Testing for Forbidden Order Patterns in an Array

Ilan Newman[1], Yuri Rabinovich[1], Deepak Rajendraprasad[1], and Christian Sohler * [2]

[1]Department of Computer Science, University of Haifa, Israel
[2]Department of Computer Science, TU Dortmund, Germany

Abstract

In this paper, we study testing of sequence properties that are defined by forbidden order patterns. A sequence $f : \{1, \ldots, n\} \to \mathbb{R}$ of length n contains a pattern $\pi \in \mathfrak{S}_k$ ($\mathfrak{S}_k$ is the group of permutations of k elements), iff there are indices $i_1 < i_2 < \cdots < i_k$, such that $f(i_x) > f(i_y)$ whenever $\pi(x) > \pi(y)$. If f does not contain π, we say f is π-free. For example, for $\pi = (2, 1)$, the property of being π-free is equivalent to being non-decreasing, i.e. monotone. The property of being $(k, k-1, \ldots, 1)$-free is equivalent to the property of having a partition into at most $k-1$ non-decreasing subsequences.

Let $\pi \in \mathfrak{S}_k$, k constant, be a (forbidden) pattern. Assuming f is stored in an array, we consider the property testing problem of distinguishing the case that f is π-free from the case that f differs in more than ϵn places from any π-free sequence. We show the following results: There is a clear dichotomy between the monotone patterns and the non-monotone ones:

- For monotone patterns of length k, i.e., $(k, k-1, \ldots, 1)$ and $(1, 2, \ldots, k)$, we design *non-adaptive* one-sided error ϵ-tests of $(\epsilon^{-1} \log n)^{O(k^2)}$ query complexity.
- For non-monotone patterns, we show that for any size-k non-monotone π, *any non-adaptive* one-sided error ϵ-test requires at least $\Omega(\sqrt{n})$ queries. This general lower bound can be further strengthened for specific non-monotone k-length patterns to $\Omega(n^{1-2/(k+1)})$.

  On the other hand, there always exists a *non-adaptive* one-sided error ϵ-test for $\pi \in \mathfrak{S}_k$ with $O(\epsilon^{-1/k} n^{1-1/k})$ query complexity. Again, this general upper bound can be further strengthened for specific non-monotone patterns. E.g., for $\pi = (1, 3, 2)$, we describe an ϵ-test with (almost tight) query complexity of $\widetilde{O}(\sqrt{n})$.

Finally, we show that adaptivity can make a big difference in testing non-monotone patterns, and develop an *adaptive* algorithm that for any $\pi \in \mathfrak{S}_3$, tests π-freeness by making $(\epsilon^{-1} \log n)^{O(1)}$ queries.

For all algorithms presented here, the running times are linear in their query complexity.

*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 it. 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. [Erg+98] and a matching lower bound was given by Fischer [Fis04]. Later Bhattacharyya et al. [Bha+12] 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 [BÓ4] and [Kit11]). 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 [MT04] in 2004. Later on Fox [Fox13] 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 can be found at `http://math.depaul.edu/bridget/patterns.html`.

Another motivation comes from the study of patterns and motifs in time series analysis [BC94; KLC02; Pat+02], 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 a sophisticated algorithm of Guillemot and Marx [GM14], 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 the property tester has *one-sided error*. In this paper only the latter type testers will be considered.

1.1 Summary of results

We establish the following results about one-sided-error testers for the problem:

1. Every monotone pattern $\pi \in \mathfrak{S}_k$, i.e., $\pi = (1, \ldots, k)$ or $\pi = (k, \ldots, 1)$, can be *non-adaptively* ϵ-tested making $\left(\epsilon^{-1} \log n\right)^{O(k^2)}$ queries.
2. For every non-monotone pattern $\pi \in \mathfrak{S}_k$, and 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.
3. The above lower bounds are almost tight:
   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.
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 an 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., $\{(1,3,2),(3,1,2)\}$, i.e., the Gilbreath shuffling. But the lower bounds do not extend. In fact, being $\{(1,3,2),(3,1,2)\}$-free can be tested *non-adaptively* with poly-log queries.

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 [Fis04].[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 $f : [n] \mapsto \mathbb{R}$ 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 is what gives as 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 interval 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

[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 [Fis04]. All our testers are order based.

there may be at most one co-ordinate in I_R that can complete it to a $(1,3,2)$-tuple. Hence finding it 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 above forces a lot of structure on the sequence in I_L. If we allow ourselves the power adaptivity, we can use this structure using a slightly modified randomized binary search to find that unique co-ordinate 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 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 [RS96]. Works on string properties related to forbidden or occurring patterns in labeled posets include the papers on testing sortedness [Erg+98; Bha+12] and the lower bound of Fischer [Fis04], and many others, see e.g., [BEF05; LN05; Alo+99; FN01], and others.

The problem of testing hereditary properties of permutations has been studied before by Hoppen et al. [Hop+11] under the rectangular distance (discrepancy of intervals) and by Klimosova et al. [KK14] 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 [Gol+00; BBM12; Dod+99; HK08;

Fis+02; Bri+10]. Recently, Chakrabarty and Seshadhri gave an optimal tester for this problem on the hypercube and on hypergrids [CS13]. In another paper they prove that the important special case of monotonicity of Boolean functions over the n-dimensional hypercube can be tested in $o(n)$ query complexity [CS16].

Finally, we note that some progress on the exact decision problem for π-freeness for permutations was obtained in [AR08], and a more efficient $O(f(k) \cdot n)$ time algorithm was presented in [GM14] 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. In Section 4, 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 hint on how this technique can be used for deriving better lower bounds for some specific patterns in Section 4.3, but leave the details to the appendix. The poly-log adaptive tester and the optimal non-adaptive tester for the pattern $(1, 3, 2)$ are entirely in the appendix. We conclude with a few remarks and some open problems in Section 5.

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; that is, 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 exits 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.

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. Further, 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, i.e, f restricted to the set $[n]\backslash(T(1) \cup \cdots \cup T(k))$ is π-free. Since f is ϵ-far from being π-free, $|T(1) \cup \cdots \cup T(k)| \geq \epsilon n$. Therefore $|T| \geq \epsilon n/k$. □

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 2.3. *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.*

3 Monotone patterns

For every $k \in \mathbb{N}$, we call the permutations $(1, \ldots, k)$ and $(k, \ldots, 1)$ as *monotone patterns*. Since testing for the monotone increasing pattern $(1, \ldots, k)$ is the same as testing for the monotone decreasing pattern $(k, \ldots, 1)$ in the reversed sequence, we restrict our discussion to testing for $\pi_k = (k, \ldots, 1)$. 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 $k \leq m$. 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 $\{0, 1, \ldots, \lfloor \log m \rfloor - 1\}$. Slice I into consecutive disjoint intervals $I_1, \ldots, I_{\lceil m/W \rceil}$, each of length W (except possibly the last one).*
4. *Pick a "slice-number" s uniformly at random from $\{1, \ldots, \lceil m/W \rceil\}$. 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, $I_L = I_{s-2\ell+1} \cup \cdots \cup I_s$, and $I_R = I_{s+1} \cup \cdots \cup I_{s+2(k-\ell)}$.*
   In the above expressions, assume $I_i = \emptyset$ if $i \notin \{1, \ldots, \lceil m/W \rceil\}$
5. *Recursively sample $(t_1, \ldots, t_\ell)$ from I_L and $(t_{\ell+1}, \ldots, t_k)$ from I_R. That is,*

$$(t_1, \ldots, t_\ell) = \text{DyadicSampler}(I_L, l) \quad \text{and} \quad (t_{\ell+1}, \ldots, t_k) = \text{DyadicSampler}(I_R, k-l).$$

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

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

Definition 3.3. *Let $t = (t_1, \ldots, t_k)$ be a k-tuple of positive integers with $t_1 < \cdots < t_k$. We define the* leap-start *and the* leap-size *of t to be*

$$\begin{aligned} \text{leap-start}(t) &= \min\{i \in [k-1] : t_{i+1} - t_i \geq t_{j+1} - t_j, \forall j \in [k-1]\}, \qquad \textit{and} \\ \text{leap-size}(t) &= \lfloor \log(t_{i+1} - t_i) \rfloor, \ \textit{where } i = \text{leap-start}(t). \end{aligned}$$

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

$$P\left[f|_t \sim \pi_k,\ t \in T^*\right] \ = \ \sum_{\ell=1}^{k-1} \frac{1}{k-1} \sum_{w=0}^{\lfloor \log n \rfloor - 1} \frac{1}{\log n} \sum_{s=1}^{\lceil n/2^w \rceil} \frac{1}{\lceil n/2^w \rceil} P\left[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}\right] \quad (1)$$

where $E_{\ell,w,s}$ 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.

Now we estimate $P\left[f|_t \sim \pi_k,\ t \in T^* \mid E_{\ell,w,s}\right]$ for an arbitrary but fixed ℓ, w, s. Let $T_{\ell,w,s} \subset T$ be all the k-tuples t in T with leap-start$(t) = \ell$, leap-size$(t) = w$, and $t_\ell \in I_s$, where $I_s = \{(s-1)W + 1, \ldots, sW\}$. 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,

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

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 $T^*_{\ell,w,s}$. In particular, if we choose any $x \in \mathbb{R}$ and define

$$\begin{aligned} L_{\ell,w,s}(x) &= \{(t_1, \ldots, t_\ell) : (t_1, \ldots, t_k) \in T_{\ell,w,s}, t_\ell \geq x\}, \text{ and} \\ R_{\ell,w,s}(x) &= \{(t_{\ell+1}, \ldots, t_k) : (t_1, \ldots, t_k) \in T_{\ell,w,s}, t_\ell \leq x\}, \end{aligned}$$

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 $T^*_{l,w,s}$. Hence the following claim is true.

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

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

We will choose x to be the minimum 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

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

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

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,

$$\begin{aligned} P\left[f|_t \sim \pi_k,\ t \in T^*\right] &\geq \frac{(2k^2 \log n)^{-\binom{k}{2}+1}}{(2k)^k} \sum_{l=1}^{k-1} \frac{1}{k-1} \sum_{w=0}^{\log n - 1} \frac{1}{\log n} \sum_{s=1}^{n/W} \frac{1}{n/W} \left(\frac{|T_{l,w,s}|}{W}\right)^k \\ &\geq \frac{(2k^2 \log n)^{-\binom{k}{2}+1}}{(2nk^2 \log n)^k} |T|^k, \end{aligned} \tag{3}$$

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. Theorem 3.4 gives us a monotonicity tester with query complexity $O(\epsilon^{-2} \log^2 n)$. However, as far as we know, none of those 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 the previous proof. It will be used again in the adaptive tester for the pattern $(1,3,2)$ which we describe in the appendix. 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 the appendix (Definition A.1, Claim A.2).

4 Lower bounds for non-adaptive testers

4.1 The pattern $(1,3,2)$

In this section we prove

Theorem 4.1. *Any one-sided-error non-adaptive ϵ-tester for the pattern $(1,3,2)$ has query complexity $\Omega(\sqrt{n})$, for every $\epsilon \leq 1/9$.*

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 4.2 (Intersection-search)**.** *The input to the problem consists of m arrays $A_j, j \in [m]$, each containing $3n$ distinct integers in an 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 $(i_1, \ldots, i_m) \in [3n]^m$ 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 2.3, 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 4.3. *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 $(i_1, \ldots, i_m)$, then surely $A_1[i_1] = \cdots = A_m[i_m]$. That is, for every possible input $(A_1, \ldots, A_m)$ 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 $\sum_{j=1}^{m} |Q_j|$.

Next we analyze intersection-search problem with two arrays ($m = 2$).

Lemma 4.4. *Let $\mathcal{A}$ be an algorithm in $\mathcal{C}$ for the intersection-search on two arrays. If $\mathcal{A}$ makes q queries and $q < n/2$, then the success probability of $\mathcal{A}$ is at most $q^2/(4n)$.*

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 4.2 has a probability of success at most $q^2/(4n)$, when the input is sampled according to $\mathcal{D}$. Recall that the deterministic algorithms we need to consider are those which outputs (i_1, i_2) only when it is sure that $A_1[i_1] = A_2[i_2]$ and outputs "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 pair.

We define $\mathcal{D}$ by prescribing a randomized procedure to select two monotone increasing arrays A_1 and A_2 of length $3n$ with n common elements. The randomness is three-fold; (i) we pick 0-1 vector $x = (x_1, \ldots, x_{3n})$ of length $3n$ uniformly at random, (ii) independently pick a set $S \subset [2n]$ of size n uniformly at random, and (iii) independently pick a $k \in [n]$ uniformly at random. The first two types of randomness will be used to ensure that any deterministic algorithm that returns a pair (i_1, i_2) can do so only if it "hits" that pair, i.e., the algorithm indeed sees $A_1[i_1]$ and $A_2[i_2]$. The third randomness makes such hits unlikely with $O(\sqrt{n})$ queries.

The two input arrays A_1 and A_2 are defined as follows:

$$A_1[i] = 2i + x_i, \qquad 1 \le i \le 3n, \text{ and} \tag{4}$$

$$A_2[i] = \begin{cases} 2(i-k), & 1 \le i \le k, \\ 2(i-k) + x_{i-k}, & k < i \le 3n, i-k \in S, \\ 2(i-k) + 1 - x_{i-k}, & k < i \le 3n, i-k \notin S. \end{cases} \tag{5}$$

With this, the input distribution $\mathcal{D}$ is fully defined, and the following properties are immediate. Both A_1 and A_2 are strictly increasing arrays of length $3n$. We have $A_1[i_1] = A_2[i_2]$ if and only if $i_2 = i_1 + k$ and $i_1 \in S$. In particular, A_1 and A_2 have exactly n common elements. Also, given only A_1 and A_2, for every $i \in [2n]$, one can know with certainty whether $i \in S$ only if either one knows the values of both $A_1[i]$ and $A_2[i+k]$ or if one knows S completely. Since the total number of queries allowed is less than $n/2$, no algorithm under our consideration can determine S completely.

Let $\mathcal{A}'$ be a deterministic algorithm for Problem 4.2. Let Q_1 and Q_2 be the set of indices for which $\mathcal{A}'$ queries the values from A_1 and A_2, respectively. Recall that Q_1 and Q_2 are fixed and do not depend on the input. Let $q = |Q_1| + |Q_2|$.

As discussed before, if $\mathcal{A}'$ outputs (i_1, i_2), then it means (i) $i_2 - i_1 = k$ and (ii) $i_1 \in S$. Since $\mathcal{A}'$ knows with certainty that $i_1 \in S$, it is necessary that $\mathcal{A}'$ knows the values of $A_1[i_1]$ and $A_2[i_2]$ and therefore, $i_1 \in Q_1$ and $i_2 \in Q_2$. Let $D = \{i_2 - i_1 : i_2 \in Q_2, i_1 \in Q_1\}$. Then we see that for $\mathcal{A}'$ to be successful, we need k to be in D. But $|D| \leq q^2/4$ and since k is chosen uniformly at random from $[n]$, the success probability of $\mathcal{A}'$ is at most $q^2/(4n)$. □

Next we show the reduction from Problem 4.2 to the problem of finding a $(1, 3, 2)$-tuple.

Let (A, B) be an input instance of Problem 4.2 on two arrays. That is, A and B are strictly increasing arrays of length $3n$ each with at least n elements in common. Define $m = 9n$ and A^r to be the reversal of A (i.e., $A^r[i] = A[3n + 1 - i]$). We construct an injective function $f : [m] \to \mathbb{R}$ as follows.

$$\begin{array}{lcll} f(2i-1) & = & A^r[i] - \frac{1}{4}, & i \in [3n] \\ f(2i) & = & A^r[i] + \frac{1}{4}, & i \in [3n] \\ f(6n+i) & = & B[i], & i \in [3n]. \end{array}$$

Notice that f is designed so that for every (i, j) that is a solution for the intersection-search problem, i.e $A[i] = B[j] = x$, the 3-tuple $(2i' - 1, 2i', 6n + j)$, $i' = 3n + 1 - i$, is a $(1, 3, 2)$-tuple in f with values $(x - \frac{1}{4}, x + \frac{1}{4}, x)$. Since there are n such disjoint pairs at least, f is ϵ-far from being $(1, 3, 2)$-free, where $\epsilon = 1/9$.

Moreover, these are the only $(1, 3, 2)$-tuples in f. This is since f is monotone increasing in the range $[6n + 1, 9n]$ and hence at most one (the third) coordinate of a $(1, 3, 2)$ can be in this range. Hence the first two coordinates of any $(1, 3, 2)$-tuple in f are in the range $[6n]$. The third coordinate must be in $[6n+1, 9n]$ (since f restricted to $[6n]$ is $(1, 3, 2)$-free). Further, the only $(1, 2)$ patterns in the range $[6n]$ of f are of the form $(2i-1, 2i)$. Hence, each $(1, 3, 2)$-pattern corresponds to $(i, j) \in [3n]^2$ such that$A[i] = B[j]$.

In short, whenever a tester for $(1, 3, 2)$-pattern finds a $(1, 3, 2)$-tuple in f, it produces a solution pair for the intersection-search problem. Theorem 4.1 now follows directly from Lemma 4.4.

4.2 General non-monotone patterns

In this section, we prove we reduce the problem of $(1, 3, 2)$-freeness to the problem of testing for any non-monotone pattern.

Theorem 4.5. *Let π be a non-monotone pattern in $\mathfrak{S}_k$, $k \geq 4$. Any one-sided-error non-adaptive ϵ-tester for π has query complexity $\Omega(\sqrt{n})$, for every $\epsilon \leq \frac{1}{9(k-2)}$.*

Let us fix an arbitrary non-monotone pattern $\pi \in \mathfrak{S}_k$ for some $k \geq 4$. First, we observe that any non-monotone pattern $\pi \in \mathfrak{S}_k$ contains a 3-length non-monotone pattern with adjacent values.

Proposition 4.6. *Let $k \geq 4$, and $\pi \in \mathfrak{S}_k$ a non-monotone pattern. Then there exists $i \in [k - 2]$ such that $(\pi^{-1}(i), \pi^{-1}(i + 1), \pi^{-1}(i + 2))$ is non-monotone.*

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

Let i be the smallest value so that $(\pi^{-1}(i), \pi^{-1}(i + 1), \pi^{-1}(i + 2))$ 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 and $M = \max_i |f(i)| + k$. We construct a function $f_\pi : [m] \to \mathbb{R}$, $m = (k - 2)n + (k - 3)$ by "interleaving" f with $(\pi' \pm M)$ as defined below:

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

$$f_\pi(s(k-2)) = f(s), \quad \text{if } s \neq 0,$$
$$f_\pi(s(k-2)+t) = \begin{cases} +M + \pi'(t), & \text{if } \pi'(t) \geq i \\ -M + \pi'(t), & \text{if } \pi'(t) < i. \end{cases}$$

One can verify that if (s_1, s_2, s_3) is a $\hat{\pi}$-tuple in f, then f_π contains π-tuple 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 $(p_1, \ldots, p_k)$ be a π-tuple in f_π. Let $\hat{P}$ be the set of indices in $(p_1, \ldots, p_k)$ corresponding to $\hat{I}$ in π (in the order-isomorphism). Observe that, $\forall p \in \hat{P}$, there are at least $(i-1)$ indices in $[m]$ (in fact, in P) that are strictly smaller in f-value than $f(p)$ and at least $(k-i-3)$ indices in $[m]$ that are strictly larger in f-value than $f(p)$. This implies that $f(p) \in [-M, M]$. Thus, $p = 0 \mod (k-2)$, $\forall p_i \in \hat{P}$. Therefore, $\hat{P}$ (after scaling down by $(k-2)$) corresponds 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 4.5 follows from Theorem 4.1.

4.3 More complex non-monotone patterns

In this section, we generalize the technique in Section 4.1 to obtain better lower bounds for specific more complex patterns. First, Lemma 4.4 is extended to m arrays

Lemma 4.7. *Let $\mathcal{A}$ be an algorithm in class $\mathcal{C}$ for Problem 4.2 on m arrays. If $\mathcal{A}$ makes $q < n/2$, then the success probability of $\mathcal{A}$ is at most $(q/m)^m/n^{m-1}$.*

Then, this "intersection search" problem is reduced to testing the pattern $(1, 2m-1, 2m-2, 2, 3, 2m-3, .., m)$. All the technical details appear in the Appendix.

5 Open problems

1. Could it be that for *any* constant-size pattern π there there exits a poly-logarithmic adaptive tester for π-freeness?
2. How does the structure of a pattern π correlates with the complexity of an optimal non-adaptive tester for π-freeness? In particular, do there exit infinitely many non-monotone π's with query complexity $O(n^{0.99})$?
3. Can any hereditary property of sequences be tested with sub-linear query complexity?

References

[Alo+99] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. "Regular Languages Are Testable with a Constant Number of Queries". In: *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. 1999, pp. 645–655.

[AR08] S. Ahal and Y. Rabinovich. "On complexity of the subpattern problem". In: *SIAM Journal on Discrete Mathematics* 22.2 (2008), pp. 629–649.

[B04] M. Bóna. *Combinatorics of permutations*. Discrete mathematics and its applications. Boca Raton, FL, London: Chapman & Hall/CRC Press, 2004. ISBN: 1-584-88434-7.

[BBM12] E. Blais, J. Brody, and K. Matulef. "Property Testing Lower Bounds via Communication Complexity". In: *Computational Complexity* 21.2 (2012), pp. 311–358.

[BC94] D. J. Berndt and J. Clifford. "Using Dynamic Time Warping to Find Patterns in Time Series". In: *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, July 1994. Technical Report WS-94-03*. 1994, pp. 359–370.

[BEF05] P. Berenbrink, F. Ergün, and T. Friedetzky. "Finding Frequent Patterns in a String in Sublinear Time". In: *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*. 2005, pp. 746–757.

[Bha+12] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. P. Woodruff. "Transitive-Closure Spanners". In: *SIAM J. Comput.* 41.6 (2012), pp. 1380–1425.

[BOH08] M. Ben Or and A. Hassidim. "The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well)". In: *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*. IEEE. 2008, pp. 221–230.

[Bri+10] J. Briët, S. Chakraborty, D. García-Soriano, and A. Matsliah. "Monotonicity Testing and Shortest-Path Routing on the Cube". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*. 2010, pp. 462–475.

[CS13] D. Chakrabarty and C. Seshadhri. "Optimal bounds for monotonicity and lipschitz testing over hypercubes and hypergrids". In: *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. 2013, pp. 419–428.

[CS16] D. Chakrabarty and C. Seshadhri. "An o(n) Monotonicity Tester for Boolean Functions over the Hypercube". In: *SIAM J. Comput.* 45.2 (2016), pp. 461–472.

[Dod+99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. "Improved Testing Algorithms for Monotonicity". In: *Randomization, Approximation, and Combinatorial Algorithms and Techniques, Third International Workshop on Randomization and Approximation Techniques in Computer Science, and Second International Workshop on Approximation Algorithms for Combinatorial Optimization Problems RANDOM-APPROX'99, Berkeley, CA, USA, August 8-11, 1999, Proceedings*. 1999, pp. 97–108.

[Erg+98] F. Ergün, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. "Spot-Checkers". In: *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*. 1998, pp. 259–268.

[Fis+02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. "Monotonicity testing over general poset domains". In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. 2002, pp. 474–483.

[Fis04] E. Fischer. "On the strength of comparisons in property testing". In: *Inf. Comput.* 189.1 (2004), pp. 107–116.

[FN01] E. Fischer and I. Newman. "Testing of matrix properties". In: *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*. 2001, pp. 286–295.

[Fox13] J. Fox. "Stanley-Wilf limits are typically exponential". In: *CoRR* abs/1310.8378 (2013).

[GM14] S. Guillemot and D. Marx. "Finding small patterns in permutations in linear time". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2014, pp. 82–101.

[Gol+00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. "Testing Monotonicity". In: *Combinatorica* 20.3 (2000), pp. 301–337.

[HK08] S. Halevy and E. Kushilevitz. "Testing monotonicity over graph products". In: *Random Struct. Algorithms* 33.1 (2008), pp. 44–67.

[Hop+11] C. Hoppen, Y. Kohayakawa, C. G. T. de A. Moreira, and R. M. Sampaio. "Testing permutation properties through subpermutations". In: *Theor. Comput. Sci.* 412.29 (2011), pp. 3555–3567.

[Kit11] S. Kitaev. *Patterns in Permutations and Words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011.

[KK07] R. M. Karp and R. Kleinberg. "Noisy binary search and its applications". In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 881–890.

[KK14] T. Klimosova and D. Král'. "Hereditary properties of permutations are strongly testable". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*. 2014, pp. 1164–1173.

[KLC02] E. J. Keogh, S. Lonardi, and B. Y. 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, July 23-26, 2002, Edmonton, Alberta, Canada*. 2002, pp. 550–556. DOI: 10.1145/775047.775128.

[LN05] O. Lachish and I. Newman. "Testing Periodicity". In: *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*. 2005, pp. 366–377.

[MT04] A. Marcus and G. Tardos. "Excluded permutation matrices and the Stanley–Wilf conjecture". In: *Journal of Combinatorial Theory, Series A* 107.1 (2004), pp. 153–160.

[Pat+02] P. Patel, E. J. Keogh, J. Lin, and S. Lonardi. "Mining Motifs in Massive Time Series Databases". In: *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. 2002, pp. 370–377. DOI: 10.1109/ICDM.2002.1183925.

[RS96] R. Rubinfeld and M. Sudan. "Robust Characterizations of Polynomials with Applications to Program Testing". In: *SIAM J. Comput.* 25.2 (1996), pp. 252–271.

[Wik16] Wikipedia. *Vitali covering lemma — Wikipedia, The Free Encyclopedia*. [Online; accessed 14-April-2016]. 2016.

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

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 $\mathfrak{S}_3$.

We have shown that, 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 (Section 4.1). Nevertheless we describe an adaptive one-sided-error tester for the $(1,3,2)$-pattern which makes only poly-log many queries (Section A.1). 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 A.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 A.1. *Given a function $f : [n] \to \mathbb{R}$, the* f-interval *of an ordered pair $(i,j) \in [n]^2$ is $\big(f(i), f(j)\big)$, if $f(i) \le f(j)$, and $\big(f(j), f(i)\big)$, otherwise. 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 a tuple (i,j) dominates a set T of tuples (and denote it by $(i,j) \succ_f T$), if it dominates* at least one *tuple in T.*

Claim A.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 joint probability that t is $(2,1)$-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 $L_{\ell,w,s}(x) = \{t_1 : (t_1,t_2) \in T_{\ell,w,s},\ f(t_1) \ge x\}$ and $R_{\ell,w,s}(x) = \{t_2 : (t_1,t_2) \in T_{\ell,w,s},\ f(t_1) \le x\}$.

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 $f(t_1) \ge x \ge f(t_2')$ (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 $P\left[f|_t \sim \pi_k,\ t \in T^*,\ t \succ_f T \mid E_{\ell,w,s}\right]$. □

A.1 An adaptive $(1,3,2)$-tester

The goal of this section is to prove

Theorem A.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 $|T| \ge \epsilon n/3$. The set T is partitioned into two types of

tuples based on their leap-start (Definition 3.3): $T_1 = \{(i,j,k) \in T |\ j - i \geq k - j\}$ 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 A.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

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

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 A.4, we see that it finds a $(1,3,2)$-tuple in f, and therefore rejects f, with probability close to 1 when $|T_1| \geq \epsilon n/6$.

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

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, we show in the next section that this limitation is grave enough to rule out poly-log non-adaptive testers for the pattern $(1,3,2)$. Recall however, that our current goal is only to design an adaptive tester.

To explain the intuition behind the adaptive tester (Algorithm A.8 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 which dominates a $(1,2)$-pair in $\{(t_1,t_2) : (t_1,t_2,t_3) \in T\}$ (Claim A.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).

Remark. We use the domination property of the dyadic sampler (Claim A.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.

We first describe the version of random binary search that we use and state its performance for the search problem we have in our hand. The proof of it is deferred to the next section.

Problem A.5 (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 $\mathbb{R}$; and a "query-range" $Q \subset F$ which is also an open interval in $\mathbb{R}$. 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 for some positive constant ϵ, (ii) $f|_A$ contains a monotone increasing sequence of length at least $(1-\log^{-5} m)|A|$, and (iii) there exists an index i in the above monotone sequence such that $f(i) \in Q$. The goal is to find an index $i \in I$ such that $f(i) \in Q$ using minimum number of queries to f.*

Algorithm A.6 (FilteredBinarySearch(f, F, Q))**.** *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 $\mathbb{R}$; and a "query-range" $Q \subset F$ which is also an open interval in $\mathbb{R}$. 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 $I \neq \emptyset$:

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 $f(i) > \sup(Q)$ or $f(i) < \inf(Q)$, respectively.*

Theorem A.7. *Let $f : [m] \to \mathbb{R}$ and $F \subset \mathbb{R}$ be such that $A = f^{-1}(F)$ has size ϵm and $f|_A$ is $(1/\log^5 m)$-close to monotone increasing. Then there exists a set $A' \subset A$ with $|A'| \geq |A|(1 - 1/\log m)$, so that if $f^{-1}(Q)$ contains any element of A', Algorithm A.6 succeeds with probability at least $(1 - 1/\log m)$ after at most $O\left(\epsilon^{-1} \log^4 m\right)$ queries to f.*

We defer the proof of Theorem A.7 to the next section and move on to describing Test 2 instead.

Algorithm A.8 ($\mathcal{A}_2(f,\epsilon)$)**.** *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 $q = 100\epsilon^{-2} \log^{20} n$.*
   (The total number of queries made to f will be limited to $O(q)$.)

[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 A.8 for Case 1.

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 I 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 one 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) *(Algorithm A.6) allowing it to make a maximum of q queries to f. 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 A.9. *Let $f : [n] \to \mathbb{R}$ contain a matching T_2 of ϵn $(1,3,2)$-tuples with leap-start 2. Then Algorithm A.8 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 $O\left(\epsilon^{-2} \log^{20} n\right)$ queries to f.*

Proof. The claim on query complexity is obvious. We only need to analyze the probability of success. Next, we define success for each step of Algorithm A.8 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 $w \in \{0, \lceil \log n \rceil - 1\}$ 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 $p_1 \geq 1/\log n$.

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-type argument over s will show 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, when Step 2 is successful, Step 3 succeeds with probability $p_2 \geq \left(\frac{1}{2}\epsilon \log^{-1} n\right)$.

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 A.8 in Step 3. In particular, $|I_L| \leq 3W$ and $|I_R| \leq 2W$. Notice that every tuple $(i,j,k) \in T_{2,w,s}$ satisfy $i, j \in I_L$ (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 $p_3 = 1 - o(1)$.

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 $(\log^{-5} n)$-far from monotone. Otherwise, we rely on the next two steps.

If $f|_{I'_R}$ is $(\log^{-5} n)$-far from monotone, then f contains a matching M of $(2,1)$-pairs from $I'_R \times I'_R$ with $|M| \geq \frac{1}{2}(\log^{-5} n)|I'_R|$. 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 $|M| \geq \frac{1}{8}\left(\epsilon \log^{-6} n\right) W$. So a single call to the dyadic sampler returns a $(2,1)$-pair from $I'_R \times I'_R$ with probability at least $\Omega(\epsilon^2 \log^{-14} n)$ (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 are successful and $f|_{I'_R}$ is $(\log^{-5} n)$-close to being monotone. Let $S = \{(i,j) : (i,j,k) \in T, f(i) > f(i_0)\}$, which is matching of $(1,2)$-pairs. Step 6 is successful if it returns a $(1,2)$-pair (i,j) which dominates S

Since Step 4 is successful, $f(i_0) < m_T$ and thus $|S| \geq \frac{1}{2}|T|$. By Lemma A.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 there exists at least one index $k \in I'_R$ such that $f(k) \in \big(f(i), f(j)\big)$. The last step is successful if it finds such an index $k \in I'_R$.

We use Theorem A.7 here. The set I'_R plays the role of A in the theorem. Since Step 4 is successful, We know that $|A| = |I'_R| = \delta |I_R|$ where $\delta \geq \frac{1}{8}\left(\epsilon \log^{-1} n\right)$. Moreover, we are already under the assumption that f restricted to A is $(1/\log^5 n)$-close to monotone increasing. Hence the theorem guarantees the existence of a set $A' \subset A$ with size $(1 - o(1))|A|$ of "quickly searchable" indices. We condition on the assumption that the index $k \in A$ that we are searching for, belongs to A'. This happens with probability $1 - o(1)$. Indeed, one can define $S' \subset S$ as $S' = \{(i,j) : (i,j,k) \in T, f(i) > f(i_0), k \in A'\}$ and demand, in the previous step, that the dyadic sampler on I_L returns a $(1,2)$-pair (i,j) which dominates S'. The estimate on probability follows since $|S'| \geq (1 - o(1))|S|$. Once $k \in A'$, Algorithm A.6 succeeds within $O(\delta^{-1} \log^4 n) \ll q$ queries, with probability $1 - o(1)$. Thus the whole algorithm succeeds with probability $\Omega(\epsilon^3 \log^{-6} n)$. □

Repeating Algorithm A.8 $O(\epsilon^{-3} \log^6 n)$ times returns a $(1,3,2)$-tuple in f with probability close to 1 when $|T_2| \geq \epsilon n/6$. Thus, Theorem A.3 follows from Lemmas A.4 and A.9.

A.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 adaptive tester 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 $T(3) \subset I_R$. 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 $\Omega(\epsilon^2 \log^{-2} n)$, 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 A.2). If we repeat this dyadic sampling $\Omega(\epsilon^{-1}\sqrt{n} \log^2 n)$ times, then with high probability, we get a collection D of $\Omega(\epsilon\sqrt{n})$ 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 $|K| \geq |D| = \Omega(\epsilon\sqrt{n})$. Now

any collection of $\Omega(\epsilon^{-1}\sqrt{n})$ uniform samples from I_R hits a member of K with high probability. Hence if we sample $\Omega(\epsilon^{-1}\sqrt{n}\log^2 n)$ pairs from I_L using the dyadic sampler and an equal number of uniform point-samples from I_R, we hit a $(1,3,2)$-pair 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.

A.3 Proof of Theorem A.7

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 literature, mainly to accommodate noisy answers of different types, see [KK07; BOH08]. 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 will not necessarily be monotone, however, there will be a filter range search $F \subseteq \mathbb{R}$, so that $I(F) = f^{-1}(F)$ is relative large and $f|_{f^{-1}(F)}$ is very close to monotone. Moreover, F is a priori known to the search algorithm. That is, there is 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. 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 f is very non-monotone, 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 A.10. *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 a 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.

Lemma A.11. *Let $S \subset [n]$ with $|S| \geq \delta n$. For every $\gamma < 1$, at most $3\gamma(1-\delta)(1-\gamma)^{-1}n$ 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 [Wik16].

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 the 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 $|S_\gamma \setminus P|$. 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 $|S_\gamma| \leq \sum_{I \in \mathcal{P}} 3\mu(I) = 3|S \cap P|$. $\square$

Remark. Lemma A.11 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 $\delta = 1 - 1/\log^5 n$ and $\gamma = 1 - 1/\log^3 n$, then at most $3n/\log^2 n$ 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.

Now we analyze Algorithm A.6, which 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.

Proof of Theorem A.7. Our first application of Lemma A.11 is to $A \subset [m]$ with γ_1 set to $\epsilon/\log^2 m$. By the lemma, the set $A_{\gamma_1} \subset A$ of γ_1-deserted elements has size at most $3\gamma_1 m \leq 3|A|/\log^2 m$. Since $f|_A$ is $(1/\log^5 m)$-close to monotone, there exists a monotone nondecreasing sequence of length $\delta|A|$ in A, where $\delta = 1 - 1/\log^5 m$. Let $M \subset A$ be the support of this large monotone sequence. A second application of Lemma A.11 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 $3|A|/\log^2 m$. The set A' in the statement of the theorem is $M \setminus (A_{\gamma_1} \cup M_{\gamma_2})$. It is clear that $|A'| \gg |A|(1 - 1/\log m)$.

Let $i \in A'$ be such that $i \in f^{-1}(Q)$. When we run Algorithm A.6, 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 $k \le 100\log m$ is, by union bound, at most $(100\log m)(1/\log^3 m + 1/\log^3 m) \le 1/(2\log m)$. 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 $(1 - 1/\log m)$.

The total number of queries made to f in this case is at most $O(\gamma_1^{-1}\log m \log\log m)$ since A_k happens for all $k \le 100\log m$. □

B Improving the lower bounds for more complex patterns

In this section, we generalize the technique in Section 4.1 to obtain better lower bounds for more complex patterns. The main task is to analyze the "intersection search" problem with more arrays. We prove the Lemma 4.7 stated in Section 4.3.

Lemma 4.7. *Let $\mathcal{A}$ be an algorithm in class $\mathcal{C}$ for Problem 4.2 on m arrays. If $\mathcal{A}$ makes $q < n/2$, 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 4.2 has a probability of success at most $q^2/(8n)$, when the inputs are sampled according to $\mathcal{D}$. Recall that the deterministic algorithms we need to consider are those which outputs $(i_1, \ldots, i_m)$ only when it is sure that $A_1[i_1] = \cdots = A_m[i_m]$ and outputs "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 $(i_1, \ldots, i_m)$.

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

The arrays $A_1, \ldots, A_k$ are defined as follows:

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

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

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

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 $i \in S$. In particular, exactly n elements are common to all the arrays. Also, given only the arrays, for every $i \in [2n]$, one can know with certainty whether $i \in S$ only if either one knows all the values $A_1[i], A_2[i + k_2], \ldots, A_m[i + k_m]$ 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 4.2. Let $Q_j, i \in [m]$ 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 $i_1 \in S$, it is necessary that $\mathcal{A}'$ knows the values of $A_j[i_j], \forall j \in [m]$. Therefore, $i_j \in Q_j, \forall j \in [m]$ and hence $k_j \in D_j = \{i_j - i_1 : i_j \in Q_j, i_1 \in Q_1\}, \forall j \in [2, m]$. Since $|Q_1 \times \cdots \times Q_m| \leq (q/m)^m$ and there are n^{m-1} choices for $(k_2, \ldots, k_m)$ the success probability of $\mathcal{A}'$ is at most $(q/m)^m/n^{m-1}$. □

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 4.2. That is, A_1, A_2 and A_3 are each arrays of $3n$ integers sorted ascendingly, 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 4.7 with $m = 3$.

Theorem B.1. *Any one-sided-error non-adaptive ϵ-tester for the pattern $(1, 5, 4, 2, 3)$ has query complexity $\Omega(n^{2/3})$, for every $\epsilon \leq 1/15$.*

We end this section with the remark that the above can be generalized via the intersection-search problem on m arrays to prove that any 1-sided non-adaptive ϵ-tester for the pattern $(1, 2m - 1, 2m - 2, 2, 3, 2m - 3, .., m)$ requires $\Omega(n^{1-1/m})$ queries for every $\epsilon \leq 1/(6m - 3)$.