \frac{3}{4}m \right) \leq e^{-\frac{3}{40} \gamma n \log n} < e^{-\frac{1}{2} \ln n} < \frac{1}{e^{2\frac{1}{6}}}$.

Now we run algorithm $A$ on $S^*$ which, by hypothesis, succeeds with probability at least $1 - e^{-\gamma \log n} = 1 - 2^{-\gamma c \log n} = 1 - n^{-\gamma \log c} \geq 1 - \frac{1}{15}$. This means that the element returned by $A$ is relevant with probability at least $1 - \frac{2}{15}$. Concerning the running time, notice that the set $S^*$ can be built in $O(\ell \log q \cdot \log n) = O(\frac{5}{6} \log n)$ while the execution of $A$ requires time $T(m)$ where $m = \Theta(\log n)$.

\[\blacktriangleright\]

D. Omitted Proofs from Section 5

D.1 Proof of Lemma 9

\textbf{Proof.} The tournament consists of $\log n$ rounds. The number of matches that take place in round $i$ is $\frac{2}{5}$ and, for each match, $O(2^i)$ comparisons are performed. It follows that the total number of comparisons performed in each round is $O(n)$ and, since there are $O(\log n)$ rounds, the overall running time is $O(n \log n)$.

\[\blacktriangleright\]
D.2 Proof of Corollary 11

Proof. After the $i$th round, $\frac{n}{2^i}$ elements are selected, and each of these elements can be seen as the winner of a sub-tournament among a set of $2^i$ elements in $S$. Since these $\frac{n}{2^i}$ sets of $2^i$ elements are disjoint, the above observation together with Lemma 8 implies the claim. ◀

E Analysis of the Algorithm of Section 6

We start by bounding the probability that a single relevant (resp. non-relevant) element passes all the tests.

Lemma 20. A relevant element passes all the tests with a probability of at least $7/8$.

Proof. By Lemma 2, the probability that a relevant element passes the $i$th test is at least $1 - e^{-3 \cdot 2^{i-1}} > 1 - \frac{1}{9^i}$. As a consequence, a relevant element passes all the tests with probability at least:
\[
\prod_{i=1}^{\log_2 n} \left( 1 - \frac{1}{9^i} \right) \geq 1 - \sum_{i=1}^{\log_2 n} \frac{1}{9^i} \geq 1 - \sum_{i=1}^{\infty} \frac{1}{9^i} \geq 1 - \frac{1}{8} = \frac{7}{8},
\]
where we used Weierstrass product inequality and the fact that $\sum_{i=0}^{\infty} \frac{1}{9^i} = \frac{9}{8}$.

Lemma 21. A non-relevant element passes all the tests with a probability of at most $4^{-n}$.

Proof. For a non-relevant element to pass all the tests, more than half of the queries of each test must return the wrong answer. By Lemma 2, this happens with probability at most $e^{-3 \cdot 2^{i-1}} < \frac{1}{2^{i-1}}$ for the $i$th test. Let $\sigma = \sum_{i=1}^{\log_2 n} 2^{i-1} = \sum_{i=0}^{\log_2 n} 2^i = 2^{\log_2 n + 1} - 1 = 2n - 1$. We can upper bound the sought probability as follows:
\[
\prod_{i=1}^{\log_2 n} \frac{1}{2^{i-1}} = \frac{1}{2^n} = \frac{1}{2^{\log_2 n}} \leq 4^{-n}.
\]

The above two lemmas allow us to bound the overall success probability of our process:

Lemma 22. The probability that the process returns a relevant element is at least $1 - 3^{-n}$.

Proof. If the process does not return a relevant element then the following condition must be true: (i) all relevant elements fail one of the tests; or (ii) one non-relevant element passes all the tests.

For the former, since there are at least $\frac{3n}{4}$ relevant elements, Lemma 20 bounds the probability to be at most $(\frac{1}{2})^{\frac{3n}{4}} < (\frac{1}{2})^n$. For the latter, since there are at most $m/4$ non-relevant elements, Lemma 21 bounds the probability to be at most $\frac{1}{2} \cdot 4^{-n}$. It is clear that $(\frac{1}{2})^n + \frac{1}{2} \cdot 4^{-n} < 3^{-n}$ for all $n \geq 1$.

We now analyze the total number of queries, which also provides an asymptotic upper bound to the total running time:

Lemma 23. The total number of queries for relevant elements is at most $37 \cdot c_p \cdot n$ with probability at least $1 - 4^{-n}$.

Proof. Once a relevant element passes all the test, the process terminates and the total number of queries for that element is $\sum_{i=1}^{\log_2 n + 1} (2^{i-1} \cdot 6c_p + 1) = 6c_p(2n-1)+\log n+1 < 13 \cdot c_p \cdot n$.

Therefore, it is sufficient to bound the number of queries for relevant elements that do not pass all the tests. We say that a query bad if it is performed on relevant element that does not pass all the tests. Let $X_j$ be a random variable corresponding to the number of bad queries for the $j$th relevant element, and let $X = \sum_j X_j$. We will derive a Chernoff bound for $X$ and, to this aim, we start by bounding $\mu = \mathbb{E} [2^m X]$ where $m = \frac{1}{12c_p}$. If the $j$th relevant element stops at the $i$th phase, then (i) it causes $\sum_{\ell=1}^{i} (2^{\ell-1} \cdot 6c_p + 1) = i + 6c_p(2^{i-1} - 1) < 2^{i+1} \cdot 6c_p$ bad queries in total, and (ii) it fail the $i$th test. Therefore, the probability that a relevant element stops at the
The total number of queries for non-relevant elements is at most $24 \cdot c_p \cdot n$ with probability at least $1 - 4^{-n}$.

**Proof.** Let $X_j$ be a random variable corresponding to the number of queries for the $j$th non-relevant element, so that $X = \sum_j X_j$ is the total number of queries for non-relevant elements. We will derive a Chernoff bound for $X$, thus we start by bounding $\mathbb{E}[2^{tX}]$ where $t = \frac{1}{24p}$. If an element stops in the $i$th phase, it takes $\sum_{i=1}^i \left(2^{c-1} \cdot 6cp + 1\right) = i + 6cp(2^i - 1) < 2^{i+1} \cdot 6cp$ queries in total, and it must pass the first $i - 1$ tests. Therefore, the probability that a non-relevant element stops in the $i$th phase is at most the probability that it passes the $(i - 1)\text{th}$ test, which is at most $e^{-3 \cdot 2^{i-1}} < 2^{-2^{i+1}}$. Hence, we have:

$$\mathbb{E} \left[ 2^{tX} \right] \leq \frac{1 + \log n}{2^{12cp}} \cdot 2^{-2^{i+1}} = \sum_{i=1}^{1+\log n} 2^{2^{2^{i+1}}} = \sum_{i=1}^{1+\log n} 2^{2^{i+1}} < \sum_{i=1}^{\infty} 2^{-i} = 1,$$

implying that $\mathbb{E} \left[ 2^{tX} \right] \leq \prod_{j=1}^n \mathbb{E} \left[ 2^{tX} \right] < 1$. Then:

$$\Pr(X \geq 24 \cdot c_p \cdot n) = \Pr \left( \frac{2^{tX}}{n} \geq 2^{2n} \right) \leq \Pr \left( \frac{2^{tX}}{2^{2n}} \geq 2^{2n} \mathbb{E} \left[ \frac{2^{tX}}{2^{2n}} \right] \right) \leq \frac{\mathbb{E} \left[ 2^{tX} \right]}{2^{2n}} \leq 4^{-n}.$$

The above two lemmas allow us to state the following:

**Theorem 25.** FT-Retrievial($\frac{2}{3}n$) can be solved in $O(n)$ worst-case time with probability at least $1 - 2^{-n}$.

**Proof.** The claim directly follows from Lemma 22, Lemma 24 and Lemma 23. Notice that, although the process might take more than $61 \cdot c_p \cdot n$ queries, the probability of such an event is at most $2^{-n}$. Hence, to achieve the stated worst-case complexity, it suffices to stop the process as soon as the number of queries exceeds $61 \cdot c_p \cdot n$. If the process is stopped in this way or if all the $m$ elements are examined but no element passes all the tests, then any arbitrary element is returned.

#### F Omitted Proofs from Section 7

**F.1 Proof of the Chernoff Bound used in Lemma 13**

**Lemma 26.** Let $X_1, \ldots, X_n$ be a sequence of independent Bernoulli random variables with $\Pr(X_i = 1) = p_i$ for $1 \leq i \leq n$, let $X = \sum_{i=1}^n X_i$, and let $\mu = \mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i] = \sum_{i=1}^n p_i$. Then for any $\delta > e$, $\Pr(X \geq \delta \mu) \leq e^{-\frac{\delta^2 \mu}{3}}$.

**Proof.** We first bound $\mathbb{E}[e^{tX_i}]$:

$$\mathbb{E}[e^{tX_i}] = p_i e^t + (1 - p_i) = 1 + p_i (e^t - 1) \leq e^{p_i (e^t - 1)}.$$
Since $X_1, \ldots, X_n$ are independent, we can write:
\[
\mathbb{E}[e^{tX}] = \prod_{i=1}^{n} E[e^{tX_i}] \leq \prod_{i=1}^{n} e^{p_i(e^t - 1)} = \exp \left( \sum_{i=1}^{n} p_i(e^t - 1) \right) = e^{(e^t - 1)p}.
\]

Let $t = \ln \delta$. Since $\delta > e$, $t > 1$, we have:
\[
\Pr(X \geq \delta) = \Pr(e^t X \geq e^t \delta) \leq \frac{E[e^{tX}]}{e^{t \delta}} \leq \frac{e^{(e^t - 1)p}}{e^{t \delta p}} = e^{(e^t - 1)p} = e^{(e - 1) \ln \delta} = e^{e - 1 - \ln \delta} < e^{e - 1 - \ln \delta}.
\]

By hypothesis $\delta > e$ and hence $\delta - 1 - \ln \delta \leq -\frac{\delta \ln \delta}{\delta}$, implying that:
\[
\Pr(X \geq \delta) \leq e^{(\delta - 1 - \ln \delta) \mu} \leq e^{-\frac{\mu \delta}{\ln \delta}}.
\]

**F.2 Proof of Lemma 15**

**Proof.** For an element in $S_{3/4}^+$ to pass the preliminary test and all the following tests, more than half of the queries to $O_2$ for the last test (i.e., the one conducted in phase $\eta$) must return the wrong answer. By Lemma 16 and by the definition of $O_2$, this happens with probability at most $e^{-2^{9/23} \ln n} \leq e^{-\frac{\mu \ln n}{23}} \leq e^{-n} < 2^{-n}$.

**F.3 Proof of Theorem 18**

**Proof.** Since we stop the knockout tournament at the $(\log \log n)^{th}$ iteration, the time required to pre-select the $\frac{n}{\log n}$ element in $S'$ is $O(n \log \log n)$ in the worst-case, as shown by Corollary 11. Moreover, since each query takes $O(1)$ comparisons, Lemma 15 also implies that the modified multi-phase process requires $O(n)$ expected time. Finally, by Lemmas 13 and 16 and by the union bound, the overall failure probability can be at most $2^{-\frac{\mu}{32}} 2^{-\frac{\mu}{32}} < 2^{-\mu}$.

**G Designing the Oracles $O_1$ and $O_2$ used in Section 7.2**

Here we show how to design the two oracles $O_1$ and $O_2$ needed in Section 7.2. In what follows we assume, w.l.o.g., that the probability $p$ of a comparison fault is at most $\frac{1}{3m}$ (if this is not the case then it the strategy of Lemma 15 can be used to achieve the desired error probability). We also let $m = |S| = \frac{n}{\log n}$.

To answer a query to $O_1$ for an element $x \in S'$, it suffices to sample with replacement two elements $x_1, x_2$ from $S'$ and compare them with $x$. If either $x_1$ or $x_2$ appears to be smaller than $x$ we answer the query reporting $x$ to be non-relevant. Otherwise we report $x$ to be relevant. We now show that this strategy satisfies the required constraints and, in particular, we choose $p_1 = \frac{5}{11}$.

If $x \in S_{1/6}^-$ then the probability that $x_1$ (resp. $x_2$) either coincides with $x$ or it is larger than $x$ according to the true ordering of the elements is at least $\frac{m - |S_{1/6}|}{m} \geq \frac{m - \lfloor m/6 \rfloor}{m} \geq \frac{5}{6} - \frac{1}{m}$. Therefore, the probability that the above is true for both $x_1$ and $x_2$ is at least: $\left( \frac{5}{6} - \frac{1}{m} \right)^2 \geq \frac{25}{36} - \frac{5}{3m}$, which is at least $\frac{25}{36} - \frac{5}{3m} \geq \frac{25}{36}$ for sufficiently large values of $m$. By union bound the probability that the two comparisons are correct is at least $1 - \frac{1}{100}$ and hence the probability of correctly reporting $x$ as relevant is at least $1 - \frac{1}{100} - \frac{5}{11} = 1 - \frac{5}{11} = 1 - p_1$.

If $x \in S_{1/3}^+$ then the probability that $x_1$ (resp. $x_2$) is smaller than $x$ according to the true ordering of the elements is at least $\frac{|S_{1/3}|}{m} \geq \frac{\lfloor m/3 \rfloor}{m}$. Therefore, the probability that both $x_1$ and $x_2$ are smaller or equal to $x$ is at most $\left( \frac{1}{3} - \frac{1}{3} \right)^2 = \frac{4}{9}$. Once again, the two comparisons
are correct with a probability of at least $1 - \frac{1}{100}$ and hence $x$ is wrongly reported to be relevant with probability at most $\frac{4}{10} + \frac{1}{100} < \frac{3}{11} = p_1$.

Concerning $O_2$, to answer a query for an element $x \in S'$ it suffices to sample a simple element $x_1$ from $S$ and to compare it with $x$. If $x_1$ is larger than $x$, then $x$ is reported to be relevant, otherwise it is reported to be non-relevant. Once we choose $p_2 = \frac{2}{5}$, it is immediate to check that the probability that $x$ is reported to be relevant is at least $1 - \left(\frac{1}{3} + \frac{1}{m} + \frac{1}{200}\right) \geq 1 - p_2$ if $x \in S_{1/3}$ and at most $\frac{1}{4} + \frac{1}{200} < p_2$ if $x \in S_{3/4}^+$. 