Testing for Forbidden Order Patterns in an Array[*]

Ilan Newman[†] Yuri Rabinovich[‡] Deepak Rajendraprasad[§] Christian Sohler[¶]

October 8, 2018

Abstract

A sequence $f : [n] \to \mathbb{R}$ contains a pattern $\pi \in \mathcal{S}_k$, i.e., a permutations of $[k]$, iff there are indices $i_1 < \cdots < i_k$, such that $f(i_x) > f(i_y)$ whenever $\pi(x) > \pi(y)$. Otherwise, f is π-free. We study the property testing problem of distinguishing, for a fixed π, between π-free sequences and the sequences which differ from any π-free sequence in more than ϵn places. Our main findings are as follows:

- For monotone patterns, i.e., $\pi = (k, k-1, \ldots, 1)$ and $\pi = (1, 2, \ldots, k)$, there exists a *non-adaptive* one-sided error ϵ-test of $(\epsilon^{-1} \log n)^{O(k^2)}$ query complexity. For any other π, any non-adaptive one-sided error test requires $\Omega(\sqrt{n})$ queries.

  The latter lower-bound is tight for $\pi = (1, 3, 2)$. For specific $\pi \in \mathcal{S}_k$ it can be strengthened to $\Omega(n^{1-2/(k+1)})$. The general case upper-bound is $O(\epsilon^{-1/k} n^{1-1/k})$.
- For *adaptive* testing the situation is quite different. In particular, for any $\pi \in \mathcal{S}_3$ there exists an adaptive ϵ-tester of $(\epsilon^{-1} \log\ n)^{O(1)}$ query complexity.

---

[*] A preliminary version of this paper has appeared in the Proceedings of SODA 2017 [39].

[†] Department of Computer Science, University Haifa, Haifa, Israel. E-mail: ilan@cs.haifa.ac.il. This research was supported by The Israel Science Foundation, number 497/17.

[‡] Department of Computer Science, University of Haifa, Haifa, Israel. E-mail: ilan@cs.haifa.ac.il.

[§] Indian Institute of Technology Palakkad, Kerala, India. Email: deepak@iitpkd.ac.in. This work was done while visiting the Caesarea Rothschild Institute, University of Haifa

[¶] The author acknowledges the support of ERC grant 307696.

1 Introduction

Property testing is a framework for studying sampling algorithms for approximately deciding if a large object has a given property or is far away from having that property. In this paper we consider property testing questions for forbidden patterns of sequences. In the early days of property testing, studying monotonicity of a sequence, i.e. the property of being sorted, was a major theme. The first testing algorithm for monotonicity was developed by Ergün et al. [21] and a matching lower bound was given by Fischer [22]. Later Bhattacharyya et al. [9] developed a very simple and elegant tester for this property.

In this paper, we study generalizations of monotonicity of a sequence. The properties we are considering are defined by forbidden patterns. A forbidden pattern of size k is defined by a permutation $\pi \in \mathfrak{S}_k$ of $\{1, \ldots, k\}$ in the following way: a sequence $f : \{1, \ldots, n\} \to \mathbb{R}$ contains a pattern π, iff there is a sequence of k indices $i_1 < i_2 < \cdots < i_k$ such that $f(i_x) < f(i_y)$ whenever $\pi(x) < \pi(y)$. A sequence is π-free, if it does not contain π. For example, a $(2, 1)$-free sequence is sorted non-decreasingly, i.e. monotone. A $(k, k-1, \ldots, 1)$-free sequence is a sequence that can be partitioned into at most $k-1$ monotone (non-decreasing) subsequences.

One motivation to study pattern free sequences comes from combinatorics. The notion of being π-free has been extensively studied in the area of combinatorics of permutations (see, for example the books [13] and [35]). One of the major open questions in the area was the famous Stanley-Wilf conjecture from the 80's about the growth rate of the number of π-free permutations. I.e., if $s_n(\pi)$ denotes the number of π-free permutations, then $\lim_{n\to\infty} s_n(\pi)^{1/n}$ exists, and is finite. This was proven by Marcus and Tardos [38] in 2004. Later, Fox [25] proved that most Stanley-Wilf limits are exponential in contrast to previous belief that all are polynomial in the pattern length.

Forbidden patterns in permutations have many applications in combinatorics. To bring one example, the permutations that can be obtained from the identity permutation using a Gilbreath shuffle are characterized by the forbidden patterns $(1, 3, 2)$ and $(3, 1, 2)$. A Gilbreath shuffle is a two step shuffling procedure for a deck of cards, where the deck is first cut into two piles putting the second one in a reverse order, and then riffling the piles together. A database of applications of permutations with forbidden patterns is available online [42].

Another motivation comes from the study of patterns and motifs in time series analysis [8, 33, 40], for example, series of measurements from sensors, stock market data or data of an electrocardiogram. One difficulty in this area is that time series contain noise and may be sampled at different and varying frequencies. This implies that patterns have to be approximate. While the notion of pattern used in this paper is certainly less local than what is typically used in data analysis, we still believe that ideas from our paper could potentially be interesting for this area, for example, in the context of developing sampling algorithms for motif discovery.

Although testing for a constant size pattern can be done in linear time using the sophisticated algorithm of Guillemot and Marx [29], this may be too slow if one would like to test several long sequences (and potentially also their subsequences) for several patterns. Therefore, we are considering a sampling approach. Given a fixed pattern $\pi \in \mathfrak{S}_k$, a sampling algorithm that accepts with probability at least $2/3$ all π-free sequences and rejects with probability $2/3$ all sequences that differ in more than ϵn values from every π-free sequence is called *property tester*. If the algorithm always accepts when the input is π-free, we say the property tester has *one-sided error*. In this paper only the latter type of testers will be considered.

1.1 Summary of results

Let $\pi \in \mathfrak{S}_k$ be a fixed pattern. We establish the following results about one-sided-error testers for being π-free:

1. The monotone patterns i.e., $\pi = (1, \ldots, k)$ or $\pi = (k, \ldots, 1)$, can be *non-adaptively* ϵ-tested for, making $\left(\epsilon^{-1} \log n\right)^{O(k^2)}$ queries.
2. For every non-monotone pattern $\pi \in \mathfrak{S}_k$, any *non-adaptive* $\frac{1}{9k}$-tester for π must make $\Omega(\sqrt{n})$ queries. Moreover, for every odd $k \geq 3$, there exists a pattern $\pi \in \mathfrak{S}_k$ such that any *non-adaptive* $\frac{1}{3k}$-tester for π must make $\Omega\left(n^{1-2/(k+1)}\right)$ queries. The above was recently improved [5] to $\Omega\left(n^{1-1/(k-1)}\right)$ queries.
3. Every pattern $\pi \in \mathfrak{S}_k$ can be *non-adaptively* ϵ-tested making $O\left(\epsilon^{-1/k} \cdot n^{1-1/k}\right)$ queries. Moreover, for $k = 3$, this can be improved to $\widetilde{O}(\sqrt{n}/\epsilon)$ queries. The above was recently improved [5] to $O\left(n^{1-1/(k-1)}\right)$ queries.
4. Any non-monotone $\pi \in \mathfrak{S}_3$, can be *adaptively* ϵ-tested making $\left(\epsilon^{-1} \log n\right)^{O(1)}$ queries. The adaptive tester runs a binary search in a nearly sorted array as a subroutine, and its analysis may be of independent interest.

To sum up, we show that the complexity of the problem of testing for π-freeness crucially depends on the structure of π, a phenomenon similar to the one occurring in the Stanley-Wilf circle of problems.

The testers above can be extended to testing avoidance of finite collections of permutations, e.g., testing for $\{(1,3,2),(3,1,2)\}$-freeness, which is testing whether the given sequence can be obtained by a Gilbreath shuffle of a monotone sequence. But the lower bounds do not extend. In fact, we note that being $\{(1,3,2),(3,1,2)\}$-free can be tested *non-adaptively* with poly-log queries (Section 4.3).

Item (4) provides a new example of a natural property having an exponential gap between *adaptive* and *non-adaptive* query complexity of *order based* testers. Earlier examples of such problems were designed in [22].[1]

A last remark is that our results apply to arbitrary sequences rather than permutations.

1.2 Our techniques

It is easy to prove that a sequence that is ϵ-far from being π-free must contain many disjoint copies of π. It follows, via a standard probabilistic argument, that we can test π-freeness using $O(\epsilon^{1/k} n^{1-1/k})$ queries. We can improve this running time in some interesting cases by exploiting the structure of π.

The case of monotone patterns is special because they allow for a relatively simple recursive attack. Assume that $f : [n] \to \mathbb{R}$ is ϵ-far from $(1, \ldots, k)$-free. We start by guessing (in a particular non-uniform way) two disjoint but adjacent intervals I_L and I_R of $[n]$ such that there is a collection T of relatively many forbidden $(1, \ldots, k)$-tuples with their first l points in I_L and the last $k - l$ points in I_R, for some fixed $l \in [k-1]$. The concatenation of a $(1, \ldots, l)$-tuple from I_L and a $(1, \ldots, k-l)$-tuple from I_R will be a $(1, \ldots, k)$-tuple whenever the last value of the first piece is smaller than the first value of the second piece. Hence we can employ a "median-split and concatenate" argument to reduce the problem of $(1, \ldots, k)$-testing to testing for two monotone patterns of smaller length. This yields a poly-logarithmic tester for monotone patterns.

This approach, unfortunately, works only for monotone patterns. For example, let us consider the pattern $(1,3,2)$. Even if we find two adjacent intervals I_L and I_R as before with relatively many $(1,3,2)$-tuples with their first two coordinates in I_L and the third coordinate in I_R, we do not have a median-split argument to take us forward. It may as well turn out that, for every $(1,2)$-pair in I_L there may be at most one coordinate in I_R that can complete it to a $(1,3,2)$-tuple. Hence finding that using random sampling is very unlikely using $o(\sqrt{n})$ queries. We exploit this possibility to establish a lower bound of $\Omega(\sqrt{n})$ for the running time of non-adaptive $(1,3,2)$-testers. Moreover, we combine our $(1,2)$-tester and a uniform sampler to design a non-adaptive $(1,3,2)$-tester with $\widetilde{O}(\sqrt{n})$ running time. On the other hand, this extreme situation described

[1] A tester for sequence properties is called *order based* if it makes decisions based only on the relative order of the queried values and not on their actual values [22]. All our testers are order based.

above forces a lot of structure on the sequence in I_L. If we allow ourselves the power of adaptivity, we can use this structure using a slightly modified randomized binary search to find that unique coordinate in I_R which can complete the chosen $(1, 2)$-pair in I_L to a $(1, 3, 2)$-tuple. This leads to a poly-logarithmic adaptive tester for the $(1, 3, 2)$-tuple. The version of randomized binary search that we employ and analyze may be of an independent interest.

1.3 Generalizations of the problem

The definition of π-freeness can easily be extended to partially ordered domains. Given a function $f : D \to \mathbb{R}$ whose domain is partially ordered by $\preceq$ contains a pattern $\pi \in \mathfrak{S}_k$, if there are $i_1 \preceq i_2 \preceq \cdots \preceq i_k$ such that $f(i_x) > f(i_y)$ whenever $\pi(x) > \pi(y)$. Now the pattern $(2, 1)$ corresponds to monotonicity testing over partially ordered domains, one of the most widely studied problem in property testing.

We remark that the result of testing π-freeness for a pattern of constant length k in time $O(\epsilon^{1/k}|D|^{1-1/k})$ extends to this more general setting implying that we can always test the problem in sub-linear query complexity. The question of classifying the patterns that can be efficiently tested in this more general setting is an interesting open problem.

Delete distance vs. Hamming distance. The *delete distance* between two functions f and g in $\mathbb{R}^{[n]}$ is n minus the length of a longest common subsequence of f and g. Since one can delete the entries where f and g differ from both of them to get a common subsequence of length $n - d(f, g)$, the delete distance is at most the Hamming distance. But on the other hand, the Hamming distance can be much larger than the delete distance. For example, $f : i \mapsto i$ and $g : i \mapsto i + 1$ has delete distance 1 and Hamming distance n between them. Nevertheless, it can be seen that for any pattern $\pi \in \mathfrak{S}_k$, for any $k \leq n$, the Hamming distance of f to the set F_π of π-free functions is at most the delete distance of f from F_π. Hence, our results continue to hold when the metric used to define the notion of being ϵ-far from a class of functions is the delete distance.

1.4 Other related work

Property testing was introduced by Rubinfeld and Sudan [41]. The study of combinatorial properties was initiated by Goldreich, Goldwasser and Ron [27]. Works on string properties related to forbidden or occurring patterns in labeled posets include the papers on testing sortedness [21, 9] and the lower bound of Fischer [22], and many others, see e.g., [7, 37, 3, 24], and citations therein.

The problem of testing hereditary properties of permutations has been studied before by Hoppen et al. [31] under the rectangular distance (discrepancy of intervals) and by Klimošová and Král [36] under Kendall's tau-distance (the normalized number of transpositions). Unlike the edit distance used in this paper, in both the distance measures discussed above, local changes do not contribute much to the distance: For example, the sequence $(2, 1, 4, 3, \ldots, n, n - 1)$ is close to being monotone with respect to these two distances, while according to the edit distance it is not. Therefore, the results are not comparable.

Another line of research relevant to our problem (mostly to the extension to partial orders discussed in the previous subsection) is monotonicity testing [28, 11, 20, 30, 23, 14, 2, 21, 9, 18, 34, 12, 4, 19]. Recently, Chakrabarty and Seshadhri gave an optimal tester for this problem on the hypercube and on hypergrids [15, 16]. In another paper they prove that the important special case of monotonicity of Boolean functions over the d-dimensional hypercube can be tested in $o(d)$ query complexity [17]. This was improved by Khot, Minzer and Safra to an $\tilde{O}(\sqrt{d})$ tester, which is nearly optimal for the problem [34]. More recently, Black et al. [10] proved that one can test monotonicity of Boolean functions over $[n]^d$ with $o(d) \log^{O(1)} n$ queries.

Finally, we note that some progress on the exact decision problem for π-freeness for permutations was obtained in [1], and a more efficient $O(f(k) \cdot n)$ time algorithm was presented in [29] where f is a doubly exponential function in the length of the pattern k. Importantly, due to the existence of this algorithm, the running times of all the testers developed in this paper are linear in the number of queries they make.

1.5 Organization of the paper

We describe and analyze the poly-logarithmic non-adaptive tester for monotone patterns in Section 3. The poly-log adaptive tester and the optimal non-adaptive tester for the pattern $(1,3,2)$ are in the Section 4. In Section 5, we prove the polynomial lower bound for non-adaptive testing of the pattern $(1,3,2)$ and then extend the result to all non-monotone patterns. We discuss how this technique can be used for deriving better lower bounds for some specific patterns in Appendix Section A. We conclude with a few remarks and some open problems in Section 6.

2 Notation and preliminaries

We denote the set $\{1,\ldots,n\}$ by $[n]$ and the symmetric group of permutations of $[n]$ by $\mathfrak{S}_n$. We represent a permutation $\pi \in \mathfrak{S}_n$ in the one-line notation $(\pi(1),\ldots,\pi(n))$. Our domain is the set of functions $f:[n]\to\mathbb{R}$ equipped with the (Hamming) distance: $d(f,g) = |\{i : f(i) \neq g(i)\}|$. We may speak about a function $f:[n]\to\mathbb{R}$, equivalently, as an array f of real numbers.

A function (array) $f:[n]\to\mathbb{R}$ is said to *contain* a pattern $\pi \in \mathfrak{S}_k$ for some $k \leq n$, if there exists a subsequence of f that is order-isomorphic to π; namely, a sequence of indices $i = (i_1,\ldots,i_k) \in [n]^k$, $i_1 < \cdots < i_k$, such that, for every pair $a,b \in [k]$, $f(i_a) < f(i_b)$ whenever $\pi(a) < \pi(b)$. The function f is called π*-free* if it does not contain π. Otherwise, every k-tuple $i = (i_1,\ldots,i_k) \in [n]^k$, $i_1 < \cdots < i_k$, such that the subsequence in $f(i_1)\ldots,f(i_k)$ is order-isomorphic to π is called a π*-tuple* in f and we write $f|_i \sim \pi$. For example, f is nondecreasing if and only if it is $(2,1)$-free.

We say that a function $f:[n]\to\mathbb{R}$ is ϵ*-far from being* π*-free*, for a pattern $\pi \in \mathfrak{S}_k$ if $d(f,g) \geq \epsilon n$ for every π-free function $g:[n]\to\mathbb{R}$. We say that a pattern $\pi \in \mathfrak{S}_k$ is ϵ*-testable* with one-sided error using $q = q(\epsilon,n)$ queries, if there exists a randomized algorithm $\mathcal{A}$ which makes at most q queries to any function $f:[n]\to\mathbb{R}$ and accepts it with probability 1 if it is π-free, and rejects it with probability at least 0.5 if it is ϵ-far from being π-free. Here, a query to f is made by specifying an index $i \in [n]$ on which the answer is $f(i)$. We say that $\mathcal{A}$ is *non-adaptive* if it chooses all the q query locations before it makes the first query to f.

The definitions above are made for one fixed permutation $\pi \in \mathfrak{S}_k$, as most results are for this case. However, the definitions can be extended to collection of forbidden permutations A of maximum length k. Namely, f is A-free if it does not contain any $\pi \in A$. The distance to being A-free is defined appropriately.

Most of our analysis start by picking a collection of disjoint k-tuples. The next basic proposition will be used recurrently.

Definition 2.1 *Let T be a set of k-tuples of $[n]$. We denote by $T(i)$, for each $i \in [k]$, the set of the i-th coordinates of the k-tuples in T: e.g., $T(1) = \{a_1 | \ (a_1,\ldots,a_k) \in T\}$. We call the set of k-tuples $T^* = T(1)\times\cdots\times T(k)$ the* closure *of T. We say that T is a* matching *if every pair of k-tuples in T are disjoint as sets.*

Proposition 2.2 *Let $f:[n]\to\mathbb{R}$ be ϵ-far from being π-free for some pattern $\pi \in \mathfrak{S}_k$. Then there is a matching T of π-tuples in f with $|T| \geq \epsilon n/k$.*

Proof. Let T be a maximal matching of π-tuples in f and let $I = T(1)\cup\cdots\cup T(k)$. By definition, f restricted to the set $[n]\setminus I$ is π-free. We can make function f π-free for the whole domain $[n]$ by replacing for each $i \in I$ the value $f(i)$ by $f(j)$, where j is the largest integer $j \notin I$ with $j < i$. Since f is ϵ-far from being π-free, $|T(1) \cup \cdots \cup T(k)| \geq \epsilon n$. Therefore $|T| \geq \epsilon n/k$. ■

Next we discuss a simple result that shows that testing for a constant size set of forbidden patterns is essentially as difficult as testing for each forbidden pattern individually.

For constant-size A, it is obvious that if one can test π-freeness for every $\pi \in A$ in at most q queries, then being A-free can be tested in $O(q)$ queries. The other way around is not necessarily true, at least for non-adaptive testing as we show by a counter example in Section 6.

Proposition 2.3 *Let A be a set of forbidden permutations of maximum length k. If a function $f : [n] \to \mathbb{R}$ is ϵ-far from A-free then f is $\frac{\epsilon}{|A|}$-far from π-free for at least one $\pi \in A$. In particular, if one can test π-freeness with 1-sided error for every $\pi \in A$ in at most $q(n, k, \epsilon)$ queries, then being A-free can be tested in $|A| \cdot q(n, k, \frac{\epsilon}{|A|})$ queries. If one can test π-freeness with 2-sided error for every $\pi \in A$ in at most $q(n, k, \epsilon)$ queries, then being A-free can be tested in $O(|A| \log |A| \cdot q(n, k, \frac{\epsilon}{|A|}))$ queries.*

Proof. For sake of contradiction assume that f is ϵ-far from A-free but $\epsilon/|A|$-close to being π-free all patterns π in A. Then for each $\pi \in A$ we can delete fewer than $\epsilon n/|A|$ elements from f (viewed as a sequence) to obtain a π-free sequence. Thus, overall we can delete fewer than ϵn elements from f to obtain an A-free sequence. Hence f is ϵ-close to A-free, which is a contradiction. This implies the result for one-sided error using $|A| \cdot q(n, k, \frac{\epsilon}{|A|})$ queries by applying each tester individually. If the tester has 2-sided error, we first need to amplify the tester so that it errs with probability at most $1/(3|A|)$ (which can be done with an $O(\log |A|)$ factor overhead by repeating each of the tests and taking the majority of the outcomes. The tester for A-freeness rejects, if one of the individual testers rejects. The result then follows from the union bound. ■

Consider a function $f : [n] \to \mathbb{R}$ which contains a matching T of ϵn π-tuples for some $\pi \in \mathfrak{S}_k$. A standard second moment argument implies that if one samples $O(\epsilon^{-1/k} n^{1-1/k})$ indices from $[n]$ uniformly at random, then with high probability there is a π-tuple in f among the sampled indices. Hence the next result.

Theorem 2.4 *Every pattern π can be tested in $O\left(\epsilon^{-1/k} n^{1-1/k}\right)$ queries using a non-adaptive one-sided-error algorithm, where k is the length of π.*

Proof. It suffices to show that if a function $f : [n] \to \mathbb{R}$ is ϵ-far from being π-free, then a random uniform subset $Q \subset [n]$ of size $q = (\epsilon/k)^{-1/k} n^{1-1/k}$ contains the pattern π with a constant positive probability. Then by standard amplification (two times would be enough) we would reach a probability $1/2$ of rejection.

By Proposition 2.2, $f : [n] \to \mathbb{R}$ which is ϵ-far from being π_k-free, contains $m = \epsilon n/k$ disjoint π-tuples. Let A_i be the event that that Q contains the i'th member of this set. Then, the probability that Q contains a pattern π is at least

$$P[\cup A_i] \;\geq\; \sum_{1}^{m} P[A_i] - \sum_{1 \leq i < j \leq m} P[A_i \cap A_j] \;=\; m \cdot \binom{n-k}{q-k} / \binom{n}{q} \;-\; \binom{m}{2} \cdot \binom{n-2k}{q-2k} / \binom{n}{q} \;=$$

$$=\; m \cdot \frac{\prod_{i=0}^{k-1}(q-i)}{\prod_{i=0}^{k-1}(n-i)} - \binom{m}{2} \cdot \frac{\prod_{i=0}^{2k-1}(q-i)}{\prod_{i=0}^{2k-1}(n-i)} = \frac{\epsilon n}{k}\left(\frac{q}{n}\right)^k \cdot (1-o(1)) - \frac{1}{2}\left[\frac{\epsilon n}{k}\left(\frac{q}{n}\right)^k\right]^2 (1-o(1)) \;=0.5-o(1)$$

■

3 Monotone patterns

For every $k \in \mathbb{N}$, we call the permutations $(1, \dots, k)$ and $(k, \dots, 1)$ *monotone patterns*. Since testing for the monotone increasing pattern $(1, \dots, k)$ is the same as testing for the monotone decreasing pattern $(k, \dots, 1)$ in the reversed sequence, we restrict our discussion to testing for $\pi_k = (k, \dots, 1)$. The goal of this section is to prove:

Theorem 3.1 *Every monotone pattern π_k can be tested in $(k\epsilon^{-1} \log n)^{O(k^2)}$ queries using a non-adaptive one-sided-error algorithm.*

The tester is conceptually simple. We show that sampling a k-tuple of points, under a suitable distribution will return a π_k-tuple in f with sufficient probability. We do not explicitly state the distribution according to which we sample the k-tuples. Instead, we explicitly describe the sampling procedure:

Algorithm 3.2 (DyadicSampler(I, k)) *The input to the algorithm consists of an interval I of m natural numbers and a natural number $k \le m$. The output is a k-tuple $t \in I^k$ generated as described below.*

1. *If $k = 1$, return $t \in I$ picked uniformly at random and terminate.*
2. *Otherwise, pick a "split-point" $\ell \in [k-1]$ uniformly at random. The k-tuple t returned by the algorithm will be a concatenation of an ℓ-tuple and a $(k-\ell)$-tuple sampled recursively from two adjacent and disjoint subintervals I_L and I_R of I to be selected next.*
3. *Fix a "slice-width" $W = 2^w$, where w is chosen uniformly at random from $\{0, 1, \dots, \lfloor \log m \rfloor - 1\}$. Slice I into consecutive disjoint intervals $I_1, \dots, I_{\lceil m/W \rceil}$, each of length W (except possibly the last one).*
4. *Pick a "slice-number" s uniformly at random from $\{1, \dots, \lceil m/W \rceil\}$. Define I_L to be the union of the 2ℓ consecutive slices up to s, and I_R to be the union of the $2(k-\ell)$ consecutive slices after s. That is, $I_L = I_{s-2\ell+1} \cup \dots \cup I_s$, and $I_R = I_{s+1} \cup \dots \cup I_{s+2(k-\ell)}$.*
   In the above expressions, assume $I_i = \emptyset$ if $i \notin \{1, \dots, \lceil m/W \rceil\}$
5. *Recursively sample $(t_1, \dots, t_\ell)$ from I_L and $(t_{\ell+1}, \dots, t_k)$ from I_R. That is,*

$$(t_1, \dots, t_\ell) = \textsc{DyadicSampler}(I_L, \ell),$$
$$(t_{\ell+1}, \dots, t_k) = \textsc{DyadicSampler}(I_R, k-\ell).$$

6. *Return the concatenated tuple $t = (t_1, \dots, t_k)$ and terminate.*

Theorem 3.1 follows immediately from the following stronger theorem. We need the following definitions for its proof:

Definition 3.3 *Let $t = (t_1, \dots, t_k)$ be a k-tuple of positive integers with $t_1 < \dots < t_k$. We define the* leap-start *of t to be the smallest $i \in [k-1]$ such that $t_{i+1} - t_i \ge t_{j+1} - t_j, \forall j \in [k-1]$ and the* leap-size *of t to be $\lfloor \log(t_{i+1} - t_i) \rfloor$, where $i = \text{leap-start}(t)$.*

Theorem 3.4 *Let t be the k-tuple generated by a call to Algorithm 3.2 with arguments $([n], k)$. For any function $f : [n] \to \mathbb{R}$ which contains a matching T of π_k-tuples, the joint probability that t is a π_k-tuple in f and $t \in T^*$ is at least $(|T|/n)^k (2k^2 \log n)^{-\binom{k+1}{2}+1}$.*

Proof. The proof is by an induction on k. The statement is easily verified for $k = 1$ (where the only nontrivial event is $t \in T^*$).

The event that the k-tuple $t = (t_1, \ldots, t_k)$ returned by Algorithm 3.2 is a π_k-tuple in f is denoted by $f|_t \sim \pi_k$. We want to estimate the probability of the joint event $[f|_t \sim \pi_k,\ t \in T^*]$.

Since Algorithm 3.2, at its top level, makes three independent and uniform random choices, namely split-point, slice-width, and slice-number, we can write the total probability of success as the expected value of the conditional probabilities $P[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}]$, where the expectation is over uniform choice of the split-point ℓ from $[k-1]$, uniform choice of the slice-width W from $\{2^w : 0 \le w \le \lfloor \log n \rfloor - 1\}$, and uniform choice of the slice-number s from $[\lceil n/W \rceil]$ and $E_{\ell,w,s}$ is the event that the three choices made by Algorithm 3.2 are ℓ as the split point, $W = 2^w$ as the slice-width and s as the slice number. That is,

$$P[f|_t \sim \pi_k,\ t \in T^*] = \mathbb{E}\left(P[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}]\right) \tag{1}$$

where

$$\mathbb{E}() = \sum_{\ell=1}^{k-1} \frac{1}{k-1} \sum_{w=0}^{\lfloor \log n \rfloor - 1} \frac{1}{\log n} \sum_{s=1}^{\lceil n/2^w \rceil} \frac{1}{\lceil n/2^w \rceil}()$$

Now we estimate $P[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}]$ for an arbitrary but fixed ℓ, w, s. Let $T_{\ell,w,s} \subset T$ be all the k-tuples t in T with $\text{leap-start}(t) = \ell$, $\min\{\text{leap-size}(t), \lfloor \log n \rfloor - 1\} = w$, and $t_\ell \in I_s$, where $I_s = [(s-1)W + 1, sW]$. Notice that every k-tuple $(t_1, \ldots, t_k) \in T_{\ell,w,s}$ has $t_1, \ldots, t_\ell \in I_L$ and $t_{\ell+1}, \ldots, t_k \in I_R$, where I_L and I_R are the intervals selected by Algorithm 3.2 once the event $E_{\ell,w,s}$ has occurred. Moreover, we include every k-tuple t with $\text{leap-size}(t) > \lfloor \log n \rfloor - 1$ in $T_{\ell,w,s}$, where $\ell = \text{leap-start}(t)$, $w = \lfloor \log n \rfloor - 1$ and s chosen so that for the corresponding I_L and I_R, we have $t_1, \ldots, t_\ell \in I_L$ and $t_{\ell+1}, \ldots, t_k \in I_R$. Hence

$$T = \bigcup_{\ell=1}^{k-1} \bigcup_{w=0}^{\lfloor \log n \rfloor - 1} \bigcup_{s=1}^{\lceil n/2^w \rceil} T_{\ell,w,s}. \tag{2}$$

The key combinatorial observation we make here is that if u and v are two k-tuples in $T_{\ell,w,s}$ such that $f(u_\ell) > f(v_{\ell+1})$, the k-tuple $(u_1, \ldots, u_\ell, v_{\ell+1}, \ldots, v_k)$ is a π_k-tuple in $T^*_{\ell,w,s}$ (caution: do not confuse $T^*_{\ell,w,s}$ with $T_{\ell,w,s}$ here). In particular, if we choose any $x \in \mathbb{R}$ and define $L_{\ell,w,s}(x) = \{(t_1, \ldots, t_\ell) : (t_1, \ldots, t_k) \in T_{\ell,w,s},\ t_\ell \ge x\}$ and $R_{\ell,w,s}(x) = \{(t_{\ell+1}, \ldots, t_k) : (t_1, \ldots, t_k) \in T_{\ell,w,s},\ t_\ell \le x\}$, we see that the concatenation of any π_ℓ-tuple in $L_{\ell,w,s}(x)^*$ and any $\pi_{k-\ell}$-tuple in $R_{\ell,w,s}(x)^*$ results in a π_k-tuple in $T^*_{\ell,w,s}$. Hence the following claim is true.

CLAIM 3.4.1. For every $x \in \mathbb{R}$, the probability $P[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}]$ is at least $p_1 \cdot p_2$, where

$$p_1 = P\left[f|_{(t_1,\ldots,t_\ell)} \sim \pi_\ell,\ (t_1, \ldots, t_\ell) \in L_{\ell,w,s}(x)^*\right],$$
$$p_2 = P\left[f|_{(t_{\ell+1},\ldots,t_k)} \sim \pi_{k-\ell},\ (t_{\ell+1}, \ldots, t_k) \in R_{\ell,w,s}(x)^*\right].$$

We will choose x to be the maximum value so that the corresponding set $L = L_{\ell,w,s}(x)$ has size at least $\frac{\ell}{k}|T_{\ell,w,s}|$. This also ensures that the corresponding $R = R_{\ell,w,s}(x)$ has size at least $\frac{k-\ell}{k}|T_{\ell,w,s}|$

By the induction hypothesis, we know that

$$p_1 \ge \left(\frac{|L|}{2\ell W}\right)^\ell \left(2\ell^2 \log(2\ell W)\right)^{-\binom{\ell+1}{2}+1}$$
$$\ge \left(\frac{|T_{\ell,w,s}|}{2kW}\right)^\ell \left(2k^2 \log n\right)^{-\binom{\ell+1}{2}+1},$$

and similarly

$$p_2 \geq \left(\frac{|T_{\ell,w,s}|}{2kW}\right)^{k-\ell} \left(2k^2 \log n\right)^{-\binom{k-\ell+1}{2}+1}.$$

Thus, $p_1 p_2 \geq \left(\frac{|T_{\ell,w,s}|}{2kW}\right)^{k} \left(2k^2 \log n\right)^{-\binom{k}{2}+1}$.

Substituting this lower bound for $P\left[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}\right]$ in Eqn. (1), and using the standard fact that for nonnegative x_i's

$$\sum_{i=1}^{m} \frac{1}{m} x_i^k \geq \left(\sum_{i=1}^{m} \frac{1}{m} x_i\right)^k$$

successively three times, we get,

$$P\left[f|_t \sim \pi_k,\ t \in T^*\right] \geq \frac{(2k^2 \log n)^{-\binom{k}{2}+1}}{(2nk^2 \log n)^k} |T|^k, \tag{3}$$

as claimed in the theorem. ■

As we noted before, non-adaptive one-sided-error monotonicity testers with query complexity $O(\epsilon^{-1} \log n)$ are known. This complexity is asymptotically optimal by [22]. Theorem 3.4 (for $k = 2$) is a monotonicity tester with query complexity $O(\epsilon^{-2} \log^2 n)$, which is not optimal. However, as far as we know, none of the asymptotically optimal testers have the additional useful property that the tester will return a π_2-tuple which belongs to T^*, the closure of an implicitly assumed collection T of π_2-tuples. This property is quite useful for many inductive arguments as demonstrated in the previous proof. It will be used again in the adaptive tester for the pattern $(1,3,2)$. The $(2,1)$-tester we designed has a further useful property which helps us in using it as a subroutine in the non-adaptive and adaptive $(1,3,2)$-testers. We defer the discussion of this property to Section 4 (Definition 4.1, Claim 4.2) where we describe those $(1,3,2)$ testers.

4 Adaptive and non-adaptive testers for $(1,3,2)$-freeness

Unlike for the monotone patterns, a non-adaptive one-sided-error tester for $(1,3,2)$-pattern needs to make $\Omega(\sqrt{n})$ queries to f. This will be shown in Section 5.1. Nevertheless we describe an adaptive one-sided-error tester for the $(1,3,2)$-pattern which makes only poly-log many queries (Section 4.1). This is the most technical part of the paper.

We also describe a non-adaptive one-sided-error tester with $\widetilde{O}(\epsilon^{-1}\sqrt{n})$ queries showing that the lower bound is nearly tight (Section 4.2). Before proceeding to the testers for the pattern $(1,3,2)$, we prove one additional property of the dyadic sampler (Algorithm 3.2), which will play a crucial role there.

Definition 4.1 *For a function $f : [n] \to \mathbb{R}$, the f-*interval *of an ordered pair $(i,j) \in [n]^2$ is $\big(\min\{f(i), f(j)\}, \max\{f(i), f(j)\}\big)$. We say that an ordered pair (i,j)* dominates *an ordered pair (i',j') in f (and denote it by $(i,j) \succ_f (i',j')$), if the f-interval of (i,j) contains the f-interval of (i',j'). In particular every pair dominates itself. Further, we say that (i,j) dominates a set T of pairs (and denote it by $(i,j) \succ_f T$), if it dominates* at least one *pair in T.*

Claim 4.2 *Let $t = (t_1, t_2)$ be the ordered pair generated by a call to Algorithm 3.2 with arguments $([n], 2)$. For any function $f : [n] \to \mathbb{R}$ which contains a matching T of $(2,1)$-tuples, the probability that t is a $(2,1)$-tuple in f, $t \in T^*$, and $t \succ_f T$ is at least $(|T|/n)^2 (8 \log n)^{-2}$.*

Proof. Note that the lower bound on probability stated above is equal to the one guaranteed by Theorem 3.4 for $k = 2$. So this claim is stronger only because of the demand that $t \succ_f T$. We reexamine the proof of Theorem 3.4, with $k = 2$, to show that this additional requirement is obtained for free.

Claim 3.4.1 in the previous proof bounds the probability $P\left[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}\right]$ from below by the product of the probabilities p_1 and p_2. When $k = 2$ (and hence $\ell = 1$) they reduce to $p_1 = P[t_1 \in L_{\ell,w,s}(x)]$ and $p_2 = P[t_2 \in R_{\ell,w,s}(x)]$. Recall that $L_{\ell,w,s}(x) = \{t_1 : (t_1, t_2) \in T_{\ell,w,s},\ f(t_1) \geq x\}$ and $R_{\ell,w,s}(x) = \{t_2 : (t_1, t_2) \in T_{\ell,w,s},\ f(t_1) \leq x\}$.

For a $t_2 \in R_{\ell,w,s}$, let t'_2 denote the partner of t_2 in T (i.e., $(t'_2, t_2) \in T$), which is unique since T is a matching. Then, for every $t_1 \in L_{\ell,w,s}$, we have $f(t_1) \geq x \geq f(t'_2)$ (by definition of the set $R_{\ell,w,s}$) and thus $(t_1, t_2) \succ_f (t'_2, t_2) \in T$. That is, any pair from $L_{\ell,w,s}(x) \times R_{\ell,w,s}(x)$ dominates T and hence $p_1 p_2$ is a valid lower bound on $P\left[f|_t \sim \pi_k,\ t \in T^*,\ t \succ_f T \mid E_{\ell,w,s}\right]$. ■

4.1 An adaptive $(1, 3, 2)$-tester

The goal of this section is to prove

Theorem 4.3 *The pattern $(1, 3, 2)$ can be tested with one-sided-error in $(\epsilon^{-1} \log n)^{O(1)}$ queries using an adaptive algorithm.*

The ϵ-test that we are going to describe will make queries to f and reject f if and only if it finds a $(1, 3, 2)$-tuple among the queried points. The one-sidedness is obvious. We will only need to show that the test rejects an ϵ-far input with high enough probability.

Consider a function $f : [n] \to \mathbb{R}$ that is ϵ-far from being $(1, 3, 2)$-free. Then by Proposition 2.2, there is a matching T of $(1, 3, 2)$-tuples of size $|T| \geq \epsilon n/3$. The set T is partitioned into two types of tuples based on their leap-start (Definition 3.3): $T_1 = \{(i, j, k) \in T \mid j - i \geq k - j\}$ and $T_2 = T \setminus T_2$. That is, all the tuples in T_1 have leap-start 1 while those in T_2 have leap-start 2.

The two cases are of a different nature. We will present two tests; the first has high probability of success when $|T_1|$ is large, while the second has high probability of success when $|T_2|$ is large. The full test will run both these tests. Notice that at least one of T_1 or T_2 has size $\epsilon n/6$ or more.

Test 1: Test for the case $|T_1| \geq \epsilon n/6$:

The tester for this case is again DYADICSAMPLER$([n], 3)$, and the analysis is also similar to the case of $(3, 2, 1)$-testing. Intuitively, the structural reason for the success of the dyadic sampler when T_1 is large is the following. When the sampler chooses 1 as the split point at the top level, and recursively samples an index $t_1 \in I_L$ and a pair $(t_2, t_3) \in I_R^2$, their concatenation (t_1, t_2, t_3) is a $(1, 3, 2)$-tuple if (t_2, t_3) is a $(2, 1)$-pair with $f(t_3) > f(t_1)$.

Lemma 4.4 *Let t be the 3-tuple sampled by a call to Algorithm 3.2 with arguments $([n], 3)$. For any function $f : [n] \to \mathbb{R}$ which contains a matching T of $(1, 3, 2)$-tuples, all of which have leap-start 1, the probability that t is a $(1, 3, 2)$-tuple in f is at least $(|T|/n)^3(18 \log n)^{-5}$.*

Proof. In fact, we will prove that under the same hypothesis, the joint probability that t is a $(1, 3, 2)$-tuple in f and $t \in T^*$ is at least $(|T|/n)^3(18 \log n)^{-5}$. Notice that this bound is the same as the one in Theorem 3.4 with $k = 3$. The proof is similar to that of Theorem 3.4 with the pattern $\pi = (1, 3, 2)$ taking the role of $\pi_3 = (3, 2, 1)$ there, once we make the following two observations:

The first observation is that, in Eqn. (2), for every w and s, $T_{2,w,s}$ is empty by definition since all tuples in T have leap-start 1. So we can ignore the case $\ell = 2$ in the analysis.

The second observation is that, when $\ell = 1$, we can make a claim similar to Claim 3.4.1 for the probability $P\left[f|_t \sim (1,3,2),\ t \in T^* \mid E_{\ell,w,s}\right]$. Redefine

$$L_{1,w,s}(x) = \{t_1 : (t_1,t_2,t_3) \in T_{1,w,s}, t_1 \le x\}, \text{ and}$$
$$R_{1,w,s}(x) = \{(t_2,t_3) : (t_1,t_2,t_3) \in T_{1,w,s}, t_1 \ge x\}.$$

Then the concatenation of any $t_1 \in L_{1,w,s}(x)$ and any $(2,1)$-tuple (t_2,t_3) in $R_{1,w,s}(x)^*$ is a $(1,3,2)$-tuple in $T^*_{1,w,s}$. Therefore, the probability $P\left[f|_t \sim (1,3,2),\ t \in T^* \mid E_{1,w,s}\right]$ is bounded below by $p_1 p_2$ where p_1 and p_2 are as defined in the proof there. ■

Test 1 repeats the dyadic sampler $O(\epsilon^{-3}\log^5 n)$ times. By Lemma 4.4, we see that it finds a $(1,3,2)$-tuple in f, and therefore rejects f with probability close to 1 when $|T_1| \ge \epsilon n/6$.

Test 2. Test for the case $|T_2| \ge \frac{\epsilon n}{6}$.

This case is different from the previous one since we cannot make a claim similar to Claim 3.4.1 for the probability $P\left[f|_t \sim (1,3,2),\ t \in T^* \mid E_{\ell,w,s}\right]$ when $\ell = 2$. The concatenation of a $(1,2)$-pair (t_1,t_2) from the left interval I_L and an index t_3 from the right interval I_R is a $(1,3,2)$-tuple in f only if $f(t_3) \in (f(t_1), f(t_2))$. Hence we cannot do the earlier median-split and concatenate argument. In fact, Theorem 5.1 shows that this limitation is grave enough to rule out poly-log non-adaptive testers for the pattern $(1,3,2)$.

To explain the intuition behind the adaptive tester (Algorithm 4.12 below) without getting into the quantitative details, let us assume that for two disjoint intervals I_L and I_R in $[n]$ (with I_L to the left of I_R), there is a large matching T of $(1,3,2)$-tuples with $T(1) \cup T(2) \subset I_L$ and $T(3) \subset I_R$. Let $i_0 \in I_L$ be an index such that $f(i_0)$ is smaller than the median f-value in $T(1)$. Let $I'_R = \{i \in I_R : f(i) > f(i_0)\}$.

We sample a pair (j,k) from I_R using the dyadic sampler. If $f|_{I'_R}$ has many $(2,1)$-tuples, then with good probability, (j,k) is a $(2,1)$-tuple from I'_R. In this case, (i_0,j,k) is a $(1,3,2)$-tuple and we are done.[2]

On the other hand, if the dyadic sampler on I_R does not succeed after sufficiently many trials, one can infer that $f|_{I'_R}$ is very close to monotone. Next, we run the dyadic sampler on I_L to get a pair (i,j). With good probability, (i,j) is a $(1,2)$-pair with $f(i) \ge f(i_0)$ and which dominates a $(1,2)$-pair in $\{(t_1,t_2) : (t_1,t_2,t_3) \in T\}$ (Claim 4.2). Finally, we search for t_3 in I_R using a version of random binary search that performs well in a nearly sorted array. If the search succeeds in finding an index $k \in I_R$ such that $f(k) \in \big(f(i), f(j)\big)$, we return the $(1,3,2)$-tuple (i,j,k).

Remarks 4.5 *We use the domination property of the dyadic sampler (Claim 4.2) for $(1,2)$-tuples rather than $(2,1)$-tuples. A proof for it follows by symmetry. We chose to write the proof for $(2,1)$-tuples to have notational consistency with the proof of Theorem 3.4.*

Before we formally state and analyse Test 2 (Algorithm 4.12), we discuss the version of random binary search that is used as a subroutine in Algorithm 4.12.

Problem 4.6 (SEARCH IN NEARLY MONOTONE EMBEDDED SEQUENCES) *The input for the problem consists of a function $f : I \to \mathbb{R}$, where I is an interval of m natural numbers; a "filter-range" F which is an open interval in $\mathbb{R}$; a "query-range" $Q \subset F$ which is also an open interval in $\mathbb{R}$; and a positive real number ϵ. The interval I and the ranges F and Q are explicitly given, while f is available via query access. Furthermore, the following three promises are also given: (i) the preimage $A = f^{-1}(F)$ has size at least ϵm. (ii) $f|_A$ contains a monotone increasing sequence of length at least $(1 - \epsilon/\log^5 m)|A|$, and (iii) there exists an index i in the above monotone sequence such that $f(i) \in Q$. The goal is to find (w.h.p.) an index $i \in I$ such that $f(i) \in Q$ using at most $\mathrm{poly}(\epsilon^{-1}\log m)$ queries to f.*

[2]In fact, this part of the tester actually subsumes the tester for Case 1 (large T_1) and hence we can choose not to run the tester for Case 1 separately. For the sake of clarity, we do not analyze the performance of Algorithm 4.12 for Case 1.

Algorithm 4.7 *(*FILTEREDBINARYSEARCH(f, F, Q, ϵ)*) The input to the algorithm consists of a function $f : I \to \mathbb{R}$, where I is an interval of m natural numbers; a "filter-range" F which is an open interval in $\mathbb{R}$; a "query-range" $Q \subset F$ which is also an open interval in $\mathbb{R}$; and a positive real number ϵ. The output is either an index $i \in I$ such that $f(i) \in Q$ or FAIL.*

The algorithm repeats the following two steps as long as $I \neq \emptyset$ and the total number of queries made to f is less than $1000\epsilon^{-2} \log^4 m$. If either of the above happens, then the algorithm returns FAIL.

1. *Pick $i \in I$ independently and uniformly at random till $f(i) \in F$.*
2. *If $f(i) \in Q$, then return i and terminate. Otherwise, continue after narrowing the search interval I to either the left or right of i based on whether $f(i) \geq \sup(Q)$ or $f(i) \leq \inf(Q)$, respectively.*

Notice that Algorithm 4.7 is a standard random binary search in which a basic random query is replaced with independent random queries until one gets a value in a specified filter range; the filter range being specified along with the input. We analyze the performance of this algorithm for Problem 4.6 and show that there exists a very large subset A' of indices in $f^{-1}(F)$ for which this strategy works (Theorem 4.11).

Assume that $f : [n] \to \mathbb{R}$ is a function that is represented by the sequence of f-values in an array of size n; $f(1), \ldots, f(n)$. The goal is to search for a value x in the array. That is, to find i such that $f(i) = x$. When f is monotone, a deterministic binary search is the optimal search algorithm making $1 + \log n$ queries in the worst case. Many variants of binary search were considered in the literature, mainly to accommodate noisy answers of different types, see [32, 6]. We need a different variant that is closely related to the above, but as far as we know, not simply reducible to any of the previous results.

In our case, f is not necessarily monotone or even close to be so, however, there will be a filter range $F \subseteq \mathbb{R}$, so that $I(F) = f^{-1}(F)$ is relative large and $f|_{f^{-1}(F)}$ is very close to monotone. Moreover, F is available to the algorithm by an explicit decision oracle that for a given a, it will answer whether $a \in F$.

Our intention is to do a randomized binary search for i. If f would be monotone on $f|_{f^{-1}(F)}$, a simulation of the deterministic binary search would still find a required $i \in Q$, if one exists, in $O(\log m)$ queries. The only difficulty is to sample the next query so as to be in $f^{-1}(F)$ and to split $f^{-1}(F)$ into two large enough subsets. Since $f^{-1}(F)$ has large enough density in $[m]$, the first event will happen with high probability once we choose enough uniform samples from $[m]$. Moreover, the algorithm can immediately verify whether this event has occurred. The second event will happen with high enough probability for the random query x, conditioned on the event that $x \in f^{-1}(F)$ (This event cannot be verified by the algorithm, but the correctness does not need this).

In our case, $f|_{f^{-1}(F)}$ is not monotone but is guaranteed to be extremely close to monotone. That is, there exists a large subset M of $f^{-1}(F)$ in which f is monotone. This is not enough to carry the above argument though, as after making some queries, even if all queries are what a perfectly monotone f would be consistent with, the interval $[m]$ shrinks to possibly an interval in which M is not dense enough , which will prevent further success. This brings in the need for the next definition and lemma which maps a global density condition to a local density condition.

Definition 4.8 *Given a set $S \subset [n]$ and a $\gamma \in [0, 1]$, an element $i \in S$ is called γ-*deserted*, if there exists an interval $I \subset [n]$ containing i such that $|S \cap I| < \gamma|I|$.*

Suppose $f : [n] \to \mathbb{R}$ is a function (array) that is monotone increasing over a set $S \subset [n]$. Given $x = f(i)$ for some $i \in S$, we would like to find i using a randomized binary search. The binary search will be on the "right track" as long as we compare x to values of f on points in S alone. If i is γ-deserted, then the binary search for it may fall into an interval of $[n]$ in which the density of S-elements is low, and then continuing on the right track will be unlikely. So we would like the number of γ-deserted elements to be small. This motivates the following lemma.

Lemma 4.9 *Let $S \subset [n]$ with $|S| \geq \delta n$. For every $\gamma < 1$, at most $3\gamma(1-\delta)n/(1-\gamma)$ elements of S are γ-deserted.*

Proof. Let $S_\gamma \subset S$ be the set of γ-deserted elements in $[n]$. We bound $|S_\gamma|$ from above using an argument similar to the one used in a standard proof of the Vitali covering lemma [43].

We define the measure of an interval $I \subset [n]$ be $\mu(I) = |S \cap I|$. For each $i \in S_\gamma$, let I_i denote a maximal interval in $[n]$ containing i with $\mu(I_i) < \gamma |I_i|$. Let $\mathcal{I} = \{I_i : i \in S_\gamma\}$. Notice that no interval in $\mathcal{I}$ is properly contained in another.

Consider the greedy procedure which constructs a maximal collection $\mathcal{P} \subset \mathcal{I}$ of pairwise disjoint intervals, by iteratively choosing a maximum-measure interval from among the intervals in $\mathcal{I}$ which are disjoint with every interval already added to $\mathcal{P}$.

Let $P = \bigcup_{I \in \mathcal{P}} I$. Observe that

$$|P| \leq |S \cap P| + (1-\delta)n, \quad \text{and}$$
$$|S \cap P| = \sum_{I \in \mathcal{P}} \mu(I) < \sum_{I \in \mathcal{P}} \gamma |I| \leq \gamma |P|.$$

Combining the two estimates, we conclude that

$$|S \cap P| \leq \frac{\gamma(1-\delta)}{(1-\gamma)} n.$$

Next, we bound $|S_\gamma|$. If $i \in S_\gamma$ is not in P, then the greedy procedure did not include I_i in $\mathcal{P}$. By definition of the greedy procedure, if the interval $I_i \in \mathcal{I}$ is not in $\mathcal{P}$, then there exists an interval $I \in \mathcal{P}$ overlapping with I_i such that $\mu(I) \geq \mu(I_i)$. Hence, if we enlarge each interval $I \in \mathcal{P}$ so that it covers the nearest $\mu(I)$ more elements from S on both sides, then this collection of enlarged intervals from $\mathcal{P}$ cover all the elements of S_γ. Hence $|S_\gamma| \leq \sum_{I \in \mathcal{P}} 3\mu(I) = 3|S \cap P|$. ■

Remarks 4.10 *Lemma 4.9 will be used in two different regimes. The first regime is when δ is extremely close to 1. Then one can choose γ also quite close to 1 and still have very few elements of G to be γ-deserted. In particular, if $\delta = 1 - \epsilon/\log^5 n$ and $\gamma = 1 - 1/\log^3 n$, then at most $3\epsilon n/\log^2 n$ elements in G are γ-deserted. A second regime is when δ is close to 0. In this case, we choose $\gamma \ll \delta$ so that at most $3\gamma n \ll \delta n$ elements in G are γ-deserted.*

Theorem 4.11 *Let $f : [m] \to \mathbb{R}$, $F \subset \mathbb{R}$ and $\epsilon > 0$ be such that $A = f^{-1}(F)$ has size ϵm and $f|_A$ is $(\epsilon/\log^5 m)$-close to monotone increasing. Then there exists a set $A' \subset A$ with $|A'| \geq |A|(1 - \epsilon/\log m)$, so that if $f^{-1}(Q)$ contains any element of A', Algorithm 4.7 succeeds with probability at least $(1 - 1/\log m)$.*

Proof. Our first application of Lemma 4.9 is to $A \subset [m]$ with γ_1 set to $\epsilon^2/\log^2 m$. By the lemma, the set $A_{\gamma_1} \subset A$ of γ_1-deserted elements has size at most $3\gamma_1 m \leq 3\epsilon|A|/\log^2 m$. Since $f|_A$ is $(\epsilon/\log^5 m)$-close to monotone, there exists a monotone nondecreasing sequence of length $\delta|A|$ in A, where $\delta = 1 - \epsilon/\log^5 m$. Let $M \subset A$ be the support of this large monotone sequence. A second application of Lemma 4.9 is to M as a subset of A with $\gamma_2 = 1 - 1/\log^3 m$. (Technically, we apply the lemma to the set M' which corresponds to M once we remap A to an interval $[|A|]$ preserving the order.) We conclude that the set $M_{\gamma_2} \subset M$ of γ_2-deserted elements in A has size at most $3\epsilon|A|/\log^2 m$. The set A' in the statement of the theorem is $M \setminus (A_{\gamma_1} \cup M_{\gamma_2})$. It is clear that $|A'| \gg |A|(1 - \epsilon/\log m)$.

Let $i \in A'$ be such that $i \in f^{-1}(Q)$. When we run Algorithm 4.7, we say that the algorithm is on the *right track* if $i \in I$, where I is the current search interval. Since i is not γ_1-deserted in $[m]$, as long as the binary search is on the right track, a single query has a probability at least γ_1 of hitting an element in A. Let A_k be the event that the algorithm, while it is in its k-th iteration, hits an element $i \in A$ in Step 1 within the first $10\gamma_1^{-1} \log\log m$ trials. Then $P(A_k) \gg 1 - 1/\log^3 m, \forall k$. Let M_k denote the event that, the first element from A that the algorithm hits in its k-th iteration belongs to M. Since i is not γ_2-deserted in A, $P(M_k) \geq \gamma_2 = 1 - 1/\log^3 m$. Once both these events happen, the algorithm takes one more step in the right track by spending at most $10\gamma_1^{-1} \log\log m$ queries in the k-th iteration.

The probability that either A_k or M_k fail to happen for some $k \leq 100 \log m$ is, by union bound, at most $(100 \log m)(1/\log^3 m + 1/\log^3 m) \leq 1/(2\log m)$. The probability that a random binary search takes more than $100 \log m$ steps on an array of length m is much smaller than $1/(2\log m)$. Hence the algorithm succeeds with probability at least $(1 - 1/\log m)$.

Since A_k happens for all $k \leq 100 \log m$, the total number of queries made to f in this case is at most $1000\gamma_1^{-1} \log m \cdot \log\log m \leq 1000\epsilon^{-2} \log^4 m$. ∎

Now we are ready to give the complete description of Test 2.

Algorithm 4.12 ($\mathcal{A}_2(f, \epsilon)$) *The input is a function $f : [n] \to \mathbb{R}$. The output is either a $(1, 3, 2)$-tuple in f or FAIL.*

1. *Let $q = 100\epsilon^{-4} \log^{20} n$.*
   (The total number of queries made to f will be limited to $O(q)$.)
2. *Fix a "slice-width" $W = 2^w$, where w is chosen uniformly at random from $\{0, 1, \ldots, \lceil \log n \rceil - 1\}$. Slice $[n]$ into consecutive disjoint intervals $I_1, \ldots, I_{\lceil n/W \rceil}$, each of length W (except possibly the last one).*
3. *Pick a "slice-number" s uniformly at random from $\{1, \ldots, \lceil n/W \rceil - 1\}$.*
   Define $I_L = I_{s-2} \cup I_{s-1} \cup I_s$ and $I_R = I_{s+1} \cup I_{s+2}$.
   (In the above expressions, assume $I_i = \emptyset$ if $i \notin \{1, \ldots, \lceil n/W \rceil\}$)
4. *Query f at q points chosen independently and uniformly at random from I_L. Let i_0 be the index with a smallest f-value among these q points.*
   Let $I'_L = \{i \in I_L : f(i) > f(i_0)\}$ and $I'_R = \{i \in I_R : f(i) > f(i_0)\}$.
5. *Repeat* DyadicSampler$(I_R, 2)$ *independently q times. If it returns a $(2, 1)$-pair (j, k) in $I'_R \times I'_R$, then return the $(1, 3, 2)$-tuple (i_0, j, k) and terminate.*
   (Otherwise we show that, with high probability, $f|_{I'_R}$ is nearly monotone.)
6. *Run* DyadicSampler$(I_L, 2)$ *once. If it returns a $(1, 2)$-pair (i, j) in $I'_L \times I'_L$, then proceed to next step. Otherwise return FAIL.*
7. *Let $f_R = f|_{I_R}$. Define the "query range" $Q = \big(f(i), f(j)\big)$, and the "filter-range" $F = \big(f(i_0), \infty\big)$. Run* FilteredBinarySearch(f_R, F, Q, ϵ') *(Algorithm 4.7) where $\epsilon' = \frac{1}{8}\epsilon \log^{-1} n$. If it succeeds in returning an index $k \in I_R$ with $f(k) \in Q$, then return the $(1, 3, 2)$-tuple (i, j, k). Otherwise return FAIL.*

Lemma 4.13 *Let $f : [n] \to \mathbb{R}$ contain a matching T_2 of ϵn $(1, 3, 2)$-tuples with leap-start 2. Then Algorithm 4.12 called with arguments (f, ϵ) returns a $(1, 3, 2)$-tuple in f with a probability at least $\Omega\left(\epsilon^3/\log^6 n\right)$. Moreover, the algorithm makes at most $O\left(\epsilon^{-4} \log^{20} n\right)$ queries to f.*

Proof. The claim on query complexity is obvious once we notice that the the FilteredBinarySearch called in Step 7 makes at most $(\epsilon')^{-2} \log^4 n$ queries which is $O(q)$. We only need to analyze the probability of success. Next, we define success for each step of Algorithm 4.12 and provide a lower bound on the success probability of each step conditioned on the event that every step prior to it is successful.

Step 2. For each $w \in \{0, \dots, \lceil \log n \rceil - 1\}$ let $T_{2,w}$ denote the tuples (i,j,k) in T_2 with leap-size $\lfloor \log(k-j) \rfloor = w$. Step 2 is considered successful if it chooses a w so that $|T_{2,w}| \geq \epsilon n / \log n$. Since $\bigcup_{w=0}^{\lceil \log n \rceil - 1} T_{2,w} = T$, there exists at least one w with $|T_{2,w}| \geq \epsilon n / \log n$. Hence Step 2 succeeds with probability $p_2 \geq 1/\log n$.

Step 3. For the w chosen in previous step, and for each $s \in [\lceil n/2^w \rceil]$, let $T_{2,w,s}$ denote the tuples (i,j,k) in $T_{2,w}$ with $j \in I_s$, where I_s is the s-th slice of width $W = 2^w$ in $[n]$. Step 3 is considered successful if it chooses an s so that $|T_{2,w,s}| \geq \left(\frac{1}{2}\epsilon \log^{-1} n\right) W$. If the previous step is successful, we have $|T_{2,w}| \geq \epsilon n / \log n$. A Markov inequality over s implies that $|T_{2,w,s}| \geq \left(\frac{1}{2}\epsilon \log^{-1} n\right) W$, for at least $\left(\frac{1}{2}\epsilon \log^{-1} n\right)$ fraction of choices of s from $[\lceil n/W \rceil]$. Hence, Step 3 succeeds with probability $p_3 \geq \left(\frac{1}{2}\epsilon \log^{-1} n\right)$, conditioned on the event that Step 2 is successful.

Step 4. In what follows, we assume that the previous steps are successful and thus $|T_{2,w,s}| \geq \left(\frac{1}{2}\epsilon \log^{-1} n\right) W$. Let $T = T_{2,w,s}$ and m_T be the median f-value in $T(1)$. Step 4 is considered successful if $f(i_0) < m_T$. Let I_L and I_R be the intervals chosen by Algorithm 4.12 in Step 3. In particular, $|I_L| \leq 3W$ and $|I_R| \leq 2W$. Notice that every tuple $(i,j,k) \in T_{2,w,s}$ satisfies $i, j \in I_L$ (since $j - i < k - j < 2W$) and $k \in I_R$. The probability that a single i chosen uniformly at random from I_L has $f(i) > m_T$ at least $\frac{1}{2}|T|/|I_L| \geq \frac{1}{12}\epsilon \log^{-1} n$. So the probability that no i from the q trials has $f(i) < m_T$ is $o(1)$. That is, when steps 2 and 3 are successful, Step 4 succeeds with probability $p_4 = 1 - o(1)$.

Step 5. Step 5 is considered successful if it returns a $(2,1)$-pair (j,k) in $I'_R \times I'_R$. In this case the entire algorithm is successful and hence it terminates. We expect this step to succeed only if $f|_{I'_R}$ is $(\epsilon \log^{-5} n)$-far from monotone. Otherwise, we rely on the next two steps. If $f|_{I'_R}$ is $(\epsilon \log^{-5} n)$-far from monotone, then f contains a matching M of $(2,1)$-pairs from $I'_R \times I'_R$ with $|M| \geq \frac{1}{2}(\epsilon \log^{-5} n)|I'_R|$. When all the previous steps are successful, $|I'_R| \geq \frac{1}{2}|T| \geq \frac{1}{4}\left(\epsilon \log^{-1} n\right) W$ and so $|M| \geq \frac{1}{8}\left(\epsilon^2 \log^{-6} n\right) W$. So a single call to the dyadic sampler returns a $(2,1)$-pair from $I'_R \times I'_R$ with probability at least $\Omega(\epsilon^4 \log^{-14} n)$ (Theorem 3.4). Therefore at least one of the q trials succeed with probability $1 - o(1)$. That is, in this case, Step 5 succeeds with probability $p_5 = 1 - o(1)$ and thus the whole algorithm succeeds with probability $\Pi_{t=2}^5 p_i = \Omega(\epsilon \log^{-2} n)$.

Step 6. In what follows, we assume that the steps 2 to 4 were successful and Step 5 was not successful. Step 6 is considered successful if it returns a $(1,2)$-pair from $I_L \times I_L$ which can be extended to a $(1,3,2)$-tuple (i,j,k) where k is a member of I'_R that can be quickly found by running Algorithm 4.7 in the interval I_R with filter-range $F = (f(i_0), \infty)$ and query-range $Q = (f(i), f(j))$. The set I'_R plays the role of A in Theorem 4.11. Since Step 4 is successful, We know that $|A| = |I'_R| \geq \epsilon' |I_R|$ where $\epsilon' = \frac{1}{8}\left(\epsilon \log^{-1} n\right)$. Moreover, since Step 5 failed, we are already under the assumption that f restricted to A is $(\epsilon/\log^5 n)$-close to monotone increasing. Hence the theorem guarantees the existence of a set $A' \subset A$ with size $(1 - \epsilon'/\log m))|A|$ of "quickly searchable" indices.

Consider the matching of $(1,2)$-pairs $S = \{(i,j) : (i,j,k) \in T, f(i) > f(i_0)\}$ and its subset $S' = \{(i,j) : (i,j,k) \in T, f(i) > f(i_0), k \in A'\}$. Since Step 4 is successful, $f(i_0) < m_T$ and thus $|S| \geq \frac{1}{2}|T|$. By Theorem 4.11, $|S'| \geq (1 - o(1))|S|$. By Claim 4.2, Step 6 returns a $(1,2)$-pair (i,j) from $(S')^*$ which dominates S' with probability $p_6 = \Omega(\epsilon^2 \log^{-4} n)$. (Remark: We would have repeated this step also q times if there was any way of deciding whether a pair dominates S.)

Step 7. If Step 6 is successful, then Algorithm 4.7 succeeds in finding k with probability $1 - o(1)$. Thus the whole algorithm succeeds with probability $\Omega(\epsilon^3 \log^{-6} n)$. ∎

Repeating Algorithm 4.12 $O(\epsilon^{-3} \log^6 n)$ times returns a $(1,3,2)$-tuple in f with probability close to 1 when $|T_2| \geq \epsilon n/6$. Thus, Theorem 4.3 follows from Lemmas 4.4 and 4.13.

4.2 A non-adaptive $(1,3,2)$-tester

Recall that only Test 2 in the adaptive $(1,3,2)$-tester was adaptive. When the majority of the tuples in the matching T of $(1,3,2)$-tuples have leap-start 1, Test 1 succeeds in finding a $(1,3,2)$-tuple with high probability using only $O(\epsilon^{-3}\log^5 n)$ queries. Now we describe a non-adaptive $\widetilde{O}(\sqrt{n})$ tester which will succeed with high probability when the majority of tuples in T have leap-start 2.

To explain the intuition behind the non-adaptive tester without getting into the quantitative details, let us assume that for two disjoint intervals I_L and I_R of length m in $[n]$ (with I_L to the left of I_R), there is a large matching T of $(1,3,2)$-tuples with $T(1)\cup T(2)\subset I_L$ and $T(3)\subset I_R$. If $|T|\approx\epsilon|I_L|$, then, if we sample a pair (i,j) from I_L using the dyadic sampler, with probability at least $\Omega(\epsilon^2\log^{-2} m)$, the pair (i,j) is a $(1,2)$-pair which dominates a $(1,2)$-pair in $T(1,2)=\{(t_1,t_2)\ :\ (t_1,t_2,t_3)\in T\}$ (Claim 4.2). If we repeat this dyadic sampling $\Omega(\epsilon^{-1}\sqrt{m}\log^2 m)$ times, then with high probability, we get a collection D of $\Omega(\epsilon\sqrt{m})$ many $(1,2)$-pairs in I_L, each of which dominates a different $(1,2)$-pair in $T(1,2)$.

Since T is a matching, for each pair $(i,j)\in D$, there is a distinct index $k\in I_R$, such that (i,j,k) is a $(1,3,2)$-tuple. Let $K\subset I_R$ be the collection of such indices. Then $|K|\geq|D|=\Omega(\epsilon\sqrt{m})$. Now any collection of $\Omega(\epsilon^{-1}\sqrt{m})$ uniform samples from I_R hits a member of K with high probability. Hence if we sample $\Omega(\epsilon^{-1}\sqrt{m}\log^2 m)$ pairs from I_L using the dyadic sampler and an equal number of uniform point-samples from I_R, we hit a $(1,3,2)$-tuple with very high probability.

Wrapping this up inside the dyadic slicing argument that we have used twice before, we can conclude that testing for $(1,3,2)$-pattern can be done in $\widetilde{O}(\epsilon^{-1}\sqrt{n})$ queries. As [5] proves a similar result for any further details in this paper.

4.3 A non-adaptive $\{(1,3,2),(3,1,2)\}$-tester

Here we justify the note made in Section 1.1 about the testability of Gilbreath shuffling, i.e., $\{(1,3,2),(3,1,2)\}$-freeness. In general, ϵ-testing for a finite set of forbidden patterns $A\subseteq\mathfrak{S}_k$ can be done as discussed in Proposition 2.3. However, the lower bounds do not follow, and indeed $\{(1,3,2),(3,1,2)\}$-freeness can be tested using $\mathrm{poly}(\log n)$ queries.

We sketch here why such non-adaptive 1-sided error testing works. If the given sequence if ϵ-far from being $\{(1,3,2),(3,1,2)\}$-free, then there exists an $(\epsilon n/6)$-sized matching of either $(1,3,2)$ or $(3,1,2)$ tuples. Let us assume the former to be the case (the analysis for the latter being similar). If a majority of $(1,3,2)$-tuples in this large matching have leap-start 1, then the Dyadic Sampler succeeds in finding a $(1,3,2)$-tuple with high probability because a median-split and concatenate works in this case. Otherwise we can assume that there exists two adjacent intervals I_L and I_R (I_L to the left of I_R) such that there exists a "large" matching T of $(1,3,2)$-tuples with $T(1)\cup T(2)\subset I_L$ and $T(3)\subset I_R$. By "large", we mean that the size of T is of the order of of $\epsilon|I_L|$ and $\epsilon|I_R|$ up to poly-logarithmic factors. Now we partition T into T_1, T_2 and T_3 by sorting the tuples in T according to their value of the third coordinate and assigning the first one-third among the tuples to T_1, the middle one-third to T_2 and the rest to T_3. The pertinent combinatorial observation to make here is that if we find some $i\in T_1(1)$, $j\in T_3(2)$ and $k\in T_2(3)$, then if $i<j$, (i,j,k) forms a $(1,3,2)$-tuple and if $i>j$, (j,i,k) forms a $(3,1,2)$-tuple. A uniform sampling in I_L and I_R has sufficient probability to find such an i, j and k.

5 Lower bounds for non-adaptive testers

5.1 The pattern $(1,3,2)$.

The permutation $(1,3,2)\in\mathfrak{S}_3$ is a smallest pattern that is not monotone. Note that testing a function $f:[n]\to\mathbb{R}$ for the pattern $(2,3,1)$ is equivalent to testing for the $(1,3,2)$ pattern in the reversal of f

and testing for $(3,1,2)$ and $(2,1,3)$ patterns are, respectively, equivalent to testing for $(1,3,2)$ and $(2,3,1)$ patterns in $(-f)$. Hence $(1,3,2)$-testing is equivalent to testing of every non-monotone pattern in $\mathfrak{S}_3$.

Theorem 5.1 *Any one-sided-error non-adaptive ϵ-tester for the pattern $(1,3,2)$ has query complexity $\Omega(\sqrt{n})$, for every $\epsilon \leq 1/4$.*

Proof. For the sake of contradiction, assume that there exists a one-sided error non-adaptive $(1/4)$-tester, $\mathcal{A}$, for the pattern $(1,3,2)$ with query complexity $q < \sqrt{n}/2$. We will show that the success probability of $\mathcal{A}$ is less than $1/4$, contradicting the assumption that $\mathcal{A}$ is a tester. Note that, by success probability, we mean the probability by which $\mathcal{A}$ rejects an input that is $(1/4)$-far from being $(1,3,2)$-free.

We do so using Yao's principle. That is, we define a distribution $\mathcal{D}$ over the inputs and show that any deterministic algorithm for the task has probability of success at most $1/4$ when inputs are sampled according to $\mathcal{D}$. A deterministic algorithm for the problem is allowed to query the values of the input f on a predetermined set $Q \subset [n]$ of q indices and either accept or reject f. It is easy to see that if f restricted to Q is $(1,3,2)$-free, there exists an $f' : [n] \to \mathbb{R}$ which is $(1,3,2)$-free and $\forall i \in Q, f'(i) = f(i)$. For instance, one can construct f' by setting $\forall j \in [n], f'(j) = f(j^*)$, where $j^* \in [n]$ is the nearest index to j that is also in Q. This means that, an algorithm which has to accept all the $(1,3,2)$-free inputs, can reject an input only if it finds a $(1,3,2)$-tuple in f among the indices in Q.

The distribution $\mathcal{D}$ over input arrays of length $n = 4m$ is formed by picking a number k uniformly at random from $[m]$ and selecting the input to be the array f_k defined as follows.

$$\begin{array}{lcll} f_k(2m-2i+1) & = & 3i+1, & i \in [m] \\ f_k(2m-2i+2) & = & 3i+3, & i \in [m] \\ f_k(2m+i) & = & 3(i-k)+2, & i \in [2m] \end{array}$$

Figure 5.1 illustrates f_k when $m = 5$ and $k = 3$. Notice that, for every $i \in [m]$, the tuple $(2m-2i+1, 2m-2i+2, 2m+i+k)$ is a $(1,3,2)$-tuple in f_k and hence f_k is $(1/4)$-far from being $(1,3,2)$-free. Moreover, these are the only $(1,3,2)$-tuples in f_k because (i) the first half of f is $(1,3,2)$-free, (ii) the second half is $(2,1)$-free, and (iii) the only $(1,2)$-pairs in the first half are $(2m-2i+1, 2m-2i+2), i \in [m]$.

Let $\mathcal{A}'$ be any deterministic one-sided error algorithm for the problem. Let $Q \subset [n]$ be the set of indices for which $\mathcal{A}'$ queries the input. For the family of inputs f_k defined above, Q will contain a $(1,3,2)$-tuple only if $\exists i \in [m]$ such that $\{2m-2i+1, 2m-2i+2, 2m+i+k\} \subset Q$. Hence the set D defined as $D = \{(y-2m) - \lfloor (2m-x+1)/2 \rfloor : x, y \in Q, x \leq 2m < y\}$ should contain k. Since $|D| \leq (q/2)^2$ which is less than $n/16$, and k is chosen uniformly at random from $[n/4]$, the probability that D contains k is less than $1/4$. Therefore, the probability that $\mathcal{A}$ succeeds in rejecting inputs sampled according to $\mathcal{D}$ is at most $1/4$. ∎

Remarks 5.2 *A more complicated argument which could be used to show for each $m \geq 2$, the existence of a pattern of length $(2m-1)$ for which every non-adaptive tester needs to have a query complexity of $\Omega(n^{1-1/m})$ was used in a preliminary version of this draft (SODA2017). We are moving that argument to the appendix since these lower bounds for $m \geq 3$ were recently improved in [5].*

5.2 General non-monotone patterns

Intuitively it sounds natural that testing for not containing a longer non-monotone pattern is as hard as testing $(1,3,2)$-freeness, as discovering a non-monotone π in f discovers also a $(1,3,2)$ subpattern (or one of the other similarly hard to test non-monotone length-3 patterns). However, this does not make a formal proof. In this section we present such a proof in a strong setting (which includes also 2-sided error testing). We

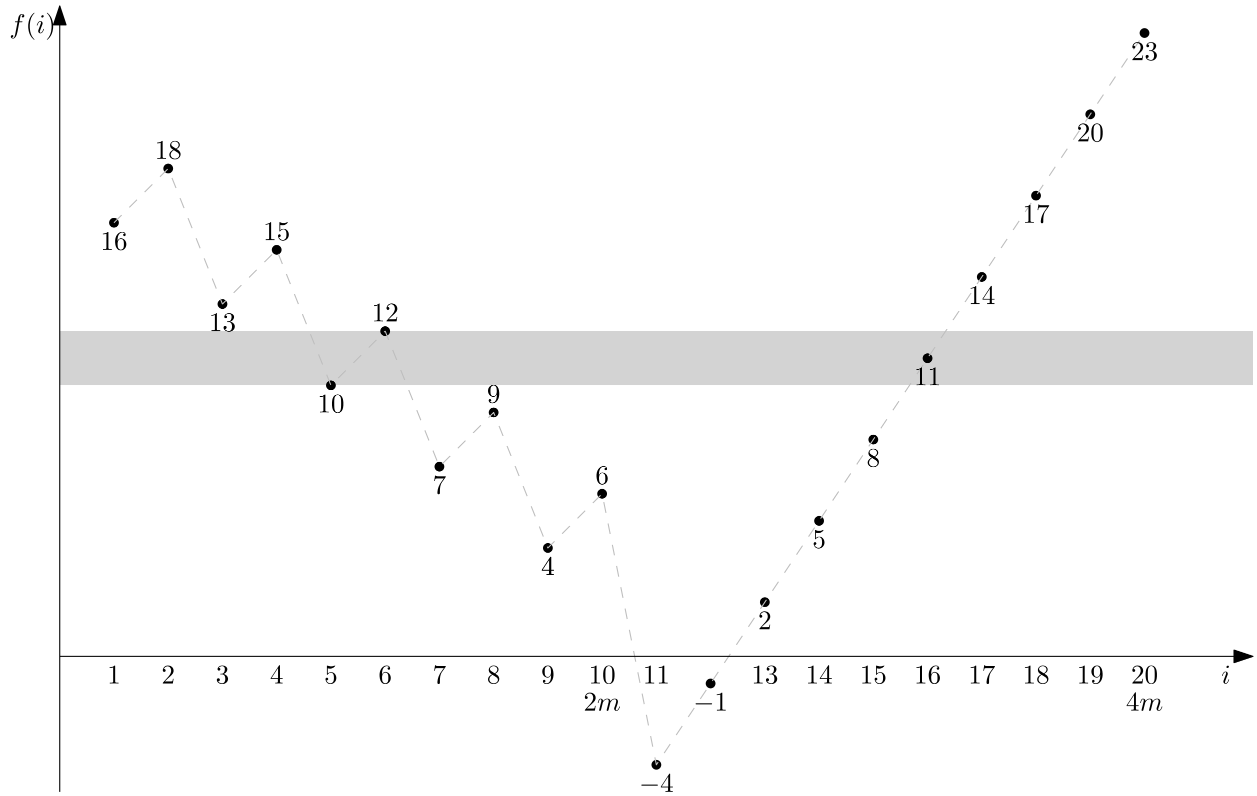

Figure 1: An illustration of the function f_k used in the proof of Theorem 5.1 with $m = 5$ and $k = 3$. The shaded region contains one of the $(1, 3, 2)$ triples in f_k.

show that the problem of $(1, 3, 2)$-freeness can be reduced to the problem of testing for any non-monotone pattern.

Theorem 5.3 *Let π be a non-monotone pattern in $\mathfrak{S}_k$, $k \geq 4$. Testing for π-freeness can be reduced to $(3, 1, 2)$-freeness (for adaptive / non-adaptive setting, and for 1-sided / 2-sided error testers). In particular this implies that any one-sided-error non-adaptive ϵ-tester for π has query complexity $\Omega(\sqrt{n})$, for every $\epsilon \leq \frac{1}{9(k-2)}$.*

For the set of input functions $f : [n] \to \mathbb{R}$ that the theorem holds for, we assume that there is a known positive real M such that $f([n]) \subseteq [-M, M]$.

We first observe that any non-monotone pattern $\pi \in \mathfrak{S}_k$ contains a length-3 non-monotone pattern with *values* in a contiguous interval of $[k]$.

Proposition 5.4 *Let $k \geq 4$, and $\pi \in \mathfrak{S}_k$ a non-monotone pattern. Then there exists $i \in [k-2]$ such that $\pi|_{\{\pi^{-1}(i), \pi^{-1}(i+1), \pi^{-1}(i+2)\}}$ is non-monotone.*

We break the non-monotone pattern π into two subsequences $\hat{\pi} \in \mathfrak{S}_3$, and $\pi' \in \mathfrak{S}_{k-3}$ as follows.

Let i be the smallest value so that $\pi|_{\{\pi^{-1}(i), \pi^{-1}(i+1), \pi^{-1}(i+2)\}}$ is non-monotone. Define $\hat{I} = \{\pi^{-1}(i), \pi^{-1}(i+1), \pi^{-1}(i+2)\}$ and $I' = [k] \setminus \hat{I}$. Let $\hat{\pi} \in \mathfrak{S}_3$ and $\pi' \in \mathfrak{S}_{k-3}$ be the permutations order isomorphic to the restriction of π on, respectively, $\hat{I}$ and I'.

Let $f : [n] \to \mathbb{R}$ be any function for which we want to test for $\hat{\pi}$-freeness. We construct a function $f_\pi : [m] \to \mathbb{R}$, $m = (k-2)n + (k-3)$, with the goal that f contains $\hat{\pi}$ if and only if f_π contains π. Moreover, if f is far from being $\hat{\pi}$-free then f_π is far (with somewhat smaller distance) from being π-free. Doing it in a "local" way will imply the reduction, and the corresponding lower bound.

To better understand the construction, consider first the following example. Let $k = 8$ and $\pi = (1, 2, 3, 6, 8, 4, 7, 5)$. The interval $\{i, i+1, i+2\}$ of Proposition 5.4 in this case is $[4, 5, 6]$; values that appear non-monotonically in π, namely in the order $(6, 4, 5)$ corresponding to the order permutation $\hat{\pi} = (3, 1, 2)$. Hence $i = 4$. $\pi' = (1, 2, 3, 5, 4)$ in this case, as this is the induced order in π on the values not in $[4, 5, 6]$.

Let $f : [n] \to \mathbb{R}$ that we want to test for being $(3, 1, 2)$-free (that is $\hat{\pi}$-free). In f_π we construct, $f_\pi(6j) = f(j),\ j = 1 \dots n$. Thus every $(3, 1, 2)$-tuple (t_1, t_2, t_3) in f will also correspond to a $(3, 1, 2)$-tuple in f_π in the places $(6t_1, 6t_2, 6t_3)$. In the 5-consecutive indices before each $6j$ we will insert values that are independent of the value of f, and are outside $[-M, M]$. They will augment any $(3, 1, 2)$-tuple of the form $(6t_, 6t_2, 6t_3)$ to a π-tuple. This will be done by placing in the 5 values before each $6j$, the values $-M-3,\ -M-2,\ -M-1, M+2, M+1$. Note that by doing so, the $(3, 1, 2)$-tuple $(6t_1, 6t_2, 6t_3)$ is augmented to the π-tuple $(6t_1 - 5, 6t_1 - 4, 6t_1 - 3, 6t_1, 6t_2 - 2, 6t_2, 6t_3 - 1, 6t_3)$.

Further, due to the above local augmentation, a matching of $(3, 1, 2)$ tuples in f will correspond to the same size matching of π-tuples in f_π. Moreover, no other π-tuples will be formed. Thus testing $(3, 1, 2)$-freeness for f will be reduced to testing π freeness for f_π ($1/6$ of the distance parameter as the length is increased by a factor of 6).

We end now with a formal description of the reduction. Let $f : [n] \to [-M, M]$, and $\pi, \hat{\pi}, \pi'$ and i as above.

For every $s \in \{0, \dots, n\}$ and $t \in [k-3]$,

$$f_\pi(s(k-2)) = f(s), \quad \text{if } s \neq 0,$$

$$f_\pi(s(k-2) + t) = \begin{cases} +M + \pi'(t), & \text{if } \pi'(t) \geq i \\ -M - k + \pi'(t), & \text{if } \pi'(t) < i. \end{cases}$$

One can verify that if (s_1, s_2, s_3) is a $\hat{\pi}$-tuple in f, then f_π contains π-tuple in the "$(k-3)$-neighborhood" of (s_1, s_2, s_3), that is, among the set of indices $\bigcup_{i=1}^{3} \{j \in [m] : |j - (k-2)s_i| \leq k-3\}$. Moreover, if f contains a matching of t $\hat{\pi}$-tuples, then f_π contains a matching of t π-tuples.

On the other hand, let $(p_1, \dots, p_k)$ be a π-tuple in f_π. Let $\hat{P}$ be the set of indices in $(p_1, \dots, p_k)$ corresponding to $\hat{I}$ in π (in the order-isomorphism). The first observation is that, $f_\pi(i) \in [-M, M]$ if and only if $i = 0 \mod (k-2)$. Furthermore, we observe that the image of f_π has exactly $i-1$ values smaller than $-M$ and exactly $(k-i-3)$ values larger than M. The third observation is that, for every $p \in \hat{P}$, there are at least $(i-1)$ indices in $[m]$ (in fact, in P) that are strictly smaller in f_π-value than $f_\pi(p)$ and at least $(k-i-3)$ indices in $[m]$ that are strictly larger in f_π-value than $f_\pi(p)$. These three observations suffice to conclude that $f_\pi(p) \in [-M, M]$. Thus, $p = 0 \mod (k-2)$, $\forall p_i \in \hat{P}$. Therefore, $\hat{P}$ (after scaling down by $(k-2)$) corresponds to a $\hat{\pi}$-tuple in f.

Based on these two observations we conclude that, if f is $\hat{\pi}$-free, then f_π is π-free and if f is ϵ-far from being $\hat{\pi}$-free, then f_π is $\epsilon/(k-2)$-far from being π-free. Hence an ϵ-tester for $\hat{\pi}$ reduces to an $\epsilon/(k-2)$-tester for π. Since testing for any non-monotone pattern in $\mathfrak{S}_3$ is equivalent to testing for the pattern $(1, 3, 2)$, Theorem 5.3 follows from Theorem 5.1.

6 Open problems and further discussion

In a preliminary version of this draft (SODA2017), we posed three open problems, of which two where already solved. We give an account of these problems, with an additional one here below.

1. The main open problem concerns the complexity of general testing for non-monotone patterns. We have seen that for length-3 patterns there is a poly-logarithmic adaptive tester, while we have shown the impossibility of non-adaptive tester of poly-logarithmic complexity.

   Could it be that for *any* constant-size pattern π there there exists a poly-logarithmic adaptive tester for π-freeness? What happens if we allow 2-sided error testing?

2. How does the structure of a pattern π correlate with the complexity of an optimal non-adaptive tester for π-freeness? There are partial results in this direction. Ben-Eliezer and Canonne have given characterizations for the hardest patterns for non-adaptive testing [5]. In particular, they also construct patterns of arbitrary large constant length k, that can be tested non-adaptively in say $O(n^{2/3})$.

3. Being π-free is an hereditary property of strings. Being hereditary here means that if $f : [n] \to \mathbb{R}$ has the property, then so does $f|_{[n]\setminus\{i\}}$ for any $i \in [n]$. In the SODA17 proceedings we have posed the question whether any hereditary property of sequences can be tested with sub-linear query complexity? This was answered negatively in [26].

4. Our upper bounds hold of course for $f : [n] \to [n]$ being a permutation. However, our lower bounds do not hold for permutations. Is it true that permutations can be tested for being π-free much more efficient than general sequences?

References

[1] Shlomo Ahal and Yuri Rabinovich. On complexity of the subpattern problem. *SIAM Journal on Discrete Mathematics*, 22(2):629–649, 2008.

[2] Nir Ailon, Bernard Chazelle. Information theory in property testing and monotonicity testing in higher dimension. Inf. Comput. 204(11): 1704–1717, 2006.

[3] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pp. 645–655, 1999.

[4] Aleksandrs Belovs, Eric Blais. A polynomial lower bound for testing monotonicity. In *Proceedings of the 48th ACM Symposium on Theory of Computing Conference*, pp. 1021–1032, 2016.

[5] Omri Ben-Eliezer and Clément L. Canonne. Improved Bounds for Testing Forbidden Order Patterns. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2093–2112, 2018.

[6] Michael Ben Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 221–230. IEEE, 2008.

[7] Petra Berenbrink, Funda Ergün, and Tom Friedetzky. Finding frequent patterns in a string in sublinear time. In *Proceedings of the 13th Annual European Symposium on Algorithms*, pp. 746–757, 2005.

[8] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop. Technical Report WS-94-03*, pages 359–370, 1994.

[9] Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David P. Woodruff. Transitive-closure spanners. *SIAM J. Comput.*, 41(6):1380–1425, 2012.

[10] Hadley Black, Deeparnab Chakrabarty, C. Seshadhri. A $o(d)\Delta polylogn$ Monotonicity Tester for Boolean Functions over the Hypergrid $[n]^d$. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2133-2151, 2018.

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

[12] Piotr Berman, Sofya Raskhodnikova, Grigory Yaroslavtsev. L_p-testing. In *Proceedings of the 46th ACM Symposium on Theory of Computing Conference*, pp. 164–173, 2014.

[13] Miklós Bóna. *Combinatorics of permutations*. Discrete mathematics and its applications. Chapman & Hall/CRC Press, 2004.

[14] Jop Briët, Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Monotonicity testing and shortest-path routing on the cube. In *Proceedings of the 14th International Workshop on Randomization and Computation*, pp. 462–475, 2010.

[15] Deeparnab Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and lipschitz testing over hypercubes and hypergrids. In *Proceedings of the 45th ACM Symposium on Theory of Computing Conference*, pp. 419–428, 2013.

[16] Deeparnab Chakrabarty, C. Seshadhri. An Optimal Lower Bound for Monotonicity Testing over Hypergrids. In *Proceedings of the 17th International Workshop on Randomization and Computation*, pp. 425–435, 2013.

[17] Deeparnab Chakrabarty and C. Seshadhri. An o(n) monotonicity tester for boolean functions over the hypercube. *SIAM J. Comput.*, 45(2):461–472, 2016.

[18] Xi Chen, Rocco Servedio, Li-Yang Tan. New Algorithms and Lower Bounds for Monotonicity Testing. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science*, pp. 286-295, 2014.

[19] Xi Chen, Erik Waingarten, Jinyu Xie. Beyond Talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th ACM Symposium on Theory of Computing Conference*, pp. 523-536, 2017.

[20] Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *Proceedings of the 3rd International Workshop on Randomization and Approximation Techniques in Computer Science*, pp. 97–108, 1999.

[21] Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pp. 259–268, 1998.

[22] Eldar Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004.

[23] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pp. 474–483, 2002.

[24] Eldar Fischer and Ilan Newman. Testing of matrix properties. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing*, pp. 286–295, 2001.

[25] Jacob Fox. Stanley-wilf limits are typically exponential. *CoRR*, abs/1310.8378, 2013.

[26] Cody R. Freitag, Eric Price, and William J. Swartworth. Testing Hereditary Properties of Sequences. In *Proceeding of the 21st Workshop on Randomization and Computation, pp. 44:1–44:10, 2017*.

[27] Oded Goldreich, Shafi Goldwasser, Dana Ron. Property Testing and its Connection to Learning and Approximation. Journal of the ACM, 45(4): 653-750, 1998.

[28] Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

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

[30] Shirley Halevy and Eyal Kushilevitz. Testing monotonicity over graph products. *Random Struct. Algorithms*, 33(1):44–67, 2008.

[31] Carlos Hoppen, Yoshiharu Kohayakawa, Carlos Gustavo T. de A. Moreira, and Rudini Menezes Sampaio. Testing permutation properties through subpermutations. *Theor. Comput. Sci.*, 412(29):3555–3567, 2011.

[32] Richard M Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 881–890, 2007.

[33] Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 550–556, 2002.

[34] Subhash Khot, Dor Minzer, Muli Safra. On Monotonicity Testing and Boolean Isoperimetric Type Theorems. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science*, pp. 52-58, 2015.

[35] Sergey Kitaev. *Patterns in Permutations and Words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011.

[36] Tereza Klimošová and Daniel Král. Hereditary properties of permutations are strongly testable. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1164–1173, 2014.

[37] Oded Lachish and Ilan Newman. Testing periodicity. In *Proceedings of the 9th International Workshop on Randomization and Computation*, pp. 366–377, 2005.

[38] Adam Marcus and Gábor Tardos. Excluded permutation matrices and the stanley–wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107(1):153–160, 2004.

[39] Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for Forbidden Order Patterns in an Array. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1582-1597, 2017.

[40] Pranav Patel, Eamonn J. Keogh, Jessica Lin, and Stefano Lonardi. Mining motifs in massive time series databases. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pp. 370–377, 2002.

[41] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[42] Bridget Eileen Tenner. Database of Permutation Pattern Avoidance. http://math.depaul.edu/bridget/patterns.html.

[43] Wikipedia. Vitali covering lemma — wikipedia, the free encyclopedia, 2016. [Online; accessed 14-April-2016].

A More complex non-monotone patterns

In this section, we prove

Theorem A.1 *Any one-sided-error non-adaptive ϵ-tester for the $(2m-1)$-pattern $(1, 2m-1, 2m-2, 2, 3, 2m-3, .., m)$ has query complexity $\Omega(n^{1-1/m})$ for every $\epsilon \leq 1/(6m-3)$.*

Our strategy is to define a certain search task, which we call the "intersection-search". Let $\mathcal{C}$ be the class of algorithms for this search task. We show that every algorithm from $\mathcal{C}$ has a large query complexity. Finally, we reduce the intersection-search task to the testing of $(1, 3, 2)$-freeness.

Problem A.2 (Intersection-search) *The input to the problem consists of m arrays $A_j, j \in [m]$, each containing $3n$ distinct integers in ascending order. It is promised that at least n elements are common to all the m arrays. The goal is to find an m-tuple $(i_1, \dots, i_m) \in [3n]^m$ such that $A_1[i_1] = \cdots = A_m[i_m]$.*

Notice that this is an easy task for a randomized adaptive algorithm. Selecting constantly many elements from A_1 uniformly at random and searching for their location in each of $A_j, j \geq 2$ using a binary search is bound to succeed with high probability. This suggests an adaptive algorithm of $O(m \log n)$ query complexity. However, we are interested here in non-adaptive algorithms. In this setting it can be seen, as in Theorem 2.4, that $O\left(n^{1-1/m}\right)$ query locations from each $A_j, j \in [m]$ independently and uniformly at random will contain a witness with high probability. We argue next that one cannot do much better. For this we first define formally the class of algorithms $\mathcal{C}$ that we are willing to accept.

Definition A.3 *An algorithm for intersection-search is in the class $\mathcal{C}$ if it operates as follows:*

It first chooses m sets $Q_j \subset [3n], j \in [m]$, according to some distribution, before seeing any values in the input arrays. It then queries each array A_j at the indices in Q_j. After it sees all the query outcomes, it is free to do any amount of computation. It then outputs one of two types of answers: either an m-tuple in $[3n]^m$ or "fail". If it outputs an m-tuple $(i_1, \dots, i_m)$, then surely $A_1[i_1] = \cdots = A_m[i_m]$. That is, for every possible input $(A_1, \dots, A_m)$ consistent with the values viewed, it holds that $A_1[i_1] = \cdots = A_m[i_m]$.

The algorithm is said to succeed if it returns an m-tuple. The success probability of the algorithm is the worst-case success probability, i.e., the minimum over all inputs. The query complexity of the algorithm is $\sum_{j=1}^{m} |Q_j|$.

Lemma A.4 *Let $\mathcal{A}$ be an algorithm in class $\mathcal{C}$ for Problem A.2 on m arrays. If $\mathcal{A}$ makes $q < n/2$ queries, then the success probability of $\mathcal{A}$ is at most $(q/m)^m / n^{m-1}$.*

Proof. We use Yao's principle to lower bound the success probability of $\mathcal{A}$. That is, we define a distribution $\mathcal{D}$ on the valid inputs to the problem, and show that any deterministic non-adaptive algorithm for Problem A.2 has a probability of success at most $(q/m)^m / n^{m-1}$, when the inputs are sampled according to $\mathcal{D}$. Recall that the deterministic algorithms we need to consider are those which output $(i_1, \dots, i_m)$ only when it is sure that $A_1[i_1] = \cdots = A_m[i_m]$ and output "fail" otherwise. The success probability of such a deterministic algorithm is the proportion of inputs (under the distribution $\mathcal{D}$) for which it outputs a tuple $(i_1, \dots, i_m)$.

We define $\mathcal{D}$ by prescribing a randomized procedure to select m monotone increasing length-$3n$ arrays $A_1, \dots, A_m$ with n elements common to all of them. The randomness is four-fold; (i) we pick a 0-1 vector $x = (x_1, \dots, x_{3n})$ uniformly at random, (ii) independently pick a set $S \subset [2n]$ of size n uniformly at random, (iii) independently pick a vector $p = (p_1, \dots, p_{3n}) \in \{2, \dots, m\}^{3n}$ uniformly at random, and (iv) independently pick a vector $k = (k_2, \dots, k_m) \in [n]^{m-1}$ uniformly at random. The first three types of randomness will be used to ensure that a 0-error algorithm can return a tuple $(i_1, \dots, i_m)$ only if it hits the tuple, i.e., the algorithm indeed queries $A_j[i_j]$ for all $j \in [m]$. The fourth randomness makes such hits unlikely within $O(n^{1-1/m})$ queries.

The arrays $A_1, \dots, A_m$ are defined as follows:

$$A_1[i] = 2i + x_i, \quad 1 \leq i \leq 3n,$$

and for $2 \le j \le m$,

$$A_j[i] = \begin{cases} 2(i-k_j), & 1 \le i \le k_j, \\ 2(i-k_j) + x_{i-k_j}, & k_j < i \le 3n, \\ & p_{i-k_j} \neq j \\ 2(i-k_j) + x_{i-k_j}, & k_j < i \le 3n, \\ & p_{i-k_j} = j, \\ & i - k_j \in S, \\ 2(i-k_j) + 1 - x_{i-k_j}, & k_j < i \le 3n, \\ & p_{i-k_j} = j, \\ & i - k_j \notin S. \end{cases}$$

With this, the input distribution $\mathcal{D}$ is fully defined, the following properties are immediate. All the m arrays $A_j, j \in [m]$ are strictly increasing arrays of length $3n$. We have $A_1[i_1] = \cdots = A_m[i_m]$ if and only if $i_j = i_1 + k_j, \forall j \in [2, m]$ and $i_1 \in S$. In particular, exactly n elements are common to all the arrays. Also, given only the arrays, for every $i_1 \in [2n]$, one can know with certainty that $i_1 \in S$ only if either one knows all the values $A_1[i_1], A_2[i_1 + k_2], \ldots, A_m[i_1 + k_m]$ or if one knows S completely. Since the total number of queries allowed is less than n, no algorithm under our consideration can determine S completely.

Let $\mathcal{A}'$ be a deterministic algorithm for Problem A.2. Let $Q_j, j \in [m]$ be the set of indices for which $\mathcal{A}'$ queries the values from A_j. Recall that the sets Q_j are fixed and do not depend on the input. Let $q = |Q_1| + \cdots + |Q_m|$.

As discussed before, if $\mathcal{A}'$ outputs a tuple $(i_1, \ldots, i_m)$, then it means (i) $i_j - i_1 = k_j, \forall j \in [2, m]$ and (ii) $i_1 \in S$. Since $\mathcal{A}'$ knows with certainty that $i_1 \in S$ (condition (ii) above), it is necessary that $\mathcal{A}'$ knows the values of $A_j[i_j], \forall j \in [m]$. Therefore, $i_j \in Q_j, \forall j \in [m]$. Combining this observation with condition (i) above, we see that the vector $k = (k_2, \ldots, k_m)$ should belong the set $D = \{(i_2 - i_1, \ldots, i_m - i_1) : i_j \in Q_j,\ \forall j \in [m]\}$. Since each element of $Q_1 \times \cdots \times Q_m$ results in at most one new element of D, we have $|D| \le |Q_1 \times \cdots \times Q_m| \le (q/m)^m$. As there are n^{m-1} choices for the vector k, the success probability of $\mathcal{A}'$ is at most $|D|/n^{m-1} \le (q/m)^m/n^{m-1}$. ■

It is not difficult to reduce the intersection-search problem on m arrays to one-sided testing for a particular pattern of length $2m+1$. We illustrate the technique by reducing the intersection-search problem on three arrays to testing for the pattern $(1, 5, 4, 2, 3)$.

Let (A_1, A_2, A_3) be an input instance of Problem A.2. That is, A_1, A_2 and A_3 are each arrays of $3n$ integers sorted in ascending order, with at least n elements in common to all three. Define $m = 15n$ and A_j^r to be the reversal of A_j (i.e., $A_j^r[i] = A_j[3n+1-i]$). We construct an injective function $f : [m] \to \mathbb{R}$ as follows.

$$\begin{aligned} f(2i-1) &= A_1^r[i] - (1/3), && i \in [3n], \\ f(2i) &= A_1^r[i] + (1/3), && i \in [3n], \\ f(6n+2i-1) &= A_2[i] + (1/4), && i \in [3n], \\ f(6n+2i) &= A_2[i] - (1/4), && i \in [3n], \\ f(12n+i) &= A_3^r[i], && i \in [3n]. \end{aligned}$$

We have designed f so that for every (i_1, i_2, i_3) that is a solution for the intersection-search problem, i.e $A_1[i_1] = A_2[i_2] = A_3[i_3]$, the 5-tuple $(2i_1'-1, 2i_1', 6n+2i_2-1, 6n+2i_2, 12n+i_3')$, where $i_1' = 3n+1-i_1$

and $i'_3 = 3n + 1 - i_3$, is a $(1, 5, 4, 2, 3)$-tuple in f. Since there are n such disjoint pairs at least, f is ϵ-far from being $(1, 3, 2)$-free, where $\epsilon = 1/15$.

Moreover, these are the only $(1, 5, 4, 2, 3)$-tuples in f. This is because (i) f is $(1, 5, 4)$-free, or equivalently $(1, 3, 2)$-free in the range $[6n]$, (ii) f is $(5, 4, 2)$-free and $(4, 2, 3)$-free in the range $[6n + 1, 12n]$, (iii) f is $(4, 2, 3)$-free in the range $[12n + 1, 15n]$, and (iv) the only $(1, 5)$-pairs of f in the range $[3n]$ and $(4, 2)$-pairs of f in the range $[6n + 1, 12n]$ are adjacent odd and even indices.

In short, whenever a tester for $(1, 5, 4, 2, 3)$-pattern finds a $(1, 5, 4, 2, 3)$-tuple in f, it produces a solution for the intersection-search problem. Hence the next result follows from Lemma A.4 with $m = 3$.