Electronic Colloquium on Computational Complexity, Report No. 87 (2012)

The Deterministic and Randomized Query Complexity of a Simple Guessing Game

Peyman Afshani[1], Manindra Agrawal[2], Benjamin Doerr[3], Kasper Green Larsen[1], Kurt Mehlhorn[3], Carola Winzen[3]

[1]MADALGO, Department of Computer Science, Aarhus University, Denmark
[2]Indian Institute of Technology Kanpur, India
[3]Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

We study the LeadingOnes game, a Mastermind-type guessing game first regarded as a test case in the complexity theory of randomized search heuristics. The first player, Carole, secretly chooses a string $z \in \{0,1\}^n$ and a permutation π of $[n]$. The goal of the second player, Paul, is to identify the secret (z,π) with a small number of queries. A query is a string $x \in \{0,1\}^n$, and the score of x is

$$f_{z,\pi}(x) := \max\{i \in [0..n] \mid \forall j \le i : z_{\pi(j)} = x_{\pi(j)}\},$$

the length of the longest common prefix of x and z with respect to π. We are interested in the number of queries needed by Paul to identify the secret.

By using a relatively straightforward strategy, Paul can identify the secret with $O(n \log n)$ queries and recently only a modest improvement of this to $O(n \log n / \log\log n)$ was available [DW12].

In this paper, we completely resolve the problem by offering the following results. We show that when limited to deterministic strategies, $O(n \log n)$ queries is the best possible. On the other hand, by using randomization Paul can find the secret code with an expected number of $O(n \log\log n)$ queries, which we prove is optimal by matching it with a lower bound of the same asymptotic magnitude. Finally, we prove that a number of problems that are naturally related to our problem (such as deciding whether a sequence of queries and scores is consistent) can be solved in polynomial time.

Keywords: Query complexity; randomized algorithms; guessing games; Mastermind.

1 Introduction

Guessing games and query complexity have a long history in theoretical computer science due to a wide range of applications. Originating from recent attempts to understand the difficulty of problems for randomized search heuristics, we analyze the following LeadingOnes-game, which distantly resembles the 2-color variant of the well-known Mastermind game. In our game, the first player Carole (also called Codemaker) secretly chooses a string $z \in \{0,1\}^n$ and a permutation π of $[n]$. The goal of the second player, called Paul or Codebreaker, is to identify the secret code (z,π) with a small number of queries. A query simply is a string $x \in \{0,1\}^n$. Carole's answer to a query x is its score

$$f_{z,\pi}(x) := \max\{i \in [0..n] \mid \forall j \le i : z_{\pi(j)} = x_{\pi(j)}\},$$

the length of the longest common prefix of x and z with respect to π.

ISSN 1433-8092

It is easy to see that Paul can find the secret using $O(n \log n)$ queries by a binary search strategy. He starts by querying the all-zeros string $x^0 := (0, \dots, 0)$ and the all-ones string $x^1 := (1, \dots, 1, 1)$. The scores determine $z_{\pi(1)}$. By flipping a set I of bits in the better of the two strings, Paul can determine whether $\pi(1) \in I$ or not. This allows to find $\pi(1)$ via a binary search strategy in $O(\log n)$ queries. Once $\pi(1)$ and $z_{\pi(1)}$ are known, Paul may iterate this strategy on the remaining bit positions to determine $\pi(2)$ and $z_{\pi(2)}$, and so on. This yields an $O(n \log n)$ query strategy for identifying the secret code. An upper bound of $O(n \log n / \log \log n)$ for a randomized strategy was recently shown in [DW12]. In this work, we greatly expand our understanding of the LeadingOnes game.

(1) We first show that every *deterministic* strategy for Paul requires $\Omega(n \log n)$ queries in the worst case (see Section 3). Hence the binary search approach is asymptotically optimal among deterministic strategies.

(2) We then exhibit a *randomized strategy* for Paul that finds the secret with an expected number of $O(n \log \log n)$ queries (Section 4), i.e., on average, we determine a pair $(z_{\pi(i)}, \pi(i))$ in $O(\log \log n)$ queries. Putting this into an information-theoretic perspective, this results means that we succeed in learning an average of $O(\log n / \log \log n)$ bits of information in each query.

(3) The above bound is asymptotically optimal, as shown in Section 5. In contrast, for most guessing games (Mastermind for constant number of colors, coin weighting games, many liar games), the asymptotic query complexity equals the information-theoretic lower bound.

(4) Finally, we show that the decision problem of whether a set of scores is consistent, the search problem of finding a secret that is consistent with a set of queries and scores, and the counting problem of determining the number of secrets consistent with a search history, all can be solved in polynomial time. This is again different from Mastermind, where corresponding problems are either NP-complete or #P-complete.

Relation to Other Work: Our interest for the LeadingOnes problem stems from the *theory of randomized search heuristics*. While problem-independent randomized search heuristics such as evolutionary algorithms or ant colony optimization proved to be highly successful in efficiently solving difficult problems, our understanding of these methods still is limited. The application of techniques from the algorithms and complexity area in the last 15 years has resulted in some progress. On the algorithms analysis side, this lead to several proven bounds on the run-time of several randomized search heuristics (see [AD11]). On the complexity side, Droste, Jansen, and Wegener [DJW06] suggested query complexity, i.e., the minimum number of search points to be evaluated to find the optimum, as a natural lower bound.

One of the benchmark problems often regarded in the field of evolutionary computation is the LeadingOnes test function

$$\textsc{Lo} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$$

assigning to every bit string the length of the longest prefix with all entries equal to one. The set of functions $f_{(z,\pi)}$ generalize Lo to a class of instances invariant under automorphisms of the hypercube. Our result that the query complexity of the LeadingOnes function class is $\Theta(n \log \log n)$ in particular implies that no general-purpose randomized search heuristics can solve this problem evaluating less than $\Theta(n \log \log n)$ search points. Note that all classic search heuristics need a quadratic number of search point evaluations for the LeadingOnes problem.

The archetypal guessing game actually being played is *Mastermind* (note, though, that a number of applications have been found, e.g., in the context of comparing DNA-sequences [Goo09a] and API-level attacks on user PIN data [FL10]). In the original Mastermind game, Carole chooses a secret code $z \in [k]^n$. She returns to each of Paul's queries $x \in [k]^n$ the number $\text{eq}(z, x)$ of positions in which x and z agree and the number $w(z, x)$ of additional colors in x that appear in z (formally, $w(z,x) := \max_{\pi \in S_n} |\{i \in [n] \mid z_i = x_{\pi(i)}\}| - \mathrm{eq}(z,x)$). On the board, $\mathrm{eq}(z,x)$ is typically indicated by black answer-pegs, and $w(z,x)$ is usually indicated by white answer-pegs. Paul's task is to identify z with as few queries as possible.

Mastermind has been studied intensively since the sixties [ER63, Knu77, Chv83, CCH96, Goo09b, Vig12] and thus even before it was invented as a board game. In particular, [ER63, Chv83] show that for all n and $k \leq n^{1-\varepsilon}$, Paul can find the secret code by simply guessing $\Theta(n \log k / \log n)$ random queries. This can be turned into a deterministic strategy having the same asymptotic complexity. The information-theoretic lower bound of $\Omega(n \log k / \log n)$ shows that this is best possible, and also, that there is no difference between the randomized and deterministic case. Similar situations have been observed for a number of guessing, liar, and pusher-chooser games (see, e.g., [Pel02, Spe94]). Our results show that things are different for the LeadingOnes game.

The same is true for the hardness of finding or counting solutions consistent with previous queries and scores. For Mastermind with suitably many colors and black and white answer-pegs, Stuckman and Zhang showed that is is NP-hard to decide whether or not a Mastermind guessing history is feasible, cf. [SZ06]. Goodrich [Goo09b] showed a corresponding result for the game with black answer-pegs only. Most recently, Viglietta has shown that both hardness results apply also to the setting with only two colors [Vig12]. He also shows that computing the number of secrets that are consistent with a given Mastermind guessing history is #P-complete. In contrast, for the LeadingOnes-game both problems can be solved efficiently (Section 6).

Other problems related to the LeadingOnes-game include coin weighing, vector reconstruction, and graph reconstruction games, as well as uniquely decodable codes for noiseless n-user adder channels. In coin weighing, one is given a set of coins and a (beam or spring) balance. The goal is to identify the (relative or exact) weight of each coin using as few weighings as possible. A number of different versions of coin weighing problems exist. A good survey can be found in Bshouty's paper on polynomial time algorithms for coin weighing problems [Bsh09]. For most coin weighing problems, the information-theoretic lower bound is tight.

Query complexities are also studied in the context of *decision tree complexity* and *boolean functions*. A well-known example of the former is the $\Omega(n \log n)$ bound for comparison-based sorting algorithms. The queries are comparisons, and the secret is the unknown permutation of the input. Lower bounds for more complex queries, e.g., linear functions of the input, are also known. In the latter context, see [BdW02] for a survey, the goal is to evaluate a given Boolean function f on a secret argument x by querying bits of x. For example, x could be the adjacency matrix of a graph and f is one (zero) if the graph is connected (not connected). For graph connectivity the deterministic query complexity is $\Theta(n^2)$ and the randomized complexity is about $\Omega(n^{4/3})$.

2 Preliminaries

For all positive integers $k \in \mathbb{N}$ we define $[k] := \{1, \ldots, k\}$ and $[0..k] := [k] \cup \{0\}$. By e_k^n we denote the kth unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ of length n. For a set $I \subseteq [n]$ we define $e_I^n := \sum_{i \in I} e_i^n = \oplus_{i \in I} e_i^n$, where $\oplus$ denotes the bitwise exclusive-or. We say that we *create* y *from* x *by flipping* I or that we *create* y *from* x *by flipping the entries in position* I if we set $y := x \oplus e_I^n$. By S_n we denote the set of all permutations of $[n]$. For $r \in \mathbb{R}_{\geq 0}$, let $\lceil r \rceil := \min\{n \in \mathbb{N}_0 \mid n \geq r\}$. and $\lfloor r \rfloor := \max\{n \in \mathbb{N}_0 \mid n \leq r\}$. To increase readability, we sometimes omit the $\lceil \cdot \rceil$ signs; that is, whenever we write r where an integer is required, we implicitly mean $\lceil r \rceil$.

Let $n \in \mathbb{N}$. For $z \in \{0,1\}^n$ and $\pi \in S_n$ we define

$$f_{z,\pi} : \{0,1\}^n \to [0..n], \quad x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : z_{\pi(j)} = x_{\pi(j)}\}\,.$$

z is called the *target string* of $f_{z,\pi}$ and π is called the *target permutation*. Paul must identify target string and target permutation by asking a sequence $(x^1, x^2, \ldots)$ of queries. The answer to query x^i is the score $s^i = f_{z,\pi}(x^i)$. Paul can stop after t queries if there is only a *single* pair $(z,\pi) \in \{0,1\}^n \times S_n$ with $s^i = f_{z,\pi}(x^i)$ for $1 \leq i \leq t$.

A randomized strategy for Paul is a sequence of $p^{(0)}, p^{(1)}, \ldots$ of probability distributions over $\{0,1\}^n$. For any t, the distribution $p^{(t)}$ may only depend on the first $t-1$ queries and scores. A strategy is deterministic if all distributions are one-point distributions. The expected complexity of a randomized strategy on input (z,π) is the expected number of queries required to identify the secret, and the expected complexity of a strategy is the worst case over all secrets.

A simple information-theoretic argument gives a $\Omega(n)$ lower bound. The search space has size $2^n n!$, since the unknown code is an element of $\{0,1\}^n \times S_n$. That is, we need to "learn" $\Omega(n \log n)$ bits of information. Each score is a number between 0 and n, i.e., we learn at most $O(\log n)$ bits of information per query, and the $\Omega(n)$ bound follows.

We remark that knowing z allows Paul to determine π with $n-1$ queries $z \oplus e_i^n$, $1 \leq i < n$. Observe that $\pi^{-1}(i)$ equals $f_{z,\pi}(z \oplus e_i^n) + 1$. In evolutionary computation, one is frequently only interested in finding an x maximizing $f_{z,\pi}(x)$. In our situation, $x = z$, and hence *finding an optimizing argument* is no easier (up to $O(n)$ questions) than *learning* the secret (z,π).

An observation crucial to all our proofs is the fact that a vector $(V_1, \ldots, V_n)$ of subsets of $[n]$, together with a top score query (x^*, s^*), captures the knowledge provided by a *guessing history* $\mathcal{H} = (x^i, s^i)_{i=1}^t$ about the secret (z,π). Intuitively, V_i corresponds to the possible values of $\pi(i)$ and we call V_i the *candidate set* for position i. Note that, however, that in general not all values in a candidate set could be valid. For instance, if at some stage $V_1 = \{2,4,5\}$, $V_2 = \{2,4\}$, $V_3 = \{2,5\}$, and $V_4 = \{5,7\}$, the algorithm can deduce that $\pi(4) = 7$.

Theorem 1. *Let $t \in \mathbb{N}$, and let $\mathcal{H} = (x^i, s^i)_{i=1}^t$ be a guessing history. Construct sets $V_1, \ldots, V_n \subseteq [n]$ according to the following rules:*

(1) If there are h and ℓ with $j \leq s^h \leq s^\ell$ and $x_i^h \neq x_i^\ell$, then $i \notin V_j$.

(2) If there are h and ℓ with $s = s^h = s^\ell$ and $x_i^h \neq x_i^\ell$, then $i \notin V_{s+1}$.

(3) If there are h and ℓ with $s^h < s^\ell$ and $x_i^h = x_i^\ell$, then $i \notin V_{s^h+1}$.

(4) If i is not excluded by one of the rules above, then $i \in V_j$.

Furthermore, let $s^* := \max\{s^1, \ldots, s^t\}$ and let $x^* = x^j$ for some j with $s^j = s^*$.

A pair (z,π) is consistent with $\mathcal{H}$ if and only if (a) $f_{z,\pi}(x^*) = s^*$ and (b) $\pi(i) \in V_i$ for all $i \in [n]$.

Proof. Let (z,π) satisfy conditions (a) and (b). We show that (z,π) is consistent with $\mathcal{H}$. To this end, let $h \in [t]$. We need to show that $f^h := f_{z,\pi}(x^h) = s^h$.

Assume $f^h < s^h$. Then $z_{\pi(f^h+1)} \neq x^h_{\pi(f^h+1)}$. Since $f^h + 1 \leq s^*$, this implies (together with (a)) that $x^h_{\pi(f^h+1)} \neq x^*_{\pi(f^h+1)}$. Rule (1) yields $\pi(f^h+1) \notin V_{f^h+1}$; a contradiction to (b).

Similarly, if we assume $f^h > s^h$, then $x^h_{\pi(s^h+1)} = z_{\pi(s^h+1)}$. We distinguish two cases. If $s^h < s^*$, then by condition (a) we had $x^h_{\pi(s^h+1)} = x^*_{\pi(s^h+1)}$. By rule (3) this would imply $\pi(s^h+1) \notin V_{s^h+1}$; a contradiction to (b).

On the other hand, if $s^h = s^*$ was true, then $x^h_{\pi(s^h+1)} = z_{\pi(s^h+1)} \neq x^*_{\pi(s^h+1)}$ by (a). Rule (2) would imply $\pi(s^h+1) \notin V_{\pi(s^h+1)}$, again contradicting (b).

Necessity is trivial. □

The following update rules maintain the V_j's. In the beginning, let $V_j := [n]$, $1 \leq j \leq n$. After the first query, do nothing. For all subsequent queries, do the following: Let I be the set of indices in which the current query x and the current best query x^* agree. Let s be the objective value of x and let s^* be the objective value of x^*.

Rule 1: If $s < s^*$, then $V_i \leftarrow V_i \cap I$ for $1 \leq i \leq s$ and $V_{s+1} \leftarrow V_{s+1} \setminus I$.

Rule 2: If $s = s^*$, then $V_i \leftarrow V_i \cap I$ for $1 \leq i \leq s^* + 1$.

Rule 3: If $s > s^*$, then $V_i \leftarrow V_i \cap I$ for $1 \leq i \leq s^*$ and $V_{s^*+1} \leftarrow V_{s^*+1} \setminus I$. We further replace $s^* \leftarrow s$ and $x^* \leftarrow x$.

3 The Deterministic Query Complexity

Theorem 2. *The deterministic query complexity of the* LeadingOnes*-game with n positions is $\Theta(n \log n)$.*

As mentioned in the introduction, the upper bound can be achieved by an algorithm that resembles binary search and iteratively identifies $\pi(1), \ldots, \pi(n)$ and the corresponding bit values $z_{\pi(1)}, \ldots, z_{\pi(n)}$.

The lower bound follows from the observation that an adversary can force $\Omega(\log n)$ queries for every two positions revealed. He proceeds in phases. In each phase, he reveals two positions and forces $\Omega(\log n)$ queries. We outline the first phase. The adversary answers 0 or 1 to each query in the phase. The answer can always be given in a way such that the size of V_1 at most halves; see the update rules in the preceding section. Once $|V_1| = 2$, the adversary chooses $i' = \pi(1) \in V_1$ and $i'' = \pi(2) \in V_2$ arbitrarily, sets $z_{i'} = x^*_{i'}$ and $z_{i''} = 1 - x^*_{i''}$, removes i' and i'' from $V_3, \ldots, V_n$, and moves on to the next phase. Here x^* is a query with maximal score.

4 A Randomized $O(n \log\log n)$ Winning Strategy

We show that Paul needs only $O(n \log\log n)$ queries to identify the secret code if we allow him to make random decisions.

Theorem 3. *The randomized query complexity of the* LeadingOnes*-game with n positions is $O(n \log\log n)$.*

The strategy has two parts. In the first part, we identify the positions $\pi(1), \ldots, \pi(q)$ and the corresponding bit values $z_{\pi(1)}, \ldots, z_{\pi(q)}$ for some $q \in n - \Theta(n/\log n)$ with $O(n \log\log n)$ queries. In the second part, we find the remaining $n - q \in \Theta(n/\log n)$ positions and entries using the binary search algorithm with $O(\log n)$ queries per position. Part 1 is outlined below; the details are in the appendix.

The Strategy: We remind the reader that, at any given stage of the LeadingOnes-game, all information that the queries reveal about the secret (z, π) is stored in the candidate sets $V_1, \ldots, V_{s^*+1}$. Here and in the following, by s^* we denote the current best score. By x^* we denote a corresponding query, i.e., $f_{z,\pi}(x^*) = s^*$. For brevity, we write f for $f_{z,\pi}$.

There is a trade-off between learning more information about π by reducing the sets $V_1, \ldots, V_{s^*+1}$ and increasing the score s^*. In our $O(n \log\log n)$ winning strategy, we alternate between increasing the score s^* and reducing the sizes of the candidate sets V_i.

To illustrate the main ideas, we describe the first steps. We start by querying $f(x)$ and $f(y)$, where x is arbitrary and $y = x \oplus 1^n$. By swapping x and y if needed, we may assume $f(x) = 0 < f(y)$. We now run a randomized binary search for finding $\pi(1)$. We choose uniformly at random a subset $F_1 \subseteq V_1$ ($V_1 = [n]$ in the beginning) of size $|F_1| = |V_1|/2$. We query $f(y')$

where y' is obtained from y by flipping the bits in F_1. If $f(y') > f(x)$, we set $V_1 \leftarrow V_1 \setminus F_1$; we set $V_1 \leftarrow F_1$ otherwise. This ensures $\pi(1) \in V_1$. We stop the binary search once $\pi(2) \notin V_2$ is sufficiently likely; the analysis will show that $\Pr[\pi(2) \in V_1] \leq (\log n)^{-d}$ for some large enough constant d is a good choice.

We now start pounding on V_2. Let $\{x, y\} = \{y, y \oplus 1_{V_1}\}$. If $\pi(2) \notin V_2$, one of $f(x)$ and $f(y)$ is one and the other is larger than one. Swapping x and y, if necessary, we may assume $f(x) = 1 < f(y)$. We now use randomized binary search to reduce the size of V_2 to $n/(\log_n)^d$.

At this point we have $|V_1|, |V_2| \leq n/\log_n^d$. We hope that $\pi(3) \notin V_1 \cup V_2$, in which case we can continue as before.

At some point the probability that $\pi(i) \notin V_1 \cup \ldots \cup V_{i-1}$ drops below a certain threshold and we cannot ensure to make progress anymore by simply querying $x \oplus ([n] \setminus (V_1 \cup \ldots \cup V_{i-1}))$. This situation is reached when $i = \log n$ and hence we abandon the previously described strategy once $s^* = \log n$. At this point, we move our focus from increasing the current best score s^* to reducing the size of the candidate sets $V_1, \ldots, V_{s^*}$. We reduce their sizes to at most $n/(\log^2 n)^d$. This will be discussed below.

Once the sizes $|V_1|, \ldots, |V_{s^*}|$ have been reduced to at most $n/(\log^2 n)^d$, we move our focus back to increasing s^*. The probability that $\pi(s^* + 1) \in V_1 \cup \ldots \cup V_{s^*}$ will be small enough (details below), and we proceed as before by flipping $[n] \setminus (V_1 \cup \ldots \cup V_{s^*})$ and reducing the size of V_{s^*+1} to $n/(\log n)^d$. Again we iterate this process until $s^* = 2 \log n$. At this point we reduce the sizes of $V_{\log n+1}, \ldots, V_{2 \log n}$ to $n/(\log^2 n)^d$. We say that we *add these candidate sets to the second level*. They were on the *first level* before, i.e., when their size was still $n/(\log n)^d$. We continue with this until $\log^2 n$ sets have been added to the second level. At this point we reduce the sizes of $V_1, \ldots, V_{\log^2 n}$ to at most $n/(\log^4 n)^d$, thus adding them to the *third level.*

In total we have $t = O(\log \log n)$ levels. Each time $x_i := \log^{2^{i-1}} n$ sets have been added to the ith level, we reduce the size of these sets to $n/(\log^{2^i} n)^d = n/x_{i+1}^d$, thus adding them to the $(i + 1)$st level.

When x_t sets have been added to the tth level, we reduce the size of each of these sets to one and move them to the last level, level $t + 1$. Thus $V_j = \{\pi(j)\}$ for each set V_j on level $t + 1$.

We need to discuss how we handle failures; i.e., if $f(y) = f(x) = s^*$ or $f(y), f(x) > s^*$ holds. In this case we know that $\pi(s^* + 1)$ has not been flipped. Hence $\pi(s^* + 1) \in \cup_{j=1}^{s^*} V_j$ must hold. Therefore, we must reduce the size of $\cup_{j=1}^{s^*} V_j$ to “free” $\pi(s^* + 1)$. We immediately abort the first level by reducing the size of each of the V_is currently on that level to n/x_2^d, thus adding them to the second level. Should we still not make progress by flipping all bits in $[n] \setminus \cup_{j=1}^{s^*} V_j$, we continue by reducing the size of each level-2 set to n/x_3^d, and so on, until we eventually have $\pi(s^* + 1) \notin \cup_{j=1}^{s^*} V_j$, in which case we can continue with the “increase the score by one, then reduce the size of the candidate sets”-procedure described above.

We describe how to reduce the sizes of the up to $x_{\ell-1}$ candidate sets from some value $\leq n/x_{\ell-1}^d$ to the target size n/x_ℓ^d of level ℓ with an expected number of $O(1) x_{\ell-1} d (\log x_\ell - \log x_{\ell-1}) / \log x_{\ell-1}$ queries. We make us of a procedure `subroutine2` for this reduction: It reduces the sizes of at most k candidate sets to a kth fraction of their original size using at most $O(k)$ queries, where k is a parameter. We use `subroutine2` with parameter $k = x_{\ell-1}$ repeatedly to achieve the full reduction.

`subroutine2` is given a set J of at most k indices and a y with $f(y) \geq \max J$. The goal is to reduce the size of each candidate set V_j, $j \in J$, below a target size m where $m \geq |V_j|/k$ for all $j \in J$. The routine works in phases. Let J be the set of indices of the candidate sets that are still above the target size at the beginning of a phase. For each $j \in J$, we randomly choose a subset $F_j \subseteq V_j$ of size $|V_j|/k$. We create a new bit string y' from y by flipping all candidates $\cup_{j \in J} F_j$. A condition on the sets $V_j, j \in J$ ensures that we have either $f(y') \geq \max J$

or $f(y') = j-1$ for some $j \in J$.[1]

In the first case, i.e., if $f(y') \geq \max J$, none of the sets V_j was *hit*, and for all $j \in J$ we can remove the subset F_j from V_j. We call such queries "*off-trials*". An off-trial reduces the size of all sets V_j, $j \in J$, to a $(1-1/k)$th fraction of their original size.

If, on the other hand, we have $f(y') = j-1$ for some $j \in J$, we can replace V_j by set F_j as $\pi(j) \in F_j$ must hold.[1] Since $|F_j| = |V_j|/k \leq m$ by assumption, this set has now been reduced to its target size and we can remove it from J.

We continue in this way until at least half of the indices are removed from J and at least ck off-trials occurred, for some constant c satisfying $(1-1/k)^{ck} \leq 1/2$. Consider any j that is still in J. The size of V_j was reduced by a factor $(1-1/k)$ at least ck times. Thus its size was reduced to at most half its original size. We now may half k without destroying the invariant $m \geq |V_j|/k$ for $j \in J$. The effect of halving k is that the relative size of the sets F_j will be doubled for the sets V_j that still take part in the reduction process.

Lemma 4. *Let $k \in \mathbb{N}$, let $J \subseteq [n]$ be a set of at most k indices with $V_j \cap V_p = \emptyset$ for $j \in J$ and $p \in [\max J] \backslash \{j\}$, and let $y \in \{0,1\}^n$ be such that $f(y) \geq \max J$, and let $m \in \mathbb{N}$ be such that $m \geq |V_j|/k$ for all $j \in J$.*

In expectation it takes $O(k)$ queries until `subroutine2`(k, J, m, y) *has reduced the size of V_j to at most m for each $j \in J$.*

Proof of Theorem 3: It remains to show that the first phase of our winning strategy requires at most $O(n \log\log n)$ queries. If no failure happened, the expected number of queries was bounded by

$$\begin{aligned}
&\frac{q}{x_t}\left(\frac{x_t \log n}{\log x_t} + \frac{x_t}{x_{t-1}}\left(\frac{x_{t-1}dc(\log x_t - \log x_{t-1})}{\log x_{t-1}}\right.\right.\\
&\qquad\qquad\left.\left.+\frac{x_{t-1}}{x_{t-2}}\left(\ldots + \frac{x_2}{x_1}\left(\frac{x_1 dc(\log x_2 - \log x_1)}{\log x_1} + x_1 d \log x_1\right)\right)\right)\right)\\
\leq & ndc\left(\frac{\log n}{\log x_t} + \frac{\log x_t}{\log x_{t-1}} + \ldots + \frac{\log x_2}{\log x_1} + \log x_1 - t\right), \qquad (1)
\end{aligned}$$

where c is the constant hidden in the $O(1)$-term in Lemma 4. To verify this formula, observe that we fill the $(i-1)$st level x_i/x_{i-1} times before level i is filled with its x_i candidate sets. To add x_{i-1} candidate sets from level $i-1$ to level i, we need to reduce their size from n/x_{i-1}^d to n/x_i^d. By Lemma 4 this requires at most $x_{i-1}dc(\log x_i - \log x_{i-1})/\log x_{i-1}$ queries. The additional $x_1 d \log x_1$ term accounts for the queries caused by the randomized binary search algorithm through which we initially reduce the sizes of the V_is to n/x_1^d—requiring $d \log x_1$ queries per call. Finally, the term $x_t \log n/\log x_t$ accounts for the final reduction of the V_is to a set containing only one single element (at this stage we shall finally have $V_i = \{\pi(i)\}$). More precisely, this term is $(x_t(\log n - d\log x_t))/\log x_t$ but we settle for upper bounding this expression by the term given in the formula.

Next we need to bound the number of queries caused by *failures*. We show that, on average, not too many failures happen. More precisely, we show that the expected number of level-i failures is at most $n^2/((n-q)(x_i^{d-1}-1))$. By Lemma 4, each such level-i failure causes an additional number of at most $1 + x_i dc(\log x_{i+1} - \log x_i)/\log x_i$ queries; (the 1 counts for the

[1]This can be ensured by the somewhat technical condition that $V_j \cap V_p = \emptyset$ for all $j \in J$ and $p \in [\max J] \backslash \{j\}$. This condition is satisfied throughout our strategy.

query in which we failed to increase the current best score). Thus

$$\sum_{i=1}^{t} \frac{n^2}{(n-q)(x_i^{d-1}-1)} \left(1 + \frac{x_i dc(\log x_{i+1} - \log x_i)}{\log x_i}\right) \tag{2}$$

bounds the expected number of additional queries caused by failures.

We recall the settings of the x_i. We have $x_1 = \log n$ and $x_i = \log^{2^{i-1}} n$. We further have $t \in \Theta(\log\log n)$, so that $\log x_t = \Omega(\log n)$. The parameter d is some constant ≥ 4. With these parameter settings, formula (1) evaluates to

$$ndc\left(\frac{\log n}{\log x_t} + 2(t-1) + \log\log n - t\right) = O(n \log\log n)$$

and, somewhat wasteful, we can bound formula (2) from above by

$$\frac{n^2 dc}{n-q} \sum_{i=1}^{t} x_i^{-(d-3)} = O(n\log n) \sum_{i=0}^{t-1} x_1^{-(d-3)2^i} < O(n\log n)(x_1^{d-3}-1)^{-1} = O(n)\,.$$

This shows that the overall expected number of queries sums to $O(n\log\log n) + O(n) = O(n \log\log n)$.

5 The $\Omega(n \log\log n)$ Lower Bound

Theorem 5. *The best winning strategy for the* LeadingOnes*-game with n positions requires $\Omega(n \log\log n)$ queries.*

Let Π be a permutation drawn uniformly among all the permutations of $[n]$ (in this section, we use capital letters to denote random variables). Given such a permutation, we let our target string Z be the one satisfying $Z_{\Pi(i)} = (i \bmod 2)$ for $i = 1, \ldots, n$. Since Z is uniquely determined by the permutation Π, we will mostly ignore the role of Z in the rest of this section. Finally, we use $F(x)$ to denote the value of the random variable $f_{Z,\Pi}(x)$ for $x \in \{0,1\}^n$. We will also use the notation $a \equiv b$ to mean that $a \equiv b \bmod 2$.

By fixing the random coins (the easy direction of Yao's principle), a randomized solution with expected t queries implies the existence of a deterministic query scheme with expected t queries over our distribution. We lower bound t for such a deterministic query scheme.

A deterministic query scheme is a decision tree in which each node v is labeled with a string $x_v \in \{0,1\}^n$. Each node has $n+1$ children, numbered from 0 to n, and the ith child is traversed if $F(x_v) = i$. To guarantee correctness, no two inputs can end up in the same leaf.

For a node v in the decision tree, we define $\max_v$ as the largest value of F seen along the edges from the root to v. Note that $\max_v$ is not a random variable and in fact, at any node v and for any ancestor u of v, conditioned on the event that the search path reaches v, the value of $F(x_u)$ is equal to the index of the child of u that lies on the path to v. Finally, we define S_v as the subset of inputs (as outlined above) that reach node v.

We use a potential function which measures how much "information" the queries asked have revealed about Π. Our goal is to show that the expected increase in the potential function after asking each query is small. Our potential function depends crucially on the candidate sets. The update rules for the candidate sets are slightly more specific than the ones in Section 2 because we now have a fixed connection between the two parts of the secret. We denote the candidate set for $\pi(i)$ at node v with V_i^v. The precise definition of candidate sets is as follows:

$$V_i^{w_t} = \begin{cases} V_i^v \cap P_{i \bmod 2}^v & \text{if } i \leq t\,, \\ V_i^v \cap P_{t \bmod 2}^v & \text{if } i = t+1\,, \\ V_i^v & \text{if } i > t+1. \end{cases}$$

As with the upper bound case, the candidate sets have some very useful properties. These properties are slightly different from the ones observed before, due to the fact that some extra information has been announced to the query algorithm. We say that a candidate set V_i^v is *active (at v)* if the following conditions are met: (i) at some ancestor node u of v, we have $F(x_u) = i-1$, (ii) at every ancestor node w of u we have $F(x_w) < i-1$, and (iii) $i < \min\{n/3, \max_v\}$. We call $V^v_{\max_v+1}$ *pseudo-active (at v)*.

Lemma 6. *The candidate sets have the following properties:*

(i) Two candidate sets V_i^v and V_j^v with $i < j \le \max_v$ and $i \not\equiv j$ are disjoint.

(ii) An active candidate set V_j^v is disjoint from any candidate set V_i provided $i < j < \max_v$.

(iii) The candidate set V_i^v, $i \le \max_v$ is contained in the set $V^v_{\max_v+1}$ if $i \equiv \max_v$ and is disjoint from it if $i \not\equiv \max_v$.

(iv) For two candidate sets V_i^v and V_j^v, $i < j$, if $V_i^v \cap V_j^v \ne \emptyset$ then $V_i^v \subset V_j^v$.

We define the potential of an active candidate set V_i^v as $\log\log(2n/|V_i^v|)$. This is inspired by the upper bound: the potential increase of one corresponds to a candidate set advancing one level in the upper bound context (in the beginning, a set V_i^v has size n and thus its potential is 0 while at the end its potential is $\Theta(\log\log n)$. With each level, the quantity n divided by the size of V_i is squared). We define the potential at a node v as

$$\varphi(v) = \log\log\frac{2n}{|V^v_{\max_v+1}| - \mathrm{Con}_v} + \sum_{j\in A_v}\log\log\frac{2n}{|V_j^v|},$$

in which A_v is the set of indices of active candidate sets at v and Con_v is the number of candidate sets contained inside $V^v_{\max_v+1}$. Note that from Lemma 6, it follows that $\mathrm{Con}_v = \lfloor \max_v/2\rfloor$.

After some lengthy calculations, it is possible to prove the following lemma. The full proof is presented in Appendix C. Here instead, we give only a very high level overview.

Lemma 7. *Let v be a node in T and let $\mathbf{i}_v$ be the random variable giving the value of $F(x_v)$ when $\Pi \in S_v$ and 0 otherwise. Also let $w_0, \ldots, w_n$ denote the children of v, where w_j is the child reached when $F(x_v) = j$. Then, $\mathrm{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = O(1)$.*

Note that we have $\mathrm{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = \sum_{a=0}^n \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v))$. We consider two main cases: $F(x_v) \le \max_v$ and $F(x_v) > \max_v$. In the first case, the maximum score will not increase in w_a which means w_a will have the same set of active candidate sets. In the second case, the pseudo-active candidate set $V^v_{\max_v+1}$ will turn into an active set $V^{w_a}_{\max_v+1}$ at w_a and w_a will have a new pseudo-active set. While this second case looks more complicated, it is in fact the less interesting part of the analysis. This is because the probability of suddenly increasing the score by α is extremely small (we will show that it is roughly $O(2^{-\Omega(\alpha)})$) which subsumes any significant potential increase for values of $a > \max_v$.

Let $a_1, \ldots, a_{|A_v|}$ be the indices of active candidate sets at v sorted in increasing order. We also define $a_{|A_v|+1} = \max_v + 1$. For a candidate set V_i^v, and a Boolean $b \in \{0,1\}$, let $V_i^v(b) = \{j \in V_i^v \mid x_v[j] = b\}$. Clearly, $|V_i^v(0)| + |V_i^v(1)| = |V_i^v|$. For even a_i, $1 \le i \le |A_v|$, let $\varepsilon_i = |V_i^v(1)|/|V_i^v|$, and for odd i, let $\varepsilon_i = |V_i^v(0)|/|V_i^v|$. Also, let $\varepsilon_i' := \Pr[a_i \le F(x_v) < a_{i+1} - 1 | \Pi \in S_v \wedge F(x_v) \ge a_i]$. Note that $\varepsilon_i' = 0$ if $a_{i+1} = a_i + 1$.

With these definitions, it is clear that we have $|V_{a_i}^{w_j}| = |V_{a_i}^v|$ for $j < a_i - 1$, $|V_{a_i}^{w_j}| = \varepsilon_i |V_{a_i}^v|$ for $j = a_i - 1$, and $|V_{a_i}^{w_j}| = (1-\varepsilon_i)|V_{a_i}^v|$ for $j > a_i - 1$. The important fact is that we can also bound other probabilities using the ε_i's and ε_i''s: we can show that $\Pr[F(x_v) = a_i - 1 | \Pi \in S_v] \le \varepsilon_i \Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon_j')$ and $\Pr[a_i \le F(x_v) < a_{i+1} - 1 | \Pi \in S_v] \le \varepsilon_i'(1-\varepsilon_i)\Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon_j')$. Thus, for the most part, proving Lemma 7 reduces to proving an inequality based on ε_i's and ε_i''s. We say for the most part since the potential function also crucially depends on the $|V_j^v|$s (and less importantly on Con_v). This presents a challenge since it is impossible to bound the $|V_j^v|$s using ε_i's and ε_i''s. To overcome this problem, we observe that by Lemma 6 any two active candidate sets are disjoint, enabling us to divide the potential function into smaller summations: for $\varphi(v)$ (as well as $\varphi(w_a)$), the summation over $j \in A_v$ is divided into $O(\log n)$ summations such that the ith summation is over all the indices j such that $|V_j^v|$ is roughly 2^i. Thus, in the ith summation, we can replace $|V_j^v|$ with 2^i which enables us to turn the problem into proving a general inequality. This is merely a very high level overview of the potential function analysis and there are many details and technical difficulties. For the complete proof see Appendix C.

5.1 Potential at the End

Intuitively, if the maximum score value increases after a query, it increases, on average, only by a constant factor. In fact, in Appendix C, we prove that the probability of increasing the maximum score value by α after one query is $O(2^{-\alpha})$. Thus, it follows from the definition of the active candidate sets that when the score reaches $n/3$, there are $\Omega(n)$ active candidate sets, on average. However, by Lemma 6, the active candidate sets are disjoint. This means that a fraction of them (again at least $\Omega(n)$ of them), must be small, or equivalently, their total potential is $\Omega(n \log\log n)$. In the rest of this section, we prove this intuition.

Given an input (z, π), we say an edge e in the decision tree is *increasing* if e corresponds to an increase in the maximum score and it is traversed given the input (z, π). We say that an increasing edge is *short* if it corresponds to an increase of at most c in the maximum function score (in which c is a sufficiently large constant) and we call it *long* otherwise. Let N be the random variable denoting the number of increasing edges seen on input Π before reaching a node with score greater than $n/3$. Let L_j be the random indicator variable taking the value 0 if the jth increasing edge is short, and taking the value equal to the amount of increase in the score along this edge if not. If $j > N$, then we define $L_j = 0$. Also let W_j be the random variable corresponding to the node of the decision tree where the jth increase happens. As discussed, in Appendix C we prove that for every node v, $\Pr[L_j \geq \alpha | W_j = v] \leq 2^{-\Omega(\alpha)}$. We want to upper bound $\sum_{j=1}^{n} \mathrm{E}[L_j]$ (there are always at most n increasing edges). From the above, we know that

$$\begin{aligned}
\mathrm{E}[L_j] &\leq \mathrm{E}[L_j \mid N \geq j] \\
&= \sum_{v \in T} \sum_{i=c+1}^{n} i \cdot \Pr[L_j = i \wedge W_j = v \mid N \geq j] = \sum_{v \in T} \sum_{i=c+1}^{n} i \cdot \Pr[L_j = i \wedge W_j = v] \\
&= \sum_{v \in T} \sum_{i=c+1}^{n} i \cdot \Pr[L_j = i \mid W_j = v] \Pr[W_j = v] \leq \sum_{v \in T} \sum_{i=c+1}^{n} \frac{i}{2^{\Omega(i)}} \Pr[W_j = v] \\
&\leq \sum_{v \in T} \frac{1}{2^{\Omega(c)}} \Pr[W_j = v] \leq \frac{1}{2^{\Omega(c)}} ,
\end{aligned}$$

where the summation is taken over all nodes v in the decision tree T. The computation shows $\sum_{j=1}^{n} \mathrm{E}[L_j] \leq n/2^{\Omega(c)}$. By Markov's inequality, we get that with probability at least 3/4, we have $\sum_{j=1}^{n} L_j \leq n/2^{\Omega(c)}$. Thus, when the function score reaches $n/3$, short edges must account for $n/3 - n/2^{\Omega(c)}$ of the increase which is at least $n/6$ for a large enough constant c. Since any short edge has length at most c, there must be at least $n/(6c)$ short edges. As discussed, this implies existence of $\Omega(n)$ active candidate sets that have size $O(1)$, meaning, their contribution to the potential function is $\Omega(\log \log n)$. Combined with Lemma 7 this proves Theorem 5.

6 Polynomial Time Feasibility Checks

Let $\mathcal{H} := (x^i, s^i)_{i=1}^t$ be a vector of search points $x^i \in \{0,1\}^n$ and scores $s^i \in [0..n]$. We call $\mathcal{H}$ a *guessing history.* $\mathcal{H}$ is *feasible* if there exists a pair $(z, \pi) \in \{0,1\}^n \times S_n$ that is *consistent* with it; i.e., $f_{z,\pi}(x^i) = s^i$ for all $i \in [t]$.

Theorem 8. *It is decidable in polynomial time whether a guessing history is feasible. Furthermore, we can efficiently compute the number of pairs consistent with it.*

Both statements in Theorem 8 are based on Theorem 1. Given a guessing history we first compute the sets $V_1, \ldots, V_n$ as described in Theorem 1. Next we construct a bipartite graph $G = G(V_1, \ldots, V_n)$ with node set $[n]$ on both sides. Connect j to all nodes in V_j on the other side. Permutations π with $\pi(j) \in V_j$ for all j are in one-to-one correspondence to perfect matchings in G. If there is no perfect matching, the history in infeasible. Otherwise, let π be any permutation with $\pi(j) \in V_j$ for all j. Set $z_{\pi(i)} := x^*_{\pi(i)}$ for $i \in [s^*]$, and $z_{\pi(i)} := 1 - x^*_{\pi(i)}$ for $i \notin [s^*]$. The pair (z, π) is consistent with $\mathcal{H}$.

For the counting problem we first observe that for any fixed consistent permutation π, the number of strings z such that (z, π) is consistent with $\mathcal{H}$ equals $2^{n-(s^*+1)}$ if $s^* < n$, and it equals one if $s^* = n$. By Hall's condition a perfect matching exists if and only if $| \cup_{j \in J} V_j| \geq |J|$ for every $J \subseteq [n]$. Constructing the permutations in a greedy fashion, one shows that the number of consistent permutations is equal to $\prod_{1 \leq i \leq n} (|V_i| - |\{j < i \mid V_j \subseteq V_i\}|)$.

References

[AD11] Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics.* World Scientific, 2011.

[BdW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288:21–43, 2002.

[Bsh09] Nader H. Bshouty. Optimal algorithms for the coin weighing problem with a spring scale. In *Proc. of the 22nd Conference on Learning Theory (COLT'09)*, 2009.

[CCH96] Zhixiang Chen, Carlos Cunha, and Steven Homer. Finding a hidden code by asking questions. In *Proc. of the 2nd Annual International Conference on Computing and Combinatorics (COCOON'96)*, volume 1090 of *Lecture Notes in Computer Science*, pages 50–55. Springer, 1996.

[Chv83] Vasek Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.

[DJW06] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544, 2006.

[DW12] Benjamin Doerr and Carola Winzen. Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes. In *Proc. of Artificial Evolution (EA'11)*, volume 7401 of *Lecture Notes in Computer Science.* Springer, 2012. To appear. Available online at http://www.mpi-inf.mpg.de/~winzen/DoerrWinzen_LeadingOnes.pdf.

[ER63] Paul Erdős and Alfréd Rényi. On two problems of information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 8:229–243, 1963.

[FL10] Riccardo Focardi and Flaminia L. Luccio. Cracking bank pins by playing Mastermind. In *Proceedings of the 5th international conference on Fun with algorithms (FUN'10)*, pages 202–213. Springer-Verlag, 2010.

[Goo09a] Michael T. Goodrich. The Mastermind attack on genomic data. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy (SP'09)*, pages 204–218. IEEE, 2009.

[Goo09b] Michael T. Goodrich. On the algorithmic complexity of the Mastermind game with black-peg results. *Information Processing Letters*, 109:675–678, 2009.

[Knu77] Donald E. Knuth. The computer as master mind. *Journal of Recreational Mathematics*, 9:1–5, 1977.

[Pel02] Andrzej Pelc. Searching games with errors — fifty years of coping with liars. *Theoretical Computer Science*, 270:71–109, 2002.

[Spe94] Joel Spencer. Randomization, derandomization and antirandomization: Three games. *Theoretical Computer Science*, 131:415–429, 1994.

[SZ06] Jeff Stuckman and Guo-Qiang Zhang. Mastermind is NP-complete. *INFOCOMP Journal of Computer Science*, 5:25–28, 2006.

[Vig12] Giovanni Viglietta. Hardness of Mastermind. In *Proc. of the 6th International Conference on Fun with Algorithms (FUN'12)*, volume 7288 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 2012.

Appendix

A Details on the Deterministic Lower Bound

We repeat Theorem 2.

Theorem 2. *The deterministic query complexity of the* LeadingOnes*-game with n positions is $\Theta(n \log n)$.*

As mentioned in the main text, the upper bound of Theorem 2 can be achieved by an algorithm that resembles binary search and iteratively identifies $\pi(1), \ldots, \pi(n)$ and the corresponding bit values $z_{\pi(1)}, \ldots, z_{\pi(n)}$. The algorithm itself is discussed in Section 4, cf. Algorithm 3. We provide below the more interesting lower bound.

Proof. We show that an adversary can force $\Omega(\log n)$ queries for every two positions revealed. We proceed in phases. In each phase, we (= adversary) reveal two positions and force $\Omega(\log n)$ queries. At the beginning of the kth phase, $k \geq 0$, $\pi(1)$ up to $\pi(2k)$ and $z_{\pi(1)}$ to $z_{\pi(2k)}$ are fixed, V_{2k+1} to V_n are equal to $[n] \setminus \{\pi(1), \ldots, \pi(2k)\}$, and the largest score ever returned is $2k-1$. Let z be a string with score $2k-1$; z is undefined at the beginning of the zeroth phase.

We start phase k. Queries that disagree with z in one of the already fixed positions or are equal to z are answered in the unique way. A clever algorithm will ask no such query. Let x^* be the first query in the phase that is different from z and agrees with z in the already fixed positions; in the zeroth phase, x^* is the first query. We (= the adversary) answer with value $2k+1$. Let $V' = V_{2k+1}$ and $V'' = V_{2k+2}$ be the candidate sets for the $(2k+1)$st and the $(2k+2)$th position, respectively. As long as we only answer with values $2k$ and $2k+1$, the sets V_{2k+3} to V_n are not affected and remain the full set $[n] \setminus \{\pi(1), \ldots, \pi(2k)\}$. Since the largest value return before query x^* is $2k-1$, the answer $2k+1$ to query x^* affects only sets V_1 to V_{2k} according to rule (3).[2] Thus $V' = V'' = [n]$ after query x^*

Let x be the next query. Let $I = \{i \mid x_i = x_i^*\}$ be the set of positions in which x and x^* agree. Let us now see how the sets V' and V'' change according to the score.

- If we give x score $2k$, then V' becomes $V' \setminus I$ and V'' is unchanged according to rule (1)
- If we give x score $2k+1$, then V' becomes $V' \cap I$ and V'' becomes $V'' \cap I$ according to rule (2).

We conclude that $V' \subseteq V''$ is an invariant no matter how we answer. We can proceed as long as $|V'| \geq 1$, $|V''| \geq 1$ and $|V' \cup V''| \geq 2$, i.e., as long as Hall's condition guarantees the existence of a permutation. All three conditions are subsumed by $|V'| \geq 2$. The strategy is clear now.

If $|V' \cap I| \geq |V'|/2$, we assign to x the score $2k+1$, and if $|V' \cap I| < |V'|/2$ and hence $|V' \setminus I| \geq |V'|/2$, we assign to it score $2k$. In this way, the cardinality of V' is at most halved. In particular, if $|V'| \geq 3$ before the query x, then $|V'| \geq 2$ after the query. If $|V'| \geq 3$ still, we continue with the phase.

If $|V'| = 2$, choose $i' = \pi(2k+1) \in V'$ and $i'' = \pi(2k+2) \in V''$ arbitrarily, set $z_{i'} = x_{i'}^*$ and $z_{i''} = 1 - x_{i''}^*$, remove i' and i'' from V_{2k+3} to V_n, and move on to the next phase.

We forced $\log(n-2k) - 1$ queries in the kth phase, $0 \leq k < \lfloor n/2 \rfloor$. Thus we force $\Omega(n \log n)$ queries in total. □

[2] Rule numbers refer to the rules as stated after Theorem 1.

B Details on the Randomized $O(n \log \log n)$ Winning Strategy

For readability purposes, we repeat the material presented in the main text. The reader only interested in the proof details may skip Section B.1 and most parts of Section B.4.

Theorem 3. *The randomized query complexity of the* LeadingOnes*-game with* n *positions is* $O(n \log \log n)$.

The winning strategy verifying this bound has two parts. In the beginning, we use Algorithm 1 to identify the positions $\pi(1), \ldots, \pi(q)$ and the corresponding bit values $z_{\pi(1)}, \ldots, z_{\pi(q)}$; for some $q \in n - \Theta(n/\log n)$. As we shall see below, this requires $O(n \log \log n)$ queries, cf. Theorem 11. Once $z_{\pi(1)}, \ldots, z_{\pi(q)}$ have been identified, we can find the remaining $n - q \in \Theta(n/\log n)$ positions and entries using the binary search algorithm described in the introduction. This is the second phase of our algorithm. As the binary search strategy requires at most $O(\log n)$ queries per position, the second phase contributes only a linear number of queries—an effort that is negligible compared to the one of the first phase.

B.1 Outline of the Proof for Theorem 3

The main idea behind Algorithm 1 is the following. We alternate between increasing the scores by one and reducing the sizes of the V_is. More precisely, we maintain a string x and an integer s with $f(x) \geq s$ and have already reduced some elements from V_1 to V_s. The sets V_{s+1} to V_n are still equal to $[n]$. Initially, $s = 0$ and x is arbitrary. The sets V_1 to V_s are arranged into $t+1$ levels. The sets in level i have larger index than the sets in level $i+1$. Level $t+1$ contains an initial segment of sets; all sets on level $t+1$ are singletons. On level i, $1 \leq i \leq t$, we can have up to x_i sets. The size of any set on level i is at most n/x_i^d, where $d \geq 1$ is some parameter to be fixed later. Once x_i sets have been added to the ith level, we reduce the size of each of them to at most n/x_{i+1}^d using `subroutine2`, which we describe in Section B.3. The reduced sets are added to the $(i+1)$st level. `subroutine2` essentially allows us to simultaneously reduce the sizes of k sets to a kth fraction of their original size using at most $O(k)$ queries. After moving the sets from level i to level $i+1$, we either start filling the ith level again (in case the $(i+1)$st level has some capacity left), or we reduce the sizes of the sets on the $(i+1)$st level. Once the tth level contains x_t sets, we reduce the size of each of these sets to one, again employing `subroutine2`, and move them to level $t+1$. Thus $V_j = \{\pi(j)\}$ for each set V_j on level $t+1$.

At level one (calls *Advance*(1)) we attempt to advance s. Let s^* denote the current value of s and let x^* be the current bit string x. We ensure $f(x^*) \geq s^*$. We want to start working on V_{s^*+1}. We hope (this hope will be true with sufficiently high probability, as the analysis shows) that $\pi(s^*+1) \notin \cup_{j \leq s^*} V_j$. We create from x^* a new bit string y by flipping in x^* all bits that are *known* not to be the image of $1, \ldots, s^*$ under π. That is, we set $y := x^* \oplus \left([n] \setminus \cup_{j=1}^{s^*} V_j\right)$ and query $f(y)$. If $f(y) = s^*$ and $f(x) > s^*$, we swap the two strings. This step is needed to ensure correctness of `subroutine1`.

If $f(y) > s^*$ and $f(x^*) = s$ holds, we know that $\pi(s^*+1) \in [n] \setminus \cup_{j=1}^{s^*} V_j$ holds. In this case, we immediately reduce the size of the set V_{s^*+1} to n/x_1^d. This can be done by the binary search like algorithm, `subroutine1`, in $d \log x_1$ queries. This subroutine is described in Section B.2.

If, on the other hand, a *failure* happens; i.e., if in line 17 $f(y) = f(x^*) = s^*$ or $f(x^*) > s^*$ holds, then we know that $\pi(s^*+1)$ has not been flipped. Hence $\pi(s^*+1) \in \cup_{j=1}^{s^*} V_j$ must hold. Therefore, we must reduce the size of $\cup_{j=1}^{s^*} V_j$ to "free" $\pi(s^*+1)$. We immediately abort the first level by reducing the size of each of the V_is currently on that level to n/x_2^d, thus adding them to the second level. Should we still not make progress by flipping all bits in $[n] \setminus \cup_{j=1}^{s^*} V_j$, we continue by reducing the size of each level-2 set to n/x_3^d, and so on, until we eventually have

Algorithm 1: The $O(n \log\log n)$ winning strategy for the LeadingOnes-game with n positions.

```
1  Input: Number of levels $t$. Maximal number $x_1, \dots, x_t \in \mathbb{N}$ of $V$s in each level. Score
   $q \in n - \Theta(n/\log n)$ that is to be achieved in the first phase. Positive integer $d \in \mathbb{N}$.
2  Main Procedure
3  $V_1, \dots, V_n \leftarrow [n]$ ; //$V_i$ is the set of candidates for $\pi(i)$
4  $s \leftarrow 0$ ; //$s$ counts the number of successful iterations
5  Choose $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
6  $J \leftarrow \emptyset$;
7  while $|J| < q$ do
8      $J' \leftarrow Advance(t)$;
9      Reduce the size of the sets $V_j$ with $j \in J'$ to 1 calling subroutine2$(x_t, J', 1, x)$;
10     $J \leftarrow J \cup J'$;
11 Identify the remaining $q$ bits from $x$ and $y := x \oplus \left([n] \setminus \cup_{j=1}^{s} V_j\right)$ using the binary search
   algorithm subroutine1 ; //phase 2

   where Advance is the following function.
12 $Advance$(level $\ell$) //returns a set $J$ of up to $x_\ell$ indices such that $|V_j| \le n/x_\ell^d$ for all $j \in J$
13 $J \leftarrow \emptyset$;
14 while $|J| \le x_\ell$ do
15     if $\ell = 1$ then
16         Create $y$ from $x$ by flipping all bits in $[n] \setminus \cup_{j=1}^{s} V_j$ and query $f(y)$;
17         if $f(x) > s$ and $f(y) = s$ then swap $x$ and $y$;
18         if $f(x) = s$ and $f(y) > s$ then
19             $s \leftarrow s + 1$;
20             $V_s \leftarrow$ subroutine1$(x, y, n/x_1^d)$ ; //Reduce $|V_s|$ to $n/x_1^d$
21             $J \leftarrow J \cup \{s\}$;
22             $x \leftarrow y$ ; //establishes $f(x) \ge s$
           else
24             break ; //failure on level 1
       else
26         $J' \leftarrow Advance(\ell - 1)$;
27         if $J' \neq \emptyset$ then
28             Reduce the size of the sets $V_j$ with $j \in J'$ to $n/x_\ell^d$ using
28             subroutine2$(x_{\ell-1}, J', n/x_\ell^d, x)$;
29             $J \leftarrow J \cup J'$;
           else
31             break ; //failure on level $\ell$
32 return $J$;
```

$\pi(s^* + 1) \notin \cup_{j=1}^{s^*} V_j$, in which case we can continue with the "increase the score by one, then reduce the size of the candidate sets"-procedure described above.

It is essential to design the levels in such a way that the probability for such failures is small. As we shall see below, we achieve this by designing the levels such that their capacities grow exponentially. More precisely, we set $x_{i+1} = x_i^2$ for all $i = 1, \dots, t-1$. The first level has a capacity of $x_1 = \log n$ sets. Thus, $t = O(\log\log n)$ levels are sufficient if we want the last level, level t, to have a capacity that is linear in n. Our choice of the x_i's also guarantees that x_i divides x_{i+1} and hence a call *Advance*(ℓ) returns no more than x_ℓ elements (exactly x_ℓ elements if the call ends without a failure).

Our strategy is formalized by Algorithm 1. In what follows, we first present the two subroutines, `subroutine1` and `subroutine2`. In Section B.4, we present the full proof of Theorem 3.

B.2 Subroutine 1

`subroutine1` is called by the function *Advance*(1). Simulating a randomized binary search, it allows us to reduce the number of potential candidates for $\pi(i)$ from some value $v < n$ to some value $\ell < v$ in $\log v - \log \ell$ queries.

Algorithm 2: The algorithm `subroutine1`(x, y, ℓ) reduces the size of the set $V_{f(x)+1}$ from v to ℓ in $\log v - \log \ell$ queries.

```
1 Input: Two strings $x, y \in \{0,1\}^n$ with $f(x) < f(y)$. An integer $\ell \in \mathbb{N}$.
2 Initialization: $V \leftarrow \{j \in [n] \mid x_j \neq y_j\}$ ; //potential candidates for $\pi(f(x)+1)$
3 while $|V| > \ell$ do
4     Uniformly at random select a subset $F \subseteq V$ of size $|V|/2$;
5     Create $y'$ from $y$ by flipping all bits in $F$ and query $f(y')$;
6     if $f(y') > f(x)$ then $V \leftarrow V \setminus F$;
7     else $V \leftarrow F$;
8 Output: Set $V$ of size at most $\ell$.
```

Lemma 9. *Let $x, y \in \{0,1\}^n$ with $f(x) < f(y)$. Set $V := \{j \in [n] \mid x_j \neq y_j\}$ and $v := |V|$. Let $\ell \in \mathbb{N}$, $\ell < v$. Algorithm 2 reduces the size of V to ℓ using at most $\lceil \log v - \log \ell \rceil$ queries.*

Proof. For the correctness of the algorithm we note that, by definition, we have $x_{\pi(i)} = y_{\pi(i)} = z_{\pi(i)}$ for all $i \in [f(x)]$. Therefore, either we have $f(y') > f(x)$ in line 5 or we have $f(y') = f(x)$. In the former case, the bit $y_{\pi(f(x)+1)}$ was not flipped, and hence $\pi(f(x)+1) \notin F$ must hold. In the latter case the bit in position $\pi(f(x)+1)$ bit must have been flipped and we can infer $\pi(f(x)+1) \in F$.

The runtime bound is easily verified using the fact that the size of the set V halves in each iteration. □

It is now obvious how `subroutine1` (with the "select uniformly at random select"-statement in line 4 replaced by "arbitrarily select") yields a deterministic $O(n \log n)$ query algorithm for the LeadingOnes-game. This is Algorithm 3.

Algorithm 3: A deterministic $O(n \log n)$ winning strategy for the LeadingOnes-game.

```
1 Initialization: $x \leftarrow (0, \ldots, 0)$, $y \leftarrow (1, \ldots, 1)$;
2 Query $f(x)$ and $f(y)$;
3 for $i = 1, ..., n-1$ do
4     if $f(y) > f(x)$ then rename $x$ and $y$;
5     $V \leftarrow$ subroutine1$(x, y, 1)$ ; //$V = \{\pi(i)\}$
6     Update $x$ by flipping $\pi(i)$ and query $f(x)$;
```

Note also that we apply this algorithm in the second phase of Algorithm 1 for identifying the last $\Theta(n/\log n)$ entries of z. This can be done as follows. When we leave the first phase of Algorithm 1, we have $|V_1| = \ldots = |V_q| = 1$ and $f(x) \geq q$. Create y from x by flipping all bits in $[n] \setminus \cup_{i=1}^{q} V_i$ and query $f(y)$. Then jump into the **for**-loop of Algorithm 3. This shows how to determine the remaining $q \in \Theta(n/\log n)$ bits of the target string z in a linear number of additional queries.

As mentioned, we call `subroutine1` also in the first level of Algorithm 1 (line 19) to reduce the size of $V_{f(x)+1}$ to n/x_1^d, or, put differently, to reduce the number of candidates for $\pi(f(x)+1)$ to n/x_1^d. As the initial size of $V_{f(x)+1}$ is at most n, this requires at most $d \log x_1$ queries by Lemma 9.

B.3 Subroutine 2

We describe the second subroutine of Algorithm 1, `subroutine2`. This routine is used to reduce the sizes of the up to $x_{\ell-1}$ candidate sets returned by a recursive call $Advance(\ell - 1)$ from some value $\leq n/x_{\ell-1}^d$ to at most the target size of level ℓ, which is n/x_ℓ^d. As we shall see below, this requires an expected number of $O(1)x_{\ell-1}d(\log x_\ell - \log x_{\ell-1})/\log x_{\ell-1}$ queries.The pseudo-code of `subroutine2` is given in Algorithm 4. `subroutine2` performs a subtask of this reduction: It reduces the sizes of at most k candidate sets to a kth fraction of their original size using at most $O(k)$ queries, where k is a parameter. We use `subroutine2` with parameter $k = x_{\ell-1}$ repeatedly to achieve the full reduction.

`subroutine2` is given a set J of at most k indices and a y with $f(y) \geq \max J$. The goal is to reduce the size of each candidate set V_j, $j \in J$, below a target size m where $m \geq |V_j|/k$ for all $j \in J$. The routine works in phases. Let J be the set of indices of the candidate sets that are still above the target size at the beginning of an iteration. For each $j \in J$, we randomly choose a kth fraction of the potential candidates for $\pi(j)$. That is, we randomly choose a subset $F_j \subseteq V_j$ of size $|V_j|/k$. We create a new bit string y' from some y by flipping all candidates $\cup_{j \in J} F_j$. A condition on the set $(V_j)_{j \in J}$ ensures that we have either $f(y') \geq \max J$ or $f(y') = j - 1$ for some $j \in J$.

In the first case, i.e., if $f(y') \geq \max J$, none of the sets V_j was *hit*, and for all $j \in J$ we can remove the subset F_j from V_j as the elements in F_j are no longer candidates for $\pi(j)$. We call such queries "*off-trials*". An off-trial reduces the size of all sets V_j, $j \in J$, to a $(1 - 1/k)$th fraction of their original size.

If, on the other hand, we have $f(y') = j - 1$ for some $j \in J$, we can replace V_j by set F_j as $\pi(j) \in F_j$ must hold.[3] Since $|F_j| = |V_j|/k \leq m$ by assumption, this set has now been reduced to its target size and we can remove it from J. We further know that for all $h \in J$ with $h < j$ the set V_h was not hit. Thus, we can safely remove F_h from V_h.

We continue in this way until at least half of the indices are removed from J and at least ck off-trials occurred, for some constant c satisfying $(1 - 1/k)^{ck} \leq 1/2$. Consider any j that is still in J. The size of V_j was reduced by a factor $(1 - 1/k)$ at least ck times. Thus its size was reduced to at most half its original size. We now may half k without destroying the invariant $m \geq |V_j|/k$ for $j \in J$. The effect of halving k is that the relative size of the sets F_j will be doubled for the sets V_j that still take part in the reduction process.

We repeat Lemma 4.

Lemma 4. *Let $k \in \mathbb{N}$, let $J \subseteq [n]$ be a set of at most k indices with $V_j \cap V_p = \emptyset$ for $j \in J$ and $p \in [\max J] \setminus \{j\}$, and let $y \in \{0,1\}^n$ be such that $f(y) \geq \max J$, and let $m \in \mathbb{N}$ be such that $m \geq |V_j|/k$ for all $j \in J$.*

[3]This again can be ensured by the somewhat technical condition that $V_j \cap V_p = \emptyset$ for all $j \in J$ and $p \in [\max J] \setminus \{j\}$.

Algorithm 4: `subroutine2`(k, J, m, y). This subroutine is used in the main program to reduce the size of at most k sets V_j, $j \in J$, to a kth fraction of their original size using only $O(k)$ queries.

```
1  Input: Positive integer $k \in \mathbb{N}$, a set $J \subseteq [n]$ with $|J| \leq k$, target size $m \in \mathbb{N}$ with
   $|V_j|/k \leq m$ for all $j \in J$, and a string $y \in \{0,1\}^n$ with $f(y) \geq \max J$. The sets $(V_j)_{j \in J}$
   are pairwise disjoint. $V_j$, $j \in J$, is also disjoint from $V_p$ for any $p \in [\max J] \setminus \{j\}$.
2  for $j \in J$ do if $|V_j| \leq m$ then delete $j$ from $J$ ; //$V_j$ is already small enough
3  while $J \neq \emptyset$ do
4      $o \leftarrow 0$ ; //counts the number of off-trials that do not hit any of the $V_i$
5      $\ell = |J|$ ; //$|V_j|/k \leq m$ for all $j \in J$
6      repeat
7          for $j \in J$ do Uniformly at random choose a subset $F_j \subseteq V_j$ of size $|V_j|/k$;
8          Create $y'$ from $y$ by flipping in $y$ the entries in positions $\cup_{j \in J} F_j$ and query $f(y')$;
9          if $f(y') \geq \max J$ then
10             $o \leftarrow o + 1$ ; //"off"-trial that did not hit any of the $V_j$
11             for $j \in J$ do $V_j \leftarrow V_j \setminus F_j$;
12         else
13             $V_{f(y')+1} \leftarrow F_{f(y')+1}$ ; //set $V_{f(y')+1}$ is hit
14             for $j \in J$ do if $j \leq f(y')$ then $V_j \leftarrow V_j \setminus F_j$;
15         for $j \in J$ do if $|V_j| \leq m$ then delete $j$ from $J$;
16     until $o \geq c \cdot k$ and $|J| \leq \ell/2$ //$c$ is chosen such that $(1 - 1/k)^{ck} \leq 1/2$;
17     $k \leftarrow k/2$;
18 Output: Sets $V_j$ with $|V_j| \leq m$ for all $j \in J$.
```

In expectation it takes $O(k)$ queries until Algorithm 4 has reduced the size of V_j to at most m for each $j \in J$.

Proof. Let c be some constant. We show below that—for a suitable choice of c—after an expected number of at most ck queries both conditions in line 16 are satisfied. Assuming this to hold, we can bound the total expected number of queries until the size of each of the V_js has been reduced to m by

$$\sum_{h=0}^{\log k} ck/2^h < 2ck\,,$$

as desired.

In each iteration of the repeat-loop we either hit an index in J and hence remove it from J or we have an off-trial. The probability of an off-trial is at least $(1-1/k)^k$ since $|J| \leq k$ always. Thus the probability of an off-trial is at least $(2e)^{-1}$ and hence the condition $o \geq ck$ holds after an expected number of $O(k)$ iterations.

As long as $|J| \geq \ell/2$, the probability of an off-trial is at most $(1 - 1/k)^{\ell/2}$ and hence the probability that a set is hit is at least $1 - (1 - 1/k)^{\ell/2}$. Since $\ln(1 - 1/k) \leq -1/k$ we have $(1 - 1/k)^{\ell/2} = \exp(\ell/2 \ln(1 - 1/k)) \leq \exp(-\ell/(2k))$ and hence $1 - (1 - 1/k)^{\ell/2} \geq 1 - \exp(-\ell/(2k)) \geq \ell/(2k)$. Thus the expected number of iterations to achieve $\ell/2$ hits is $O(k)$.

If a candidate set V_j is hit in the repeat-loop, its size is reduced to $|V_j|/k$. By assumption, this is bounded by m. If V_j is never hit, its size is reduced at least ck times by a factor $(1-1/k)$. By choice of c, this is at most half of its original size. Thus after replacing k by $k/2$ we still have $|V_j|/k \leq m$ for $j \in J$. □

Corollary 10. *Let $k \in \mathbb{N}$, J, and y be as in Lemma 4. Let further $d \in \mathbb{N}$ and $x \in \mathbb{R}$ such that $\max_{j \in J} |V_j| = n/x^d$. Let $y \in \mathbb{R}$ with $y > x$.*

Using at most $d(\log y - \log x)/\log k$ calls to Algorithm 4 we can reduce the maximal size $\max_{j \in J} |V_j|$ to n/y^d. The overall expected number of queries needed to achieve this reduction is $O(1)kd(\log y - \log x)/\log k$.

Proof. The successive calls can be done as follows. We first call `subroutine2`$(k, J, n/(kx^d), y)$. By Lemma 4 it takes an expected number of $O(k)$ queries until the algorithm terminates. The sets V_j, $j \in J$, now have size at most $n/(kx^d)$. We next call `subroutine2`$(k, J, n/(k^2x^d), y)$. After the hth such call we are left with sets of size at most $n/(k^hx^d)$. For $h = d(\log y - \log x)/\log k$ we have $k^h \geq (y/x)^d$. The total expected number of queries at this point is $O(1)kd(\log y - \log x)/\log k$. □

B.4 Proof of Theorem 3

It remains to show that the first phase of Algorithm 1 takes at most $O(n \log\log n)$ queries.

Theorem 11. *Let $q \in n - \Theta(n/\log n)$. Using Algorithm 1, we can identify positions $\pi(1), \ldots, \pi(q)$ and the corresponding entries $z_{\pi(1)}, \ldots, z_{\pi(q)}$ of z in these positions using at most $O(n \log\log n)$ queries.*

We prove Theorem 11. The proof of the required probabilistic statements is postponed to Sections B.5 and B.6.

If there is no failure in any call of *Advance*, the expected number of queries is bounded by

$$\begin{aligned}
&\frac{q}{x_t}\left(\frac{x_t \log n}{\log x_t} + \frac{x_t}{x_{t-1}}\left(\frac{x_{t-1}dc(\log x_t - \log x_{t-1})}{\log x_{t-1}}\right.\right.\\
&\qquad\qquad\qquad \left.\left. + \frac{x_{t-1}}{x_{t-2}}\left(\ldots + \frac{x_2}{x_1}\left(\frac{x_1 dc(\log x_2 - \log x_1)}{\log x_1} + x_1 d \log x_1\right)\right)\right)\right)\\
\leq\, & ndc\left(\frac{\log n}{\log x_t} + \frac{\log x_t}{\log x_{t-1}} + \ldots + \frac{\log x_2}{\log x_1} + \log x_1 - t\right), \qquad (3)
\end{aligned}$$

where c is the constant hidden in the $O(1)$-term in Corollary 10. To verify this formula, observe that each call of *Advance*(i) makes x_i/x_{i-1} calls to *Advance*$(i-1)$. After each such call we reduce the size of x_{i-1} candidate sets from n/x_{i-1}^d to n/x_i^d. By Corollary 10, this requires at most $x_{i-1}dc(\log x_i - \log x_{i-1})/\log x_{i-1}$ queries. The additional $x_1 d \log x_1$ term accounts for the queries caused by the calls *Advance*(1) through which we reduce the sizes of the V_is by the randomized binary search algorithm, `subroutine1`, to n/x_1^d—requiring $d \log x_1$ queries per call. Finally, the term $x_t \log n/\log x_t$ accounts for the final reduction of the V_is to a set containing only one single element (at this stage we shall finally have $V_i = \{\pi(i)\}$). More precisely, this term is $(x_t(\log n - d \log x_t))/\log x_t$ but we settle for upper bounding this expression by the term given in the formula.

Next we need to bound the number of queries caused by *failures*. In Sections B.5 and B.6 we show that, on average, not too many failures happen. More precisely, we show that the expected number of level-i failures is at most $n^2/((n-q)(x_i^{d-1}-1))$. By Corollary 10, each such level-i failure causes an additional number of at most $1 + x_i dc(\log x_{i+1} - \log x_i)/\log x_i$ queries; the 1 counts for the failured query in line 15. Thus, in total we get an additional number of at most

$$\sum_{i=1}^{t} \frac{n^2}{(n-q)(x_i^{d-1}-1)}\left(1 + \frac{x_i dc(\log x_{i+1} - \log x_i)}{\log x_i}\right) \qquad (4)$$

expected queries caused by failures.

We recall the settings of the x_i: We set $x_1 := \log n$ and we choose the x_j such that $x_j = x_{j-1}^2$, $j = 2, \ldots, t$. We further require that $\log x_t = \Omega(\log n)$, which can be achieved by choosing $t \in \Theta(\log\log n)$. In what follows, we do not specify the choice of d, but note that any choice $d \geq 4$ is good enough. With this parameter setting, formula (3) evaluates to

$$ndc\left(\frac{\log n}{\log x_t} + 2(t-1) + \log\log n - t\right) = O(n\log\log n)$$

and, somewhat wasteful, we can bound formula (4) from above by

$$\frac{n^2dc}{n-q}\sum_{i=1}^{t} x_i^{-(d-3)} = O(n\log n)\sum_{i=0}^{t-1} x_1^{-(d-3)2^i} < O(n\log n)(x_1^{d-3}-1)^{-1} = O(n)\,,$$

where the first equation is by construction of the x_is, the inequality uses the fact that the geometric sum is dominated by the first term, and the last equality stems from our choice $x_1 = \log n$. This shows that the overall expected number of queries sums to $O(n\log\log n) + O(n) = O(n\log\log n)$.

Key to the failure analyses in Sections B.5 and B.6 is the following observation.

Lemma 12. *Each V_j has the property that $V_j \setminus \{\pi(j)\}$ is random, i.e., is a random subset of $[n] \setminus \{\pi(j)\}$ of size $|V_j| - 1$.*

Proof. V_j is initialized to $[n]$; thus the claim is true initially. In `subroutine1`, a random subset F of V_j is chosen and V_j is reduced to F (if $\pi(j) \in F$) or to $V_j \setminus F$ (if $\pi(j) \notin F$). In either case, the claim stays true. The same reasoning applies to `subroutine2`. $\square$

B.5 Failures on Level One

Lemma 13. *A call of Advance(1) (i.e., lines 14 to 23) requires at most $x_1 + x_1 d\log x_1$ queries.*

Proof. The two occasions where queries are made are in line 15 and in line 19. Line 15 is executed at most x_1 times, each time causing exactly one query. As shown in Lemma 9, each call to `subroutine1` in line 19 causes at most $d\log x_1$ queries. The subroutine is called at most x_1 times. $\square$

Lemma 14. *Let $q \in n - \Theta(n/\log n)$ be the number of indices i for which we determine $\pi(i)$ and $z_{\pi(i)}$ in the first phase.*

The probability that any particular call of Advance(1) fails is at most $n(n-q)^{-1}\sum_{i=1}^{t} x_i^{-(d-1)}$.

Proof. A failure happens if we cannot increase the score of x by flipping the bits in $[n] \setminus \cup_{i=1}^{s} V_i$, i.e., if $f(y) \leq s$ holds in line 15 of Algorithm 1. This is equivalent to $\pi(s+1) \in \cup_{i=1}^{s} V_i$.

Let ℓ be the number of indices $i \in [n]$ for which V_i is on the last level; i.e., $\ell := |\{i \in [n] \mid |V_i| = 1\}|$ is the number of sets V_i which have been reduced to singletons already. Note that these sets satisfy $V_i = \{\pi(i)\}$. Therefore, they cannot contain $\pi(s+1)$ and we do not need to take them into account.

By our random construction of the V_is (cf. Lemma 12), the probability that $\pi(s+1) \in \cup_{i=1}^{s} V_i$ is at most $|\cup_{i=\ell+1}^{s} V_i|/(n-\ell)$ and hence bounded by $\sum_{i=\ell+1}^{s} |V_i|/(n-\ell)$.

At any time during the run of Algorithm 1, there are at most x_i sets V_j on the ith level. By construction, the size of each such level-i set is at most nx_i^{-d}. Hence, we can bound the probability that $\pi(s+1) \in \cup_{i=1}^{s} V_i$ from above by

$$\frac{n}{n-\ell} \sum_{i=1}^{t} x_i^{-(d-1)} .$$

□

By the exponential growth of the values $x_1, \dots, x_t$ and the fact that we call line 15 a total number of $q < n$ times to improve upon the current best score, from Lemma 14 we immediately get the following.

Corollary 15. *The expected number of level* 1 *failures is less than*

$$qn(n-q)^{-1}(x_1^{d-1}-1)^{-1} \leq n^2(n-q)^{-1}\left(x_1^{d-1}-1\right)^{-1} .$$

B.6 Failures at Higher Levels

Lemma 16. *As in Lemma 14 let q be the number of indices i for which we determine $\pi(i)$ and $z_{\pi(i)}$ in the first phase. Let $i \in [t]$.*

The probability that a particular call of Advance(i) fails (level i failure) is at most $n(n-q)^{-1}\left(x_i^{d-1}-1\right)^{-1}$.

Proof. This proof is similar to the one of Lemma 14: A failure on level i occurs only if $\pi(s+1) \in \cup_{j=1}^{s} V_j$ and the size of all V_js has been reduced already to at most n/x_i^d. There are at most x_j sets V on each level $j \geq i$. The size of each level-j V is at most n/x_j^d, by construction. By Lemma 12, the probability that $\pi(s+1) \in \cup_{j=1}^{s} V_j$ is at most

$$\frac{n}{n-\ell} \sum_{j=i}^{t} \frac{1}{x_j^{d-1}} , \tag{5}$$

where ℓ denotes again the number of sets that have been reduced to singletons already.

By definition, we have $x_j \geq x_i^{(2^{j-i})}$ and in particular we have $x_j \geq x_i^{j-i}$. Therefore expression (5) can be bounded from above by

$$\frac{n}{n-\ell} \sum_{j=1}^{t-i} \left(\frac{1}{x_i^{d-1}}\right)^j < \frac{n}{n-\ell}\left(x_i^{d-1}-1\right)^{-1} .$$

□

By the same reasoning as in Section B.5, from Lemma 16 we immediately get the following.

Corollary 17. *Let $i \in [t]$.*

The expected number of level i failures is less than

$$nq(n-q)^{-1}(x_i^{d-1}-1)^{-1} \leq n^2(n-q)^{-1}(x_i^{d-1}-1)^{-1} .$$

C Details on the Lower Bound

C.1 Candidate Sets in the Lower Bound Context

At node v of the decision tree, the candidate set V_i^v, intuitively corresponds to the possible values of $\pi(i)$. At the root node r, we have $V_i^r = [n]$ for all i. Let v be a node in the tree and let $w_0, \ldots, w_n$ be its children (w_i is traversed when the score i is returned). Let P_0^v (resp. P_1^v) be the set of positions in x_v that contain 0 (resp. 1). Thus, formally, $P_0^v = \{i \mid x_v[i] = 0\}$ and $P_1^v = \{i \mid x_v[i] = 1\}$.[4] The precise definition of candidate sets is as follows:

$$V_i^{w_t} = \begin{cases} V_i^v \cap P_{i \bmod 2}^v & \text{if } i \le t \\ V_i^v \cap P_{t \bmod 2}^v & \text{if } i = t+1 \\ V_i^v & \text{if } i > t+1. \end{cases}$$

Note that, as was the case with the candidate sets defined in Section 2, not all values in a candidate set could be valid. But as with the upper bound case, the candidate sets have some very useful properties. These properties are also slightly different from the ones observed before, due to the fact that some extra information has been announced to the query algorithm. We reiterate the definition of an active candidate set. We say that a candidate set V_i^v is *active (at v)* if the following conditions are met: (i) at some ancestor node u of v, we have $F(x_u) = i - 1$ and (ii) at every ancestor node w of u, we have $F(x_w) < i - 1$, and (iii) $i < \min\{n/3, \max_v\}$. We call $V_{\max_v + 1}^v$ *pseudo-active (at v).* The following theorem can be proved using similar ideas as in Theorem 1

Lemma 6. *The candidate sets have the following properties:*

(i) Two candidate sets V_i^v and V_j^v with $i < j \le \max_v$ and $i \not\equiv j$ are disjoint.

(ii) An active candidate set V_j^v is disjoint from any candidate set V_i provided $i < j < \max_v$.

(iii) The candidate set V_i^v, $i \le \max_v$ is contained in the set $V_{\max_v + 1}^v$ if $i \equiv \max_v$ and is disjoint from it if $i \not\equiv \max_v$.

(iv) For two candidate sets V_i^v and V_j^v, $i < j$, if $V_i^v \cap V_j^v \ne \emptyset$ then $V_i^v \subset V_j^v$.

Proof. Let w be the ancestor of v where the function returns score $\max_v$.

To prove (i), observe that in w, one of V_i^w and V_j^w is intersected by P_0^w while the other is intersected by P_1^w and thus they are made disjoint.

To prove (ii), we can assume $i \equiv j$ as otherwise the result follows from the previous case. Let u be the ancestor of v such that $F(x_u) = j - 1$ and that in any ancestor of u, the score returned by the function is smaller than $j - 1$. At u, V_j^u is intersected with $P_{j-1 \bmod 2}^u$ while V_i^v is intersected with $P_{i \bmod 2}^u$. Since $i \equiv j$, it follows that they are again disjoint.

For (iii), the latter part follows as in (i). Consider an ancestor v' of v and let w_j be the jth child of v' that is also an ancestor of v. We use induction and we assume $V_i^{v'} \subset V_{\max_v + 1}^{v'}$. If $j < \max_v$, then $V_{\max_v+1}^{v'} = V_{\max_v+1}^{w_j}$ which means $V_i^{w_j} \subset V_{\max_v+1}^{w_j}$. If $j = \max_v$, then $V_{\max_v+1}^{w_j} = V_{\max_v+1}^{v'} \cap P_{\max_v \bmod 2}^{v'}$ and notice that in this case also $V_i^{w_j} = V_i^{v'} \cap P_{i \bmod 2}^{v'}$ which still implies $V_i^{w_j} \subset V_{\max_v+1}^{w_j}$.

To prove (iv), first observe that the statement is trivial if $i \not\equiv j$. Also, if the function returns score $j - 1$ at any ancestor of v, then by the same argument used in (ii) it is possible to show that $V_i^v \cap V_j^v = \emptyset$. Thus assume $i \equiv j$ and the function never returns value $j - 1$. In this case, it is easy to see that an inductive argument similar to (iii) proves that $V_i^{v'} \subset V_j^{v'}$ for every ancestor v' of v. □

[4] To prevent our notations from becoming too overloaded, here and in the remainder of the section we write $x = (x[1], \ldots, x[n])$ instead of $x = (x_1, \ldots, x_n)$

Corollary 18. *Every two distinct active candidate sets V_i^v and V_j^v are disjoint.*

Lemma 19. *Consider a candidate set V_i^v and let $i_1 < \cdots < i_k < i$ be the indices of candidate sets that are subsets of V_i^v. Let $\sigma := (\sigma_1, \cdots, \sigma_i)$ be a sequence without repetition from $[n]$ and let $\sigma' := (\sigma_1, \cdots, \sigma_{i-1})$. Let n_σ and $n_{\sigma'}$ be the number of permutations in S_v that have σ and σ' as a prefix, respectively. If $n_\sigma > 0$, then $n_{\sigma'} = (|V_i^v| - k)n_\sigma$.*

Proof. Consider a permutation $\pi \in S_v$ that has σ as a prefix. This implies $\pi(i) \in V_i^v$. For an element $s \in V_i^v$, $s \neq i_j$, $1 \leq j \leq k$, let π_s be the permutation obtained from π by placing s at position i and placing $\pi(i)$ where s used to be. Since $s \neq i_j$, $1 \leq j \leq k$, it follows that π_s has σ' as prefix and since $s \in V_i^v$ it follows that $\pi_s \in S_v$. It is easy to see that for every permutation in S_v that has σ as a prefix we will create $|V_i^v| - k$ different permutations that have σ' as a prefix and all these permutations will be distinct. Thus, $n_{\sigma'} = (|V_i^v| - k)n_\sigma$. □

Corollary 20. *Consider a candidate set V_i^v and let $i_1 < \cdots < i_k < i$ be the indices of candidate sets that are subsets of V_i^v. Let $\sigma' := (\sigma_1, \cdots, \sigma_{i-1})$ be a sequence without repetition from $[n]$ and let $\sigma_1 := (\sigma_1, \cdots, \sigma_{i-1}, s_1)$ and $\sigma_2 := (\sigma_1, \cdots, \sigma_{i-1}, s_2)$ in which $s_1, s_2 \in V_i^v$. Let n_{σ_1} and n_{σ_2} be the number of permutations in S_v that have σ_1 and σ_2 as a prefix, respectively. If $n_{\sigma_1}, n_{\sigma_2} > 0$, then $n_{\sigma_1} = n_{\sigma_2}$.*

Proof. Consider a sequence $s_1, \cdots, s_i$ without repetition from $[n]$ such that $s_j \in V_j^v$, $1 \leq j \leq i$. By the previous lemma $\Pr[\Pi(1) = s_1 \wedge \cdots \wedge \Pi(i-1) = s_{i-1} \wedge \Pi(i) = s_i] = \Pr[\Pi(1) = s_1 \wedge \cdots \wedge \Pi(i-1) = s_{i-1}] \cdot (1/|V_i^v|)$. □

Corollary 21. *If V_i^v is active, then we have:*

(i) $\Pi(i)$ *is independent of* $\Pi(1), \cdots, \Pi(i-1)$.

(ii) $\Pi(i)$ *is uniformly distributed in* V_i^v.

C.2 Potential Function Analysis

The main result of this section is the following.

Lemma 7. *Let v be a node in T and let $\mathbf{i}_v$ be the random variable giving the value of $F(x_v)$ when $\Pi \in S_v$ and 0 otherwise. Also let $w_0, \ldots, w_n$ denote the children of v, where w_j is the child reached when $F(x_v) = j$. Then, $\mathrm{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = O(1)$.*

We write

$$\mathrm{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = \sum_{a=0}^{n} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)).$$

Before presenting the formal analysis of this potential function, we note that we have two main cases: $F(x_v) \leq \max_v$ and $F(x_v) > \max_v$. In the first case, the maximum score will not increase in w_a which means w_a will have the same set of active candidate sets. In the second case, the pseudo-active candidate set $V_{\max_v+1}^v$ will turn into an active set $V_{\max_v+1}^{w_a}$ at w_a and w_a will have a new pseudo-active set. While this second case looks more complicated, it is in fact the less interesting part of the analysis since the probability of suddenly increasing the score by α is extremely small (we will show that it is roughly $O(2^{-\Omega(\alpha)})$) which subsumes any significant potential increase for values of $a > \max_v$. As discussed, we divide the above summation into two parts: one for $a \leq \max_v$ and another for $a > \max_v$:

$$\mathrm{E}[\varphi(w_{\mathbf{i}_v}) - \varphi(v) \mid \Pi \in S_v] = \sum_{a=0}^{\max_v} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)) + \tag{6}$$

$$\sum_{a=0}^{n/3-\max_v -1} \Pr[F(x_v) = \max_v + 1 + a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)). \tag{7}$$

To bound the above two summations, it is clear that we need to know how to handle $\Pr[F(x_v) = a|\Pi \in S_v]$. In the next section, we will prove lemmas that will do this.

C.2.1 Bounding Probabilities

Let $a_1, \ldots, a_{|A_v|}$ be the indices of active candidate sets at v sorted in increasing order. We also define $a_{|A_v|+1} = \max_v + 1$. For a candidate set V_i^v, and a Boolean $b \in \{0,1\}$, let $V_i^v(b) = \{j \in V_i^v \mid x_v[j] = b\}$. Clearly, $|V_i^v(0)| + |V_i^v(1)| = |V_i^v|$. For even a_i, $1 \leq i \leq |A_v|$, let $\varepsilon_i = |V_i^v(1)|/|V_i^v|$ and for odd i let $\varepsilon_i = |V_i^v(0)|/|V_i^v|$. This definition might seem strange but is inspired by the following observation.

Lemma 22. *For $i \leq |A_v|$, $\Pr[F(x_v) = a_i - 1|\Pi \in S_v \wedge F(x_v) > a_i - 2] = \varepsilon_i$.*

Proof. Note that $F(x_v) = a_i - 1$ happens if and only if $F(x_v) > a_i - 2$ and $x_v[\Pi(a_i)] \not\equiv a_i$. Since $V_{a_i}^v$ is an active candidate set, the lemma follows from Corollary 21 and the definition of ε_i. □

Let $\varepsilon'_i := \Pr[a_i \leq F(x_v) < a_{i+1} - 1|\Pi \in S_v \wedge F(x_v) \geq a_i]$. Note that $\varepsilon'_i = 0$ if $a_{i+1} = a_i + 1$.

Lemma 23. *For $i \leq |A_v|$ we have $|V_{a_i}^{w_j}| = |V_{a_i}^v|$ for $j < a_i - 1$, $|V_{a_i}^{w_j}| = \varepsilon_i |V_{a_i}^v|$ for $j = a_i - 1$, and $|V_{a_i}^{w_j}| = (1-\varepsilon_i)|V_{a_i}^v|$ for $j > a_i - 1$. Also,*

$$\Pr[F(x_v) = a_i - 1|\Pi \in S_v] = \varepsilon_i \Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j), \tag{8}$$

$$\Pr[a_i \leq F(x_v) < a_{i+1} - 1|\Pi \in S_v] = \varepsilon'_i(1-\varepsilon_i)\Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j), \tag{9}$$

Proof. Using Lemma 22, it is verified that

$$\begin{aligned}
&\Pr[F(x_v) > a_i - 1|\Pi \in S_v] = \Pr[F(x_v) > a_i - 2 \wedge F(x_v) \neq a_i - 1|\Pi \in S_v] \\
&= \Pr[F(x_v) \neq a_i - 1|F(x_v) > a_i - 2 \wedge \Pi \in S_v] \Pr[F(x_v) > a_i - 2|\Pi \in S_v] \\
&= (1-\varepsilon_i) \Pr[F(x_v) > a_i - 2|\Pi \in S_v].
\end{aligned}$$

Similarly, using the definition of ε'_i we can see that

$$\begin{aligned}
&\Pr[F(x_v) > a_i - 2|\Pi \in S_v] = \Pr[F(x_v) \notin \{a_{i-1}, \ldots, a_i - 2\} \wedge F(x_v) > a_{i-1} - 1|\Pi \in S_v] \\
&= \Pr[F(x_v) \notin \{a_{i-1}, \ldots, a_i - 2\} \,|F(x_v) > a_{i-1} - 1 \wedge \Pi \in S_v] \Pr[F(x_v) > a_{i-1} - 1|\Pi \in S_v] \\
&= (1-\varepsilon'_{i-1}) \Pr[F(x_v) > a_{i-1} - 1|\Pi \in S_v].
\end{aligned}$$

Using these, we get that

$$\Pr[F(x_v) > a_i - 1|\Pi \in S_v] = (1-\varepsilon_i)\Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)$$

and

$$\Pr[F(x_v) > a_i - 2|\Pi \in S_v] = \Pi_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j).$$

Equalities (8) and (9) follow from Lemma 19 and Lemma 22. The rest of the lemma follows directly from the definition of ε_i and the candidate sets. □

Lemma 24. *Let $b \in \{0,1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v+1}(b)|$. Then,*

$$\Pr[F(x_v) = \max_v | \Pi \in S_v] = \frac{k - \mathrm{Con}_v}{|V^v_{\max_v+1}| - \mathrm{Con}_v} \prod_{i=1}^{|A_v|} (1-\varepsilon_i)(1-\varepsilon'_i).$$

Proof. Conditioned on $F(x_v) > \max_v - 1$, $F(x_v)$ will be equal to $\max_v$ if $x_v[\Pi(\max_v+1)] = b$. By definition, the number of positions in $V^v_{\max_v+1}$ that satisfy this is k. However, $V^v_{\max_v+1}$ contains Con_v candidate sets but since $V^v_{\max_v+1}$ can only contain a candidate set V^v_i if $i \equiv \max_v$ (by Lemma 6), it follows from Lemma 19 that $\Pr[F(x_v) = \max_v | \Pi \in S_v \wedge F(x_v) > \max_v - 1] = (k - \mathrm{Con}_v)/(|V^v_{\max_v+1}| - \mathrm{Con}_v)$. The lemma then follows from the previous lemma. □

Lemma 25. *Let $b \in \{0,1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v+1}(b)|$. Then,*

$$\Pr[F(x_v) > \max_v | \Pi \in S_v] \le \frac{|V^v_{\max_v+1}| - k}{|V^v_{\max_v+1}| - \mathrm{Con}_v}$$

Proof. From the previous lemma we have that $\Pr[F(x_v) = \max_v | \Pi \in S_v \wedge F(x_v) > \max_v - 1] = (k - \mathrm{Con}_v)/(|V^v_{\max_v+1}| - \mathrm{Con}_v)$. Thus,

$$\Pr[F(x_v) > \max_v | \Pi \in S_v \wedge F(x_v) > \max_v - 1] = \frac{|V^v_{\max_v+1}| - k}{|V^v_{\max_v+1}| - \mathrm{Con}_v}.$$

Using induction, it is possible to prove that

$$\Pr[F(x_v) > \max_v | \Pi \in S_v] = \frac{|V^v_{\max_v+1}| - k}{|V^v_{\max_v+1}| - \mathrm{Con}_v} \prod_{i=1}^{|A_v|} (1-\varepsilon_i)(1-\varepsilon'_i)$$

which implies the lemma. □

Remember that P^v_0 (resp. P^v_1) are the set of positions in x_v that contain 0 (resp. 1).

Lemma 26. *Let $x_0 = |P^v_0|$ and $x_1 = |P^v_1|$. Let b_i be a Boolean such that $b_i \equiv i$. For $a \ge \max_v + 2$*

$$\Pr[F(x_v) = a \mid \Pi \in S_v] \le \left(\prod_{i=\max_v+2}^{a} \frac{x_{b_i} - \lfloor i/2 \rfloor}{n - i + 1} \right) \left(1 - \frac{x_{b_{a+1}} - \lfloor (a+1)/2 \rfloor}{n - a} \right).$$

Proof. Notice that we have $V^v_i = [n]$ for $i \ge \max_v + 2$ which means $i - 1$ is the number of candidates sets V^v_j contained in V^v_i and among those $\lfloor i/2 \rfloor$ are such that $i \equiv j$. Consider a particular prefix $\sigma = (\sigma_1, \cdots, \sigma_{i-1})$ such that there exists a permutation $\pi \in S_v$ that has σ as a prefix. This implies that $\sigma_j \in P^v_{b_j}$. Thus, it follows that there are $x_{b_i} - \lfloor i/2 \rfloor$ elements $s \in P^v_{b_i}$ such that the sequences $(\sigma_1, \cdots, \sigma_{i-1}, s)$ can be the prefix of a permutation in S_v. Thus by Corollary 20, and for $i \ge \max_v + 2$,

$$\Pr[F(x_v) = i - 1 | \Pi \in S_v \wedge F(x_v) \ge i - 1] = 1 - \frac{x_{b_i} - \lfloor i/2 \rfloor}{n - i + 1}$$

and

$$\Pr[F(x_v) \ge i | \Pi \in S_v \wedge F(x_v) \ge i - 1] = \frac{x_{b_i} - \lfloor i/2 \rfloor}{n - i + 1}.$$

□

Corollary 27. *For* $\max_v + 1 \leq a \leq n/3$ *we have*

$$\Pr[F(x_v) = a \mid \Pi \in S_v] = 2^{-\Omega(a-\max_v)} \cdot \left(1 - \frac{x_{b_{a+1}} - \lfloor (a+1)/2 \rfloor}{n-a}\right).$$

Proof. Since $x_{b_i} + x_{b_{i+1}} = n$, it follows that

$$\left(\frac{x_{b_i} - \lfloor i/2 \rfloor}{n-i+1}\right)\left(\frac{x_{b_{i+1}} - \lfloor i/2 \rfloor}{n-i+2}\right) \leq \left(\frac{x_{b_i} - \lfloor i/2 \rfloor}{n-i+1}\right)\left(\frac{x_{b_{i+1}} - \lfloor i/2 \rfloor}{n-i+1}\right) \leq \frac{1}{2}.$$

□

Now we analyze the potential function.

C.2.2 Bounding (6)

We have,

$$\varphi(w_a) - \varphi(v) = \log \frac{\log \frac{2n}{|V^{w_a}_{\max_{w_a}+1}| - \mathrm{Con}_{w_a}}}{\log \frac{2n}{|V^{v}_{\max_v+1}| - \mathrm{Con}_v}} + \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V^{w_a}_j|}}{\log \frac{2n}{|V^v_j|}}. \tag{10}$$

When $a \leq \max_v$ we have, $\max_v = \max_{w_a}$ and $\mathrm{Con}_v = \mathrm{Con}_{w_a}$. For $a < \max_v$, we also have $V^{w_a}_{\max_v+1} = V^v_{\max_v+1}$. It is clear from (10) that for $a_i \leq a < a_{i+1} - 1$, all the values of $\varphi(w_a) - \varphi(v)$ will be equal. Thus,

$$\begin{aligned}
(6) \quad = \quad & \sum_{i=1}^{|A_v|} \Pr[F(x_v) = a_i - 1 | \Pi \in S_v](\varphi(w_{a_i - 1}) - \varphi(v)) + && (11)\\
& \sum_{i=1}^{|A_v|} \Pr[a_i \leq F(x_v) < a_{i+1} - 1 | \Pi \in S_v](\varphi(w_{a_i}) - \varphi(v)) + && (12)\\
& \Pr[F(x_v) = \max_v | \Pi \in S_v](\varphi(w_{\max_v}) - \varphi(v)). && (13)
\end{aligned}$$

Analyzing (11). We write (11) using (10) and Lemma 23(8). Using inequalities, $1 - x \leq e^{-x}$, for $0 \leq x \leq 1$, $\log(1+x) \leq x$ for $x \geq 0$, and $\sum_{1 \leq i \leq k} y_i \log 1/y_i \leq Y \log(k/Y)$ for $y_i \geq 0$ and

$Y = \sum_{1 \le i \le k} y_i$, we get the following:

$$
\begin{aligned}
(11) = \sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\sum_{j=1}^{i} \log \frac{\log \frac{2n}{|V_{a_j}^{w_{a_i-1}}|}}{\log \frac{2n}{|V_{a_j}^{v}|}} \right) &= \\
\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\log \frac{\log \frac{2n}{|V_{a_i}^{w_{a_i-1}}|}}{\log \frac{2n}{|V_{a_i}^{v}|}} + \sum_{j=1}^{i-1} \log \frac{\log \frac{2n}{|V_{a_j}^{w_{a_i-1}}|}}{\log \frac{2n}{|V_{a_j}^{v}|}} \right) &= \\
\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\log \frac{\log \frac{2n}{|V_{a_i}^{v}|} + \log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^{v}|}} + \sum_{j=1}^{i-1} \log \frac{\log \frac{2n}{|V_{a_j}^{v}|} + \log \frac{1}{1-\varepsilon_j}}{\log \frac{2n}{|V_{a_j}^{v}|}} \right) &= \\
\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j)(1-\varepsilon'_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^{v}|}} \right) + \sum_{j=1}^{i-1} \log \left(1 + \frac{\log \frac{1}{1-\varepsilon_j}}{\log \frac{2n}{|V_{a_j}^{v}|}} \right) \right) &\le \\
\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^{v}|}} \right) + \sum_{j=1}^{i-1} \log \left(1 + \log \frac{1}{1-\varepsilon_j} \right) \right) &\le \\
\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j) \left(\log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^{v}|}} \right) + \sum_{j=1}^{i-1} \log \frac{1}{1-\varepsilon_j} \right) &= \\
\sum_{i=1}^{|A_v|} \varepsilon_i \log \left(1 + \frac{\log \frac{1}{\varepsilon_i}}{\log \frac{2n}{|V_{a_i}^{v}|}} \right) \prod_{j=1}^{i-1} (1-\varepsilon_j) \, + & \qquad (14) \\
\sum_{i=1}^{|A_v|} \varepsilon_i \prod_{j=1}^{i-1} (1-\varepsilon_j) \left(\sum_{j=1}^{i-1} \log \frac{1}{1-\varepsilon_j} \right) . & \qquad (15)
\end{aligned}
$$

To bound (14), we use the fact that any two active candidate sets are disjoint. We break the summation into smaller chunks. Observe that $\prod_{j=1}^{i-1}(1-\varepsilon_j) \le e^{-\sum_{j=1}^{i-1} \varepsilon_j}$. Thus, let J_t, $t \ge 0$, be the set of indices such that for each $i \in J_t$ we have $2^t - 1 \le \sum_{j=1}^{i-1} \varepsilon_j < 2^{t+1}$. Now define $J_{t,k} = \left\{ i \in J_t \mid n/2^{k+1} \le |V_{a_i}^v| \le n/2^k \right\}$, for $0 \le k \le \log n$ and let $s_{t,k} = \sum_{i \in J_{t,k}} \varepsilon_i$. Observe that by the disjointness of two active candidate sets, $|J_{t,k}| \le 2^{k+1}$.

$$(14) = \sum_{t=0}^{\log n}\sum_{k=1}^{\log n}\sum_{i\in J_{t,k}} \varepsilon_i \log\left(1+\frac{\log\frac{1}{\varepsilon_i}}{\log\frac{2n}{|V^v_{a_i}|}}\right)\prod_{j=1}^{i-1}(1-\varepsilon_j) \le$$
$$\sum_{t=0}^{n}\sum_{k=1}^{\log n}\sum_{i\in J_{t,k}} \varepsilon_i \log\left(1+\frac{\log\frac{1}{\varepsilon_i}}{k}\right)e^{-\sum_{j=1}^{i-1}\varepsilon_j} \le$$
$$\sum_{t=0}^{n}\sum_{k=1}^{\log n}\sum_{i\in J_{t,k}} \varepsilon_i \frac{\log\frac{1}{\varepsilon_i}}{k}e^{-2^t+1} \le \sum_{t=0}^{n}\sum_{k=1}^{\log n}\frac{s_{t,k}\log\frac{|J_{t,k}|}{s_{t,k}}}{k}e^{-2^t+1} \le$$
$$\sum_{t=0}^{n}\sum_{k=1}^{\log n}\frac{s_{t,k}(k+1)+s_{t,k}\log\frac{1}{s_{t,k}}}{k}e^{-2^t+1} \le \sum_{t=0}^{n}2^{t+2}e^{-2^t+1} + \sum_{t=0}^{n}\sum_{k=1}^{\log n}\frac{s_{t,k}\log\frac{1}{s_{t,k}}}{k}e^{-2^t+1} \le$$
$$O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\lg\lg n}\sum_{k=2^{r-1}}^{2^r}\frac{s_{t,k}\log\frac{1}{s_{t,k}}}{2^{r-1}}e^{-2^t+1}.$$

Now define $S_{t,r} = \sum_{2^{r-1}\le k<2^r} s_{t,k}$. Remember that $\sum_{r=1}^{\log\log n} S_{t,r} < 2^{t+1}$. Thus,

$$(14) \le O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\lg\lg n}\frac{S_{t,r}\log\frac{2^{r-1}}{S_{t,r}}}{2^{r-1}}e^{-2^t+1} =$$
$$O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\lg\lg n}\frac{S_{t,r}(r-1)+S_{t,r}\log\frac{1}{S_{t,r}}}{2^{r-1}}e^{-2^t+1} \le$$
$$O(1)+\sum_{t=0}^{n}\sum_{r=1}^{\lg\lg n}\frac{2^{t+1}(r-1)}{2^{r-1}}e^{-2^t+1} + \sum_{t=0}^{n}\sum_{r=1}^{\lg\lg n}\frac{1}{2^{r-1}}e^{-2^t+1} = O(1).$$

To bound (15), define J_t as before and let $p_i = \prod_{j=1}^{i-1} 1/(1-\varepsilon_j)$. Observe that the function $\log(1/(1-x))$ is a convex function which means if $s_i := \sum_{j=1}^{i-1}\varepsilon_j$ is fixed, then $\prod_{j=1}^{i-1} 1/(1-\varepsilon_j)$ is minimized when $\varepsilon_j = s_i/(i-1)$. Thus,

$$p_i \ge \left(\frac{1}{1-\frac{s_i}{i-1}}\right)^{i-1} \ge \left(1+\frac{s_i}{i-1}\right)^{i-1} \ge 1+\binom{i-1}{j}\left(\frac{s_i}{i-1}\right)^j$$

in which j can be chosen to be any integer between 0 and $i-1$. We pick $j := \max\{\lfloor s_i/8\rfloor, 1\}$. Since $i \ge s_i$, we get for $i \in J_t$,

$$p_i \ge 1+\frac{\left(\frac{i-1}{2}\right)^j}{j^j}\left(\frac{s_i}{i-1}\right)^j \ge\ 1+\left(\frac{s_i}{2j}\right)^j \ge 2^{s_i/8} \ge 2^{2^{t-4}}.$$

Thus, we can write

$$(15) = \sum_{i=1}^{|A_v|}\varepsilon_i\frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t}\varepsilon_i\frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t} O\left(\frac{\varepsilon_i 2^t}{2^{2^{t-4}}}\right) \le$$
$$\sum_{t=0}^{\log n} O\left(\frac{2^{2t+1}}{2^{2^{t-4}}}\right) = O(1).$$

To bound (15), define J_t as before and let $p_i = \prod_{j=1}^{i-1} 1/(1-\varepsilon_j)$. Observe that the function $\log(1/(1-x))$ is a convex function which means if $s_i := \sum_{j=1}^{i-1} \varepsilon_j$ is fixed, then $\prod_{j=1}^{i-1} 1/(1-\varepsilon_j)$ is minimized when $\varepsilon_j = s_i/(i-1)$. Thus,

$$p_i \geq \left(\frac{1}{1-\frac{s_i}{i-1}}\right)^{i-1} \geq \left(1+\frac{s_i}{i-1}\right)^{i-1} \geq 1 + \binom{i-1}{j}\left(\frac{s_i}{i-1}\right)^j$$

in which j can be chosen to be any integer between 0 and $i-1$. We pick $j := \max\{\lfloor s_i/8 \rfloor, 1\}$. Since $i \geq s_i$, we get for $i \in J_t$,

$$p_i \geq 1 + \frac{\left(\frac{i-1}{2}\right)^j}{j^j}\left(\frac{s_i}{i-1}\right)^j \geq \ 1 + \left(\frac{s_i}{2j}\right)^j \geq 2^{s_i/8} \geq 2^{2^{t-4}}.$$

Thus, we can write

$$(15) = \sum_{i=1}^{|A_v|} \varepsilon_i \frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon_i \frac{\log p_i}{p_i} = \sum_{t=0}^{\log n}\sum_{i\in J_t} O\left(\frac{\varepsilon_i 2^t}{2^{2^{t-4}}}\right) \leq \sum_{t=0}^{\log n} O\left(\frac{2^{2t+1}}{2^{2^{t-4}}}\right) = O(1).$$

Analyzing (12). The analysis of this equation is very similar to (11). We can write (12) using (10) and (9). We also use the same technique as in analyzing (15).

$$(12) = \sum_{i=1}^{|A_v|} \varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\frac{\log\frac{2n}{|V_{a_j}^{w_{a_i}}|}}{\log\frac{2n}{|V_{a_j}^{v}|}}\right) =$$
$$\sum_{i=1}^{|A_v|} \varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\frac{\log\frac{2n}{|V_{a_j}^{v}|}+\log\frac{1}{1-\varepsilon_j}}{\log\frac{2n}{|V_{a_j}^{v}|}}\right) =$$
$$\sum_{i=1}^{|A_v|} \varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\left(1+\frac{\log\frac{1}{1-\varepsilon_j}}{\log\frac{2n}{|V_{a_j}^{v}|}}\right)\right) \leq$$
$$\sum_{i=1}^{|A_v|} \varepsilon'_i(1-\varepsilon_i)\prod_{j=1}^{i-1}(1-\varepsilon_j)(1-\varepsilon'_j)\left(\sum_{j=1}^{i}\log\frac{1}{1-\varepsilon_i}\right).$$

Let $s_i := \sum_{j=1}^{i} \varepsilon_j$, and $s'_i := \sum_{j=1}^{i-1} \varepsilon'_j$. Similar to the previous case, let J_t, $t \geq 0$, be the set of indices such that for each $i \in J_t$ we have $2^t - 1 \leq s_i + s'_i \leq 2^{t+1}$. Also define $p_i = \prod_{j=1}^{i} 1/(1-\varepsilon_j)$ and $p'_i = \prod_{j=1}^{i-1} 1/(1-\varepsilon'_j)$. Using the previous techniques we can get that $p_i \geq 2^{s_i/8}$ and $p'_i \geq 2^{s'_i/8}$. We get

$$(12) \leq \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon'_i \frac{\log p_i}{p_i p'_i} \leq \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon'_i \frac{s_i}{8\cdot 2^{s_i/8}2^{s'_i/8}} \leq \sum_{t=0}^{\log n}\sum_{i\in J_t} \varepsilon'_i \frac{2^{t+1}}{8\cdot 2^{(2^t-1)/8}} \leq \sum_{t=0}^{\log n} \frac{2^{2t+2}}{8\cdot 2^{(2^t-1)/8}} = O(1).$$

Analyzing (13). Let $b \in \{0,1\}$ be such that $b \equiv \max_v$ and let $k := |V^v_{\max_v+1}(b)|$. We use Lemma 24. Observe that we still have $\mathrm{Con}_v = \mathrm{Con}_{w_{\max_v}}$. Thus we have,

$$(13) \le \frac{(k - \mathrm{Con}_v)}{|V^v_{\max_v+1}| - \mathrm{Con}_v} \prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon_i')(\varphi(w_{\max_v}) - \varphi(v)) =$$

$$\frac{(k - \mathrm{Con}_v)}{|V^v_{\max_v+1}| - \mathrm{Con}_v} \prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon_i') \left(\log \frac{\log \frac{2n}{|V^{w_{\max_v}}_{\max_v+1}| - \mathrm{Con}_{w_{\max_v}}}}{\log \frac{2n}{|V^v_{\max_v+1}| - \mathrm{Con}_v}} + \sum_{i=1}^{|A_v|} \log \frac{\log \frac{2n}{|V_i^{w_{\max_v}}|}}{\log \frac{2n}{|V_i^v|}} \right) \le$$

$$\frac{(k - \mathrm{Con}_v)}{|V^v_{\max_v+1}| - \mathrm{Con}_v} \prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon_i') \left(\log \frac{\log \frac{2n}{k - \mathrm{Con}_v}}{\log \frac{2n}{|V^v_{\max_v+1}| - \mathrm{Con}_v}} + \sum_{i=1}^{|A_v|} \log\left(1 + \log \frac{1}{1-\varepsilon_i}\right) \right) \le$$

$$\frac{(k - \mathrm{Con}_v)}{|V^v_{\max_v+1}| - \mathrm{Con}_v} \log \frac{\log \frac{2n}{k - \mathrm{Con}_v}}{\log \frac{2n}{|V^v_{\max_v+1}| - \mathrm{Con}_v}} + O(1) = O(1).$$

C.2.3 Bounding (7)

The big difference here is that the candidate set $V^{w_a}_{\max_v+1}$ becomes an active candidate set (if of course $\max_v + 1 < n/3$) at w_a while $V^v_{\max_v+1}$ was not active at v. Because of this, we have

$$\varphi(w_a) - \varphi(v) \le \log\log \frac{2n}{|V^{w_a}_{a+1}| - \mathrm{Con}_{w_a}} + \log\log \frac{2n}{|V^{w_a}_{\max_v+1}|} - \log\log \frac{2n}{|V^v_{\max_v+1}| - \mathrm{Con}_v} + \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}}.$$

Thus, we get that

$$\sum_{a > \max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v](\varphi(w_a) - \varphi(v)) =$$

$$\sum_{a > \max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v] \quad \log\log \frac{2n}{|V^{w_a}_{a+1}| - \mathrm{Con}_{w_a}} + \tag{16}$$

$$\sum_{a > \max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v] \quad \log \left(\frac{\log \frac{2n}{|V^{w_a}_{\max_v+1}|}}{\log \frac{2n}{|V^v_{\max_v+1}| - \mathrm{Con}_v}} \right) + \tag{17}$$

$$\sum_{a > \max_v}^{n/3} \Pr[F(x_v) = a \mid \Pi \in S_v] \quad \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}}. \tag{18}$$

Using the previous ideas, it is easy to see that we can bound (18) as

$$
\begin{aligned}
(18) \leq \sum_{a>\max_v}^{n/3} \Pr[F(x_v)=a \mid \Pi \in S_v \wedge F(x_v) > \max_v] \cdot \Pr[F(x_v) > \max_v \mid \Pi \in S_v] \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}} &\leq \\
\sum_{a>\max_v}^{n/3} \Pr[F(x_v)=a \mid \Pi \in S_v \wedge F(x_v) > \max_v] \cdot \prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon'_i) \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}} &\leq \\
\prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon'_i) \sum_{j \in A_v} \log \frac{\log \frac{2n}{|V_j^{w_a}|}}{\log \frac{2n}{|V_j^v|}} &\leq \\
\prod_{i=1}^{|A_v|}(1-\varepsilon_i)(1-\varepsilon'_i) \sum_{j \in A_v} \log \left(1 + \log \frac{1}{1-\varepsilon_j}\right) &\leq \\
\prod_{i=1}^{|A_v|}(1-\varepsilon_i) \sum_{j \in A_v} \log \frac{1}{1-\varepsilon_j} &\leq \\
\prod_{i=1}^{|A_v|}(1-\varepsilon_i) \log \left(\frac{1}{\prod_{j \in A_v}(1-\varepsilon_j)}\right) &= O(1).
\end{aligned}
$$

To analyze (17) by Lemma 25 we know that $\Pr[F(x_v) > \max_v | \Pi \in S_v] \leq \frac{|V^v_{\max_v+1}|-k}{|V^v_{\max_v+1}|-\mathrm{Con}_v}$ in which k is as defined in the lemma. Note that in this case $|V^{w_a}_{\max_v+1}| = |V^v_{\max_v+1}| - k$. This implies

$$
\begin{aligned}
(17) \leq \sum_{a>\max_v}^{n} \Pr[F(x_v)=a \mid \Pi \in S_v] \log \left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}} \right) &= \\
\left(\sum_{a>\max_v}^{n} \Pr[F(x_v)=a \mid \Pi \in S_v] \right) \log \left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}} \right) &= \\
\Pr[F(x_v) > \max_v \mid \Pi \in S_v] \log \left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}} \right) &\leq \\
\frac{|V^v_{\max_v+1}|-k}{|V^v_{\max_v+1}|-\mathrm{Con}_v} \log \left(\frac{\log \frac{2n}{|V^v_{\max_v+1}|-k}}{\log \frac{2n}{|V^v_{\max_v+1}|-\mathrm{Con}_v}} \right) &= O(1).
\end{aligned}
$$

It is left to analyze (16). Let $x_0 = |P_0^v|$ and $x_1 = |P_1^v|$. Let b_i be a Boolean such $b_i \equiv i$. Note that we have $|V^{w_{\max_v+a}}_{\max_v+a+1}| = x_{b_{\max_v+a}} = n - x_{b_{\max_v+a+1}}$. Using Corollary 27 we can write (16) as follows:

$$
(16) \leq \sum_{a=1}^{n/3-\max_v} \Pr[F(x_v)=\max_v+a \mid \Pi \in S_v] \log\log \frac{2n}{|V^{w_{\max_v+a}}_{\max_v+a+1}| - \mathrm{Con}_{w_{\max_v+a}}} \leq
$$

$$
\sum_{a=1}^{n/3-\max_v} 2^{-\Omega(a)} \left(1 - \frac{x_{b_{\max_v+a+1}} - \lfloor(\max_v+a+1)/2\rfloor}{n-\max_v-a}\right) \log\log \frac{2n}{n - x_{b_{\max_v+a+1}} - \lfloor(\max_v+a)/2\rfloor} = O(1).
$$

D Details on the Polynomial Time Feasibility Checks

We repeat Theorem 8.

Theorem 8. *It is decidable in polynomial time whether a guessing history $\mathcal{H} = (x^i, s^i)_{i=1}^t$ is feasible. Furthermore, we can efficiently compute the number of consistent pairs $(z, \pi) \in \{0,1\}^n \times S_n$.*

We first show how to check feasibility in polynomial time.

Proof of the first part of Theorem 8: polynomial time feasibility check. Let $\mathcal{H} = (x^i, s^i)_{i=1}^t$ be given. Construct the sets $V_1, \ldots, V_n$ as described in Theorem 1. Now construct a bipartite graph $G(V_1, \ldots, V_n)$ with node set $[n]$ on both sides. Connect j to all nodes in V_j on the other side. Permutations π with $\pi(j) \in V_j$ for all j are in one-to-one correspondence to perfect matchings in this graph. If there is no perfect matching, the history in infeasible. Otherwise, let π be any permutation with $\pi(j) \in V_j$ for all j. We next construct z. We use the obvious rules:

a. If $i = \pi(j)$ and $j \leq s^h$ for some $h \in [t]$ then set $z_i := x_i^h$.

b. If $i = \pi(j)$ and $j = s^h + 1$ for some $h \in [t]$ then set $z_i := 1 - x_i^h$.

c. If z_i is not defined by one of the rules above, set it to an arbitrary value.

We need to show that these rules do not lead to a contradiction. Assume otherwise. There are three ways, in which we could get into a contradiction. There is some $i \in [n]$ and some $x^h, x^\ell \in \{0,1\}^n$

- setting z_i to opposite values by rule (a)
- setting z_i to opposite values by rule (b)
- setting z_i to opposite values by rules (b) applied to x^h and rule (a) applied to x^ℓ.

In each case, we readily derive a contradiction. In the first case, we have $j \leq s^h$, $j \leq s^\ell$ and $x_i^h \neq x_i^\ell$. Thus $\pi(j) = i \notin V_j$ by rule (1). In the second case, we have $j = s^h + 1 = s^\ell + 1$ and $x_i^h \neq x_i^\ell$. Thus $i \notin V_j$ by (2). In the third case, we have $j = s^h + 1$, $j \leq s^\ell$, and $x_i^h = x_i^\ell$. Thus $i \notin V_j$ by (3).

Finally, the pair (z, π) defined in this way is clearly consistent with the history. □

Next we show how to efficiently compute the number of consistent pairs. We recall Hall's condition for the existence of a perfect matching in a bipartite graph. A perfect matching exists if and only if $|\cup_{j \in J} V_j| \geq |J|$ for every $J \subseteq [n]$.

Proof of the second part of Theorem 8: efficiently counting consistent pairs. According to Theorem 1, the guessing history $\mathcal{H}$ can be equivalently described by a state $(V_1, \ldots, V_{s^*+1}, x^*, s^*)$. How many pairs (z, π) are compatible with this state?

Once we have chosen π, there are exactly $2^{n-(s^*+1)}$ different choices for z if $s^* < n$ and exactly one choice if $s^* = n$. The permutations can be chosen in a greedy fashion. We fix $\pi(1), \ldots, \pi(n)$ in this order. When we choose $\pi(i)$, the number of choices left is $|V_i|$ minus the number of $\pi(j)$, $j < i$, lying in V_i. If V_j is disjoint from V_i, $\pi(j)$ never lies in V_i and if V_j is contained in V_i, $\pi(j)$ is always contained in V_i. Thus the number of permutations is equal to

$$\prod_{1 \leq i \leq n} \left(|V_i| - |\{j < i \mid V_j \subseteq V_i\}|\right) .$$

It is easy to see that the greedy strategy does not violate Hall's condition. □

The set V_i is the candidate set of $\pi(i)$. The proof of Theorem 8 explains which values in V_i are actually possible as value for $\pi(i)$. A value $\ell \in V_i$ is feasible if there is a perfect matching in the graph $G(V_1, \ldots, V_n)$ containing the edge (i, ℓ). The existence of such a matching can be decided in polynomial time; we only need to test for a perfect matching in the graph $G \setminus \{i, \ell\}$. Hall's condition says that there is no such perfect matching if there is a set $J \subseteq [n] \setminus \{i\}$ such that $| \cup_{j \in J} V_j \setminus \{\ell\}| < |J|$. Since G contains a perfect matching (assuming a consistent history), this implies $| \cup_{j \in J} V_j| = |J|$, i.e., J is tight for Hall's condition. We have shown: Let $\ell \in V_i$. Then ℓ is infeasible for $\pi(i)$ if and only if there is a tight set J with $i \notin J$ and $\ell \in \cup_{j \in J} V_j$. Since the V_i form a laminar family, minimal tight sets have a special form; the consist of an i and all j such that V_j is contained in V_i. In the counting formula for the number of permutations such i are characterized by $|V_i| - |\{j < i \mid V_j \subseteq V_i\}| = 1$. In this situation, the elements of V_i are infeasible for all $\pi(j)$ with $j > i$. We may subtract V_i from each V_j with $j > i$.

If Hall's condition is tight for some J, i.e., $| \cup_{j \in J} V_j| = |J|$, we can learn $\pi|J$ easily. We have $V_j = [n]$ for $j > s^* + 1$ and hence the largest index in J is at most $s^* + 1$. Perturb x^* by flipping each bit in $\cup_{j \in J} V_j$ exactly once. The objective values determine the permutation. We conclude that we might as well give away $\pi|J$ once Hall's condition becomes tight for J.