Robust-Sorting and Applications to Ulam-Median

Ragesh Jaiswal
IT Delhi, India

Amit Kumar
IIT Delhi, India

Jatin Yadav[1]
IIT Delhi, India

Abstract

Sorting is one of the most basic primitives in many algorithms and data analysis tasks. Comparison-based sorting algorithms, like quick-sort and merge-sort, are known to be optimal when the outcome of each comparison is error-free. However, many real-world sorting applications operate in scenarios where the outcome of each comparison can be noisy. In this work, we explore settings where a bounded number of comparisons are potentially corrupted by erroneous agents, resulting in arbitrary, adversarial outcomes.

We model the sorting problem as a query-limited tournament graph where edges involving erroneous nodes may yield arbitrary results. Our primary contribution is a randomized algorithm inspired by quick-sort that, in expectation, produces an ordering close to the true total order while only querying $\tilde{O}(n)$ edges. We achieve a distance from the target order π within $(3+\epsilon)|B|$, where B is the set of erroneous nodes, balancing the competing objectives of minimizing both query complexity and misalignment with π. Our algorithm needs to carefully balance two aspects – identify a pivot that partitions the vertex set evenly and ensure that this partition is "truthful" and yet query as few "triangles" in the graph G as possible. Since the nodes in B can potentially *hide* in an intricate manner, our algorithm requires several technical steps that ensure that progress is made in each recursive step.

Additionally, we demonstrate significant implications for the Ulam-k-Median problem. This is a classical clustering problem where the metric is defined on the set of permutations on a set of d elements. Chakraborty, Das, and Krauthgamer gave a $(2-\varepsilon)$ FPT approximation algorithm for this problem, where the running time is super-linear in both n and d. We give the first $(2-\varepsilon)$ FPT linear time approximation algorithm for this problem. Our main technical result gives a strengthening of the results in Chakraborty et al. by showing that a good 1-median solution can be obtained from a constant-size random sample of the input. We use our robust sorting framework to find a good solution from such a random sample. We feel that the notion of robust sorting should have applications in several such settings.

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

Keywords and phrases Sorting, clustering, query complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.100

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2502.07653`

Funding *Ragesh Jaiswal*: The author acknowledges the support from the SERB, MATRICS grant. *Jatin Yadav*: The author acknowledges support from Google PhD fellowship.

Acknowledgements We thank anonymous reviewers for their valuable feedback and suggestions.

[1] Corresponding author

52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025).
Editors: Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis
Article No. 100; pp. 100:1–100:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Sorting is one of the most basic primitives in many algorithms and data analysis tasks. The classical model of comparison-based sorting has been extensively studied, where one aims to sort a list of n objects using pairwise comparisons. It is well-known that, in this model, sorting requires at least $n \log n$ comparisons in the worst case. Popular algorithms such as merge sort, quick-sort, and heap-sort achieve this bound with $O(n \log n)$ comparisons.

However, in practical scenarios, sorting is often applied to very large datasets where errors or imperfections in the comparisons are unavoidable. In real-world applications involving noisy data or large-scale distributed systems, comparisons may occasionally be faulty due to hardware imperfections, data corruption, or other noisy sources. Thus, it is crucial to extend classical sorting algorithms to handle such imperfections effectively. An approach for dealing with such noisy errors in sorting was initiated by Feige, Raghavan, Peleg, and Upfal [7]. In their model, each comparison's outcome is flipped independently with some known probability p. Assuming p is a constant, repeatedly querying a pair $\theta(\log n)$ times would ensure that the outcome is correct with high probability. However, this would lead to $O(n \log^2 n)$ number of comparisons. [7] showed that one could instead obtain an algorithm that outputs the sorted order with high probability and performs $O(n \log n)$ comparisons. There has been much recent work [19, 12] on obtaining tight dependence on the parameter p.

Our work is rooted in adversarial settings where each key is controlled by an agent, and some of the agents may be erroneous. Thus, any comparison involving one of the erroneous agents could result in an arbitrary, though deterministic, outcome. Such models are of practical relevance. For example, in distributed computing environments, each data element may be processed by different nodes, and some nodes could behave erratically due to hardware failures or software bugs. Similarly, in data integration tasks, comparisons may be unreliable due to discrepancies between data sources with inconsistent quality. In these scenarios, it is natural to seek sorting algorithms that are robust to a bounded number of adversarial errors.

To formalize this setting, we model the outcome of all pair-wise comparisons between n keys as a tournament graph G (recall that a tournament is a directed graph that has a directed arc between each pair of vertices) and assume that there is a total order π on the vertices of G. If there were no erroneous edges, then the edges of G would be consistent with π (i.e., G has a directed arc (u, v) iff u appears before v in π). However, there are some "bad" nodes B in this graph: querying any edge involving such a bad node as an end-point can lead to an (adversarially) arbitrary outcome. But the outcome of querying any other edge is consistent with π.

The algorithm queries some of the edges of G, and then outputs an ordering on all the vertices of G. The goal is to output an ordering with minimum distance (measured in terms of the length of the longest common subsequence) with respect to π.

Observe that there are two competing objectives here: query as few edges of G as possible and minimize the mismatch with the hidden input sequence. If we query all the $O(n^2)$ edges of G, it is not difficult to show that a 2-approximation algorithm for the feedback vertex set on tournaments [17] can be used to obtain an ordering that has distance at most $3|B|$ from π. In this work, we address the following question:

> *Can we get an efficient algorithm that queries $\tilde{O}(n)$ edges in G, and outputs an ordering that has distance at most $O(|B|)$ from π?*

We answer this question in the affirmative and give a randomized algorithm that comes within distance $(3 + \epsilon)|B|$ of π. Furthermore, it runs in $\tilde{O}(n)$ time. Our algorithm is based on careful identification of good pivots which are likely to be outside the set B. Since the

algorithm does not know the set B, we can only use indirect approaches for this. Our algorithm relies on finding directed triangles in G – we know that there must be at least one vertex of B in such a triangle. However, each failed attempt to identify such a triangle increases the query complexity. Thus, the algorithm needs to carefully balance these two aspects – identify a pivot that partitions the vertex set evenly and ensure that this partition is "truthful" and yet query as few triangles in the graph G as possible. Direct implementations of these ideas do not work for several subtle reasons: (i) a pivot belonging to the set B may be able to *hide* itself by participating in few triangles, and yet, may create large number of misalignments if used for partitioning the set of elements, (ii) elements in $V \setminus B$ will result in truthful partitions of V, but may be involved in lot of triangles (involving elements in B), and hence, the algorithm may not use them to act as pivots. The solution lies in a technically more involved strategy: first, ensure by random sampling that there are not too many triangles in G. Subsequently, we show that an element that results in an almost balanced partition and is not involved in too many triangles can be used as a pivot. Proving this fact lies at the heart of our technical contribution.

Implications for Ulam-k-Median Problem. Our result on robust sorting has interesting implications for another well-studied problem, namely the Ulam-k-Median problem. Given a positive integer d, let $[d]$ denote the set $\{1, \ldots, d\}$ and Π^d the set of all permutations over $[d]$. We can define a metric Δ over Π^d as follows: given σ_1, σ_2, $\Delta(\sigma_1, \sigma_2) \coloneqq d - |\mathsf{lcs}(\sigma_1, \sigma_2)|$, where $\mathsf{lcs}(\sigma_1, \sigma_2)$ denotes the length of the longest common subsequence between σ_1 and σ_2. The metric (Π^d, Δ) is also popularly known as the Ulam metric. An instance of this problem is specified by a subset $S \subseteq \Pi^d$ and an integer k. The goal is to find a subset F of k permutations in Π^d (which may not lie in S) such that the objective value

$$\sum_{\sigma \in S} \min_{\pi \in F} \Delta(\pi, \sigma)$$

is minimized. In other words, it is the k-median problem where the metric is given by Δ on Π^d. When $k = 1$, there is a simple 2-approximation algorithm (which works for any metric): output the best point in the input set S.

Breaking the approximation guarantee of 2 for specific metrics, such as the Ulam metric, was open for a long time. Recently, a sequence of works [5, 6] breaks this 2-factor barrier for the Ulam k-median problem. The algorithm is a fixed parameter tractable (FPT) algorithm (with respect to the parameter k) that gives an approximation guarantee of $(2 - \delta)$ for a very small but fixed constant $\delta > 0$. The running time of this algorithm is $(k \log nd)^{O(k)} nd^3$, which can be written as $f(k) \cdot poly(nd)$ and hence is FPT time. Note that the running time is super-linear in n and cubic in d. This contrasts with several FPT approximation algorithms for the k-median/means problems in the literature [16, 13, 3, 2, 11] that have linear running time in the input size. For example, the running time of the FPT $(1 + \varepsilon)$-approximation algorithms for the Euclidean k-median/means problems in [16, 13, 3, 2] is linear in nd, where d is the dimension. Similarly, the running time for the FPT constant factor approximation algorithms [11] in the metric setting is linear in n. Thus, we ask:

> *Is it possible to break the barrier of 2-approximation for the Ulam k-median problem using an FPT algorithm with a* ***linear*** *running time in the input size, i.e., nd ?*

We show that the answer is affirmative using two crucial ideas, one of which relies on the robust sorting problem. We build on the ideas of [6] for designing a $(2 - \delta)$-approximation for the 1-median problem. They show that for any input set S, there exist five permutations

$\sigma_1, ..., \sigma_5$ in S from which we can find a permutation $\tilde{\sigma}$ that has a small distance with respect to the optimal σ^*. To obtain linear in n dependence on the running time, we show that a constant size randomly sampled subset of input permutations contains these five permutations with high probability. To obtain linear dependence on d, given guesses $\sigma_1, ..., \sigma_5$ for the five permutations, we can assign each pair of symbols (a, b) a direction based on the *majority* vote among these permutations. Now, it turns out that a good solution to the corresponding robust sorting instance is close to the optimal solution σ^*.

2 Preliminaries and Problem Definition

We discuss two problems – *Robust Sorting and Ulam median.* We start with the robust sorting.

Robust Sorting. We are given a set of n elements V. There is an (unknown) total order π over V. However, we have imperfect access to the total order π. We are given an implicit directed graph $G = (V, E)$, where we have a directed edge between every pair of elements in V (such graphs are often denoted *tournaments*). In the ideal (zero error) scenario, the edge set E would correspond to π, i.e., for every distinct pair $u, v \in V$, we have the directed arc (u, v) iff u comes before v in π. However, we allow the graph G to represent π in an imperfect manner as formalized below:

▶ **Definition 1** (Imperfect representation)**.** *We say that a tournament $G = (V, E)$ is B-imperfect with respect to a total order π on V, where B is a subset of V, if for every pair of distinct $u, v \in V \setminus B$, G has the arc (u, v) iff u comes before v in π. For an integer b, we say that G is b-imperfect w.r.t. π if there is a subset B of V, $|B| \leq b$ such that G is B-imperfect with respect to π.*

In other words, if G is B-imperfect w.r.t. π, then the arcs with both end-points outside B represent π correctly, but we cannot give any guarantee for arcs incident with vertices in B. We shall often ignore the reference to π if it is clear from the context. For edge $(u, v) \in E$, we will sometimes use the notation $u < v$ or $v > u$. Note that the relations $<, >$ are not transitive, except when $|B| = 0$. Similarly, we will sometimes say that u is *lesser than* (resp. *greater than*) v if $(u, v) \in E$ (resp. $(v, u) \in E$).

The Robust Sort problem is as follows: given a set of points V with an implicit total order π, and a query access to the edges in a tournament G, output an ordering $\widetilde{\pi}$ on V which maximizes $\mathsf{lcs}(\pi, \widetilde{\pi})$ while using as few queries to G as possible. Here $\mathsf{LCS}(\pi, \widetilde{\pi})$ denotes the longest common subsequence between π and $\widetilde{\pi}$, and $\mathsf{lcs}(\pi, \widetilde{\pi})$ denotes the length of $\mathsf{LCS}(\pi, \widetilde{\pi})$. Observe that when G is 0-imperfect w.r.t. π, one can obtain the total order π with $O(n \log n)$ queries to G.

The Robust Sort problem can be reduced to the well-studied Feedback Vertex Set problem on tournaments (FVST). A feedback vertex set in a directed graph is a subset of vertices whose removal makes the graph acyclic. The Feedback Vertex Set in Tournaments (FVST) problem is to find a feedback vertex set of the smallest size in a given Tournament graph. If we are allowed to query all the edges in G, then an α-approximation algorithm for FVST can be utilized to obtain an $(\alpha + 1)$-approximation algorithm for Robust Sort as follows: *find an α-approximate feedback vertex set B' and then find a topological ordering π_1 on $V \setminus B'$. Output the concatenation $\widetilde{\pi}$ of any arbitrary ordering on B' with π_1.* Clearly, $\mathsf{LCS}(\pi, \widetilde{\pi}) \geq |\pi_1| - |B| = |V| - |B'| - |B| \geq |V| - (\alpha + 1)|B|$. Hence, using exponential time, we can find an optimal feedback vertex set and therefore a 2-approximate solution to

Robust Sort. However, since feedback vertex set on tournaments is NP-hard to approximate better than a factor of 2, and a polynomial time 2-approximation algorithm is known [17], 3-approximation is the best we can get from such a direct reduction to FVST. Indeed, consider a simple example where $V = \{v_1, v_2, v_3, v_4\}$, $\pi = (v_2, v_1, v_3, v_4), B = \{v_2\}$. Consider a B-imperfect tournament G, such that for all unordered pairs (v_i, v_j) with $i < j$, there is an arc from v_i to v_j in G, except the direction of the arc is reversed for (v_2, v_3) and (v_2, v_4). In this example, a 2-approximation algorithm might return $\{v_3, v_4\}$ as the feedback vertex set. Hence, $\pi_1 = (v_1, v_2)$ is the only topological ordering on the remaining vertices, and we might return $\widetilde{\pi} = (v_4, v_3, v_1, v_2)$. Here, $\mathsf{LCS}(\pi, \widetilde{\pi}) = 1 = |V| - 3|B|$.

Although the algorithm of Lokshtanov et al. [17] is also inspired by quick-sort, it queries $\Omega(n^2)$ edges and runs in $O(n^{12})$ time. In our setting, we are constrained by near-linear running time and, hence, can only check a small subset of triangles for consistency. However, this causes several other issues: a bad pivot can masquerade as a good one, and a good pivot may not get a chance to partition the elements into almost equal halves. The fact that we are on a very tight budget in terms of the number of queries and errors makes the algorithm and analysis quite subtle.

Ulam median. The Ulam k-median problem is simply the k-median problem defined over the Ulam metric (Π^d, Δ) – *given a set $S \subset \Pi^d$ of n elements and a positive integer k, find a set C of k elements (called centers) such that the objective function $Obj(S, C) := \sum_{s \in S} \min_{c \in C} \Delta(s, c)$ is minimized.* Here, Π^d is the set of all permutations of $[d] := \{1, 2, 3, ..., d\}$, which can also be seen as all d-length strings with distinct symbols from set $\{1, 2, ..., d\}$. The distance function Δ is defined as $\Delta(x, y) := d - \mathsf{lcs}(x, y)$, where $\mathsf{lcs}(x, y)$ denotes the length of the *longest common subsequence* of permutations $x, y \in \Pi^d$. There is a trivial 2-approximation algorithm and it has been a long open challenge to obtain a better approximation. Breaking this barrier of 2 was recently achieved by Chakraborty et al. [6] who gave a parameterized algorithm (with parameter k) with a running time of $(k \log nd)^{O(k)} nd^3$ and approximation guarantee of $(2 - \delta)$ for a small constant δ. Our goal was to achieve the same using a parameterized algorithm with running time $\tilde{O}(f(k) \cdot nd)$.

3 Our Results

Our first result gives an algorithm for Robust Sort that queries $\widetilde{O}(n)$ (here $\widetilde{O}$ hides poly-logarithmic factors) edges in G and achieves nearly the same guarantees as that obtained by an efficient algorithm querying all the edges in G followed by a reduction to FVST.

▶ **Theorem 2.** *Consider an instance of Robust Sort given by a tournament graph $G = (V, E)$, where $|V| = n$, and a parameter $\varepsilon > 0$. Suppose G is b-imperfect w.r.t. an ordering π on V. Then, there is an efficient algorithm that queries $O\left(\frac{n \log^3 n}{\varepsilon^2}\right)$ edges in G and outputs a sequence $\widetilde{\pi}$ such that expectation of $\mathsf{lcs}(\pi, \widetilde{\pi})$ is at least $n - (3 + \varepsilon)b$. Further, the algorithm does not require knowledge of the quantity b and has running time $O\left(\frac{n \log^3 n}{\varepsilon^2}\right)$ (assuming each query takes constant time).*

It is worth emphasizing that the parameter b may not be constant. In fact, much of the technical difficulty lies in handling cases when b may be sub-linear. Following is our main result for the Ulam k-median problem.

▶ **Theorem 3.** *There is a randomized algorithm for the Ulam k-median problem that runs in time $\tilde{O}((2k)^k \cdot nd)$ and returns a center set C with $Obj(S, C) \leq (2 - \delta) \cdot OPT$ with probability at least 0.99, where δ is a small but fixed constant.*

3.1 Our Techniques

We now give an overview of our techniques for the robust sorting and the Ulam-k-median problems.

3.1.1 Robust Sorting

Consider an instance of `ROBUST-SORT` given by a graph G on a vertex set V of size n. Assume G is B-imperfect for some $B \subseteq V$, $|B| = b$. We shall call the elements of B "bad" elements and the rest "good" elements. Observe that every directed triangle in G must contain at least one bad element. One potential idea is to keep finding and removing directed triangles in G, until G is acyclic (recall that a tournament is acyclis if and only if it has no triangles). Suppose we remove t triangles. As each removed triangle has at least one bad element, $t \leq b$ and the number of remaining bad elements is at most $b - t$. Hence, the number of remaining good elements is at most $n - b - 2t - (b - t) = n - 2b - t \geq n - 3b$. Thus, we produce an ordering $\widetilde{\pi}$ with $\mathsf{lcs}(\pi, \widetilde{\pi}) \geq n - 3b$. However, this approach uses a quadratic number of queries on G. On the other hand, there are many sorting algorithms that use $O(n \log n)$ queries in the classical setting, i.e., the 0-imperfect setting. Direct generalizations of such sorting algorithms fail to get the required guarantees on the lcs between the output $\widetilde{\pi}$ and π.

For example, consider Merge Sort. Here, bad elements can result in the merge procedure to output permutations with a large distance with respect to π. Indeed, suppose we partition the input set V into equal sized V_1, V_2 and recursively get good orderings $\widetilde{\pi}_1$ and $\widetilde{\pi}_2$ on them, respectively. Further, assume that all the good elements in V_1 appear before those in V_2. However, it is possible that the first element in $\widetilde{\pi}_1$ happens to be a bad element x, and x turns out to be larger than all the elements in V_2. In such a setting, the merge procedure would place all the elements in V_2 before x, which is clearly an undesirable outcome.

We now show that using randomized quick sort directly would also lead to an undesirable outcome. In randomized quick sort, one chooses a pivot at random and recursively solves the problem on the elements smaller than the pivot and those larger than the pivot, placing the pivot between the two recursive outputs. For the feedback arc set in tournaments (FAST) problem, where the goal is to minimize the number of inverted pairs, this algorithm was shown to return a 3-approximation [1] in expectation. However, this simple algorithm does not work for our problem. Indeed, let $b \ll n$ and consider a random input where a subset B of size b is chosen at random as the set of bad elements, and a random permutation is selected among the set of good elements. Edges where both endpoints are good respect the permutation, and each edge incident on a bad element is oriented randomly. Now, the random quick-sort algorithm chooses a bad pivot x with probability b/n – assume this event happens. Let V_1, V_2 be the elements less than and greater than x, respectively (with respect to the graph G). Since x is a bad pivot, our assumption implies that V_1, V_2 form a roughly random partition of V into equal-sized subsets. Thus, if $X \coloneqq V \setminus B$ denote the set of good elements and X_1, X_2 be the left and the right half of X respectively, then roughly $|X_1|/2$ elements of X_1 will end up in V_2 and similarly for X_2. Note that $|X_1| = |X_2| = \frac{n-b}{2}$. Hence, roughly $(n - b)/4$ elements will be *wrongly* placed by the pivot x. Let $f(n, b)$ denote the distance between the output produced by this algorithm on an instance of size n containing b bad elements and the ordering π. Then, we have shown that $f(n, b)$ is at least $\Omega(n - b)$

with a high probability if we choose a bad pivot. Thus, we get the approximate recurrence (note that both V_1, V_2 will have roughly $b/2$ bad elements):

$$f(n,b) \approx \frac{b}{n} \cdot \Omega(n-b) + 2f(n/2, b/2)$$

It is easy to verify that this results in $f(n,b) = \Omega(b \log n)$, whereas we desire an output for which this quantity is $O(b)$.

The above analysis indicates that we cannot afford to have "arbitrarily" bad elements as pivots. The following idea takes care of examples as above: when we choose a pivot p, check (logarithmic number of times) if p forms a triangle with randomly sampled pair (x, y). If a triangle is found, we can remove p, x, y (and hence, at least one bad element gets removed) and try a new pivot; otherwise, it is guaranteed that p would be involved in very few triangles (note that in the example above, a triangle would be found with constant probability if the pivot is a bad one). However, this simple idea also does not work. The reason is as follows: (i) randomly chosen good elements, which would ideally act as good pivots, are involved in a lot of triangles and, hence, would not be selected as pivots, and (ii) there could be bad elements, which are not involved in too many triangles, and hence, would sneak in as a pivot. It may seem that the latter scenario is not undesirable – if a bad element participates in a few triangles, then it perhaps acts like a good element and can be used to partition the input set. Unfortunately the quantity $f(n,b)$ as defined above turns out to be large for this algorithm. This happens for the following subtle reason: say there are b bad elements, and suppose each of the good elements participates in many triangles. Then, with probability $(1 - b/n)$ (which can be considered to be close to 1), the algorithm picks a good element as a random pivot and finds a triangle containing it. This would reduce the problem size by only three elements, i.e., the recursive problem has almost the same size. On the other hand, with probability b/n, which may be small, the algorithm partitions using a bad element as a pivot. As outlined above, when we pick a bad element as a pivot, the resulting partition may have many good elements on the wrong side of the partition. Thus, in the overall calculation, the large misalignment created due to these low probability events overwhelms the expected value of $f(n,b)$. In other words, one obtains a recurrence of the form:

$$f(n,b) \approx \frac{b}{n} \cdot m_b + f(n-3, b-1)$$

where $f(n-3, b-1)$ refers to the sub-problem obtained when a triangle is found, and m_b refers to the misalignment caused by a typical bad element. Since a bad element participates in few triangles, it is possible that $m_b \ll \Omega(n-b)$ (comparing with the previous recurrence above), but still, this can be high enough to lead to a recurrence where $f(n,b)$ is not $O(b)$. The issue arises because the problem size does not shrink sufficiently to balance the expected misalignment. One way to handle this is to consider separately the case where there are too many triangles in the tournament. So, before picking a potential pivot and testing its goodness, in a *pre-pivoting step*, we check randomly chosen triples for triangles and remove them in case they are found. This ensures that at the time of pivot selection, there is a reasonable chance the problem size shrinks without too much increase in the expected misalignment. Our algorithm and analysis basically work by balancing these quantities in a carefully devised inductive argument. One of the key technical insights is the following: given two orderings σ_1 and σ_2 of a partition V_1 and V_2 of V respectively, we define the notion of *concatenation loss* – this captures the extra misalignment (with respect to the implicit ordering) created by concatenating σ_1 and σ_2. Our key technical result shows that if such a partitioning is created by a pivot involved in a few triangles, then the corresponding concatenation loss of the orderings on V_1 and V_2 output by the algorithm is small.

3.1.2 Ulam-k-Median

There is a trivial and well-known 2-approximation algorithm for the Ulam 1-median problem – *output the best permutation from the input.* To break the barrier of 2, one must consider a stronger version of the triangle inequality that holds specifically for the Ulam metric. Let $\sigma_1, ..., \sigma_n$ be the permutations in the input and let σ^* denote the optimal 1-median. Let I_{σ_i} denote the subset of symbols that are not in $\mathsf{LCS}(\sigma_i, \sigma^*)$, i.e., the misaligned symbols in σ_i and σ^*. So, $\Delta(\sigma_i, \sigma^*) = |I_{\sigma_i}|$. The following (stronger version) of the triangle inequality holds for the Ulam metric: $\Delta(\sigma_i, \sigma_j) \leq |I_{\sigma_i}| + |I_{\sigma_j}| - |I_{\sigma_i} \cap I_{\sigma_j}| = \Delta(\sigma_i, \sigma^*) + \Delta(\sigma_j, \sigma^*) - |I_{\sigma_i} \cap I_{\sigma_j}|$. Chakraborty et al. [6] exploit this inequality to break the 2-approximation barrier. Even though the technical details are intricate, at a very high level, the key idea in [6] is to show that either one of the input permutations is a good center or there are five permutations $\sigma_1, ..., \sigma_5$ such that $\forall i, j \in \{1, 2, 3, 4, 5\}$ with $i \neq j$, $|I_{\sigma_i} \cap I_{\sigma_j}|$ is small, i.e., the number of *common* misaligned symbols is small.

We strengthen this result as follows – we show that either there are a significant number of permutations that will be good centers, or there are a significant number of permutations with a small number of pair-wise common misaligned symbols. This allows us to argue that an η-sized (for some **constant** η) random subset of permutations is sufficient to find a good center, so we do not need to consider $\binom{n}{5}$ possibilities for finding a good center. Chakraborty et al. [6] gave a similar sampling lemma. However, they required a random sample of size $O(\log n)$. Using this result would lead to an additional $(\log n)^{O(k)}$ factor in the running time for the k-median problem. One of our key contributions is showing that a constant-sized sample suffices. The number of common misaligned symbols in the five permutations being small implies that for most pairs (a, b) of symbols, their relative order in σ^* matches that in at least 3 out of 5 permutations $\sigma_1, ..., \sigma_5$. This is used to find a permutation with a good agreement with σ^* and hence is a good center. [6] uses an $O(d^3)$ procedure to find such a center, whereas we improve this to $\tilde{O}(d)$ by using our robust sorting algorithm. For the Ulam k-median problem, we use our sampling-based algorithm, `ULAM1`, within the D-sampling framework of [13] to obtain an $\tilde{O}(f(k) \cdot nd)$-time algorithm. Here is the summary of the key ideas: Let $S_1, ..., S_k$ be the dataset partition that denotes the optimal k clustering, and let $\sigma_1^*, ..., \sigma_k^*$ denote the optimal centers, respectively. Let us try to use `ULAM1` to find good centers for each of $S_1, ..., S_k$. We would need η uniformly sampled points each from $S_1, ..., S_k$. The issue is that the optimal clustering $S_1, ..., S_k$ is not known. If the clusters were balanced, i.e., $|S_1| \approx |S_2| \approx ... \approx |S_k|$, then uniformly sampling $O(\eta k)$ points from S and then considering all possible partitions of these points would give the required uniform samples $X_1, ..., X_k$ from each of the optimal clusters. We can then use `ULAM1(`X_i`)` to find good center candidates for S_i. In the general case, where the clusters may not be balanced, we use the D-sampling technique[2] to boost the probability of sampling from small-sized optimal clusters, which may get ignored when sampling uniformly at random from S.

The details of our algorithm for the Ulam median problem are given in the full version of the paper available on ArXiv (`https://arxiv.org/abs/2502.07653`).

3.2 Related Work

We have already seen the connection between robust sorting and the Feedback Vertex Set in Tournaments (FVST) problem [17, 18]. Another problem related to the FVST problem is the Feedback Arc Set in Tournaments (FAST) problem [1, 15], where the goal is to find an ordering of the nodes of a given tournament such that the number of edges going backward

[2] D-sampling is sampling proportional to the distance of an element from the closest previously chosen center.

(an edge is said to go backward if it is directed from a node that comes later to a node that comes earlier as per the ordering) is minimized. This is the restricted variant of the maximum acyclic subgraph problem [8], where the goal is to find the maximum subset of edges that induces a directed acyclic graph (DAG) in a directed graph. The FAST problem may be seen as robust sorting under adversarial corruption of edges rather than adversarial corruption of nodes, as in our formulation. The FVST and FAST problems are known to the NP-hard. A 2-approximation, which is tight under UGC, is known [17] for the FVST problem, and a PTAS is known [15] for the FAST problem.

Several works have been done on sorting in the presence of a noisy comparison operator, also called noisy sorting. Feige et al. [7] consider a noise model in which the comparison operator gives the correct answer independently with probability at least $(1/2+\gamma)$ each time a query is made on an element pair. This can be regarded as *noisy sorting with resampling* since we can get the correct answer for a pair by repeatedly querying the operator on the same pair. So, each time a comparison needs to be made for a pair, by repeatedly querying $O(\log n)$ times, one can obtain a $O(n \log^2 n)$ algorithm. A better algorithm with $O(n \log n)$ queries can be obtained [7, 14]. In more recent works [19, 12], a deeper investigation was made into the constant, which is dependent on the bias γ, hidden in the $O(n \log n)$ sorting algorithm of [7] for noisy sorting **with** resampling. A more interesting noise model was considered by Braverman and Mossel [4], where the ordering algorithm cannot repeat a comparison query.[3] This is called the *noisy sorting* ***without*** *resampling (NSWR)* problem. The NSWR problem can also be seen as a stochastic version of the Feedback Arc Set on Tournament (FAST) problem – the tournament is generated using the noisy comparator (with respect to some total order π), and the goal is to find an ordering of the vertices such that the number of edges going backward is minimized. [4] gave an algorithm that runs in time $n^{O((\beta+1)\gamma^{-4})}$ and outputs an optimal ordering with probability at least $(1-n^{-\beta})$. Another objective function in this setting is to minimize the maximum dislocation and total dislocation of elements, where the dislocation of an element is the difference between its ranks in π and the output ordering. Optimal bounds for this have been achieved in a recent sequence of works [10, 9].

The key references [5, 6] for the Ulam median problem have already been discussed. The detailed discussions on the Ulam median problem can be found in the full version of the paper.

4 Algorithm for Robust Sort

In this section, we present our algorithm for Robust Sort (Algorithm 1). The algorithm discards a subset V' of V and returns an ordering π' on the remaining elements $V \setminus V'$. We can obtain an ordering $\widetilde{\pi}$ on V by appending an arbitrary ordering on V' to π'. Since the size of V' shall turn out to be small, $\mathsf{lcs}(\pi, \pi')$ will be close to $\mathsf{lcs}(\pi, \widetilde{\pi})$.

Algorithm 1 is based on a divide-and-conquer strategy similar to quick-sort. Consider a recursive sub-problem given by a subset S of V. We choose a parameter $k = O\left(\frac{\log^2 N}{\epsilon^2}\right)$ (line 1.4), where $N = |V|$, and then proceeds in four steps:

1. **Testing random triplets for triangles**: In this step, we randomly sample $O(k \log N)$ triplets of elements from S uniformly at random (line 1.6). For each such triplet (x, y, z), we check if it forms a triangle, i.e., if G contains the arcs $(x, y), (y, z)$ and (z, x). If so, we discard (elements in) the triangle (line 1.9) and go back to the beginning of the procedure. Note that checking whether a triplet is a triangle requires three queries to G.

[3] This can also be modeled by the constraint that the errors are independent but *persistent*, i.e., if a comparison is repeated, then you get the same answer.

Algorithm 1 ROBUST-SORT(S).

1.1 **Input:** A subset S of the original set T of elements
1.2 **Output:** An ordering π' on a subset of S
1.3 **Global:** $N = |T|, \epsilon$
1.4 $k \leftarrow \left(\frac{10000 \log N}{\epsilon}\right)^2$ /* A parameter deciding the amount of testing to be done */
1.5 **while** $|S| > 0$ **do**
1.6 **for** $i = 1, 2, \ldots 72k \cdot \log N$ **do**
1.7 Sample three elements x, y, z independently and uniformly at random from S
1.8 **if** x, y, z *form a triangle* **then**
1.9 $S \leftarrow S \setminus \{x, y, z\}$
1.10 Go to line 1.5
1.11 **for** $r = 1, 2, \ldots 36 \cdot \log N$ **do**
1.12 Sample an element $p \in S$ uniformly at random /* Choose a random pivot */
1.13 $L_1, R_1 \leftarrow \phi, \phi$
1.14 $k' \leftarrow 10^5 \cdot \log N$
1.15 **for** $j = 1, 2, \ldots k'$ **do**
1.16 Sample an element $s \in S \setminus \{p\}$ uniformly at random
1.17 **if** $s < p$ **then**
1.18 $L_1 \leftarrow L_1 \cup \{s\}$
1.19 **else**
1.20 $R_1 \leftarrow R_1 \cup \{s\}$
1.21 **if** $\min(|L_1|, |R_1|) \leq \frac{k'}{5} + \frac{k'}{40}$ **then**
1.22 Go to the next iteration of the line 1.11 /* Imbalanced partition */
1.23 **for** $j = 1, 2, \ldots k$ **do**
1.24 Sample two elements x, y independently and uniformly at random from $S \setminus \{p\}$
1.25 **if** x, y, p *form a triangle* **then**
1.26 $S \leftarrow S \setminus \{x, y, p\}$
1.27 Go to line 1.5
1.28 $L, R \leftarrow \phi, \phi$
1.29 **for** $s \in S$ **do**
1.30 **if** $s < p$ **then**
1.31 $L \leftarrow L \cup \{s\}$
1.32 **else**
1.33 $R \leftarrow R \cup \{s\}$
1.34 **return** $\pi' =$ (ROBUST-SORT(L), p, ROBUST-SORT(R)) /* Recursively sort L and R */
1.35 **return** $\{\}$ /* Too many bad elements, return an empty sequence */
1.36 **return** $\{\}$ /* No more elements left, return an empty sequence */

2. **Finding a balanced pivot**: In this step, it tries to find a good pivot – this pivot finding step is repeated $O(\log N)$ times (line 1.11) to ensure that one of these succeeds with high probability). We first select a randomly chosen element p of S as the pivot (line 1.12). Then we check whether it is a *balanced* pivot as follows: we sample (with replacement) $k' := O(\log N)$ elements from S and partition these k' elements with respect to p (lines 1.15–1.20). Let L_1 and R_1 denote the partitioning of these k' elements with respect to p. In line 1.21, we check if both these sets are of size at least $k'/5 + k'/40$. If not, we repeat the process of finding a pivot (line 1.22). Otherwise, we continue to the next step. Note that if this pivot finding iteration fails for $36 \log N$ trials, then we discard all elements of S (line 1.35).
3. **Testing for triangles involving the pivot**: In this step, we test if the balanced pivot p chosen in the previous step forms a triangle with randomly chosen pairs of elements from S. More formally, we repeat the following process k times (line 1.23): sample two elements x and y (with replacement) from $S \setminus \{p\}$. If (x, y, p) forms a triangle, we discard these three points from S (line 1.26) and go back to the beginning of the procedure (line 1.5).
4. **Recursively solving the subproblems**: Assuming the pivot p found in the second step above does not yield a triangle in the third step above, we partition the entire set S with respect to p into two sets L and R respectively (lines 1.28–1.33). We recursively call the algorithm on L and R and output the concatenation of the orderings returned by the recursive calls (line 1.34).

5 Analysis

In this section, we analyse Algorithm 1. Let the input instance be given by a tournament $G = (V, E)$, and assume that G is B-imperfect with respect to a total order π on V, for some subset B of V. Let $|V| = N$. Since G is B-imperfect, we know that G induced on $V \setminus B$ is a DAG. We shall refer to the elements in $V \setminus B$ as *good* elements. Let π_g be the restriction of π on the good elements. We begin with some key definitions:

▶ **Definition 4** (Balanced partition). *Let p be an element of a subset $S \subseteq V$. A partition $L \cup R$ of $S \setminus \{p\}$ with respect to the pivot p is said to be balanced if* $\min(|L|, |R|) \geq \frac{|S|}{5}$.

▶ **Definition 5** (Support and loss of a sequence). *Let σ be a sequence on a subset of elements in V. The* support *of the sequence σ, denoted $\mathsf{supp}(\sigma)$, is defined as $\mathsf{LCS}(\sigma, \pi_g)$, i.e., the longest subsequence of good elements in σ which appear in the same order as in π. If there are multiple choices for $\mathsf{supp}(\sigma)$, we choose the one that is lexicographically smallest with respect to the indices in σ. Let $\mathsf{loss}(\sigma)$, the loss of σ, be defined as the number of elements in σ that are not in $\mathsf{supp}(\sigma)$.*

▶ **Definition 6** (Concatenation Loss). *Consider two sequences σ_1 and σ_2. Let σ be the sequence formed by the concatenation of σ_1 and σ_2. The* concatenation loss *of sequences σ_1 and σ_2, denoted $\mathsf{concatloss}(\sigma_1, \sigma_2)$, is defined as $|\mathsf{supp}(\sigma_1)| + |\mathsf{supp}(\sigma_2)| - |\mathsf{supp}(\sigma)| = \mathsf{loss}(\sigma) - \mathsf{loss}(\sigma_1) - \mathsf{loss}(\sigma_2)$.*

Fix a subset $S \subseteq V$, and consider the recursive call `ROBUST-SORT`(S) corresponding to S. Observe that the set S changes during the run of this recursive call – we shall use the index t to denote an iteration of the **while** loop in line 1.5 and use S_t to denote the set S during this iteration. Note that each iteration of the while loop either ends with the removal of a

triangle or with recursive calls to smaller subproblems (L and R). We define several failure events with respect to an iteration of the while loop and show that these events happen with low probability:

- **Triangle Detection Test Failure**: This event, denoted $\mathcal{F}_1$, happens when $G[S_t]$ has at least $\frac{|S_t|^3}{24k}$ triangles, but the **for** loop in lines 1.6–1.10 does not find any triangle.
- **Balanced Partition Failure**: This failure event, denoted $\mathcal{F}_2$, occurs when the procedure executes lines 1.28–1.33 and then makes recursive calls to L and R, but the partition $L \cup R$ is not a balanced partition of $S_t \setminus \{p\}$.
- **Density Test Failure**: This event, denoted $\mathcal{F}_3$, happens when $|S_t|$ has at most $|S_t|/3$ bad elements, but we execute line 1.22 in each of the $36 \log N$ iterations of the **for** loop (line 1.11). In other words, we are not able to find a good partition of the sampled k' elements in any of the $36 \log N$ iterations of this **for** loop.

We now show that with high probability, none of the failure events happen.

▶ **Lemma 7.** *Let N be large enough (greater than some constant). Then* $\Pr[\mathcal{F}_i] \leq \frac{1}{N^3} \forall i \in \{1, 2, 3\}$.

Proof. We first consider $\mathcal{F}_1$. Suppose S_t has at least $\frac{|S_t|^3}{24k}$ triangles. The probability that an iteration of the **for** loop in lines 1.6–1.10 does not find a triangle is at most $1 - \frac{1}{24k}$. Thus, the probability that none of the $72k \log N$ iterations find a triangle is at most $\left(1 - \frac{1}{24k}\right)^{72k \log N} \leq \frac{1}{N^3}$.

We now analyze the event $\mathcal{F}_2$. Suppose we choose a pivot p in line 1.12 which is not balanced w.r.t. S_t. We need to argue that we shall execute line 1.22 (which happens when $\min(|L_1|, |R_1|) \leq \frac{k'}{5} + \frac{k'}{40}$) with high probability. Let L and R denote the set of elements in $S_t \setminus \{p\}$ that are lesser than and greater than p, respectively. Assume $|L| < |S_t|/5$ (the other case is similar). Thus, the expected size of L_1 is at most $\frac{k'}{5}$. It follows from standard Chernoff bounds that the probability that $|L_1| > \frac{k'}{5} + \frac{k'}{40}$ is at most: $e^{\frac{-\left(\frac{1}{40}\right)^2 \cdot \frac{k'}{5}}{3}} \leq e^{-4 \log N} \leq \frac{1}{N^4}$. Since we can execute line 1.12 at most $36 \log N$ times during a particular iteration t of the **while loop**, it follows by union bound that $\Pr[\mathcal{F}_2] \leq \frac{1}{N^3}$. (for large enough N).

Finally, we consider the event $\mathcal{F}_3$. Suppose S_t has at most $|S_t|/3$ bad elements, i.e., there are at least $2|S_t|/3$ good elements. Consider the middle (in the ordering π_g) $|S_t|/6$ good elements of S_t. Any such element has at least $\frac{1}{2} \cdot (2|S_t|/3 - |S_t|/6) = |S_t|/4$ good elements on either side. It follows that if the pivot p is chosen among these $|S_t|/6$ elements, then the expected size of the sets L_1, R_1 defined in lines 1.13–1.20 is at least $\frac{k'}{4} = (\frac{k'}{5} + \frac{k'}{40}) + \frac{k'}{40}$. Thus, for such a pivot, using the Chernoff bound, the probability of executing line 1.22 (that is, $\min(|L_1|, |R_1|) \leq \frac{k'}{5} + \frac{k'}{40}$) is at most $1/4$. Hence, the probability that we do not execute line 1.35 in a particular iteration of the **for** loop in line 1.11 is at least $\frac{1}{6} \cdot \frac{3}{4} = \frac{1}{8}$. Thus, $\Pr[\mathcal{F}_3]$ = the probability that we execute line 1.35 in each of the iterations of this **for** loop is at most $\left(1 - \frac{1}{8}\right)^{36 \log N} \leq \frac{1}{N^3}$. ◀

Induction Hypothesis. Given a subset S of V, let $\pi'(S)$ be the sequence generated by `ROBUST-SORT`(S). Note that $\pi'(S)$ is a random sequence. Given integers n and b, let $\mathcal{S}(n, b)$ denote all subsets S of size n of V which have b bad elements. Let $L(n, b)$ denote the maximum expected loss of the sequence output by Algorithm 1 when run on a subset $S \in \mathcal{S}(n, b)$. More formally,

$$L(n, b) \coloneqq \max_{S \in \mathcal{S}(n,b)} \mathbf{E}[\mathsf{loss}(\pi'(S))].$$

We now state the induction hypothesis that shall prove the main result Theorem 2:

$$L(n,b) \leq 3b + cb\log n, \quad \forall n \in \mathbb{N}, 0 \leq b \leq n \leq N, \text{ where } c = \frac{\varepsilon}{\log N} \tag{1}$$

We prove the above result by induction on n. When $n = 1$, the result follows trivially. Now assume it is true for all $L(n', b')$, where $n' \leq n-1$. Now, we would like to show it for $L(n,b)$ for some given $b \leq n$. Fix a subset $S \in \mathcal{S}(n,b)$. We need to show that

$$\mathbf{E}[\mathsf{loss}(\pi'(S))] \leq 3b + cb\log n. \tag{2}$$

We first check an easy case.

▷ Claim 8. Suppose S has at least $\frac{n^3}{24k}$ triangles, where k is as stated in line 1.4, then $\mathbf{E}[\mathsf{loss}(\pi'(S))] \leq 3b + cb\log n$.

Proof. We know by Lemma 7 that the failure event $\mathcal{F}_1$ happens with probability at most $1/N^3$. Thus, with probability at least $1 - 1/N^3$, the algorithm shall make a recursive call on a subset S' obtained from S by removing a triangle (x,y,z) found in line 1.9 (notice that, whenever we remove a triangle $\{x,y,z\}$, we start from scratch after setting $S \leftarrow S \setminus \{x,y,z\}$). This triangle must contain at least 1 bad element. We can assume that it has exactly one bad element (otherwise, the induction hypothesis applied on S' only gets stronger because the r.h.s. of Equation (1) is monotonically increasing with b). Thus, $\mathbf{E}[\mathsf{loss}(\pi'(S))] \leq \left(1 - \frac{1}{N^3}\right)(L(n-3, b-1) + 3) + \frac{n}{N^3}$, where the second term on the r.h.s. appears because the loss can never exceed n. Applying induction hypothesis on $L(n-3, b-1)$, we see that the above is at most

$$3(b-1) + c(b-1)\log(n-3) + 3 + \frac{n}{N^3} \leq 3b + cb\log n - c\log(n-3) + \frac{n}{N^3} \leq 3b + cb\log n,$$

where the second inequality uses $c = \frac{\epsilon}{\log N}$ and assumes that N exceeds a large enough constant (for a fixed ϵ) ◁

Henceforth, assume that S has at most $\frac{n^3}{24k}$ triangles. Further, if the algorithm finds a triangle in line 1.9, then the same argument as above applies. Thus, we can condition on the event that no triangles are found in the **for** loop in lines 1.6–1.10. We can also assume that $b \leq n/3$, otherwise the condition (2) follows trivially. Assume that $\mathcal{F}_3$ does not occur (we shall account for this improbable event later). In that case, we know that we shall find a balanced pivot p of S during one of the iterations of the **for** loop in line 1.11, and such that the condition in line 1.21 will not hold. Let L and R be as defined in lines 1.28–1.33. We now give a crucial definition.

▶ **Definition 9.** *Let $p \in S$ be a pivot and L and R be the partition of $S \setminus \{p\}$ consisting of elements lesser than and greater than p, respectively. Define M_p as: $M_p := \max_{X,Y} \mathsf{concatloss}(X,Y)$, where X varies over all sequences of distinct elements of L and Y varies over all sequences of distinct elements of R.*

Recall that, we say that a is *less than* b, if $(a,b) \in E$. Using the definition of loss, it is easy to verify that $M_p = 0$ if p is a good element. For an element $p \in S$, let q_p denote the probability that no triangles are found in the **for** loop in line 1.23 conditioned on the fact that p is chosen in the pivot in line 1.12 and the execution reaches the **for** loop in line 1.23. We have the following key lemma:

▶ **Lemma 10.** *For any $p \in S$, $\left(1 - \frac{2T_p}{n^2}\right)^k \leq q_p \leq \left(1 - \frac{M_p^2}{4n^2}\right)^k$, where T_p denotes the number of triangles in S containing p.*

Proof. We first prove the lower bound on q_p. Consider an element $p \in S$. The probability that during an iteration of the **for** loop in line 1.23, we pick the (unordered) triplet $\{x, y, p\}$ for a triangle (x, y, p) is $\frac{2}{n^2}$. Since there are T_p triangles containing p, the probability that we pick a triangle containing p is at most $\frac{2T_p}{n^2}$. Thus, the probability that we do not pick any such triangle during the k iterations of the **for** loop in line 1.23 is at least $\left(1 - \frac{2T_p}{n^2}\right)^k$.

Now, we prove the upper bound. Assume $M_p > 0$; otherwise, the claim is trivial. Let L and R denote the partition of $S \setminus \{p\}$ with respect to the pivot p. Let X and Y be sequences over subsets of L and R respectively such that $M_p = \mathsf{concatloss}(X, Y)$. Recall that $\mathsf{supp}(X)$ (or $\mathsf{supp}(Y)$) is the longest common subsequence between X (or Y) and the optimal sequence $\pi_g^\star$ on the good elements. In particular, $\mathsf{supp}(X)$ and $\mathsf{supp}(Y)$ consist of sorted good elements. Let X_1 denote the suffix of length $M_p/2$ of $\mathsf{supp}(X)$, and Y_1 denote the prefix of length $M_p/2$

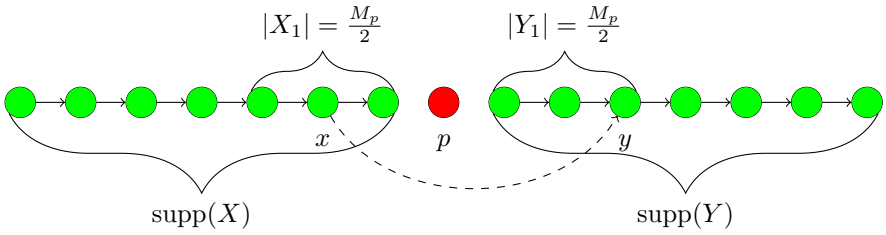

Figure 1 An arc from x to y is a contradiction to the concatenation loss being M_p. Thus, the tuple (x, p, y) forms a triangle $x \to p \to y \to x$.

of Y. We claim that for any $x \in X_1, y \in Y_1, (x, y, p)$ forms a triangle. Suppose not. Then, x is less than y. But then, consider concatenating the prefix of $\mathsf{supp}(X)$ till x and the suffix of Y from y (see Figure 1). These sequence of good elements will be sorted and hence a subsequence of $\pi_g^\star$. But the length of this sequence is larger than $\mathsf{supp}(X) + \mathsf{supp}(Y) - M_p$. This implies that $\mathsf{concatloss}(X, Y) < \mathsf{supp}(X) + \mathsf{supp}(Y) - (\mathsf{supp}(X) + \mathsf{supp}(Y) - M_p) = M_p$, a contradiction. Thus, we see that there are at least $\frac{M_p^2}{4}$ triangles involving p and elements of $S \setminus \{p\}$. So, any particular iteration of the **for** loop in line 1.23 shall find such a triangle with probability at least $\frac{M_p^2}{4n^2}$. This proves the desired upper bound on q_p. ◀

Let $\mathsf{Bal}(S)$ denote the set of all balanced pivots of S, that is, the elements p of S for which the partition $L \cup R$ of $S \setminus \{p\}$ is balanced ($\min(|L|, |R|) \geq \frac{n}{5}$). For a pivot p, let h_p denote the probability that, when p is chosen as pivot (in line 1.12), we get a balanced partition (L_1, R_1) of the sampled elements, that is, $\min(L_1, R_1) > \frac{k'}{5} + \frac{k'}{40}$ (and hence, line 1.22 is not executed). Recall that $b \leq \frac{n}{3}$. Hence, assuming $\mathcal{F}_3$ does not occur, there must be some pivot, among the (up to) $36 \log N$ sampled pivots, that passes the balanced partition test (i.e., line 1.22 is not executed for this pivot). Let γ_p denote the probability that this pivot is p. It is easy to see that $\gamma_p = \frac{h_p}{\sum_{u \in S} h_u}$.

▶ **Lemma 11.** *Let* $\mathsf{Mid}(S)$ *denote the set of middle (in the ordering* π_g*)* $|S|/6$ *good elements of* S*. We have:*

(i) $\sum_{p \in S \setminus \mathsf{Bal}(S)} \gamma_p = \Pr[\mathcal{F}_2] \leq \frac{1}{N^3}$.

(ii) *For all* $p \in \mathsf{Mid}(S)$, $\gamma_p \geq \frac{1}{2n}$.

(iii) *For all* $p \in B$, $\gamma_p \leq \frac{12}{n}$.

Proof. The first statement follows from the definition of $\mathcal{F}_2$ and Lemma 7. Now, notice that for $p \in \mathsf{Mid}(S), h_p \geq \frac{3}{4}$ (this was argued in the proof of Lemma 7 when bounding the probability of F_3), and $\sum_{p \in S} h_p \leq n$. Hence, $\gamma_p \geq \frac{1}{2n}$ for all $p \in \mathsf{Mid}(S)$. For the third statement, note that $\sum_{p \in S} h_p \geq \sum_{p \in \mathsf{Mid}(S)} h_p \geq \frac{n}{6} \cdot \frac{1}{2} = \frac{n}{12}$. Also, $h_p \leq 1$ for all $b \in B$. Hence, $\gamma_p \leq \frac{12}{n}$ for all $p \in B$. ◀

Now, once a pivot p passes the partition test, there are two possibilities:

(a) With probability q_p, we partition S into L and R with respect to p and recursively call `ROBUST-SORT`(L) and `ROBUST-SORT`(R). Let the output of these recursive calls be sequences σ_L and σ_R, respectively (recall that the output of `ROBUST-SORT`(L) or `ROBUST-SORT`(R) can be a sequence on a subset of L or R respectively). Let σ denote the concatenated sequence (σ_L, p, σ_R). If p is a good element, then $M_p = 0$, and hence $\mathsf{loss}(\sigma) = \mathsf{loss}(\sigma_1) + \mathsf{loss}(\sigma_2)$. If p is a bad element, $\mathsf{loss}(\sigma)$ is same as the loss of the sequence (σ_1, σ_2). The latter quantity, by the definition of M_p, is at most $\mathsf{loss}(\sigma_1) + \mathsf{loss}(\sigma_2) + M_p$. Let L and R have b_L and b_R bad elements, and let their sizes be n_L and n_R. For, $p \in S \setminus \mathsf{Bal}(S), \mathbf{E}[\mathsf{loss}(\sigma)] \leq n$. For $p \in \mathsf{Bal}(S)$,

$$\begin{aligned}\mathbf{E}[\mathsf{loss}(\sigma)] &\leq \mathbf{E}[\mathsf{loss}(\sigma_L)] + \mathbf{E}[\mathsf{loss}(\sigma_R)] + M_p \\ &\leq 3b_L + cb_L \log n_L + 3b_R + cb_R \log n_R + M_p \\ &\leq 3b + cb \log \max(n_L, n_R) + M_p \\ &\leq 3b + cb \log n - cb \log(5/4) + M_p,\end{aligned}$$

where the second inequality follows from the induction hypothesis, and the last inequality uses $\max(n_L, n_R) \leq \frac{4n}{5}$.

(b) With probability $(1 - q_p)$, a triangle is found in lines 1.23–1.27. In this case, we effectively recurse on a subset S' of S, where S' has $n - 3$ elements and at most $b - 1$ bad elements. In this case, the induction hypothesis implies that the expected loss of the sequence returned by `ROBUST-SORT`(S) is, at most

$$\mathbf{E}[\pi'(S')] + 3 \leq 3(b-1) + c(b-1)\log(n-3) + 3 \leq 3b + cb \log n$$

where the expectation is over the choice of S' and the `ROBUST-SORT`(S') procedure.

Putting everything together, we see that (conditioned on $\mathcal{F}_3$ not happening), $\mathbf{E}[\mathsf{loss}(\pi'(S))|\mathcal{F}_3$ does not happen] is at most (recall that B is the set of bad elements):

$$\begin{aligned}&3b + cb\log n - cb\log(5/4) \sum_{p \in \mathsf{Bal}(S) \setminus B} q_p\gamma_p + \sum_{p \in S \cap B} q_p\gamma_p \cdot M_p + \sum_{p \in S \setminus \mathsf{Bal}(S)} \gamma_p n \\ \leq &3b + cb\log n - cb\log(5/4) \sum_{p \in \mathsf{Mid}(S)} \frac{1}{2n} q_p + \sum_{p \in S \cap B} \frac{12}{n} q_p \cdot M_p + \frac{n}{N^3} \\ \leq &\ 3b + cb\log n - cb\log(5/4) \sum_{p \in \mathsf{Mid}(S)} \frac{1}{2n}\left(1 - \frac{2T_p}{n^2}\right)^k + \frac{12}{n} \sum_{p \in S \cap B} M_p \left(1 - \frac{M_p^2}{4n^2}\right)^k + \frac{1}{N^2}\end{aligned} \tag{3}$$

where the first inequality uses the observation that $\mathsf{Mid}(S) \subseteq \mathsf{Bal}(S) \setminus B$, and utilizes the bounds on γ_p derived in Lemma 11. The second inequality uses the bounds on q_p derived in Lemma 10. Now, observe that, the expression $x(1 - x^2)^k$ over $x > 0$, is maximized at $x = \frac{1}{\sqrt{2k+1}}$, and hence,

$$M_p\left(1-\frac{M_p^2}{4n^2}\right)^k = 2n\frac{M_p}{2n}\left(1-\left(\frac{M_p}{2n}\right)^2\right)^k \le \frac{2n}{\sqrt{2k+1}}\left(1-\frac{1}{2k+1}\right)^k \le \frac{2n}{\sqrt{2k+1}}$$

Substituting this in (3), we see that $\mathbf{E}[\mathsf{loss}(\pi'(S))|\mathcal{F}_3\text{does not happen}]$ is at most:

$$3b + cb\log n - cb\log(5/4)\sum_{p\in\mathsf{Mid}(S)}\frac{1}{2n}\left(1-\frac{2T_p}{n^2}\right)^k + \frac{12|S\cap B|}{n}\frac{2n}{\sqrt{2k+1}} + \frac{1}{N^2}$$

$$\le 3b + cb\log n - cb\log(5/4)\cdot\frac{1}{2n}\cdot\sum_{p\in\mathsf{Mid}(S)}\left(1-\frac{2T_p}{n^2}\right)^k + \frac{12\sqrt{2}b}{\sqrt{k}} + \frac{1}{N^2}$$

After Claim 8, we had assumed that $\sup_p T_p \le \frac{n^3}{24k}$. Therefore, there can be at most $n/12$ elements for which $T_p > \frac{n^2}{2k}$. It follows that there are at least $n/6 - n/12 = n/12$ elements p of $\mathsf{Mid}(S)$ for which $T_p \le \frac{n^2}{2k}$. Therefore, the expression above is, at most:

$$3b + cb\log n - cb\log(5/4)\cdot\frac{1}{2n}\cdot\frac{n}{12}\left(1-\frac{1}{k}\right)^k + \frac{12\sqrt{2}b}{\sqrt{k}} + \frac{1}{N^2}$$

$$\le 3b + cb\log n - \frac{c}{96}\log(5/4)b + \frac{12\sqrt{2}b}{\sqrt{k}} + \frac{1}{N^2}$$

$$\le 3b + cb\log n - \frac{b\epsilon}{\log N}\left(\frac{\log(5/4)}{96} - \frac{12\sqrt{2}}{10000}\right) + \frac{1}{N^2}$$

$$\le 3b + cb\log n - \frac{1}{N^2}$$

where the first inequality uses the fact that $k \ge 2$ and hence, $(1-1/k)^k \ge 1/4$, the second one uses the fact that $c = \frac{\epsilon}{\log N}$, and the last inequality assumes that N exceeds a large enough constant (for a fixed ϵ), and $b \ge 1$ (for $b = 0$, $L(n,b) = 0$ trivially holds). Now,

$$\begin{aligned}\mathbf{E}[\mathsf{loss}(\pi'(S))] =& \Pr[\mathcal{F}_3 \text{ does not happen}]\cdot\mathbf{E}[\mathsf{loss}(\pi'(S))|\mathcal{F}_3 \text{ does not happen}]\\ &+ \Pr[\mathcal{F}_3 \text{ happens}]\cdot\mathbf{E}[\mathsf{loss}(\pi'(S))|\mathcal{F}_3 \text{ happens}]\\ &\le 1\left(3b + cb\log n - \frac{1}{N^2}\right) + \frac{1}{N^3}\cdot n\\ &\le 3b + cb\log n\end{aligned}$$

Finally, using $c = \frac{\epsilon}{\log N}$, we see that the expected loss is at most $(3+\epsilon)b$. This proves the statement about the quality of the output permutation in Theorem 2. It remains to calculate the expected number of queries by the algorithm.

▶ **Lemma 12.** *Conditioned on the failure event $\mathcal{F}_2$ not happening during any of the recursive calls of* `ROBUST-SORT`*, the number of queries made by the algorithm is $O\left(\frac{N\log^3 N}{\epsilon^2}\right)$. The same bound holds for the running time of the algorithm.*

Proof. It is easy to verify that each iteration of the **while loop** (1.5) takes $O(\frac{\log^3 N}{\epsilon^2})$ time, and either finds a triangle or divides the problem into two smaller subproblems (in this case taking an additional $O(|S|)$ time). Since $\mathcal{F}_2$ does not happen, we have that the subproblem sizes are at most $O(\frac{4}{5}|S|)$. Hence, the time complexity recursion (which subsumes the query complexity) is:

$$T(n) = \max\left(T(n-3), \max_{n/5\le n_1\le 4n/5}(T(n_1) + T(n-1-n_1) + O(n))\right) + O\left(\frac{\log^3 N}{\epsilon^2}\right).$$

It is easy to inductively prove that $T(n) = O\left(n\log n + n\frac{\log^3 N}{\epsilon^2}\right)$. ◀

The probability of $\mathcal{F}_2$ happening during an iteration of the **while loop** (1.5) is at most $1/N^3$ (Lemma 7). Since there are up to N iterations of the while loop overall, using union bound, the probability that $\mathcal{F}_2$ never happens is at least $1 - 1/N^2$. Also, the worst-case running time (and the query complexity) of ROBUST-SORT is $O(N^2)$. Hence, the expected running time (also the expected query complexity) of the algorithm is at most:

$$\left(1 - 1/N^2\right) \cdot O\left(\frac{N \log^3 N}{\epsilon^2}\right) + \frac{1}{N^2} \cdot N^2 = O\left(\frac{N \log^3 N}{\epsilon^2}\right).$$

Note that, when proving $L(n, b) \leq 3b + cb \log n$, where $c = \frac{\epsilon}{\log N}$, we assumed that N is large enough at certain places. It can be easily verified that $N = \Omega(\frac{1}{\epsilon^{2/3}})$ suffices in all these places. To extend the small loss guarantee to smaller N, we can simply tweak our algorithm to run the triangle removal algorithm if $N = O(\frac{1}{\epsilon^{2/3}})$, resulting in a loss of at most $3b$, with a run-time of $O(\frac{1}{\epsilon^2})$. This completes the proof of Theorem 2.

6 Conclusion and open problems

We give an algorithm for robust sorting. For a total order π on a set V of elements, we are given an imperfect comparison operator that behaves in the following manner: There is an (unknown) subset $B \subset V$ such that for every pair $a, b \in V \setminus B$, the comparator is consistent with the total order π, but if even one of a, b is from B, then the comparator can give an arbitrary (but deterministic) response. We give an algorithm that outputs an ordering π' such that the order of at least $|V| - (3 + \epsilon) \cdot |B|$ elements are consistent with π (i.e., $LCS(\pi, \pi') \geq |V| - (3+\epsilon)|B|$). Our algorithm runs in time $\tilde{O}(|V|/\epsilon^2)$. This means that it does not compare every pair of elements. Even though $3 + \epsilon$ was sufficient for the Ulam median application, it is natural to ask whether the factor $3 + \epsilon$ can be improved. More concretely, we identify the following open problem:

> *What is the best constant α, such that there exists a randomized algorithm that asks $\tilde{O}(|V|)$ queries in expectation, and outputs an ordering π' with $\mathbf{E}[\mathit{LCS}(\pi, \pi')] \geq |V| - \alpha|B|$? In particular, as our algorithm provides a $(3+\varepsilon)$-approximation, it would be interesting to see if $\alpha \leq 3$.*

The above open question restricts the number of queries to $\tilde{O}(|V|)$. However, the problem remains interesting even if there is no bound on the number of queries. In particular, we ask the following open question:

> *What is the best constant β, such that there exists a polynomial time randomized algorithm with no bound on the number of queries, that outputs an ordering π' with $\mathbf{E}[\mathit{LCS}(\pi, \pi')] \geq |V| - \beta|B|$? In particular, as a reduction to FVST yields a 3-approximation to Robust Sort, it would be interesting to see if $\beta < 3$.*

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5), November 2008. doi:10.1145/1411509.1411513.

2 Anup Bhattacharya, Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. On Sampling Based Algorithms for k-Means. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, volume 182 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2020.13.

3 Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Faster algorithms for the constrained k-means problem. *Theory of Computing Systems*, 62(1):93–115, 2018. doi:10.1007/s00224-017-9820-7.

4 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 268–276, USA, 2008. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1347082.1347112.

5 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. *Approximating the Median under the Ulam Metric*, pages 761–775. SIAM, 2021. doi:10.1137/1.9781611976465.48.

6 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Clustering Permutations: New Techniques with Streaming Applications. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:24, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2023.31.

7 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

8 Alan Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999. doi:10.1007/s004930050052.

9 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal Sorting with Persistent Comparison Errors. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2019.49.

10 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal dislocation with persistent errors in subquadratic time. *Theory of Computing Systems*, 64(3):508–521, 2020. doi:10.1007/s00224-019-09957-5.

11 Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. FPT Approximation for Constrained Metric k-Median/Means. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2020.14.

12 Yuzhou Gu and Yinzhan Xu. Optimal bounds for noisy sorting. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1502–1515, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585131.

13 Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. A simple D^2-sampling based PTAS for k-means and other clustering problems. *Algorithmica*, 70(1):22–46, 2014. doi:10.1007/s00453-013-9833-9.

14 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *SODA '07*, pages 881–890, USA, 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283478.

15 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 95–103, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250806.

16 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, February 2010. doi:10.1145/1667053.1667054.

17 Daniel Lokshtanov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-approximating feedback vertex set in tournaments. *ACM Trans. Algorithms*, 17(2), April 2021. doi:10.1145/3446969.

18 Matthias Mnich, Virginia Vassilevska Williams, and László A. Végh. A 7/3-Approximation for Feedback Vertex Sets in Tournaments. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International*

Proceedings in Informatics (LIPIcs), pages 67:1–67:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2016.67.

19 Ziao Wang, Nadim Ghaddar, and Lele Wang. Noisy sorting capacity. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 2541–2546, 2022. doi:10.1109/ISIT50566.2022.9834370.