

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 $f: [n] \rightarrow \mathbb{R}$, 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 $\Theta((\log n)^{\lceil \log_2 k \rceil})$. 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 $\Theta(\log n)$.

1 Introduction

For an integer $k \in \mathbb{N}$ and a function (or sequence) $f: [n] \rightarrow \mathbb{R}$, a *length- k monotone subsequence of f* is a tuple of k indices, $(i_1, \dots, i_k) \in [n]^k$, such that $i_1 < \dots < i_k$ and $f(i_1) < \dots < f(i_k)$. More generally, for a permutation $\pi: [k] \rightarrow [k]$, a *π -pattern of f* is given by a tuple of k indices $i_1 < \dots < i_k$ 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 , the goal is to test whether a function $f: [n] \rightarrow \mathbb{R}$ is π -free or ε -far from π -free (that is, any π -free function g differs from f on at least εn inputs). They considered the monotone case as a particularly interesting instance; monotone patterns are naturally connected to monotonicity testing and the longest increasing subsequence, and so testing for the existence of monotone subsequences can shine new light on these classic problems. The algorithmic task, which is the subject of this paper, is the following.

*Weizmann Institute of Science. email: omrib@mail.tau.ac.il. Research conducted while the author was at Tel Aviv University.

[†]University College London, email: s.letzter@ucl.ac.uk. Research supported by Dr. Max Rössler, the Walter Haefner Foundation, and the ETH Zurich Foundation.

[‡]Stanford University, email: eaw@cs.columbia.edu.

For $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$, design a randomized algorithm that, given query access to a function $f: [n] \rightarrow \mathbb{R}$, 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 $f: [n] \rightarrow \mathbb{R}$ which is ε -far from $(12 \dots k)$ -free, outputs a length- k monotone subsequence of f with probability at least $9/10$, with query complexity and running time of $O_{k,\varepsilon}(\log n)$.*¹

For the precise bound on the query complexity and running time, see Lemma 3.2. The algorithm underlying Theorem 1.1 is *adaptive*² and solves the testing problem with *one-sided error*,³ since a length- k monotone subsequence is evidence for not being $(12 \dots k)$ -free. The algorithm improves on a recent result of Ben-Eliezer, Canonne, Letzter and Waingarten [BECLW19] who gave a non-adaptive algorithm for finding length- k monotone patterns with query complexity $O_{k,\varepsilon}((\log n)^{\lceil \log_2 k \rceil})$, which in itself improved upon a $O_{k,\varepsilon}((\log n)^{O(k^2)})$ upper bound by Newman et al. [NRRS17]. The focus of [BECLW19] was on *non-adaptive* algorithms, and they gave a lower bound of $\Omega((\log n)^{\lceil \log_2 k \rceil})$ 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.

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 (as long as, say, $\varepsilon > n^{-0.99}$) 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 dependence on n in the query complexity by more than a constant factor. Interestingly, Fischer [Fis04] shows that for any adaptive algorithm for monotonicity testing on $f: [n] \rightarrow \mathbb{R}$ there is a non-adaptive algorithm which is 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 $\Omega((\log n)^{\lceil \log_2 k \rceil})$ non-adaptive lower bound of [BECLW19] implies that this is no longer true for $k \geq 4$.

Harnessing adaptivity to improve algorithmic performance is a notoriously difficult problem in many branches of property testing, typically requiring a good structural understanding of the task at hand. In the context of testing for forbidden order patterns, non-adaptive algorithms are quite weak: the non-adaptive query complexity is $\Omega(n^{1/2})$ for all non-monotone order patterns [NRRS17],

¹To simplify the discussion, in the rest of this exposition we are generally not interested in the exact dependence on the parameters k and ε , and for convenience we often use notions like $O_{k,\varepsilon}(\cdot)$ and $\Omega_{k,\varepsilon}(\cdot)$ suppressing this dependence.

²An 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*.

³An 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.

and as high as $n^{1-1/(k-\Theta(1))}$ for almost all patterns of length k [BC18]. Prior to our work, the only case for which adaptive algorithms were known to outperform their non-adaptive counterparts was for patterns of length 3 in [NRRS17]. It is generally believed that there should be some separation between adaptive and non-adaptive testing algorithms for pattern detection; in fact, a conjecture of [NRRS17] suggests that for non-monotone patterns, the adaptive query complexity for testing π -freeness is polylogarithmic in n for *any* fixed-length π , an exponential improvement over non-adaptive algorithms. While this conjecture is still wide open, our work is the first to show any kind of separation between adaptive and non-adaptive algorithms for patterns of length more than 3.

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 to determine, given a sequence $f: [n] \rightarrow \mathbb{R}$, the maximum k for which f contains a length- k increasing subsequence. It is very closely related to other fundamental algorithmic problems in sequences, such as computing the edit distance, Ulam distance, or distance from monotonicity (for example, the latter equals n minus the LIS length), and has been 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 over 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 $f: [n] \rightarrow \mathbb{R}$ 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] running in time $n^{k/4+o(k)}$ provides the state-of-the-art. The analogous *counting* problem has also been actively studied, see [EL19] and the references within.

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]).

1.2 Main Ideas and Techniques

We now describe some intuition behind the proof of Theorem 1.1. We note that the algorithm considers several cases and combines ideas from [NRRS17] and [BECLW19] with new structural and algorithmic components. In this overview, technical details established in [NRRS17] and [BECLW19]

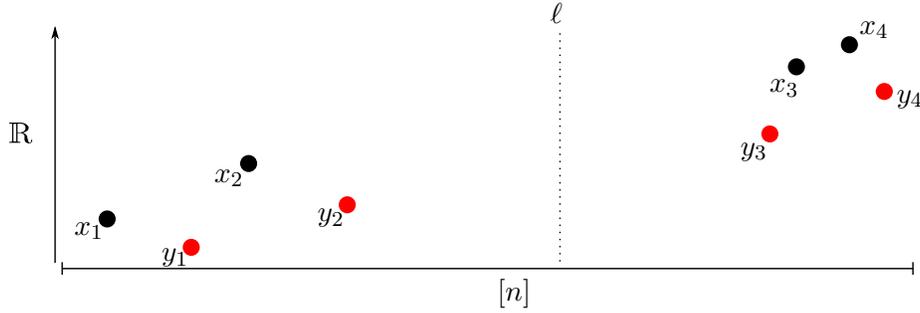


Figure 1: A sequence $f: [n] \rightarrow \mathbb{R}$ with two disjoint monotone subsequences of length 4, as well as an index $\ell \in [n]$. The sequences are $x = (x_1, x_2, x_3, x_4)$ and $y = (y_1, y_2, y_3, y_4)$. Notice that both x and y have the largest gap between consecutive elements at index 2, i.e., $|x_3 - x_2|$ and $|y_3 - y_2|$ are the largest gaps between consecutive indices in x and y . Furthermore, ℓ cuts both x and y with slack.

are noted but excluded; the purpose is to highlight the challenges and novel ideas arising specifically from this work. (We include a short overview of the non-adaptive algorithms of [NRRS17] and [BECLW19] in Appendix A.)

Fix $k \in \mathbb{N}$ and $\varepsilon > 0$, and suppose that $f: [n] \rightarrow \mathbb{R}$ 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 ; indeed, otherwise, greedily eliminating these subsequences gives a $(12\dots k)$ -free function differing in strictly fewer than εn inputs.

For simplicity in this overview, assume that k is even and that all $\varepsilon n/k$ length- k increasing subsequences of f in \mathcal{C} , $(x_1, x_2, \dots, x_k) \in [n]^k$, satisfy that $|x_{k/2+1} - x_{k/2}| \geq |x_{i+1} - x_i|$ for all $i \in [k-1]$ (the non-adaptive lower bound of $\Omega_\varepsilon((\log n)^{\lceil \log_2 k \rceil})$ holds even in this restricted case). We say that an index $\ell \in [n]$ *cuts* (x_1, \dots, x_k) *with slack* if

$$x_{k/2} + \frac{x_{k/2+1} - x_{k/2}}{3} \leq \ell \leq x_{k/2+1} - \frac{x_{k/2+1} - x_{k/2}}{3},$$

or, informally, if ℓ lies “roughly in the middle” between $x_{k/2}$ and $x_{k/2+1}$ – which, by the above assumption, form the largest gap among consecutive indices of the increasing subsequence (see Figure 1). Additionally, the *width* of (x_1, \dots, x_k) is set to be $\lfloor \log(x_{k/2+1} - x_{k/2}) \rfloor$. We consider the subset of \mathcal{C} consisting of length- k monotone subsequences of width w which are cut by ℓ with slack,

$$\mathcal{C}_{\ell, w} = \{(x_1, \dots, x_k) \in \mathcal{C} : \text{width}(x_1, \dots, x_k) = w, \ell \text{ cuts } (x_1, \dots, x_k) \text{ with slack}\},$$

and note that if $(x_1, \dots, x_k) \in \mathcal{C}_{\ell, w}$, then $x_1, \dots, x_{k/2} \in [\ell - k \cdot 2^w, \ell]$ and $x_{k/2+1}, \dots, x_k \in [\ell, \ell + k \cdot 2^w]$, since $|x_{k/2+1} - x_{k/2}|$ was maximal. Motivated by this observation, the *density* of width- w copies in \mathcal{C} around ℓ is measured by

$$\tau_{\mathcal{C}}(\ell, w) = \frac{1}{2^w} \cdot |\mathcal{C}_{\ell, w}|,$$

and the total density (over all widths) of \mathcal{C} around ℓ is measured by

$$\tau_{\mathcal{C}}(\ell) = \sum_{w=1}^{\log n} \tau_{\mathcal{C}}(\ell, w).$$

The algorithms of [NRRS17] and [BECLW19] proceed in a recursive manner. Each step (implicitly) considers an index $\ell \in [n]$ where the total density $\tau_{\mathcal{C}}(\ell)$ is high, namely at least $\Omega_k(\varepsilon)$, as well as a

width w where $\tau_{\mathcal{C}}(\ell, w)$ is high. At a very high level, the algorithm can recurse on the sub-intervals $[\ell - k \cdot 2^w, \ell]$ and $[\ell, \ell + k \cdot 2^w]$, where the lower bound on $\tau_{\mathcal{C}}(\ell, w)$ implies sufficiently many increasing subsequences exist in each interval. The number of possible widths w is $\Theta(\log n)$, and the algorithms consider all $\Theta(\log n)$ options in each recursive step, thereby losing a $\Theta(\log n)$ factor in the query complexity at each step. The main challenge of [NRRS17, BECLW19] is obtaining the “best” lower bound on $\tau_{\mathcal{C}}(\ell, w)$ for some $w \in [\log n]$ and determining the number of recursive steps necessary. In particular, the insight of [BECLW19] is that for a fixed ℓ with $\tau_{\mathcal{C}}(\ell) = \Omega(\varepsilon)$, if all $\tau_{\mathcal{C}}(\ell, w)$ are relatively small, i.e., $\tau_{\mathcal{C}}(\ell, w) \ll \varepsilon/k$ for all $w \in [\log n]$, then the monotone subsequences in the vicinity of ℓ must be arranged in an easy-to-find way. Therefore, we may assume that whenever $\ell \in [n]$ satisfies $\tau_{\mathcal{C}}(\ell) = \Omega_k(\varepsilon)$, there exists one (unknown) $w \in [\log n]$ where $\tau_{\mathcal{C}}(\ell, w) = \Omega_k(\varepsilon)$.⁴

With adaptivity, the hope is that an algorithm considering an index $\ell \in [n]$ with $\tau_{\mathcal{C}}(\ell) = \Omega_k(\varepsilon)$ can choose *one* (unknown) width w satisfying $\tau_{\mathcal{C}}(\ell, w) = \Omega_k(\varepsilon)$, and recurse only on that width. The algorithm may devote $\Theta_{k,\varepsilon}(\log n)$ queries to consider all $\Theta(\log n)$ possible widths, and the benefit is that recursing on a single width incurs a $\Theta_{k,\varepsilon}(\log n)$ *additive* loss in the query complexity, as opposed to the $\Theta_{k,\varepsilon}(\log n)$ *multiplicative* loss incurred by [NRRS17, BECLW19]. It is not too hard for an algorithm to choose a width \hat{w} where $\hat{w} \geq w$. For example, the algorithm may query $O_k(1/\varepsilon)$ randomly sampled indices from $[\ell - k \cdot 2^{w_0}, \ell]$ and $[\ell, \ell + k \cdot 2^{w_0}]$ for each $w_0 \in [\log n]$, and let \hat{w} be the largest w_0 where some increasing pair is found. The fact that the unknown $w \in [\log n]$ satisfies $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ implies that with high constant probability, indices x_1 and y_k from $(x_1, \dots, x_k), (y_1, \dots, y_k) \in \mathcal{C}_{\ell, w}$ are sampled, and by an observation from [NRRS17], with high enough probability, $f(x_1) \leq f(y_k)$ (see Appendix A for a more thorough discussion on this point). This, in turn, implies $\hat{w} \geq w$.

When \hat{w} is not much larger than w , our recursive step proceeds similarly to [NRRS17, BECLW19]; we call this the *fitting* case. The problem is that \hat{w} may be too large, a case we refer to as *overshooting*. Consider the execution selecting a too large width \hat{w} , for example, $\hat{w} = w_0 = \log n/2$ even though the “correct” width w satisfies $w \ll w_0$. The decision of setting \hat{w} to w_0 is based on finding an increasing pair among the $O_k(1/\varepsilon)$ sampled indices at distance $\Theta_k(\sqrt{n})$ from ℓ , and not finding an increasing pair at any larger distance. However, not only is it the case that increasing pairs at distance $\Theta_k(\sqrt{n})$ from ℓ may not combine to form length- k increasing subsequences, but also the density $\tau_{\mathcal{C}}(\ell, w_0)$ may be extremely small. Intuitively, the promise that $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ ensures that $\Omega_k(\varepsilon 2^w)$ length- k increasing subsequences exists in $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ which are cut with slack by ℓ . When w_0 is much larger, these length- k increasing subsequences constitute a tiny (at most $O_k(2^{w-w_0})$) fraction of the interval $[\ell - k \cdot 2^{w_0}, \ell + k \cdot 2^{w_0}]$ the algorithm would recurse on.

Due to the density $\tau_{\mathcal{C}}(\ell, w_0)$ being potentially very small, at this point, it is not clear how to proceed with our wrong (too large) choice of $\hat{w} = w_0$ as the width to recurse on. To overcome this, we prove a robust structural theorem, drawing a much more favorable picture as to which widths are good for recursion. The robust structural theorem asserts the following. For sufficiently many “well-behaved” $\ell \in [n]$ where $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$, any interval J containing $[\ell - k \cdot 2^w, \ell + k \cdot 2^w] \subset J$ has $\Omega_k(\varepsilon|J|)$ pairwise-disjoint length- k increasing subsequences. These length- k increasing subsequences are not cut with slack by ℓ , a condition which was crucial for [NRRS17, BECLW19]; however, the algorithm’s choice of \hat{w} means it found an increasing pair at distance $\Theta_k(2^{\hat{w}})$. We exploit this with an adaptive algorithm in a somewhat surprising manner, which we expand on now.

⁴The case where $\tau_{\mathcal{C}}(\ell) = \Omega(\varepsilon)$ but all values of $\tau_{\mathcal{C}}(\ell, w)$ are small is known as the *growing suffixes* case. Here, there is an $O_{k,\varepsilon}(\log n)$ algorithm even in the non-adaptive regime.

New Algorithm When Overshooting. Let $\ell \in [n]$ be a “well-behaved” index with $\tau_{\mathcal{C}}(\ell) \geq \Omega_k(\varepsilon)$, and let w be the unknown width where $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$. Suppose that the widest increasing pair (\mathbf{x}, \mathbf{y}) found by the algorithm (which sets $\widehat{w} \approx \log_2 |\mathbf{y} - \mathbf{x}|$), satisfies $\widehat{w} \gg w$. Even though the algorithm has “committed” to a width \widehat{w} which is too large, we will algorithmically exploit the fact that (\mathbf{x}, \mathbf{y}) is an increasing pair lying very far apart, and containing the interval $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$. Specifically, since (\mathbf{x}, \mathbf{y}) are very far away, the algorithm may fit $k - 2$ intervals J_1, \dots, J_{k-2} between \mathbf{x} and \mathbf{y} which lie adjacent to each other, satisfying the following conditions:

- J_1 contains the interval $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$.
- J_{i+1} lies immediately after J_i , for any $i \in [k - 3]$.
- $|J_{i+1}| \geq |J_i| \cdot \alpha_{k,\varepsilon}$ for all $i \in [k - 3]$, and a large fixed constant $\alpha_{k,\varepsilon} > 1$.

A consequence of the robust structural theorem, and the fact that J_1, \dots, J_{k-2} have exponentially increasing lengths is that each J_i contains a collection \mathcal{T}_i of $\Omega_k(\varepsilon|J_i|)$ disjoint length- k increasing subsequences. For each $i \in [k - 2]$, define two sets \mathcal{A}_i and \mathcal{B}_i as follows. Let \mathcal{A}_i be the collection of prefixes (a_1, \dots, a_{i+1}) of \mathcal{T}_i with $f(a_{i+1}) < f(\mathbf{y})$, and let \mathcal{B}_i be the collection of suffixes (a_{i+1}, \dots, a_k) of \mathcal{T}_i with $f(a_{i+1}) \geq f(\mathbf{y})$. 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 an increasing subsequence of length k . Specifically, the algorithm to handle overshooting aims to (recursively) find shorter increasing subsequences in J_1, \dots, J_{k-2} , with the hope of combining them together into an increasing subsequence of length k . Concretely, for any $i \in [k - 2]$, we make two recursive calls of our algorithm on J_i : one for an $(i + 1)$ -increasing subsequence in J_i , with values smaller than $f(\mathbf{y})$,⁵ and a second one for a $(k - i)$ -increasing subsequence in J_i whose values are at least $f(\mathbf{y})$. 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 $f(\mathbf{y}) > f(\mathbf{x})$, which combines with \mathbf{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 $f(\mathbf{y})$, which combines with \mathbf{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 $f(\mathbf{y})$, and a length- $(k - i - 1)$ monotone pattern in J_{i+1} with values at or above $f(\mathbf{y})$; together these two patterns combine to form a $(12 \dots k)$ -pattern.

In all cases, a k -increasing subsequence is found with good probability. See Figure 2 for an example. The benefit is that the algorithm spends $\Theta_{k,\varepsilon}(\log n)$ queries to identify one fixed width $\widehat{w} \in [\log n]$. Then, there are $2(k - 2)$ recursive calls each aiming to find an increasing subsequence of length strictly less than k . The $\Theta_{\varepsilon,k}(\log n)$ loss in the query complexity is additive per recursive step;

⁵Technically 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.

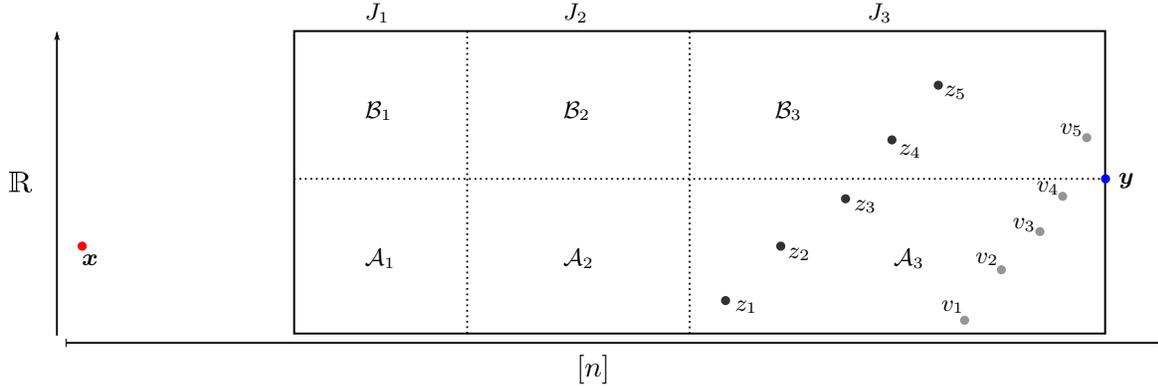


Figure 2: We consider the “overshooting case” for $k = 5$. Specifically, the algorithm considers an index $\ell \in [n]$ with $\tau_{\mathcal{C}}(\ell) = \Omega_k(\varepsilon)$ and, for some unknown $w \in [\log n]$, $\tau_{\mathcal{C}}(\ell, w) = \Omega_k(\varepsilon)$. Furthermore, in trying to identify a correct width \hat{w} , the algorithm samples an increasing pair (\mathbf{x}, \mathbf{y}) with $\log_2 |\mathbf{x} - \mathbf{y}| \approx \hat{w} \gg w$. The algorithm will consider at least $k - 2$ geometrically increasing intervals between \mathbf{x} and \mathbf{y} ; these are displayed as J_1, J_2 , and J_3 ; by virtue of the robust structural theorem, each J_i contains $\Omega_k(\varepsilon|J_i|)$ disjoint length- k monotone subsequences. \mathcal{A}_i contains those length- k monotone subsequences where the $(i + 1)$ -th index is above $f(\mathbf{y})$ and \mathcal{B}_i contains those whose $(i + 1)$ -th index is below $f(\mathbf{y})$. As an example, $(z_1, z_2, z_3, z_4, z_5) \in \mathcal{B}_4$ and $(v_1, v_2, v_3, v_4, v_5) \in \mathcal{A}_4$. The crucial properties are: (i) for all $i \in [k - 2]$ any $(12 \dots i)$ -pattern in \mathcal{A}_i and any $(12 \dots (k - i))$ -pattern in \mathcal{B}_{i+1} may be combined into a $(12 \dots k)$ -pattern, (ii) any $(12 \dots (k - 1))$ -pattern in \mathcal{B}_1 may be combined with \mathbf{x} since $f(\mathbf{y}) > f(\mathbf{x})$, and (iii) any $(12 \dots (k - 1))$ -pattern in \mathcal{A}_4 may be combined with \mathbf{y} . The reasoning may proceed as follows: if $|\mathcal{B}_1|$ is large, we find a $(12 \dots (k - 1))$ -pattern and combine it with \mathbf{x} ; so, assume $|\mathcal{B}_1|$ is small, which implies $|\mathcal{A}_1|$ must be large. If $|\mathcal{B}_2|$ is large, then a (12) -pattern from \mathcal{A}_1 and a $(12 \dots (k - 2))$ -pattern from \mathcal{B}_2 may be combined; so assume $|\mathcal{B}_2|$ is small which implies $|\mathcal{A}_2|$ is large, \dots . Eventually, we deduce that we may assume $|\mathcal{A}_{k-2}|$ is large, and a $(12 \dots (k - 1))$ -pattern in \mathcal{A}_{k-2} may be combined with \mathbf{y} .

this leads to the $\Theta_{\varepsilon, k}(\log n)$ query complexity bound which was impossible in the non-adaptive algorithms of [NRRS17, BECLW19], as these needed to explore all possible widths $\hat{w} \in [\log n]$ in each recursive step.

Organization The rest of the paper is organized as follows. In Section 1.3, we present relevant notation. Section 2 establishes the stronger structural result required for our adaptive algorithm. Section 3 contains all new algorithmic components and the proof of the $O_{k, \varepsilon}(\log n)$ query complexity upper bound for our algorithm. For convenience, in Appendix A we provide a brief description of the previous (non-adaptive) testing results on $(12 \dots k)$ -freeness from [NRRS17, BECLW19].

1.3 Notation

All logarithms considered are base 2. We consider functions $f: I \rightarrow \mathbb{R}$, 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 $I = \{a, a + 1, \dots, b\}$. At many places throughout the paper, we think of augmenting the image with a special character $*$

to consider $f: I \rightarrow \mathbb{R} \cup \{*\}$. The character $*$ 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 $f: I \rightarrow \mathbb{R} \cup \{*\}$, the set T is a collection of disjoint monotone subsequences of length k if it consists of tuples $(i_1, \dots, i_k) \in I^k$, where $i_1 < \dots < i_k$ and $f(i_1) < \dots < f(i_k)$ (in particular, $f(i_1), \dots, f(i_k) \neq *$), and furthermore, for any two tuples (i_1, \dots, i_k) and (i'_1, \dots, 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., $E(T) = \cup_{(i_1, \dots, i_k) \in T} \{i_1, \dots, i_k\}$. Finally, we let $\text{poly}(\cdot)$ denote a large enough polynomial whose degree is (bounded by) a universal constant.

2 Stronger Structural Dichotomy

In this section, we prove a robust structural dichotomy for functions $f: [n] \rightarrow \mathbb{R}$ that are ε -far from $(12 \dots k)$ -free, which strengthens the dichotomy proved in [BECLW19]. In their paper, it is shown 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*. In Section 2.1, we define and describe these notions and state the original (non-robust) structural result from [BECLW19]. Then, in Section 2.2, we establish a substantially stronger structural dichotomy, better suited for our purposes. The proof of the stronger dichotomy combines the original one as a black-box with additional combinatorial ideas.

2.1 The Non-Robust Structural Decomposition

For completeness, we first introduce the non-robust structural result from [BECLW19]. As the formal definitions are somewhat complicated, we start with an informal description of the growing suffixes and splittable intervals conditions. For the purpose of this discussion, let \mathcal{C} be any collection of $\Theta_{k,\varepsilon}(n)$ disjoint $(12 \dots k)$ -copies in f . We use the notation introduced in Section 1.2.

- **Growing suffixes:** there exist $\Omega_{k,\varepsilon}(n)$ values of $\ell \in [n]$ where $\tau_{\mathcal{C}}(\ell) \geq \Theta_k(\varepsilon)$ and $\tau_{\mathcal{C}}(\ell, w) \ll \tau_{\mathcal{C}}(\ell)$ for every $w \in [\log n]$. In words, many $\ell \in [n]$ are such 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 (non-robust):** 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 , the last $k - c$ entries lie in R_i (none of these entries lies in M_i), and every such c entry lies below every $c + 1$ entry.

Informally, the non-robust structural dichotomy from [BECLW19] asserts that any f that is ε -far from $(12\dots k)$ -free either satisfies the growing suffixes condition, or the non-robust splittable intervals condition (or both). These two notions are formally defined next; the precise definition for growing suffixes is slightly more complicated than described above (but understanding it is not essential for this work, as the growing suffixes procedure from [BECLW19] will eventually only be used as a black box). 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 $S_1, \dots, S_{\eta_\ell}$ are a partition of $(\ell, n]$ into intervals of geometrically 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 $|D_t(\ell)|/|S_t(\ell)| \leq \alpha$ for all $t \in [\eta_\ell]$, and $\sum_{t=1}^{\eta_\ell} |D_t(\ell)|/|S_t(\ell)| \geq \beta$.*
2. *For every $t, t' \in [\eta_\ell]$ 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*

$$T^{(L)} = \{(i_1, \dots, i_c) \in I^c : (i_1, \dots, i_c) \text{ is a prefix of a } k\text{-tuple in } T\}, \text{ and}$$

$$T^{(R)} = \{(j_1, \dots, j_{k-c}) \in I^{k-c} : (j_1, \dots, j_{k-c}) \text{ is a suffix of a } k\text{-tuple in } T\}.$$

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 $(i_1, \dots, i_c) \in T^{(L)}$ and $(j_1, \dots, j_{k-c}) \in T^{(R)}$; and there is a partition of I into three consecutive 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 $(I_1, T_1), \dots, (I_s, T_s)$ 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]. We remark that in their paper, the theorem is stated with respect to two parameters, k, k_0 ; for our purpose it suffices to set $k_0 = k$. Also, here we allow f to take the value $*$, which is not the case in [BECLW19]. Nevertheless, as their proof takes into account only the elements of a given family T^0 of disjoint length- k increasing subsequences, which in particular are non- $*$ elements, the same proof would work here.

Theorem 2.3 ([BECLW19], Theorem 2.2). *Let $k, n \in \mathbb{N}$, $\varepsilon \in (0, 1)$, and $C > 0$, and let $I \subseteq [n]$ be an interval. Let $f: I \rightarrow \mathbb{R} \cup \{*\}$ 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 $p \leq \text{poly}(k \log(1/\varepsilon))$ 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 $(I_1, T_1), \dots, (I_s, T_s)$, such that

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

We do not include the proof here, as it is rather technical (though still lightweight compared to the “iterated structural result” required for non-adaptive algorithms, see [BECLW19, Theorem 2.3]).

2.2 Robustifying the Structural Result

We are now ready to establish the robust structural foundations – specifically, a *growing suffixes* versus *robust splittable intervals* dichotomy – lying at the heart of our adaptive algorithm. The next lemma will eventually imply that the splittable intervals 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 $I_1, \dots, I_s \subset I$ 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 $h \in G$,

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

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 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. \square

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

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

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 from the definition of B , 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 $\sum_{h \in [s]: I_h \subset J} |I_h| \geq (\alpha/4)|J|$, as required. \square

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 $f: I \rightarrow \mathbb{R} \cup \{*\}$ 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 $p \leq \text{poly}(k \log(1/\varepsilon))$ 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 $1 \leq c < k$, 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 $(I_1, T_1), \dots, (I_s, T_s)$, such that*

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

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

Proof. Apply Theorem 2.3. Let $\alpha^* \in (0, 1)$ and p^* be parameters such that $\alpha^* \geq \Omega(\varepsilon/k^5)$ and $p^* \leq \text{poly}(k \log(1/\varepsilon))$, 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 $(I_1, T_1), \dots, (I_s, T_s)$ be as guaranteed there. In particular, we have $\sum_{h=1}^s |I_h| \geq (1/p^* \alpha^*) |T^0|$. By Lemma 2.4, there is a subset $G \subset [s]$ such that $\sum_{h \in G} |I_h| \geq (1/4p^* \alpha^*) |T^0| \geq (\varepsilon/4p^* \alpha^*) |I| = (\varepsilon/p\alpha) |I|$; and, for every interval J in I that contains an interval I_h with $h \in [G]$, $\sum_{h \in [s]: I_h \subset J} |I_h| \geq (\varepsilon/4p^* \alpha^*) |J|$. 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. \square

3 The Algorithm

Our aim in this section is to prove the existence of a randomized algorithm, **Find-Monotone** $_k(f, \varepsilon, \delta)$, that receives as input a function $f: I \rightarrow \mathbb{R} \cup \{*\}$ (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 $(12 \dots k)$ -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 5 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 3 and 4. The majority of the section is devoted to the proof that **Find-Monotone** indeed outputs a $(12 \dots k)$ -pattern with high probability as claimed. Specifically, we shall prove the following theorem.

Theorem 3.1. *Let $k \in \mathbb{N}$. The randomized algorithm **Find-Monotone** $_k(f, \varepsilon, \delta)$, described in Figure 5, satisfies the following. Given a function $f: I \rightarrow \mathbb{R} \cup \{*\}$ and parameters $\varepsilon, \delta \in (0, 1)$, if f contains at least $\varepsilon |I|$ disjoint $(12 \dots k)$ -patterns, then **Find-Monotone** $_k(f, \varepsilon, \delta)$ outputs a $(12 \dots k)$ -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 former is taken from [BECLW19] whereas the proofs of the latter two 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 $f: I \rightarrow \mathbb{R} \cup \{*\}$, 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*

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

3.1 The Sample-Suffix Sub-Routine

We restate 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-Suffixk`(f, ε, δ) which takes three inputs: query access to a function $f: I \rightarrow \mathbb{R} \cup \{*\}$, where $I \subset \mathbb{N}$ 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-Suffixk`(f, ε, δ) finds a length- k monotone subsequence of f with probability at least $1 - \delta$. The query complexity of `Sample-Suffixk`(f, ε, δ) is at most*

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

There are two differences between this statement and the statement of Lemma 3.1 from [BECLW19].

First, here we have error probability δ , whereas in [BECLW19] the error probability is $1/10$. In order to achieve error probability δ in Lemma 3.1 of [BECLW19], we perform $O(\log(1/\delta))$ independent repetitions of `Sample-Suffixk`, as described in [BECLW19]. These are reflected in the query complexity.

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

The algorithm whose existence is guaranteed by Theorem 3.3 is very simple. First “guess” the value of α (up to a factor of 2); as $\varepsilon \leq \alpha \leq 1$, this comes at a cost of a factor of $O(\log(1/\varepsilon))$ in the query complexity. For each guess α' for the value of α , sample, uniformly at random, $\Theta(\alpha/\varepsilon\delta)$ values of $\ell \in I$. As $|H| \geq (\varepsilon/\alpha)|I|$, with probability at least $1 - \delta/2$ a value $\ell \in H$ is sampled in this way. Finally, for each ℓ sampled in the previous step, and for each $t \in [\log n]$, sample, uniformly at random, $\Theta(1/(\alpha'\delta))$ values in $(\ell + 2^{t-1}, \ell + 2^t]$. For a value $\ell \in H$ sampled in the previous step, denote by E_t the event that an element of $D_t(\ell)$ is sampled in the last step (recall the definition of $D_t(\ell)$ from Theorem 2.1). By the properties of the sets $D_t(\ell)$, a set of elements from distinct sets $D_t(\ell)$ forms an increasing subsequence. It thus suffices to show that, with probability at least $1 - \delta/2$, the sum $\sum_{t \in [\log n]} E_t$ is at least k . To do this, we utilize the assumption on the sets $D_t(\ell)$ which asserts that the sum $\sum_t |D_t(\ell)|/2^{t-1}$ is large (at least $\Omega(k\alpha)$) while the value of $|D_t(\ell)|/2^{t-1}$ is small (at most α) for every t , to conclude that the expectation of $\sum_t E_t$ is large ($\Omega(k)$), which implies that it is at least k with probability at least $1 - \delta/2$.

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.

As the algorithm may appear unintuitive, let us remind the reader of the setup in which this subroutine is relevant (see also Section 1.2). By Theorem 2.6, either the growing suffixes condition or the splittable intervals condition hold. The former case is handled by Theorem 3.3, so we assume that the latter holds. Now assume that we sampled an element \mathbf{x} which is the first element of a length- c increasing subsequence from a set L_i as described in Theorem 2.2. We then sample,

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

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

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

1. For every $\kappa \in [k-2]$, let $f_\kappa, f'_\kappa: J_\kappa \rightarrow \mathbb{R} \cup \{*\}$ be given by:

$$f_\kappa(i) = \begin{cases} f(i) & f(i) < f(y) \\ * & \text{o.w.} \end{cases} \quad \text{and} \quad f'_\kappa(i) = \begin{cases} f(i) & f(i) \geq f(y) \\ * & \text{o.w.} \end{cases}. \quad (3)$$

2. Call **Find-Monotone** $_{\kappa+1}(f_\kappa, \varepsilon/2, \delta/(2k))$ for every $\kappa \in [k-2]$.
3. Call **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 3: Description of the Find-Within-Interval subroutine.

uniformly at random, elements \mathbf{y} from $[\mathbf{x}, \mathbf{x} + 2^t]$. The splittable intervals condition implies that we will find, with high probability, an element \mathbf{y} which is the last element of a length- $(k-c)$ increasing subsequence from R_i . In particular, $f(\mathbf{y}) > f(\mathbf{x})$. However, even if we did indeed sample such \mathbf{y} , we may have sampled many other values of \mathbf{y}' with $f(\mathbf{y}') > f(\mathbf{x})$, and we do not know of a way of determining which of these values is the “correct” one. Instead, we take \mathbf{y}_0 to be the largest sampled \mathbf{y}' such that $f(\mathbf{y}') > f(\mathbf{x})$. The case where \mathbf{y}_0 is close to \mathbf{y} is taken care of by Theorem 3.4, so we assume that \mathbf{y}_0 is much larger than \mathbf{y} .

We now have elements \mathbf{x} and \mathbf{y}_0 , and all that we know is that they contain a large portion of an interval I_i from the splittable intervals condition. It is not hard to see (this is shown in the proof of Theorem 3.1) that $[\mathbf{x}, \mathbf{y}_0]$ can be partitioned into $k-2$ intervals J_1, \dots, J_{k-2} , each of which contains many disjoint length- k increasing subsequences. To continue, our only hope is use the induction hypothesis to find shorter increasing subsequences in the intervals. For example, if there are many disjoint length- $(k-1)$ increasing subsequences in J_1 that lie above \mathbf{x} , then one such subsequence is likely to be detected by a recursive call to the main algorithm, and together with \mathbf{x} it will form a length- k increasing subsequence. If there are few such length- $(k-1)$ subsequences, this means that there are many disjoint length-2 increasing subsequences in J_1 that lie below \mathbf{x} (because for every length- k increasing subsequence, either its $(k-1)$ -suffix lies above \mathbf{x} , or its 2-prefix lies above \mathbf{x}). We can then use a recursive call to detect such a sequence, and hope to complete it to a length- k subsequence using a length- $(k-2)$ subsequence from J_2 that lies above \mathbf{x} . Continuing with this logic, it follows that with high probability we can find an increasing subsequence of length k using \mathbf{x} and J_1, J_i and J_{i+1} for some i , or J_{k-2} and \mathbf{y}_0 .

Lemma 3.4. *Consider the randomized algorithm, **Find-Within-Interval** $_k(f, \varepsilon, \delta, x, y, \mathcal{J})$, de-*

scribed in Figure 3, which takes six inputs:

- Query access to a function $f: I \rightarrow \mathbb{R} \cup \{*\}$,
- Two parameters $\varepsilon, \delta \in (0, 1)$,
- Two points $x, y \in I$ where $x < y$ and $f(x) < f(y)$, and
- A collection $\mathcal{J} = (J_1, \dots, J_{k-2})$ of $k-2$ disjoint intervals that 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 $f|_{J_\kappa}: J_\kappa \rightarrow \mathbb{R} \cup \{*\}$, contains $\varepsilon|J_\kappa|$ disjoint $(12\dots k)$ -patterns. Then, assuming that Theorem 3.1 holds for every k' with $1 \leq k' < k$, the procedure **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}_κ be a collection of at least $\varepsilon|J_\kappa|$ disjoint $(12\dots k)$ -patterns in J_κ . We form the following two collections, of suffixes and prefixes of $(12\dots k)$ -patterns in \mathcal{C}_κ .

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

Note that for each $(12\dots k)$ -pattern in \mathcal{C}_κ , either its $(\kappa+1)$ -prefix is in \mathcal{A}_κ , or its $(k-\kappa)$ -suffix is in \mathcal{B}_κ . Thus, at least one of \mathcal{A}_κ and \mathcal{B}_κ 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 $(12\dots(\kappa+1))$ -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 $(12\dots(k-\kappa))$ -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- $*$ elements in f_κ lie below the non- $*$ elements of $f'_{\kappa+1}$, we can combine the $(12\dots(\kappa+1))$ -pattern in f_κ with the $(12\dots(k-\kappa-1))$ -pattern in $f'_{\kappa+1}$. \square

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 **Find-Good-Split** $_k(f, \varepsilon, \delta, c, \xi)$.

Input: Query access to a function $f: I \rightarrow \mathbb{R} \cup \{*\}$, 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 $i_1 < \dots < i_k$ with $f(i_1) < \dots < f(i_k)$, or fail.

1. Repeat the following procedure $t = c_1 k / (\varepsilon \xi^2) \cdot \log(1/\delta)$ times:

(a) Sample $\mathbf{w}, \mathbf{z} \sim I$, and consider the functions $f_{\mathbf{z}, \mathbf{w}}: I \cap (-\infty, \mathbf{z}) \rightarrow \mathbb{R} \cup \{*\}$ and $f'_{\mathbf{z}, \mathbf{w}}: I \cap [\mathbf{z}, \infty) \rightarrow \mathbb{R} \cup \{*\}$ given by

$$f_{\mathbf{z}, \mathbf{w}}(i) = \begin{cases} f(i) & f(i) < f(\mathbf{w}) \\ * & \text{o.w.} \end{cases} \quad \text{and} \quad f'_{\mathbf{z}, \mathbf{w}}(i) = \begin{cases} f(i) & f(i) \geq f(\mathbf{w}) \\ * & \text{o.w.} \end{cases}. \quad (4)$$

(b) Run **Find-Monotone** $_c(f_{\mathbf{z}, \mathbf{w}}, \varepsilon \xi / 3, \delta / 3)$ and **Find-Monotone** $_{k-c}(f'_{\mathbf{z}, \mathbf{w}}, \varepsilon \xi / 3, \delta / 3)$.

2. If both runs of Line 1b are successful for some iteration and some \mathbf{w}, \mathbf{z} and c , then we output the combination of their outputs which forms a length- k increasing subsequence of f ; otherwise, output fail.

Figure 4: Description of the Find-Good-Split subroutine.

Lemma 3.5. Consider the randomized algorithm **Find-Good-Split** $_k(f, \varepsilon, \delta, c, \xi)$, described in Figure 4, which takes as input five parameters:

- Query access to a function $f: I \rightarrow \mathbb{R} \cup \{*\}$,
- 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 **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

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

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

as the $(|T|/3)$ -largest and $(2|T|/3)$ -largest elements in $\{f(i_c) : (i_1, \dots, i_c) \in T^{(L)}\}$ (taking multiplicity into account). Let $T_l^{(L)}$ be the $(12 \dots c)$ -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 $(12 \dots c)$ -pattern from $T_l^{(L)}$ can be combined with any $(12 \dots (k - c))$ -pattern from $T_h^{(R)}$ to form a $(12 \dots k)$ -pattern. Furthermore, there exists at least $|T|/3$ indices in I' whose function value lies in $[m_1, m_2]$.

Consider the event, defined over the randomness of $\mathbf{w}, \mathbf{z} \sim I$ that: $\mathbf{z} \in M$; and \mathbf{w} satisfies $f(\mathbf{w}) \in [m_1, m_2]$. 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 \mathbf{z} , and there are at least $|T|/3 \geq (\varepsilon/3)|I'| \geq (\varepsilon\xi/3)|I|$ indices for \mathbf{w} , so the probability that the pair (\mathbf{z}, \mathbf{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 $(12 \dots c)$ -patterns in $T_l^{(L)}$ all lie in $f_{\mathbf{z}, \mathbf{w}}$, and the $(12 \dots (k - c))$ -patterns in $T_h^{(R)}$ all lie in $f'_{\mathbf{z}, \mathbf{w}}$. In particular, $f_{\mathbf{z}, \mathbf{w}}$ contains at least $|T|/3 \geq (\varepsilon/3)|I'| \geq (\varepsilon\xi/3)|I|$ disjoint $(12 \dots c)$ -patterns, and $f'_{\mathbf{z}, \mathbf{w}}$ similarly contains at least $(\varepsilon\xi/3)|I|$ disjoint $(12 \dots (k - c))$ -patterns. Thus, by the inductive hypothesis, Line 1b finds a $(12 \dots c)$ -pattern in $f_{\mathbf{z}, \mathbf{w}}$ and a $(12 \dots (k - c))$ -pattern in $f'_{\mathbf{z}, \mathbf{w}}$ with probability at least $1 - 2\delta/3$, and these can be combined to give a $(12 \dots k)$ -pattern of f . \square

3.4 The Main Algorithm

Consider the description of the main algorithm in Figure 5. 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 $\mathbf{x} \sim I$ 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 5, samples $\mathbf{x} \neq *$ 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 $\text{Sample-Suffix}_k(f, \varepsilon/p, \delta)$ in Line 1 outputs a length- k monotone subsequence of f with probability at least $1 - \delta$.

Now suppose that (2) of Theorem 2.6 holds, and let $(I_1, T_1), \dots, (I_s, T_s)$ 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

Subroutine **Find-Monotone** $_k(f, \varepsilon, \delta)$.

Input: Query access to a function $f: I \rightarrow \mathbb{R} \cup \{*\}$, 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 $P: \mathbb{R} \rightarrow \mathbb{R}$ is a polynomial of large enough (constant) degree.

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

1. Run **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 $\mathbf{x} \sim I$ uniformly at random. If $f(\mathbf{x}) = *$, proceed to the next iteration. Otherwise, if $k = 1$ output \mathbf{x} and proceed to Step 3, and if $k \geq 2$ proceed to the next step.
 - (b) For each $t \in [\log n]$, sample $\mathbf{y}_t \sim [\mathbf{x} + 2^t/(12k), \mathbf{x} + 2^t]$ uniformly at random. If there exists at least one t where $f(\mathbf{y}_t) > f(\mathbf{x})$, set

$$\mathbf{y} = \max \{ \mathbf{y}_t : t \in [\log n] \text{ and } f(\mathbf{y}_t) > f(\mathbf{x}) \}, \quad (5)$$

let $t^* \in [\log n]$ be the index for which $\mathbf{y}_{t^*} = \mathbf{y}$, and continue to the next line. Otherwise, i.e. if $f(\mathbf{y}_t) \not> f(\mathbf{x})$ for every t , continue to the next iteration.

- (c) If $k = 2$, output (\mathbf{x}, \mathbf{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, \dots, J_{k-2} appearing in order within $[\mathbf{x}, \mathbf{y}]$, given by setting, for every $i \in [k - 2]$,

$$J_i = \left[\mathbf{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-1-i)}, \mathbf{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)} \right), \quad (6)$$

and run **Find-Within-Interval** $_k(f, \varepsilon/2p, \delta/2, \mathbf{x}, \mathbf{y}, \mathcal{J})$.

- ii. For each $t' \in [t^* - 3k \log \ell, t^*]$ do the following. Consider the interval $J_{t'} = [\mathbf{x} - 2^{t'}, \mathbf{x} + 2^{t'}]$, and the restricted function $g_{t'}: J_{t'} \rightarrow \mathbb{R} \cup \{*\}$ given by $g_{t'} = f|_{J_{t'}}$. For every $c_0 \in [k - 1]$, run **Find-Good-Split** $_k(g_{t'}, \varepsilon/(c_2 k^5), \delta/2, c_0, 1/4)$.

3. If a length- k monotone subsequence of f is found, output it. Otherwise, output fail.

Figure 5: Description of the **Find-Monotone** $_k$ subroutine.

$J \supset I_h$ for some $h \in [s]$ contains $(\varepsilon/p)|J|$ disjoint $(12 \dots k)$ -patterns. Let **Event** be the event that, for a particular iteration of Lines 2a and 2b, \mathbf{x} is the 1-entry of some k -tuple from T_h , for some $h \in [s]$, and \mathbf{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. $\Pr[\text{Event}] \geq \varepsilon\alpha/(2p)$.

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 $\mathbf{x} \sim I$ in Line 2a that $\mathbf{x} \in A_h$ is at least ε/p . Fix such an \mathbf{x} , and consider $t \in [\log n]$ for which $|I_h| \leq 2^t < 2|I_h|$. Notice that $B_h \subset [\mathbf{x} + 2^t/(12k), \mathbf{x} + 2^t]$ 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 $\mathbf{y}_t \sim [\mathbf{x} + 2^t/(12k), \mathbf{x} + 2^t]$ that $\mathbf{y}_t \in B_h$ is at least $|B_h|/2^t \geq |T_h|/(2|I_h|) \geq \alpha/2$. \square

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 \mathbf{x} is the first index of a k -tuple in T_h , \mathbf{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 \mathbf{y} , as defined in (5), satisfies $\mathbf{y} \geq \mathbf{y}_t$ (as $f(\mathbf{y}) > f(\mathbf{x})$) and hence $t^* \geq t$.

Note that if $k = 2$, the pair (\mathbf{x}, \mathbf{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, \dots, J_{k-2} appear in order from left-to-right, and they lie in $[\mathbf{x}, \mathbf{y}]$ (as $\mathbf{y} = \mathbf{y}_{t^*} \in [\mathbf{x} + 2^{t^*}/(12k), 2^{t^*}]$). Second, in the next claim we show that for every $i \in [k-2]$, the interval J_i contains $(\varepsilon/2p)|J_i|$ disjoint $(12 \dots k)$ -patterns.

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

Proof. Let J'_i be the interval given by

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

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. \square

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 $g: J \rightarrow \mathbb{R} \cup \{*\}$ 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 **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. \square

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 $O(k)$.⁶
- 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 ,⁷ 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.

⁶In fact, this complexity can be improved to $O(1)$ if, instead of working with functions of the form $f: I \rightarrow \mathbb{R} \cup \{*\}$, we would have worked with function $f: I \rightarrow \mathbb{R}$ and received the interval of “non- $*$ values” as an input to the recursive call.

⁷More 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 $P: \mathbb{R} \rightarrow \mathbb{R}$ 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 $\text{Find-Monotone}_k(f, \varepsilon, \delta)$. Let

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

By Lemma 3.3, as well as an inspection of Figure 3 and Figure 4, we have:

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

Lastly, inspecting Figure 5, we have

$$\begin{aligned}\Phi(k, \varepsilon, \delta) &\leq \Phi^{(1)}(k, \varepsilon/p_{k,\varepsilon}, \delta) + \\ &\quad c_1 p_{k,\varepsilon} \cdot \frac{k^5}{\varepsilon^2} \cdot \log(1/\delta) \cdot \left(1 + \log n + \Phi^{(2)}(k, \varepsilon/(2p_{k,\varepsilon}), \delta/2) + \Phi^{(3)}(k, \varepsilon/(c_2 k^5), \delta/2, 1/4)\right) \\ &\leq q_{k,\varepsilon} \cdot \frac{1}{\varepsilon^2} \cdot \log(1/\delta) \cdot \log n + q_{k,\varepsilon} \cdot \frac{1}{\varepsilon^3} \cdot (\log(1/\delta))^2 \cdot \Phi(k - 1, \varepsilon/q_{k,\varepsilon}, \delta/(3k)) \\ &\leq \left(k^k \cdot (\log(1/\varepsilon))^k \cdot \frac{1}{\varepsilon} \cdot \log(1/\delta)\right)^{O(k)} \cdot \log n,\end{aligned}$$

where $Q: \mathbb{R} \rightarrow \mathbb{R}$ 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 $(k^k (\log(1/\varepsilon))^k (1/\varepsilon) \log(1/\delta))^{O(k)}$. \square

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)*, pages 1469–1494, 2019.
- [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 *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:16, 2019.
- [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.

- [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.
- [EL19] Chaim Even-Zohar and Calvin Leng. Counting small permutation patterns. Available as arXiv:1911.01414, 2019.
- [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):252–271, 1996.
- [RSSS19] Aviad Rubinfeld, 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)*, pages 1121–1145. 2019.
- [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.

A Non-Adaptive Algorithms

For completeness, in this appendix we briefly review the previously known non-adaptive algorithms for our problem of testing for $(12\dots k)$ -patterns. The relevant notions appear in Section 1.2. In particular, $f, \mathcal{C}, \mathcal{C}_{\ell, w}$ are as defined in the beginning of that section.

A.1 The $(\log n)^{O(k^2)}$ Algorithm of Newman et al. [NRRS17]

For simplicity, let us first focus on the case $k = 2$ (i.e., monotonicity testing). 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 $f(x) \leq m$, and the “2-entry” y' of some $(x', y') \in \mathcal{C}_{\ell, w}$ where $f(x') \geq m$, then (x, y') would form an increasing pair, since $x < \ell < y'$ and $f(x) \leq m \leq f(x') < f(y')$. By definition, the number of entries x as well as the number of entries y' within $[\ell - 2^w, \ell + 2^w]$ as above is at least $\Omega(\tau_{\mathcal{C}}(\ell, w) \cdot 2^w)$. 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 such y' , which together 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 $\mathbb{E}_{\ell \in [n]}[\tau_{\mathcal{C}}(\ell)] = \Omega(\varepsilon)$. Moreover, $\tau_{\mathcal{C}}(\ell, w) \leq O(1)$ 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 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 $\Theta(\log n/\varepsilon)$ 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 same reasoning is applied recursively, leading to the $(\log n)^{O(k^2)}$ -query algorithm of [NRRS17].

A.2 The $O((\log n)^{\lceil \log_2 k \rceil})$ Algorithm of Ben-Eliezer et al. [BECLW19]

The improved non-adaptive algorithm [BECLW19] makes use of the (non-robust) structural dichotomy discussed in Section 2.1. Consider a sequence f that is ε -far from $(12\dots k)$ -free. Then f contains a family \mathcal{C} of $\Omega_k(\varepsilon n)$ disjoint length- k increasing subsequences. This family \mathcal{C} can be replaced by a family \mathcal{C}' of size $\Omega_k(\varepsilon n)$ that has the additional so-called “greedy” property: if $(x_1, \dots, x_k), (y_1, \dots, y_k) \in \mathcal{C}'$ and if $x_1 < y_1, x_c < y_c$ and $x_{c+1} > y_{c+1}$, then $f(x_{c+1}) > f(y_{c+1})$ (see Figure 1). Such a family \mathcal{C}' can be constructed from \mathcal{C} using a simple greedy algorithm.

Given such a family \mathcal{C}' , suppose that ℓ is known to have the “growing suffixes” property (with respect to the family \mathcal{C}'). Due to the “greedy” property of the family \mathcal{C}' , given a set of length- k increasing subsequences that are cut by ℓ with slack and whose widths are sufficiently far apart, their $(c+1)$ -values are guaranteed to form an increasing subsequence. The growing suffix assumption on ℓ implies that, roughly speaking, there are many widths for which there are many length- k increasing subsequences that are cut by ℓ with slack. Thus, one can show that by sampling $\Theta_k(1/\varepsilon)$ elements from the interval $[\ell, \ell + 2^t]$, for each $t \in [\log n]$, a k -increasing subsequence is found, with high

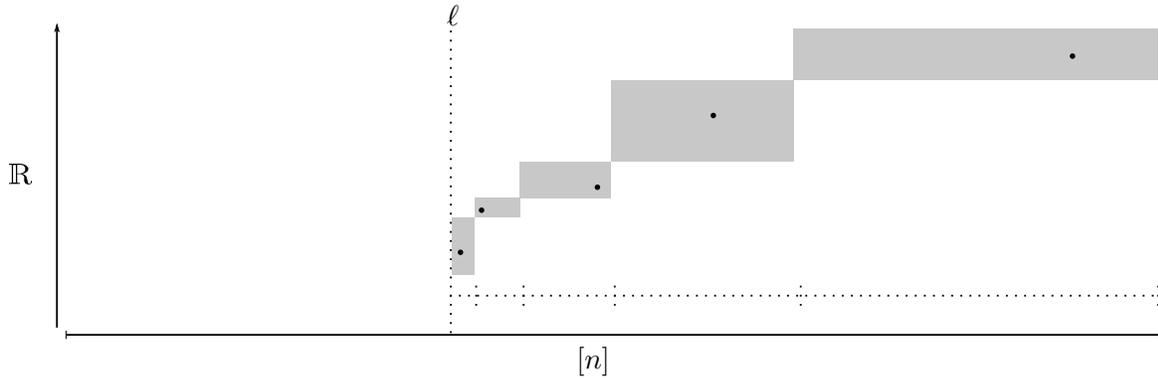


Figure 6: A sequence $f: [n] \rightarrow \mathbb{R}$ is displayed which has a growing suffix starting at position ℓ . Specifically, suppose we consider all length- k monotone subsequences belonging to a family \mathcal{C}' of size $\Omega_{\varepsilon,k}(n)$ which satisfies the “greedy” property (see Figure 1), and in all of which the c -index and $(c+1)$ -index form the largest gap between consecutive elements. We have divided intervals beginning at ℓ at geometrically increasing lengths, and shaded those intervals which contain $\Omega_{\varepsilon,k}(1)$ -fraction of indices as a $(c+1)$ -index of a k -tuple in \mathcal{C}' which is cut with slack by ℓ at the particular width. From the fact that sequences in \mathcal{C}' satisfy the greedy property, the shaded regions increase. An algorithm which samples one $(c+1)$ -index at many intervals succeeds in building a long monotone subsequence. If an algorithm queries $O_{k,\varepsilon}(1)$ random positions at all $O(\log n)$ intervals, we expect to have many intervals where some $(c+1)$ -index of a k -tuple in \mathcal{C}' sampled, thus finding a long increasing subsequence.

probability. By performing this procedure for $\Omega_{k,\varepsilon}(1)$ uniformly randomly chosen values of ℓ , a length- k increasing subsequence is found, with high probability, if the first case in the structural theorem holds; this procedure has query complexity $O_{k,\varepsilon}(\log n)$. (See Figure 6 for an illustration of what an algorithm hopes to achieve, and why $O_{k,\varepsilon}(\log n)$ queries suffice.)

Now suppose that, instead, the “splittable intervals” case holds. By making $\Theta_{k,\varepsilon}(1)$ random samples, we can find, with high probability, a value ℓ that lies in M_i for some $i \in [s]$; let w be its width (defined, as above, to be $\lfloor \log |I_i| \rfloor$). By the assumption on ℓ , the interval $[\ell - 2^w, \ell]$ contains many disjoint length- c increasing subsequences, the interval $[\ell, \ell + 2^w]$ contains many disjoint length- $(k - c)$ increasing subsequences, and two such subsequences can be combined to form the desired length- k increasing subsequences. Thus we can “guess” the width w (with a cost of a factor of $\log n$ in the query complexity), and repeat the same reasoning in the intervals $[\ell - 2^w, \ell]$ and $[\ell, \ell + 2^w]$. We remark that such reasoning would yield an algorithm with a query complexity of $\Theta_{k,\varepsilon}((\log n)^{k-1})$ in the worst case. In order to obtain the optimal exponent of $\lfloor \log_2 k \rfloor$ one can aim to find, instead of an element of M_i , a particular element in one of the increasing subsequences in either L_i or R_i . To facilitate this idea, [BECLW19] goes on to prove a more detailed structural theorem, obtained (with some effort) by iterating the structural result mentioned above.