Fragile Complexity of Comparison-Based Algorithms*

Peyman Afshani
Aarhus University
peyman@cs.au.dk

Rolf Fagerberg
University of Southern Denmark
rolf@imada.sdu.dk

David Hammer
Goethe University Frankfurt
University of Southern Denmark
hammer@imada.sdu.dk

Riko Jacob
IT University of Copenhagen
rikj@itu.dk

Irina Kostitsyna
TU Eindhoven
i.kostitsyna@tue.nl

Ulrich Meyer
Goethe University Frankfurt
umeyer@ae.cs.uni-frankfurt.de

Manuel Penschuck
Goethe University Frankfurt
mpenschuck@ae.cs.uni-frankfurt.de

Nodari Sitchinava
University of Hawaii at Manoa
nodari@hawaii.edu

Abstract

We initiate a study of algorithms with a focus on the computational complexity of individual elements, and introduce the *fragile complexity* of comparison-based algorithms as the maximal number of comparisons any individual element takes part in. We give a number of upper and lower bounds on the fragile complexity for fundamental problems, including Minimum, Selection, Sorting and Heap Construction. The results include both deterministic and randomized upper and lower bounds, and demonstrate a separation between the two settings for a number of problems. The depth of a comparator network is a straight-forward upper bound on the worst case fragile complexity of the corresponding fragile algorithm. We prove that fragile complexity is a different and strictly easier property than the depth of comparator networks, in the sense that for some problems a fragile complexity equal to the best network depth can be achieved with less total work and that with randomization, even a lower fragile complexity is possible.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Algorithms, comparison based algorithms, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.2

* This material is based upon work partially performed while attending AlgoPARC Workshop on Parallel Algorithms and Data Structures at the University of Hawaii at Manoa, in part supported by the National Science Foundation under Grant No. CCF-1745331.

27th Annual European Symposium on Algorithms (ESA 2019).
Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 2; pp. 2:1–2:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version https://arxiv.org/abs/1901.02857

Funding *Rolf Fagerberg*: Supported by the Independent Research Fund Denmark, Natural Sciences, grant DFF-7014-00041.
David Hammer: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.
Ulrich Meyer: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.
Manuel Penschuck: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.
Nodari Sitchinava: Supported by the National Science Foundation under Grant No. CCF-1533823.

Acknowledgements We thank Steven Skiena for posing the original problem, and we thank Michael Bender, Rob Johnson, and Pat Morin for helpful discussions.

1 Introduction

Comparison-based algorithms is a classic and fundamental research area in computer science. Problems studied include minimum, median, sorting, searching, dictionaries, and priority queues, to name a few, and by now a huge body of work exists. The cost measure analyzed is almost always the total number of comparisons needed to solve the problem, either in the worst case or the expected case. Surprisingly, very little work has taken the viewpoint of the individual elements, asking the question: *how many comparisons must each element be subjected to?*

This question not only seems natural and theoretically fundamental, but is also practically well motivated: in many real world situations, comparisons involve some amount of destructive impact on the elements being compared, hence, keeping the maximum number of comparisons for each individual element low can be important. One example of such a situation is ranking of any type of consumable objects (wine, beer, food, produce), where each comparison reduces the available amount of the objects compared. Here, classical algorithms like QuickSort, which takes a single object and partitions the whole set with it, may use up this pivot element long before the algorithm completes. Another example is sports, where each comparison constitutes a match and takes a physical toll on the athletes involved. If a comparison scheme subjects one contestant to many more matches than others, both fairness to contestants and quality of result are impacted. The selection process could even contradict its own purpose—what is the use of finding a national boxing champion to represent a country at the Olympics if the person is injured in the process? Notice that in both examples above, quality of elements is difficult to measure objectively by a numerical value, hence one has to resort to relative ranking operations between opponents, i.e., comparisons. The detrimental impact of comparisons may also be of less directly physical nature, for instance if it involves a privacy risk for the elements compared, or if bias in the comparison process grows each time the same element is used.

▶ **Definition 1.** *We say that a comparison-based algorithm $\mathcal{A}$ has* fragile complexity *$f(n)$ if each individual input element participates in at most $f(n)$ comparisons. We also say that $\mathcal{A}$ has* work *$w(n)$ if it performs at most $w(n)$ comparisons in total. We say that a particular element e has fragile complexity $f_e(n)$ in A if e participates in at most $f_e(n)$ comparisons.*

In this paper, we initiate the study of algorithms' fragile complexity—comparison-based complexity from the viewpoint of the individual elements—and present a number of upper and lower bounds on the fragile complexity for fundamental problems.

1.1 Previous work

One body of work relevant to what we study here is the study of sorting networks, propelled by the 1968 paper of Batcher [5]. In sorting networks, and more generally comparator networks, the notions of depth and size correspond to fragile complexity and standard worst case complexity,[1] respectively, since a network with depth $f(n)$ and size $w(n)$ easily can be converted into a comparison-based algorithm with fragile complexity $f(n)$ and work $w(n)$.

Batcher gave sorting networks with $\mathcal{O}(\log^2 n)$ depth and $\mathcal{O}(n \log^2 n)$ size, based on clever variants of the MergeSort paradigm. A number of later constructions achieve the same bounds [9, 14, 15, 18], and for a long time it was an open question whether better results were possible. In the seminal result in 1983, Ajtai, Komlós, and Szemerédi [1, 2] answered this in the affirmative by constructing a sorting network of $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n \log n)$ size. This construction is quite complex and involves expander graphs [20, 21], which can be viewed as objects encoding pseudorandomness, and which have many powerful applications in computer science and mathematics. The size of the constant factors in the asymptotic complexity of the AKS sorting network prevents it from being practical in any sense. It was later modified by others [7, 11, 16, 19], but finding a simple, optimal sorting network, in particular one not based on expander graphs, remains an open problem. Comparator networks for other problems, such as selection and heap construction have also been studied [4, 6, 13, 17, 22]. In all these problems the size of the network is super-linear.

As comparator networks of depth $f(n)$ and size $w(n)$ lead to comparison-based algorithms with $f(n)$ fragile complexity and $w(n)$ work, a natural question is, whether the two models are equivalent, or if there are problems for which comparison-based algorithms can achieve either asymptotically lower $f(n)$, or asymptotically lower $w(n)$ for the same $f(n)$.

One could also ask about the relationship between parallelism and fragile complexity. We note that parallel time in standard parallel models generally does not seem to capture fragile complexity. For example, even in the most restrictive exclusive read and exclusive write (EREW) PRAM model it is possible to create n copies of an element e in $\mathcal{O}(\log n)$ time and, thus, compare e to all the other input elements in $\mathcal{O}(\log n)$ time, resulting in $\mathcal{O}(\log n)$ parallel time but $\Omega(n)$ fragile complexity. Consequently, it is not clear whether Richard Cole's celebrated parallel merge sort algorithm [8] yields a comparison-based algorithm with low fragile complexity as it copies some elements.

1.2 Our contribution

In this paper we present algorithms and lower bounds for a number of classical problems, summarized in Table 1. In particular, we study finding the Minimum (Section 2), the Selection problem (Section 3), and Sorting (Section 4).

Minimum. The case of the deterministic algorithms is clear: using an adversary lower bound, we show that the minimum element needs to suffer $\Omega(\log n)$ comparisons and a tournament tree trivially achieves this bound (Subsection 2.1). The randomized case, however, is much more interesting. We obtain a simple algorithm where the probability of the minimum element suffering k comparisons is doubly exponentially low in k, roughly $1/2^{2^k}$ (see Subsection 2.2). As a result, the $\Theta(\log n)$ deterministic fragile complexity can be lowered to $O(1)$ expected or even $O(\log \log n)$ with high probability. We also show this latter high probability case is lower

[1] For clarity, in the rest of the paper we call standard worst case complexity *work*.

Problem		*Upper*		*Lower*
		$f(n)$	$w(n)$	$f(n)$
Minimum	Determ. (Sec. 2)	$\mathcal{O}(\log n)$ [T 2]	$\mathcal{O}(n)$	$f_{\min} = \Omega(\log n)$ [T 2]
	Rand. (Sec. 2)	$\langle \mathcal{O}(\log_\Delta n)^\dagger, \mathcal{O}(\Delta + \log_\Delta n)^\dagger \rangle$ [T 9] $\langle \mathcal{O}(1)^\dagger, \mathcal{O}(n^\varepsilon) \rangle$ (setting $\Delta = n^\varepsilon$) $O(\frac{\log n}{\log\log n})^\dagger$ [Cor 11]	$\mathcal{O}(n)$	$\langle \Omega(\log_\Delta n)^\dagger, \Delta \rangle$ [T 10] $\Omega(\frac{\log n}{\log\log n})^\dagger$ [Cor 11]
		$\langle O(\log_\Delta n \log\log\Delta)^\ddagger,$ $O(\Delta + \log_\Delta n \log\log\Delta)^\ddagger \rangle$ [T 9]	$\mathcal{O}(n)$ $\mathcal{O}(n)$	$f_{\min} =$ $= \Omega(\log\log n)^\ddagger$ [T 14]
Selection	Determ. (Sec. 3)	$\mathcal{O}(\log n)$ [T 15]	$\mathcal{O}(n)$ [T 15]	$\Omega(\log n)$ [Cor 3]
	Rand. (Sec. 3)	$\left\langle \mathcal{O}(\log\log n)^\dagger, \mathcal{O}\left(\sqrt{n}\right)^\dagger \right\rangle$ [T 17] $\left\langle \mathcal{O}\left(\frac{\log n}{\log\log n}\right)^\dagger, \mathcal{O}(\log^2 n)^\dagger \right\rangle$ [T 17]	$\mathcal{O}(n)^\dagger$	$\langle \Omega(\log_\Delta n)^\dagger, \Delta \rangle$ [T 10]
Merge	Determ. (Sec. 4)	$\mathcal{O}(\log n)$ [T 24]	$\mathcal{O}(n)$	$\Omega(\log n)$ [Lem 18]
Heap Constr.	Determ. (Sec. 5)	$\mathcal{O}(\log n)$ [Obs 26]	$\mathcal{O}(n)$	$\Omega(\log n)$ [T 2]

Table 1 Summary of presented results. Notation: $f(n)$ means fragile complexity; $w(n)$ means work; $\langle f_m(n), f_{rem}(n) \rangle$ means fragile complexity for the selected element (minimum/median) and for the remaining elements, respectively – except for lower bounds, where it means ⟨expected for the selected, limit for remaining⟩; † means holds in expectation; ‡ means holds with high probability $(1 - 1/n)$. $\varepsilon > 0$ is an arbitrary constant.

bounded by $\Omega(\log\log n)$ (Subsection 2.3). Furthermore, we can achieve a trade-off between the fragile complexity of the minimum element and the other elements. Here $\Delta = \Delta(n)$ is a parameter we can choose freely that basically upper bounds the fragile complexity of the non-minimal elements. We can find the minimum with $O(\log_\Delta n)$ expected fragile complexity while all the other elements suffer $O(\Delta + \log_\Delta n)$ comparisons (Subsection 2.3). Furthermore, this is tight: we show an $\Omega(\log_\Delta n)$ lower bound for the expected fragile complexity of the minimum element where the maximum fragile complexity of non-minimum elements is at most Δ.

Selection. Minimum finding is a special case of the selection problem where we are interested in finding an element of a given rank. As a result, all of our lower bounds apply to this problem as well. Regarding upper bounds, the deterministic case is trivial if we allow for $O(n \log n)$ work (via sorting). We show that this can be reduced to $O(n)$ time while keeping the fragile complexity of all the elements at $O(\log n)$ (Section 3). Once again, randomization offers a substantial improvement: e.g., we can find the median in $O(n)$ expected work and with $O(\log\log n)$ expected fragile complexity while non-median elements suffer $O(\sqrt{n})$ expected comparisons, or we can find the median in $O(n)$ expected work and with $O(\log n/\log\log n)$ expected fragile complexity while non-median elements suffer $O(\log^2 n)$ expected comparisons.

Sorting and other results. The deterministic selection, sorting, and heap construction fragile complexities follow directly from the classical results in comparator networks [2, 6]. However, we show a separation between comparator networks and comparison-based algorithms for the problem of Median (Section 3) and Heap Construction (Section 5), in the sense that depth/fragile complexity of $\mathcal{O}(\log n)$ can be achieved in $\mathcal{O}(n)$ work for comparison-based

algorithms, but requires $\Omega(n \log n)$ [4] and $\Omega(n \log \log n)$ [6] sizes for comparator networks for the two problems, respectively. For sorting the two models achieve the same complexities: $\mathcal{O}(\log n)$ depth/fragile complexity and $\mathcal{O}(n \log n)$ size/work, which are the optimal bounds in both models due to the $\Omega(\log n)$ lower bound on fragile complexity for Minimum (Theorem 2) and the standard $\Omega(n \log n)$ lower bound on work for comparison-based sorting. However, it is an open problem whether these bounds can be achieved by simpler sorting algorithms than sorting networks, in particular whether expander graphs are necessary. One intriguing conjecture could be that any comparison-based sorting algorithm with $\mathcal{O}(\log n)$ fragile complexity and $\mathcal{O}(n \log n)$ work implies an expander graph. This would imply expanders, optimal sorting networks and fragile-optimal comparison-based sorting algorithms to be equivalent, in the sense that they all encode the same level of pseudorandomness.

We note that our lower bound of $\Omega(\log^2 n)$ on the fragile complexity of MergeSort (Theorem 19) implies the same lower bound on the depth of any sorting network based on binary merging, which explains why many of the existing simple sorting networks have $\Theta(\log^2 n)$ depth. Finally, our analysis of MergeSort on random inputs (Theorem 23) shows a separation between deterministic and randomized fragile complexity for such algorithms. In summary, we consider the main contributions of this paper to be:

- the introduction of the model of fragile complexity, which we find intrinsically interesting, practically relevant, and surprisingly overlooked
- the separations between this model and the model of comparator networks
- the separations between the deterministic and randomized setting within the model
- the lower bounds on randomized minimum finding

Due to space constraints, some proofs only appear in the full paper [**?**].

2 Finding the minimum

2.1 Deterministic Algorithms

As a starting point, we study deterministic algorithms that find the minimum among an input of n elements. Our results here are simple but they act as interesting points of comparison against the subsequent non-trivial results on randomized algorithms.

▶ **Theorem 2.** *The fragile complexity of finding the minimum of n elements is $\lceil \log n \rceil$.*

Proof. The upper bound is achieved using a perfectly balanced tournament tree. The lower bound follows from a standard adversary argument. ◀

Observe that in addition to returning the minimum, the balanced tournament tree can also return the second smallest element, without any increase to the fragile complexity of the minimum. We refer to this deterministic algorithm that returns the smallest and the second smallest element of a set X as TournamentMinimum(X).

▶ **Corollary 3.** *For any deterministic algorithm $\mathcal{A}$ that finds the median of n elements, the fragile complexity of the median element is at least $\lceil \log n \rceil - 1$.*

Proof. By a standard padding argument with $n - 1$ small elements. ◀

2.2 Randomized Algorithms for Finding the Minimum

We now show that finding the minimum is provably easier for randomized algorithms than for deterministic algorithms. We define $f_{\min}$ as the fragile complexity of the minimum

and f_{rem} as the maximum fragile complexity of the remaining elements. For deterministic algorithms we have shown that $f_{\min} \geq \log n$ regardless of f_{rem}. This is very different in the randomized setting. In particular, we first show that we can achieve $\mathbb{E}[f_{\min}] = O(1)$ and $f_{\min} = O(1) + \log\log n$ with high probability (we later show this high probability bound is also tight, Theorem 14).

```
1:  procedure SampleMinimum(X) ▷ Returns the smallest and 2nd smallest element of X
2:      if |X| ≤ 8 return TournamentMinimum(X)
3:      Let A ⊂ X be a uniform random sample of X, with |A| = ⌈|X|/2⌉
4:      Let B ⊂ A be a uniform random sample of A, with |B| = ⌊|X|^{2/3}⌋
5:                  ▷ The minimum is either in (i) C ⊆ X \ A, (ii) D ⊆ A \ B or (iii) B
6:      (b1, b2) = SampleMinimum(B)        ▷ the minimum participates only in case (iii)
7:      Let D = {x ∈ A \ B | x < b2}        ▷ the minimum is compared once only in case (ii)
8:      Let (a'1, a'2) = SampleMinimum(D)                          ▷ only case (ii)
9:      Let (a1, a2) = TournamentMinimum(a'1, a'2, b1, b2)         ▷ case (ii) and (iii)
10:     Let C = {x ∈ X \ A | x < a2}                               ▷ only case (i)
11:     Let (c1, c2) = TournamentMinimum(C)                        ▷ only case (i)
12:     return TournamentMinimum(a1, a2, c1, c2)                   ▷ always
```

First, we show that this algorithm can actually find the minimum with expected constant number of comparisons. Later, we show that the probability that this algorithm performs t comparisons on the minimum drops roughly *doubly exponentially* on t.

We start with the simple worst-case analysis.

▶ **Lemma 4.** *Algorithm* SampleMinimum*(X) achieves $f_{min} \leq 3\log|X|$ in the worst case.*

Proof. First, observe that the smallest element in Lines 9 and 12 participates in at most one comparison because pairs of elements are already sorted. Then the fragile complexity of the minimum is defined by the maximum of the three cases:

- **(i)** One comparison each in Lines 10 and 12, plus (by Theorem 2) $\lceil\log|C|\rceil \leq \log|X|$ comparisons in Line 11.
- **(ii)** One comparison each in Lines 7, 9, and 12, plus the recursive call in line 8.
- **(iii)** One comparison each in Lines 6, 9, and 12, plus the recursive call in line 6.

The recursive calls in lines 8 and 6 are on at most $|X|/2$ elements because $B \subset A$, $D \subset A$, and $|A| = \lceil|X|/2\rceil$. Consequently, the fragile complexity of the minimum is defined by the recurrence

$$T(n) \leq \begin{cases} \max\{3 + T(n/2), 2 + \log n\} & \text{if } n > 8 \\ 3 & \text{if } n \leq 8 \end{cases},$$

which solves to $T(n) \leq 3\log n$.

◀

▶ **Lemma 5.** *Assume that in Algorithm* SampleMinimum*, the minimum y is in $X \setminus A$, i.e. we are in case (i). Then $\mathbf{Pr}[|C| = k \mid y \notin A] \leq \frac{k}{2^k}$ for any $k \geq 1$ and $n \geq 7$.*

Proof. There are $\binom{n-1}{\lceil n/2\rceil}$ possible events of choosing a random subset $A \subset X$ of size $\lceil n/2\rceil$ s.t. $y \notin A$. Let us count the number of the events $\{|C| = k \mid y \notin A\}$, which is equivalent to a_2, the second smallest element of A, being larger than exactly $k+1$ elements of X.

For simplicity of exposition, consider the elements of $X = \{x_1, \ldots, x_n\}$ in sorted order. The minimum $y = x_1 \notin A$, therefore, a_1 (the smallest element of A) must be one of the k elements $\{x_2, \ldots, x_{k+1}\}$. By the above observation, $a_2 = x_{k+2}$. And the remaining $\lceil n/2\rceil - 2$ elements of A are chosen from among $\{x_{k+3}, \ldots, x_n\}$. Therefore,

$$\mathbf{Pr}[|C| = k \mid y \notin A] = \frac{k \cdot \binom{n-(k+2)}{\lceil n/2 \rceil - 2}}{\binom{n-1}{\lceil n/2 \rceil}} = k \cdot \frac{(n-(k+2))!}{(\lfloor n/2 \rfloor - k)!(\lceil n/2 \rceil - 2)!} \cdot \frac{(\lceil n/2 \rceil)!(\lfloor n/2 \rfloor - 1)!}{(n-1)!}$$

Rearranging the terms, we get:

$$\mathbf{Pr}[|C| = k \mid y \notin A] = k \cdot \frac{(n-(k+2))!}{(n-1)!} \cdot \frac{(\lceil n/2 \rceil)!}{(\lceil n/2 \rceil - 2)!} \cdot \frac{(\lfloor n/2 \rfloor - 1)!}{(\lfloor n/2 \rfloor - k)!}$$

There are two cases to consider:

$$\begin{aligned} k = 1: \qquad \mathbf{Pr}[|C| = k \mid y \notin A] &= 1 \cdot \frac{1}{(n-1)(n-2)} \cdot \lceil n/2 \rceil \left(\lceil n/2 \rceil - 1\right) \cdot 1 \\ &\leq \frac{1}{(n-1)(n-2)} \cdot \frac{(n+1)}{2} \cdot \frac{(n-1)}{2} \\ &= \frac{n+1}{4 \cdot (n-2)} \leq \frac{1}{2} = \frac{k}{2^k} \qquad \text{for every } n \geq 5. \end{aligned}$$

$$\begin{aligned} k \geq 2: \qquad \mathbf{Pr}[|C| = k \mid y \notin A] &= k \cdot \frac{1}{\prod_{i=1}^{k+1}(n-i)} \cdot \lceil n/2 \rceil \left(\lceil n/2 \rceil - 1\right) \cdot \prod_{i=1}^{k-1} \left(\left\lfloor \frac{n}{2} \right\rfloor - i\right) \\ &\leq k \cdot \frac{1}{\prod_{i=1}^{k+1}(n-i)} \cdot \frac{n+1}{2} \cdot \frac{n-1}{2} \cdot \prod_{i=1}^{k-1} \frac{n-2i}{2} \\ &\leq \frac{k}{2^{k+1}} \cdot (n+1)(n-1) \cdot \frac{\prod_{i=1}^{k-1}(n-2i)}{\prod_{i=1}^{k+1}(n-i)} \\ &\leq \frac{k}{2^{k+1}} \cdot (n+1)(n-1) \cdot \frac{n-2}{(n-1)(n-2)(n-3)} \\ &= \frac{k}{2^{k+1}} \cdot \frac{n+1}{n-3} \leq \frac{k}{2^{k+1}} \cdot 2 = \frac{k}{2^k} \qquad \text{for every } n \geq 7. \end{aligned}$$

◀

▶ **Theorem 6.** *Algorithm* SampleMinimum *achieves* $\mathbb{E}[f_{min}] \leq 9$.

Proof. By induction on the size of X. In the base case $|X| \leq 8$, clearly $f_{\min} \leq 3$, implying the theorem.

Now assume that the calls in Line 8 and Line 6 have the property that $\mathbb{E}[f(b_1)] \leq 9$ and $\mathbb{E}[f(a'_1)] \leq 9$. Both in case (ii) and case (iii), the expected number of comparisons of the minimum is $\leq 9 + 3$. Case (i) happens with probability at least $1/2$. In this case, the expected number of comparisons is 2 plus the ones from Line 11. By Lemma 5 we have $\mathbf{Pr}[|C| = k \mid \text{case (i)}] \leq k2^{-k}$. Because TournamentMinimum (actually any algorithm not repeating the same comparison) uses the minimum at most $k-1$ times, the expected number of comparisons in Line 11 is $\sum_{k=1}^{\lfloor n/2 \rfloor}(k-1)k2^{-k} \leq \sum_{k=1}^{\infty}(k-1)k2^{-k} \leq 4$. Combining the bounds we get $\mathbb{E}[f_{\min}] \leq \frac{9+3}{2} + \frac{2+4}{2} = 9$. ◀

Observe that the above proof did not use anything about the sampling of B, and also did not rely on TournamentMinimum.

▶ **Lemma 7.** *For $|X| > 2$ and any $\gamma > 1$:* $\mathbf{Pr}\left[|D| \geq \gamma |X|^{1/3}\right] < |X| \exp(-\Theta(\gamma))$

Proof. Let $n = |X|$, $a = |A| = \lceil n/2 \rceil$ and $b = |B| = \left\lfloor n^{2/3} \right\rfloor$. The construction of the set B can be viewed as the following experiment. Consider drawing without replacement from an urn with b blue and $a - b$ red marbles. The i-th smallest element of A is chosen into B iff the i-th draw from the urn results in a blue marble. Then $|D| \geq \gamma |X|^{1/3} = \gamma n^{1/3}$ implies that this experiment results in at most one blue marble among the first $t = \gamma n^{1/3}$ draws. There are precisely $t + 1$ elementary events that make up the condition $|D| \geq t$, namely that the i-th draw is a blue marble, and where $i = 0$ stands for the event "all t marbles are red". Let us denote the probabilities of these elementary events as p_i.

Observe that each p_i can be expressed as a product of t factors, at least $t - 1$ of which stand for drawing a red marble, each upper bounded by $1 - \frac{b-1}{a}$. The remaining factor stands for drawing the first blue marble (from the urn with $a - i$ marbles, b of which are blue), or another red marble. In any case we can bound

$$p_i \leq \left(1 - \frac{b-1}{a}\right)^{t-1} \leq \left(1 - \frac{b-1}{a}\right)^{\gamma n^{1/3} - 1} = \exp\left(-\Theta\left(\frac{b\gamma n^{1/3}}{a}\right)\right).$$

Summing the $t + 1$ terms, and observing $t + 1 < n$ if the event can happen at all, we get

$$\mathbf{Pr}[|D| \geq \gamma |X|^{1/3}] < n \cdot \exp\left(-\Theta\left(\frac{\gamma n^{1/3} n^{2/3}}{n/2}\right)\right) = n \cdot \exp\left(-\Theta(\gamma)\right).$$

◀

▶ **Theorem 8.** *There is a positive constant c, such that for any parameter $t \geq c$, the minimum in the Algorithm* SampleMinimum*(X) participates in at most $O(t + \log\log|X|)$ comparisons with probability at least $1 - \exp(-2^t) 2 \log\log|X|$.*

Proof. Let $n = |X|$ and y be the minimum element. In each recursion step, we have one of three cases: (i) $y \in C \subseteq X \setminus A$, (ii) $y \in D \subseteq A \setminus B$ or (iii) $y \in B$. Since the three sets are disjoint, the minimum always participates in at most one recursive call. Tracing only the recursive calls that include the minimum, we use the superscript $X^{(i)}, A^{(i)}, B^{(i)}, C^{(i)}$, and $D^{(i)}$ to denote these sets at depth i of the recursion.

Let h be the first recursive level when $y \in C^{(h)}$, i.e., $y \notin A^{(h)}$. It follows that y will not be involved in the future recursive calls because it is in a single call to TournamentMinimum. Thus, at this level of recursion, the number of comparisons that y will accumulate is equal to $O(1) + \log|C^{(h)}|$. To bound this quantity, let $k = 4 \cdot 2^t$. Then, by Lemma 5, $\mathbf{Pr}[|C^{(h)}| > k] \leq k 2^{-k} = 4 \cdot 2^t \cdot 2^{-4 \cdot 2^t} = 4 \cdot 2^t \cdot 4^{-2^t} \cdot 4^{-2^t}$. Since $4x4^{-x} \leq 1$ for any $x \geq 1$, $\mathbf{Pr}[|C^{(h)}| > k] \leq 4^{-2^t}$ for any $t \geq 0$. I.e., the number of comparisons that y participates in at level h is at most $O(1) + \log k = O(1) + t$ with probability at least $1 - 4^{-2^t} \geq 1 - \exp(-2^t)$.

Thus, it remains to bound the number of comparisons involving y at the recursive levels $i \in [1, h-1]$. In each of these recursive levels $y \notin C^{(i)}$, which only leaves the two cases: (ii) $y \in D^{(i)} \subseteq A^{(i)} \setminus B^{(i)}$ and (iii) $y \in B^{(i)}$. The element y is involved in at most $O(1)$ comparisons in lines 7, 9 and 12. The two remaining lines of the algorithm are lines 6 and 8 which are the recursive calls. We differentiate two types of recursive calls:

- Type 1: $|X^{(i)}| \leq 2^{4t}$. In this case, by Lemma 4, the algorithm will perform $O(t)$ comparisons at the recursive level i, as well as any subsequent recursive levels.
- Type 2: $|X^{(i)}| > 2^{4t}$. In this case, by Lemma 7 on the set $X^{(i)}$ and $\gamma = |X^{(i)}|^{1/3}$ we get:

$$\mathbf{Pr}[|D^{(i)}| \geq \gamma |X^{(i)}|^{1/3}] < |X^{(i)}| \exp\left(-\Theta\left(|X^{(i)}|^{1/3}\right)\right) < \exp\left(-\Theta\left(|X^{(i)}|^{1/3}\right)\right)$$

Note that since $|X^{(i)}|^{1/3} > 2^t$, by the definition of the Θ-notation, there exists a positive constant c, such that $\exp\left(-\Theta\left(|X^{(i)}|^{1/3}\right)\right) < \exp(-2^t)$. Thus, it follows that with probability $1-\exp(-2^t)$, we will recurse on a subproblem of size at most $\gamma|X^{(i)}|^{1/3} \leq |X^{(i)}|^{2/3}$. Let G_i be this (good) event, and thus $\mathbf{Pr}[G_i] \geq 1-\exp(-2^t)$.

Observe that the maximum number of times we can have good events of type 2 is very limited. With every such good event, the size of the subproblem decreases significantly and thus eventually we will arrive at a recursive call of type 1. Let j be this maximum number of "good" recursive levels of type 2. The problem size at the j-th such recursive level is at most $n^{(2/3)^{j-1}}$ and we must have that $n^{(2/3)^{j-1}} > 2^{4t}$ which reveals that we must have $j = O(\log\log n)$.

We are now almost done and we just need to use a union bound. Let G be the event that at the recursive level h, we perform at most $O(1)+t$ comparisons, and all the recursive levels of type 2 are good. G is the conjunction of at most $j+1$ events and as we have shown, each such event holds with probability at least $1-\exp(-2^t)$. Thus, it follows that G happens with probability $1-(j+1)\exp(-2^t) > 1-2\log\log n\exp(-2^t)$. Furthermore, our arguments show that if G happens, then the minimum will only particpate in $O(t+j) = O(t+\log\log n)$ comparisons. ◀

The major strengths of the above algorithm is the doubly exponential drop in probability of comparing the minimum with too many elements. Based on it, we can design another simple algorithm to provide a smooth trade-off between $f_{\min}$ and f_{rem}. Let $2 \leq \Delta \leq n$ be an integral parameter. We will design an algorithm that achieves $\mathbb{E}[f_{\min}] = O(\log_\Delta n)$ and $f_{\min} = O(\log_\Delta n \cdot \log\log\Delta)$ whp, and $f_{\text{rem}} = \Delta + O(\log_\Delta n \cdot \log\log\Delta)$ whp. For simplicity we assume n is a power of Δ. We build a fixed tournament tree T of degree Δ and of height $\log_\Delta n$ on X. For a node $v \in T$, let $X(v)$ be the set of values in the subtree rooted at v. The following code computes $m(v)$, the minimum value of $X(v)$, for every node v.

1: **procedure** TreeMinimum$_\Delta(X)$
2: For every leaf v, set $m(v)$ equal to the single element of $X(v)$.
3: For every internal node v with Δ children $u_1,\ldots,u_\Delta$ where the values $m(u_1),\ldots,m(u_\Delta)$ are known, compute $m(v)$ using SimpleMinimum algorithm on input $\{m(u_1),\ldots,m(u_\Delta)\}$.
4: Repeat the above step until the minimum of X is computed.

The correctness of TreeMinimum$_\Delta$ is trivial. So it remains to analyze its fragile complexity.

▶ **Theorem 9.** *In* TreeMinimum$_\Delta$, $\mathbb{E}[f_{min}] = O(\log_\Delta n)$ *and* $\mathbb{E}[f_{rem}] = \Delta + O(\log_\Delta n)$. *Furthermore, with high probability,* $f_{min} = O\left(\frac{\log n \log\log\Delta}{\log\Delta}\right)$ *and* $f_{rem} = O\left(\Delta + \frac{\log n \log\log\Delta}{\log\Delta}\right)$.

Proof. First, observe that $\mathbb{E}[f_{\min}] = O(\log_\Delta n)$ is an easy consequence of Theorem 6. Now we focus on high probability bounds. Let $k = c \cdot h\log\ln\Delta$, and $h = \log_\Delta n$ for a large enough constant c. There are h levels in T. Let $\mathbf{f}_i$ be the random variable that counts the number of comparisons the minimum participates in at level i of T. Observe that these are independent random variables. Let $f_1,\ldots,f_h$ be integers such that $f_i \geq 1$ and $\sum_{i=1}^h f_i = k$, and let c' be the constant hidden in the big-O notation of Theorem 8. Use Theorem 8 h times (with n set to Δ, and $t = f_i$), and also bound $2\log\log\Delta < \Delta$ to get

$$\mathbf{Pr}\Big[\mathbf{f}_1 \geq c'(f_1+\log\log\Delta) \vee \cdots \vee \mathbf{f}_h \geq c'(f_h+\log\log\Delta)\Big] \leq \Delta^h e^{-\sum_i 2^{f_i}} \leq \Delta^h e^{-h2^{k/h}}$$

where the last inequality follows from the inequality of arithmetic and geometric means (specifically, observe that $\sum_{i=1}^h 2^{f_i}$ is minimized when all f_i's are distributed evenly).

Now observe that the total number of different integral sequences $f_1, \ldots, f_h$ that sum up to k is bounded by $\binom{h+k}{h}$ (this is the classical problem of distributing k identical balls into h distinct bins). Thus, we have

$$\mathbf{Pr}[f_{\min} = O(k + h \log\log \Delta)] \leq \binom{h+k}{h} \cdot \Delta^h \frac{1}{e^{h \cdot 2^{k/h}}} \leq \left(\frac{e(h+k)}{h}\right)^h \cdot \Delta^h \frac{1}{e^{h \cdot 2^{k/h}}}$$
$$\leq \left(\frac{O\left(\frac{k}{h}\right) \cdot \Delta}{e^{2^{k/h}}}\right)^h = \left(\frac{O\left(\Delta^2\right)}{e^{2^{c \log\ln \Delta}}}\right)^h < \left(\frac{O(\Delta^2)}{e^{\ln^c \Delta}}\right)^h < \left(\frac{\Delta^3}{\Delta^{\ln^{c-1} \Delta}}\right)^h < \Delta^{-ch} = n^{-c}$$

where in the last step we bound $(\ln \Delta)^{c-1} - 3 > c$ for large enough c and $\Delta \geq 3$. This is a high probability bound for $f_{\min}$. To bound f_{rem}, observe that for every non-minimum element x, there exists a lowest node v such that x is not $m(v)$. If x is not passed to the ancestors of v, x suffers at most Δ comparisons in v, and below v x behaves like the minimum element, which means that the above analysis applies. This yields that whp we have $f_{\text{rem}} = \Delta + O\left(\frac{\log n \log\log \Delta}{\log \Delta}\right)$. ◀

2.3 Randomized Lower Bounds for Finding the Minimum

2.3.1 Expected Lower Bound for the Fragile Complexity of the Minimum.

The following theorem is our main result.

▶ **Theorem 10.** *In any randomized minimum finding algorithm with fragile complexity of at most Δ for any element, the expected fragile complexity of the minimum is at least $\Omega(\log_\Delta n)$.*

Note that this theorem implies the fragile complexity of finding the minimum:

▶ **Corollary 11.** *Let $f(n)$ be the expected fragile complexity of finding the minimum (i.e. the smallest function such that some algorithm achieves $f(n)$ fragile complexity for all elements (minimum and the rest) in expectation). Then $f(n) = \Theta(\frac{\log n}{\log\log n})$.*

Proof. Use Theorem 9 as the upper bound and Theorem 10, both with $\Delta = \frac{\log n}{\log\log n}$, observing that if $f(n)$ is an upper bound that holds with high probability, it is also an upper bound on the expectation. ◀

To prove Theorem 10 we give a lower bound for a *deterministic* algorithm $\mathcal{A}$ on a random input of n values, $x_1, \ldots, x_n$ where each x_i is chosen iid and uniformly in $(0, 1)$. By Yao's minimax principle, the lower bound on the expected fragile complexity of the minimum when running $\mathcal{A}$ also holds for any randomized algorithm.

We prove our lower bound in a model that we call "comparisons with additional information (CAI)": if the algorithm $\mathcal{A}$ compares two elements x_i and x_j and it turns out that $x_i < x_j$, then the value x_j is revealed to the algorithm. Clearly, the algorithm can only do better with this extra information. The heart of the proof is the following lemma which also acts as the "base case" of our proof.

▶ **Lemma 12.** *Let Δ be an upper bound on f_{rem}. Consider T values $x_1, \ldots, x_T$ chosen iid and uniformly in $(0, b)$. Consider a deterministic algorithm $\mathcal{A}$ in CAI model that finds the minimum value y among $x_1, \ldots, x_T$. If $T > 1000\Delta$, then with probability at least $\frac{7}{10}$ $\mathcal{A}$ will compare y against an element x such that $x \geq b/(100\Delta)$.*

Proof. By simple scaling, we can assume $b = 1$. Let p be the probability that $\mathcal{A}$ compares y against a value larger than $1/(100\Delta)$. Let I_{small} be the set of indices i such that $x_i < 1/(100\Delta)$. Let $\mathcal{A}'$ be a deterministic algorithm in CAI model such that:

- $\mathcal{A}'$ is given all the indices in I_{small} (and their corresponding values) except for the index of the minimum. We call these the known values.
- $\mathcal{A}'$ minimizes the probability p' of comparing the y against a value larger than $1/(100\Delta)$.
- $\mathcal{A}'$ finds the minimum value among the unknown values.

Since $p' \leq p$, it suffices to bound p' from below. We do this in the remainder of the proof.

Observe that the expected number of values x_i such that $x_i < 1/(100\Delta)$ is $T/(100\Delta)$. Thus, by Markov's inequality, $\mathbf{Pr}[|I_{\text{small}}| \leq T/(10\Delta)] \geq \frac{9}{10}$. Let's call the event $|I_{\text{small}}| \leq T/(10\Delta)$ the good event. For algorithm $\mathcal{A}'$ all values smaller than $1/(100\Delta)$ except for the minimum are known. Let U be the set of indices of the unknown values. Observe that a value x_i for $i \in U$ is either the minimum or larger than $1/(100\Delta)$, and that $|U| = T - |I_{\text{small}}| + 1 > \frac{9}{10}T$ (using $\Delta \geq 1$) in the good event. Because $\mathcal{A}'$ is a deterministic algorithm, the set U is split into set F of elements that have their first comparison against a known element, and set W of those that are first compared with another element with index in U. Because of the global bound Δ on the fragile complexity of the known elements, we know $|F| < \Delta \cdot |I_{\text{small}}| \leq \Delta T/(10\Delta) = T/10$. Combining this with the probability of the good event, by union bound, the probability of the minimum being compared with a value greater than $1/(100\Delta)$ is at least $1 - (1 - \frac{9}{10}) - (1 - \frac{8}{9}) \geq 7/10$. ◀

Based on the above lemma, our proof idea is the following. Let $G = 100\Delta$. We would like to prove that on average $\mathcal{A}$ cannot avoid comparing the minimum to a lot of elements. In particular, we show that, with constant probability, the minimum will be compared against some value in the range $[G^{-i}, G^{-i+1}]$ for every integer i, $1 \leq i \leq \frac{\log_G n}{2}$. Our lower bound then follows by an easy application of the linearity of expectations. Proving this, however, is a little bit tricky. However, observe that Lemma 12 already proves this for $i = 1$. Next, we use the following lemma to apply Lemma 12 over all values of i, $1 \leq i \leq \frac{\log_G n}{2}$.

▶ **Lemma 13.** *For a value b with $0 < b < 1$, define $p_k = \binom{n}{k} b^i (1-b)^{n-k}$, for $0 \leq k \leq n$. Choosing $x_1, \ldots, x_n$ iid and uniformly in $(0,1)$ is equivalent to the following: with probability p_k, uniformly sample a set I of k distinct indices in $\{1, \ldots, n\}$ among all the subsets of size k. For each $i \in I$, pick x_i iid and uniformly in $(0, b)$. For each $i \notin I$, pick x_i iid and uniformly in $(b, 1)$.*

Proof. It is easy to see that choosing $x_1, \ldots, x_n$ iid uniformly in $(0,1)$ is equivalent to choosing a point X uniformly at random inside an n dimensional unit cube $(0,1)^n$. Therefore, we will prove the equivalence between (i) the distribution defined in the lemma, and (ii) choosing such point X.

Let Q be the n-dimensional unit cube. Subdivide Q into 2^n rectangular region defined by the Cartesian product of intervals $(0, b)$ and $(b, 1)$, i.e., $\{(0, b), (b, 1)\}^n$ (or alternatively, bisect Q with n hyperplanes, with the i-th hyperplane perpendicular to the i-th axis and intersecting it at coordinate equal to b).

Consider the set R_k of rectangles in $\{(0, b), (b, 1)\}^n$ with exactly k sides of length b and $n - k$ sides of length $1 - b$. Observe that for every choice of k (distinct) indices $i_1, \ldots, i_k$ out of $\{1, \ldots, n\}$, there exists exactly one rectangle r in R_k such that r has side length b at dimensions $i_1, \ldots, i_k$, and all the other sides of r has length $n - k$. As a result, we know that the number of rectangles in R_k is $\binom{n}{k}$ and the volume of each rectangle in R_k is $b^k(1-b)^k$. Thus, if we choose a point X randomly inside Q, with probability p_k it will fall inside a

rectangle r in R_k; furthermore, conditioned on this event, the dimensions $i_1, \ldots, i_k$ where r has side length b is a uniform subset of k distinct indices from $\{1, \ldots, n\}$. ◀

Remember that our goal was to prove that with constant probability, the minimum will be compared against some value in the range $[G^{-i}, G^{-i+1}]$ for every integer i, $1 \leq i \leq \frac{\log_G n}{2}$. We can pick $b = G^{-i+1}$ and apply Lemma 13. We then observe that it is very likely that the set of indices I that we are sampling in Lemma 13 will contain many indices. For every element x_i, $i \in I$, we are sampling x_i independently and uniformly in $(0, b)$ which opens the door for us to apply Lemma 12. Then we argue that Lemma 12 would imply that with constant probability the minimum will be compared against a value in the range $(b/G, b) = (G^{-i}, G^{-i+1})$. The lower bound claim of Theorem 10 then follows by invoking the linearity of expectations.

We are ready to prove that the minimum element will have $\Omega(\log_\Delta n)$ comparisons on average.

Proof of Theorem 10. First, observe that we can assume $n \geq (100,000\Delta)^2$ as otherwise we are aiming for a trivial bound of $\Omega(1)$. ##(N:) Δ can be as large as n. How is this inequality possible? Or should we bound $\Delta \leq \sqrt{n}$ ##

We create an input set of n values $x_1, \ldots, x_n$ where each x_i is chosen iid and uniformly in $(0, 1)$. Let $G = 100\Delta$. Consider an integer i such that $1 \leq i < \frac{\log_G n}{2}$. ##(N:) in the beginning of this section $i \leq \frac{\log_G n}{2}$. ## We are going to prove that with constant probability, the minimum will be compared against a value in the range (G^{-i}, G^{-i+1}), which, by linearity of expectation, shows the stated $\Omega(\log_\Delta n)$ lower bound for the fragile complexity of the minimum.

Consider a fixed value of i. Let S be the set of indices with values that are smaller than G^{-i+1}. Let p be the probability that $\mathcal{A}$ compares the minimum against an x_j with $j \in S$ such that $x_j \geq G^{-i}$. To prove the theorem, it suffices to prove that p is lower bounded by a constant. Now consider an algorithm $\mathcal{A}'$ that finds the minimum but for whom all the values other than those in S have been revealed and furthermore, assume $\mathcal{A}'$ minimizes the probability of comparing the minimum against an element $x \geq G^{-i}$ (in other words, we pick the algorithm which minimizes this probability, among all the algorithms). Clearly, $p' \leq p$. In the rest of the proof we will give a lower bound for p'.

Observe that $|S|$ is a random variable with binomial distribution. Hence $\mathbb{E}[|S|] = nG^{-i+1} > \sqrt{n}$ where the latter follows from $i < \frac{\log_G n}{2}$. By the properties of the binomial distribution we have that $\mathbf{Pr}\left[|S| < \frac{\mathbb{E}[|S|]}{100}\right] < \frac{1}{10}$. Thus, with probability at least $\frac{9}{10}$, we will have the "good" event that $|S| \geq \frac{\mathbb{E}[|S|]}{100} \geq \frac{\sqrt{n}}{100}$.

In case of the good event, Lemma 13 implying that conditioned on S being the set of values smaller than G^{-i+1}, each value x_j with $j \in S$ is distributed independently and uniformly in the range $(0, G^{-i+1})$. As a result, we can now invoke Lemma 12 on the set S with $T = |S|$. ##(R:) strictly speaking, we are missing a scaling lemma ## Since $n \geq (100,000\Delta)^2$ we have $T = |S| \geq \frac{\sqrt{n}}{100} \geq \frac{100,000\Delta}{100}$. By Lemma 12, with probability at least $\frac{7}{10}$, the minimum will be compared against a value that is larger than G^{-i}.

Thus, by law of total probability, it follows that in case of a good event, with probability $\frac{7}{10}$ the minimum will be compared to a value in the range (G^{-i}, G^{-i+1}). However, as the good event happens with probability $\frac{9}{10}$, it follows that with probability at least $1-(1-\frac{7}{10})-(1-\frac{9}{10}) = \frac{6}{10}$, the minimum will be compared against a value in the range (G^{-i}, G^{-i+1}). ◀

2.3.2 Lower bound for the fragile complexity of the minimum whp.

With Theorem 8 in Subsection 2.2, we show in particular that SampleMinimum guarantees that the fragile complexity of the minimum is at most $\mathcal{O}(\log\log n)$ with probability at least $1 - 1/n^c$ for any $c > 1$. (By setting $t = 2\log\log n$).

Here we show that this is optimal up to constant factors in the fragile complexity.

▶ **Theorem 14.** *For any constant $\varepsilon > 0$, there exists a value of n_0 such that the following holds for any randomized algorithm $\mathcal{A}$ and for any $n > n_0$: there exists an input of size n such that with probability at least $n^{-\varepsilon}$, $\mathcal{A}$ performs $\geq \frac{1}{2}\log\log n$ comparisons with the minimum.*

Proof. We use (again) Yao's principle and consider a fixed deterministic algorithm $\mathcal{A}$ working on the uniform input distribution, i.e., all input permutations have probability $1/n!$. Let $f = \frac{1}{2}\log\log n$ be the upper bound on the fragile complexity of the minimum. Let $k = 2^f = \sqrt{\log n}$ and let S be the set of the k smallest input values. Let π be a uniform permutation (the input) and $\pi(S)$ be the permutation of the elements of S in π. Observe that $\pi(S)$ is a uniform permutation of the elements of S. We reveal the elements not in S to $\mathcal{A}$. So, $\mathcal{A}$ only needs to find the minimum in $\pi(S)$. By Theorem 2 there is at least one "bad" permutation of S which forces algorithm $\mathcal{A}$ to do $\log k = f$ comparisons on the smallest element. Observe $\log k! < \log k^k = k\log k = \sqrt{\log n}\frac{1}{2}\log\log n$. Observe that there exists a value of n_0 such that for $n > n_0$ the right hand side is upper bounded by $\varepsilon\log n$, so $k! \leq n^\varepsilon$, for $n > n_0$. Hence, the probability of a "bad" permutation is at least $1/k! > n^{-\varepsilon}$. ◀

3 Selection and median

The (n,t)-*selection problem* asks to find the t-th smallest element among n elements of the input. The simplest solution to the (n,t)-selection problem is to sort the input. Therefore, it can be solved in $\mathcal{O}(\log n)$ fragile complexity and $\mathcal{O}(n\log n)$ work by using the AKS sorting network [1]. For comparator networks, both of these bounds are optimal: the former is shown by Theorem 2 (and in fact it applies also to any algorithm) and the latter is shown in the full version of this paper [**?**].

In contrast, in this section we show that comparison-based algorithms can do better: we can solve Selection deterministically in $\Theta(n)$ work and $\Theta(\log n)$ fragile complexity, thus, showing a separation between the two models. However, to do that, we resort to constructions that are based on expander graphs. Avoiding usage of the expander graphs or finding a simpler optimal deterministic solution is an interesting open problem (see Section 6). Moreover, in Subsection 3.2 we show that we can do even better by using randomization.

3.1 Deterministic selection

▶ **Theorem 15.** *There is a deterministic algorithm for* Selection *which performs $\mathcal{O}(n)$ work and has $\mathcal{O}(\log n)$ fragile complexity.*

Proof sketch. It suffices to just find the median since by simple padding we can generalize the solution for the (n,t)-selection problem.

We use *ε-halvers* that are the central building blocks of the AKS sorting network. An ε-halver approximately performs a partitioning of an array of size n into the smallest half and the largest half of the elements. More precisely, for any $m \leq n/2$, at most εn of the m smallest elements will end up in the right half of the array, and at most εn of the m largest elements will end up in the left half of the array. Using expander graphs, a comparator network

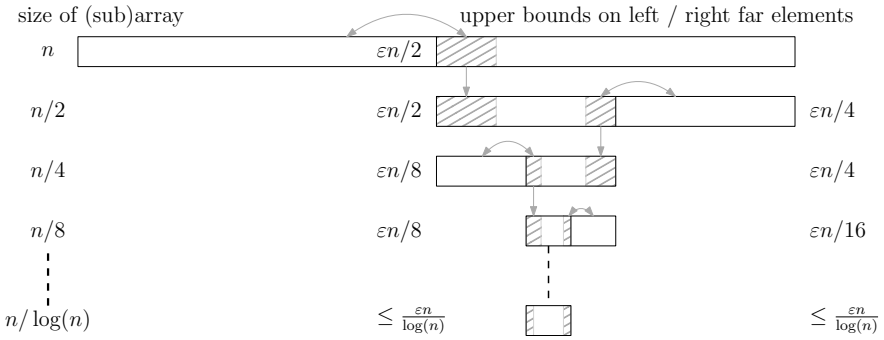

Figure 1 Illustration of the alternating division process using ε-halvers.

implementing an ε-halver of constant depth can be built [1, 3]. We use the corresponding comparison-based algorithm of constant fragile complexity.

The idea is to use ε-halvers to find roughly $\frac{n}{\log n}$ elements with rank between $(1+\alpha)\frac{n}{2}$ and $(2-\alpha)\frac{n}{2}$ and also roughly $\frac{n}{\log n}$ elements with rank between $\alpha\frac{n}{2}$ and $(1-\alpha)\frac{n}{2}$, for some constant $0 < \alpha < 1$. This is done by repeatedly using ε-halvers but alternating between selecting the left half and the right half (Figure 1). Using these, we filter the remaining elements and discard a constant fraction of them. Then we recurse on the remaining elements. The details are a bit involved, as we have to guarantee that no element accumulates too many comparisons throughout the recursions. We have to do some bookkeeping as well as some additional ideas to provide this guarantee. Details can be found in the full version of the paper [**?**]. ◀

▶ **Corollary 16.** *There is a deterministic algorithm for partition which performs $\mathcal{O}(n)$ work and has $\mathcal{O}(\log n)$ fragile complexity.*

Proof. At the end of the SELECTION algorithm, the set of elements smaller (larger) than the median is the union of the respective filtered sets (sets $\mathcal{L}$ and $\mathcal{R}$ in the proof in the full version of the paper [**?**]) and the first (last) half of the sorted set in the base case of the recursion. Again, simple padding generalizes this to (n,t)-partition for arbitrary $t \neq \frac{n}{2}$. ◀

3.2 Randomized selection

In the full paper [**?**], we present the details of an expected work-optimal selection algorithm with a trade-off between the expected fragile complexity $f_{\text{med}}(n)$ of the selected element and the maximum expected fragile complexity $f_{\text{rem}}(n)$ of the remaining elements. In particular, we obtain the following combinations:

▶ **Theorem 17.** *Randomized selection is possible in expected linear work, while achieving expected fragile complexity of the median $\mathbb{E}[f_{med}(n)] = \mathcal{O}(\log\log n)$ and of the remaining elements $\mathbb{E}[f_{rem}(n)] = \mathcal{O}(\sqrt{n})$, or $\mathbb{E}[f_{med}(n)] = \mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ and $\mathbb{E}[f_{rem}(n)] = \mathcal{O}(\log^2 n)$.*

4 Sorting

Recall from Section 1 that the few existing sorting networks with depth $\mathcal{O}(\log n)$ are all based on expanders, while a number of $\mathcal{O}(\log^2 n)$ depth networks have been developed based on binary merging. Here, we study the power of the mergesort paradigm with respect to fragile complexity. We first prove that any sorting algorithm based on binary merging must have

a worst-case fragile complexity of $\Omega(\log^2 n)$. This provides an explanation why all existing sorting networks based on merging have a depth no better than this. We also prove that the standard mergesort algorithm on random input has fragile complexity $\mathcal{O}(\log n)$ with high probability, thereby showing a separation between the deterministic and the randomized situation for binary mergesorts. Finally, we demonstrate that the standard mergesort algorithm has a worst-case fragile complexity of $\Theta(n)$, but that this can be improved to $\mathcal{O}(\log^2 n)$ by changing the merging algorithm to use exponential search. The omitted proofs can be found in the full paper [?].

▶ **Lemma 18.** *Merging of two sorted sequences A and B has fragile complexity at least $\lfloor \log_2 |A| \rfloor + 1$.*

▶ **Theorem 19.** *Any binary mergesort has fragile complexity $\Omega(\log^2 n)$.*

Proof. The adversary is the same as in the proof of Lemma 21, except that as scapegoat element for a merge of A and B it always chooses the scapegoat from the *larger* of A and B. We claim that for this adversary, there is a constant $c > 0$ such that for any node v in the mergetree, its scapegoat element has participated in at least $c \log^2 n$ comparisons in the subtree of v, where n is the number of elements merged by v. This implies the theorem.

We prove the claim by induction on n. The base case is $n = \mathcal{O}(1)$, where the claim is true for small enough c, as the scapegoat by Lemma 18 will have participated in at least one comparison. For the induction step, assume v merges two sequences of sizes n_1 and n_2, with $n_1 \geq n_2$. By the base case, we can assume $n_1 \geq 3$. Using Lemma 18, we would like to prove for the induction step

$$c \log^2 n_1 + \lfloor \log n_2 \rfloor + 1 \geq c \log^2 (n_1 + n_2). \tag{1}$$

This will follow if we can prove

$$\log^2 n_1 + \frac{\log n_2}{c} \geq \log^2 (n_1 + n_2)\,. \tag{2}$$

The function $f(x) = \log^2 x$ has first derivative $2(\log x)/x$ and second derivative $2(1-\log x)/x^2$, which is negative for $x > e = 2.71\ldots$. Hence, $f(x)$ is concave for $x > e$, which means that first order Taylor expansion (alias the tangent) lies above f, i.e., $f(x_0)+f'(x_0)(x-x_0) \geq f(x)$ for $x_0, x > e$. Using $x_0 = n_1$ and $x = n_1+n_2$ and substituting the first order Taylor expansion into the right side of (2), we see that (2) will follow if we can prove

$$\frac{\log n_2}{c} \geq 2 \frac{\log n_1}{n_1} n_2\,,$$

which is equivalent to

$$\frac{\log n_2}{n_2} \geq 2c \frac{\log n_1}{n_1}\,. \tag{3}$$

Since $n_1 \geq n_2$ and $(\log x)/x$ is decreasing for $x \geq e$, we see that (3) is true for $n_2 \geq 3$ and c small enough. Since $\log(3)/3 = 0.366\ldots$ and $\log 2/2 = 0.346\ldots$, it is also true for $n_2 = 2$ and c small enough. For the final case of $n_2 = 1$, the original inequality (1) reduces to

$$\log^2 n_1 + \frac{1}{c} \geq \log^2 (n_1 + 1)\,. \tag{4}$$

Here we can again use concavity and first order Taylor approximation with $x_0 = n_1$ and $x = n_1 + 1$ to argue that (4) follows from

$$\frac{1}{c} \geq 2\frac{\log n_1}{n_1}\,.$$

which is true for c small enough, as $n_1 \geq 3$ and $(\log x)/x$ is decreasing for $x \geq e$. ◀

▶ **Theorem 20.** *Standard* MergeSort *with linear merging has a worst-case fragile complexity of* $\Theta(n)$.

▶ **Lemma 21.** *Standard* MergeSort *has fragile complexity* $\Omega(\log^2 n)$.

Proof. In MergeSort, when merging two sorted sequences A and B, no comparisons between elements of A and B have taken place before the merge. Also, the sorted order of $A \cup B$ has to be decided by the algorithm after the merge. We can therefore run the adversary argument from the proof of Lemma 18 in all nodes of the mergetree of MergeSort. If the adversary reuses scapegoat elements in a bottom-up fashion—that is, as scapegoat for a merge of A and B chooses one of the two scapegoats from the two merges producing A and B—then the scapegoat at the root of the mergetree has participated in

$$\Omega(\sum_{i=0}^{\log n} \log 2^i) = \Omega(\sum_{i=0}^{\log n} i) = \Omega(\log^2 n)$$

comparisons, by Lemma 18 and the fact that a node at height i in the mergetree of standard MergeSort operates on sequences of length $\Theta(2^i)$. ◀

▶ **Observation 1.** *Consider two sorted sequences* $A = (a_1, \ldots, a_n)$ *and* $B = (b_1, \ldots, b_n)$. *In linear merging, the fragile complexity of element* a_i *is at most* $\ell + 1$ *where* ℓ *is the largest number of elements from* B *that are placed directly in front of* a_i *(i.e.* $b_j < \ldots < b_{j+\ell-1} < a_i$*).*

▶ **Lemma 22.** *Let* $X = \{x_1, \ldots, x_{2k}\}$ *be a finite set of distinct elements, and consider a random bipartition* $X_L, X_R \subset X$ *with* $|X_L| = |X_R| = k$ *and* $X_L \cap X_R = \varnothing$*, such that* $\mathbf{Pr}[x_i \in X_L] = 1/2$. *Consider an arbitrary ordered set* $Y = \{y_1, \ldots, y_m\} \subset X$ *with* $m \leq k$. *Then* $\mathbf{Pr}[Y \subseteq X_L \vee Y \subseteq X_R] < 2^{1-m}$.

Proof.

$$\mathbf{Pr}[Y \subseteq X_L \vee Y \subseteq X_R] = 2\prod_{i=1}^{m} \mathbf{Pr}[\, y_i \in X_L \,|\, y_1, \ldots y_{i-1} \in X_L \,] = 2\frac{(2k)^{-m}k!}{(k-m)!} \leq 2 \cdot 2^{-m}.$$ ◀

▶ **Theorem 23.** *Standard* MergeSort *with linear merging on a randomized input permutation has a fragile complexity of* $\mathcal{O}(\log n)$ *with high probability.*

Proof. Let $Y = (y_1, \ldots, y_n)$ be the input-sequence, π^{-1} be the permutation that sorts Y and $X = (x_1, \ldots, x_n)$ with $x_i = y_{\pi^{-1}(i)}$ be the sorted sequence. Wlog we assume that all elements are unique[2], that any input permutation π is equally likely[3], and that n is a power of two.

[2] If this is not the case, use input sequence $Y' = ((y_1, 1), \ldots, (y_n, n))$ and lexicographical compares.
[3] If not shuffle it before sorting in linear time and no fragile comparisons.

Merging in one layer. Consider any merging-step in the mergetree. Since both input sequences are sorted, the only information still observable from the initial permutation is the bi-partitioning of elements into the two subproblems. Given π, we can uniquely retrace the mergetree (and vice-versa): we identify each node in the recursion tree with the set of elements it considers. Then, any node with elements $X_P = \{y_\ell, \ldots, y_{\ell+2k-1}\}$ has children

$$X_L = \left\{ x_{\pi(i)} \;\middle|\; \ell \le \pi(i) \le \ell + k - 1 \right\} = \{y_\ell, \ldots, y_{\ell+k-1}\},$$
$$X_R = \left\{ x_{\pi(i)} \;\middle|\; \ell + k \le \pi(i) \le \ell + 2k - 1 \right\} = \{y_{\ell+k}, \ldots, y_{\ell+2k-1}\}.$$

Hence, locally our input permutation corresponds to an stochastic experiment in which we randomly draw exactly half of the parent's elements for the left child, while the remainder goes to right.

This is exactly the situation in Lemma 22. Let N_i be a random variable denoting the number of comparisons of element y_i in the merging step. Then, from Observation 1 and Lemma 22 it follows that $\mathbf{Pr}[N_i = m{+}1] \le 2^{-m}$. Therefore N_i is stochastically dominated by $N_i \preceq 1{+}Y_i$ where Y_i is a geometric random variable with success probability $p = 1/2$.

Merging in all layers. Let $N_{j,i}$ be the number of times element y_i is compared in the j-th recursion layer and define $Y_{j,i}$ analogously. Due to the recursive partitioning argument, $N_{j,i}$ and $Y_{j,i}$ are iid in j. Let N_i^T be the total number of comparisons of element i, i.e. $N_i^T \preceq \log_2 n + \sum_{j=1}^{\log_2 n} Y_{j,i}$. Then a tail bound on the sum of geometric variables (Theorem 2.1 in [12]) yields:

$$\mathbf{Pr}\left[\sum\nolimits_{j=1}^{\log_2 n} Y_{j,i} \ge \lambda \mathbb{E}\left[\sum\nolimits_{j=1}^{\log_2 n} Y_{j,i}\right] = 2\lambda \log_2 n\right] \overset{[12]}{\le} \exp\left(-\frac{1}{2}\frac{2\ln n}{\ln 2}[\lambda - 1 - \log\lambda]\right) = n^{-2},$$

where we set $\lambda \approx 3.69$ in the last step solving $\lambda - \log\lambda = 2\log 2$. Thus, we bound the probability $\mathbf{Pr}\left[N_i^T \ge (1{+}2\lambda)\log_2 n\right] \le n^{-2}$.

Fragile complexity. It remains to show that with high probability no element exceeds the claimed fragile complexity. We use a union bound on N_i^T for all i:

$$\mathbf{Pr}\left[\max_i\{N_i^T\} = \omega(\log n)\right] \le n\mathbf{Pr}\left[N_i^T = \omega(\log n)\right] \le 1/n. \qquad \blacktriangleleft$$

▶ **Theorem 24.** *Exponential merging of two sequences $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ has a worst-case fragile complexity of $\mathcal{O}(\log n)$.*

▶ **Corollary 25.** *Applying Theorem 24 to standard* MergeSort *with exponential merging yields a fragile complexity of $\mathcal{O}(\log^2 n)$ in the worst-case.*

5 Constructing binary heaps

▶ **Theorem 26.** *The fragile complexity of the standard binary heap construction algorithm of Floyd [10] is $\mathcal{O}(\log n)$.*

The above observation is easy to verify (shown in the full paper [**?**]). We note that this fragile complexity is optimal by Theorem 2, since Heap Construction is stronger than Minimum. Brodal and Pinotti [6] showed how to construct a binary heap using a comparator network in $\Theta(n \log\log n)$ size and $\mathcal{O}(\log n)$ depth. They also proved a matching lower bound on the size of the comparator network for this problem. This, together with Observation 26 and the fact that Floyd's algorithm has work $\mathcal{O}(n)$, gives a separation between work of fragility-optimal comparison-based algorithms and size of depth-optimal comparator networks for Heap Construction.

6 Conclusions

In this paper we introduced the notion of fragile complexity of comparison-based algorithms and we argued that the concept is well-motivated because of connections both to real world situations (e.g., sporting events), as well as other fundamental theoretical concepts (e.g., sorting networks). We studied the fragile complexity of some of the fundamental problems and revealed interesting behavior such as the large gap between the performance of deterministic and randomized algorithms for finding the minimum. We believe there are still plenty of interesting and fundamental problems left open. Below, we briefly review a few of them.

- The area of comparison-based algorithms is much larger than what we have studied. In particular, it would be interesting to study "geometric orthogonal problems" such as finding the maxima of a set of points, detecting intersections between vertical and horizontal line segments, *kd*-trees, axis-aligned point location and so on. All of these problems can be solved using algorithms that simply compare the coordinates of points.
- Is it possible to avoid using expander graphs to obtain simple deterministic algorithms to find the median or to sort?
- Is it possible to obtain a randomized algorithm that finds the median where the median suffers $O(1)$ comparisons on average? Or alternatively, is it possible to prove a lower bound? If one cannot show a $\omega(1)$ lower bound for the fragile complexity of the median, can we show it for some other similar problem?

References

1. M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Symposium on Theory of Computation*, STOC '83, pages 1–9. ACM, 1983. URL: http://doi.acm.org/10.1145/800061.808726.
2. M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, Mar 1983. URL: https://doi.org/10.1007/BF02579338.
3. M. Ajtai, J. Komlós, and E. Szemerédi. Halvers and expanders. In IEEE, editor, *FOCS'92*, pages 686–692, Pittsburgh, PN, October 1992. IEEE Computer Society Press.
4. V. E. Alekseev. Sorting algorithms with minimum memory. *Kibernetika*, 5(5):99–103, 1969.
5. K. E. Batcher. Sorting networks and their applications. *Proceedings of AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.
6. G. Stølting Brodal and M. C. Pinotti. Comparator networks for binary heap construction. In *Proc. 6th Scandinavian Workshop on Algorithm Theory*, volume 1432 of *LNCS*, pages 158–168. Springer Verlag, Berlin, 1998. doi:10.1007/BFb0054364.
7. V. Chvátal. Lecture notes on the new AKS sorting network. Technical Report DCS-TR-294, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1992, October.
8. R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.
9. M. Dowd, Y. Perl, L. Rudolph, and M. Saks. The periodic balanced sorting network. *J. ACM*, 36(4):738–757, 1989, October.
10. R. W. Floyd. Algorithm 245: Treesort. *Commun. ACM*, 7(12):701, December 1964. URL: http://doi.acm.org/10.1145/355588.365103.
11. M. T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In David B. Shmoys, editor, *STOC'14*, pages 684–693. ACM, 2014. URL: http://dl.acm.org/citation.cfm?id=2591796.
12. S. Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018. doi:https://doi.org/10.1016/j.spl.2017.11.017.
13. S. Jimbo and A. Maruoka. A method of constructing selection networks with $O(\log n)$ depth. *SIAM Journal on Computing*, 25(4):709–739, 1996.

14 I. Parberry. The pairwise sorting network. *Parallel Processing Letters*, 2(2-3):205–211, 1992.

15 B. Parker and I. Parberry. Constructing sorting networks from k-sorters. *Information Processing Letters*, 33(3):157–162, 30 November 1989.

16 M. S. Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5(1):75–92, 1990.

17 N. Pippenger. Selection networks. *SIAM Journal on Computing*, 20(5):878–887, 1991.

18 V. R. Pratt. *Shellsort and Sorting Networks*. Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1972.

19 J. I. Seiferas. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53(3):374–384, 2009.

20 S.Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *BAMS: Bulletin of the American Mathematical Society*, 43:439–561, 2006.

21 S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

22 A. Yao and F. F. Yao. Lower bounds on merging networks. *J. ACM*, 23(3):566–571, 1976.