New Sublinear Algorithms and Lower Bounds for LIS Estimation

Ilan Newman
University of Haifa, Israel

Nithin Varma
University of Haifa, Israel

Abstract

Estimating the length of the longest increasing subsequence (LIS) in an array is a problem of fundamental importance. Despite the significance of the LIS estimation problem and the amount of attention it has received, there are important aspects of the problem that are not yet fully understood. There are no better lower bounds for LIS estimation than the obvious bounds implied by testing monotonicity (for adaptive or nonadaptive algorithms). In this paper, we give the first nontrivial lower bound on the complexity of LIS estimation, and also provide novel algorithms that complement our lower bound.

Specifically, we show that for every $\epsilon \in (0, 1)$, every nonadaptive algorithm that outputs an estimate of the LIS length in an array of length n to within an additive error of ϵn has to make $\log^{\Omega(\log(1/\epsilon))} n$ queries. Next, we design nonadaptive LIS estimation algorithms whose complexity decreases as the number of distinct values, r, in the array decreases. We first present a simple algorithm that makes $\tilde{O}(r/\epsilon^3)$ queries and approximates the LIS length with an additive error bounded by ϵn. This algorithm has better complexity than the best previously known adaptive algorithm (Saks and Seshadhri; 2017) for the same problem when $r \ll \operatorname{poly}\log(n)$. We use our algorithm to construct a nonadaptive algorithm with query complexity $\tilde{O}(\sqrt{r} \cdot \operatorname{poly}(1/\lambda))$ that, when the LIS is of length at least λn, outputs a multiplicative $\Omega(\lambda)$-approximation to the LIS length. Our algorithm improves upon the state of the art nonadaptive LIS estimation algorithm (Rubinstein, Seddighin, Song, and Sun; 2019) in terms of the approximation guarantee.

Finally, we present a $O(\log n)$-query nonadaptive erasure-resilient tester for monotonicity. Our result implies that lower bounds on erasure-resilient testing of monotonicity does not give good lower bounds for LIS estimation. It also implies that nonadaptive tolerant testing is strictly harder than nonadaptive erasure-resilient testing for the natural property of monotonicity.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases longest increasing subsequence, monotonicity, distance estimation, sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.100

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2010.05805`

Funding *Ilan Newman*: Supported by The Israel Science Foundation, grant number 497/17.
Nithin Varma: Supported by The Israel Science Foundation, grant number 497/17 and by the PBC Fellowship for Postdoctoral Fellows by the Israeli Council of Higher Education.

1 Introduction

Estimating the length of the longest increasing subsequence (LIS) in an array is a problem of fundamental importance. For arrays of length n, one can solve this problem exactly in time $O(n \log n)$ using dynamic programming [9] or patience sorting [2]. Approximating the length of the LIS has also been well-studied, and there are several sublinear-time

algorithms [15, 1, 19, 18] for this task. In the approximation task, for a real-valued array A of size n, the goal is to estimate the length of the LIS within an additive error (of ϵn) or multiplicative error. An additive ϵn-approximation algorithm for this problem can also be used to estimate, with the same approximation guarantee, the Hamming distance of A to the closest sorted array[1] (a.k.a. distance to monotonicity).

Early sublinear-time algorithms for LIS estimation [15, 1] provided multiplicative $(2+o(1))$-approximation for the distance to monotonicity, and thereby, additive $\frac{n}{2}$-approximation to the length of the LIS. Saks and Seshadhri [19] made a major improvement to the state of the art, and presented an algorithm that approximates the LIS length to within an additive error of ϵn for arbitrary $\epsilon \in (0,1)$. All these algorithms have query complexity polylogarithmic[2] in n for constant ϵ. Subsequently, Rubinstein, Seddighin, Song, and Sun [18] presented a nonadaptive algorithm that computes a multiplicative $\Omega(\lambda^3)$-approximation to the LIS length, with query complexity $\tilde{O}(\sqrt{n}\cdot\text{poly}(1/\lambda))$, where λ is the ratio of the LIS length to n. In a very recent work (independent and parallel to ours), Mitzenmacher and Seddighin [11] developed a sublinear algorithm for LIS estimation with query complexity $\tilde{O}(n^{1-\Omega(\epsilon)}\cdot\text{poly}(1/\lambda))$ that obtains an approximation ratio of $\Omega(\lambda^\epsilon)$ for arbitrary $\epsilon \in (0,1)$.

Despite the significance of the LIS estimation problem and the amount of attention it has received, there are important aspects of the problem that are not yet fully understood. There is no better lower bound on the query complexity of LIS estimation, for adaptive or nonadaptive algorithms, other than the obvious bound of $\Omega(\log n)$ implied by monotonicity testing [8]. Another issue is to investigate whether the input length n is the right parameter to express the complexity of LIS estimation algorithms. In other words, it is unknown whether there are other input parameters that capture the fine-grained complexity of LIS estimation by making use of the underlying combinatorics of the problem.

In this paper we address both these issues. We prove the first nontrivial lower bound on the query complexity of nonadaptive algorithms for additive error LIS estimation. We also design nonadaptive LIS estimation algorithms whose query complexity is parameterized in terms of the number of distinct values in the input array.

Lower Bound for LIS Estimation. We show that there is no nonadaptive algorithm that approximates the LIS length to arbitrary additive error and has query complexity polylogarithmic in n. Specifically, for arbitrary constant $\epsilon \in (0,1)$, every nonadaptive LIS estimation algorithm that has an additive error bounded by ϵn has to make $\log^{\Omega(\log(1/\epsilon))} n$ queries. Interestingly, our lower bound construction uses ideas from the lower bound [4] on the query complexity of 1-sided error nonadaptive testers for the property of $(k,\ldots,2,1)$-freeness. This is the first lower bound that improves upon the obvious lower bound of $\Omega(\log n)$.

One general approach for proving lower bounds on the complexity of LIS estimation was proposed by Dixit, Raskhodnikova, Thakurta, and Varma [6], who showed that lower bounds for erasure-resilient testing of monotonicity provides lower bounds for estimating the distance to monotonicity up to an additive error. We prove that this method cannot provide a nontrivial lower bound for LIS estimation, by showing a $O(\log n)$-query *nonadaptive* algorithm for erasure-resilient monotonicity testing.

---

[1] It is necessary and sufficient to modify the values that do not belong to an LIS to make the array sorted.

[2] The query complexity of the algorithm by Saks and Seshadhri [19] depends on the approximation parameter ϵ as $O((1/\epsilon)^{1/\epsilon})$ and hence is within aforementioned bound only if ϵ is constant. In particular, the query complexity ceases to be sublinear as as soon as ϵ is $O(1/\log(n))$.

Sublinear Algorithms for LIS Estimation. Our starting point here is to understand the dependence of the query complexity of LIS estimation on the range size of an input array. This is a major direction of study for the simpler problem of monotonicity testing, since the only tight lower bound [8] holds for exponential range. Recently, Pallavoor, Raskhodnikova, and Varma [14], and Belovs [3], gave efficient algorithms for monotonicity testing whose query complexity beats the above lower bound when range size is small. There were no explicit results on LIS estimation for limited range size before our work.[3] In this paper, we give efficient *nonadaptive* LIS estimation algorithms whose complexity is parameterized by r, the number of distinct values in the array, which is always at most the range size. Our algorithms improve upon the state of the art algorithms in both complexity and approximation guarantee when the range is small.

We first show a $\tilde{O}(r/\epsilon^3)$-query nonadaptive algorithm for LIS estimation, of additive error ϵn, for arbitrarily small ϵ. In particular, when the LIS length is a constant fraction of n, our algorithm can be used to get a multiplicative $(1 \pm \epsilon)$-approximation for the LIS length. We add that our algorithm is the only sublinear nonadaptive algorithm giving this approximation guarantee when $r = o(n)$. Furthermore, when $r = o(\log^k n)$ (for an appropriate power k), our algorithm outperforms the adaptive algorithm of Saks and Seshadhri [19], not only in terms of the dependence of query complexity on the input size n, but also in terms of its dependence on the approximation parameter ϵ. Hence, our algorithm bridges the gap between the known $\Omega(\text{poly} \log n)$-query algorithm for the general range and the $O(1)$-query algorithm for the Boolean range.

An additional main result of this paper is a $\tilde{O}(\sqrt{r})$-query nonadaptive algorithm that gives a multiplicative approximation to the LIS length even when the LIS is relatively small. Namely, the algorithm makes $\tilde{O}(\sqrt{r} \cdot \text{poly}(1/\lambda))$ queries and outputs a multiplicative $\Omega(\lambda)$-aproximation to the LIS length, where λ denotes the LIS length normalized by the input length. This is an improvement over the algorithm by Rubinstein, Seddighin, Song, and Sun [18], which makes $\tilde{O}(\sqrt{n} \cdot \text{poly}(1/\lambda))$ nonadaptive queries and outputs a multiplicative $\Omega(\lambda^3)$-approximation to the LIS length. Our algorithm improves upon [18] in terms of approximation guarantee (even in the general case of $r = n$) as well as query complexity (when $r \ll n$), and further, works for any value of r. Finally, the query complexity of our algorithm is always better than that of the recent LIS estimation algorithm by Mitzenmacher and Seddighin [11] that outputs a multiplicative $\Omega(\lambda^\epsilon)$-approximation to the LIS length for arbitrary $\epsilon \in (0, 1)$.[4]

Separating Distance Estimation from Erasure-Resilient Testing. As mentioned before, a general method for proving lower bounds on distance estimation (or tolerant testing [15]) is via proving lower bounds on erasure-resilient testing [6].

Our nonadaptive erasure-resilient tester for monotonicity with complexity $O(\log n)$ and our lower bound on the query complexity of nonadaptive algorithms for LIS estimation imply that nonadaptive tolerant testing is strictly harder than nonadaptive erasure-resilient testing for the natural property of monotonicity, thereby making progress towards solving an open question raised by Raskhodnikova, Ron-Zewi, and Varma [16].

[3] For the case of Boolean arrays, Berman, Raskhodnikova, and Yaroslavtsev [5] showed that one can approximate the LIS length to within an additive error of ϵn by making $O(1/\epsilon^2)$ queries.

[4] We point out that the LIS estimation algorithm of Mitzenmacher and Seddighin [11] uses the algorithm of Rubinstein et al. [18] as a subroutine. By using our algorithm instead, the query complexity of the algorithm of Mitzenmacher and Seddighin [11] can be improved.

1.1 Discussion of Results and Overview of Techniques

In this section, we state our results more formally, and provide an overview of the techniques used to prove them. We use ideas from [18], [12] and [4]. Given a real-valued array A of length n, an LIS in A is the longest nondecreasing sequence of values in A. In other words, the LIS is a largest cardinality set $\mathcal{L}$ of indices such that for $u, v \in \mathcal{L}$, we have $u < v$ if and only if $A[u] \leq A[v]$. We abuse notation and also use the term LIS to denote $|\mathcal{L}|$ when this is clear from the context. A real-valued array of length n can be equivalently viewed as a function from $[n]$ to the reals. Adopting this view, we use the term *monotone array* to refer to a sorted array. Throughout, we denote by r, the number of (or a guaranteed upper bound on) distinct values in the array. That is, $r = |R|$ for $R = \{A[i] : \ i \in [n]\}$. Thus, for the unrestricted case it is assumed that $r = n$.

1.1.1 Lower bound on the query complexity of nonadaptive LIS estimation algorithms

Our first result proves that there is no nonadaptive algorithm that approximates the LIS length in an array of length n to within an additive error of ϵn and has query complexity polylogarithmic in n, for arbitrary constant $\epsilon \in (0, 1)$.

▶ **Theorem 1.1.** *For every $\epsilon \in (0, 1)$, every nonadaptive algorithm that on an array A of length n, outputs an additive ϵn-approximation to the length of the LIS in A, has to make $\log^{\Omega(\log(1/\epsilon))} n$ queries.*

We note that this is the first lower bound on LIS estimation that is not directly implied by the lower bound for testing monotonicity [8].

To prove our lower bound, we construct two distributions with different LIS lengths such that every deterministic nonadaptive algorithm distinguishing the distributions with probability at least 2/3, has query complexity $\log^{\omega(1)}(n)$. More specifically, for every natural number h, we construct distributions $\mathcal{D}_0^{(h)}$ and $\mathcal{D}_1^{(h)}$ that are supported on inputs whose LIS lengths differ by $\exp(-h)$. We then prove that every deterministic nonadaptive algorithm that takes input from the union of the supports of $\mathcal{D}_0^{(h)}$ and $\mathcal{D}_1^{(h)}$, and aims to correctly identify the distribution from which the input is taken, either fails for most inputs or makes $\Omega(\log^h n)$ queries. Interestingly, our lower bound construction uses ideas from the lower bound of Ben-Eliezer, Canonne, Letzter, and Waingarten [4] on the query complexity of 1-sided error nonadaptive testers for the property of $(k, \ldots, 2, 1)$-freeness, where an array A of length n is $(k, \ldots, 2, 1)$-free if there are no k indices $i_1 < i_2 < \cdots < i_k$ such that $A[i_1] > A[i_2] > \cdots > A[i_k]$.

Using reductions from erasure-resilient testing

As mentioned before, a general method for proving lower bounds on distance estimation is via proving lower bounds on erasure-resilient testing [6].

▶ **Definition 1.2** (Erasure-resilient monotonicity tester). *Given $\epsilon, \alpha \in (0, 1)$ and a real-valued array A containing at most α-fraction of* erased *values*[5]*, the goal of an α-erasure-resilient ϵ-tester for monotonicity is to determine whether A can be completed to a monotone array or whether every completion of A has Hamming distance at least ϵn to monotonicity.*

[5] Erasures are made adversarially before the tester makes its queries and the tester is unaware of the location of the erasures. A tester that queries the value at an erased location is returned a special symbol $\perp$.

Dixit, Raskhodnikova, Thakurta and Varma [6] observed that the complexity of erasure-resilient (ER) testing a property, falls in between the complexity of standard testing the property and estimating the distance to that property (with additive error). Hence, a lower bound on the complexity of ER testing monotonicity implies the same lower bound for estimating the LIS length up to an additive error. The only previously known ER tester for monotonicity [6] is adaptive and has query complexity $O(\log(n)/\epsilon)$. Hence, a nontrivial lower bound for (adaptive) LIS estimation cannot be obtained this way.

We present a *nonadaptive* ER tester that makes $O(\log n)$ queries and works for all fraction of erasures. This makes the results on ER testing monotonicity tight, and also shows that one cannot obtain a lower bound for LIS estimation via ER testing.

▶ **Theorem 1.3.** *Let $\epsilon, \alpha \in (0,1)$ such that $\alpha + \epsilon < 1$. There exists a nonadaptive α-erasure-resilient ϵ-tester for monotonicity that makes $O\left(\frac{\log n}{\epsilon^2} + \frac{1}{\epsilon^3}\right)$ queries for n-length arrays.*

The ER testers designed by Dixit et al. [6] for various properties, are all either adaptive, or obtained by repeating a (standard) tester that makes independent and uniformly distributed queries. Our tester is different, and is in this sense, the first nontrivial nonadaptive ER tester for a natural property. Consider an array A of length n with at most α fraction of erasures, where $\alpha \in [0,1)$. Our tester samples an index $s \in [n]$ uniformly at random and does a randomized binary search for s on the array as if it were monotone. It queries the array values on these indices, and looks for violations to monotonicity on the search path to s. In case there are no erasures, this is a good strategy to detect a violation to monotonicity [7]. However, when values at a constant fraction of indices are erased, it could be the case that most of the values on the search path are erased. We show that a slightly modified version of this tester can be used for testing monotonicity. Specifically, our tester, in addition to querying the values along the binary search path, also queries the indices in a small constant-sized interval around the search point s. To analyze this modified tester, we rely on a combinatorial lemma by Newman, Rabinovich, Rajendraprasad, and Sohler [12]. A nonerased index $x \in [n]$ is γ-deserted for $\gamma \in (0,1)$ if there exists an interval $I \subseteq [n]$ such that $x \in I$ and at most γ fraction of the values in I are nonerased. Roughly speaking, the lemma implies that the fraction of γ-deserted indices in A is proportional to $\gamma \cdot \alpha$. Using this, we are able to argue that, with high probability, the index s that we sample as the search point is not γ-deserted (for an appropriate choice of γ) and that it forms a violation with enough other nonerased indices, so as to ensure a high probability of success.

1.1.2 Parameterized and nonadaptive algorithms for LIS estimation

We present efficient nonadaptive LIS estimation algorithms. The novelty is that we parameterize the complexity of LIS estimation algorithms in terms of the number of distinct values r in an array. We first show an LIS estimation algorithm with query complexity $\tilde{O}(r)$.

▶ **Theorem 1.4.** *There exists a nonadaptive algorithm that, given a real-valued array A of length n containing at most r distinct values, and a parameter $\epsilon \in (0,1)$, makes $\tilde{O}(r/\epsilon^3)$ queries and outputs, with probability at least $2/3$, an estimate for the LIS size that is accurate to within additive ϵn-error. Moreover, the queries of the algorithm are uniformly and independently distributed, and the algorithm runs in time $\tilde{O}(r/\epsilon^3)$.*

We mention that the approximation guarantee provided by the algorithm is quite strong and holds even for non-constant error parameter ϵ. It matches the approximation guarantee of the adaptive LIS estimation algorithm by Saks and Seshadhri [19], which makes polylog(n)

queries when $\epsilon = \theta(1)$. In particular, when the length of the LIS $\mathcal{L}$ is a constant fraction of n, our algorithm can be used to get a multiplicative $(1 \pm \epsilon)$-approximation for the LIS length. We add that our algorithm is the only nonadaptive sublinear algorithm giving this approximation guarantee as soon as $r = o(n)$. Furthermore, when $r = o(\log^k n)$ (for an appropriate power k), our algorithm performs much better than the algorithm of Saks and Seshadhri, not only in terms of the dependence of query complexity on the input size n, but also in terms of the dependence on the approximation parameter ϵ.

The high level idea of the algorithm is that it is enough to restrict attention to special subarrays that are *dense* and *nice*, as elaborated in the following. Let $\mathcal{L}$ be a fixed unknown LIS in the input array. A subarray is dense if a constant fraction of its indices belong to $\mathcal{L}$, and it is nice if the LIS takes at most one distinct value in the subarray. Informally, we divide the array into $O(r/\epsilon)$ subarrays. This will make most dense subarrays nice with respect to $\mathcal{L}$ (for an appropriate density parameter). We then sample $O(\log r)$ indices in each subarray to find the values that are "typical" in each subarray.

Our goal is to output as an estimate for $|\mathcal{L}|$, the size of $\mathcal{L}'$, which is the restriction of $\mathcal{L}$ to such typical values. This will naturally be an underestimate, but with a small additive error. To estimate the size of $\mathcal{L}'$, we consider all possible increasing sequences of the typical values, taking one value from each subarray. Since most subarrays are nice, the size of $\mathcal{L}'$ restricted to such a sequence of values is quite close to $|\mathcal{L}'|$. Finally, for a given nice subarray A_i, the largest subsequence in A_i that takes one given value v can be easily determined – this is just the distance to the array taking the value v everywhere.

Next, we use the above $\tilde{O}(r)$-query algorithm to obtain a nonadaptive LIS estimation algorithm with query complexity $\tilde{O}(\sqrt{r})$.

▶ **Theorem 1.5.** *There exists a nonadaptive algorithm that, given a real-valued array A of length n containing at most r distinct values and $|\mathrm{LIS}(A)| = \lambda \cdot n$, makes $\tilde{O}(\sqrt{r} \cdot \mathrm{poly}(1/\lambda))$ queries and outputs, with probability at least $2/3$, an estimate est such that $\Omega(\lambda \cdot |\mathrm{LIS}(A)|) \leq \mathsf{est} \leq O(|\mathrm{LIS}(A)|)$. Moreover, the algorithm runs in time $\tilde{O}(r \cdot \mathrm{poly}(1/\lambda))$.*

As mentioned before, this result is an improvement over a recent LIS estimation algorithm by Rubinstein, Seddighin, Song and Sun [18], in terms of the approximation guarantee. Additionally, the complexity of our algorithm improves as the number of distinct values in the input array decreases. Another advantage of our algorithm (also that of [18]) is that its query complexity is sublinear, even if λ is sub-constant.

Our $\tilde{O}(\sqrt{r})$-query nonadaptive algorithm is somewhat complicated. In the following, we present a high-level description of the algorithm. We denote the input array by A and use $\mathcal{L}$ to denote a fixed LIS in A. We visualize the array values as points in an $r \times n$ grid G_n. The vertical axis of G_n represents the range R of the array and is labeled with the at most r distinct array values in increasing order and the horizontal axis is labeled with the indices in $[n]$. We refer to an index-value pair in the grid as a point. The grid has n points, to which we do not have direct access. We use queries to the array to form some *approximate* picture of the location of points in this grid, and use it to estimate $|\mathcal{L}|$.

The main idea is to build, in $\tilde{O}(\sqrt{r})$ queries, a data structure that possesses enough information to compute an estimate est, which is a lower bound on $|\mathcal{L}|$ and is also a reasonably good approximation. Roughly speaking, the first step in building this data structure is the following. We divide the $r \times n$ grid G_n into y^* rows and x columns that partitions G_n into a $y^* \times x$ grid G' of boxes, where $y^* = \Theta(\sqrt{r})$ and $x = \Theta(\sqrt{r})$. Specifically, we divide the interval $[n]$ into x contiguous subarrays. For $i \in [x]$, let D_i denote the i-th subarray. Additionally, we divide the range R into y^* contiguous intervals of array values, where for $j \in [y^*]$, we use I_j to denote the j-th interval when the intervals are sorted in the nondecreasing order of values. The set of boxes in G' is then $\{(D_i, I_j) : \ i \in [x], j \in [y^*]\}$.

For simplicity, we assume that $r = n$ for the rest of the high-level description. The $y^* \times x$ grid of boxes G' induces a poset on the y^*x boxes, which is similar to the natural poset defined on G_n. Namely, for two boxes in G' (or for two points in G_n), we have $(D_i, I_j) \preceq (I_t, D_s)$ (or $(i, j) \leq (t, s)$) if $i \leq s$ and $j \leq t$. The points in $\mathcal{L}$ form a chain in the above poset in G_n. Conversely, each chain in the poset G_n forms an increasing subsequence in the array A. Further, the boxes in G' through which $\mathcal{L}$ passes also forms a chain in the poset in G'. On the other hand, every chain of boxes in the poset in G' induces a number of chains in the poset in G_n, but of possibly quite different lengths. Our strategy is to find a small collection of chains in the poset in G' that cover all boxes through which the fixed $\mathcal{L}$ passes, and then to estimate the length of an LIS in each of these chains of boxes.

Let $I \subseteq R$ be a subset of the range R of values and B be a subarray of A. The density of the box (B, I), denoted by $\mathsf{den}(B, I)$, is defined to be the fraction of indices in the subarray B whose values belong to the interval I. In other words, for each box $(D_i, I_j) \in G'$, its density $\mathsf{den}(D_i, I_j)$ is the fraction of indices in the subarray D_i whose values land in the interval I_j. For $\beta < 1$, a box (D_i, I_j) is said to be β-dense, if $\mathsf{den}(D_i, I_j) \geq \beta$. There can be at most $\frac{1}{\beta}$ boxes that are β-dense in any particular subarray D_i.

Suppose that we know (a good approximation of) the density of every box in G' (this is what we require from our data structure, and this will be achieved via sampling). Then, we may restrict our attention to the at most x/β dense boxes in G' and compute the LIS only in the corresponding part of G_n. This is obviously an underestimate of the size of $\mathcal{L}$, but one that can be afforded; deleting every box that is not β-dense from the chain of boxes that $\mathcal{L}$ passes through causes the deletion at most βn points from $\mathcal{L}$.

We note that the same global idea is also used in the algorithms of [18] and (implicitly) also of [19], but in a completely different setting (and grid sizes) which makes the first one weaker in term of approximation guarantees, and the second one necessarily adaptive.

Next, in order to further reduce the number of possible chains of boxes in which we need to compute LIS, we note that we can delete large antichains of boxes from G', while not decreasing the LIS size by much. For this, we first consider a finer partition of each dense box into dense cells of nearly equal densities, and then define a poset on the set of all dense cells in the whole array. We then remove large antichains from this latter poset and argue that the removal of dense cells participating in these antichains does not *hurt* the LIS too much. Finally, by using Dilworth's theorem, we are able to obtain a collection of a *constant* number of chains in G', that covers the restriction of $\mathcal{L}$ to the undeleted boxes.

The next idea is to estimate the LIS in each of the constantly many remaining chains. This results in a loss of a multiplicative constant factor in the LIS size estimation.

At this point, we have reduced the problem to the estimation of the LIS in a given fixed chain of β-dense boxes in G'. Such a chain can be partitioned into two chains, one that contains only strictly horizontal chains on disjoint subarrays, and the other that contains only strictly vertical chains on disjoint interval ranges (see Figure 3). We will estimate the LIS in each, losing possibly another multiplicative 2-factor, which we are prepared to accept.

The final idea is the following. For the vertical going chain, one can just sample a constant number of vertically going subchains, estimate the LIS length in each one of them, and use these estimates to estimate the LIS length in the vertical chain. By the Hoeffding bound, this will be a good approximation. When $r = n$, estimating the LIS in a single vertical going subchain is trivial; we just query all $n/x = \tilde{O}(\sqrt{n})$ points in the subarray D_i corresponding to that subchain. For smaller r, this is not possible, and what we do is to reduce to the algorithm implied in Theorem 1.4, using the fact that most vertically going chains span a small range (this later fact will have to be argued from the way the data structure is formed).

For horizontally going chains, we will need a bit more from our data structure. The partition G' of G_n will be such that every layer formed by $i \in [y^*]$ contains either a small fraction of points from $\mathcal{L}$, or it contains only one range value. This is the only place in which we actually make use of the fact that $y^* = \Omega(\sqrt{n})$, which lower bounds the query complexity of the algorithm. Having this guarantee on the grid G', it would have been enough to sample a constant number of horizontally going subchains, and estimate the LIS within. Further, by the guarantee above, each horizontal layer in G' contains only a small number of distinct values. This implies that we could again employ our algorithm of Theorem 1.4. However, this will make the whole algorithm adaptive (as one has to "locate" the horizontal segments). Instead, we show that we can concentrate on short (spanning a constant number of boxes) subchains, which will allow us to employ the algorithm given by Theorem 1.4 nonadaptively.

1.1.3 Separating erasure-resilient testing from tolerant testing

Tolerant testing is a generalization of property testing [17, 10] defined by Parnas, Ron and Rubinfeld [15]. Specifically, a $(\delta, \epsilon + \delta)$-tolerant tester of monotonicity distinguishes, with probability at least $2/3$, between the cases that the distance of A to monotonicity is less than δn and at least $(\epsilon + \delta)n$, where $\epsilon, \delta \in (0, 1)$.

It has been observed [15] that a tolerant tester for a property is equivalent to an algorithm for estimating the distance to that property with an additive error guarantee. Hence, the task of estimating the LIS up to an additive error is equivalent to tolerant monotonicity testing. This allows us to restate Theorem 1.1 in terms of tolerant testing as follows.

▶ **Theorem 1.6.** *For every $\epsilon \in (0, 1)$, there exists a constant $\delta \in (0, 1)$ such that every nonadaptive 2-sided error $(\delta, \delta + \epsilon)$-tolerant tester of monotonicity has query complexity $\log^{\Omega(\log(1/\epsilon))} n$.*

Theorem 1.6 and Theorem 1.3 together imply that for the property of monotonicity, nonadaptive tolerant testing is strictly harder than nonadaptive ER testing, and also significantly less efficient than adaptive tolerant testing. Our results make progress towards answering the open question raised by Raskhodnikova, Ron-Zewi, and Varma [16] on the existence of natural properties for which one can show a separation between tolerant testing and ER testing in terms of query complexity.

1.2 Organization

We set our notations in Section 2. The proof for an important special case of our lower bound on the query complexity of nonadaptive LIS estimation (Theorem 1.6) is presented in Section 3 and the proof in its full generality is deferred to the full version [13]. Our nonadaptive and parameterized algorithm for multiplicative error LIS estimation (Theorem 1.5) and its analysis are presented in Section 4. Our algorithm for additive error LIS estimation (Theorem 1.4), and our nonadaptive erasure-resilient monotonicity tester (Theorem 1.3) can both be found in the full version [13]. All omitted proofs can also be found in the full version [13].

2 Notations and Preliminaries

For a natural number n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For a real-valued array A of length n, we use $A[i]$ to denote the i-th entry of A for $i \in [n]$. For $x \leq y \in [n]$ we denote by $[x, y]$ the set $\{x, x + 1, \ldots, y\}$. The array A is monotone if for every two indices $u, v \in [n]$

such that $u < v$, we have $A[u] \leq A[v]$. If A is not monotone, two indices $u, v \in [n]$ are said to violate monotonicity if $u < v$ and $A[u] > A[v]$. For $\epsilon \in (0, 1)$, we say that A is ϵ-far from monotone if the values on at least $\epsilon \cdot n$ indices need to be modified to get a monotone array. A is ϵ-close to monotone if there is a way to modify the values on fewer than $\epsilon \cdot n$ indices to get a monotone array. For a parameter $\alpha \in [0, 1)$, we say that A is α-erased, if at most α fraction of the array values evaluate to a special symbol $\perp$. An assignment of values to the erased points in an array is called a completion. An α-erased array is monotone if there exists a completion that is monotone; it is ϵ-far from monotone if every completion is ϵ-far from monotone. We assume that algorithms accesses an input array A via an oracle; that is when the algorithm makes a query $i \in [n]$, the oracle returns a special symbol $\perp$ if the array value at index i is erased, and $A[i]$ otherwise. An algorithm is *adaptive* if its queries depend on the answers to its previous queries, and is *nonadaptive* otherwise. A partially ordered set (poset) is a set $\mathcal{P}$ associated with a reflexive, transitive, antisymmetric order relation $\preceq$ on its elements. We denote the poset by $\langle \mathcal{P}, \preceq \rangle$. A chain in $\langle \mathcal{P}, \preceq \rangle$ of length k is a sequence of elements $x_1 \preceq x_2 \preceq \cdots \preceq x_k$. An antichain is a set $S \subseteq \mathcal{P}$ such that for $u, v \in S$ neither $u \preceq v$ nor $v \preceq u$.

3 Lower Bounds for Nonadaptive LIS Estimation

In this section, we prove our lower bounds (Theorem 1.1) on the query complexity of 2-sided error nonadaptive algorithms for estimating the distance of real-valued arrays of length n from monotonicity up to an additive error of $\epsilon \cdot n$ for some constant $\epsilon \in (0, 1)$. Equivalently, our lower bounds also hold for algorithms that $(\delta, \epsilon + \delta)$-tolerant test monotonicity for constants $\delta, \epsilon \in (0, 1)$. Interestingly, our lower bounds use ideas from the lower bound on the query complexity of 1-sided error nonadaptive testers for the property of $(k, \ldots, 2, 1)$-freeness [4], where an array A of length n is $(k, \ldots, 2, 1)$-free if there are no k indices $i_1 < i_2 < \cdots < i_k$ such that $A[i_1] > A[i_2] > \cdots > A[i_k]$.

An algorithm is said to be *comparison-based* if its decisions are based only on the ordering relation between the queried values, and not on the values themselves. The following Lemma 3.1, which follows from the work of Fischer [8], states that it is enough to restrict our attention to comparison-based algorithms.

▶ **Lemma 3.1** ([8]). *There is an optimal comparison-based algorithm for computing an additive ϵn-approximation to the LIS in real-valued arrays of length n for all constant $\epsilon \in (0, 1)$.*

Even though Fischer [8] proves the above statement in the context of testing monotonicity in the standard model, his proof also works for the case of tolerant testing monotonicity, and in turn for LIS estimation. In the rest of this section, we restrict our attention to comparison-based algorithms for LIS estimation.

3.1 An $\Omega(\log^2 n)$ Lower Bound

As a starting point, we prove an $\Omega(\log^2 n)$ lower bound. Throughout this section, we assume that n is of the form 2^{2^x} for some natural number x. We prove the lower bound using Yao's method. Specifically, we describe two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ over real-valued arrays of length n, with different distances to monotonicity (Lemma 3.2), and show that every deterministic nonadaptive comparison-based algorithm distinguishing these distributions with probability at least $2/3$, has to make $\Omega(\log^2 n)$ queries (Lemma 3.3).

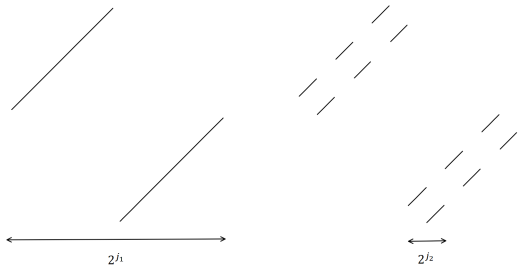

Figure 1 The relative values in j_1-blocks of $\mathcal{D}_0$ either look like the left or right diagrams above, with equal probability.

For ease in describing our distributions, we first define some notation. We think of the indices of an array of length n as the leaves of an ordered binary tree $\mathcal{T}$ of height $\log(n)+1$. We associate bit positions in the $\log n$-bit representation of the numbers in $[n]$ with the non-leaf nodes of $\mathcal{T}$. The root is associated with the most significant bit (or the bit position $\log n$). Every node at distance $i \in [\log(n)-1]$ from the root is associated with bit position $\log(n)-i$. The bit position associated with a node is also referred to as its *level* (and the level of the root is $\log n$). The edges connecting a node with its left and right children are labeled 0 and 1, respectively. Clearly, the string obtained by concatenating all the edge labels on a root-to-leaf path in $\mathcal{T}$ gives the binary representation of the index corresponding to the leaf. For two indices $x, y \in [n]$ (which are, by definition, the leaves in $\mathcal{T}$), we use $\mathsf{LCA}(x,y)$ to denote the lowest common ancestor of x and y in $\mathcal{T}$. For $j \in [\log n]$ there are obviously $n/2^j$ subtrees of $\mathcal{T}$, each rooted at level j. In a left to right ordering, these $n/2^j$ subtrees partition $[n]$ into blocks of size 2^j. The "the ℓ-th j-block" is the ℓ-th block from left with size 2^j.

The distributions $\mathcal{D}_0$ and $\mathcal{D}_1$

We first describe the steps that are common to constructing the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$. Sample numbers $j_1, j_2 \in [\log n]$ such that $j_1 < \log(n) - 14$, and $j_2 < j_1 - 14$. We refer to the numbers j_1, j_2 as the *scales* of the distributions.

We start with the monotone array A in which $A[u] = u$ for all $u \in [n]$. Swap the array values between the left and right halves of every j_1-block. See the left part of Figure 1 to see how the relative values in each j_1-block of the array will look like at this point. For $\ell \in [n/2^{j_1}]$, let B_ℓ denote the ℓ-th j_1-block.

- **Distribution** $\mathcal{D}_0$: Independently for each $\ell \in [n/2^{j_1}]$:
  1. with probability $\frac{1}{2}$, for each j_2-block inside B_ℓ, swap the array values between the left and right halves of that j_2-block.
- **Distribution** $\mathcal{D}_1$: Independently for each $\ell \in [n/2^{j_1}]$:
  1. with probability $\frac{1}{2}$, for each j_2-block inside the left half of B_ℓ, swap the array values between the left and right halves of that j_2-block,
  2. with the remaining probability $\frac{1}{2}$, for each j_2-block inside the right half of B_ℓ, swap the array values between the left and right halves of that j_2-block.

The relative values taken by the array A on indices in an arbitrary j_1-block in distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ can be visualized as in Figures 1 and 2. We note that in both $\mathcal{D}_0, \mathcal{D}_1$, all values in the ℓ-th j_1-block are smaller than the values in the $(\ell+1)$-th j_1-block for all $\ell \in [(n/2^{j_1})-1]$.

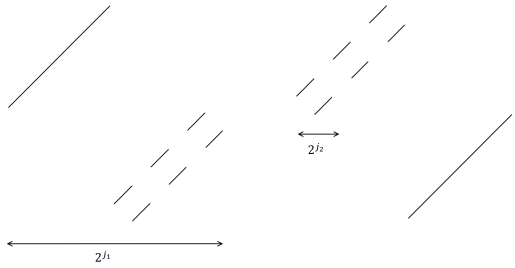

Figure 2 The relative values in j_1-blocks of $\mathcal{D}_1$ either look like the left or right diagrams above, with equal probability.

▶ **Lemma 3.2.** *The distance to monotonicity of every array sampled from $\mathcal{D}_0$ is, with probability $1 - \delta/2$, within $\frac{5}{8} \pm \delta$, where $\delta \leq \frac{1}{2^6}$. The distance to monotonicity of every array sampled from $\mathcal{D}_1$ is equal to $1/2$.*

Proof. Consider an array A sampled from one of the distributions. Let $j_1 > j_2$ be the scales used. First, observe that there are no violations to monotonicity across j_1-blocks. Therefore, it is enough to focus on repairing individual j_1-blocks and making them monotone (without inducing new violations). Consider a j_1-block B_ℓ for $\ell \in [n/2^{j_1}]$.

Assume that A is constructed from $\mathcal{D}_0$, and that B_ℓ is such that the values in the left and right halves of every j_2-block in B_ℓ are swapped (happens with probability $\frac{1}{2}$ for that block). Then we need to modify the values of at least $3/4$ fraction of indices in B_ℓ to make it monotone, since B_ℓ contains an exact cover by disjoint decreasing subsequences, each of size 4. Further, it is easy to see that by correcting a $3/4$ fraction of indices we can make B_ℓ monotone. In case swapping of values is done for none of the j_2-blocks in B_ℓ (happens with probability $\frac{1}{2}$), then we can repair B_ℓ if and only if we modify the values on half the indices in B_ℓ. Therefore, the expected distance to monotonicity of each block B_ℓ, and thereby, of A is equal to $5/8$. Note that the specific values of the scales did not matter in the above.

Let $\delta = 1/2^6$. We now show that the distance of A from monotonicity is $\frac{5}{8} \pm \delta$, with high probability. For a block $B_\ell, \ell \in [n/2^{j_1}]$, let $\mathsf{dist}(B_\ell)$ denote the distance of the block from monotonicity, normalized by the block length 2^{j_1}. We can see that the random variable $(\sum_\ell \mathsf{dist}(B_\ell))/(n/2^{j_1})$ corresponds to the normalized Hamming distance of A from monotonicity. By Hoeffding's bound, we have, $\Pr\left[\left|\frac{\sum_\ell \mathsf{dist}(B_\ell)}{n/2^{j_1}} - \frac{5}{8}\right| > \frac{1}{2^6}\right] \leq \frac{2}{\exp(8)} \leq \frac{1}{2^7} = \frac{\delta}{2}$.

Assume now that A is constructed from $\mathcal{D}_1$. For $\ell \in [n/2^{j_1}]$, if the swap of values happens within every j_2-block in the left half of B_ℓ, then we can repair B_ℓ by setting every value in the left half of that block to the smallest value in the right half of the same block. An analogous repair can be done if the swap happens in the right hand side of the block. In both cases, we only change values of at most half the number of indices in each block. The reader can easily convince themselves that at least half the values per block need to be changed to make a j_1-block monotone, which concludes the proof for the given scales. As before, the argument is independent of the choice of the scales. ◀

▶ **Lemma 3.3.** *Every comparison-based nonadaptive deterministic algorithm that, with probability at least $2/3$, distinguishes the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ has to make $\Omega(\log^2 n)$ queries.*

Proof. Consider an arbitrary deterministic comparison-based nonadaptive algorithm T that makes $o(\log^2 n)$ queries and aims to distinguish $\mathcal{D}_0$ and $\mathcal{D}_1$. Let $Q \subseteq [n]$ denote the set of queries that T makes.

Consider an array A sampled according to one of the distributions. Recall that $\mathcal{T}$ denotes an ordered binary tree whose leaves are the indices of A. Let $j_1, j_2 \in [\log n]$ such that $j_2 < j_1$ denote the scales used while constructing A. Let E denote the (bad) event that Q contains four indices $w < x < y < z \in [n]$ such that for some $\ell \in [n/2^{j_1}]$, (1) Each of $\{w, x, y, z\}$ belongs to the same ℓ-th j_1-block, (2) $\mathsf{LCA}(w, x)$ and $\mathsf{LCA}(y, z)$ are both nodes in $\mathcal{T}$ at level j_2, and (3) $\mathsf{LCA}(x, y)$ is at level j_1. By an argument of Ben-Eliezer, Canonne, Letzter, and Waingarten [4], the probability, over the choice of the scales j_1, j_2, of the event E is at most $1/3$. In the rest of the proof, we fix the scales (j_1, j_2) for which E does not happen.

Let $x, y \in Q$ be such that $\mathsf{LCA}(x, y)$ is at level j_2 in $\mathcal{T}$. In the rest of the proof, for simplicity, we refer to such queries as being j_2-*cousins*. Let B be a j_1-block, and let $Q_L(B)$ be the queries in Q that are in the left half of B, and $Q_R(B)$ the queries in Q that are in the right half of B. By our conditioning, for each j_1-block B, either all j_2-cousins in B belong to $Q_L(B)$ or to $Q_R(B)$ but not both. Consider the half of B that does not contain any j_2-cousins. In the algorithm's view, the array values in that half are increasing, whether A is sampled from $\mathcal{D}_0$ or $\mathcal{D}_1$.

Assume that A is sampled from $\mathcal{D}_0$. We show that there is a way to sample an array A' from $\mathcal{D}_1$ such that the algorithm's view on A and A' are identical. The scales of A' have to be identical to those of A. We only need to specify how swapping of values is done inside the j_1-blocks, as part of constructing A'.

Note that we only need to consider the j_1-blocks in which at least two queries fall. Consider such a block B, and assume that $Q_R(B)$ contains no j_2-cousins. Consider the case that swapping of values was done within every j_2-block inside the block B while constructing A. Then, in A', we swap the values only within the j_2-blocks inside the left half of B. In the other case that no swapping of values (within j_2-blocks) was done while constructing A, we swap the values only within the j_2-blocks inside the right half of B. It is easy to see that the relative values within block B in the array A' are consistent with that of an array sampled from $\mathcal{D}_1$. One can make similar arguments about coupling the arrays A and A' on blocks B such that the only occurrences of indices $x, y \in Q \cap B$ that are j_2-cousins are in the right half of B.

We conclude that for any scales j_1, j_2 for which E does not happen, the view of the algorithm making queries Q is identical on A' and A. Hence the algorithm cannot distinguish between the case that the array is sampled from $\mathcal{D}_0$ or from $\mathcal{D}_1$ for such scales. As this is true for any scales for which E does not happen, this concludes the proof.

Observe that the only place in the analysis where we made use of the bound on the number of queries is in arguing that the event E happens with low probability. ◀

3.2 A $\log^{\omega(1)} n$ Lower Bound

Next, we strengthen the $\Omega(\log^2 n)$ lower bound and prove Theorem 1.1. We make use of the idea of Ben-Eliezer et al. [4] for lower bounding query complexity of nonadaptive detection of larger forbidden order patterns. The idea is to use more than just two scales. This, in turn, makes the difference between the distances of arrays sampled from the two distributions smaller, which is why we only get a $\log^{\omega(1)} n$ lower bound.

Let $h \geq 2$ be an integer parameter. We describe the two distributions $\mathcal{D}_0^{(h)}$ and $\mathcal{D}_1^{(h)}$ on real-valued arrays, such that no comparison-based deterministic nonadaptive algorithm that makes $o(\log^h n)$ queries can distinguish between the distributions with high probability. Further, the distance to monotonicity of each array sampled from $\mathcal{D}_0^{(h)}$ will be significantly different than that of arrays sampled from $\mathcal{D}_1^{(h)}$.

The distributions are defined recursively, where, for the base case $h = 2$, let $\mathcal{D}_0^{(2)}$ and $\mathcal{D}_1^{(2)}$ be equal to $\mathcal{D}_0$ and $\mathcal{D}_1$ defined with scales j_1, j_2 as in Section 3.1, respectively. The details of the distributions and the proof of lower bound for the general case is much more technical and it is deferred to the full version [13].

4 Parameterized and Nonadaptive Algorithm for LIS Estimation

Our final goal in this paper is to present a sublinear algorithm that, for an array of length n containing at most r distinct values, approximates the LIS length within a bounded multiplicative error (Theorem 1.5). Our algorithm is described in the following subsections, and it uses as a subroutine, the algorithm guaranteed by Theorem 1.4 that approximates the LIS length within a bounded additive error. The description and analysis of the latter algorithm can be found in the full version of the paper [13].

4.1 $\tilde{O}(\sqrt{r})$-Query Nonadaptive Algorithm

Let $\mathcal{L}$ denote the set of points in an arbitrary and fixed LIS in the input array A. For simplicity of the presentation, we assume that our algorithm knows a lower bound λn on $|\mathcal{L}|$. Disregarding this assumption, the algorithm will output, with high probability, a lower bound estimate of the size of an increasing sequence in A. If λn is indeed a bound as assumed, it will be guaranteed that the estimate is within the multiplicative error that is stated. Hence λ can be checked by running the algorithm, in parallel, for a geometrically decreasing sequence of λ's. The reader may think of $\lambda < 1$ as a small constant (although the algorithm works for $\lambda = o(1)$ as well).

Throughout this section, we visualize the array values as points in an $r \times n$ grid G_n. The vertical axis of G_n represents the range R of the array and is labeled with the at most r distinct array values in increasing order and the horizontal axis is labeled with the indices in $[n]$. We refer to an index-value pair in the grid as a point. The grid has n points, to which we do not have direct access.

We divide the $r \times n$ grid G_n into y^* rows and x columns that partitions G_n into a $y^* \times x$ grid G' of boxes, where $y^* = \Theta(\sqrt{r})$ and $x = \Theta(\sqrt{r})$. Specifically, we divide the interval $[n]$ into x contiguous subintervals. For $i \in [x]$, let D_i denote the subarray induced by the indices in the i-th subinterval. Additionally, we divide the range R into y^* contiguous intervals of array values, where for $j \in [y^*]$, we use I_j to denote the j-th interval when the intervals are sorted in the nondecreasing order of values. The set of boxes in G' is then $\{(D_i, I_j) : \ i \in [x], j \in [y^*]\}$.

The $y^* \times x$ grid of boxes G' induces a poset $\langle \mathcal{P}, \preceq \rangle$ on the $y^* x$ boxes, which is similar to the natural poset defined on G_n. Namely, for two boxes in G' (or for two points in G_n), we have $(D_i, I_j) \preceq (D_s, I_t)$ (or $(i, j) \leq (s, t)$) if $i \leq s$ and $j \leq t$. The points in $\mathcal{L}$ form a chain in the above poset in G_n. Further, each chain in the poset in G_n forms an increasing subsequence in the array A. The boxes in G' through which $\mathcal{L}$ passes also forms a chain in the poset in G'. Every chain of boxes in the poset in G' induces a number of chains in the poset in G_n, but of possibly quite different lengths.

▶ **Definition 4.1** (Density of a box). *Let $I \subseteq R$ be a subset of the range R of values and B be a subarray of A. The density of the box (B, I), denoted by $\mathsf{den}(B, I)$, is defined to be the fraction of indices in the subarray B whose values belong to the interval I.*

In other words, for each box $(D_i, I_j) \in G'$, its density $\mathsf{den}(D_i, I_j)$ is the fraction of points in the subarray D_i that land in the box (D_i, I_j). For $\beta < 1$ (that the reader can think of as a small constant), a box (D_i, I_j) is said to be β-dense, if $\mathsf{den}(D_i, I_j) \geq \beta$.

4.1.1 Forming the grid G' of boxes

Our goal in this section is to describe a procedure that determines the grid G' of boxes. Specifically, as we do not know the range R and only know an upper bound r on its size $|R|$, we start by forming an approximation of R and an approximation of the densities of subinterval ranges in R in the array A. To do this, we first partition R into $\tilde{O}(\sqrt{r})$ sub-ranges called *layers*. For the sake of generality, we describe the procedure for a subarray B of A.

More generally, given a subarray B, and a parameter y, our goal is to partition the range R into roughly y intervals of roughly equal densities, where the densities are with respect to B. We note that although the size r of the range R might be relatively large, it is possible that some values appear in B much more frequently than others. One of our goals is to identify such values and well-approximate their densities. We now define a "nice" partition as follows. Given y and B, a nice y-partition of the values in B is a partition $R = \cup_{i=1}^{y^*} I_i$, if for each $i \in [y^*]$, either I_i contains only one value v_i and $\mathsf{den}(B, I_i) \geq \frac{1}{2y}$, or $\mathsf{den}(B, I_i) \leq \frac{2}{y}$. In the former case, we call I_i a single-valued layer. In the latter, we say that I_i is a multi-valued layer (although in an extreme case it might contain only one value). We also require $y^* \leq 2y$.

Next, we describe our procedure $\textsc{Layering}(B, y, t)$ that forms a y-nice partition of a subarray B, along with a good approximation of the densities of the single-valued layers. This is quite technical, although standard. We advise the reader to avoid it on first reading, and assume that, when needed, we have a nice partition along with a good approximation to the densities of layers.

Algorithm 1 Layering.

procedure $\textsc{Layering}(B, y, t)$

Goal: To divide the set of array values in the subarray B into roughly y contiguous intervals of roughly equal densities. The parameter t is used to control the success probability.

Sample a set of $\ell = t \cdot y \log y$ indices S from B, uniformly at random and independently. ▷ Note that a value v is expected to appear in proportion to its density in the array. Hence the collection of values obtained is a multiset of size ℓ.

We sort the multiset of values $V = \{B[p] : p \in S\}$ to form a strictly increasing sequence $\mathsf{seq} = (v'_1 < \ldots < v'_q)$, where with each $i \in [q]$ we associate a weight w_i that equals the multiplicity of v'_i in the multiset V of values. ▷ Note that $\sum_{i \in [q]} w_i = \ell$.

We now partition the sequence $W = (w_1, \ldots, w_q)$ into maximal disjoint contiguous subsequences $W_1, \ldots W_{y^*}$ such that for each $j \in [y^*]$, either $\sum_{w \in W_j} w < 2t \log y$, or W_j contains only one member w for which $w > t \log y$.

Note that this can be done greedily as follows. If $w_1 > t \log y$ then W_1 will contain only w_1, otherwise W_1 will contain the maximal subsequence $(w_1, \ldots, w_i)$ whose sum is at most $2t \log y$. We then delete the members of W_1 from W and repeat the process. For $i \in [y^*]$, let $w(W_i)$ denote the total weight in W_i.

Correspondingly, we obtain a partition of the sequence seq of sampled values into at most y^* subsequences $\{S_i\}_{i \in [y^*]}$. Some subsequences contain only one value of weight at least $t \log y$ and are called *single-valued*. The remaining subsequences are called *multi-valued*.

For a subsequence S_i, let $\alpha_i = \min(S_i)$ and $\beta_i = \max(S_i)$. Let $\beta_0 = -\infty$. Note that $\alpha_i \leq \beta_i$ and $\beta_{i-1} < \alpha_i$ for all $i \in [y^*]$.

For $i \in [y^*]$, we associate with the subsequence S_i, an interval (layer) $I_i \subseteq R$, where $I_i = (\beta_{i-1}, \beta_i]$, and an approximate density $\widetilde{\mathsf{den}}(B, I_i) = w(W_i)/\ell$.

end procedure

Let $\{I_i\}_{i=1}^{y^*}$ be the set of layers that are created by a call to LAYERING(B, y, t). Recall that $w(W_i) \geq t \log y$ if I_i is a single-valued layer and $w(W_i) < 2t \log y$ if I_i is multi-valued, where W_i denotes the sum of multiplicities of the values in the sample S_i.

For a multi-valued layer I_i, let E_i denote the event that $\mathsf{den}(B, I_i) < \frac{4}{y}$. For a single-valued layer I_i such that $\mathsf{den}(B, I_i) \geq \frac{1}{2y}$, let E_i denote the event that $\frac{\mathsf{den}(B,I_i)}{2} \leq \widetilde{\mathsf{den}}(B, I_i) \leq \frac{3}{2}\mathsf{den}(B, I_i)$. For a single-valued layer I_i such that $\mathsf{den}(B, I_i) < \frac{1}{2y}$, let E_i be the event that $\widetilde{\mathsf{den}}(B, I_i) \leq \frac{3}{2}\mathsf{den}(B, I_i)$. Let $E = \bigcap_{i=1}^{y^*} E_i$. The following claim asserts that the layering above well-represents the structure of the range w.r.t. B.

▷ Claim 4.2. LAYERING(B, y, t) returns a collection of intervals $\{I_i\}_{i=1}^{y^*}$ such that, $y^* \leq 2y$, and $\Pr(E) = 1 - \exp(\Omega(-t))$.

We now define the grid G' of boxes as follows. We first use the procedure LAYERING on the original array, $B = A$, with parameters $y = \frac{\sqrt{r}}{\epsilon}$ and $t = O(1)$, where the value of t is set to ensure a success probability of 99/100 in Claim 4.2. This defines the set of y^* layers that partitions R as $R = \bigcup_{i \in y^*} I_i$. Next we partition $[n]$ into $x = \epsilon \cdot \sqrt{r}$ contiguous intervals $D_1, \ldots D_x$ each of size n/x, which defines G' as the grid of boxes $\{(D_i, I_j) : (j, i) \in [y^*] \times [x]\}$ in the $r \times n$ grid, some of which may be empty, while some may contain many points.

We set $\beta = \epsilon^3 \lambda$. Next, our goal is to find all the β-dense boxes in G' by making $\tilde{O}(\sqrt{r})$ queries and then restrict our attention only to these boxes. As described in the high level overview Introduction, doing this will not make the LIS in this restricted array too short. This is made formal in the following claim.

▷ Claim 4.3. The number of points in $\mathcal{L}$ that belong to boxes that are not β-dense is at most $\beta n \cdot (1 + 2y/x)$.

We do not know which boxes are β-dense. We approximate this by sampling, and this is formally presented below as algorithm GRIDDING. The algorithm assumes the partition of $[n]$ and of the range R as above. As before, $t = O(1)$ in this procedure can be set appropriately to ensure a large constant success probability.

Algorithm 2 GRIDDING.

```
1: procedure GRIDDING(A, {I_j}_{j∈[y*]}, {D_i}_{i∈[x]}, β)
2:     for i ∈ [x] do
3:         Sample ℓ = t · (1/β) · log(x/β) indices from D_i uniformly and independently at random.
4:         for j ∈ [y*] do
5:             Label box (D_i, I_j) as dense if and only if the values on at least (3/4)βℓ points
   from the sample fall into the box; namely, if for at least (3/4)βℓ indices sampled from D_i,
   the values are in I_j.
6:         end for
7:     end for
8: end procedure
```

Let D be the event that all β-dense boxes are tagged as *dense* by the procedure, and that every box that is not $\beta/8$-dense is not tagged as dense.

▷ Claim 4.4. $\Pr[D] \geq 1 - \exp(-\Omega(t))$.

From now on, we assume that we have the grid G' for which the events E and D hold. This is the initialization of our data structure as described in the Introduction. We now refine the data structure as follows.

A β-dense box may have density that is anything in $[\beta, 1]$. For a better approximation guarantee, we need to identify the regions with density nearly equal to β. To achieve this, we perform the following finer layering using the procedure LAYERING on each dense box.

Finer layering of each dense box. For each $i \in [x]$, call LAYERING on the array D_i with $y = 1/\beta$ and $t' = \Theta(\log(x/\beta))$. Here, we do not collapse the single-valued intervals into a single layer, but rather just leave them as different layers of the same value and density β.

Let $\{I'_k\}_{k \in [y_i^*]}$ be the set of intervals returned by the procedure. We restrict our attention to the boxes (D_i, I'_k) contained in some β-dense box (D_i, I_j), and call them β-dense cells.

Fix D_i. The number of β-dense cells in D_i is at most $2y = 2/\beta$. Claim 4.2 asserts that, with probability $1 - \exp(-\Omega(t')) = 1 - \frac{\beta}{100x}$, each D_i is layered so that each β-dense cell has true density at most $3\beta/2$ and at least $\beta/8$. Additionally, the portion of a β-dense box that is not covered by β-dense cells has true density smaller than β. This implies that for all $i \in [x]$ this event happens with probability at least $99/100$. We denote this event by F, and assume in what follows that F happens.

4.1.2 Chain reduction

In this section, we define a poset over dense cells and argue that in order to well-approximate the LIS, it is enough to restrict our attention to LIS's in a few chains in this poset.

Since dense cells, by definition, are contained inside dense boxes, we denote dense cells using triplets (D_i, I_j, k), where this triplet denotes the k-th dense cell inside the dense box $(D_i, I_j), i \in [\epsilon\sqrt{r}], j \in [\sqrt{r}/\epsilon]$.

Recall that there is a poset $\langle \mathcal{P}, \preceq \rangle$ on the dense boxes. Now, we define another poset $\langle \mathcal{P}^\star, \preceq^\star \rangle$ whose elements are the (at most) $2x/\beta$ dense cells. The order relation $\preceq^\star$ is defined by $(D_i, I_j, k) \preceq^\star (D_{i'}, I_{j'}, k')$ if and only if either $(D_i, I_j) \neq (D_{i'}, I_{j'})$ and $(D_i, I_j) \preceq (D_{i'}, I_{j'})$, or if $j' = j, i' = i$ and $k \leq k'$. Note that the poset $\preceq^\star$ is not consistent with a grid poset, it rather inherits the order from $\mathcal{P}$ for cells in different boxes.

Let $\mathcal{L}_1$ be the LIS $\mathcal{L}$ restricted to dense boxes, let $C(\mathcal{L}_1, \mathcal{P})$ be the set of dense boxes in which $\mathcal{L}_1$ passes, and let $C(\mathcal{L}_1, \mathcal{P}^\star)$ be the set of dense cells in which $\mathcal{L}_1$ passes. We observe that $C(\mathcal{L}_1, \mathcal{P})$ and $C(\mathcal{L}_1, \mathcal{P}^\star)$ are chains in the corresponding posets.

Our goal now is to show that there are a small number of chains in $\mathcal{P}^\star$ that cover $C(\mathcal{L}_1, \mathcal{P}^\star)$. This is done as follows.

- For parameter $\tau = 5/\lambda$, repeatedly remove antichains of size larger than τ from $\mathcal{P}^\star$. Here, by removing, we mean the deletion of the points in the cells of the corresponding antichain from the array[6].

Let the resulting poset be denoted by $\mathcal{P}^{\star\star}$. The maximum antichain in this poset has size at most τ, and Dilworth's theorem implies that there is a decomposition of $\mathcal{P}^{\star\star}$ into at most τ chains. These chains, being made of dense cells, is naturally extended to at most τ (possibly intersecting) chains of the poset $\mathcal{P}$. Let these chains be $C_1, \ldots, C_\tau$. We bound the "loss" to the LIS incurred by chain reduction in Claim 4.5.

▷ Claim 4.5. Conditioned on the events $E \cup D \cup F$, the number of points in $\mathcal{L}_1$ that does not belong to $\cup_{i=1}^{\tau} C_i$ is at most $4n/\tau$.

[6] For a single-valued cell $a = (D_i, I_j, k)$ taking the value v, there might be other points with value v in the dense box containing a. When we remove a from $\mathcal{P}^\star$, we "mentally" remove some $\beta n/x$ points of value v from the box (D_i, I_j). This is not algorithmically done, but will just be used in the analysis.

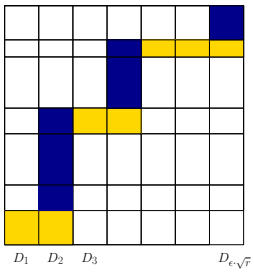

Figure 3 A staircase like chain, and its decomposition into two chains, one that contains only horizontal blocks and one that contains only vertical blocks. In each of the chains, no two blocks share a layer or a subarray.

Let $\mathcal{L}_2$ denote the LIS $\mathcal{L}_1$ after chain reduction. The following claim is straightforward.

▷ Claim 4.6. There exists $i \in [\tau]$ such that $|\text{LIS}(C_i)| \geq \frac{|\mathcal{L}_2|}{\tau}$.

We point out that no query is made at this stage.

4.1.3 Estimating the LIS restricted to poset chains

Let $\mathcal{P}'$ denote the poset obtained from $\mathcal{P}$ after removing the large antichains in $\mathcal{P}^\star$. At this point, we have covered the poset $\mathcal{P}'$ using at most τ chains $C_1, \ldots, C_\tau$. This reduces the LIS estimation to estimating the LIS in one of these chains.

In what follows, we fix such one chain C, and denote by $\mathcal{L}(C)$, the LIS in the array restricted to C. The chain C is composed of a sequence of horizontal and vertical blocks, arranged in a staircase manner (see Figure 3), where a horizontal block is a sequence of contiguous boxes in the chain from the same layer, and a vertical block is a sequence of contiguous dense boxes in the chain that belong to the same subarray.

Let H_1, H_2 be two maximal horizontal blocks in C. Blocks H_1, H_2 have a subarray D_i in common if there are boxes $(D_i, I_j) \in H_1$ and $(D_i, I_s) \in H_2$. In particular, there is a vertical block between them. Two horizontal chains have at most one subarray in common, and if this happens, then the common subarray D_i defines the rightmost box of the "lower" horizontal chain (the horizontal chain in the lower layer) and the leftmost box of the "upper" horizontal chain. We conclude that if we arrange the horizontal blocks from bottom to top as $H_1, \ldots H_s$, and remove the rightmost box from H_i if it has a common subarray with H_{i+1}, we get a sequence of horizontal blocks in which no two share a subarray. We use C_H to denote this subchain of C. Notice that $C_V = C \setminus C_H$ is a chain that contains only vertical blocks, where no two share a layer (see Figure 3, as how the whole chain C decomposes into a chain of horizontal blocks and a chain of vertical blocks).

To estimate the size of $\mathcal{L}(C)$, we estimate the LIS within C_H and C_V separately, and use the larger for the size estimate for $\mathcal{L}(C)$. In the following, we denote $\mathcal{L}_H$ and $\mathcal{L}_V$ for $\text{LIS}(C_H)$ and $\text{LIS}(C_V)$, respectively. The main advantage of this decomposition of C into C_H and C_V is given by the following observation.

▶ **Observation 4.7.** *For any chain C, we have $|\mathcal{L}_H| = \sum_{B \in C_H} |LIS(B)|$, where the sum is over the horizontal blocks B in C_H. A similar statement holds for C_V in which case, horizontal blocks are replaced with vertical blocks.*

The observation leads to an immediate adaptive way to approximate the lengths of $\mathcal{L}_H$ and $\mathcal{L}_V$. We sample a constant number of blocks from the chain, estimate the LIS in each block, and normalize to estimate the LIS in the entire chain. By the Hoeffding bound, we can

see that the estimate is accurate enough with high probability. The adaptivity is needed to locate each block, and to estimate the LIS within a block (horizontal and vertical), which we did not yet specify how to do. To avoid adaptivity, we will rely on the fact that if $\mathcal{L}_V$ is large, then it must contain a large number of small vertical blocks. Thus, sampling uniformly in $[n]$ will hit such blocks frequently enough to facilitate the Hoeffding bound above. Further, for the estimation of LIS within a short vertical block, we will use the algorithm guaranteed by Theorem 1.4. For horizontal blocks, we need some further relaxations. We will show below that we may restrict ourselves to short horizontal blocks, due to the choice of parameters in the formation of the grid G'. This again will facilitate the use of the algorithm guaranteed by Theorem 1.4. The details now follow.

Estimating the length of LIS in a horizontal chain. A horizontal block belonging to a multi-valued layer is referred to as a multi-valued horizontal block, and a horizontal block belonging to a single-valued layer is a single-valued horizontal block. We treat these horizontal blocks separately.

Let $m = \epsilon/\lambda^2$. Let $\mathcal{L}'_H$ denote the restriction of $\mathcal{L}_H$ after deleting multi-valued horizontal blocks containing more than m boxes. We first show that the length of $\mathcal{L}'_H$ is not much smaller than $\mathcal{L}_H$.

▷ Claim 4.8. $|\mathcal{L}'_H| \leq |\mathcal{L}_H| - 4\lambda^2\epsilon n$.

Let C'_H denote the chain obtained after removing multi-valued horizontal blocks containing more than m boxes. If there are at most $\phi = \epsilon\lambda^2 y$ multi-valued horizontal blocks in the chain C'_H, then, by removing all of them, we end up losing only $\phi \cdot \frac{4n}{y} \leq 4n\epsilon\lambda^2$ points (as each multi-valued layer has density at most $4/y$). If there are at least ϕ multi-valued horizontal blocks in C'_H, then the average number of values in such a horizontal block is at most $\frac{r}{\phi} \leq \frac{r}{\epsilon\lambda^2 y} = \frac{\sqrt{r}}{\lambda^2}$ by our choice of y. That is, with probability at least $1 - \frac{1}{100\log(\tau)}$, a uniformly random multi-valued horizontal block in C'_H contains at most $\frac{100\sqrt{r}\log(\tau)}{\lambda^2}$ values. Thus, we have reduced the problem to estimating the LIS in a collection of (possibly very long) single-valued horizontal blocks and several short multi-valued horizontal blocks containing $O(\frac{\sqrt{r}\log(\tau)}{\lambda^2})$ values.

In the following, we use the term *segment* to denote a subarray composed of $2m$ subarrays $\{D_i, D_{i+1}, \ldots, D_{i+2m-1}\}$ for some $i \in [x-2m+1]$. A segment is said to contain a multi-valued horizontal block H if all the subarrays forming H are contained in the segment.

Fix $r' = \frac{100\sqrt{r}\log(\tau)}{\lambda^2}$. Our algorithm for estimating the length of $\mathcal{L}_H$ is as follows:

1. Sample $t\log^2(\tau)$ uniformly random segments.
2. For each sampled segment B, query $s = \Theta\left(\frac{m\log(\tau)}{\beta} \cdot \frac{r'}{\epsilon^3\lambda^6}\log\left(\frac{r'^2}{\epsilon\lambda}\right)\right)$ points uniformly and independently at indexes from B and run the algorithm given by Theorem 1.4 with parameters r' (for the number of distinct values) and $\epsilon\lambda^2$ (for approximation guarantee) using the samples that fall into the multi-valued horizontal block H contained in the segment B, if any.
3. Estimate the contribution to the LIS from multi-valued horizontal blocks by summing the answers returned by the algorithm in the previous steps and then normalizing appropriately.
4. Estimate the contribution to the LIS from single-valued horizontal blocks by summing the estimates of the densities of all single-valued horizontal blocks in C_H (as we already know these estimates from the Gridding stage).
5. Output an estimate L_H of the length of $\mathcal{L}_H$ by summing the above two estimates.

Clearly, the contribution to $\mathcal{L}_H$ from single-valued horizontal blocks is estimated within multiplicative $(1 \pm \frac{1}{2})$-error, by our conditioning on the event F. We show the following.

▷ Claim 4.9. With probability $1 - O(\frac{\log(\tau)}{\tau^2})$, the contribution to $\mathcal{L}_H$ from multi-valued horizontal blocks is estimated within an additive error of $\epsilon\lambda^2 n$.

Estimating the length of the LIS in a vertical chain. Let $\nu = \epsilon\lambda^2$. We may assume that the vertical chain is composed of at least $\nu \cdot x$ vertical blocks, for otherwise, we can abandon the entire vertical chain by incurring a "loss" to the LIS amounting to at most $\nu \cdot n$ points. Additionally, since the boxes from different vertical blocks belong to different layers, using a similar averaging argument as before, we can show that with probability at least $1 - \frac{1}{100\log(\tau)}$, a uniformly random vertical block contains at most $\frac{100\sqrt{\tau}\log(\tau)}{\lambda^2}$ distinct values.

Therefore, in order to estimate the length of the LIS in the vertical chain, we sample $O(\log(\tau))$ subarrays $D_i, i \in [x]$ and run the pseudo-solution-based LIS estimation algorithm, restricted to the vertical box, if any, that belongs to this subarray while making sure that the success probability is at least $1 - \frac{1}{100\log\tau}$ and the error parameter is $\epsilon\lambda^2$. The details of how to implement this procedure nonadaptively are identical to how we implemented the estimation of the LIS in C_H in the preceding section. The query complexity is also identical.

▷ Claim 4.10. With probability $1 - O(\frac{\log(\tau)}{\tau^2})$, we estimate the contribution of vertical blocks to within an additive error of $\epsilon\lambda^2 n$.

4.1.4 Correctness, approximation guarantee, and query complexity

In this section, we complete the analysis of our algorithm and finish the proof of Theorem 1.5.

Success probability. The probability that any of Layering, Gridding and Finer Gridding fail is at most 3/100. For a specific chain of boxes, by Claims 4.9 and 4.10, we know that estimating the length of LIS within them is within the approximation guarantee with probability at least $1 - O(\frac{\log(\tau)}{\tau^2})$. By a union bound over all τ chains, we can see that the probability of incorrectly estimating the LIS length in some chain is at most 1/100. Thus, overall, the failure probability is at most a small constant.

Query complexity. The query complexity is clearly $\tilde{O}(\sqrt{r} \cdot \text{poly}(1/\lambda))$ from the description of the algorithm.

Approximation guarantee. Consider a fixed true LIS $\mathcal{L}$. The loss to $\mathcal{L}$ due to ignoring boxes that are not β-dense ($\beta = \epsilon^2\lambda$) is at most $\epsilon^3\lambda n + \epsilon\lambda n$. The loss to $\mathcal{L}$ due to antichain removal is at most $4n/\tau$, which is equal to $4\lambda n/5$. The resulting increasing sequence has length at least $|\mathcal{L}| - \epsilon^3\lambda n - \epsilon\lambda n - 4\lambda n/5$, which is at least $(1 - \epsilon^3 - \epsilon - 4/5) \cdot |\mathcal{L}|$, since, by our assumption $|\mathcal{L}| \geq \lambda n$. After chain decomposition, the length of the LIS in the best chain is at least $(1 - \epsilon^3 - \epsilon - 4/5) \cdot |\mathcal{L}|/\tau$, which is equal to $\frac{\lambda}{5} \cdot |\mathcal{L}| \cdot (1/5 - \epsilon^3 - \epsilon)$. Since we split the chains into horizontal and vertical chains, we further lose a factor of 2, and the resulting LIS length becomes $\frac{\lambda}{10} \cdot |\mathcal{L}| \cdot (1/5 - \epsilon^3 - \epsilon)$. In case of horizontal chains, we additionally lose a $9\epsilon\lambda^2 n$ and in the case of vertical chains, we additionally lose $\epsilon\lambda^2 n$. That is the length of LIS in the (best) horizontal chain is at least $\frac{\lambda}{10} \cdot |\mathcal{L}| \cdot (1/5 - \epsilon^3 - 11\epsilon)$. Finally, using Claims 4.9 and 4.10, we can see that we estimate the lengths of the best horizontal and vertical chains to within a constant multiplicative factor. Overall, the approximation guarantee is multiplicative $\Omega(\lambda)$.

References

1 Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures & Algorithms*, 31(3):371–383, 2007. doi:10.1002/rsa.20167.

2 David Aldous and Persi Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bull. Amer. Math. Soc.*, 34:413–432, 1999. doi:10.1090/S0273-0979-99-00796-X.

3 Aleksandrs Belovs. Adaptive lower bound for testing monotonicity on the line. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018*, pages 31:1–31:10, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.31.

4 Omri Ben-Eliezer, Clément L. Canonne, Shoham Letzter, and Erik Waingarten. Finding monotone patterns in sublinear time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1469–1494, 2019. doi:10.1109/FOCS.2019.000-1.

5 Piotr Berman, Sofya Raskhodnikova, and Grigory Yaroslavtsev. L_p-testing. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014*, pages 164–173. ACM, 2014. doi:10.1145/2591796.2591887.

6 Kashyap Dixit, Sofya Raskhodnikova, Abhradeep Thakurta, and Nithin Varma. Erasure-resilient property testing. *SIAM J. Comput.*, 47(2):295–329, 2018. doi:10.1137/16M1075661.

7 Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000.

8 Eldar Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004. doi:10.1016/j.ic.2003.09.003.

9 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discret. Math.*, 11(1):29–35, 1975. doi:10.1016/0012-365X(75)90103-X.

10 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

11 Michael Mitzenmacher and Saeed Seddighin. Improved sublinear time algorithm for longest increasing subsequence. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1934–1947. SIAM, 2021. doi:10.1137/1.9781611976465.115.

12 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. *Random Struct. Algorithms*, 55(2):402–426, 2019.

13 Ilan Newman and Nithin Varma. New sublinear algorithms and lower bounds for LIS estimation. *CoRR*, abs/2010.05805, 2021. arXiv:2010.05805.

14 Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and Nithin Varma. Parameterized property testing of functions. *ACM Trans. Comput. Theory*, 9(4):17:1–17:19, 2018. doi:10.1145/3155296.

15 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 6(72):1012–1042, 2006.

16 Sofya Raskhodnikova, Noga Ron-Zewi, and Nithin M. Varma. Erasures vs. errors in local decoding and property testing. In *Proceedings of the Innovations in Theoretical Computer Science Conference, (ITCS) 2019*, pages 63:1–63:21, 2019.

17 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

18 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1121–1145, 2019. doi:10.1109/FOCS.2019.00071.

19 Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM Journal on Computing*, 46(2):774–823, 2017.