Optimal Adaptive Detection of Monotone Patterns

Omri Ben-Eliezer* Shoham Letzter† Erik Waingarten‡

Abstract

We investigate adaptive sublinear algorithms for detecting monotone patterns in an array. Given fixed $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$, consider the problem of finding a length-k increasing subsequence in an array $f\colon [n] \to \mathbb{R}$, provided that f is ε-far from free of such subsequences. Recently, it was shown that the non-adaptive query complexity of the above task is $\Theta((\log n)^{\lfloor \log_2 k \rfloor})$. In this work, we break the non-adaptive lower bound, presenting an adaptive algorithm for this problem which makes $O(\log n)$ queries. This is optimal, matching the classical $\Omega(\log n)$ adaptive lower bound by Fischer [2004] for monotonicity testing (which corresponds to the case $k = 2$), and implying in particular that the query complexity of testing whether the longest increasing subsequence (LIS) has constant length is $\Theta(\log n)$.

*Tel-Aviv University, email: omrib@mail.tau.ac.il
†ETH Institute for Theoretical Studies, ETH Zurich, email: shoham.letzter@eth-its.ethz.ch
‡Columbia University, email: eaw@cs.columbia.edu

1 Introduction

For an integer $k \in \mathbb{N}$ and a function (or sequence) $f\colon [n] \to \mathbb{R}$, a *length-$k$ monotone subsequence of f* is a tuple of k indices, $(i_1, \ldots, i_k) \in [n]^k$, such that $i_1 < \cdots < i_k$ and $f(i_1) < \cdots < f(i_k)$. More generally, for a permutation $\pi\colon [k] \to [k]$, a *π-pattern of f* is given by a tuple of k indices $i_1 < \cdots < i_k$ such that $f(i_{j_1}) < f(i_{j_2})$ whenever $j_1, j_2 \in [k]$ satisfy $\pi(j_1) < \pi(j_2)$. A sequence f is π-free if there are no subsequences of f with order pattern π. Pattern avoidance and detection in an array is a central problem in theoretical computer science and combinatorics, dating back to the work of Knuth [Knu68] (from a computer science perspective), and Simion and Schmidt [SS85] (from a combinatorics perspective); see also the survey [Vat15]. Studying the computational problem from a *sublinear* algorithms perspective, Newman, Rabinovich, Rajendraprasad, and Sohler [NRRS17] initiated the study of property testing for forbidden order patterns in a sequence. For a fixed $k \in \mathbb{N}$ and a pattern π of length k, we want to test whether a function $f\colon [n] \to \mathbb{R}$ is π-free or ε-far from π-free.[1] They explicitly considered the monotone case as a particularly interesting instance; monotone patterns are naturally connected to monotonicity testing and the longest increasing subsequence and can shine new light on these classic problems. Note that being free of length-k monotone increasing subsequences is equivalent, as a simple special case of Dilworth's theorem [Dil50], to being *decomposable* into $k-1$ monotone non-increasing subsequences. The algorithmic task, which is the subject of this paper, is the following:

> *For $2 \le k \in \mathbb{N}$ and $\varepsilon > 0$, design a randomized algorithm that, given query access to a function $f\colon [n] \to \mathbb{R}$, distinguishes with probability at least $9/10$ between the case that f is free of length-k monotone subsequences and the case that it is ε-far from free of length-k monotone subsequences.*

This paper gives an algorithm with optimal dependence in n for solving the above problem. We state the main theorem next, and discuss connections to monotonicity testing and LIS shortly after.

Theorem 1.1. *Fix $k \in \mathbb{N}$. For any $\varepsilon > 0$, there exists an algorithm that, given query access to a function $f\colon [n] \to \mathbb{R}$ which is ε-far from $(12\ldots k)$-free, outputs a length-k monotone subsequence of f with probability at least $9/10$, with query complexity and running time[2] of $\mathrm{poly}(1/\varepsilon) \cdot \log n$.*

The algorithm underlying Theorem 1.1 is *adaptive*[3] and solves the testing problem with *one-sided error*,[4] since a length-k monotone subsequence is evidence for not being $(12\ldots k)$-free. The algorithm improves on a recent result of Ben-Eliezer, Canonne, Letzter and Waingarten [BECLW19] who gave an algorithm for finding length-k monotone patterns with query complexity $\mathrm{poly}(1/\varepsilon) \cdot (\log n)^{\lfloor \log_2 k \rfloor}$, which in itself improved upon a $\mathrm{poly}(1/\varepsilon) \cdot (\log n)^{O(k^2)}$ upper bound by Newman et al. [NRRS17]. The focus of [BECLW19] was on *non-adaptive* algorithms, and they gave a lower bound of $\Omega\big((\log n)^{\lfloor \log_2 k \rfloor}\big)$ queries for non-adaptive algorithms achieving one-sided error. Hence, Theorem 1.1 implies a natural separation between the power of adaptive and non-adaptive algorithms for finding monotone subsequences.

[1] A function f is ε-far from π-free if any π-free function g differs on a εn inputs.

[2] Generally, along the context of the introduction, we allow the $\mathrm{poly}(1/\varepsilon)$ term to depend on k; the precise bound we obtain here is $\big((k\log(1/\varepsilon))^k (1/\varepsilon)\big)^{O(k)} \cdot \log n$. See Lemma 3.2 for more details.

[3] An algorithm is *non-adaptive* if its queries do not depend on the answers to previous queries, or, equivalently, if all queries to the function can be made in parallel. Otherwise, if the queries of an algorithm may depend on the outputs of previous queries, then the algorithm is *adaptive*.

[4] An algorithm for testing property $\mathcal{P}$ has *one-sided error* if it has perfect completeness, i.e., it always outputs "yes" if $f \in \mathcal{P}$; otherwise, the algorithm is said to have two-sided error.

Theorem 1.1 is optimal, even among two-sided error algorithms. In the case $k = 2$, corresponding to monotonicity testing, there is a $\Omega(\log n/\varepsilon)$ lower bound[5] for both non-adaptive and adaptive algorithms [EKK+00, Fis04, CS14], even with two-sided error. A simple reduction suggested in [NRRS17] shows that the same lower bound (up to a multiplicative factor depending on k) holds for any fixed $k \geq 2$. Thus, an appealing consequence of Theorem 1.1 is that the natural generalization of monotonicity testing, which considers forbidden monotone patterns of fixed length longer than 2, does not affect the query complexity by more than a constant factor. Interestingly, Fischer [Fis04] shows that for any adaptive algorithm for monotonicity testing on $f\colon [n] \to \mathbb{R}$ there is a non-adaptive algorithm at least as good in terms of query complexity (even if we only restrict ourselves to one-sided error algorithms). That is, adaptivity does not help at all for $k = 2$. In contrast, the separation between our $O(\log n)$ adaptive upper bound and the $\Omega\big((\log n)^{\lfloor \log_2 k \rfloor}\big)$ non-adaptive lower bound of [BECLW19] implies that this is no longer true for $k \geq 4$.

As an immediate consequence, Theorem 1.1 gives an optimal testing algorithm for the longest increasing subsequence (LIS) problem in a certain regime. The classical LIS problem asks one to determine, given a sequence $f\colon [n] \to \mathbb{R}$, what is the maximum k for which f contains a length-k increasing subsequence. It is very closely related to other fundamental algorithmic problems in sequences, like the computation of edit distance, Ulam distance, or distance from monotonicity (for example, the latter equals n minus the LIS length), and was thoroughly investigated from the perspective of sublinear-time algorithms [PRR06, ACCL07, SS17, RSSS19] and streaming algorithms [GJKK07, SW07, GG10, SS13, EJ15, NS15]. In the property testing regime, the corresponding decision task is to distinguish between the case where f has LIS length at most k (where k is given as part of the input) and the case that f is ε-far from having such a LIS length. Theorem 1.1 in combination with the aforementioned lower bounds (which readily carry on to this setting) yield a tight bound on the query complexity of testing whether the LIS length is a constant.

Corollary 1.2. *Fix $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$. The query complexity of testing whether $f\colon [n] \to \mathbb{R}$ has LIS length at most k is $\Theta(\log n)$.*

1.1 Related Work

Considering general permutations π of length k and *exact* computation, Guillemot and Marx [GM14] showed how to find a π-pattern in a sequence f in time $2^{O(k^2 \log k)}n$, later improved by Fox [Fox13] to $2^{O(k^2)}n$. In the regime $k = \Omega(\log n)$, an algorithm of Berendsohn, Kozma, and Marx [BKM19] provides the state-of-the-art.

For *approximate* computation of general patterns π, the works of [NRRS17, BC18] investigate the query complexity of property testing for forbidden order patterns. When π is of length 2, the problem considered is equivalent to testing monotonicity, one of the most widely-studied problems in property testing, with works spanning the past two decades. Over the years, variants of monotonicity testing over various partially ordered sets have been considered, including the line $[n]$ [EKK+00, Fis04, Bel18, PRV18, Ben19], the Boolean hypercube $\{0,1\}^d$ [DGL+99, BBM12, BCGSM12, CS13, CST14, CDST15, KMS15, BB15, CS16, CWX17, CS19], and the hypergrid $[n]^d$ [BRY14, CS14, BCS18]. We refer the reader to [Gol17, Chapter 4] for more on monotonicity testing, and a general overview of the field of property testing (introduced in [RS96, GGR98]).

Understanding the power of adaptivity seems to be a notoriously difficult problem in property testing. In the context of testing for forbidden order patterns, non-adaptive algorithms are rather weak: the non-adaptive

[5] The precise lower bound is of the form $\Omega(\log(\varepsilon n)/\varepsilon)$, and is equivalent to the aforementioned one as long as, say, $\varepsilon > n^{-0.99}$.

query complexity is $\Omega(n^{1/2})$ for all non-monotone order patterns [NRRS17], and as high as $n^{1-1/(k-\Theta(1))}$ for most order patterns of length k [BC18]. Prior to our work (which shows separation between adaptive and non-adaptive algorithms for monotone patterns), the only case for which adaptive algorithms were known to outperform their non-adaptive counterparts have been for patterns of length 3 in [NRRS17], and an intriguing conjecture from the same paper suggests that in fact, the query complexity for testing π-freeness is polylogarithmic in n for *any* fixed-length π – depicting an exponential separation from the non-adaptive case (for non-monotone patterns).

1.2 Main Ideas and Techniques

We now describe the intuition behind the proof of Theorem 1.1. There are two main technical components: 1) a new structural result for functions $f\colon [n] \to \mathbb{R}$ with many length-k monotone subsequences which strengthens a theorem of [BECLW19], and 2) new (adaptive) algorithmic components which lead to the $O(\log n)$-query algorithm. We start by explaining the $(\log n)^{O(k^2)}$ upper bound of Newman et al. [NRRS17] and the structural decomposition of [BECLW19].

Fix $k \in \mathbb{N}$ and $\varepsilon > 0$, and suppose that $f\colon [n] \to \mathbb{R}$ is ε-far from $(12\dots k)$-free, that is, ε-far from free of length-k increasing subsequences. Notice that f must contain a collection $\mathcal{C}$ of at least $\varepsilon n/k$ pairwise-disjoint increasing subsequences of length k.[6] For simplicity, consider $k = 2$ first (which corresponds to the classical problem of monotonicity testing). For any $x < \ell < y \in [n]$, we say that ℓ *cuts the pair* (x, y) *with slack* if $x + (y-x)/3 \le \ell \le y - (y-x)/3$, or, informally, if ℓ lies roughly "in the middle" of x and y. Additionally, the *width* of the pair (x, y) is $\lfloor \log(y-x) \rfloor$. Define the collection of copies from $\mathcal{C}$ of width w around ℓ by

$$\mathcal{C}_{\ell,w} = \{(x,y) \in \mathcal{C} : \text{width}(x,y) = w,\ \ell \text{ cuts } (x,y) \text{ with slack}\}.$$

Finally, the *density* of copies from $\mathcal{C}$ of width w around ℓ, and the total density of $\mathcal{C}$ around ℓ, are defined by

$$\tau_{\mathcal{C}}(\ell, w) = \frac{1}{2^w} \cdot |\mathcal{C}_{\ell,w}| \ ; \qquad \tau_{\mathcal{C}}(\ell) = \sum_{w=1}^{\log n} \tau_{\mathcal{C}}(\ell, w).$$

A polylogarithmic-query algorithm. Fix a location $\ell \in [n]$ and a width $w \in [\log n]$, and consider drawing $\Theta(1/\tau_{\mathcal{C}}(\ell, w))$ indices from the interval $[\ell - 2^w, \ell + 2^w]$ uniformly at random, querying f in all of these locations. Letting m be the median of the set $\{f(x) : (x,y) \in \mathcal{C}_{\ell,w}\}$, if we manage to query the "1-entry" x of some $(x,y) \in \mathcal{C}_{\ell,w}$ where $f(x) \le m$, and the "2-entry" y' of some $(x',y') \in \mathcal{C}_{\ell,w}$ where $f(x') \ge m$, then (x, y') would form a valid (12)-pattern, since $x < \ell < y'$ and $f(x) \le m \le f(x') \le f(y')$. By definition, the number of entries x as well as the number of entries y' within $[\ell - 2^w, \ell + 2^w]$ which may be sampled is at least $\Omega\left(\tau_{\mathcal{C}}(\ell, w) \cdot 2^w\right)$. Therefore, with good probability, $\Theta(1/\tau_{\mathcal{C}}(\ell, w))$ uniform queries from the interval will hit at least one such x and one y', which would form the desired (12)-pattern.

We claim that many values of $\ell \in [n]$ have some width $w \in [\log n]$ where the density $\tau_{\mathcal{C}}(\ell, w)$ is large. First, a simple double counting argument shows $\mathbb{E}_{\ell \in [n]}[\tau_{\mathcal{C}}(\ell)] = \Omega(\varepsilon)$. On the other hand, $\tau_{\mathcal{C}}(\ell, w) \le O(1)$ for any width $w \in [\log n]$, and so $\tau_{\mathcal{C}}(\ell) = O(\log n)$. Consequently, the probability that a random $\ell \in [n]$ satisfies $\tau_{\mathcal{C}}(\ell) = \Omega(\varepsilon)$ is $\Omega(\varepsilon / \log n)$. It suffices to pick $\Theta(\log n/\varepsilon)$ uniformly random $\ell \in [n]$ in order for one to

[6] Otherwise, greedily eliminating these subsequences gives a $(12\dots k)$-free function differing in strictly less than εn inputs.

satisfy $\tau_{\mathcal{C}}(\ell) = \Omega(\varepsilon)$ with high probability; and, if this event holds, then there exists $w \in [\log n]$ for which $\tau_{\mathcal{C}}(\ell, w) = \Omega(\varepsilon/\log n)$. We now leverage the querying paradigm described in the previous paragraph: if for any $\ell \in [n]$ as above and any $w \in [\log n]$ we query $\approx \log n/\varepsilon$ uniform locations in $[\ell - 2^w, \ell + 2^w]$, then we shall find a (12)-pattern with good probability. In total, this procedure makes $O(\log^3 n/\varepsilon^2)$ non-adaptive queries.

To deal with general fixed $k \geq 2$ and $\varepsilon > 0$, the (essentially) same reasoning is applied recursively, leading to the $(\log n)^{O(k^2)}$-query algorithm of [NRRS17].

Structural decomposition. [BECLW19] established a structural theorem for functions $f\colon [n] \to \mathbb{R}$ that are ε-far from $(12\dots k)$-free, which led to improved non-adaptive algorithms. Specifically, they show that any f which is ε-far from $(12\dots k)$-free satisfies at least one of two conditions: either f contains many *growing suffixes*, or it can be decomposed into *splittable intervals*. For the purpose of this discussion, let $\mathcal{C}$ be any collection of $\Theta_{k,\varepsilon}(n)$ disjoint $(12\dots k)$-copies in f.[7]

- **Growing suffixes:** there exist $\Omega_{k,\varepsilon}(n)$ values of $\ell \in [n]$ where[8] $\tau_{\mathcal{C}}(\ell) \geq \Theta_k(\varepsilon)$ and $\tau_{\mathcal{C}}(\ell, w) \leq \Theta(\tau_{\mathcal{C}}(\ell)/k)$ for every $w \in [\log n]$. In other words, many $\ell \in [n]$ have that the sum of local densities, $\tau_{\mathcal{C}}(\ell)$ of $(12\dots k)$-patterns in intervals of growing widths is not too small, and furthermore, the densities are not concentrated on any small set of widths w. Any such ℓ is said to be the starting point of a growing suffix.
- **Splittable intervals:** there exist $c \in [k-1]$ and a collection of pairwise-disjoint intervals $I_1, \dots, I_s \subset [n]$ with $\sum_{i=1}^{s} |I_i| = \Theta_{k,\varepsilon}(n)$, so that each I_i contains a dense collection of disjoint $(12\dots k)$-patterns of a particular structure. Specifically, each such interval I_i can be partitioned into three disjoint intervals L_i, M_i, R_i (in this order), each of size $\Omega_k(|I_i|)$, where I_i fully contains $\Omega_{k,\varepsilon}(|I_i|)$ disjoint copies of $(12\dots k)$-patterns, in which the first c entries lie in L_i, and the last $k-c$ entries lie in R_i (none of these entries lies in M_i).

[BECLW19] proceeds by devising an $O_{k,\varepsilon}(\log n)$-query non-adaptive algorithm for the growing suffixes case, and an $O_{k,\varepsilon}((\log n)^{\lfloor \log_2 k \rfloor})$-query non-adaptive algorithm for the splittable intervals case. Thus, in order to obtain an $O_{k,\varepsilon}(\log n)$-query *adaptive* algorithm, it suffices to develop such an algorithm under the splittable intervals assumption.

Robustifying the structural decomposition. The splittable intervals condition, however, does not seem strong enough for our purposes: in order to utilize it, one would seemingly have to "identify", in some way, which parts of our sequence constitute splittable intervals, which is not clear how to do efficiently. In order to bypass this issue, we substantially strengthen the structural theorem. The stronger statement asserts that any $f\colon [n] \to \mathbb{R}$ that is ε-far from $(12\dots k)$-free either satisfies the growing suffixes condition, defined previously, or a *robust* version of the splittable intervals condition, defined as follows.

[7] To simplify the discussion, in the rest of this exposition we will generally not be interested in the exact dependence on the parameters k and ε, and for convenience we often use notions like $O_{k,\varepsilon}(\cdot)$ and $\Omega_{k,\varepsilon}(\cdot)$ that hide this dependence.

[8] We have previously defined the notions of cutting with slack and density only for the case $k = 2$, but they generalize rather naturally to any k. First, define the *gap index* of a $(12\dots k)$-pattern in f in locations $x_1 < \dots < x_k \in [n]$ as the smallest integer $c \in [k-1]$ maximizing $x_{c+1} - x_c$; the above copy is cut by ℓ with slack if $x_i + (x_{c+1} - x_c)/3 \leq \ell \leq x_{c+1} - (x_{c+1} - x_c)/3$. The gap-width of the copy is $\log(x_{c+1} - x_c)$. The definitions of $\mathcal{C}_{\ell,w}$, $\tau_{\mathcal{C}}(\ell, w)$, $\tau_{\mathcal{C}}(\ell)$ can then be generalized in a straightforward manner, replacing "width" with "gap width" wherever relevant.

- **Robust splittable intervals:** there exist $c \in [k-1]$ and a collection of pairwise-disjoint intervals $I_1, \ldots, I_s \subset [n]$ satisfying the same properties as in the "splittable intervals" setting described above (with slightly different dependence on ε and k in the $\Theta_{k,\varepsilon}(\cdot)$ term). Additionally, *any* interval $J \subset [n]$ which contains an interval I_j is itself far from $(12\ldots k)$-free, i.e. it contains a collection of $\Omega_{k,\varepsilon}(|J|)$ disjoint $(12\ldots k)$-copies.

Towards an algorithm. At a high level, the algorithms of [BECLW19] and [NRRS17] proceed in a recursive manner where each step tries to find the relevant width considered (which is one of $\Omega(\log n)$ options). Since their algorithms are non-adaptive, they consider all $\Omega(\log n)$ options in recursive steps, and hence, suffer a logarithmic factor with each step. Since our algorithm is adaptive, we want to choose *one* of the widths to recurse on. The algorithm will ensure that the width considered is large enough. When the width chosen is not too much larger, our recursive step proceeds similarly to [NRRS17]; we call this the *fitting case*. However, the width considered may be too large; we call this case *overshooting*. In order to deal with the overshooting case, we algorithmically utilize the robust structural theorem in a somewhat surprising manner in order to detect a $(12\ldots k)$-copy.

We now expand on the above idea and provide an informal description. As [BECLW19] gives an $O_{k,\varepsilon}(\log n)$-query algorithm when our function f satisfies the growing suffixes condition, we may assume that f satisfies the *robust splittable intervals* condition. Consider sampling, for $O_{k,\varepsilon}(1)$ repetitions, an index $\boldsymbol{x} \in [n]$ uniformly at random, and for each $t \in [\log n]$, a random index $\boldsymbol{y}_t \in [\boldsymbol{x}, \boldsymbol{x} + 2^t]$. Consider the following event:

> The index $\boldsymbol{x}$ is a (sufficiently well-behaved)[9] first element in some $(12\ldots k)$-pattern falling in some robust splittable interval I_j, and for $t^* \in [\log n]$ satisfying $|I_j| \leq 2^{t^*} \leq 2|I_j|$, $\boldsymbol{y}_{t^*}$ is a (well-behaved) $(c+1)$-th element in some $(12\ldots k)$-pattern falling in I_j.

We claim that the above event occurs with high (constant) probability for at least one choice of $\boldsymbol{x}$, and that when this event does occur, the algorithm can be recursively applied without incurring a multiplicative logarithmic factor. Indeed, suppose that the above holds for some $\boldsymbol{x}$.[10] We set $\boldsymbol{y}$ to be $\boldsymbol{y}_t$, where t is the largest such that $f(\boldsymbol{x}) < f(\boldsymbol{y}_t)$ holds, and notice in particular that $t \geq t^*$. This means that $\boldsymbol{x} < \boldsymbol{y}$ and $f(\boldsymbol{x}) < f(\boldsymbol{y})$.

The *fitting case* occurs when t (achieving the maximum above) is roughly the same as t^*. To handle this case, we recurse by finding a $(12\ldots c)$-patterns in L_j, and $(12\ldots(k-c))$-pattern in R_j. At a high level, if one takes $\Theta_{k,\varepsilon}(1)$ independent uniform samples $\boldsymbol{z}$ from $[\boldsymbol{x}, \boldsymbol{y}]$, then one of them is likely to fall in the middle part M_j of I_j, so that $L_j \subset [\boldsymbol{x} - 2^t, \boldsymbol{z}]$ and $R_j \subset [\boldsymbol{z}, \boldsymbol{y} + 2^t]$, allowing us to proceed recursively. While this description omits a few details, the intuition proceeds similarly to [NRRS17], except that the recursion occurs only on one width, namely, t, and does not lose multiplicative logarithmic factors as in the previous approaches.

[9] Recall that, in the first polylogarithmic-query qlgorithm described above, we hoped to hit a "1-entry" x whose value $f(x)$ is no higher than some suitable median value; the "well-behaved" requirements are of similar flavor, and do not incur more than a constant overhead on the query complexity.

[10] More precisely, our algorithm runs this procedure for any of our choices of $\boldsymbol{x}$, without "knowing" which of them satisfies the above event. Since the total number of choices is $O_{k,\varepsilon}(1)$, this incurs only a constant overhead.

The overshooting component. The other case, of *overshooting*, occurs when t is significantly larger than t^*. We expand on the main ideas here in more detail; the strong guarantee given by the robust splittable intervals condition adds a "for all" element into the structural characterization, which is able to treat the problem posed by overshooting in a rather surprising and non-standard way. Since t is much larger than $\log |I_j|$, there exist $k-2$ intervals $J_1, \ldots, J_{k-2} \subset [\boldsymbol{x}, \boldsymbol{y}]$ satisfying the following conditions:

- J_1 lies immediately after the interval I_j (recall that I_j is the interval containing $\boldsymbol{x}$).
- J_{i+1} lies immediately after J_i, for any $i \in [k-3]$.
- $|J_1| = \lceil |I_j| \cdot \alpha_{k,\varepsilon} \rceil$ and $|J_{i+1}| = \lceil |J_i| \cdot \alpha_{k,\varepsilon} \rceil$ for any $i \in [k-3]$, for some large enough $\alpha_{k,\varepsilon} > 1$.

For any $i \in [k-2]$, set J'_i to be the minimal interval containing both I_j and J_i. The robust splittable intervals condition asserts that (since each J'_i contains the splittable interval I_j) the number of disjoint $(12\ldots k)$-copies in J'_i is proportional to $|J'_i|$, and provided that $\alpha_{k,\varepsilon}$ is large enough, this means that $J_i = J'_i \setminus J'_{i-1}$ also contains a collection $\mathcal{T}_i$ of $\Omega_{k,\varepsilon}(|J_i|)$ disjoint $(12\ldots k)$-copies. We now define two sets $\mathcal{A}_i$ and $\mathcal{B}_i$ as follows. Let $\mathcal{A}_i$ be the collection of prefixes $(a_1, \ldots, a_{i+1})$ of k-tuples from $\mathcal{T}_i$ with $f(a_{i+1}) < f(y)$, and let $\mathcal{B}_i$ be the collection of suffixes $(a_{i+1}, \ldots, a_k)$ of k-tuples from $\mathcal{T}_i$ with $f(a_{i+1}) \geq f(y)$. As $|\mathcal{T}_i| = |\mathcal{A}_i| + |\mathcal{B}_i|$, one of $\mathcal{A}_i$ and $\mathcal{B}_i$ is large (i.e. has size at least $\Omega_{k,\varepsilon}(|J_i|)$).

This seemingly innocent combinatorial idea can be exploited non-trivially to find a $(12\ldots k)$-copy. Specifically, the algorithm to handle overshooting aims to find (recursively) shorter increasing subsequences in $J_1, \ldots, J_{k-2}$, with the hope of combining them together into a $(12\ldots k)$-copy. Concretely, for any $i \in [k-2]$, we make two recursive calls of our algorithm on J_i: one for a $(k-i)$-increasing subsequence in J_i whose values are at least $f(y)$,[11] and a second for an $(i+1)$-increasing subsequence in J_i, with values smaller than $f(y)$. By induction, the first recursive call succeeds with good probability if $|\mathcal{A}_i|$ is large, while the second call succeeds with good probability if $|\mathcal{B}_i|$ is large. Since for any i either $|\mathcal{A}_i|$ or $|\mathcal{B}_i|$ must be large, at least one of the following must hold.

- $\mathcal{B}_1$ is large. In this case we are likely to find a length-$(k-1)$ monotone pattern in J_1 with values at least $f(y) > f(x)$, which combines with $\boldsymbol{x}$ to form a length-k monotone pattern.
- $\mathcal{A}_{k-2}$ is large. Here we are likely to find a length-$(k-1)$ monotone pattern in J_{k-2} whose values lie below $f(y)$, which combines with $\boldsymbol{y}$ to form a length-k monotone pattern.
- There exists $i \in [k-3]$ where both $\mathcal{A}_i$ and $\mathcal{B}_{i+1}$ are large. Here we will find, with good probability, a length-$(i+1)$ monotone pattern in J_i with values below $f(y)$, and a length-$(k-i-1)$ monotone pattern in J_{i+1} with values above $f(y)$; together these two patterns combine to form a $(12\ldots k)$-pattern.

In all cases, a k-increasing subsequence is found with good probability.

Finally, for the query complexity, our algorithm (which runs both the "fitting" component and the "overshooting" component, to address both cases) makes $O_{k,\varepsilon}(\log n)$ queries: each call makes $O_{k,\varepsilon}(\log n)$ queries in itself and $O_{k,\varepsilon}(1)$ additional calls recursively, where the recursion depth is bounded by k. It follows that the total query complexity is of the form $O_{k,\varepsilon}(\log n)$.

[11] Technically speaking, our algorithm can be configured to only look for increasing subsequences whose values lie in some range; we use this to make sure that shorter increasing subsequences obtained from the recursive calls of the algorithm can eventually be concatenated into a valid length-k one.

1.3 Notation

All logarithms considered are base 2. We consider functions $f\colon I \to \mathbb{R}$, where $I \subseteq [n]$, as the inputs and main objects of study. An *interval* in $[n]$ is a set $I \subseteq [n]$ of the form $I = \{a, a+1, \ldots, b\}$. At many places throughout the paper, we think of augmenting the image with a special character $*$ to consider $f\colon I \to \mathbb{R} \cup \{*\}$. $*$ can be thought of as a *masking* operation: In many cases, we will only be interested in entries x of f so that $f(x)$ lies in some prescribed (known in advance) range of values $R \subseteq \mathbb{R}$, so that entries outside this range will be marked by $*$. Whenever the algorithm queries $f(x)$ and observes $*$, it will interpret this as an incomparable value (with respect to ordering) in $\mathbb{R}$. As a result, $*$-values will never be part of monotone subsequences. We note that augmenting the image with $*$ was unnecessary in [NRRS17, BECLW19] because they only considered non-adaptive algorithms. We say that for a fixed $f\colon I \to \mathbb{R} \cup \{*\}$, the set T is a collection of disjoint monotone subsequences of length k if it consists of tuples $(i_1, \ldots, i_k) \in I^k$, where $i_1 < \cdots < i_k$ and $f(i_1) < \cdots < f(i_k)$, and furthermore, for any two tuples $(i_1, \ldots, i_k)$ and $(i'_1, \ldots, i'_k)$, their intersection (as sets) is empty. We also denote $E(T)$ as the union of indices in k-tuples of T, i.e., $E(T) = \cup_{(i_1,\ldots,i_k)\in T}\{i_1, \ldots, i_k\}$. Finally, we let $\mathrm{poly}(\cdot)$ denote a large enough polynomial whose degree is (bounded by) a universal constant.

2 Stronger structural dichotomy

In this section, we establish the structural foundations – specifically, the *growing suffixes* versus *robust splittable intervals* dichotomy – lying at the heart of our adaptive algorithm. We start with the definitions. The first is the definition of a growing suffix setting, as given in [BECLW19]. For what follows, for an index $\ell \in [n]$ define $\eta_\ell = \lceil \log_2(n-\ell) \rceil$, and for any $t \in [\eta_\ell]$ set $S_t(\ell) = [a+2^{t-1}, a+2^t) \cap [n]$. Note that the intervals $S_1, \ldots, S_{\eta_\ell}$ are a partition of $(\ell, n]$ into intervals of exponentially increasing length (except for maybe the last one). Finally, the tuple $S(\ell) = (S_t(\ell))_{t\in[\eta_\ell]}$ is called the *growing suffix* starting at ℓ.

Definition 2.1 (Growing suffixes (see [BECLW19], Definition 2.4))**.** *Let $\alpha, \beta \in [0,1]$. We say that an index $\ell \in [n]$ starts an (α,β)-*growing suffix *if, when considering the collection of intervals $S(\ell) = \{S_t(\ell) : t \in [\eta_\ell]\}$, for each $t \in [\eta_\ell]$ there is a subset $D_t(\ell) \subseteq S_t(\ell)$ of indices such that the following properties hold.*

1. *We have $|D_t(\ell)|/|S_t(\ell)| \le \alpha$ for all $t \in [\eta_\ell]$, and $\sum_{t=1}^{\eta_\ell} |D_t(\ell)|/|S_t(\ell)| \ge \beta$.*
2. *For every $t, t' \in [\eta_a]$ where $t < t'$, if $a \in D_t(\ell)$ and $a' \in D_{t'}(\ell)$, then $f(a) < f(a')$.*

The second definition, also from [BECLW19], describes the (non-robust) splittable intervals setting.

Definition 2.2 (Splittable intervals (see [BECLW19], Definition 2.5))**.** *Let $\alpha, \beta \in (0,1]$ and $c \in [k-1]$. Let $I \subseteq [n]$ be an interval, let $T \subseteq I^k$ be a set of disjoint, length-k monotone subsequences of f lying in I, and define*

$$T^{(L)} = \{(i_1, \ldots, i_c) \in I^c : (i_1, \ldots, i_c) \text{ is a prefix of a } k\text{-tuple in } T\}, \text{ and}$$
$$T^{(R)} = \{(j_1, \ldots, j_{k-c}) \in I^{k-c} : (j_1, \ldots, j_{k-c}) \text{ is a suffix of a } k\text{-tuple in } T\}.$$

We say that the pair (I,T) is (c,α,β)-splittable *if $|T|/|I| \geq \beta$; $f(i_c) < f(j_1)$ for every $(i_1,\ldots,i_c) \in T^{(L)}$ and $(j_1,\ldots,j_{k-c}) \in T^{(R)}$; and there is a partition of I into three adjacent intervals $L,M,R \subseteq I$ (that appear in this order, from left to right) of size at least $\alpha|I|$, satisfying $T^{(L)} \subseteq L^c$ and $T^{(R)} \subseteq R^{k-c}$.*

A collection of disjoint interval-tuple pairs $(I_1,T_1),\ldots,(I_s,T_s)$ is called a (c,α,β)-splittable collection of T *if each (I_j,T_j) is (c,α,β)-splittable and the sets $(T_j : j \in [s])$ partition T.*

The following theorem presents the growing suffixes versus (non-robust) splittable intervals dichotomy, which is among the main structural results of [BECLW19].[12]

Theorem 2.3 ([BECLW19])**.** *Let $k,n \in \mathbb{N}$, $\varepsilon \in (0,1)$, and $C > 0$, and let $I \subseteq [n]$ be an interval. Let $f\colon I \to \mathbb{R} \cup \{*\}$ be a function and let $T^0 \subseteq I^k$ be a set of at least $\varepsilon|I|$ disjoint monotone subsequences of f of length k. Then there exist $\alpha \in (0,1)$ and $p > 0$ satisfying $\alpha \geq \Omega(\varepsilon/k^5)$ and $p \leq \mathrm{poly}(k\log(1/\varepsilon))$ such that at least one of the following conditions holds.*

1. ***Growing suffixes:*** *There exists a set $H \subseteq [n]$, of indices that start an $(\alpha, Ck\alpha)$-growing suffix, satisfying $\alpha|H| \geq (\varepsilon/p)n$.*
2. ***Splittable intervals (non-robust):*** *There exist an integer c with $1 \leq c < k$, a set T, with $E(T) \subseteq E(T^0)$, of disjoint length-k monotone subsequences, and a $(c,1/(6k),\alpha)$-splittable collection of T, consisting of disjoint interval-tuple pairs $(I_1,T_1),\ldots,(I_s,T_s)$, such that*

$$\alpha \sum_{h=1}^{s} |I_h| \geq |T^0|/p. \tag{1}$$

As argued in Section 1.2, the splittable intervals condition does not seem strong enough by itself to be useful for adaptive algorithms. Therefore, we next aim to establish a stronger structural dichotomy, asserting that f either satisfies the growing suffixes condition, or a *robust* version of the splittable intervals condition. The next lemma will imply that the growing suffixes condition can be robustified by merely throwing away a subset of "bad" splittable intervals.

Lemma 2.4. *Let $\alpha \in (0,1)$ and let $I \subset \mathbb{N}$ be an interval. Suppose that $I_1,\ldots,I_s \subset I$ are disjoint intervals such that $\sum_{h=1}^{s} |I_h| \geq \alpha|I|$. Then there exists a set $G \subset [s]$ such that*

$$\sum_{h\in G} |I_h| \geq (\alpha/4)|I|,$$

and for every interval $J \subset I$ that contains an interval I_h with $h \in G$,

$$\sum_{h\in[s]\colon\, I_h\subset J} |I_h| \geq (\alpha/4)|J|.$$

Proof. Let $B \subseteq [s]$ be the set of indices h for which there is an interval $J_h \supseteq I_h$ satisfying $\sum_{h\in[s]:I_h\subseteq J} |I_h| < (\alpha/4)|J|$. For each $h \in B$ fix such a containing interval $J(I_h)$.

Let $\mathcal{J}$ be a minimal subset of $\{J(I_h) : h \in B\}$ with the following property: for any $h \in B$ there exists $J \in \mathcal{J}$ containing I_h. Such a minimal subset clearly exists, since $\{J(I_h) : h \in B\}$ itself satisfies this property (but

[12] In [BECLW19], the theorem is stated with respect to two parameters, k, k_0. For our purpose it suffices to set $k_0 = k$.

is not necessarily minimal). The next claim asserts that no vertex is covered more than three times by sets in $\mathcal{J}$.

Claim 2.5. *Every element $x \in I$ is contained in at most three intervals from $\mathcal{J}$.*

Proof. The proof follows from the minimality of $\mathcal{J}$. Consider first the case where $x \in I_{h^*}$ for some $h^* \in B$. Let $J_L = [a_L, b_L]$ be an interval from $\mathcal{J}$ that contains x, and whose left-most element a_L is furthest to the left among all intervals from $\mathcal{J}$ that contain x; pick $J_R = [a_R, b_R]$ symmetrically, with b_R being furthest possible to the right; and let $J_M = [a_M, b_M]$ be an interval from $\mathcal{J}$ that contains I_h. We claim that $\mathcal{J}$ does not have any other intervals that contain x. Suppose, to the contrary, that there exists $J = [a, b] \in \mathcal{J}$ containing x where $J \neq J_L, J_R, J_M$; note that by definition of J_L and J_M, $a_L \leq a$ and $b_R \geq b$.

We claim that $\mathcal{J} \setminus \{J\}$ covers all intervals I_h with $h \in [B]$; it suffices to show that for any $h \in B$ such that $I_h \subset J$, one of the intervals J_L, J_R, J_M covers I_h. Consider $h \in B$ such that $I_h \subset J$, and write $I_h = [c, d]$. If $h = h^*$, then $I_h \subset J_M$. If I_h lies to the left of I_{h^*}, then $d < x \leq b_L$, and $c \geq a \geq a_L$, so $I_h \subseteq J_L$. Similarly, if I_h lies to the right of I_h, then $I_h \subseteq J_R$. It follows that, indeed, intervals from $\mathcal{J} \setminus \{J\}$ cover all intervals in $\{I_h : h \in B\}$, contradicting the minimality of $\mathcal{J}$.

Now, if x is not contained in any interval of I_h with $h \in B$, then we can show similarly that there are at most two intervals from $\mathcal{J}$ that contain x, by defining J_L and J_R as above. □

Let U be the union of intervals from $\mathcal{J}$. In light of the above claim,

$$\sum_{h \in B} |I_h| \leq \sum_{J \in \mathcal{J}} \left(\sum_{h \in [s] \colon I_h \subseteq J} |I_h| \right) < \frac{\alpha}{4} \cdot \sum_{J \in \mathcal{J}} |J| \leq \frac{3\alpha}{4} \cdot |U| \leq \frac{3\alpha}{4} \cdot |I|,$$

where the first inequality holds because each I_h with $h \in B$ is covered by an interval in $\mathcal{J}$; the second inequality follows as $\mathcal{J}$ consists of sets $J(I_h)$ with $h \in B$; the third inequality follows from the claim; and the last one holds because $U \subset I$. Finally, let $G = [s] \setminus [B]$. By assumption on $\sum_h |I_h|$ and the previous line,

$$\sum_{h \in G} |I_h| = \sum_{h \in [s]} |I_h| - \sum_{h \in B} |I_h| \geq \alpha |I| - \frac{3\alpha}{4} \cdot |I| = \frac{\alpha}{4} \cdot |I|,$$

and every interval J that contains an interval I_h with $h \in G$ satisfies $\sum_{h \in [s] \colon I_h \subset J} |I_h| \geq (\alpha/4)|J|$, as required. □

The robust version of the structural dichotomy is stated below; the proof follows easily from the basic structural dichotomy in combination with the last lemma.

Theorem 2.6 (Robust structural theorem)**.** *Let $k, n \in \mathbb{N}$, $\varepsilon \in (0, 1)$, and $C > 0$, and let $I \subseteq [n]$ be an interval. Let $f \colon I \to \mathbb{R} \cup \{*\}$ be an array and let $T^0 \subseteq I^k$ be a set of at least $\varepsilon |I|$ disjoint length-k monotone subsequences of f. Then there exist $\alpha \in (0, 1)$ and $p > 0$ with $\alpha \geq \Omega(\varepsilon / k^5)$ and $p \leq \mathrm{poly}(k \log(1/\varepsilon))$ such that at least one of the following holds.*

1. ***Growing suffixes:*** *There exists a set $H \subseteq [n]$, of indices that start an $(\alpha, Ck\alpha)$-growing suffix, satisfying $\alpha |H| \geq (\varepsilon / p) n$.*

2. ***Robust splittable intervals:*** *There exist an integer c with $1 \le c < k$, a set T, with $E(T) \subseteq E(T^0)$, of disjoint length-k monotone subsequences, and a $(c, 1/(6k), \alpha)$-splittable collection of T, consisting of disjoint interval-tuple pairs $(I_1, T_1), \ldots, (I_s, T_s)$, such that*

$$\alpha \sum_{h=1}^{s} |I_h| \ge (\varepsilon/p)|I|, \tag{2}$$

Moreover, if $J \subset I$ is an interval where $J \supset I_h$ for some $h \in [s]$, J contains at least $(\varepsilon/p)|J|$ disjoint $(12\ldots k)$-patterns from T^0.

Proof. Apply Theorem 2.3. Let $\alpha^* \in (0,1)$ and p^* be parameters such that $\alpha^* \ge \Omega(\varepsilon/k^5)$ and $p^* \le \mathrm{poly}(k \log(1/\varepsilon))$, as guaranteed by the theorem. Set $\alpha = \alpha^*$ and $p = 4p^*$. If Condition 1 holds in the application of Theorem 2.3, then the analogous growing suffix condition in Theorem 2.6 clearly holds. So suppose that Condition 2 in Theorem 2.3 holds, and let c and $(I_1, T_1), \ldots, (I_s, T_s)$ be as guaranteed there. In particular, we have $\sum_{h=1}^{s} |I_h| \ge (1/p^*\alpha^*)|T^0|$. By Lemma 2.4, there is a subset $G \subset [s]$ such that $\sum_{h \in G} |I_h| \ge (1/4p^*\alpha^*)|T^0| \ge (\varepsilon/4p^*\alpha^*)|I| = (\varepsilon/p\alpha)|I|$; and, for every interval J in I that contains an interval I_h with $h \in [G]$, $\sum_{h \in [s]:\ I_h \subset J} |I_h| \ge (\varepsilon/4p^*\alpha^*)|J|$. Since each I_h contains at least $\alpha^*|I_h|$ disjoint length-k increasing subsequences, it follows that J contains at least $(\varepsilon/4p^*)|J| = (\varepsilon/p)|J|$ length-k increasing subsequences. Taking T to be the union of T_h over $h \in G$, along with the pairs (I_h, T_h) with $h \in G$, we obtain the required robust splittable intervals. □

3 The Algorithm

Our aim in this section is to prove the existence of a randomized algorithm, $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$, that receives as input a function $f\colon I \to \mathbb{R} \cup \{*\}$ (where $I \subset \mathbb{N}$ is an interval), and parameters $\varepsilon, \delta \in (0,1)$, and satisfies the following: if f contains $\varepsilon|I|$ disjoint $(12\ldots k)$-patterns, then the algorithm outputs such a pattern with probability at least $1-\delta$; and the running time of the algorithm is $O_{k,\varepsilon}(\log n)$. To this end, we describe such an algorithm in Figure 3 below. This algorithm uses three subroutines: `Sample-Suffix`, `Find-Within-Interval`, and `Find-Good-Split`, the first of which is given in [BECLW19], and the latter two are described below, in Figures 1 and 2. The majority of the section is devoted to the proof that `Find-Monotone` indeed outputs a $(12\ldots k)$-pattern with high probability as claimed. Specifically, we shall prove the following theorem.

Theorem 3.1. *Let $k \in \mathbb{N}$. The randomized algorithm $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$, described in Figure 3, satisfies the following. Given a function $f\colon I \to \mathbb{R} \cup \{*\}$ and parameters $\varepsilon, \delta \in (0,1)$, if f contains $\varepsilon|I|$ disjoint $(12\ldots k)$-patterns, then $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$ outputs a $(12\ldots k)$-pattern of f with probability at least $1-\delta$.*

Our proof proceeds by induction on k. It relies on Lemmas 3.3, 3.4, 3.5, the proofs of the latter two of which assume that Theorem 3.1 holds for smaller k. We first state and prove these lemmas, and then we prove Theorem 3.1.

To complete the picture, we need to upper-bound the query complexity and running time of `Find-Monotone`. We do this in the following lemma, whose proof we delay to the end of the section.

Lemma 3.2. *Let $f\colon I \to \mathbb{R} \cup \{*\}$, where I is an interval of length at most n. The query complexity and running time of* `Find-Monotone`$_k(f, \varepsilon, \delta)$ *are at most*

$$\left(k^k \cdot (\log(1/\varepsilon))^k \frac{1}{\varepsilon} \cdot \log(1/\delta)\right)^{O(k)} \cdot \log n.$$

3.1 The `Sample-Suffix` Sub-Routine

We re-state Lemma 3.1 from [BECLW19] which gives the `Sample-Suffix`$_k$ subroutine, with a few adaptations to fit our needs.

Lemma 3.3 ([BECLW19])**.** *Consider any fixed value of $k \in \mathbb{N}$, and let $C > 0$ be a large enough constant. There exists a non-adaptive and randomized algorithm,* `Sample-Suffix`$_k(f, \varepsilon, \delta)$ *which takes three inputs: query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, where $I \subset [n]$ is an interval, a parameter $\varepsilon \in (0,1)$, and an error probability bound $\delta \in (0,1)$. Suppose there exists $\alpha \in (0,1)$, and a set $H \subseteq I$ of $(\alpha, Ck\alpha)$-growing suffixes of f satisfying $\alpha|H| \geq \varepsilon|I|$. Then,* `Sample-Suffix`$_k(f, \varepsilon, \delta)$ *finds a length-k monotone subsequence of f with probability at least $1 - \delta$. The query complexity of* `Sample-Suffix`$_k(f, \varepsilon, \delta)$ *is at most*

$$\frac{\log n}{\varepsilon} \cdot \mathrm{polylog}(1/\varepsilon) \cdot \log(1/\delta).$$

The few adaptations that Lemma 3.3 has in comparison to Lemma 3.1 from [BECLW19] are with respect to the error probability going from $9/10$ to $1-\delta$, and the fact that we are considering functions $f\colon I \to \mathbb{R} \cup \{*\}$ as opposed to $f\colon [n] \to \mathbb{R}$.

In order to achieve error probability $1-\delta$ in Lemma 3.1 of [BECLW19], we perform $O(\log(1/\delta))$ independent repetitions of `Sample-Suffix`$_k$, as described in [BECLW19]. These are reflected in the query complexity.

The second difference is that we consider functions $f\colon I \to \mathbb{R} \cup \{*\}$. Inspecting the proof of Lemma 3.1 in [BECLW19], one can see that `Sample-Suffix`$_k$ is guaranteed to output, with high probability, $(12\ldots k)$-patterns whose indices are specified in Definition 2.1. Since the algorithm is non-adaptive, enforcing that indices not partaking in growing suffices not be used (by making them $*$) does not affect that analysis.

3.2 Handling Overshooting: The `Find-Within-Interval` Sub-Routine

In this section, we describe the `Find-Within-Interval` subroutine, addressing the overshooting case as explained in Section 1.2.

Lemma 3.4. *Consider the randomized algorithm,* `Find-Within-Interval`$_k(f, \varepsilon, \delta, x, y, \mathcal{J})$*, described in Figure 1, which takes six inputs:*

- *Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$,*
- *Two parameters $\varepsilon, \delta \in (0,1)$,*
- *Two points $x, y \in I$ where $x < y$ and $f(x) < f(y)$, and*

Subroutine $\texttt{Find-Within-Interval}_k(f, \varepsilon, \delta, x, y, \mathcal{J})$.

Input: Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$, two inputs $x, y \in I$ where $x < y$ and $f(x) < f(y)$, and $\mathcal{J} = (J_1, \ldots, J_{k-2})$ which is a collection of disjoint intervals appearing in order inside $[x, y]$.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. For every $\kappa \in [k-2]$, let $f_\kappa, f'_\kappa\colon J_\kappa \to \mathbb{R} \cup \{*\}$ be given by:
$$f_\kappa(i) = \begin{cases} f(i) & f(i) < f(y) \\ * & \text{o.w.} \end{cases} \qquad \text{and} \qquad f'_\kappa(i) = \begin{cases} f(i) & f(i) \geq f(y) \\ * & \text{o.w.} \end{cases} . \tag{3}$$
2. Call $\texttt{Find-Monotone}_{\kappa+1}(f_\kappa, \varepsilon/2, \delta/(2k))$ for every $\kappa \in [k-2]$.
3. Call $\texttt{Find-Monotone}_{k-\kappa}(f'_\kappa, \varepsilon/2, \delta/(2k))$ for every $\kappa \in [k-2]$.
4. Consider the set of all indices that are output in Lines 2 and 3, together with x and y. If there is a length-k increasing subsequence among these indices, output it. Otherwise, output fail.

Figure 1: Description of the `Find-Within-Interval` subroutine.

- *A collection $\mathcal{J} = (J_1, \ldots, J_{k-2})$ of $k-2$ disjoint intervals which appear in order (i.e., J_κ comes before $J_{\kappa+1}$) within the interval $[x, y]$,*

and outputs either a length-k increasing subsequence of f, or fail.

Suppose that for every $\kappa \in [k-2]$, the function $f|_{J_\kappa}\colon J_\kappa \to \mathbb{R} \cup \{*\}$, contains $\varepsilon|J_\kappa|$ disjoint $(12\ldots k)$-patterns. Then, assuming that Theorem 3.1 holds for every k' with $1 \leq k' < k$, $\texttt{Find-Within-Interval}_k(f, \varepsilon, \delta, x, y, \mathcal{J})$ outputs a length-k monotone subsequence of f with probability at least $1 - \delta$.

Proof. For each $\kappa \in [k-2]$, let $\mathcal{C}_\kappa$ be a collection of at least $\varepsilon|J_\kappa|$ disjoint $(12\ldots k)$-patterns in J_κ. We form the following two collections, of suffixes and prefixes of $(12\ldots k)$-patterns in $\mathcal{C}_\kappa$.

$$\mathcal{A}_\kappa = \{(i_1, \ldots, i_{\kappa+1}) : (i_1, \ldots, i_{\kappa+1}) \text{ is a prefix of a } k\text{-tuple from } \mathcal{C}_k, \text{ and } f(i_{\kappa+1}) < f(y)\}$$
$$\mathcal{B}_\kappa = \{(i_{\kappa+1}, \ldots, i_k) : (i_{\kappa+1}, \ldots, i_k) \text{ is a suffix of a } k\text{-tuple from } \mathcal{C}_k, \text{ and } f(i_{\kappa+1}) \geq f(y)\}$$

Note that for each $(12\ldots k)$-pattern in $\mathcal{C}_\kappa$, either its $(\kappa+1)$-prefix is in $\mathcal{A}_\kappa$, or its $(k-\kappa)$-suffix is in $\mathcal{B}_\kappa$. Thus, at least one of $\mathcal{A}_\kappa$ and $\mathcal{B}_\kappa$ has size at least $(\varepsilon/2)|J_\kappa|$. Say that J_κ is of type-1 if $|\mathcal{A}_\kappa| \geq (\varepsilon/2)|J_\kappa|$, and otherwise say that J_κ is of type-2 (in which case $|\mathcal{B}_\kappa| \geq (\varepsilon/2)|J_\kappa|$).

Now, if J_κ is of type-1, then Line 2, called with κ, will find a $(12\ldots(\kappa+1))$-pattern with probability at least $1 - \delta/(2k)$, by Theorem 3.1 for $\kappa + 1 < k$ (namely, the inductive hypothesis) and the lower bound on $|\mathcal{A}_\kappa|$. On the other hand, if J_κ is of type-2, Line 3 will output a $(12\ldots(k-\kappa))$-pattern with probability at least $1 - \delta/(2k)$, due to the inductive hypothesis and the lower bound on $|\mathcal{B}_\kappa|$. Thus, by a union bound, with probability at least $1 - \delta$, Line 2 outputs a pattern whenever J_κ is of type-1, and Line 3 outputs a pattern whenever J_κ is of type-2.

Notice that if J_1 is of type-2, the $(12\dots(k-1))$-pattern returned in Line 3 can be combined with x to form a $(12\dots k)$-pattern. Hence, we may assume that J_1 is of type-1. Furthermore, if J_{k-2} is of type-1, the $(12\dots(k-1))$-pattern found in Line 2 can be combined with y to form a $(12\dots k)$-pattern, and hence, we may assume that J_{k-2} is of type-2. Thus, there exists some $\kappa \in [k-3]$ where J_κ is of type-1 and $J_{\kappa+1}$ is of type-2. Since J_κ comes before $J_{\kappa+1}$, and since non-starred elements in f_κ lie below the non-$*$ elements of f'_{k+1}, we can combine the $(12\dots(\kappa+1))$-pattern in f_κ with the $(12\dots(k-\kappa-1))$-pattern in $f'_{\kappa+1}$. □

3.3 Handling the Fitting Case: The `Find-Good-Split` Sub-Routine

In this section, we describe the `Find-Good-Split` subroutine, which corresponds to the fitting case from Section 1.2.

Subroutine $\texttt{Find-Good-Split}_k(f, \varepsilon, \delta, c, \xi)$.

Input: Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$, and $c \in [k-1]$. We let $c_1 > 1$ be a large enough (absolute) constant.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. Repeat the following procedure $t = c_1 k/(\varepsilon\xi^2) \cdot \log(1/\delta)$ times:
   (a) Sample $\boldsymbol{w}, \boldsymbol{z} \sim I$, and consider the functions $f_{\boldsymbol{z},\boldsymbol{w}}\colon I \cap (-\infty, \boldsymbol{z}) \to \mathbb{R} \cup \{*\}$ and $f'_{\boldsymbol{z},\boldsymbol{w}}\colon I \cap [\boldsymbol{z}, \infty) \to \mathbb{R} \cup \{*\}$ given by
   $$f_{\boldsymbol{z},\boldsymbol{w}}(i) = \begin{cases} f(i) & f(i) < f(\boldsymbol{w}) \\ * & \text{o.w.} \end{cases} \quad \text{and} \quad f'_{\boldsymbol{z},\boldsymbol{w}}(i) = \begin{cases} f(i) & f(i) \geq f(\boldsymbol{w}) \\ * & \text{o.w.} \end{cases}. \tag{4}$$
   (b) Run $\texttt{Find-Monotone}_c(f_{\boldsymbol{z},\boldsymbol{w}}, \varepsilon\xi/3, \delta/3)$ and $\texttt{Find-Monotone}_{k-c}(f'_{\boldsymbol{z},\boldsymbol{w}}, \varepsilon\xi/3, \delta/3)$.
2. If we ever find a length-k monotone subsequence of f, output it, otherwise, output fail.

Figure 2: Description of the `Find-Good-Split` subroutine.

Lemma 3.5. *Consider the randomized algorithm* $\texttt{Find-Good-Split}_k(f, \varepsilon, \delta, c, \xi)$*, described in Figure 2, which takes as input five parameters:*

- *Query access to a function* $f\colon I \to \mathbb{R} \cup \{*\}$*,*
- *Two parameters* $\varepsilon, \delta \in (0,1)$*,*
- *An integer* $c \in [k-1]$*, and*
- *A parameter* $\xi \in (0,1]$*,*

and outputs either a length-k increasing subsequence or fail*.*

Suppose that there exists an interval-tuple pair (I', T) which is $(c, 1/(6k), \varepsilon)$-splittable and $|I'|/|I| \geq \xi$. Then, the algorithms $\texttt{Find-Good-Split}_k(f, \varepsilon, \delta, c, \xi)$ *finds a $(12\dots k)$-pattern of f with probability $1-\delta$.*

Proof. Let (I', T) be $(c, 1/(6k), \varepsilon)$-splittable, and let L, M, R be the contiguous intervals splitting I' as in Definition 2.2. Furthermore, let $T^{(L)}$ and $T^{(R)}$ be as in Definition 2.2. Writing

$$m_1 = \text{rank}\left(\left\{f(i_c) : (i_1, \ldots, i_c) \in T^{(L)}\right\}, |T|/3\right),$$
$$m_2 = \text{rank}\left(\left\{f(i_c) : (i_1, \ldots, i_c) \in T^{(L)}\right\}, 2|T|/3\right),$$

as the $(|T|/3)$-largest and $(2|T|/3)$-largest elements in $\left\{f(i_c) : (i_1, \ldots, i_c) \in T^{(L)}\right\}$ (taking multiplicity into account). Let $T_l^{(L)}$ be the $(12\ldots c)$-patterns in $T^{(L)}$ where the c-th index is at most m_1, and $T_h^{(R)}$ be the $(k-c)$-patterns in $T^{(R)}$ whose $(c+1)$-th index is larger than m_2. Notice that $|T_l^{(L)}|, |T_h^{(R)}| \geq |T|/3$, and that any $(12\ldots c)$-pattern from $T_l^{(L)}$ can be combined with any $(12\ldots(k-c))$-pattern from $T_h^{(R)}$ to form a $(12\ldots k)$-pattern. Furthermore, there exists $|T|/3$ indices in I' whose function value lies in $[m_1, m_2]$.

Consider the event, defined over the randomness of $\boldsymbol{w}, \boldsymbol{z} \sim I$ that: $\boldsymbol{z} \in M$; and $\boldsymbol{w}$ satisfies $f(\boldsymbol{w}) \in [m_1, m_2]$. This event occurs at some iteration of Line 1, with probability at least $1 - \delta/3$; this is because there are $|M| \geq |I'|/(6k) \geq (\xi/(6k))|I|$ valid indices for $\boldsymbol{z}$, and there are at least $|T|/3 \geq (\varepsilon/3)|I'| \geq (\varepsilon\xi/3)|I|$ indices for $\boldsymbol{w}$, so the probability that the pair $(\boldsymbol{z}, \boldsymbol{w})$ satisfies the requirements is at least $\varepsilon\xi^2/(18k)$. We obtain the desired bound by the setting of t, since c_1 is set to a large enough constant.

Notice that when this event occurs, the $(12\ldots c)$-patterns in $T_l^{(L)}$ all lie in $f_{\boldsymbol{z},\boldsymbol{w}}$, and the $(12\ldots(k-c))$-patterns in $T_h^{(R)}$ all lie in $f'_{\boldsymbol{z},\boldsymbol{w}}$. Thus, $f_{\boldsymbol{z},\boldsymbol{w}}$ contains at least $|T|/3 \geq (\varepsilon/3)|I'| \geq (\varepsilon\xi/3)|I|$ disjoint $(12\ldots c)$-patterns, and $f'_{\boldsymbol{z},\boldsymbol{w}}$ similarly contains at least $(\varepsilon\xi/3)|I|$ disjoint $(12\ldots(k-c))$-patterns. Thus, by the inductive hypothesis, Line 1b finds a $(12\ldots c)$-pattern in $f_{\boldsymbol{z},\boldsymbol{w}}$ and a $(12\ldots(k-c))$-pattern in $f'_{\boldsymbol{z},\boldsymbol{w}}$ with probability at least $1 - 2\delta/3$, and these can be combined to give a $(12\ldots k)$-pattern of f. □

3.4 The Main Algorithm

Consider the description of the main algorithm in Figure 3. We prove Theorem 3.1 by induction on k. The proof uses Lemma 3.3, Lemma 3.4, and Lemma 3.5.

Proof of Theorem 3.1.

Base Case: $k = 1$.

Recall that f has at least $\varepsilon|I|$ non-$*$ values. Thus, with probability at least $1 - \delta$, a non-$*$ value is observed after sampling $\boldsymbol{x} \sim I$ at least $(1/\varepsilon) \cdot \log(1/\delta)$ times. It follows that with probability at least $1 - \delta$, Line 2a of our main algorithm, given in Figure 3, samples $\boldsymbol{x} \neq *$ in one of its iterations.

Inductive Step: proof of Theorem 3.1 for $k \geq 2$, under the assumption that it holds for every k' with $1 \leq k' < k$.

Let $p = P(k\log(1/\varepsilon))$ (recall that $P(\cdot)$ is a polynomial of sufficiently large (constant) degree). Apply Theorem 2.6 to f.

Suppose, first, that (1) of Theorem 2.6 holds. So, there exists a set $H \subset [n]$ of indices that start an $(\alpha, Ck\alpha)$-growing suffix, with $\alpha|H| \geq (\varepsilon/p)n$, for some $\alpha \in (0, 1)$. By Lemma 3.3, the call for $\texttt{Sample-Suffix}_k(f, \varepsilon/p, \delta)$ in Line 1 outputs a length-k monotone subsequence of f with probability at least $1 - \delta$.

Subroutine $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$.

Input: Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$. We let $c_1, c_2, c_3 > 0$ be large enough constants, and let $p = P(k \log(1/\varepsilon))$, where $P\colon \mathbb{R} \to \mathbb{R}$ is a polynomial of large enough (constant) degree.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. Run $\texttt{Sample-Suffix}_k(f, \varepsilon/p, \delta)$.
2. Repeat the following for $c_1 \log(1/\delta) \cdot p \cdot k^5/\varepsilon^2$ many iterations:
   (a) Sample $\boldsymbol{x} \sim I$ uniformly at random. If $f(\boldsymbol{x}) = *$, proceed to the next iteration. Otherwise, if $k = 1$ output $\boldsymbol{x}$ and proceed to Step 3, and if $k \geq 2$ proceed to the next step.
   (b) For each $t \in [\log n]$, sample $\boldsymbol{y}_t \sim [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ uniformly at random. If there exists at least one t where $f(\boldsymbol{y}_t) > f(\boldsymbol{x})$, set
   $$\boldsymbol{y} = \max\{\boldsymbol{y}_t : t \in [\log n] \text{ and } f(\boldsymbol{y}_t) > f(\boldsymbol{x})\}, \tag{5}$$
   let $t^* \in [\log n]$ be the index for which $\boldsymbol{y}_{t^*} = \boldsymbol{y}$, and continue to the next line. Otherwise, i.e. if $f(\boldsymbol{y}_t) \not> f(\boldsymbol{x})$ for every t, continue to the next iteration.
   (c) If $k = 2$, output $(\boldsymbol{x}, \boldsymbol{y})$ and proceed to Step 3. If $k > 2$, continue to the next line.
   (d) Here $k \geq 3$. Set $\ell = 4p/\varepsilon$ and perform the following.
      i. Consider the collection $\mathcal{J}$ of $k-2$ intervals $J_1, \ldots, J_{k-2}$ appearing in order within $[\boldsymbol{x}, \boldsymbol{y}]$, given by setting, for every $i \in [k-2]$,
      $$J_i = \left[\boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-1-i)}, \boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)}\right), \tag{6}$$
      and run $\texttt{Find-Within-Interval}_k(f, \varepsilon/2p, \delta/2, \boldsymbol{x}, \boldsymbol{y}, \mathcal{J})$.
      ii. For each $t' \in [t^* - 3k \log \ell, t^*]$ do the following.
      Consider the interval $J_{t'} = [\boldsymbol{x} - 2^{t'}, \boldsymbol{x} + 2^{t'}]$, and the restricted function $g_{t'}\colon J_{t'} \to \mathbb{R} \cup \{*\}$ given by $g_{t'} = f|_{J_{t'}}$. For every $c_0 \in [k-1]$, run $\texttt{Find-Good-Split}_k(g_{t'}, \varepsilon/c_2, \delta/2, c_0, 1/4)$.
3. If a length-k monotone subsequence of f is found, output it. Otherwise, output fail.

Figure 3: Description of the $\texttt{Find-Monotone}_k$ subroutine.

Now suppose that (2) of Theorem 2.6 holds, and let $(I_1, T_1), \ldots, (I_s, T_s)$ be a $(c, 1/(6k), \alpha)$-splittable collection for some $\alpha \geq \Omega(\varepsilon/k^5)$ and $c \in [k-1]$, satisfying (2) and, moreover, that any $J \subset I$ with $J \supset I_h$ for some $h \in [s]$ contains $(\varepsilon/p)|J|$ disjoint $(12\ldots k)$-patterns. Let Event be the event that, for a particular iteration of Lines 2a and 2b, $\boldsymbol{x}$ is the 1-entry of some k-tuple from T_h, for some $h \in [s]$, and $\boldsymbol{y}_t$ is the $(c+1)$-entry of some (possibly other) k-tuple in T_h, where t is such that $|I_h| \leq 2^t < 2|I_h|$.

Claim 3.6. $\mathbf{Pr}[\texttt{Event}] \geq \varepsilon\alpha/(2p)$.

Proof. For each $h \in [s]$, let A_h and B_h be the collections of 1- and $(c+1)$-entries of patterns in T_h. Then

$$\sum_{h=1}^{s} |A_h| = \sum_{h=1}^{s} |T_h| \geq \alpha \sum_{h=1}^{s} |I_h| \geq \frac{\varepsilon}{p} \cdot |I|.$$

The first inequality follows from the assumption that (I_h, T_h) is $(c, 1/(6k), \alpha)$-splittable, and the second inequality follows from the assumption that (2) holds.

As a result, the probability over the draw of $\boldsymbol{x} \sim I$ in Line 2a that $\boldsymbol{x} \in A_h$ is at least ε/p. Fix such an $\boldsymbol{x}$, and consider $t \in [\log n]$ for which $|I_h| \leq 2^t < 2|I_h|$. Notice that $B_h \subset [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ since $2^{t-1} \leq |I_h| < 2^t$, and that the distance between any index of A_h and B_h is at least $|I_h|/(6k) \geq 2^t/(12k)$ since (I_h, T_h) is $(c, 1/(6k), \alpha)$-splittable. Therefore, the probability over the draw of $\boldsymbol{y}_t \sim [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ that $\boldsymbol{y}_t \in B_h$ is at least $|B_h|/2^t \geq |T_h|/(2|I_h|) \geq \alpha/2$. □

By the previous claim, since we have $c_1 \cdot \log(1/\delta) \cdot p \cdot k^5/\varepsilon^2$ iterations of Lines 2a and 2b, with probability at least $1 - \delta/2$, `Event` holds in some iteration (using the lower bound $\alpha \geq \Omega(\varepsilon/k^5)$ and the choice of c_1 as a large constant).

Consider the first execution of Line 2a and Line 2b where `Event` holds (assuming such an execution exists). Let $h \in [s]$ and $t \in [\log n]$ be the corresponding parameters, i.e., h and t are set so $\boldsymbol{x}$ is the first index of a k-tuple in T_h, $\boldsymbol{y}_t$ is the $(c+1)$-th index in another k-tuple in T_h, and $|I_h| \leq 2^t < 2|I_h|$. We consider this iteration of Line 2, and assume that `Event` holds with these parameters for the rest of the proof. Notice that $\boldsymbol{y}$, as defined in (5), satisfies $\boldsymbol{y} \geq \boldsymbol{y}_t$ (as $f(\boldsymbol{y}) > f(\boldsymbol{x})$) and hence $t^* \geq t$.

Note that if $k = 2$, the pair $(\boldsymbol{x}, \boldsymbol{y})$, which is a (12)-pattern in f, is output in Line 2c, so the proof is complete in this case. From now on, we assume that $k \geq 3$. We break up the analysis into two cases: $t^* \geq t + 3k \log \ell$ and $t^* < t + 3k \log \ell$.

Suppose $t^* \geq t + 3k \log \ell$. We now observe a few facts about the collection $\mathcal{J}$ specified in (6). First, notice that $J_1, \ldots, J_{k-2}$ appear in order from left-to-right, and they lie in $[\boldsymbol{x}, \boldsymbol{y}]$ (as $\boldsymbol{y} = \boldsymbol{y}_{t^*} \in [\boldsymbol{x} + 2^{t^*}/(12k), 2^{t^*}]$). Second, in the next claim we show that for every $i \in [k-2]$, the interval J_i contains $(\varepsilon/2p)|J_i|$ disjoint $(12\ldots k)$-patterns.

Claim 3.7. *J_i contains $(\varepsilon/2p)|J_i|$ disjoint $(12\ldots k)$-patterns.*

Proof. Let J_i' be the interval given by

$$J_i' = I_h \cup \left[\boldsymbol{x}, \boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)}\right].$$

Observe that

$$|J_i' \setminus J_i| \leq 2^t + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-1-i)} \leq \frac{2^{t^*}}{6k} \cdot \ell^{-(k-1-i)} = \frac{2}{\ell} \cdot \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)} \geq \frac{2}{\ell} \cdot |J_i'| = \frac{\varepsilon}{2p} \cdot |J_i'|,$$

where for the second inequality we used the bound $t^* - t \geq 3k \log \ell \geq \log(12) + \log k + (k-2) \log \ell$, and that $\ell = 4p/\varepsilon$.

We have by Theorem 2.6, that J'_i contains at least $(\varepsilon/p)|J'_i|$ disjoint $(12\dots k)$-patterns in f. Hence, the number of disjoint $(12\dots k)$-patterns in J_i is at least:

$$\frac{\varepsilon}{p} \cdot |J'_i| - |J'_i \setminus J_i| \geq \frac{\varepsilon}{2p} \cdot |J'_i| \geq \frac{\varepsilon}{2p} \cdot |J_i|,$$

as required. □

By Lemma 3.4, Line 2(d)i outputs a $(12\dots k)$-pattern in f with probability at least $1 - \delta/2$. By a union bound, we obtain the desired result.

Suppose, on the other hand, that $t^* \leq t + 3k \log \ell$. In this case, as $2^{t-1} \leq |I_h| \leq 2^{t^*}$ (by choice of t), for one of the values of t' considered in Line 2(d)ii we have $2^{t'-1} \leq |I_h| < 2^{t'}$; fix this t'. The interval $J_{t'}$, defined in Line 2(d)ii, hence satisfies $|I_h|/|J_{t'}| \geq 1/4$. As a result, and since $I_h \subset J_{t'}$ (because $t \leq t^*$), the function $g\colon J \to \mathbb{R} \cup \{*\}$ contains an interval-tuple pair (I_h, T_h) which is $(c, 1/(6k), \alpha)$-splittable. By Lemma 3.5, once Line 2(d)ii considers $c_0 = c$, the sub-routine $\texttt{Find-Good-Split}_k(g, \varepsilon/(c_2 k^5), \delta/2, c, 1/4)$ will output a $(12\dots k)$-pattern of $g_{t'}$ (which is also a $(12\dots k)$-pattern of f) with probability at least $1 - \delta/2$. Hence, we obtain the result by a union bound. □

3.5 Query Complexity and Running Time

It remains to prove Lemma 3.2, estimating the number of queries made by `Find-Monotone`, as well as its total running time.

Proof of Lemma 3.2. We first claim that the running time is bounded by an expression of the form $\text{poly}(k)$ times the query complexity of `Find-Monotone`, where the $\text{poly}(\cdot)$ term is of constant degree. Indeed, the only costly operations (in terms of running time) other than querying that our algorithm conducts involve:

- Determining whether the value of f at a certain point is $*$ or not; to this end, note that for any f we need to evaluate along the way, $f(x)$ is marked by $*$ if and only if it does not belong to some interval in $\mathbb{R}$, whose endpoints are determined by the recursive calls that led to it. Since the recursive depth is at most k, this means that the complexity of the above operation is $O(k)$.[13]
- Given an ordered set of queried elements Q at some point along the algorithm, determining whether these elements contain a c-increasing subsequence for $c \leq k$ (this action is taken, e.g., in the last part of `Find-Monotone`). This operation can be implemented in time $O(c|Q|)$. Now, the number of such operations that each queried element participates in is at most k,[14] and a simple double counting argument implies that the running time of these operations altogether is at most $O(k^2)$ times the total query complexity.

[13] In fact, this complexity can be improved to $O(1)$ if, instead of working with functions of the form $f\colon I \to \mathbb{R} \cup \{*\}$, we would have worked with function $f\colon I \to \mathbb{R}$ and received the interval of "non-$*$ values" as an input to the recursive call.

[14] More precisely, for the purpose of this section, if an element is queried $t > 1$ times by our algorithm then we think of it as contributing t to the total query complexity (since our goal is to prove upper bounds here – not lower bounds – this perspective is clearly valid); and in this case, the number of operations as above in which it participates is at most $k \cdot t$.

It remains now to prove the bound on the query complexity. Recall that $P\colon \mathbb{R} \to \mathbb{R}$ is a fixed polynomial; write $p_{k,\varepsilon} = P(k\log(1/\varepsilon))$. We fix n, which upper bounds the length of all intervals defining input functions. Let $\Phi(k,\varepsilon,\delta)$ be the maximum number of queries made by $\texttt{Find-Monotone}_k(f,\varepsilon,\delta)$. Let

$$\begin{aligned}
\Phi^{(1)}(k,\varepsilon,\delta) &= \text{ query complexity of } \texttt{Sample-Suffix}_k(f,\varepsilon,\delta).\\
\Phi^{(2)}(k,\varepsilon,\delta) &= \text{ query complexity of } \texttt{Find-Within-Interval}_k(f,\varepsilon,\delta,x,y,\mathcal{J}),\\
&\quad \text{where } |\mathcal{J}| = k-2.\\
\Phi^{(3)}(k,\varepsilon,\delta,\xi) &= \text{ query complexity of } \texttt{Find-Good-Split}_k(f,\varepsilon,\delta,c,\xi),\\
&\quad \text{where } c \in [k-1].
\end{aligned}$$

By Lemma 3.3, as well as an inspection of Figure 1 and Figure 2, we have:

$$\begin{aligned}
\Phi^{(1)}(k,\varepsilon,\delta) &\le p_{k,\varepsilon}\cdot\frac{1}{\varepsilon}\cdot\log(1/\delta)\cdot\log n\\
\Phi^{(2)}(k,\varepsilon,\delta) &\le 2k\cdot\Phi(k-1,\varepsilon/2,\delta/(2k))\\
\Phi^{(3)}(k,\varepsilon,\delta,\xi) &\le \frac{c_1 k\log(1/\delta)}{\varepsilon\xi^2}\cdot\Phi(k-1,\varepsilon\xi/3,\delta/3).
\end{aligned}$$

Lastly, inspecting Figure 3, we have

$$\begin{aligned}
\Phi(k,\varepsilon,\delta) &\le \Phi^{(1)}(k,\varepsilon/p_{k,\varepsilon},\delta)+\\
&\qquad c_1\cdot p_{k,\varepsilon}\cdot\frac{k^5}{\varepsilon^2}\cdot\log(1/\delta)\cdot\Big(1+\log n+\Phi^{(2)}\left(k,\varepsilon/(2p_{k,\varepsilon}),\delta/2\right)+\Phi^{(3)}\left(k,\varepsilon/(c_2k^5),\delta/2,1/4\right)\Big)\\
&\le q_{k,\varepsilon}\cdot\frac{1}{\varepsilon^2}\cdot\log(1/\delta)\cdot\log n\ + q_{k,\varepsilon}\cdot\frac{1}{\varepsilon^3}\cdot(\log(1/\delta))^2\cdot\Phi(k-1,\varepsilon/q_{k,\varepsilon},\delta/(3k))\\
&\le \left(k^k\cdot(\log(1/\varepsilon))^k\cdot\frac{1}{\varepsilon}\cdot\log(1/\delta)\right)^{O(k)}\cdot\log n,
\end{aligned}$$

where $Q\colon \mathbb{R}\to\mathbb{R}$ is a fixed polynomial of large enough (constant) degree and $q_{k,\varepsilon} = Q(k\log(1/\varepsilon))$. For the last line we use that $\Phi^{(2)}(1,\cdot,\cdot) = \Phi^{(2)}(2,\cdot,\cdot) = \Phi^{(3)}(1,\cdot,\cdot,\cdot) = \Phi^{(3)}(2,\cdot,\cdot,\cdot) = 0$, and we note that the parameter replacing ε never falls below $\varepsilon/(k\log(1/\varepsilon))^{O(k)}$, so the factor of $\log n$ at each iteration is at most $\left(k^k(\log(1/\varepsilon))^k(1/\varepsilon)\log(1/\delta)\right)^{O(k)}$. □

References

[ACCL07] Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31(3):371–383, 2007.

[BB15] Aleksandrs Belovs and Eric Blais. Quantum algorithm for monotonicity testing on the hypercube. *Theory of Computing*, 11(16):403–412, 2015.

[BBM12] Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

[BC18] Omri Ben-Eliezer and Clément L. Canonne. Improved bounds for testing forbidden order patterns. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2093–2112, 2018.

[BCGSM12] Jop Briët, Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Monotonicity testing and shortest-path routing on the cube. *Combinatorica*, 32(1):35–53, 2012.

[BCS18] Hadley Black, Deeparnab Chakrabarty, and C. Seshadhri. A $o(d) \cdot \text{polylog} n$ monotonicity tester for boolean functions over the hypergrid $[n]^d$. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2133–2151, 2018.

[BECLW19] Omri Ben-Eliezer, Clément L. Canonne, Shoham Letzter, and Erik Waingarten. Finding monotone patterns in sublinear time. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2019. To appear.

[Bel18] Aleksandrs Belovs. Adaptive Lower Bound for Testing Monotonicity on the Line. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 31:1–31:10, 2018.

[Ben19] Omri Ben-Eliezer. Testing local properties of arrays. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 11:1–11:20, 2019.

[BKM19] Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and counting permutations via CSPs. In *Proceedings of the 14th International Symposium on Parameterized and Exact Computation (IPEC)*. 2019. To appear.

[BRY14] Eric Blais, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Lower bounds for testing properties of functions over hypergrid domains. In *Proceedings of the 29th Conference on Computational Complexity (CCC)*, pages 309–320, 2014.

[CDST15] Xi Chen, Anindya De, Rocco A. Servedio, and Li-Yang Tan. Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proceedings of the 47th ACM Symposium on the Theory of Computing (STOC)*, pages 519–528, 2015.

[CS13] Deeparnab Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC)*, pages 419–428, 2013.

[CS14] Deeparnab Chakrabarty and C. Seshadhri. An optimal lower bound for monotonicity testing over hypergrids. *Theory of Computing*, 10(17):453–464, 2014.

[CS16] Deeparnab Chakrabarty and C. Seshadhri. An $o(n)$ monotonicity tester for boolean functions over the hypercube. *SIAM Journal on Computing*, 45(2):461–472, 2016.

[CS19] Deeparnab Chakrabarty and C Seshadhri. Adaptive boolean monotonicity testing in total influence time. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 20:1–20:7, 2019.

[CST14] Xi Chen, Rocco A. Servedio, and Li-Yang Tan. New algorithms and lower bounds for monotonicity testing. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–295, 2014.

[CWX17] Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond Talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th ACM Symposium on the Theory of Computing (STOC)*, pages 523–536, 2017.

[DGL+99] Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 97–108, 1999.

[Dil50] Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[EJ08] Funda Ergün and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 730–736, 2008.

[EJ15] Funda Ergün and Hossein Jowhari. On the monotonicity of a data stream. *Combinatorica*, 35(6):641–653, 2015. Short version in SODA'08 [EJ08].

[EKK+00] Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Vishwanthan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000.

[Fis04] Eldar Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004.

[Fox13] Jacob Fox. Stanley–Wilf limits are typically exponential. *arXiv:1310-8378*, 2013. Also: Advances in Mathematics, to appear.

[GG07] Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 294–304, 2007.

[GG10] Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. *SICOMP*, 39(8):3463–3479, 2010. Short version in FOCS'07 [GG07].

[GGR98] 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.

[GJKK07] Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007.

[GM14] Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–101, 2014.

[Gol17] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

[KMS15] Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric type theorems. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 52–58, 2015.

[Knu68] Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

[NRRS17] Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1582–1597, 2017.

[NS15] Timothy Naumovitz and Michael E. Saks. A polylogarithmic space deterministic streaming algorithm for approximating distance to monotonicity. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1252–1262, 2015.

[PRR06] Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.

[PRV18] Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and Nithin M. Varma. Parameterized

property testing of functions. *ACM Transactions on Computation Theory*, 9(4):17:1–17:19, 2018.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252,Äì–271, 1996.

[RSSS19] Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2019. To appear.

[SS85] Rodica Simion and Frank W. Schmidt. Restricted permutations. *European Journal of Combinatorics*, 6(4):383 – 406, 1985.

[SS10] Michael Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 458–467, 2010.

[SS13] Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1698–1709, 2013.

[SS17] Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM J. Comput.*, 46(2):774–823, 2017. Short version in FOCS'10 [SS10].

[SW07] Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007.

[Vat15] Vincent Vatter. Permutation classes. In *Handbook of Enumerative Combinatorics*, pages 777–858. Chapman and Hall/CRC, 2015.