Optimal k-Secretary with Logarithmic Memory

Mingda Qiao
University of Massachusetts Amherst, MA, USA

Wei Zhang
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We study memory-bounded algorithms for the k-secretary problem. The algorithm of Kleinberg (SODA 2005) achieves an optimal competitive ratio of $1 - O(1/\sqrt{k})$, yet a straightforward implementation requires $\Omega(k)$ memory. Our main result is a k-secretary algorithm that matches the optimal competitive ratio using $O(\log k)$ words of memory. We prove this result by establishing a general reduction from k-secretary to (random-order) *quantile estimation*, the problem of finding the k-th largest element in a stream. We show that a quantile estimation algorithm with an $O(k^\alpha)$ expected error (in terms of the rank) gives a $(1 - O(1/k^{1-\alpha}))$-competitive k-secretary algorithm with $O(1)$ extra words. We then introduce a new quantile estimation algorithm that achieves an $O(\sqrt{k})$ expected error bound using $O(\log k)$ memory. Of independent interest, we give a different algorithm that uses $O(\sqrt{k})$ words and finds the k-th largest element exactly with high probability, generalizing a result of Munro and Paterson (1980).

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Streaming algorithms, online algorithms, secretary problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.150

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2502.09834`

Acknowledgements We thank the anonymous reviewers, whose suggestions have helped improved this paper.

1 Introduction

In the classical secretary problem, n numbers are presented to an online decision-maker ("player") in random order. Upon seeing each number, the player has to make an irrevocable decision on whether to accept it – if the player accepts, the game terminates immediately; otherwise, the player moves on to the next element in the sequence. The goal of the player is to maximize the chosen number. It is well-known that there is a strategy for the player that chooses the largest element with probability at least $1/e$ for any value of n, which is essentially optimal.

The k-secretary problem is a natural extension of the above – instead of choosing a single element, the player is now allowed to accept up to k elements. For this problem, Kleinberg [27] gave a $(1 - O(1/\sqrt{k}))$-*competitive* algorithm. Here, an algorithm is α-competitive if the expected sum of the accepted elements is at least an α-fraction of the maximum possible result – the sum of the k largest numbers. It was also shown that the competitive ratio of any algorithm is at best $1 - \Omega(1/\sqrt{k})$.

In this work, we revisit the k-secretary problem under an additional memory constraint on the player's algorithm. One potential use case of the k-secretary model is the routing setting [36], in which n data packets arrive at a router in a sequential order, while the goal is to filter out certain low-quality packets and only forward k out of the n packets for


downstream processing. In this setup, as the values of both n and k can be huge compared to the storage of the router, we need the algorithm to be both optimal (in terms of the competitive ratio) and memory-efficient (using a memory sublinear, or even logarithmic, in k).

Unfortunately, Kleinberg's algorithm, while being optimal, is not designed to be memory-efficient. A straightforward implementation of the algorithm requires $\Omega(k)$ words of memory. The bottleneck of the memory usage is for finding the k-th largest element in the sequence – a problem known as *quantile estimation* – for which the straightforward method requires $\Omega(k)$ space.

One might hope to improve the memory bound using quantile estimation algorithms that are more space-efficient. We first note that there exist strong $\mathrm{poly}(k)$ memory lower bounds if we restrict ourselves to *exact* algorithms that find the k-th largest element exactly (with high probability): a lower bound of Munro and Paterson [35] suggests that this requires $\Omega(\sqrt{k})$ space. Naturally, one might hope to use algorithms that find the k-th largest element approximately (e.g., returning an element with rank $k \pm o(k)$). Guha and McGregor [22] gave an algorithm that uses $O(1)$ words of memory[1] and finds an element of rank $k \pm O(\sqrt{k} \cdot \log^2 n)$. Combining this result with Kleinberg's algorithm, however, would lead to extra $\mathrm{polylog}(n)$ factors in the competitive ratio.

In this work, we aim to answer the following questions:

In general, does an accurate quantile estimator lead to a competitive k-secretary algorithm while maintaining its memory usage? Concretely, to match the optimal competitive ratio for k-secretary, how much memory is needed?

1.1 Problem Setup

Notations. Throughout the paper, $s = (s_1, s_2, \ldots, s_n)$ denotes a random-order sequence, and we shorthand $s_{i_1:i_2}$ for the subsequence $(s_{i_1}, s_{i_1+1}, \ldots, s_{i_2})$. When it is clear from the context, we sometimes abuse the notation and let $s_{i_1:i_2}$ denote the set formed by the elements in the subsequence. For instance, $s_{i_1:i_2} \cap (-\infty, a)$ is used as a shorthand for $\{x \in \{s_{i_1}, s_{i_1+1}, \ldots, s_{i_2}\} : x < a\}$.

We will frequently use the *rank* of an element x within a contiguous subsequence, when only the elements that are strictly smaller than a threshold x' are counted.

▶ **Definition 1** (Rank). *For sequence $(s_1, s_2, \ldots, s_n)$ and $1 \le i_1 \le i_2 \le n$, we define*

$$\mathrm{Rank}_{i_1:i_2}(x; x') \coloneqq \left|\{t \in \{s_{i_1}, \ldots, s_{i_2}\} | x' > t \ge x\}\right|.$$

We also write $\mathrm{Rank}_{i_1:i_2}(x)$ as a shorthand for $\mathrm{Rank}_{i_1:i_2}(x; +\infty)$.

The k-Secretary problem. We formally define the k-secretary problem as follows.

▶ **Problem 2** (k-Secretary). *Let $x_1 > x_2 > \cdots > x_n \ge 0$ be n numbers that are non-negative, distinct, and unknown to the algorithm. The algorithm reads a uniformly random permutation of the n elements one by one. Upon seeing each element, the algorithm decides whether to accept it. The algorithm may accept at most k elements, and aims to maximize their sum.*

The assumption that all the elements are distinct is for the simplicity of the exposition. It comes without loss of generality by a standard tie-breaking argument.[2]

[1] We assume a word size of $\Theta(\log n)$.

[2] For example, we may augment each element with a random identifier with $\Theta(\log n)$ bits. This ensures distinctness with high probability, and only requires one additional word for storing each element.

We say that a k-secretary algorithm is competitive, if it is guaranteed to secure a certain fraction of the maximum possible outcome, namely, the sum of the k largest elements.

▶ **Definition 3** (Competitive ratio). *A k-secretary algorithm is α-competitive if, on any k-secretary instance $x_1 > x_2 > \cdots > x_n$, the expected sum of the accepted elements is at least $\alpha \cdot \sum_{i=1}^{k} x_i$. Here, the expectation is over the randomness both in the permutation and in the algorithm.*

Quantile estimation. Next, we formally define quantile estimation in a random-order stream.

▶ **Problem 4** (Quantile estimation). *Let $x_1 > x_2 > \cdots > x_n$ be n distinct numbers that are unknown to the algorithm. The algorithm takes n and k as inputs, and reads a uniformly random permutation of the n numbers one by one. The goal is to output the (approximately) k-th largest element x_k. When the output is $x^* \in \{x_1, x_2, \ldots, x_n\}$, the algorithm incurs an error of $|\mathrm{Rank}_{1:n}(x^*) - k|$.*

Again, the assumption that the n elements are distinct is for simplifying the notations. The results for distinct elements can be extended to the general case via tie-breaking and a more careful definition of the error.

Memory model. We informally introduce the model of computation and the memory usage of an algorithm. All our algorithms can be implemented in the word RAM model, assuming that each word can store either an integer of magnitude $\mathrm{poly}(n)$ or an element in the stream (in either k-secretary or quantile estimation). In particular, when all stream elements are bounded by $\mathrm{poly}(n)$, a word size of $\Theta(\log n)$ is sufficient. Moreover, all our algorithms are comparison-based: they only access the stream elements via pairwise comparisons, and do not perform any arithmetic operations on the elements. Therefore, a word size of $\Theta(\log n)$ would suffice if: (1) every stream element is presented as a unique identifier (which takes $O(\log n)$ bits); (2) the algorithm has access to an oracle that compares two elements specified by their identifiers.

1.2 Our Results

Our main result shows that logarithmic memory is sufficient for being optimally-competitive in the k-secretary problem.

▶ **Theorem 5.** *There is a k-secretary algorithm that uses $O(\log k)$ words and achieves a competitive ratio of $1 - O(1/\sqrt{k})$.*

Reduction from k-secretary to quantile estimation. Towards proving Theorem 5, we show that a quantile estimation algorithm leads to a competitive k-secretary algorithm with almost the same memory usage. Formally, the reduction applies to all *comparison-based* quantile estimators. Roughly speaking, an algorithm is comparison-based if it only accesses the elements in the stream via pairwise comparisons. As a result, the output of a comparison-based algorithm is invariant (up to the renaming of elements) under any order-preserving (i.e., monotone) transformations. See Definition 9 for a more formal definition.

▶ **Proposition 6.** *Suppose that, for some $\alpha \in [1/2, 1]$, there is a comparison-based quantile estimation algorithm that uses m words of memory and has an error of $O(k^\alpha)$ in expectation. Then, there is a k-secretary algorithm that uses $m + O(1)$ words and achieves a competitive ratio of $1 - O(1/k^{1-\alpha})$.*

The proposition follows from a reduction that is essentially implicit in [27] – Kleinberg's algorithm can be viewed as a special case in which the quantile estimator is exactly correct (and thus, we can take $\alpha = 1/2$). Our proof of Proposition 6 shows that this reduction is actually robust to the inaccuracy in the quantile estimator, so the error bound in quantile estimation can be smoothly translated to the sub-optimality in k-secretary.

As we explain in Section 2.4, directly following the reduction of [27] would increase the memory usage by a factor of $\log k$. We avoid this multiplicative increase by carefully modifying the reduction, so that the k-secretary algorithm would run $\log k$ copies of the quantile estimator *sequentially*, rather than in parallel.

New results for quantile estimation. In light of Proposition 6, the only missing piece towards proving Theorem 5 is a memory-efficient quantile estimator with $O(\sqrt{k})$ error.

▶ **Theorem 7.** *There is a quantile estimation algorithm that, when finding the k-th largest element, uses $O(\log k)$ words and incurs an $O(\sqrt{k})$ error in expectation over the randomness in both the algorithm and the order of the stream.*

We note that both the memory usage and the error bound are independent of n. This is because we address the *expected* error directly, rather than conditioning on a good event involving the entire length-n stream, thus avoiding a potential polylog(n) dependence.

The key idea behind the algorithm for Theorem 7 is to reduce the problem of finding the k-th largest element to finding the k'-th largest in a subsequence, for some $k' \ll k$. By shrinking the value of k fast enough, we ensure that the error incurred in the subproblems can be controlled. In Sections 2.1 and 2.2, we sketch the algorithm and its analysis in more detail.

Our second algorithm finds the k-th largest element exactly with high probability, albeit with a looser memory bound of $O(\sqrt{k})$.

▶ **Theorem 8.** *For any $m \geq 1$, there is a quantile estimation algorithm that uses $O(m)$ words of memory and incurs zero error (i.e., returns the exactly k-th largest element) with probability at least*

$$1 - 12\lfloor \log_2 k \rfloor \cdot \exp\left(-\frac{m}{12}\right) - 2 \sum_{i=0}^{\lfloor \log_2 k \rfloor - 1} \exp\left(-\frac{m^2}{32(k/2^i)}\right).$$

In particular, setting $m = O(\sqrt{k \log(1/\delta)} + \log(1/\delta))$ is sufficient for succeeding with probability $\geq 1 - \delta$.

Theorem 8 follows from the technique of [35]: the algorithm maintains m consecutive elements among the stream that has been observed so far, in the hope that the k-th largest element is among these m elements at the end. [35] showed that this strategy finds the median of a length-n random-order stream (i.e., the $k = n/2$ case) using $O(\sqrt{n})$ memory, and proved a matching $\Omega(\sqrt{n})$ lower bound. Our result extends the result of [35] to general k. In Section 2.3, we introduce the technique of [35] in more detail, and sketch our strategy for handling biased quantiles (i.e., the $k \ll n$ case).

1.3 Related Work

The secretary problem and its variants. The classical secretary problem is often attributed to [15], though the exact origin of the problem is obscure [18, 17]; different versions of the problem were studied in the contemporary work of [30, 13, 21].

The natural extension of selecting up to k elements (i.e., the k-secretary problem) was studied by [21, 3, 27, 5] under various objectives (e.g., the probability of selecting the k largest elements exactly, or the largest element being among the chosen ones). We followed the formulation of [27] in terms of the competitive ratio. [27] showed that the optimal competitive ratio is $1 - \Theta(1/\sqrt{k})$ as $k \to \infty$. [5] subsequently gave $1/e$-competitive algorithms for all $k \geq 1$, improving the result of [27] in the small-k regime. A more recent work of [4] also focused on the non-asymptotic regime, and gave a deterministic algorithm with a competitive ratio $> 1/e$ for all $k \geq 2$.

A further extension of the k-secretary problem considers selecting multiple elements subject to a more general combinatorial constraint. [5] studied the knapsack secretary problem. [7, 6] introduced the *matroid secretary problem*, in which the player is asked to select elements that form an independent set of a given matroid. This problem has been extensively studied [12, 42, 25, 14, 29, 16, 24, 43, 1], and it remains a long-standing open problem whether an $O(1)$-competitive algorithm exists in general. Other variants of the secretary problem consider alternative models for how the player accesses the elements and makes decisions, including models of interview costs [8], shortlists [2], reservation costs [11], and a "pen testing" variant [39, 19].

Quantile estimation. For the quantile estimation problem in random-order streams, [35] gave an exact selection algorithm for the $k = \lfloor n/2 \rfloor$ case (i.e., finding the median) with $O(\sqrt{n})$ memory, and proved a matching $\Omega(\sqrt{n})$ memory lower bound. Our Theorem 8 generalizes their result to general values of k, and their lower bound implies that the $O(\sqrt{k})$ memory usage cannot be improved in general (specifically, in the $n = 2k$ case).

For approximate selection, [22] proposed an algorithm that uses $O(1)$ words of memory and finds the k-th largest element up to an error of $O(\sqrt{k}\log^2 n \cdot \log(1/\delta))$ with probability $1 - \delta$. In comparison, Theorem 7 bounds the expected error and avoids the extra polylog(n) factor, both of which are crucial for the optimality of the resulting k-secretary algorithm. A subsequent work of [34] solved median estimation with an $n^{1/3+o(1)}$ error using $O(1)$ words of memory. In comparison, Theorem 7 applies to all values of k and makes the error dependent only on k and not n, at the cost of a larger exponent of $1/2$ and a logarithmic memory usage. We note that, even if we could improve the error to $O(k^{1/3})$, the reduction from k-secretary still introduces an $\Omega(\sqrt{k})$ error, which would dominate the quantile estimation error.

While we focus on the random-order setup of quantile estimation, many prior work also addressed the more challenging setup in which elements arrive in an arbitrary order [35, 31, 26, 33, 23]. The multi-pass setting, in which the algorithm may scan the stream multiple times, was also considered [35, 22].

Learning and decision making under memory constraints. More broadly, our work is part of the endeavor to understand the role of memory in learning, prediction, and decision-making. Prior work along this line studied the memory bounds for parity learning in a streaming setting [46, 45, 28, 40, 20], for the experts problem in online learning [44, 38, 37], as well as the fundamental problem of linear regression [41, 32, 10, 9].

2 Proof Overview

In this section, we give high-level sketches of our proofs and highlight some of the technical challenges. We recommend the readers to read this section before delving into the formal proofs in the appendix, as the technicalities might obscure some of the simple intuitions behind our algorithms.

We introduce our algorithm for Theorem 7 in Section 2.1. The key idea is to consider a sub-problem obtained by filtering out the elements that are above a certain threshold, which reduces the scale of the problem rapidly enough. In Section 2.2, we discuss a few technical challenges in turning an idealized analysis of the algorithm into a formal proof.

In Section 2.3, we sketch the proof of Theorem 8, which is based on a different idea of [35]. The algorithm maintains a block of m consecutive elements in the stream that has been observed so far. Whenever a new element comes, the algorithm tries to "drift" the length-m block, in the hope that the k-th largest element is always within the block. [35] framed this as a random walk problem, for which we give a new solution for general k and in the $m = \Omega(\sqrt{k})$ regime.

We end the section by sketching the proof of Proposition 6 (in Section 2.4), which reduces the k-secretary problem to quantile estimation using a variant of the algorithm of [27].

2.1 An Approximate Algorithm via Iterative Conditioning

Now, we introduce our algorithm for Theorem 7. The algorithm is a recursive one: we repeatedly reduce the problem of finding the k-th largest element to finding the k'-th largest (for some $k' \ll k$) in a subsequence of the stream. Before presenting the actual algorithm, we start by explaining why a naïve attempt fails, after which the key idea of our algorithm becomes more transparent.

A naïve reduction. How should we solve quantile estimation for a specific value of k, if we already have an algorithm (denoted by $\mathcal{A}_{k/2}$) that solves the $k' = k/2$ case with an $O(\sqrt{k'}) = O(\sqrt{k})$ error? The answer is simple: we run $\mathcal{A}_{k/2}$ on the first $n/2$ elements of the sequence and return its answer. Assuming that the output of $\mathcal{A}_{k/2}$, denoted by x^*, is exactly the $(k/2)$-th largest element among $s_{1:(n/2)}$, a simple concentration argument would show that the rank of x^* among $s_{1:n}$ is roughly $k \pm O(\sqrt{k})$. Now, let $l \coloneqq \mathrm{Rank}_{1:(n/2)}(x^*)$ denote the actual rank of x^* among the first half of the sequence. Our assumption on $\mathcal{A}_{k/2}$ implies that $l \approx k/2 \pm O(\sqrt{k})$, which further gives

$$\mathrm{Rank}_{1:n}(x^*) \approx 2l \pm O(\sqrt{l}) \approx k \pm O(\sqrt{k}).$$

At first glance, the above seems to suggest an extremely simple solution to quantile estimation: we repeatedly apply the reduction above, and reduce the value of "k" to $k/2, k/4, \ldots, 1$. For the base case that "k" equals 1, we can find the largest element exactly using $O(1)$ memory. However, this approach is equivalent to outputting the largest one among the first $\approx n/k$ elements, and its error can be easily shown to be $\Omega(k)$. What goes wrong here? The issue is that we ignored the blow-up in the error during the reduction. Define random variables $X_1, X_2, X_4, \ldots, X_{k/2}, X_k$ such that $X_{k'}$ denotes the error in the rank when we solve the instance with $k = k'$. Our discussion from the last paragraph shows that, for some universal constant $C > 0$, (X_k) satisfies the dynamics

$$X_1 = 0, \quad X_k \approx 2X_{k/2} + C\sqrt{k}. \tag{1}$$

Expanding the above gives

$$X_k \approx C\sqrt{k} + 2C\sqrt{k/2} + 4C\sqrt{k/4} + \cdots + (k/2) \cdot C\sqrt{2} = \Theta(k),$$

since the error incurred at $k' = O(1)$ would be doubled roughly $\log_2 k$ times through the iteration $X_k \approx 2X_{k/2} \pm C\sqrt{k}$, resulting in the dominant term of $\Omega(k)$ in the error.

The actual reduction. We would get an $O(\sqrt{k})$ error bound if we could shrink the value of k faster. Imagine that, instead of reducing to an instance with $k' = k/2$, we can reduce to an instance with $k' = k/100$, while the error still propagates in the same way. Then, the recursion in Equation (1) would be replaced with $X_k \approx 2X_{k/100} + C\sqrt{k}$, which leads to

$$X_k \approx C\sqrt{k} + 2C\sqrt{k/100} + 2^2 C\sqrt{k/100^2} + \cdots + 2^{\log_{100} k} C\sqrt{1} = O(\sqrt{k}).$$

Therefore, the key is to reduce the scale of the problem (measured by parameter k) at a slightly faster pace. Our algorithm does this in a fairly simple way. First, we divide the length-n sequence s into two halves $s_{1:B}$ and $s_{(B+1):n}$ for some $B \approx n/2$. Then, we subsample a random p-fraction of the first half, where $p = \Theta(m/k)$, by drawing $B_1 \sim \mathsf{Binomial}(B, p)$ and taking the first B_1 elements. We find the top m elements among $s_{1:B_1}$, denoted by $M[1], M[2], \ldots, M[m]$, using $O(m)$ memory (in the straightforward way). After reading the remaining $B - B_1$ elements in $s_{1:B}$, we can compute the rank $\mathrm{Rank}_{1:B}(M[i])$ for every $i \in [m]$. Then, we find an index i^* such that

$$\mathrm{Rank}_{1:B}(M[i^*]) < k/2 < \mathrm{Rank}_{1:B}(M[i^* + 1]).$$

Let $a' \coloneqq M[i^*]$ and $k' \coloneqq k/2 - \mathrm{Rank}_{1:B}(a')$. We know that the $(k/2)$-th largest element among $s_{1:B}$ is simply the k'-th largest element among $s_{1:B} \cap (-\infty, a')$. Therefore, we recursively find the k'-th largest element among $s_{(B+1):n} \cap (-\infty, a')$, in the hope that it will be a good approximation for the k-th largest element overall.

To make this work, we need the parameter k' for the recursive call to be smaller than k be a sufficiently large factor. This is indeed the case, since the gaps in the ranks of $M[1], M[2], \ldots, M[m]$ among $s_{1:B}$ roughly follow the geometric distribution $\mathsf{Geo}(p)$, so each of them has an expectation of $\leq 1/p = O(k/m)$. By setting $m = \Omega(\log k)$, we can ensure that every gap is smaller than $C_0 \cdot k$ (for any small constant C_0), with probability $1 - 1/\operatorname{poly}(k)$.

The reduction sketched above actually reduces quantile estimation to a slightly different problem: instead of finding the k-th largest element among $s_{1:n}$, we find the k-th largest among $s_{1:n} \cap (-\infty, a')$. This does not pose a problem as the reduction works for this generalized version as well.

2.2 A Few Technical Details

It should be noted that, while the intuition behind the algorithm is simple, it turns out to be highly non-trivial to state and analyze the algorithm rigorously.

Handling edge-cases. Formally, the recursive algorithm takes parameters n, k, and a as inputs, and aims to find the k-th largest element among $s_{1:n} \cap (-\infty, a)$, where $s_{1:n}$ is the upcoming random-order sequence of length n. The quantile estimation instance is invalid if $k > n' \coloneqq |s_{1:n} \cap (-\infty, a)|$. While the outermost call to the algorithm (where $a = +\infty$) is always valid, the algorithm might eventually lead to a recursive call in which the parameter k becomes invalid. In this case, our algorithm always returns $-\infty$ as the answer. Then, in the previous level of recursion – which makes this invalid call – the algorithm translates this $-\infty$ to the smallest element in the stream.

The second edge case is that we fail to find a good choice of a' that significantly reduces the value of k. This can, in turn, happen for three different reasons: (1) $|s_{1:B} \cap (-\infty, a)| < k/2$, in which the first half of the sequence does not have a $(k/2)$-th largest element; (2) the $(k/2)$-th largest element exists, but none of the elements with ranks in $[k/2 - C_0 \cdot k, k/2]$ get

subsampled into $s_{1:B_1}$, so none of them is present in the array M; (3) Some element with rank in $[k/2 - C_0 \cdot k, k/2]$ gets subsampled into $s_{1:B_1}$, but more than m elements with ranks in $[1, k/2]$ get subsampled, so that the one with a rank closest to $k/2$ is not kept in M.

For the latter two cases, we show that for some appropriate choice of $m = \Omega(\log k)$, their total probability is at most $1/\operatorname{poly}(k)$. Thus, if we detect that either of them happens, we can choose to return the largest element, which leads to a rank error of $k-1$ and becomes negligible after multiplying with the $1/\operatorname{poly}(k)$ probability. The first case is trickier since it *can* happen with a constant probability.[3] In that case, we return the smallest element (with a rank error of $n' - k$), and it turns out that its contribution to the overall error is still under control.

Towards a rigorous analysis. In addition to the need of handling the edge cases outlined above, the analysis of our algorithm is necessarily complicated due to the complex dependence between different parts of the algorithm.

For simplicity, we assume that the algorithm proceeds in a "typical" way, i.e., none of the edge cases happen. Furthermore, we analyze the outermost level of recursion, where the threshold parameter is $a = +\infty$ (i.e., no element gets ignored). Then, roughly speaking, our algorithm has the following three steps:

Step 1. Compute a threshold $a' \in s_{1:B}$ and let $k' \coloneqq k/2 - \operatorname{Rank}_{1:B}(a')$.
Step 2. Run the algorithm recursively on $s_{(B+1):n}$ with parameters $n - B$, k' and a'.
Step 3. Return the output x^* of the recursive call as the answer.

Then, the straightforward approach to analyzing the above would be an inductive one: Let $l \coloneqq \operatorname{Rank}_{(B+1):n}(x^*; a')$ denote the actual rank of x^* among $s_{(B+1):n} \cap (-\infty, a')$. By the inductive hypothesis, $|l - k'| = O(\sqrt{k'})$ in expectation (denoted by $l \approx k' \pm O(\sqrt{k'})$ informally). If B is sampled from $\mathsf{Binomial}(n, 1/2)$, $\{s_1, s_2, \ldots, s_B\}$ would be uniformly distributed among all subsets of $s_{1:n}$. Then, a concentration argument suggests

$$\operatorname{Rank}_{1:B}(x^*) \approx (k/2 - k') + l \pm O(\sqrt{l}) \quad \text{and} \quad \operatorname{Rank}_{(B+1):n}(x^*) \approx (k/2 - k') + l \pm O(\sqrt{k}). \tag{2}$$

In total, we would obtain $\operatorname{Rank}_{1:n}(x^*) = \operatorname{Rank}_{1:B}(x^*) + \operatorname{Rank}_{(B+1):n}(x^*) \approx k \pm O(\sqrt{k})$.

Unfortunately, the analysis above is technically incorrect. We were analyzing the *conditional* concentration of $\operatorname{Rank}_{1:B}(x^*)$ and $\operatorname{Rank}_{(B+1):n}(x^*)$ given the values of a', k' and l. Since (a', k', l) is determined by the value of B as well as ordering of s, after the conditioning, $s_{1:B}$ is no longer uniformly distributed, which invalidates the concentration argument.

Towards a rigorous analysis, we need to carefully untangle the randomness in the three steps above. Crucially, we note that our algorithm is comparison-based, i.e., the behavior of the algorithm is unchanged if we replace s with a different sequence s' with the same ordering as s. This motivates the following order in which we "realize" the randomness:

- First, we sample $B \sim \mathsf{Binomial}(n, 1/2)$.
- Then, towards analyzing Step 1, we realize the *relative ordering* of the elements $s_1, s_2, \ldots, s_B$ (out of the $B!$ possibilities). This ordering is sufficient for determining $i \coloneqq \operatorname{Rank}_{1:B}(a')$, the rank of a' among $s_{1:B}$ as well as the value of k'.
- We also realize the value of $i_1 \coloneqq \operatorname{Rank}_{1:n}(a')$. Conditioning on (B, i), this rank i_1 is identically distributed as the i-th smallest element in a size-B subset of $[n]$ chosen uniformly at random.

[3] Consider the case that $k \approx n' \coloneqq |s_{1:n} \cap (-\infty, a)|$, in which case $|s_{1:B} \cap (-\infty, a)|$ roughly follows $\mathsf{Binomial}(n', 1/2)$, and can be below $k/2$ with probability $\approx 1/2$.

- After that, we realize the *relative ordering* of the last $(n-B)$ elements $s_{(B+1):n}$ (out of the $(n-B)!$ possibilities). This, in turn, determines the rank of x^* (the output of the recursive call) among $s_{(B+1):n} \cap (-\infty, a')$, denoted by $l \coloneqq \mathrm{Rank}_{(B+1):n}(x^*; a')$.
- At this point, even after conditioning on the values of (B, i, i_1, l), the subset $\{s_1, s_2, \ldots, s_B\}$ is still uniformly distributed among all size-B subsets $S \subseteq \{s_1, s_2, \ldots, s_n\}$, subject to the constraint that the i_1-th largest element among $s_{1:n}$ (namely, a') is the i-th largest element among $s_{1:B}$.

The uniformity that we retain in the last step above allows us to rigorously prove bounds that are qualitatively similar to Equation (2).

2.3 An Exact Algorithm via Maintaining Consecutive Elements

We sketch our proof of Theorem 8, which is based on a very different algorithm. The key idea of the algorithm is to maintain *consecutive* elements among the elements that have been observed so far, which was used by [35] to solve the median selection problem (i.e., the special case that $k = n/2$).

The algorithm as a random walk (with limited control). At any time t, after reading the first t elements $s_1, s_2, \ldots, s_t$ in the random-order stream, the algorithm maintains m consecutive elements out of them. Here, "consecutive" is with respect to the sorted order of the elements, rather than the order in which they arrive. Formally, suppose that the first t elements of s, when sorted in decreasing order, are given by $x_1 > x_2 > \cdots > x_t$. The algorithm stores the elements $x_l, x_{l+1}, \ldots, x_r$ as well as the values of l and r, where $r - l + 1 = m$.

Then, what happens when the next element $x' \coloneqq s_{t+1}$ arrives? By the random-order assumption, x' is equally likely to fall into the $t+1$ intervals below:

$$(-\infty, x_t), (x_t, x_{t-1}), \ldots, (x_2, x_1), (x_1, +\infty).$$

In particular, we have the three cases below:

Case 1. $x' > x_l$. This happens with probability $l/(t+1)$. In this case, we cannot add x' to the array without breaking the invariant that the elements are consecutive. Therefore, we have to keep the m stored elements unchanged, and the parameters (l, r) become $(l', r') = (l+1, r+1)$ in the next step.

Case 2. $x' < x_r$. Similarly, with probability $(t-r+1)/(t+1)$, the condition $x' < x_r$ holds. Again, we cannot update the m elements, and the parameters in the next step remain unchanged, i.e., $(l', r') = (l, r)$.

Case 3. $x' \in (x_r, x_l)$. The most interesting case is that x' falls into one of the $r - l = m - 1$ "gaps" among $x_l > x_{l+1} > \cdots > x_r$, which happens with probability $(m-1)/(t+1)$. In this case, we would obtain $m+1$ consecutive elements among $s_{1:(t+1)}$. Then, we need to kick out one of the two elements at the ends. This gives us some freedom in deciding whether $(l', r') = (l, r)$ (by kicking out the smallest element) or $(l', r') = (l+1, r+1)$ (by kicking out the largest element).

At the end of the game, we successfully find the k-th largest element if $l \le k \le r$.

Following this strategy, the quantile estimation problem becomes a control problem: We start at time $t = m$ with $(l, r) = (1, m)$. At each of the following steps $t = m+1, m+2, \ldots, n$, the value of (l, r) transitions according to the three cases above – either it transitions deterministically (in Cases 1 and 2), or the algorithm may specify whether l and r get incremented by 1 in Case 3. The goal is to design our strategy in Case 3 (which may vary for different time step t and the values of (l, r) at that step), so that the probability of $k \in [l, r]$ is maximized at time n.

The above was exactly the idea in the work of [35], who studied the special case of $k = n/2$. For that special case, their strategy for Case 3 is to choose (l', r') such that $\left|\frac{l'+r'}{2} - \frac{t+2}{2}\right|$ is minimized. Intuitively, this makes sure that the median of the stream that has been observed so far is as close to the center of the length-m array as possible. They analyzed the resulting random walk, and showed that a memory of $m = \Theta(\sqrt{n})$ is sufficient. For the general k case, analyzing the random walk associated with the analogue of the algorithm of [35] becomes more difficult.[4]

The actual algorithm. Our algorithm behind Theorem 8 can be viewed as a *staged* strategy for the control problem of [35], which admits a slightly simpler analysis.

The algorithm is divided into $\approx \log_2 k$ stages. Roughly speaking, the goal of each stage $i \in \{0, 1, \ldots, \log_2 k\}$ is to make sure that, after reading the first $(n/k) \cdot 2^i$ elements, we have $l \leq 2^i \leq r$. What happens when we go from stage i to stage $i+1$? Suppose that, at the end of stage i, the 2^i-th largest element so far (denoted by x^*) is exactly in the middle of the array, i.e., $2^i = \frac{l+r}{2}$. As we read the additional elements in stage $(i+1)$, we maintain the m consecutive elements, such that x^* is kept in the middle of the array.

By a concentration argument, after stage $i+1$, the rank of x^* is bounded between $2^{i+1} \pm O(\sqrt{2^i}) = 2^{i+1} \pm O(\sqrt{k})$. Then, as long as $m \gg \sqrt{k}$, the actual element with rank 2^{i+1}, denoted by $\tilde{x}$, must be among the m consecutive elements maintained by the algorithm. Then, our algorithm shortens the length-m array that contains consecutive elements, so that $\tilde{x}$ becomes the center of the array. Here, we deviate from the plan outlined earlier, as the value of $r - l + 1$ can drop below m from time to time. Fortunately, by another concentration argument, as we go from stage $i+1$ to stage $i+2$, we are going to "absorb" additional elements in to the array, so that the length of the array returns to m before we shorten the array again.

The proof sketch above can be formalized into a high-probability guarantee, which states that, as long as $m = \Omega(\sqrt{k})$, the algorithm proceeds in the hoped-for manner except with a tiny probability.

Another perspective. We remark that our algorithm can alternatively be viewed as a remedy of the "naïve reduction" from Section 2.1. In Equation (1), by reducing to the $k' = k/2$ case, the error doubles and increases by an $O(\sqrt{k})$ amount, resulting in a trivial error bound of $O(k)$. In the algorithm outlined above, however, by utilizing the strategy of [35], we can offset the error by $O(m)$ at each iteration. In particular, as long as $m \gg \sqrt{k}$, the errors accumulated in all stages can be canceled out. This leads to our exact selection guarantee.

2.4 From Quantile Estimation to Secretary Problem

We sketch how an accurate quantile estimator leads to a competitive k-secretary algorithm via a reduction similar to the algorithm of [27]. Kleinberg's algorithm is a recursive one: to solve the k-secretary problem on a length-n stream, we divide the stream into two halves, each of length $\approx n/2$. We run the algorithm recursively for the $(k/2)$-secretary instance formed by the first half. In parallel, we find the $(k/2)$-th largest element, denoted by x^*, among the first half. Then, we read the second half of the sequence, and accept every element that is larger than x^*, until at least $k/2$ elements among the second half have been accepted.

[4] As [35] remarked in their work, this random walk is "difficult to analyze exactly since the transition probabilities vary with [the value of (l, r)] and with time".

To gain some intuition about the $1 - O(1/\sqrt{k})$ competitive ratio, consider a k-secretary instance that consists of k copies of 1 and $n - k$ copies of 0. To be exactly optimal, the algorithm needs to accept every single element of value 1. When running Kleinberg's algorithm, however, it holds with probability $\Omega(1)$ that the second half of the stream only contains $k/2 - \Omega(\sqrt{k})$ copies of 1, in which case we lose an $\Omega(1/\sqrt{k})$ term in the competitive ratio.

Towards proving Proposition 6, we apply Kleinberg's algorithm with the quantile estimation subroutine replaced by the (hypothetical) memory-efficient algorithm (denoted by $\mathcal{A}$) with an expected error of $O(k^\alpha)$. Then, we revisit the instance considered before. For clarity, we break ties among the k copies of 1 by replacing them with $1 - \varepsilon, 1 - 2\varepsilon, \ldots, 1 - k\varepsilon$ for some tiny value $\varepsilon \ll 1/k$. Then, the first half of the sequence contains a random subset of $\{1 - i\varepsilon : i \in [k]\}$. When we call the quantile estimator $\mathcal{A}$ to find the $k/2$-th largest element among the first half, it might hold with probability $\Omega(1)$ that $\mathcal{A}$ returns an element $x^* = 1 - i^*\varepsilon$ for some $i^* = k - \Omega(k^\alpha)$. In this case, the algorithm might reject $\Omega(k^\alpha)$ non-zero elements among the second half by mistake, resulting in a competitive ratio of at best $1 - \Omega(1/k^{1-\alpha})$.

While the argument above suggests why Proposition 6 gives the "right" dependence of the competitive ratio on α, the actual proof is slightly more complicated, since we need to argue that the impact of the error in quantile estimator on the competitive ratio is smooth, so that an upper bound on the *expected error* also translates into a competitive ratio (after considering the randomness in quantile estimator).

The reduction outlined above would not give the desired memory bound of $m + O(1)$ in Proposition 6. Kleinberg's algorithm involves running $\approx \log_2 k$ copies of the quantile estimator: roughly speaking, the i-th copy ($i \le \log_2 k$) is for finding the $k/2^i$-th largest element among $s_{1:(n/2^i)}$. Since these copies run on overlapping prefixes of the stream, they must run in parallel, and thus increasing the memory usage by a factor of $\log k$. The actual memory bound $m + O(1)$ is obtained from a slight modification to Kleinberg's algorithm: instead of finding the $(k/2)$-th largest element among $s_{1:(n/2)}$ and using it as the threshold x^* for the second half, we choose x^* as the $(k/4)$-th largest element among $s_{(n/4+1):(n/2)}$. This change only perturbs the rank of x^* by $O(\sqrt{k})$ in expectation, and would be dominated by the k^α error in the quantile estimator $\mathcal{A}$. As a result, we would run $\log k$ copies of $\mathcal{A}$ on *disjoint* intervals in s: $s_{(n/4+1):(n/2)}, s_{(n/8+1):(n/4)}, \ldots$, so it suffices to allocate m words of memory for all the $\log k$ calls to $\mathcal{A}$, and we only use an $O(1)$ extra memory.

3 Quantile Estimation in Logarithmic Memory

In this section, we state the algorithm for proving Theorem 7 and give an overview of its analysis. The full proof, as well as the proofs of the other results, is deferred to the full version.

3.1 The Algorithm

Our main algorithm (Algorithm 1) calls a recursive procedure Find-k-th (Algorithm 2) with parameters n, m, k and $+\infty$ to find the k-th largest element (among those that are smaller than $+\infty$) in the next n elements. We sketch how Algorithm 2 works in the following, in the special case that $a = +\infty$ (i.e., the first level of the recursion):

- First, it samples $B \sim \mathsf{Binomial}(n, 1/2)$ and divides the n upcoming elements (denoted by $s_1, s_2, \ldots, s_n$) into two halves: $s_{1:B}$ and $s_{(B+1):n}$.

- Then, among the first half, it further sub-samples a small fraction of $B_1 \sim \mathsf{Binomial}\left(B, \frac{2m}{3k}\right)$ elements. The hope is to find an element a' such that its rank among $s_{1:B}$ is slightly below $k/2$. (Concretely, $\mathrm{Rank}_{1:B}(a'; a) = \lfloor k/2 \rfloor - k'$ for some k' between 1 and $\delta := C_0 \cdot k$.)
- At this point, we know that the $(k/2)$-th largest element among $s_{1:B}$ is the k'-th largest element among $s_{1:B} \cap (-\infty, a')$.
- Finally, we call Algorithm 2 recursively with parameters $n' = n - B$, k' and a' to find the k'-th largest element among $s_{(B+1):n} \cap (-\infty, a')$, in the hope that it will be approximately the overall k-th largest among $s_{1:n}$.

In the more general case that $a \neq +\infty$, the algorithm essentially does the same thing, except that all elements that are larger than or equal to a are ignored.

Algorithm 1 Main Algorithm.

Input: Stream length n, memory size m, target rank k.
Output: An approximately k-th largest element among the n elements.
Call Find-k-th$(n, m, k, +\infty)$ in Algorithm 2;

Algorithm 2 Find-k-th(n, m, k, a).

Input: Stream length n, memory size m, target rank k, element threshold a, and access to random-order sequence $s = (s_1, s_2, \ldots, s_n)$
Output: The (approximately) k-th largest element among $s \cap (-\infty, a)$, or $-\infty$ if $|s \cap (-\infty, a)| < k$
Use $O(1)$ memory to keep track of: (1) the smallest element, denoted by $\underline{x} = \min\{s_1, s_2, \ldots, s_n\}$; (2) the number of elements that are strictly smaller than a, denoted by $n' = |s \cap (-\infty, a)|$;
if $k \leq m$ **then**
    Use $O(m)$ memory to find the k largest elements among $s_{1:n} \cap (-\infty, a)$;
    return the k-th largest element, or $-\infty$ if $n' < k$;
Draw $B \sim \mathsf{Binomial}\left(n, \frac{1}{2}\right)$ and $B_1 \sim \mathsf{Binomial}\left(B, \frac{2m}{3k}\right)$;
Use a length-m array M to find the m largest elements among $s_{1:B_1} \cap (-\infty, a)$;
Read until the B-th element in the string to compute the value of $\mathrm{Rank}_{1:B}(x; a)$ for each element x in array M;
if $|s_{1:B} \cap (-\infty, a)| < \lfloor k/2 \rfloor$ **then**
    Read the remaining $n - B$ elements to compute $\underline{x}$ and n';
    return $\underline{x}$, or $-\infty$ if $n' < k$;
$\delta \leftarrow C_0 \cdot k$, where C_0 is a sufficiently small constant parameter;
Find the largest element a' in M such that $\mathrm{Rank}_{1:B}(a'; a) \in [\lfloor k/2 \rfloor - \delta, \lfloor k/2 \rfloor - 1]$;
if *no such element a' exists* **then**
    Read the remaining $n - B$ elements;
    return the largest element in the array below a, or $-\infty$ if $n' < k$;
$x \leftarrow$ Find-k-th$\left(n - B, m, \lfloor k/2 \rfloor - \mathrm{Rank}_{1:B}(a'; a), a'\right)$;
if $n' < k$ **then return** $-\infty$;
if $x = -\infty$ **then return** $\underline{x}$;
return x;

Comparison-based algorithms. We note that our algorithm is *comparison-based* in the sense that it can be implemented such that the algorithm only accesses the elements in the sequence in the following three ways: (1) compare a pair of elements; (2) return an element as output; (3) pass an element as a parameter of a recursive call. One desirable property of such comparison-based algorithms is that, roughly speaking, the output of the algorithm only depends on the *relative ordering* of the elements, rather than their exact identities. We formalize this property and use it as the definition of comparison-based algorithms in the following.

▶ **Definition 9** (Comparison-based algorithms). *A quantile estimation algorithm $\mathcal{A}$ is comparison-based if, for any n and k, when finding the k-th largest element in a random-order sequence s of n distinct elements, the distribution of*

$$\mathrm{Rank}_{1:n}(\mathcal{A}(n,k,s))$$

is the same regardless of the choice of the n elements in s.

More generally, suppose that the quantile estimation problem has an additional threshold parameter a. An algorithm $\mathcal{A}$ is comparison-based if, as long as $a \notin \{s_1, s_2, \ldots, s_n\}$, the distribution of

$$\mathrm{Rank}_{1:n}(\mathcal{A}(n,k,a,s);a)$$

only depends on n, k, and $\mathrm{Rank}_{1:n}(a)$, and does not depend on a and the n elements in s.

3.2 Overview of the Analysis

The rest of this section is devoted to the analysis of Algorithm 2. We start by observing the behavior of Find-k-th in several different edge cases. We then sketch how we analyze the randomness in both the random-order stream and the algorithm, so that the randomness in different parts can be significantly decoupled.

Corner cases. Let $n' \coloneqq |\{s_1, s_2, \ldots, s_n\} \cap (-\infty, a)|$ denote the number of elements that are strictly smaller than the threshold a. When $n' < k$, the quantile estimation problem is ill-defined, in which case Find-k-th always returns $-\infty$. Note that the $n' < k$ case has the highest priority among all edge cases in the sense that, before the algorithm tries to output anything (possibly as a result of handling other corner cases), it makes sure to read through the end of the stream s to check whether $n' < k$ (and outputs $-\infty$ if so).

On Line 3, we choose to use the straightforward algorithm for quantile estimation when $k \le m$. This serves as the boundary condition of the recursion, and also ensures that the parameter $\frac{2m}{3k}$ on Line 5 is indeed in $[0,1]$, and thus valid.

On Line 10, if it turns out that $|s_{1:B} \cap (-\infty, a)| < \lfloor k/2 \rfloor$, we return the smallest element in the entire stream. To see why we do this, recall that there are n' elements in $s_{1:n} \cap (-\infty, a)$. If we are not in the edge case that $n' < k$, we should expect that there are $\approx n'/2 \ge k/2$ elements in $s_{1:B} \cap (-\infty, a)$, since $s_{1:B}$, as a set, is uniformly distributed among all subsets of $\{s_1, s_2, \ldots, s_n\}$. Therefore, whenever the edge case $|s_{1:B} \cap (-\infty, a)| < \lfloor k/2 \rfloor$ happens, we are sure that n' only exceeds k by a small amount, in which case outputting the n'-th largest element in $s \cap (-\infty, a)$ (namely, the smallest element in s) is accurate enough.

When the above does not happen, we aim to find an element a' in the first half of the stream, $s_{1:B}$, so that a' is close to the $\lfloor k/2 \rfloor$-th largest among $s_{1:B} \cap (-\infty, a)$. On Line 15, if we fail to find such an a', we return the largest element among $s_{1:n} \cap (-\infty, a)$, which has a rank of 1. While doing so leads to an error of $k-1$ in the rank, this edge case only happens with probability $1/\operatorname{poly}(k)$, so the contribution to the expected error is negligible.

If none of the edge cases above happen, Find-k-th calls itself recursively and gets an output x. The final edge case is that this x might take value $-\infty$, as a result of encountering the first edge case in the recursive call. If this happens, we translate $-\infty$ to the smallest element $\underline{x}$ in the stream.

Unravel the randomness. We will prove the error bound of Find-k-th by induction. As mentioned in Section 2.2, a rigorous analysis is complicated because, for the inductive step, we need to show that the accuracy of the recursive call

$$x^* \leftarrow \text{Find-}k\text{-th}(n-B, m, \lfloor k/2 \rfloor - \text{Rank}_{1:B}(a'; a), a')$$

(conditioning on all the parameters) implies the accuracy of the original procedure. For this purpose, we might hope that $s_{(B+1):n}$ (as a set) is uniformly distributed among all size-$(n-B)$ subsets of $s_{1:n}$, so that we can translate the rank of x^* among $s_{(B+1):n}$ to its rank among the entire stream $s_{1:n}$. Unfortunately, this uniformity might not hold, since the conditioning on a' and $\text{Rank}_{1:B}(a'; a)$ would bias the conditional distribution of $s_{(B+1):n}$.

To make the analysis valid, we need to "realize" the randomness of $s_{1:n}$ in a specific order. In the following, we sketch the analysis assuming that none of the edge cases above happen:

- First, over the randomness in $B \sim \text{Binomial}(n, 1/2)$, $s_{1:B}$ is uniformly distributed among all subsets of $s_{1:n}$. In particular, $s_{1:B} \cap (-\infty, a)$ is a uniformly random subset of $s_{1:n} \cap (-\infty, a)$. It follows that: (1) $B' \coloneqq |s_{1:B} \cap (-\infty, a)|$ follows the distribution $\text{Binomial}(n', 1/2)$, where $n' \coloneqq |s_{1:n} \cap (-\infty, a)|$; (2) Conditioning on the realization of B', $s_{1:B} \cap (-\infty, a)$ is uniformly distributed among all size-B' subsets of the size-n' set $s_{1:n} \cap (-\infty, a)$.
- We condition on the realization of B'. The algorithm finds an element $a' \in s_{1:B}$ on Line 12. Let $i \coloneqq \text{Rank}_{1:B}(a'; a)$ denote its rank among the first half (when only elements below a are considered). Note that the distribution of i is solely determined by the value of B', since the algorithm is comparison-based (Definition 9).
- Conditioning on the realization of (B', i), a' is identically distributed as the i-th largest element in a uniformly random size-B' subset of $s_{1:n} \cap (-\infty, a)$. Let $i_1 \coloneqq \text{Rank}_{1:n}(a'; a)$ denote the rank of a'. Later, we can analyze the concentration of $i_1 \mid B', i$.
- Conditioning on the realization of (B', i, i_1), the algorithm makes a recursive call (on Line 16) to find the k'-th largest among $s_{(B+1):n} \cap (-\infty, a')$, where $k' \coloneqq \lfloor k/2 \rfloor - i$. Let x^* denote the element returned by the recursive call, and $l \coloneqq \text{Rank}_{B+1:n}(x^*; a')$ be its actual rank. During the inductive proof, we will use the induction hypothesis that $l \mid B', i, i_1$ concentrates around k'.
- Conditioning on (B', i, i_1, l), x^* is the l-th largest element among $s_{(B+1):n} \cap (-\infty, a')$. Furthermore, we can verify that $s_{(B+1):n} \cap (-\infty, a')$ is uniformly distributed among all size-$(n - i_1 - (B - i))$ subsets of $s_{1:n} \cap (-\infty, a')$. Applying another concentration bound shows that $\text{Rank}_{1:n}(x^*; a')$ concentrates, which in turn implies the concentration of $\text{Rank}_{1:n}(x^*; a) = \text{Rank}_{1:n}(x^*; a') + i_1$.
- Finally, our goal is to show that the rank of x^*, $\text{Rank}_{1:n}(x^*; a)$, concentrates around k. For this purpose, we will start with the conditional concentration bound above, and take an expectation over the joint distribution of (B', i, i_1, l).

References

1 Dorna Abdolazimi, Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. Matroid partition property and the secretary problem. In *Innovations in Theoretical Computer Science (ITCS)*, pages 2:1–2:9, 2023. doi:10.4230/LIPIcs.ITCS.2023.2.

2 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In *Innovations in Theoretical Computer Science (ITCS)*, pages 1:1–1:19, 2019. doi:10.4230/LIPIcs.ITCS.2019.1.

3 Miklos Ajtai, Nimrod Megiddo, and Orli Waarts. Improved algorithms and analysis for secretary problems and generalizations. *SIAM Journal on Discrete Mathematics*, 14(1):1–27, 2001. doi:10.1137/S0895480195290017.

4 Susanne Albers and Leon Ladewig. New results for the k-secretary problem. *Theoretical Computer Science*, 863:102–119, 2021. doi:10.1016/J.TCS.2021.02.022.

5 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 16–28, 2007. doi:10.1007/978-3-540-74208-1_2.

6 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Matroid secretary problems. *Journal of the ACM (JACM)*, 65(6):1–26, 2018. doi:10.1145/3212512.

7 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Symposium on Discrete Algorithms (SODA)*, pages 434–443, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283429.

8 R Bartoszyński and Z Govindarajulu. The secretary problem with interview cost. *Sankhyā: The Indian Journal of Statistics, Series B*, pages 11–28, 1978.

9 Moise Blanchard. Gradient descent is pareto-optimal in the oracle complexity and memory tradeoff for feasibility problems. In *Foundations of Computer Science (FOCS)*, pages 2413–2435, 2024.

10 Moise Blanchard, Junhui Zhang, and Patrick Jaillet. Memory-constrained algorithms for convex optimization via recursive cutting-planes. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6156–6189, 2023.

11 Elisabet Burjons, Matthias Gehnen, Henri Lotze, Daniel Mock, and Peter Rossmanith. The secretary problem with reservation costs. In *International Computing and Combinatorics Conference*, pages 553–564, 2021. doi:10.1007/978-3-030-89543-3_46.

12 Sourav Chakraborty and Oded Lachish. Improved competitive ratio for the matroid secretary problem. In *Symposium on Discrete Algorithms (SODA)*, pages 1702–1712, 2012. doi:10.1137/1.9781611973099.135.

13 Yuan-Shih Chow, Sigaiti Moriguti, Herbert Robbins, and Stephen Mitchell Samuels. Optimal selection based on relative rank (the "secretary problem"). *Israel Journal of mathematics*, 2(2):81–90, 1964.

14 Michael Dinitz and Guy Kortsarz. Matroid secretary for regular and decomposable matroids. *SIAM Journal on Computing*, 43(5):1807–1830, 2014. doi:10.1137/13094030X.

15 Evgenii Borisovich Dynkin. The optimum choice of the instant for stopping a markov process. *Soviet Mathematics*, 4:627–629, 1963.

16 Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple o (log log (rank))-competitive algorithm for the matroid secretary problem. In *Symposium on Discrete Algorithms (SODA)*, pages 1189–1201, 2014.

17 Thomas S. Ferguson. Who solved the secretary problem? *Statistical science*, 4(3):282–289, 1989.

18 P.R. Freeman. The secretary problem and its extensions: A review. *International Statistical Review*, pages 189–206, 1983.

19 Aadityan Ganesh and Jason Hartline. Combinatorial pen testing (or consumer surplus of deferred-acceptance auctions). *arXiv preprint arXiv:2301.12462*, 2023. doi:10.48550/arXiv.2301.12462.

20 Sumegha Garg, Ran Raz, and Avishay Tal. Extractor-based time-space lower bounds for learning. In *Symposium on Theory of Computing (STOC)*, pages 990–1002, 2018. doi:10.1145/3188745.3188962.

21 John P. Gilbert and Frederick Mosteller. Recognizing the maximum of a sequence. *Journal of the American Statistical Association*, 61(313):35–73, 1966.

22 Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009. doi:10.1137/07069328X.

23 Meghal Gupta, Mihir Singhal, and Hongxun Wu. Optimal quantile estimation: beyond the comparison model. In *Foundations of Computer Science (FOCS)*, pages 1137–1158, 2024. doi:10.1109/FOCS61266.2024.00075.

24 Tony Huynh and Peter Nelson. The matroid secretary problem for minor-closed classes and random matroids. *SIAM Journal on Discrete Mathematics*, 34(1):163–176, 2020. doi:10.1137/16M1107899.

25 Patrick Jaillet, José A Soto, and Rico Zenklusen. Advances on matroid secretary problems: Free order model and laminar case. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 254–265, 2013. doi:10.1007/978-3-642-36694-9_22.

26 Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *Foundations of Computer Science (FOCS)*, pages 71–78, 2016. doi:10.1109/FOCS.2016.17.

27 Robert D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Symposium on Discrete Algorithms (SODA)*, pages 630–631, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070519.

28 Gillat Kol, Ran Raz, and Avishay Tal. Time-space hardness of learning sparse parities. In *Symposium on Theory of Computing (STOC)*, pages 1067–1080, 2017. doi:10.1145/3055399.3055430.

29 Oded Lachish. O (log log rank) competitive ratio for the matroid secretary problem. In *Foundations of Computer Science (FOCS)*, pages 326–335, 2014. doi:10.1109/FOCS.2014.42.

30 D.V. Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 10(1):39–51, 1961.

31 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record*, 27(2):426–435, 1998. doi:10.1145/276304.276342.

32 Annie Marsden, Vatsal Sharan, Aaron Sidford, and Gregory Valiant. Efficient convex optimization requires superlinear memory. In *Conference on Learning Theory (COLT)*, pages 2390–2430, 2022. URL: https://proceedings.mlr.press/v178/marsden22a.html.

33 Charles Masson, Jee E. Rim, and Homin K. Lee. Ddsketch: a fast and fully-mergeable quantile sketch with relative-error guarantees. *Proceedings of the VLDB Endowment*, 12(12):2195–2205, 2019. doi:10.14778/3352063.3352135.

34 Andrew McGregor and Paul Valiant. The shifting sands algorithm. In *Symposium on Discrete Algorithms (SODA)*, pages 453–458, 2012. doi:10.1137/1.9781611973099.39.

35 J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

36 Ryo Nakano, Kazuya Tsukamoto, Masato Tsuru, and Yuji Oie. A message forward scheduling based on a secretary problem for mobile relay nodes. *IEICE Technical Report*, 110(448):357–362, 2011.

37 Binghui Peng and Aviad Rubinstein. Near optimal memory-regret tradeoff for online learning. In *Foundations of Computer Science (FOCS)*, pages 1171–1194, 2023. doi:10.1109/FOCS57990.2023.00069.

38 Binghui Peng and Fred Zhang. Online prediction in sub-linear space. In *Symposium on Discrete Algorithms (SODA)*, pages 1611–1634, 2023. doi:10.1137/1.9781611977554.CH60.

39 Mingda Qiao and Gregory Valiant. Online pen testing. In *Innovations in Theoretical Computer Science (ITCS)*, pages 91:1–91:26, 2023. doi:10.4230/LIPIcs.ITCS.2023.91.

40 Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. *Journal of the ACM (JACM)*, 66(1):1–18, 2018. doi:10.1145/3186563.

41 Vatsal Sharan, Aaron Sidford, and Gregory Valiant. Memory-sample tradeoffs for linear regression with small error. In *Symposium on Theory of Computing (STOC)*, pages 890–901, 2019. doi:10.1145/3313276.3316403.

42 José A Soto. Matroid secretary problem in the random-assignment model. *SIAM Journal on Computing*, 42(1):178–211, 2013. doi:10.1137/110852061.

43 José A Soto, Abner Turkieltaub, and Victor Verdugo. Strong algorithms for the ordinal matroid secretary problem. *Mathematics of Operations Research*, 46(2):642–673, 2021. doi:10.1287/MOOR.2020.1083.

44 Vaidehi Srinivas, David P. Woodruff, Ziyu Xu, and Samson Zhou. Memory bounds for the experts problem. In *Symposium on Theory of Computing (STOC)*, pages 1158–1171, 2022. doi:10.1145/3519935.3520069.

45 Jacob Steinhardt, Gregory Valiant, and Stefan Wager. Memory, communication, and statistical queries. In *Conference on Learning Theory (COLT)*, pages 1490–1516, 2016. URL: http://proceedings.mlr.press/v49/steinhardt16.html.

46 Gregory Valiant and Paul Valiant. Information theoretically secure databases. *arXiv preprint arXiv:1605.02646*, 2016. arXiv:1605.02646.