Fragile complexity of some classic comparison based problems

June 14, 2018

Sort n elements such that each element participates in no more than $\mathcal{O}(\log n)$ comparisons.

1 Notation

Given so many authors, to make sure there is some uniformity in the exposition, this file contains various definitions, to make writeup consistent among all authors. Before you define a new term, please check here to see if it is already defined, define a macro in pre.tex, and add description here, as it's likely others might need to use it too.

##(P:) Please check the source code to use the right macros, so if we want to change any of the terms in the future, we can change them only in one place. ##

Variables/terminology used (when defining new terms, make sure to define both upper- and lowercase versions (and don't forget `xspace` command at the end to avoid frivioulous spaces in front of punctuation):

- $\mathcal{N}$ is comparator network
- $\mathcal{A}$ is an algorithm
- $\mathcal{I}$ is a set of inputs
- $\mathcal{T}$ is a tree
- $\mathbb{N}$ is a set of natural numbers
- $\mathbb{Z}$ is a set of integers
- $\mathcal{P}$ is a problem
- G for a graph
- V for vertices
- E for edges
- 'Fragile complexity' or 'fragile complexity' for the maximum number of comparisons per element, also used for a particular element like "fragile complexity of the minimum"
- 'Work' or 'work' for the total number of comparisons performed throughout the algorithm

- $f(n)$ for fragile complexity of an algorithm
- $w(n)$ for work of an algorithm
- $rank(x)$ rank of x (might need to indicate the set with respect to which it is defined)
- $\Pr[x < y]$ or Pr for probability
- $\mathbb{E}[X]$ for expectation value
- $\mathcal{O}$ for Big-O. If to late, we set $\mathcal{O}$as O

2 Introduction

Comparison-based algorithms is a classic and fundamental research area in computer science. Problems studied include minimum, median, sorting, searching, dictionaries, priority queues, and many more, 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 also is 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 clearly has fragile complexity $\Omega(n)$ from its partitioning phase, may use up the chosen pivot elements 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. As further examples, the detrimental impact on objects may be of less directly physical nature, for instance if each comparison of an element involves a privacy risk for it, or if bias in the comparison process grows each time the same element is used for comparison.

In this paper, we initiate the study of comparison-based complexity from the viewpoint of the individual elements. We define the *fragile complexity* of comparison-based algorithms as the maximal number of comparisons any individual element takes part in, and we give a number of upper and lower bounds on the fragile complexity complexity for fundamental problems.

2.1 Previous work

One body of work relevant to the question which we raise here is the study of sorting networks, propelled by the 1968 paper of Batcher [5]. In sorting networks, and more generally comparator networks (see Section 3 for a definition), 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 $g(n)$ easily can be converted into a comparison-based algorithm with fragile complexity $f(n)$ and work $g(n)$.

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 [10, 16, 17, 20], and for a long time it was an open question whether better results were possible. In a seminal result, Ajtai, Komlós, and Szemerédi in 1983 [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 [13, 22], 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 [8, 18, 21], but the overall ideas remained, and finding a simple, optimal sorting network, in particular one not based on expander graphs, remains an open problem. A recent suggestion by Goodrich [12] has much better constant factors, but is still based on expander graphs and has only optimal size, not optimal depth. Comparator networks for other problems, such as selection and heap construction have also been studied [4, 6, 15, 19, 23]. In all these problems the size of the network is super-linear.

Since comparator networks of depth $f(n)$ and size $g(n)$ can be transformed into comparison-based algorithms with $f(n)$ fragile complexity and $g(n)$ work, one 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 $g(n)$ for the same $f(n)$.

More generally, one could 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, consider comparison-based algorithms under the most restrictive setting in parallel computation: the exclusive read and exclusive write (EREW) PRAM model. In the EREW PRAM model, it is possible to create n copies of an element e in $\mathcal{O}(\log n)$ time and thus we can 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. As a case in point, it is not clear whether Richard Cole's celebrated parallel merge sort algorithm [9] yields a comparison-based algorithm with low fragile complexity since it performs copying of elements.

2.2 Our contribution

Table 1 summarizes the results presented in this paper. In particular, we show a separation between comparator networks and comparison-based algorithms for the problem Median, 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 $\Omega(n \log n)$ size for comparator networks. Also for Heap Construction, the above results imply a separation in work, since an $\Omega(n \log\log n)$ lower bound on the size of comparator networks for this problem has been proven by Brodal and Pinotti [6].

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 above lower $\Omega(\log n)$ bound on fragile complexity for Minimum and the standard $\Omega(n \log n)$ lower bound on work for sorting. However, it is an open problem whether these bounds can be achieved by simpler sorting algorithms than for sorting networks, in particular whether expander graphs are necessary. One

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

Problem	*Upper bound*		*Lower bound*
	$f(n)$	$w(n)$	$f(n)$
MINIMUM deterministic (Sec. 4)	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\Omega(\log n)$
MINIMUM randomized (Sec. 4)	$\langle\mathcal{O}(\varepsilon^{-1}\log\log n)^{\dagger}, \mathcal{O}(n^{\varepsilon})^{\ddagger}\rangle$, $\langle\mathcal{O}(\frac{\log n}{\log\log n})^{\dagger}, \mathcal{O}(\frac{\log n}{\log\log n})^{\ddagger}\rangle$	$\mathcal{O}(n)$	
SELECTION deterministic (Sec. 5)	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\Omega(\log n)$
MEDIAN randomized (Sec. 5)	$\langle\mathcal{O}(\log\log n)^{\dagger}, \mathcal{O}(\sqrt{n})^{\ddagger}\rangle$, $\langle\mathcal{O}(\frac{\log n}{\log\log n})^{\dagger}, \mathcal{O}(\log n)^{\ddagger}\rangle$	$\mathcal{O}(n)$	
MERGE deterministic (Sec. 8)	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\Omega(\log n)$
MERGESORT deterministic (Sec. 8)	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n\log n)$	$\Omega(\log^2 n)$
MERGESORT randomized (Sec. 8)	$\mathcal{O}(\log n)$ whp	$\mathcal{O}(n\log n)$	
HEAPCONSTRUCTION det. (Sec. 9)	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\Omega(\log n)$

Table 1: Summary of results shown in this paper. Here, $f(n)$ refers to the fragile complexity and $w(n)$ to the work (see Section 3). †: Expected for median element, ‡: Expected for other elements.

intriguing conjecture could be that any comparison-based 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 on the fragile complexity of MERGESORT 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. Our results for MERGESORT also show a separation between deterministic and randomized fragile complexity for this type of algorithms.

Some of our randomized upper bounds and deterministic lower bounds combined show that for fragile complexity there are problems where the use of randomization improves the complexity below the best deterministic bound. Others of our randomized bounds show that for MINIMUM and MEDIAN it is possible to reduce the fragile complexity of the requested element further, at the expense of the fragile complexity of the rest of the elements.

3 Definitions

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

Comparator networks. A comparator network $\mathcal{N}$ is constructed of *comparators* each consisting of two inputs and two (ordered) outputs. The value of the first output is the minimum of the two inputs and the value of the second output is the maximum of the two inputs. By this definition a comparator network $\mathcal{N}_n$ on n inputs also consists of n outputs. Figure 1 demonstrates a common visualization of comparator networks with values as horizontal wires and comparators represented by vertical arrows between pairs of wires. Each arrow points to the output that returns the minimum

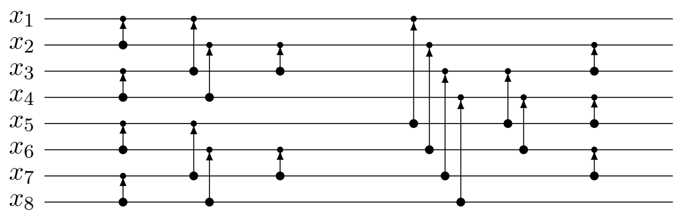

Figure 1: Batcher's Odd-Even-Mergesort network [5] with 8 inputs, depth $f(8)$=6 and size $w(8)$=19.

input value. Inputs are on the left and outputs on the right. The *size* of the comparator network is defined as the number of comparators in it, while its *depth* is defined as the number of comparators on the longest path from an input to an output.

We note that a comparator network is straightforward to execute as a comparison-based algorithm by simulating its comparators sequentially from left to right (see Figure 1), breaking ties arbitrarily. If the network has depth $f(n)$ and size $g(n)$, the comparison-based algorithm clearly has fragile complexity $f(n)$ and work $g(n)$.

Networks for problems. We define the set of inputs to the comparator network $\mathcal{N}$ by $\mathcal{I}$ and the outputs by $\mathcal{N}(\mathcal{I})$. We use the notation $\mathcal{N}(\mathcal{I})^i$ for $0 \le i \le n-1$, to represent the i-th output and $\mathcal{N}(\mathcal{I})^{i:j}$ for $0 \le i \le j \le n-1$ to represent the ordered subset of the i-th through j-th outputs.

An (n)-*sorting network* is a comparator network $\mathcal{N}_n$ such that $\mathcal{N}_n(\mathcal{I})^t$ carries the t-th smallest input value for all t. We say such a network solves the (n)-*sorting problem.*

An (n,t)-*selection network* is a comparator network $\mathcal{N}_{n,t}$ such that $\mathcal{N}_{n,t}(\mathcal{I})^0$ carries the t-th smallest input value. We say such a network solves the (n,t)-*selection problem.*

An (n,t)-*partition network* is a comparator network $\mathcal{N}_{n,t}$, such that $\mathcal{N}_{n,t}(\mathcal{I})^{0:t-1}$ carry the t smallest input values.[2] We say such a network solves the (n,t)-*partition problem.*

Clearly, an (n,t)-selection problem is asymptotically no harder than (n,t)-partition problem: let $\mathcal{N}^{\downarrow}_{n,t}(\mathcal{I})$ denote an (n,t)-partition network with all comparators reversed; then $\mathcal{N}'_{n,t}(\mathcal{I}) = \mathcal{N}^{\downarrow}_{t,t-1}(\mathcal{N}_{n,t}(\mathcal{I}))$ is an (n,t)-selection network. However, the converse is not clear: given a value of the t-th smallest element as one of the inputs, it is not obvious how to construct an (n,t)-partition network with smaller size or depth. In Section 5 we will show that the two problems are equivalent: every (n,t)-selection network also solves the (n,t)-partition problem.

Rank. Given a set X, the rank of some element e in X, denoted by $rank_X(e)$, is equal to the size of the subset of X containing the elements that are no larger than e. When the set X is clear from the context, we will omit the subscript X and simply write $rank(e)$.

4 Finding the minimum

4.1 Deterministic minimum finding

Theorem 1. *The fragile complexity of finding the minimum of n elements is at most $\lceil \log n \rceil$.*

[2]Brodal and Pinotti [6] call it an (n,t)-selection network, but we feel (n,t)-partition network is a more appropriate name.

Proof. Use a perfectly balanced binary tree as tournament tree. □

Theorem 2. *For any deterministic algorithm $\mathcal{A}$ that finds the minimum of n elements, the fragile complexity of the minimum element is at least $\lceil \log n \rceil$.*

Proof. Let S be a set of n elements, and $\mathcal{A}$ be a deterministic comparison-based algorithm that finds the minimum element of S. We describe an adversarial strategy for resolving the comparisons made by $\mathcal{A}$ that leads to the lower bound.

Consider a directed graph G on n nodes corresponding to the elements of S. With a slight abuse of notation, we use the same names for elements of S and the associated nodes in graph G. The edges of G correspond to comparisons made by $\mathcal{A}$, and are either black or red. Initially G has no edges. If $\mathcal{A}$ compares two elements, we insert a directed edge between the associated nodes pointing toward the element declared smaller by the adversarial strategy; the color is selected later on. Algorithm $\mathcal{A}$ correctly finds the minimum element if and only if, upon termination of $\mathcal{A}$, the resulting graph G has only one sink node.

Consider the following adversarial strategy to resolve comparisons made by $\mathcal{A}$: if both elements are sinks in G, the element that has already participated in more comparisons is declared smaller; if only one element is a sink in G, this element is declared smaller; and if neither element is a sink in G, the comparison is resolved arbitrarily (while conforming to the existing partial order).

We color an edge in G red if it corresponds to a comparison between two sinks; otherwise, we set the color of the edge to be black. For each element i, consider its in-degree d_i and the number of nodes r_i in G (including i itself) from which i is reachable by a directed path of only red edges.

We show by induction that $r_i \leq 2^{d_i}$ for all sinks in G. Initially, $r_i = 1 \leq 1 = 2^{d_i}$ for all i. Let algorithm $\mathcal{A}$ compare two elements i and j, where i is a sink, and let the adversarial strategy declare i to be smaller than j. Then, the resulting in-degree of i is $d_i + 1$. If the new edge is black, the number of nodes from which i is reachable via red edges does not change, and the inequality holds trivially. If the new edge is red, the resulting number of nodes from which i is reachable is $r_i + r_j \leq 2^{d_i} + 2^{d_j} \leq 2^{d_i+1}$. Therefore, when $\mathcal{A}$ terminates and only one sink is left in G, its in-degree is at least $\lceil \log n \rceil$, which corresponds to the number of comparisons the minimum element has participated in. □

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

4.2 Randomized minimum finding

In the following we present RMinimum, a randomized recursive algorithm for finding the minimum among n distinct elements. The algorithm has a tuning parameter $k(n)$ controlling the trade-off between the expected fragile complexity $f_{\min}(n)$ of the minimum element and the maximum expected fragile complexity $f_{\text{rem}}(n)$ of the remaining none-minimum elements; if n is clear from the context, we use k instead of $k(n)$. Depending on the choice of k, we obtain interesting trade-offs for the pair $\langle \mathbb{E}[f_{\min}(n)], \max \mathbb{E}[f_{\text{rem}}(n)] \rangle$ ranging from $\left\langle \mathcal{O}(\varepsilon^{-1}\log\log(n)), \mathcal{O}(n^{\varepsilon}) \right\rangle$ to $\langle \mathcal{O}(\log(n)/\log\log(n)), \mathcal{O}(\log(n)/\log\log(n)) \rangle$. Given a total ordered set X of n distinct elements, RMinimum performs the following steps:

1. Randomly group the elements into $n/2$ pairwise-disjoint pairs and use one comparison for each pair to partition X into two sets L and W of equal size: W contains all *winners*, i.e.

elements that are smaller than their partner. Set L gets the *losers*, who are larger than their partner and hence cannot be the global minimum.

2. Partition L into n/k random disjoint subsets $L_1, \dots, L_{n/k}$ each of size k and find the minimum element m_i in each L_i using a perfectly balanced tournament tree (see Theorem 1).

3. Partition W into n/k random disjoint subsets $W_1, \dots, W_{n/k}$ each of size k and filter out all elements in W_i larger than m_i to obtain $W' = \bigcup_i \{w | w \in W_i \wedge w < m_i\}$.

4. If $|W'| \leq \log^2(n_0)$ where n_0 is the initial problem size, find and return the minimum using a perfectly balanced tournament tree (see Theorem 1). Otherwise recurse on W'.

In the following $rank_X(x)$ denotes the rank of element x in set X and let $\lambda = n/\sqrt{k}$.

Lemma 1. *Let $k \leq \sqrt{n}$ and m_i be an arbitrary minimum as in step 2. Then* $\Pr[rank_X(m_i) > c\lambda] \leq \exp(-c(c+1))$.

Proof. Any m_i is larger than its random partner in step 1 and is therefore assigned to L. Then, it wins against $k-1$ other elements in L. For all losers $\ell \in L$ we have

$$\Pr[rank_X(\ell) \leq z] \geq (z-1-k)(z-k)/n^2 > (z^2 - 2zk)/n^2$$

because its winning partner required rank at most $z-1$ and we reduce the number of available small items by k to lower bound the probability (positions are taken by the winners). Define $p(z) = (z^2 - 2zk)/n^2$. Hence, we can bound the rank of m_i in terms of λ and some constant $c \geq 1$:

$$\begin{aligned}\Pr[rank_X(m_i) \geq c\lambda] &\leq (1-p(c\lambda))^k \leq \exp(-kp(c\lambda)) \leq \exp\left(-k\frac{c^2\lambda^2 - 2cn\sqrt{k}}{n^2}\right) \\ &= \exp\left(-c\underbrace{(c + 2k\sqrt{k}/n)}_{\leq c+1 \text{ as } k \leq \sqrt{n}}\right).\end{aligned}$$

□

Lemma 2. *Let W_i be an arbitrary winner partition, m_i be the minimum it uses to filter, and $W'_i = \{w \mid w \in W_i \wedge w_i < m_i\}$ be the set of its elements smaller than m_i. Then $\mathbb{E}[|W'_i|] \leq 2d\sqrt{k}$ for some constant $d > 0$.*

Proof. Let X_j be a Bernoulli random variable indicating that element $w_j \in W_i$ is smaller than m_i. Then the expected size of W'_i conditioned on the rank of m_i is $f(b) := \mathbb{E}[|W'_i| \mid rank_X(m_i) = b] = \sum_{j=1}^{k} \Pr[X_j = 1 \mid rank_X(m_i) = b] \leq 2k\frac{b}{n}$. In the last step we account for the fact that each elements $w_j \in W_i$ won against a random partner which at most doubles $\Pr[X_j = 1]$ compared to a uniformly sampled w_j. Applying the Law of Total Expectation yields

$$\begin{aligned}\mathbb{E}[|W'_i|] &= \sum_b \mathbb{E}\left[|W'_i| \,\middle|\, rank_X(m_i) = b\right] \Pr[rank_X(m_i) = b] \\ &\leq f(\lambda)\underbrace{\Pr[rank_X(m_i) \leq \lambda]}_{\leq 1} + \sum_{\ell \geq 1} f(2^\ell\lambda)\underbrace{\Pr\left[rank_X(m_i) \geq 2^{\ell-1}\lambda\right]}_{\leq \exp(-2^{2\ell-2}),\text{ Lemma 1}} \leq 2d\frac{k\lambda}{n} = 2d\sqrt{k}\,.\end{aligned}$$ □

Theorem 3. RMINIMUM *requires linear work* $w(n) = \mathcal{O}(n)$.

Proof. Consider an arbitrary recursion step with input size n'. It requires $\mathcal{O}(n')$ comparisons to tell apart winners from losers (step 1), to process the tournament tree identifying the m_i (step 2) and to filter the W_i (step 3). The same is true for the base case which consists of only a single tournament tree incurring linear work. As in each step at least half of the elements are discarded (namely L), the total work $w(n)$ is at most $w(n) \le w(n/2) + \mathcal{O}(n) = \mathcal{O}(n)$. □

Lemma 3. *Consider a randomized recursive algorithm executing* $T(n) = a(n) + T(H(n))$ *operations where* $H(n)$ *is a random variable with* $\mathbb{E}[H(n)] \le m(n)$. *If* $a(n)$ *and* $m(n)$ *are non-decreasing and concave, then* $\mathbb{E}[T(n)] = \sum_{i \ge 0} a(m^{(i)}(n))$ *where* $m^{(0)}(n) := n$ *and* $m^{(i)}(n) := m(m^{(i-1)}(n))$.

Proof. Refer to Chaudhuri and Dubhashi [7], Proposition 8. □

Theorem 4. *Let* $k(n) = n^{\varepsilon}$ *for* $0 < \varepsilon \le 1/2$. *Then* RMINIMUM *requires* $\mathbb{E}[f_{min}] = \mathcal{O}(\varepsilon^{-1}\text{loglog}n)$ *comparisons for the minimum and* $\mathbb{E}[f_{rem}] = \mathcal{O}(n^{\varepsilon})$ *for the remaining elements.*

Proof. In each recursion step elements in W (including the minimum) are compared twice; once with its random partner in step 1 and then with pivot m_i in step 3. In the base case, another $\mathcal{O}(\text{loglog}n)$ comparisons are required to find the minimum using the tournament tree.

Observe that the recursion depth $T(n)$ is a random variable; we bound its expected value from above using Lemma 3. In the lemma's notation, we have $T(n) = T(H(n)) + a(n)$ where $a(n) = 1$ and $H(n) = |W'|$ is a random variable. Due to Lemma 2 and the Linearity of Expectation, we have $H(n) \le m(n) := cn/\sqrt{k} = cn^{1-\varepsilon/2}$ for some constant $c > 0$. For $0 < \varepsilon \le 1/2$, the function $m(n)$ is concave since then $\partial^2/\partial n^2\ m(n) = c(\varepsilon/2 - 1)\varepsilon n^{-1-\varepsilon/2} \le 0$. It is easy to see that $m(n)$ is additionally non-decreasing and that $a(n)$ also satisfies both properties. Applying Lemma 3 corrected for the base case for small n yields $\mathbb{E}[f_{\min}] = \mathbb{E}[T(n)] = \mathcal{O}(\varepsilon^{-1}\text{loglog}n)$.

Elements in L are compared more frequently but are not recursed on. We therefore consider only the first recursion step. During the search for pivots m_i all elements in L are compared $\mathcal{O}(\log k)$ times each (see Theorem 1). Then, each pivot m_i is compared to all k elements in W_i. Hence, $\mathbb{E}[f_{\text{rem}}] = \mathbb{E}[T(n)] + \mathcal{O}(k) = \mathcal{O}(n^{\varepsilon})$. □

Theorem 5. *Let* $k(n) = \text{log}n/\text{loglog}n$. *Then* RMINIMUM *requires* $\mathbb{E}[f_{min}] = \mathcal{O}(\text{log}n/\text{loglog}n)$ *comparisons for the minimum and* $\mathbb{E}[f_{rem}] = \mathcal{O}(\text{log}n/\text{loglog}n)$ *for the remaining elements.*

Proof. We obtain $\mathbb{E}[f_{\min}]$ by analogous arguments as in the proof of Theorem 4, but now use a different pruning bound $m(n)$ in the recursion: $m(n) = cn/\sqrt{\text{log}n/\text{loglog}n}$. The function $m(n)$ is non-decreasing and concave because $\partial^2/\partial n^2\ m(n) \le 0$ for $n \ge 1$. Applying Lemma 3 corrected for the base case for small n yields $\mathbb{E}[f_{\min}] = \mathcal{O}(\text{log}n/\text{loglog}n)$. By the same reasoning as in Theorem 4, a pivot element takes part in $\mathbb{E}[f_{\text{rem}}] = \mathbb{E}[T(n)] + \mathcal{O}(k) = \mathcal{O}(\text{log}n/\text{loglog}n)$ comparisons. □

4.3 loglog lower bound for min whp

Lemma 4. *For any deterministic algorithm and the uniform input distribution, the probability of the minimum having* $\geq \mathrm{loglog} n^c - \mathrm{logloglog} n$ *fragile complexity is* $\geq n^{-c}$ *for* $n > 2^{2^c}$.

Proof. Choose $k = c\mathrm{log}n/\mathrm{loglog}n$. Consider choosing a uniform permutation in the following way: First, uniformly choose a subset S (smallest) of size k of the inputs. Then choose a random permutation of the elements not in S, and finally a random permutation of the elements in S.

Think of S and the permutation of the elements not in S as (arbitrary but) fixed. Now, adapt the adversarial strategy of the deterministic lower bound. If one of the participants in the comparison is not in S, the outcome is given (or that element is larger). Otherwise count the number of comparisons between two elements of S and choose to the one with more such elements as the smaller. This leads to one permutation π of the elements in S with $\mathrm{log}k$ fragile complexity of the minimum.

Hence, for π the fragile complexity of the minimum is as stated.

The probability of choosing π is $1/k!$. By

$$\mathrm{log}k! < k\mathrm{log}k = c\frac{\mathrm{log}n}{\mathrm{loglog}n}\mathrm{log}\frac{c\mathrm{log}n}{\mathrm{loglog}n} < c\mathrm{log}n$$

this probability as at least as stated. □

4.4 Expected Lower Bound of the Minimum

Situation: Δ is a hard bound on the fragile complexity of everybody.

Claim: the minimum must have expected fragile complexity of $\Omega(\log_\Delta n)$.

The input is created by drawing real values uniformly in $(0, 1)$. As the result of a comparison, the larger value is revealed to the algorithm (it might be known beforehand).

Lemma 5. *Let* $x_1, \ldots, x_n$ *be drawn uniformly at random. Let* $y = x_j$ *be the rank* k *element for* $k \geq 4\mathrm{log}n$. *Then*

$$p\left(y \leq \frac{k}{2n}\right) < 1/n$$

Proof. The decisive probability can be upper bounded using a Chernoff bound [with parameters $p = k/(2n), \mu = k/2, \delta = 1$] leading to a bound $(e/2^2)^{k/2} < (1/2)^{\mathrm{log}n} = 1/n$. □

Lemma 6. *Let* $x_1, \ldots, x_n$ *be drawn uniformly at random. Let* $y = x_j$ *be the rank* k *element for* $k \geq 4\mathrm{log}n$. *Then*

$$p\left(y \geq \frac{2k}{n}\right) < 1/n$$

Proof. The decisive probability can be upper bounded using a Chernoff bound [with parameters $p = 2k/n, \mu = 2k, \delta = 1/2$] leading to a bound $(e^{-1/2}/(1/2)^{1/2})^{2k} < (2/e)^{4\mathrm{log}n} = 1/n$. □

Perhaps we need a version of the lemmas that no element is 'wrong' whp.

Lemma 7. *Let* X *be chosen uniformly at random from* $[0, b]$. *Then* $E[-\mathrm{log}X] < 2 - \mathrm{log}b$.

Proof. We split $[0, b]$ into subintervals $(2^{-i}b, 2^{-i+1}b]$ for $i \geq 1$. Now $E[-\mathrm{log}X] \leq \sum_{i\geq 1} 2^{-i}\mathrm{log}(2^i/b) = (-\mathrm{log}b) + \sum_{i\geq 1} i2^{-i} < 2 - \mathrm{log}b$ □

Perhaps the above could be replaced be a similar statement about a uniformly chosen small rank element, simplifying the next proof.

Lemma 8. *Let $b \in [(8\log n)/n, 1]$ be arbitrary. Assume the fragile complexity of any element is Δ. Consider running a deterministic algorithm A on input $\vec{x} = (x_1, \ldots, x_n)$ drawn uniformly from $(0,1)^n$. Let $f(i)$ be the index of the element for which A did the comparison $y_i = x_{f(i)} < x_i$ to show that x_i is not the minimum. Let $S = \{i \mid x_i \leq b\}$ be the set of indices of inputs that are smaller than b (and not the minimum). If S is empty, set $X = 0$. Otherwise choose uniformly $i \in S$ and define the random variable $L = \log\frac{x_i}{y_i}$.*

Then there is an event F (fail) with $p(F) < 2/n$ such that $E[L \mid_{\neg F}] \leq \log(16\Delta)$.

Proof. Define $s = |S_{\vec{x}}|$. By Lemma 5, $s \geq bn/2$ with probability $1/n$. By Lemma 6, the largest filter element of rank s/Δ has value $\geq b/(4\Delta)$ with probability $1/n$. The two events constitutes F. In the best case for the algorithm, the $bn/2$ elements of S are filtered by the smallest s/Δ elements. We weaken the lower bound by assuming that the elements with value $\leq b/(4\Delta)$ are filtering elements with equal load. Choosing $x_i \in S$ uniformly at random translates into choosing y_i uniformly at random from $[0, b/(4\Delta)]$, and by Lemma 7 we have $E[-\log y_i] < 2 - \log(b/4\Delta)$. We estimate x_i by b and get $E[L \mid_{\neg F}] = E[\log\frac{x_i}{y_i}] \leq E[\log b - \log X] < \log b + 2 - \log b + \log 4\Delta$. □

Lemma 9. *Assume the fragile complexity of any element is at most $\Delta \geq 2^7$. Then for $n > 2^9$ the expected fragile complexity of the minimum is $(1/96)\log_\Delta n$.*

Proof. Choose an input position i and the remaining instance uniformly at random. Let $c_1, \ldots, c_k$ be the compared to values when setting $x_i = 0$ and running the algorithm. With a failure probability of $2/n$ we have

$$p\left(\exists c_j \in \left[\frac{b}{2(16\Delta)^2}, b\right]\right) \geq 1/4 \qquad (1)$$

use Lemma 8: When uniformly choosing $x \in (0, b)$, with probability $1/2$ we have $x \in (b/2, b)$. Let c_j be the element to filter such an x. Then the log-ratio of this filtering is $\log x/c_j \leq 2\log(16\Delta)$ (twice the expected value), if $c_j \in (\frac{b}{2(16\Delta)^2}, x)$. Now, (1) follows from a Markov bound.

Define $R = \frac{1}{2(16\Delta)^2}$ and $X_h = \exists c_j \in [R^{h+1}, R^h]$ and observe (1) implies $E[X_h] \geq 1/4$.

Define H as large as possible such that $R^{h+1} > \frac{2\log n}{n}$, i.e. $H = \log_R \frac{n}{2\log n} = \frac{\log n - \log 2\log n}{\log(2(8\Delta)^2)} \geq \frac{1}{12}\log_\Delta n$. Unless the construction fails with probability $2H/n < 1/2$, the expected fragile complexity of the minimum is at least $H/4$. □

5 Selection and median

5.1 Deterministic selection, comparator networks

We begin this section by discussing the (n, t)-selection problem in the setting of comparator networks. We present an upper and a matching lower bound on the size of a comparator network solving the (n, t)-selection problem. We proceed afterwards by considering the problem in the setting of comparison-based algorithms. We present an algorithm that finds the median of a set of n elements with the same fragile complexity as in the case of the comparator networks, but with total work that is asymptotically smaller. Note, that we chose to discuss the median problem here for simplicity of exposition, and our algorithm can be generalized to solve the (n, t)-selection problem. We thus show the separation between the two models.

To begin, observe that the (n, t)-selection problem can be solved by sorting the input. Therefore, the (n, t)-selection problem can be solved using a comparator network of size $\mathcal{O}(n\log n)$ and depth $\mathcal{O}(\log n)$ by using the AKS sorting network [2]. Consequently, the (n, t)-selection problem can be solved with $\mathcal{O}(\log n)$ fragile complexity and $\mathcal{O}(n\log n)$ work.

Next, we show that the size of the (n, t)-selection network using the AKS network is asymptotically tight. Before we present the lower bound theorem, we need to introduce some notation and prove two auxiliary lemmas.

Given a set X, we say two elements $x_i, x_j \in X$ are *rank-neighboring* if $|rank(x_i) - rank(x_j)| = 1$. We say a permutation $\hat{\pi}(X)$ is *rank-neighboring* if an ordered sequence X and $\hat{\pi}(X)$ differ in only two elements and these two elements are rank-neighboring. Observe that any permutation $\pi(X)$ is a composition of some number of rank-neighboring permutations.

We define the *signature* of an ordered sequence X with respect to an integer a to be a function $\sigma : \mathbb{Z} \times \mathbb{Z}^{n-1} \to \{0, 1\}^{n-1}$, such that $\sigma(a, (x_1, \ldots, x_{n-1})) = (y_1, \ldots, y_{n-1})$ and for all $1 \le i \le n-1$:

$$y_i = \begin{cases} 0 & \text{if } x_i \le a \\ 1 & \text{if } x_i > a \end{cases}$$

Lemma 10. *For any totally ordered set X of size n, any (n, t)-selection network $\mathcal{N}_{n,t}$, and for any rank-neighboring permutation $\hat{\pi}$: $\sigma(\mathcal{N}_{n,t}(X)) = \sigma(\mathcal{N}_{n,t}(\hat{\pi}(X)))$.*

Proof. Consider the two inputs x_i and x_j in which X and $\hat{\pi}(X)$ differ. During the computation the input values traverse the network until they reach the outputs. During the computation of the network $\mathcal{N}_{n,t}(X)$ element x_i (resp., x_j) starts at the i-th (resp., j-th) input and reaches the i'-th (resp., j'-th) output. During the computation of the network $\mathcal{N}_{n,t}(\hat{\pi}(X))$, element x_i (resp., x_j) starts at the j-th (resp., i-th) input. Let us determine which outputs they reach.

Consider the two paths P_i and P_j that x_i and x_j, respectively, traverse during the computation of $\mathcal{N}_{n,t}(X)$. Since $\hat{\pi}$ is a rank-neighboring permutation, the outputs of every comparator C are the same for $\mathcal{N}_{n,t}(X)$ and $\mathcal{N}_{n,t}(\hat{\pi}(X))$ except for the (set of) comparator(s) C^*, whose two inputs are x_i and x_j. Therefore, throughout the computation of $\mathcal{N}_{n,t}(\hat{\pi}(X))$, x_i and x_j only traverse the edges of the paths $P_i \cup P_j$, i.e., the outputs of $\mathcal{N}_{n,t}(\hat{\pi}(X))$ are the same as the outputs of $\mathcal{N}_{n,t}(X)$ everywhere except for, possibly, at the i'-th and j'-th output.

If $rank(x_i) < t$ and $rank(x_j) < t$, or $rank(x_i) > t$ and $rank(x_j) > t$, the signatures of these two outputs are the same. Consequently, $\sigma(\mathcal{N}_{n,t}(X)) = \sigma(\mathcal{N}_{n,t}(\hat{\pi}(X)))$. Otherwise, without loss of generality, let $rank(x_i) = t$ (the case of $rank(x_j) = t$ is symmetric). Then $i' = 0$, and $\mathcal{N}_{n,t}(X)^0 = \mathcal{N}_{n,t}(\hat{\pi}(X))^0 = x_i$. Then, $\mathcal{N}_{n,t}(X)^{j'} = \mathcal{N}_{n,t}(\hat{\pi}(X))^{j'} = x_j$, and it follows that $\sigma(\mathcal{N}_{n,t}(X)) = \sigma(\mathcal{N}_{n,t}(\hat{\pi}(X)))$. □

Lemma 11. *An (n, t)-selection network can be turned into an (n, t)-partition network.*

Proof. Since every permutation $\pi(X)$ can be obtained from X by a sequence of rank-neighboring permutations, it follows from Lemma 10 that $\sigma(\mathcal{N}_{n,t}(\pi(X))) = \sigma(\mathcal{N}_{n,t}(X))$, i.e., for every permutation of the inputs in the (n, t)-selection network, the same subset of outputs carry the values that are at most t. Thus, the (n, t)-partition network can be obtained from the (n, t)-selection network by reordering (re-wiring) the outputs such that the ones with signature 0 are the first t outputs. $\square$

Theorem 6. *An (n, t)-selection network for any $t \leq n/2$ has size $\Omega((n - t)\log(t + 1))$.*

Proof. Alekseev [4] showed the $\Omega((n - t)\log(t + 1))$ lower bound for the size of an (n, t)-partition network, and, by Lemma 11, every (n, t)-selection network also solves the (n, t)-partition problem. $\square$

Corollary 2. *The size of a comparator network that finds the median of n elements is $\Omega(n\log n)$.*

5.2 Deterministic selection, comparison-based algorithms

Above, we have shown tight upper and lower bounds on the size of comparator networks for the (n, t)-selection problem. These bounds immediately imply $\mathcal{O}(n\log n)$ work and $\mathcal{O}(\log n)$ fragile complexity for the problem of selection. We now show that we can find the median in linear work, while keeping the fragile complexity the same, by developing a comparison-based algorithm instead of relying on the simulation of comparator networks. Observe that by using a simple padding of the input, this immediately provides us with a solution to the (n, t)-selection problem for an arbitrary $t \neq \frac{n}{2}$.

Theorem 7. *There is a deterministic algorithm for finding the median which performs $\mathcal{O}(n)$ work and has $\mathcal{O}(\log n)$ fragile complexity.*

Proof. A central building block from the AKS sorting network is an *ε-halver*. 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 implementing an ε-halver in constant depth can be built [1, 3]. We use the corresponding comparison-based algorithm of constant fragile complexity.

We first use an ε-halver on the input array S of length n, dividing it into two subarrays of length $n/2$. The right of these, S_1, contains l of the smallest $n/2$ elements from S and $n/2 - l$ of the $n/2$ largest. By the properties of the ε-halver, $l \leq \varepsilon n/2$. We call these l elements in S_1 *left far*, and call the rest of the elements in S_1 *good.* The left far elements will be the smallest elements in S_1.

We next use an ε-halver on S_1, dividing it into two subarrays of S_1 of length $n/4$. The left of these, S_2, contains r of the smallest $n/4$ elements from S_1 and $n/4 - r$ of the $n/4$ largest. By the properties of the ε-halver, $r \leq \varepsilon n/4$. We call these r elements in S_2 *right far.* The right far elements will be the largest elements in S_2. Some or all of the at most $n/2$ left far elements of S_1 will appear also in S_2, and these we call the left far elements of S_2. The left far elements will be the smallest elements in S_2. The rest of the elements of S_2 are called good.

We continue in this fashion, alternatingly choosing left and right half as the next array part. This division process is illustrated in Figure 2.

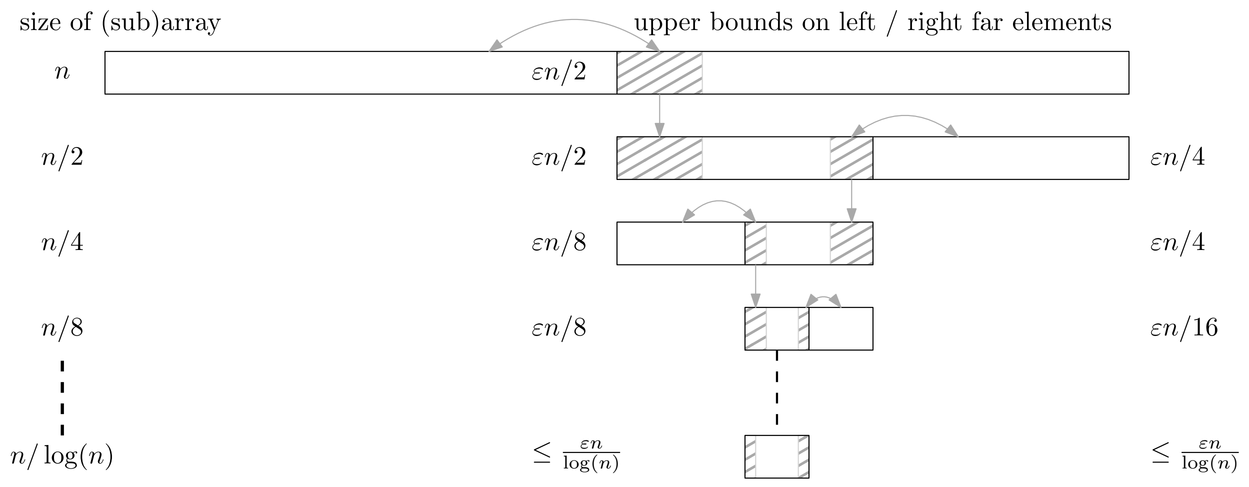

Figure 2: Illustration of the alternating division process using ε-halvers.

After i steps, we have an array part S_i of length $n_i = n/2^i$ with at most εn_i left far and at most εn_i right far elements. For an element x in S_i, let its position be its array index (in S) if S_i were sorted, and consider the difference $\Delta(x)$ between its position and its rank in S. Initially in S, $\Delta(x)$ is zero. For a good element in S_1, $\Delta(x)$ can increase from its value in in S, but at most by $l \leq \varepsilon n/2$. For a good element in S_2, $\Delta(x)$ can decrease from its value in in S_1, but at most by $l \leq \varepsilon n/4$. For a good element in S_3, $\Delta(x)$ can increase from its value in S_2, but at most by $l \leq \varepsilon n/8$, and so forth. Summing the shifts for all levels, we for a good element in S_i have $|\Delta(x)| \leq \varepsilon n$.

We stop the process after $k = \log\log n$ steps. This results in an array part S_k of length $n/\log n$. We sort the $n/\log n$ elements of S_k using an AKS-based sorting algorithm, which takes time $\mathcal{O}(n)$ and has fragile complexity $\mathcal{O}(\log n)$, and we then extract the middle $(n/\log n)/2$ of these sorted elements as the set RP (for "right pivots"). If we choose $\varepsilon < 1/4$, these are all good. For odd $k = 2t + 1$, the first index of S_k can from Figure 2 be seen to be

$$\frac{n}{2}\sum_{i=0}^{t}(\frac{1}{4})^i = \frac{n}{2} \cdot \frac{1-(1/4)^{t+1}}{1-1/4} = \frac{2n}{3}(1-\frac{1}{2^{k+1}}) = \frac{2n}{3}(1-\frac{1}{2\log n})\,.$$

For even k, the first index is it the same as for the previous odd k, leading to the first index of S_k being $2n/3(1 - 1/\log n)$. The largest rank in S for a good element x in S_k is at most the first index of S_k plus the length of S_k plus the maximal $|\Delta(x)|$, which is at most

$$\frac{2n}{3}(1-\frac{1}{2\log n}) + \frac{n}{\log n} + \varepsilon n = n(\frac{2}{3} + \frac{2}{3\log n} + \varepsilon)\,.$$

For $\varepsilon < 1/12$ and n large enough, this is at most $3n/4$. The smallest rank in S for a good element x in S_k is at least the first index of S_k minus the maximal $|\Delta(x)|$, which is at least

$$\frac{2n}{3}(1-\frac{1}{\log n}) - \varepsilon n = n(\frac{2}{3} - \frac{2}{3\log n} - \varepsilon)\,.$$

For $\varepsilon < 1/12$ and n large enough, this is at least $7n/12$. The same bounds applies to all elements x of RP, since these are good elements from S_k. We now compare all elements y of S with some element x from RP, using each of the $(n/\log n)/2$ different x for $2\log n$ out of the total n comparisons. If $x \geq y$, we mark y by $\mathcal{R}$. An element marked by $\mathcal{R}$ has rank a in S of at least $7n/12$, and all elements having a rank in S above $3n/4$ are marked by $\mathcal{R}$.

We then perform a symmetric process in the first half of S, leading to a set $\overline{S}_k$ of size $(n/\log n)$ which has a subset LP (for "left pivots") of $(n/\log n)/2$ elements whose rank in S are at least $n/4$ and at most $5n/12$. We compare all elements y of S with some element x' from LP, using each of the $(n/\log n)/2$ different x' for $2\log n$ out of the total n comparisons. If $y < x'$, we mark y by $\mathcal{L}$. An element marked by $\mathcal{L}$ has a rank in S of at most $5n/12$, and all elements having a rank in S below $n/4$ are marked by $\mathcal{L}$.

From the above rank bounds, there are for both $\mathcal{L}$ and $\mathcal{R}$ at least $n/4$ elements having that mark, and no element has both marks. We choose a set R of $n/4$ elements marked $\mathcal{R}$ and a set L of $n/4$ elements marked $\mathcal{L}$. We then create $S' = S \setminus (L \cup R)$. We have $|S'| = |S|/2$. By the rank bounds above, S' contains the median m of S. Since $|L| = |R|$, m has rank $|S'|/2$ in S', i.e., is the median of S'. However, we cannot recurse directly on S, since the elements of S_k and $\overline{S}_k$ already may have endured $\Theta(\log n)$ comparisons, and some of these elements may be in S'. Hence, we create $S'' = S' \setminus (S_k \cup \overline{S}_k)$. Since m actually may be an element of S_k or $\overline{S}_k$, we also form $LP' = LP \cap S'$ and $RP' = RP \cap S'$ and put them aside for final inspection at the end of the algorithm. After this, the rank of m in S'' can differ from $|S''|/2$ (i.e., from the rank of the median of S''), but not by more than $n/\log n$, since S_k and $\overline{S}_k$ have combined size $2n/\log n$ (it takes the deletion of two elements to move the median by one element).

We then recurse on S'', i.e., start the entire process again with S'' in the place of S. Let the sizes of the input array during the recursion be $n = n_0, n_1, n_2, \ldots$. From the size bounds above, we have $n_{i+1} \leq n_i/2$. We continue the recursion as long as $n_i \geq 72n/\log n$.

If m is still part of the input for the ith level of the recursion, its rank at that level can at most be

$$\sum_{j=0}^{i} \frac{n_j}{\log n_j} \leq \frac{1}{\log n_i} \sum_{j=0}^{i} \frac{n}{2^j} \leq \frac{2n}{\log(72n/\log n)} \leq \frac{2n}{\log(n/\log n)} < \frac{6n}{\log n} \qquad (2)$$

different from the rank of the median at that level. This bound arises by summing over the recursive levels the above bound on how far the median at one level can be in rank from the median at the next level. For the last inequality, we used the fact that $\log(n/\log n) > (\log n)/3$ for all n. As $n_i \geq 72n/\log n$, this is less than the minimal rank distance of $n_i/12$ between the median of the level and any element in the L and R sets of the level. Hence all L (R) sets removed during the recursion lie entirely below (above) the global median m. This implies that m is contained in the union M of the input of the base case and all the LP' and LR' sets formed during the recursion. As $|L| = |R|$ at each level of the recursion, it also implies that m is the median of M, since M and the L and R sets together constitute a partitioning of the original input. By the size of the base case input, the bounds on the LP' and LR' sets, and their sum (2), M contains $\mathcal{O}(n/\log n)$ elements. We finally find m by sorting M using an AKS-based sorting algorithm.

We now analyze the fragile complexity. Invoking an ε-halver incurs $\mathcal{O}(1)$ comparisons on each element, so a full division process incurs $\mathcal{O}(\log\log n)$ comparisons on each element, except for the elements in the final S_k or $\overline{S}_k$ set. These each take part in $\mathcal{O}(\log n)$ comparisons during the AKS-based sorting, and some take part in $\mathcal{O}(\log n)$ further comparisons during the marking phase. However, we ensured that this can only happen once for an element during the entire algorithm. Accounting for the recursion, which has has depth at most $\log\log n$, and the final sorting of M, we see that each element takes part in $\mathcal{O}((\log\log n)^2 + \log n) = \mathcal{O}(\log n)$ comparisons.

Regarding the work, invoking an ε-halver incurs a linear number of comparisons. By the geometrically decreasing sizes during a division process, all invocations in a division process combined incur $\mathcal{O}(n)$ comparisons. The same holds for the sorting of S_k or $\overline{S}_k$, as their size is $n/\log n$, and for the marking phase. Hence, the work on each level in the recursion is linear in its input, and by the geometrically decreasing input sizes, the entire recursion incurs $\mathcal{O}(n)$ work, as does the final sorting of M. □

5.3 Randomized selection

We present RMedian an expected work-optimal median selection algorithm with a trade-off between the expected fragile complexity $f_{\text{med}}(n)$ of the median element and the maximum expected fragile complexity $f_{\text{rem}}(n)$ of the remaining elements. By adjusting a parameter affecting the trade-off, we can vary $\langle \mathbb{E}\left[f_{\text{med}}(n)\right], \max \mathbb{E}\left[f_{\text{rem}}(n)\right]\rangle$ in the range between $\langle \mathcal{O}(\log\log n), \mathcal{O}(\sqrt{n})\rangle$ and $\langle \mathcal{O}(\log n/\log\log n), \mathcal{O}(\log n)\rangle$ (see Theorems 8 and 9).

The algorithm (detailed in Algorithm 1 in Appendix 6) takes a totally ordered set X as input. It draws a set S of $k(n)$ random samples, sorts them and subsequently uses the items in S as pivots to identify a set $C \subset X$ of values around the median, such that an equal number of items smaller and greater than the median are excluded from C. Finally, it finds the median of C, which is the median of the initial input, recursively. The recursion reaches the base case when the input is of size $\mathcal{O}(\text{polylog}\, n)$, at which point it can be sorted to trivially expose the median. RMedian employs two sets L_1 and R_1 of $n_1 = \mathcal{O}(\sqrt{k \log n})$ pivots almost surely below and above the median respectively. All candidates for C are compared to one item in L_1 and R_1 each filtering elements that are either too small or too large. The sizes of L_1, R_1 and C are balanced to achieve fast pruning and low failure probability[3] (see Lemmas 12 and 13).

To reduce the fragile complexity of elements in L_1 and R_1, most elements are prefiltered using a cascade of weaker classifiers L_i, R_i for $2 \leq i \leq b$ geometrically growing in size by a factor of $d(n)$ when moving away from the center. Filtered elements that are classified into a bucket L_i or B_i with $i < b$ are used as new pivots and effectively limit the expected load per pivot. As the median is likely to travel through this cascade, the number of filter layers is a compromise between the fragile complexity f_{med} of the median and f_{rem} of the remaining elements.

We define $k(n)$ and $d(n)$ as functions since they depend on the problem size which changes for recursive calls. If n is unambiguous from context, we denote them as k and d respectively and assume $k(n) = \Omega(n^{\varepsilon})$ for some $\varepsilon > 0$.

Lemma 12. *Consider any recursion step and let X be the set of n elements passed as the subproblem. After all elements in X are processed, the center partition C contains the median $x_m \in X$ whp.*

Proof. The algorithm can fail to move the median into bucket C only if the sample $S \subset X$ is highly skewed. More formally, we use a safety margin n_0 around the median of S and observe that if there exists $x_l, x_r \in S_C := S[k/2 - n_0 : k/2 + n_0]$ with $x_l < x_m < x_r$, the median x_m is moved into C.

This fails in case too many small or large elements are sampled. In the following, we bound the probability of the former from above; the symmetric case of too many large elements follows analogously. Consider k Bernoulli random variables X_i indicating that the i-th sample $s_i < x_m$ lies below the median and apply Chernoff's inequality $\Pr\left[\sum_i X_i > (1+\delta)\mu\right] \leq \exp(-\mu\delta^2/3)$ where

[3]RMedian is guaranteed to select the correct median. Failure results in an asymptotically insignificant increase of expected fragile complexity (see line 21 in Algorithm 1, Appendix 6).

$\mu = \mathbb{E}\left[\sum_i X_i\right]$ and $\delta < 1$:

$$\Pr\left[\neg\exists\ x_r \in S_C \text{ with } x_m < x_r\right] = \Pr\left[\sum_{i=1}^{k} X_i > k/2 + n_0\right] \leq \exp\left(-\frac{2n_0^2}{3k}\right) \overset{n_0=\sqrt{2k\log n}}{=} n^{-4/3}\,. \quad \square$$

Lemma 13. *A recursion step of* RMedian *reduces the problem size from n to $\mathcal{O}(\sqrt{n\log n})$ whp.*

Proof. The algorithm recurses on the center bucket C which contains the initial sample of size $2n_0 = \mathcal{O}(\sqrt{n\log n})$ and all elements that are not filtered out by L_1 and R_1. We pessimistically assume that each element added to C compared to the weakest classifiers in these filters (i.e., the largest element in R_1 and the smallest in L_1).

We hence bound the rank in X of the R_1's largest pivot; due to symmetry L_1 follows analogously. Using a setup similar to the proof of Lemma 12, we define Bernoulli random variables indicating that the i-th sample s_i is larger than the ℓ-th largest element in X where $\ell = n/2 + 3\sqrt{n\log(n)}$. Applying Chernoff's inequality yields the claim. $\square$

Lemma 14. *The expected fragile complexity of the median is*

$$\mathbb{E}\left[f_{med}(n)\right] = f_{med}\left(\mathcal{O}(\sqrt{n\log n})\right) + \mathcal{O}\Big(\underbrace{\frac{k}{n}\log k}_{\textit{Sampled}} + \underbrace{(1-\frac{k}{n})\log_d k}_{\textit{Not sampled}} + \underbrace{1}_{\textit{Misclassified}}\Big)\,.$$

Proof. Due to Lemma 13, a recursion step reduces the problem size from n to $\mathcal{O}(\sqrt{n\log n})$ whp resembled in the recursive summand. The remaining terms apply depending on whether the median is sampled or not: with $\Pr[x_m \notin S] = 1 - k/n$ the median is not sampled and moved towards the center triggering a constant number of comparisons in each of the $\mathcal{O}(\log_d n)$ buckets. Otherwise if the median x_m is sampled, it incurs $\mathcal{O}(\log k)$ comparisons while S is sorted. By Lemma 12, the median is then assigned to C whp and protected from further comparisons. The complementary event of x_m being misclassified has a vanishing contribution due to its small probability. $\square$

Lemma 15. *The fragile complexity of non-median elements is*

$$\mathbb{E}\left[f_{rem}\right] = f_{rem}\left(\mathcal{O}(\sqrt{n\log n})\right) + \mathcal{O}\Big(\underbrace{\log k}_{\textit{Sampled}} + \underbrace{\log_d n}_{\textit{Not sampled}} + \max(\underbrace{d^2}_{\textit{Pivot in } R_i,\ i>2}, \underbrace{\frac{nd}{k}}_{\textit{Pivot in } R_j,\ j\leq 2})\Big)\,,$$

where $d(n) = \Omega(\log^{\varepsilon} n)$ for some $\varepsilon > 0$ and $k(n) = \mathcal{O}(n/\log n)$.

Proof. The recursion is implemented analogously to proof of Lemma 14, so we discuss only the contribution of a single recursion step. Let $x \in (X \setminus \{x_m\})$ be an arbitrary non-median element.

The element x is either sampled and participates in $\mathcal{O}(\log k)$ comparisons. Otherwise it traverse the filter cascade and moves to C, L_i or R_i requiring $\mathcal{O}(\log_d n)$ comparisons.

If it becomes a median candidate (i.e. $x \in C$), x has a fragile complexity as discussed in Lemma 14 which is asymptotically negligible here. Thus we only consider the case that x is assigned to L_i or R_i and due to symmetry assume without loss of generality that $x \in R_i$. If it becomes a member of the outer-most bucket R_b, it is effectively discarded. Otherwise, it can function as a new pivot element replenishing the bucket's comparison budget. As RMedian always uses a bucket's least-frequently compared element as pivot, it suffices to bound the expected number of comparisons until a new pivot arrives.

Observe that RMEDIAN needs to find an element $y \in (R_{i-2} \cup R_{i-1})$ with $y < x$ in order to establish that $x \in R_i$. This is due to the fact that pivots can be placed near the unfavorable border of a bucket rendering them weak classifiers. We here pessimistically assume that $y \in R_{i-2}$ for simplicity's sake. By construction the initial bucket sizes n_i grow geometrically by a factor of d as i increases. Therefore, any item compared to bucket R_i continues to the next bucket with probability at most $1/d$. Consequently, bucket R_i with $i > 2$ sustains expected $\mathcal{O}(d^2)$ comparisons until a new pivot arrives.

For the two inner most buckets R_j with $j \in \{1, 2\}$, this is not true as they are not replenished. Bucket R_j ultimately receives a fraction $\mathcal{O}(n_i/k)$ of all items whp, however it is expected to process a factor d more comparisons due to the possibly weak classification in the previous bucket R_{j+1}. Since there are n_i pivots in bucket R_j, each pivot in R_j participates in $\mathcal{O}(kd/n)$ comparisons. □

Theorem 8. RMEDIAN *achieves* $\mathbb{E}[f_{med}(n)] = \mathcal{O}(\log\log n)$ *and* $\mathbb{E}[f_{rem}(n)] = \mathcal{O}(\sqrt{n})$.

Proof. Choose $k(n) = n^{\varepsilon}$, $d(n) = n^{\delta}$ with $\varepsilon = 2/3$ and $\delta = 1/12$. Then Lemmas 14 and 15 yield:

$$f_{\text{med}}(n) = f_{\text{med}}(\mathcal{O}(\sqrt{n\log n})) + \mathcal{O}(n^{\varepsilon-1}\varepsilon\log n + \frac{\varepsilon}{\delta}) = \mathcal{O}(\log\log n)\,,$$

$$f_{\text{rem}}(n) = f_{\text{rem}}(\mathcal{O}(\sqrt{n\log n})) + \mathcal{O}((\varepsilon + \frac{1}{\delta})\log n + \max(n^{2\delta}, n^{1-\varepsilon+2\delta})) = \mathcal{O}(\sqrt{n})\,.$$ □

Theorem 9. RMEDIAN *achieves* $\mathbb{E}[f_{med}(n)] = \mathcal{O}(\frac{\log n}{\log\log n})$ *and* $\mathbb{E}[f_{rem}(n)] = \mathcal{O}(\log n)$.

Proof. Choose $k(n) = n/\log^{\varepsilon} n$, $d(n) = \log^{\delta} n$ with $\varepsilon = \delta = 1/3$. Then Lemmas 14 and 15 yield:

$$f_{\text{med}}(n) = f_{\text{med}}(\mathcal{O}(\sqrt{n\log n})) + \mathcal{O}(\log^{1-\varepsilon} n + \log_{\log^{\delta} n} n) = \mathcal{O}(\frac{\log n}{\log\log n})\,,$$

$$f_{\text{rem}}(n) = f_{\text{rem}}(\mathcal{O}(\sqrt{n\log n})) + \mathcal{O}(\log n + \max(\log^{2\delta} n + \log^{\varepsilon+\delta} n)) = \mathcal{O}(\log n)\,.$$ □

Theorem 10. *For* $k = \mathcal{O}(n/\log n)$ *and* $d = \Omega(\log n)$, RMEDIAN *performs a total of* $\mathcal{O}(n)$ *comparisons in expectation, implying* $w(n) = \mathcal{O}(n)$ *expected work.*

Proof. We consider the first recursion step and analyze the total number of comparisons. RMEDIAN sorts k elements using AKS resulting in $\mathcal{O}(k\log k) = \mathcal{O}(n)$ comparisons. It then moves $\mathcal{O}(n)$ items through the filtering cascade consisting of buckets of geometrically decreasing size resulting of $\mathcal{O}(1)$ expected comparisons per item. Each bucket stores its pivots in a minimum priority queue with the number of comparisons endured by each pivot as keys. Even without exploitation of integer keys, retrieving and inserting keys is possible with $\mathcal{O}(\log n)$ comparisons each. Hence, we select a pivot and keep it for $d = \Omega(\log n)$ steps, resulting in amortized $\mathcal{O}(1)$ work per comparison. This does not affect f_{med} and f_{rem} asymptotically. Using Lemma 13, the total number of comparisons hence $g(n) = g(\sqrt{n\log n}) + \mathcal{O}(n) = \mathcal{O}(n)$. □

6 Randomized selection

6.1 Sublogarithmic for all [This currently does not work and needs a lot more work to be fixed]

Theorem 11. *There is a randomized algorithm that finds the median with high probability and an expected maximum (fragile) number comparisons of* $\mathcal{O}(\log n/\log\log n)$.

```
procedure RMEDIAN($X = \{x_1, \ldots, x_n\}, k(n), d(n)$)    ▷ Sampling phase
    Randomly sample $k$ elements from $X$ and sort $S$ with AKS
    Distribute $S$ into buckets $L_b, L_{b-1}, \ldots L_1, C, R_1, \ldots, R_{b-1}, R_b$ as follows:
        set $n_0 = 2\sqrt{k \log n}, n_1 = 3\sqrt{k \log n}, n_i = d \cdot n_{i-1}$
        $C = S[k/2 - n_0 : k/2 + n_0]$ median candidates
        $L_i = S[k/2 - n_i : k/2 - n_{i-1}]$ buckets of elements presumed smaller than median
        $R_i = S[k/2 + n_{i-1} : k/2 + n_i]$ buckets of elements presumed larger than median
                                                                    ▷ Probing phase
    for $x_i \in X \setminus S$ in random order
        for $j \in [b-1, \ldots, 1]$ in order
            $x_A \leftarrow$ arbitrary element in $L_j$ with fewest compares
            $c \leftarrow 1$ if $x_A$ is marked else 2
            if $x_i < x_A$
                add $x_i$ as new pivot to $L_{b+c}$ if $j < b - c$ and mark it, otherwise discard $x_i$
                stop processing $x_i$
            $x_B \leftarrow$ arbitrary element in $R_j$ with fewest compares
            $c \leftarrow 1$ if $x_B$ is marked else 2
            if $x_i > x_B$
                add $x_i$ as new pivot to $R_{b+c}$ if $j < b - c$ and mark it, otherwise discard $x_i$
                stop processing $x_i$
                                  ▷ By now it is established that $S[k/2 - n_1] \le x_i \le S[k/2 + n_1]$
        add $x_i$ as a median candidate to $C$
    if $\max(\sum_i |L_i|, \sum_i |R_i|) > n/2$        ▷ Partitioning too imbalanced ⇒ median not in $C$
        return DETMEDIAN(X)
    if $|C| < \log^4 n$
        sort $C$ with AKS and return median
    $k = \sum_i (|L_i| - |R_i|)$
    if $k > 0$
        add $k$ arbitrary elements from $\bigcup_i R_i$ to $C$
    else
        add $k$ arbitrary elements from $\bigcup_i L_i$ to $C$
    return RMEDIAN(C, k(n), b(n))
```

Algorithm 1: Randomized median selection

original, not working. Algorithm (find median and classify all elements):

1. Sample a set S with $|S| = n/\log^{\varepsilon}$
2. Recurse on S leading to $L < m < R$
3. each element R is participating in 10 random comparisons with other elements in R (10 random matchings)
4. R' are the local maxima of R, $|R'| \approx |R| 2^{-10}$
5. partition R' into sets of size $T = \log^{\varepsilon} n$

6. P are the R'/T local minima of the parts, $|P| = n/\log^{2\varepsilon} n$

7. filter with P, $\log^{2\varepsilon} n$ comparisons fragile

8. recurse, set size $\mathcal{O}(n/\log^{\varepsilon} n)$

$\square$

[Another attempt at randomized median (still messy for now):]

Given a set X of size n.

Given a parameter s (to be determined later), we say for $1 \leq i \leq \delta \log_s n$, $x \in X$ belongs to the i-th *band*, denoted by $\mathcal{B}_i$, if

$$\frac{n}{s^i} < \left| \frac{n}{2} - rank(x) \right| \leq \frac{n}{s^{i-1}}$$

If $0 \leq \left| \frac{n}{2} - rank(x) \right| \leq n^{1-\delta}$, we say x belongs to the "bullseye" band, $\mathcal{B}_b$.

The following procedure takes a set of size n and (approximately) partitions its elements into $c\delta \log_s(n) + 1$ "bands", where c, δ and $s = s(n)(?)$ are parameters to be determined later.

```
1: **procedure** MEDIAN($X, n$)
2:     **while** $|X| > n^{1/\log\log n}$
3:         $\delta \leftarrow \delta(|X|)$
4:         $s \leftarrow s(|X|)$
5:         $\mathcal{B} \leftarrow$ PARTITION($X, |X|, s, \delta$)
6:         $X \leftarrow \mathcal{B}_b$
7:     AKS-SORT($X$)
8:     **return** $x_{|X|/2}$
```

```
1: **procedure** PARTITION($X, n, s, \delta$)
2:     **if** $|X| < n^{1-\delta}$                ▷ NS: Not sure if we need the basecase, maybe for samples
3:         Move $X$ into $\mathcal{B}_b$ and **return** $\mathcal{B}$
4:     $k = \delta \log_s n$
5:     Let $X'$ be a random sample of $X$ of size $n' = |X|/s$ chosen uniformly at random.
6:     $X \leftarrow X \setminus X'$                                ▷ Extract the samples from $X$
7:     $\mathcal{B}' \leftarrow$ PARTITION($X', n, s, \delta$)                      ▷ NS: same $n$, $s$, and $\delta$?
8:
9:     $\mathcal{B} \leftarrow$ FILTER($X, \mathcal{B}', k, s$)          ▷ Partition elements of $X$ into sets $\mathcal{B}$ using $\mathcal{B}_i'$s as pivots
10:
11:    **for** $i = 1$ to $k$
12:        $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \mathcal{B}_i'$
13:    $\mathcal{B}_b \leftarrow \mathcal{B}_b \cup \mathcal{B}_b'$
14:    **return** $\mathcal{B}$
```

The following procedure partitions a set X around the pivots by iteratively filtering all elements of X through each band of $\mathcal{P}$

```
1: **procedure** FILTER($X, \mathcal{P}, k, s$)
2:     Let $\mathcal{B}$ be a new set of partitions
3:     **for** $i = 1$ to $k$
4:                          ▷ For each element in $\mathcal{P}_i$ compare it $|X|/|\mathcal{P}_i|$ random elements of $X$
```

```
 5:        Partition $X$ into $|\mathcal{P}_i|$ random subsets $X_1, \ldots, X_{|\mathcal{P}_i|}$ each of size $|X|/|\mathcal{P}_i|$
 6:        for $j = 1$ to $|\mathcal{P}_i|$
 7:            for all $e \in X_j$
 8:                if $\mathcal{P}_i[j] \leq e$ :  Move $e$ from $X_j$ to $\mathcal{B}_i$
 9:    Move the remaining elements in $X$ into bullseye $\mathcal{B}_b$
10:    return $\mathcal{B}$
```

7 k smallest elements

Theorem 12. *can be found with* $\log k/\log\log k$ *expected max fragile complexity (max of the k-smallest)*
A bound of Δ on everybody leads to a plus $\log_\Delta n$

Theorem 13. *this is optimal*

Proof. Note that the lower bound holds as much for the second smallest.

When identifying the k-smallest elements, either the rank k or $k+1$ element must by identified. Hence the lower bound of the min applies to the rank k element by padding, and the other elements are the smallest elements. □

7.1 Algorithm

sample with probability $1/k$ and recurse with $k' = \sqrt{k}$.

Use the rank k' element

Then apply the $\log n/\log\log n$ expected max algorithm for the rank k element.

7.2 Simple algorithm for median

Lemma 16. *The median can be found with the maximal fragile complexity being* $\log n/\log\log n$ *whp.*

Proof. Split into parts. In each part sample a fraction of size $\log^{\log\log n} n$. Sort deterministic in fragile complexity $(\log\log n)^2$. Take the rank from median plus $[.5, 1]\log^{(\log\log n)-1/2} n$ to filter down to a $\sqrt{\log n}$ fraction. With this we can set $k = \sqrt{\log n}/\log\log n$. The load from sampling is hence $(\log n/\log\log n) \cdot (\log\log n)^3/\sqrt{\log n}$ which is basically $\sqrt{\log n}$.

The probability of filtering with an element on the wrong side of the median is by a Chernoff bound with n is sample size and ignoring constants $p = 1/\sqrt{\log n}$ $\mu = \log^{(\log\log n)-1/2} n$ $\delta = \sqrt{\log n}$ Hence the probability is negligible.

□

8 Sorting

Recall from Section 2 that for sorting networks, the few existing networks with depth $\mathcal{O}(\log n)$ are all based on expanders, while a number of $\mathcal{O}(\log^2 n)$ depth networks exist, many of which are based on merging. We here study the power of the merging paradigm with respect to fragile complexity for comparison-based algorithms, and show that fragile complexity $\mathcal{O}(\log^2 n)$ can be achieved, that this is best possible for any binary MergeSort, but also that in the expected sense, fragile complexity of standard MergeSort is $\mathcal{O}(\log n)$. We leave open the question whether finding a simple sorting algorithm (for instance, not involving expander graphs) with fragile complexity $\mathcal{O}(\log n)$ is easier than finding a sorting network of depth $\mathcal{O}(\log n)$.

8.1 Lower bound for MergeSort

Lemma 17. *Merging two sorted sequences A and B has fragile complexity at least $\lfloor \log_2 |A| \rfloor + 1$.*

Proof. A standard adversary argument: The adversary designates one element x in B to be the scapegoat and resolves in advance answers to comparisons between A and $B_1 = \{y \in B \mid y < x\}$ by $B_1 < A$ and answers to comparisons between A and $B_2 = \{y \in B \mid x < y\}$ by $A < B_2$. There are still $|A| + 1$ total orders on $A \cup B$ compatible with these choices, one for each position of x in the sorted order of A. Only comparisons between x and members of A can make some of these total orders incompatible with answers given by the adversary. Since the adversary can always choose to answer such comparisons in a way which at most halves the number of compatible orders, at least $\lfloor \log_2 |A| \rfloor + 1$ comparisons involving x have to take place before a single total order is known. □

By standard MergeSort, we mean the algorithm which divides the n input elements into two sets of sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, recursively sorts these, and then merges the resulting two sorted sequences into one. The merge algorithm is not restricted, unless we specify it explicitly (which we only do for the upper bound, not the lower bound).

Lemma 18. *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 17 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

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

comparisons, by Lemma 17 and the fact that a node at height i in the mergetree of standard MergeSort operates on sequences of length $\Theta(2^i)$. □

We now show that making unbalanced merges cannot improve the fragile complexity of binary MergeSort.

Theorem 14. *Any binary mergesort (even if unbalanced) has fragile complexity $\Omega(\log^2 n)$.*

Proof. The adversary is the same as in the proof of Lemma 18, 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 claim 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 17 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 17, we would like to prove for the induction step

$$c\log^2 n_1 + \lfloor \log n_2 \rfloor + 1 \geq c\log^2(n_1 + n_2). \tag{3}$$

This will follow if we can prove

$$\log^2 n_1 + \frac{\log n_2}{c} \geq \log^2(n_1 + n_2)\,. \tag{4}$$

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 (4), we see that (4) 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

$$\frac{\log n_2}{n_2} \geq 2c\frac{\log n_1}{n_1}\,. \tag{5}$$

Since $n_1 \geq n_2$ and $(\log x)/x$ is decreasing for $x \geq e$, we see that (5) 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 (3) reduces to

$$\log^2 n_1 + \frac{1}{c} \geq \log^2(n_1 + 1)\,. \tag{6}$$

Here we can again use concavity and first order Taylor approximation with $x_0 = n_1$ and $x = n_1 + 1$ to argue that (6) 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 $x \geq e$. □

8.2 Upper bound for MergeSort with linear merging

By *linear merging*, we mean the classic sequential merge algorithm that takes two input sequence and iteratively moves the the minimum of both to the output.

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. $b_j < \ldots < b_{j+\ell-1} < a_i$).*

Theorem 15. *Standard* MergeSort *with linear merging has a worst-case fragile complexity $f(n)=\Theta(n)$.*

Proof. *Lower bound $f(n) = \Omega(n)$*: linear merging requires $\mathcal{O}(k)$ comparisons to output a sequence of length k. In standard MERGESORT, each element takes part in $\mathcal{O}(\log n)$ merges of geometrically decreasing sizes $n/2^i$ (from root), resulting in $\mathcal{O}(n)$ comparisons.

Upper bound $f(n) = \mathcal{O}(n)$: consider the input sequence $(n, 1, 2, \ldots, n{-}1)$ where $n = 2^k$. Then every node on the the left-most path of the mergetree contains element n. In each merging step, we receive $A = (1, \ldots, \ell-1, n)$ from the left child, $B = (\ell, \ldots, 2\ell{-}1)$ from the right, and produce

$$(\underbrace{1, \ldots, \ell{-}1}_{\text{from } A},\ \underbrace{\ell, \ldots, 2\ell{-}1}_{\text{from } B},\ \underbrace{\ n\ }_{\text{from } A}).$$

Hence, the whole sequence B is placed directly in front of element n, resulting in $\Theta(\ell)$ comparisons with this element according to Observation 1. Then, the sum of the geometrically increasing sequence length yields the claim. □

Lemma 19. *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 $\Pr[x_i \in X_L] = 1/2$. Consider an arbitrary ordered set $Y = \{y_1, \ldots, y_m\} \subset X$ with $m \le k$. Then $\Pr[Y \subseteq X_L \vee Y \subseteq X_R] < 2^{1-m}$.*

Proof.

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

Theorem 16. *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[4], that any input permutation π is equally likely[5], and that n is a power of two.

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

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

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 19. 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 19 it follows that $\Pr[N_i = m{+}1] \le 2^{-m}$. 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 $Y_{j,i}$

[4]If this is not the case, use input sequence $Y' = ((y_1, 1), \ldots, (y_n, n))$ and lexicographical compares.

[5]If not shuffle it before sorting in linear time and no fragile comparisons.

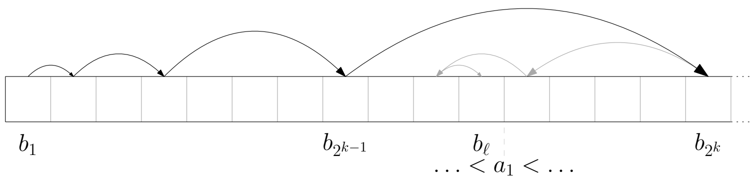

Figure 3: The Exponential search performs k doubling steps and overshoots the target b_ℓ with $b_\ell < a_1 < b_{\ell+1}$. A binary search between $b_{2^{k-1}}$ and b_{2^k} ultimately identifies b_ℓ in $\mathcal{O}(k)$ steps.

are iid in j. Let N_i^T be the total number of comparisons of element i, i.e. $N_i^T \preceq \operatorname{ld} n + \sum_{j=1}^{\log_2 n} Y_{j,i}$. Then a tail bound on the sum of geometric variables (Theorem 2.1 in [14]) yields:

$$\Pr\left[\sum\nolimits_{j=1}^{\log_2 n} Y_{j,i} \geq \lambda \mathbb{E}\left[\sum\nolimits_{j=1}^{\log_2 n} Y_{j,i}\right] = 2\lambda \log_2 n\right] \overset{[14]}{\leq} \exp\left(-\frac{1}{2}\frac{2\ln n}{\ln 2}[\lambda - 1 - \log\lambda]\right) = n^{-2},$$

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

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:

$$\Pr\left[\max_i\{N_i^T\} = \omega(\log n)\right] \leq n \Pr\left[N_i^T = \omega(\log n)\right] \leq 1/n\,. \qquad \square$$

8.3 Upper bound for MergeSort with exponential merging

We define *exponential merging* of sequences $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ as follows: if either A or B are empty, output the other one and stop. Otherwise, assume without loss of generality that m is a power of two and that there exists an $b_i \in Y$ with $a_1 < b_i$, if not append sufficiently many virtual elements $b_\top$ to B with $a_1 < b_\top$. Use an exponential search on B starting in b_1 to find all elements $b_1 < \ldots < b_\ell < a_1$ smaller than a_1. As illustrated in Fig. 3, the exponential search consists of a *doubling phase* which finds the smallest k with $a_1 < b_{2^k}$. Since the doubling phase may overshoot b_ℓ, a *binary search* between $b_{2^{k-1}}$ and y_{b^k} follows. Output $b_1, \ldots, b_\ell, a_1$ and recurse on $A' = [b_{\ell+1}, \ldots, b_m]$ and $B' = [a_2, \ldots, a_n]$ which swaps the roles of A and B.

Theorem 17. *Exponential merging of the sequences $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ has a worst-case fragile complexity of $\mathcal{O}(\log n)$.*

Proof. Without loss of generality let n be a power of two and consider a single exponential search finding the smallest k with $a_1 < b_{2^k}$. The element a_1 is compared to all $\{b_{2^i} \mid 1 \leq i \leq k\}$ during the doubling phase. We use an accounting argument to bound the fragile complexity. Element a_1 takes part in every comparison and is charged with $k = \mathcal{O}(\log n)$. It is then charged $\mathcal{O}(\log n)$ comparisons during the binary search between $b_{2^{k-1}}$ and b_{2^k}. It is then moved to the output and not considered again.

The search also potentially interacts with the 2^k elements $b_1, \ldots, b_{2^k}$ by either comparing them during the doubling phase, during the binary search or by skipping over them. We pessimistically charge each of these elements with one comparison. It then remains to show that no element takes part in more than $\mathcal{O}(\log n)$ exponential searches.

Observe that all elements $b_1, \ldots, b_{2^{k-1}}$ are moved to the output and do not take part in any more comparisons. In the worst-case, the binary search proves that element $b_{2^{k-1}+1}$ and its successors are larger than a_1. Hence at most half of the elements covered by the exponential search are available for further comparisons. To maximize the charge, we recursively setup exponential search whose doubling phases ends in b_{2^l} yielding a recursion depth of $\mathcal{O}(\log n)$.[6] □

Corollary 3. *Applying Theorem 17 to standard* MergeSort *with exponential merging yields a fragile complexity of* $\mathcal{O}(\log^2 n)$ *in the worst-case.*

9 Constructing binary heaps

Theorem 18. *The fragile complexity of the standard binary heap construction algorithm of Floyd [11] is* $\mathcal{O}(\log n)$*.*

Proof. Consider first an element sifting down along a path in the tree: as the binary tree being heapified has height $\mathcal{O}(\log(n))$ and the element moving down is compared to one child per step, the cost to this element before it stops moving is $\mathcal{O}(\log(n))$. Consider now what may happen to an element x in the tree as another element y is sifting down: x is only hit if y is swapped with the parent of x which implies that y was an ancestor of x. As the height of the tree is $\mathcal{O}(\log(n))$, at most $\mathcal{O}(\log(n))$ elements reside on the path above x. Note that the x may be moved up once as y passes by it; this only lowers the number of elements above x. In total, any element in the heap is hit at most $\mathcal{O}(\log(n))$ times during heapify. □

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 Theorem 18 and the fact that Floyd's algorithm has work $\mathcal{O}(n)$, gives for Heap Construction a separation between work of fragility-optimal comparison-based algorithms and size of depth-optimal comparator networks.

10 Nuts and Bolts sorting

Theorem 19. *Randomized Quicksort has expected* $n \log n$ *total number of comparisons but* n *fragile complexity.*

Theorem 20 (Idea)**.** *Randomized merge sort has* $\mathcal{O}(n \log n)$ *expected number of comparisons and fragile* $\mathcal{O}(\log n)$*.*

Theorem 21 (Idea)**.** *There is a variant of AKS splitting that achieves* $\mathcal{O}(n \log n)$ *total number of comparisons and is* $\mathcal{O}(\log n)$ *fragile.*

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *ACM Symposium on Theory of Computing (STOC '83)*, pages 1–9, Baltimore, USA, April 1983. ACM Press.

[6] If the following searches were shorter they would artificially limit the recursion depth. If they were longer, too many elements are removed from consideration as only the binary search range can be charged again.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c\log n$ parallel steps. *Combinatorica*, 3(1):1–19, Mar 1983.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. Halvers and expanders. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 686–692, Pittsburgh, PN, October 1992. IEEE Computer Society Press.

[4] Vladimir 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] Gerth Stølting Brodal and M. Cristina Pinotti. Comparator networks for binary heap construction. In *Proc. 6th Scandinavian Workshop on Algorithm Theory*, volume 1432 of *Lecture Notes in Computer Science*, pages 158–168. Springer Verlag, Berlin, 1998.

[7] Shiva Chaudhuri and Devdatt P. Dubhashi. Probabilistic recurrence relations revisited. *Theor. Comput. Sci.*, 181(1):45–56, 1997.

[8] 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.

[9] Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

[10] M. Dowd, Y. Perl, L. Rudolph, and M. Saks. The periodic balanced sorting network. *J. ACM*, 36(4):738–757, 1989, October.

[11] Robert W. Floyd. Algorithm 245: Treesort. *Commun. ACM*, 7(12):701, December 1964.

[12] Michael T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in $O(n\log n)$ time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 684–693. ACM, 2014.

[13] Hoory, Linial, and Wigderson. Expander graphs and their applications. *BAMS: Bulletin of the American Mathematical Society*, 43:439–561, 2006.

[14] Svante Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018.

[15] 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.

[16] Ian Parberry. The pairwise sorting network. *Parallel Processing Letters*, 2(2-3):205–211, 1992.

[17] Bruce Parker and Ian Parberry. Constructing sorting networks from k-sorters. *Information Processing Letters*, 33(3):157–162, 30 November 1989.

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

[19] Nicholas Pippenger. Selection networks. *SIAM Journal on Computing*, 20(5):878–887, 1991.

[20] Vaughan R. Pratt. *Shellsort and Sorting Networks.* Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1972.

[21] Joel I. Seiferas. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53(3):374–384, 2009.

[22] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

[23] Andrew Chi-Chih Yao and Foong Frances Yao. Lower bounds on merging networks. *J. ACM*, 23(3):566–571, 1976.