Generic Techniques for Building Top-$k$ Structures*

Saladi Rahul  
Indian Institute of Science  
saladi@iisc.ac.in

Yufei Tao  
Chinese University of Hong Kong  
taoyf@cse.cuhk.edu.hk

June 25, 2022

Abstract

A reporting query returns the objects satisfying a predicate $q$ from an input set. In prioritized reporting, each object carries a real-valued weight (which can be query dependent), and a query returns the objects that satisfy $q$ and have weights at least a threshold $\tau$. A top-$k$ query finds, among all the objects satisfying $q$, the $k$ ones of the largest weights; a max query is a special instance with $k = 1$. We want to design data structures of small space to support queries (and possibly updates) efficiently.

Previous work has shown that a top-$k$ structure can also support max and prioritized queries with no performance deterioration. This paper explores the opposite direction: do prioritized queries, possibly combined with max queries, imply top-$k$ search? Subject to mild conditions, we provide affirmative answers with two reduction techniques. The first converts a prioritized structure into a static top-$k$ structure with the same space complexity and only a logarithmic blowup in query time. If a max structure is available in addition, our second reduction yields a top-$k$ structure with no degradation in expected performance (this holds for the space, query, and update complexities). Our techniques significantly simplify the design of top-$k$ structures because structures for max and prioritized queries are often easier to obtain. We demonstrate this by developing top-$k$ structures for interval stabbing, 3D dominance, halfspace reporting, linear ranking, and $L_\infty$ nearest neighbor search in the RAM and the external memory computation models.

To appear in ACM Transactions on Algorithms.

*A preliminary version of this paper appeared in PODS’16.
1 Introduction

Reporting queries can be abstracted at a high level as follows. Let $D$ be the data domain, namely, the (possibly infinite) set from which the input elements (called objects) are chosen. Denote by $Q$ the set of all possible predicates, each being a function $q : D \rightarrow \{0, 1\}$. An object $e \in D$ satisfies $q \in Q$ if $q(e) = 1$. The input is a finite $D \subseteq \mathbb{D}$. Given a predicate $q \in Q$, a reporting query returns all the objects $e \in D$ satisfying $q$; we use $q(D)$ to represent the set of such objects.

The result size $|q(D)|$ is often exceedingly large, especially in today’s big-data era. In practice, a user would rarely be interested in all the objects in $q(D)$. For many applications, what matter most are the $k$ records (for some $k \ll |D|$) in $q(D)$ with the highest “importance” measured by an appropriate weight function. For example, while the query “find all the creditcard transactions of today” can return millions of records, a bank manager may actually just want to scrutinize the 100 transactions with the largest payments. This motivates top-$k$ search, which retrieves only the $k$ best objects and becomes increasingly important as the data volumes continue to grow rapidly.

Top-$k$ search is easy if one is willing to examine all the objects in $D$. Avoiding an exhaustive scan calls for a data structure on $D$, which we will refer to as a top-$k$ structure. This paper aims to develop such structures for a broad class of top-$k$ problems formulated next.

1.1 Problem Definitions

Top-$k$ search. Let $W$ be a set of functions $w : D \rightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of real values. An object $e$ in the input set $D$ has $w$-weight $w(e)$ under a $w \in W$. Given a predicate $q \in Q$ and a $w \in W$, a top-$k$ query returns the $k$ objects with the largest $w$-weights in $q(D)$. Note that various queries may choose different weight functions. The objective is to preprocess the input $D$ into a data structure that answers every top-$k$ query efficiently.

We say that a top-$k$ problem characterized by a triplet $(D, Q, W)$ is $\lambda$-polynomially bounded if the next two conditions hold for any input $D \subseteq \mathbb{D}$:

- there are at most $|D|^{\lambda}$ distinct $q(D)$, ranging over all the predicates $q \in Q$;
- there are at most $|D|^{\lambda}$ ways to order the objects $e \in D$ by $w(e)$, ranging over all the weight functions $w \in W$.

When $\lambda$ is a constant and need not be emphasized, we will simply say that the problem is polynomially bounded. Many top-$k$ problems in computational geometry are polynomially bounded when the dimensionality is a fixed constant (we will discuss several representatives in Section 1.4).

Computation models. We will carry out most of our analysis in the standard external memory (EM) model [12]. A machine is equipped with $M$ words of memory and a disk formatted into blocks of $B$ words each. The values of $M$ and $B$ satisfy $M \geq 2B$. An I/O either reads a disk block into memory or writes $B$ words of memory into a disk block. The time of an algorithm is measured in the number of I/Os, while the space of a structure in the number of disk blocks occupied. By setting $M$ and $B$ to appropriate constants, all our EM results hold in the classic RAM model as well.

We assume that each object in $\mathbb{D}$, each predicate in $Q$, and each function in $W$ can all be described in $O(1)$ words. We further assume that the objects of $D$ have distinct $w$-weights under every $w \in W$ (by breaking ties in a consistent manner, e.g., favoring a smaller id).

Math conventions. Set $n = |D|$. All complexities hold in the worst case by default. The notation
\(\tilde{O}(\cdot)\) hides a factor polylogarithmic to \(n\). All logarithms have base 2 by default. A function \(f(n)\) is \textit{geometrically converging} if it satisfies two conditions:

- \(f(n) = O(1)\) if \(n < B\);
- otherwise, it holds for any \(c \geq 2\) that \(\sum_{i=0}^{h} f(n/c^i) = O(f(n))\), where \(h\) is the largest integer \(i\) satisfying \(n/c^i \geq B\).

1.2 Prioritized Reporting

Top-\(k\) search is closely related to \textit{prioritized reporting}. Let \(\mathbb{D}\), \(\mathbb{Q}\), and \(\mathbb{W}\) be as defined earlier; an input is still a subset \(D\) of \(\mathbb{D}\). Given a predicate \(q \in \mathbb{Q}\), a weight function \(w \in \mathbb{W}\), and a real value \(\tau\), a \textit{prioritized query} reports all the objects \(e \in q(D)\) with \(w(e) \geq \tau\). We want to preprocess \(D\) into a data structure that can answer all prioritized queries efficiently. Such structures will be referred to as \textit{prioritized structures}.

Considerable research has been dedicated to designing top-\(k\) data structures (see Section 2 for a survey). Most solutions, interestingly, follow a common two-step process: first design a prioritized structure and then modify it somehow to answer top-\(k\) reporting query. The process turns out to be necessary. Previous work [38 – 40] has shown that prioritized reporting can be reduced to top-\(k\) reporting. Specifically, suppose that there is a structure of \(S_{top}(n)\) space that answers a top-\(k\) query in \(Q_{top}(n) + O(k/B)\) I/Os. Then, there is a prioritized structure (under the same \(\mathbb{D}\), \(\mathbb{Q}\), and \(\mathbb{W}\)) of \(S_{pri}(n)\) space that answers a query in \(Q_{pri}(n) + O(t/B)\) I/Os (where \(t\) is the number of reported objects) such that \(S_{pri}(n) = O(S_{top}(n))\) and \(Q_{pri}(n) = O(Q_{top}(n))\). This holds regardless of \(\mathbb{D}\), \(\mathbb{Q}\), and \(\mathbb{W}\). Therefore, if one does not even have a good prioritized structure, s/he should not hope to derive a good top-\(k\) structure.

What is intriguing is the opposite: \textit{how much harder is top-\(k\) reporting than prioritized reporting}? To rephrase, assume the existence of a prioritized structure consuming \(S_{pri}(n)\) space that answers a query in \(Q_{pri} + O(t/B)\) time. We want to use the structure as a \textit{black box} to design a top-\(k\) structure (under the same \(\mathbb{D}\), \(\mathbb{Q}\), and \(\mathbb{W}\)) of space \(S_{top}(n)\) and query cost \(Q_{top} + O(k/B)\). How good can the functions \(S_{top}(n)\) and \(Q_{top}(n)\) be compared to \(S_{pri}(n)\) and \(Q_{pri}(n)\)? Resolving the question has an important technical implication. Since prioritized reporting structures are already known for numerous problems, such a black-box reduction will give us top-\(k\) structures immediately for all those problems!

Unfortunately, there is not much progress towards this direction. The best understanding [39] is rather primitive:\footnote{Assuming that \(S_{pri}(n)\) is geometrically converging.}: \(S_{top}(n) = O(S_{pri}(n))\) and \(Q_{top}(n) = O(Q_{pri}(n) \log_2 n) + O((k/B) \log_2 n)\). Ideally, we would like to prove \(S_{top}(n) = O(S_{pri}(n))\) and \(Q_{top}(n) = O(Q_{pri}(n))\), which would establish the exciting fact that prioritized reporting and top-\(k\) reporting were asymptotically equivalent! When such a perfect reduction still remains elusive, the next natural question is, besides prioritized reporting, what other “assisting problem” needs to be settled before we are guaranteed a top-\(k\) structure with no performance degradation.

1.3 Our Results 1: Reductions

We show that, subject to some mild conditions, there only needs to be an \(O(\log_B n)\) gap in the query cost between top-\(k\) and prioritized reporting:
Theorem 1. Consider a polynomially bounded top-k problem \((\mathbb{D}, \mathbb{Q}, \mathbb{W})\). Suppose that there is a prioritized structure (under \(\mathbb{D}, \mathbb{Q}, \text{ and } \mathbb{W}\)) of \(S_{\text{pri}}(n)\) space and query cost \(Q_{\text{pri}}(n) + O(t/B)\) such that \(S_{\text{pri}}(n)\) is geometrically converging and \(Q_{\text{pri}}(n) \geq \log_B n\). Then, there is a top-k structure of space \(S_{\text{top}}(n)\) and query time \(Q_{\text{top}}(n) + O(k/B)\) where \(S_{\text{top}}(n) = O(S_{\text{pri}}(n))\) and

\[
Q_{\text{top}}(n) = O\left(\frac{\log n}{\log B + \log \frac{Q_{\text{pri}}(n)}{\log_B n}}\right). \tag{1}
\]

As the denominator in (1) is at least \(\log B\), we have \(Q_{\text{top}}(n) = O(Q_{\text{pri}}(n) \cdot \log_B n)\) in any case. Furthermore, if \(Q_{\text{pri}}(n) \geq (n/B)^{\epsilon}\) for an arbitrarily small constant \(\epsilon > 0\), (1) can be simplified into \(Q_{\text{top}}(n) = O(Q_{\text{pri}}(n))\). In other words, the theorem already implies the equivalence between prioritized and top-k reporting for problems demanding expensive query cost.

A top-k query degenerates into a max query when \(k = 1\). A data structure answering max queries will be referred to as a max structure. Just like prioritized reporting, max reporting is also a necessary step before one can settle top-k reporting. Our second contribution is to show that solving both prioritized and max reporting is sufficient in the expected sense:

Theorem 2. Consider an arbitrary top-k problem \((\mathbb{D}, \mathbb{Q}, \mathbb{W})\). Suppose that there is

- a prioritized structure (under \(\mathbb{D}, \mathbb{Q}, \text{ and } \mathbb{W}\)) of \(S_{\text{pri}}(n)\) space that answers a query in \(Q_{\text{pri}}(n) + O(t/B)\) time;
- a max structure (under \(\mathbb{D}, \mathbb{Q}, \text{ and } \mathbb{W}\)) of \(S_{\text{max}}(n)\) space that answers a query (i.e., \(k = 1\)) in \(Q_{\text{max}}(n)\) time. The function \(S_{\text{max}}(n)\) should be \(O(1 + n^2/B)\) and geometrically converging.

Then, there is a top-k structure of expected space \(S_{\text{top}}(n)\) and expected query time \(Q_{\text{top}}(n) + O(k/B)\) where

\[
S_{\text{top}}(n) = O\left(\frac{6n}{B \cdot Q_{\text{pri}}(n)}\right) \tag{2}
\]

\[
Q_{\text{top}}(n) = O\left(\frac{Q_{\text{pri}}(n) + Q_{\text{max}}(n)}{2}\right). \tag{3}
\]

Furthermore, if the prioritized and max structures support an update in \(U_{\text{pri}}(n)\) and \(U_{\text{max}}(n)\) I/Os respectively, then the top-k structure supports an update in \(O(U_{\text{pri}}(n) + U_{\text{max}}(n))\) expected I/Os. If any of \(U_{\text{pri}}(n)\) and \(U_{\text{max}}(n)\) is amortized, the update cost of the top-k structure is amortized expected.

Several remarks are in order:

- The theorem holds regardless of whether the problem is polynomially bounded.
- The above reduction is optimal in the sense that it incurs no performance degradation (in expectation). The space, query, and update costs of the top-k structure are all determined by the worse between the prioritized and max structures.
- Somewhat surprisingly, \(S_{\text{top}}(n)\) may even be smaller than \(O(S_{\text{max}}(n))\). For instance, consider \(S_{\text{pri}}(n) = O(n/B), S_{\text{max}}(n) = O((n/B) \log_B n),\) and \(Q_{\text{pri}}(n) \geq \log_B n;\) these lead to \(S_{\text{top}}(n) = O(n/B)\).
- The condition \(S_{\text{max}}(n) = O(n^2/B)\) essentially captures all the max structures useful in practice. The power 2 is not compulsory and can be replaced by any larger constant without affecting the asymptotic claims, as discussed in Section 4.2.

Both theorems are applicable in the RAM by setting \(B\) to appropriate constants.
1.4 Our Results 2: New Top-\(k\) Structures

Our reductions are useful for solving many top-\(k\) problems. We will discuss several representatives that either prove new results or improve previous work. For this purpose, we classify top-\(k\) problems into two categories. In the first fixed weight category, all the queries choose the same weight function. Equivalently, one can imagine that \(W\) contains a single function \(w\), such that we can as well associate each object \(e \in D\) with a fixed weight \(w(e)\). In the second non-fixed weight category, each query is free to choose its own weight function. All the results stated below in the EM model also hold in the RAM by setting \(B\) to a constant.

1.4.1 Fixed-Weight Problems

**Top-\(k\) interval stabbing.** \(D\) is the set of all intervals in \(\mathbb{R}\), namely, \(D = \{[x, y] \mid x, y \in \mathbb{R}, x \leq y\}\). Each predicate in \(Q\) specifies a real value \(q\) such that an object \([x, y]\) in \(D\) satisfies the predicate if and only if \(q \in [x, y]\). Every object \(e \in D\) carries a (fixed) weight \(w(e)\).

**Theorem 3.** For top-\(k\) interval stabbing, there is an EM structure of \(O(n/B)\) space and \(O(\log_B n + k/B)\) query time. The structure can be updated in \(O(\log_B n)\) amortized I/Os per insertion and deletion. All the complexities hold in expectation.

Our result compares more favorably with a structure of [39] that uses \(O((n/B) \log n)\) space to ensure query time \(O(\log n \cdot \log_B n + k/B)\).

**Top-\(k\) 3D dominance.** \(D\) is the set of points in \(\mathbb{R}^3\). Each predicate in \(Q\) specifies a point \(q = (x_q, y_q, z_q)\) such that an object \(e = (x_e, y_e, z_e)\) in \(D\) satisfies the predicate if and only if \(x_e \leq x_q, y_e \leq y_q,\) and \(z_e \leq z_q\). Every object \(e \in D\) carries a (fixed) weight \(w(e)\).

**Theorem 4.** For top-\(k\) 3D dominance, there is a RAM structure of \(O(n \log n / \log \log n)\) space and \(O(\log^{1.5} n + k)\) query time, both in expectation.

This compares favourably to a structure obtained by combining [39] and [5] that uses \(O(n \log n)\) space and \(O(\log^2 n + k)\) query time.

**Top-\(k\) halfspace reporting.** \(D\) is the set of points in \(\mathbb{R}^d\) where \(d\) is a fixed integer. Each predicate in \(Q\) specifies a halfspace, i.e., all the \(p \in \mathbb{R}^d\) satisfying \(p \cdot q \geq c\), where \(q\) and \(c\) are the query parameters (\(p\) and \(q\) are \(d\)-dimensional vectors, and \(c\) a real value). An object \(e \in D\) satisfies the predicate if \(e\) falls in the halfspace. Every object \(e \in D\) carries a (fixed) weight \(w(e)\).

**Theorem 5.** For top-\(k\) halfspace reporting:

- when \(d = 2\), there is a RAM structure of \(O(n)\) space and \(O(\log n + k)\) query time, both in expectation.
- when \(d \geq 4\), there is a RAM structure of \(O(n \log n)\) space and \(\tilde{O}(n^{1 - 1/\lceil d/2 \rceil} + O(k)\) query time, both in the worst case; there is an EM structure of \(O(n/B)\) space and \(O((n/B)^{1 - 1/\lceil d/2 \rceil + \epsilon} + k/B)\) query time, both in the worst case, where \(\epsilon > 0\) is an arbitrarily small constant.

The first bullet compares favourably to a structure obtained by combining [5], [21], and [39] that uses \(O(n \log n)\) space and answers a query in \(O(\log^2 n + k)\) time. For \(d \geq 4\), [39] gave a RAM structure that, when matching our query time, requires \(O(n^{1+\epsilon})\) space for some constant \(\epsilon > 0\).
1.4.2 Non-Fixed Weight Problems

**Linear ranking.** $\mathbb{D}$ is the set of points in $\mathbb{R}^d$ for some fixed integer $d$. $\mathbb{Q} = \{\text{true}\}$, namely, $\mathbb{Q}$ has only a single predicate which always evaluates to $1$. $\mathbb{W}$ is the set of linear functions of the form $w(p) = c \cdot p$ where $c$ is a parameter of $w$ and $p$ is a $d$-dimensional point (both $c$ and $p$ are $d$-dimensional vectors).

**Theorem 6.** For top-$k$ linear ranking:

- when $d = 2$, there is an EM structure of $O(n/B)$ space and $O(\log_B n + k/B)$ query time, both in expectation.
- when $d \geq 4$, there is a RAM structure of $O(n)$ space and $\tilde{O}(n^{1-1/|d/2|}) + O(k)$ query time, both in the worst case; there is an EM structure of $O(n/B)$ space and $O((n/B)^{1-1/|d/2|+\epsilon} + k/B)$ query time, both in the worst case, where $\epsilon > 0$ is an arbitrarily small constant.

**$L_\infty$ k-nearest neighbor search in 2D.** $\mathbb{D}$ is the set of points in $\mathbb{R}^2$. Given a point $q \in \mathbb{R}^2$ and an integer $k$, a query returns the $k$ points $p \in D$ with the smallest $L_\infty(p, q)$, which is the $L_\infty$-distance between $p$ and $q$. To map the problem into our top-$k$ framework, set $\mathbb{Q} = \{\text{true}\}$; design $\mathbb{W}$ such that each function $w \in \mathbb{W}$ is parameterized by a point $q \in \mathbb{R}^2$ and outputs $w(p) = -L_\infty(p, q)$ for each $p \in \mathbb{D}$.

**Theorem 7.** For $L_\infty$ k-nearest neighbor search in 2D, there is an EM structure of $O(n/B)$ space that answers a query in $O(\log_B n + k/B)$ I/Os. All the complexities hold in expectation.

## 2 Previous Work

The existing research on top-$k$ can be classified into two categories: (i) general reductions to existing problems (our paper belongs to this category) and (ii) tailored-made structures for individual problems.

The first category, to our knowledge, contains only the work [39] of Rahul and Janardan. Besides the contributions mentioned in Section 1.2, they also showed that top-$k$ reporting can be converted to approximate counting\(^2\) and conventional reporting queries. Consider a reporting problem defined by $\mathbb{D}$ and $\mathbb{Q}$ (see Section 1). Given a predicate $q \in \mathbb{Q}$, an approximate counting query returns a value between $|q(D)|$ and $c \cdot |q(D)|$ for some constant $c > 1$. Suppose that we have a structure (i) of space $S_{\text{rep}}(n)$ answering a reporting query in $Q_{\text{rep}}(n) + O(t/B)$ time, where $t$ is the number of objects reported, and (ii) of space $S_{\text{cnt}}(n)$ space answering an approximate counting query in $Q_{\text{cnt}}(n)$ time. Then, there is a top-$k$ structure with space $S_{\text{top}}(n)$ and query time $Q_{\text{top}}(n) + O(k/B)$ where

\[
S_{\text{top}}(n) = O((S_{\text{rep}}(n) + S_{\text{cnt}}(n)) \log n)
\]
\[
Q_{\text{top}}(n) = O((Q_{\text{rep}}(n) + Q_{\text{cnt}}(n)) \log n);
\]

note that the logarithms have base 2.

Regarding the second category, [28] is an excellent survey on top-$k$ research that focuses on system implementation, rather than achieving attractive performance guarantees. In the theory line, the work of [15] is the first to incorporate top-$k$ features into conventional reporting queries. Since then, the topic has grown into a sizeable literature. The most extensively studied problem is top-$k$ range reporting, whose 1D version was studied in [3,16,17,44,46], and 2D version in [39,40]. The recent work

\(^2\)The work [39] actually requires exact counting, which, as discussed here, is not necessary.
of [19] developed efficient RAM structures [19] for the top-k 2D rectangle stabbing problem and the top-k 2D point enclosure problem. The theory community has also studied top-k problems outside the class formulated in Section 1, e.g., colored reporting [36,41], text retrieval [26,29,31,35–37,43], and uncertain data search [8,11,47], etc.

3 Reduction with Worst-Case Efficiency

This section presents our first reduction and serves as a proof of Theorem 1. Our reduction utilizes k

3.1 Top-k Core-Set

**Rank sampling.** Let S be a set of objects drawn from an ordered domain. An object e ∈ S has rank i if it is the i-th greatest in S. By independently sampling each object of S with probability p, we obtain a p-sample set R. Intuitively, the object with rank kp in R ought to have rank roughly k in S. The next lemma formalizes this intuition and clarifies the accompanying conditions.

**Lemma 8.** Let S be a set of n objects and R be a p-sample set of S. Let k be an integer and δ be a real value satisfying 1 ≤ k ≤ n/4, 0 < δ < 1, and kp ≥ 3 ln(3/δ). Then, the following hold simultaneously with probability at least 1 − δ:

1. |R| > 2kp;
2. the object with rank ⌈2kp⌉ in R has rank between k and 4k in S.

**Proof.** We will need the Chernoff bounds given in the appendix. The first bullet fails with probability

\[
\Pr[|R| \leq 2kp] = \Pr[|R| \leq (2k/n) \cdot np] \leq \Pr[|R| \leq (1/2)np] \\
(\text{Chernoff bound (12)}) \leq \exp(-np/12) \leq \exp(-kp/3) \leq \delta/3.
\]

Let e be the object with rank ⌈2kp⌉ in R and ̂k be the rank of e in S. Next, we bound the probability of the event ̂k > 4k. For i ∈ [1,4k], define xi to be 1 if the i-th greatest object in S is sampled, or 0 otherwise. Let X = \(\sum_{i=1}^{4k} x_i\) and, thus, E[X] = 4kp ≥ 12 ln(3/δ). Event ̂k > 4k implies X ≤ ⌈2kp⌉ − 1. We have:

\[
\Pr[̂k > 4k] \leq \Pr[X \leq ⌈2kp⌉ - 1] = \Pr[X < 2kp] = \Pr[X < (1/2) \cdot E[X]] \\
(\text{Chernoff bound (12)}) \leq \exp(-E[X]/12) \leq \delta/3.
\]

Finally, we bound the probability of the event ̂k < k. Define Y = \(\sum_{i=1}^{k} x_i\) and, thus, E[Y] = kp ≥ 3 ln(3/δ). Event ̂k < k implies that Y ≥ ⌈2kp⌉. We have:

\[
\Pr[̂k < k] \leq \Pr[Y \geq 2kp] = \Pr[Y \geq 2 E[Y]] \\
(\text{Chernoff bound (13)}) \leq \exp(-E[Y]/3) \leq \delta/3.
\]

By the union bound, the two bullets in the lemma hold simultaneously with probability at least 1 − δ.

**Core-set.** Consider a λ-polynomially bounded top-k problem (D, Q, W). Let R be a subset of the input D. Given a weight function w ∈ W, an object in R has w-rank i in R if it has the i-th largest w-weight in R. The next lemma proves the existence of a small-size core-set that approximately captures a specific rank for all the weight functions and all the “large” queries.
Lemma 9 (Top-k Core-Set Lemma). Let \((\mathcal{D}, \mathcal{Q}, \mathcal{W})\) be a \(\lambda\)-polynomially bounded top-\(k\) problem and \(D\) be a subset of \(\mathcal{D}\) with \(n \geq 6\) objects. For any integer \(K \geq 8\lambda \ln n\), there is an \(R \subseteq D\) such that

- \(|R| \leq 24\lambda \cdot (n/K) \ln n;\)
- for any weight function \(w \in \mathcal{W}\) and any \(q \in \mathcal{Q}\) satisfying \(|q(D)| \geq 4K\), it holds that
  - \(|q(R)| > 16\lambda \ln n;\)
  - the object with \(w\)-rank \([16\lambda \ln n]\) in \(q(R)\) has \(w\)-rank between \(K\) and \(4K\) in \(q(D)\).

Proof. Set \(p = (8\lambda/K) \ln n\) and \(\delta = 1/(2n^{2\lambda})\). These values ensure:

\[Kp = 8\lambda \ln n \geq 3\ln(3/\delta)\]  

(4)

for \(n \geq 6\). Let \(R\) be a \(p\)-sample set of \(D\). We will prove that \(R\) satisfies all the conditions in the lemma with a non-zero probability.

Fix a weight function \(w \in \mathcal{W}\) and a predicate \(q \in \mathcal{Q}\) satisfying \(|q(D)| \geq 4K\). Clearly, \(q(R)\) is a \(p\)-sample set of \(q(D)\). Applying Lemma 8 on \(S = q(D)\) (the application is enabled by (4)), we assert that the following hold simultaneously with probability at least \(1 - \delta\):

- \(|q(R)| > 2Kp = 16\lambda \ln n.\)
- The object with \(w\)-rank \([2Kp]\) has \(w\)-rank between \(K\) and \(4K\) in \(q(D)\).

By \(\lambda\)-polynomially boundedness and the union bound, the above holds for all the qualifying pairs \((w, q)\) with probability at least \(1 - \delta n^{2\lambda} = 1/2\) (it suffices to consider at most \(n^{2\lambda}\) “effective” pairs that differ either in \(q(D)\) or the weight ordering under \(w\)).

Finally, as \(|R| = np\) in expectation, by Markov’s inequality, \(|R| \leq 3np = 24\lambda(n/K) \ln n\) with probability at least \(2/3\). It thus follows that all the conditions of the lemma hold with probability at least \(1 - (1/2 + 1/3) > 0\). \(\square\)

3.2 The Reduction

High-level overview. Before getting into the full details, we first present a high-level overview of our reduction. For simplicity, our discussion will consider the RAM model and assume the space and query time of the prioritized structure to be \(O(n)\) and \(O(\log n + t)\), respectively (\(t\) is the number of objects reported), i.e., \(S_{pri}(n) = O(n)\) and \(Q_{pri}(n) = \log n\). We will outline the design of a top-\(k\) structure with space \(O(n)\) and query time \(O(\log^2 n + k)\). The overview will concentrate on queries with \(q(D) \geq 4k\); the case of \(q(D) < 4k\) can be handled in a cost monitoring manner as explained in the full reduction.

The most interesting case turns out to be \(k = [48\lambda \ln n]\). The other two cases — \(k < [48\lambda \ln n]\) and \(k > [48\lambda \ln n]\) — will be reduced to the case of \(k = [48\lambda \ln n]\). To handle \(k < [48\lambda \ln n]\), we will issue a top-[48\lambda \ln n] query to find the [48\lambda \ln n] objects with the largest \(w\)-weights in \(q(D)\) and then perform \(k\)-selection [12] to extract the \(k\) objects with the maximum \(w\)-weights. Ignoring the cost of the top-[48\lambda \ln n] query, these steps take \(O(\log n)\) time.

To handle \(k > [48\lambda \ln n]\), it suffices to deal with \(k\) values of the form \(2^i[48\lambda \ln n], i \geq 1\). Indeed, any other \(k\) can be rounded up to the smallest \(2^i[48\lambda \ln n]\); from the top-(2\(^i\)[48\lambda \ln n]) objects reported, we can retrieve the top-\(k\) objects with \(k\)-selection in \(O(2^i[48\lambda \ln n]) = O(k)\) time.
Next, we explain how to support \( k = 2^i [48\lambda \ln n], i \geq 1 \). Fix an \( i \) and \( k = 2^i [48\lambda \ln n] \). Apply Lemma 9 with \( K = 2^i [48\lambda \ln n] \) to obtain a set \( R_i \subseteq D \) and build a structure to handle top-[48\lambda \ln n] queries on \( R_i \). By Lemma 9, the object with w-rank \([16\lambda \ln n]\) in \( q(R_i) \) has w-rank between \( k \) and \( 4k \) in \( q(D) \). By querying the top-[48\lambda \ln n] structure on \( R_i \), we obtain the weight \( \tau \) of the \([16\lambda \ln n]\)-th largest w-weight object in \( q(R_i) \); then, we search the prioritized structure on \( D \) with \( \tau \) followed by a k-selection step to report the top-k objects. As the sizes of \( R_1, R_2, \ldots \) form a geometrically decreasing sequence, the space occupied by the structures on all the \( R_i \)'s is \( O(n) \). Ignoring the time to perform the top-[48\lambda \ln n] query, the remaining steps take \( O(\log n + k) \) time.

Finally, we are left with the case of \( k = [48\lambda \ln n] \). Once again, we make use of Lemma 9. Applying the lemma with \( K = [48\lambda \ln n] \) on \( D \) yields a set \( R \subseteq D \) such that \( |R| \leq |D|/2 \) (this explains why the constant 48). If we have a structure to answer top-[48\lambda \ln n] queries on \( R \), then using steps similar to the ones explained earlier for \( k > [48\lambda \ln n] \), one can report the top-[48\lambda \ln n] objects in \( D \). Ignoring the time taken to query the top-[48\lambda \ln n] structure on \( R \), this can be done in \( O(\log n + k) = O(\log n) \) time.

Therefore, the original problem of finding top-[48\lambda \ln n] objects in \( D \) has been reduced to top-[48\lambda \ln n] search on \( R \). Noticing that the problem size has reduced by a factor of at least 2, we can handle the new subproblem on \( R \) via recursion, until \( O(\log n) \) objects are left, in which case a query can be answered with brute force. Ignoring the output size term, the query time \( Q(n) \) satisfies the following recurrence:

\[
Q(|D|) = O(\log n) + Q(|R|) \implies Q(n) = O(\log n) + Q(n/2) = O(\log^2 n).
\]

In summary, we have a top-k structure which uses \( O(n) \) space and query time \( O(\log^2 n + k) \).

**The full reduction.** We will now present the complete details of how to use a prioritized structure to design a top-k structure for a \( \lambda \)-polynomially bounded problem \((D, Q, W)\). A brief summary can be found in Figure 1.

Recall that the prioritized structure consumes \( S_{pri}(n) \) space and answers a prioritized query in \( Q_{pri}(n) + O(t/B) \) I/Os. Define

\[
g = \frac{Q_{pri}(n)}{\log_B n} \quad (5)
\]

\[
f = 24\lambda B \cdot Q_{pri}(n). \quad (6)
\]

Note that \( g \geq 1 \) (as required by Theorem 1) and both the following hold for a sufficiently large \( B \):

\[
\frac{24\lambda}{f} \cdot \ln n \leq \frac{1}{g \sqrt{B}} \quad (7)
\]

\[
f \geq [16\lambda \ln n]. \quad (8)
\]

We will first discuss top-k queries with \( k \leq f \) and then attend to queries with larger \( k \). Remember that, besides \( k \), a query also designates a predicate \( q \in Q \) and a weight function \( w \in W \).

**Queries with \( k \leq f \).** It suffices, in fact, to consider \( k = f \). Given a query with \( k < f \), we first issue a top-f query to find the \( f \) objects with the largest \( w \)-weights in \( q(D) \), and then perform \( k \)-selection \([12]\) in \( O(f/B) = O(Q_{pri}(n)) \) I/Os to extract the \( k \) objects with the maximum \( w \)-weights. The total cost is the time of the top-f query plus \( O(Q_{pri}(n)) \).
Solving \( k = f \)

<table>
<thead>
<tr>
<th>Top-f structure on ( D )</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Lemma 9 produces a sample ( R \subseteq D ).</td>
<td>1) Top-f query on ( q(R) ).</td>
</tr>
<tr>
<td>2) Recursive structure on ( R ).</td>
<td>2) ( \tau ) ← ( \lceil 16\lambda \ln n \rceil )-th ( w )-rank object in ( q(R) ).</td>
</tr>
<tr>
<td>3) Prioritized structure on ( D ).</td>
<td>3) Query prioritized structure on ( D ) with ( \tau ).</td>
</tr>
</tbody>
</table>

Figure 1: Summary of our reduction (\( f = 24\lambda B \cdot Q_{pri}(n) \))

At a high level, our strategy for answering a top-\( f \) query is as follows. If \(|q(D)| \) is small, we can retrieve the entire \( q(D) \) and then easily find the one with \( w \)-rank \( f \). The brute force method is too expensive for large \( q(D) \), but in such a situation \( q(D) \) is eligible for Lemma 9. This motivates us to identify some object \( e \in q(D) \) with \( w \)-rank in \([f, 4f]\) and then apply \( f \)-selection on those objects in \( q(D) \) with \( w \)-weights at most \( w(e) \). Lemma 9 with the help of (8) indicates that the retrieval of \( e \) is another top-\( f \) query on a subset of \( D \). This naturally points to a recursive approach to construct our structure.

If \(|q(D)| \leq 4f\), we answer a top-\( f \) query using a prioritized structure on \( D \) directly. For this purpose, we do not need any counting structure for estimating \(|q(D)| \). Instead, we can perform a prioritized query (defined in Section 1.2) with predicate \( q \) and threshold \( \tau = -\infty \) in a cost monitoring manner: either the query terminates by itself, or we terminate it manually as soon as \( 4f + 1 \) objects have been reported. The number of I/Os is at most \( Q_{pri}(n) + O(f/B) = O(Q_{pri}(n)) \). In the former case, we obtain the result of the top-\( f \) query by performing \( f \)-selection on the \( 4f + 1 \) objects fetched, while in the latter case it must hold that \(|q(D)| > 4f\).

For queries with \(|q(D)| > 4f\), we build a set of structures as follows. First, take a core-set \( R_1 \) of \( D \) using Lemma 9 with \( K = f \) and build a prioritized structure on \( R_1 \). This process is then carried out recursively: for every \( i \geq 2 \), take a core-set \( R_{i+1} \) of \( R_i \) with \( K = f \) and build a prioritized structure on \( R_{i+1} \). The recursion ends at some \( i = h \) where \(|R_h| \leq 4f\).

For convenience, let us treat \( D \) as \( R_0 \). For each \( R_i \) \((0 \leq i \leq h - 1)\), \( f \geq 8\lambda \ln n \geq 8\lambda \ln |R_i| \). Hence, Lemma 9 shows

\[
|R_{i+1}| \leq \frac{24\lambda \cdot |R_i|}{f} \ln |R_i| \leq \frac{24\lambda \cdot |R_i|}{f} \ln n \leq \frac{|R_i|}{g\sqrt{B}} \tag{9}
\]

where the last inequality used (7). The total space of all the prioritized structures is \( O(S_{pri}(n)) \) because \( S_{pri}(n) \) is geometrically converging. Furthermore, (9) indicates that \( h = O(\log_{g\sqrt{B}} n) \).
Lemma 10. Consider any top-$f$ query with predicate $q$ and weight function $w$. For $i \in [0,h]$, the query’s result on $R_i$ can be computed in $c \cdot (h-i+1) \cdot Q_{pri}(n)$ I/Os for some constant $c \geq 1$.

Proof. For $i = h$, we can do so by performing $f$-selection on $R_h$ directly in $O(f/B)$ I/Os, which is at most $c_1 \cdot Q_{pri}(n)$ for some constant $c_1$. Inductively, assuming that the claim is true for all $i \geq j+1$ (where $1 \leq j \leq h-1$), next we will prove its correctness for $i = j$.

Our algorithm proceeds differently in two scenarios.

- Case $|q(R_j)| \leq 4f$: We use the prioritized structure on $R_j$ to solve the top-$f$ query (on $R_j$) in the cost monitoring manner explained earlier. The cost is $Q_{pri}(|R_j|) + O(f/B) \leq c_2 \cdot Q_{pri}(n)$ for some constant $c_2$.

- Case $|q(R_j)| > 4f$: According to Lemma 9, $|q(R_{j+1})| \geq \lceil 16\ln |q(R_j)| \rceil$; meanwhile, (8) ensures $f \geq \lceil 16\ln |q(R_j)| \rceil$. Equipped with these facts, we answer the top-$f$ query on $R_j$ as follows. First, retrieve the object $e$ with $w$-weights at least $w(e)$ in $Q_{pri}(|R_j|)$ by issuing a top-$f$ query on $R_{j+1}$; this incurs $c \cdot (h-j) \cdot Q_{pri}(n)$ I/Os by the inductive assumption. By Lemma 9, the $w$-rank of $e$ in $q(R_j)$ is between $f$ and $4f$. Then, we deploy the prioritized structure on $R_j$ to fetch all the objects of $q(R_j)$ with $w$-weights at least $w(e)$ in $Q_{pri}(|R_j|) + O(f/B) \leq c_2 \cdot Q_{pri}(n)$ I/Os. The query result (on $R_j$) can now be obtained from these objects with $f$-selection in at most $c_3 \cdot Q_{pri}(n)$ I/Os for some constant $c_3$.

We choose $c$ to be $\max\{c_1, c_2 + c_3\}$, which ensures
\[
c \cdot (h-j) \cdot Q_{pri}(n) + (c_2 + c_3) \cdot Q_{pri}(n) \leq c \cdot (h-j+1) \cdot Q_{pri}(n)
\]
thereby establishing Lemma 10. \hfill \square

Hence, the cost of answering a top-$f$ query on $D$ is $O(h \cdot Q_{pri}(n)) = O(Q_{pri}(n) \cdot \log_{g\sqrt{B}} n)$. Plugging in the definition (5) for $g$ gives the query complexity claimed in Theorem 1.

Queries with $k > f$. We apply Lemma 9 to take a core-set $R[i]$ of $D$ with $K = 2^{i-1}f$ for each $i \in [1,h']$, where $h'$ is the largest integer $i$ satisfying $2^{i-1}f \leq n$. It is easy to verify from (6) that $h' = O(\log(n/B))$. Our structure has two components:

- A prioritized structure on $D$.

- On each $R[i]$ where $1 \leq i \leq h'$, build a top-$f$ structure, namely, the structure we just explained for answering queries with $k \leq f$. Since $|R[i]| \leq 24\lambda \cdot \frac{n}{2^{i-1}f} \cdot \ln n$, all these top-$f$ structures use in total
\[
O\left(\sum_{i=1}^{h'} S_{pri} \left( \frac{24\lambda \ln n}{2^{i-1}f} \right) \right) = O(S_{pri}(n) + h') = O(S_{pri}(n))
\]
space, where the derivation used the facts that (i) $S_{pri}(n)$ is geometrically converging, (ii) $Q_{pri}(n) = \Omega(\log_B n)$, and (ii) $S_{pri}(n)$ obviously needs to be $\Omega(n/B)$ (implying $h' = O(S_{pri}(n))$).

The total space of our structure is therefore $O(S_{pri}(n))$.

Now consider a top-$k$ query $q$ with $f < k \leq n$. First, if $k \geq n/2$, we answer it by performing $k$-selection on the entire $D$ in $O(n/B) = O(k/B)$ I/Os. Consider now $k < n/2$. Identify the smallest $i \in [1,h']$ such that $2^{i-1}f \geq k$. Fix the value of $K$ to $2^{i-1}f$ in the rest of the section; note that $k \leq K < 2k$. Then:
• if $|q(D)| \leq 4K$, we use the prioritized structure on $D$ to answer the query with $Q_{\text{pri}}(n) + O(K/B)$ I/Os in the cost monitoring manner explained earlier.

• if $|q(D)| > 4K$, Lemma 9 indicates that $q(R[i])$ has size at least $\lceil 16\ln n \rceil$, and the object $e$ with $w$-rank $\lceil 16\ln n \rceil$ in $q(R[i])$ has $w$-rank between $K$ and $4K$ in $q(D)$. As $f \geq \lceil 16\ln n \rceil$, we can retrieve $e$ by issuing a top-$f$ query on $R[i]$, which costs $O(Q_{\text{pri}}(n) \log g_{\sqrt{B}} n) I/Os$, as proved earlier. Then, use the prioritized structure on $D$ to extract the objects in $q(D)$ whose $w$-weights are at least $w(e)$, which entails $Q_{\text{pri}}(n) + O(K/B)$ I/Os. Finally, the query result can be produced with $k$-selection in $O(K/B)$ I/Os.

Overall, the query cost is $O(Q_{\text{pri}}(n) \log g_{\sqrt{B}} n + K/B)$. This completes the whole proof of Theorem 1.

4 Reduction with Expected Efficiency

This section presents our second reduction and serves as a proof of Theorem 2. We are given (i) a prioritized structure of $S_{\text{pri}}(n)$ space answering a prioritized query in $Q_{\text{pri}}(n) + O(t/B)$ I/Os, and (ii) a max structure of $S_{\text{max}}(n)$ space answering a max query in $Q_{\text{max}}(n)$ I/Os. The objective is to design a top-$k$ structure with no performance degradation (in expectation). We will focus on $n \geq B \cdot Q_{\text{max}}(n)$; otherwise, a top-$k$ query can be trivially answered by performing $k$-selection on the whole $D$ in $O(n/B) = O(Q_{\text{max}}(n))$ I/Os.

4.1 Rank Sampling, Again

Our rank sampling result in Lemma 8 falls short for the subsequent discussion. Notice that the lemma is concerned with the sample with rank $\lceil 2kp \rceil$ which, under the working condition $kp \geq 3\ln(3/\delta)$, is strictly greater than $1$. This is problematic because, to apply max (i.e., top-1) queries, we need to understand the behavior of rank 1 in the sample set. Fortunately, we can prove an alternative result, which is less general than Lemma 8 but serves the purpose of understanding rank 1 in the sample set.

**Lemma 11.** Let $S$ be a set of $n$ objects, $K$ be a real value satisfying $2 \leq K \leq n/4$, and $R$ be a $(1/K)$-sample set of $S$. The following hold simultaneously with probability at least 0.09:

• $|R| \geq 1$;

• if $e$ is the largest object in $R$, then the rank of $e$ in $S$ is greater than $K$ but at most $4K$.

**Proof.** The first bullet fails only if no objects in $D$ are sampled, which occurs with a probability

$$(1 - 1/K)^n \leq (1 - 1/K)^{4K} \leq 1/e^4$$

where the last inequality used the fact that $(1 - x)^1/x < 1/e$ for all $x > 0$.

Consider the largest object $e$ in $R$. Denote by $\tilde{K}$ the rank of $e$ in $D$. The event $\tilde{K} > 4K$ occurs only if none of the $4K$ largest objects in $D$ are sampled. Hence:

$$\Pr[\tilde{K} > 4K] = (1 - 1/K)^{4K} \leq 1/e^4.$$ 

On the other hand, the event $\tilde{K} \leq K$ occurs only if at least one of the $K$ largest objects in $D$ was sampled. Hence:

$$\Pr[\tilde{K} \leq K] = 1 - (1 - 1/K)^K \leq 1 - 1/e^2.$$ 

12
where the last inequality used the fact that \((1 - 1/x)^x \geq 1/e^2\) for all \(x \geq 2\). By the union bound, the probability of violating at least one bullet of Lemma 11 is at most

\[
2/e^4 + (1 - 1/e^2) < 0.91
\]

which completes the proof. \(\Box\)

4.2 The Reduction

We now describe how to construct a top-k structure from the available prioritized and max structures.

**Structure.** Fix a constant \(\sigma = 1/20\) and define for each integer \(i \geq 1\):

\[
K_i = B \cdot Q_{\text{max}}(n) \cdot (1 + \sigma)^{i-1}.
\]

Let \(h\) be the largest \(i\) such that \(K_i \leq n/4\; \text{clearly}, \; h = O(\log(n/B))\). Our structure includes:

- a prioritized structure on \(D\), and
- a max structure on \(R_i\) for each \(i \in [1, h]\), where \(R_i\) is a \((1/K_i)\)-sample set of \(D\).

**Query.** Recall that a query chooses a predicate \(q \in \mathbb{Q}\), a weight function \(w \in \mathbb{W}\), and an integer \(k\). If \(k < B \cdot Q_{\text{max}}(n)\), we treat the query as a top-\((B \cdot Q_{\text{max}}(n))\) query instead, i.e., extracting the \(B \cdot Q_{\text{max}}(n)\) objects with the largest \(w\)-weights in \(q(D)\). The original query’s result can then be obtained with \(k\)-selection. The cost is \(O(Q_{\text{max}}(n))\) plus the time of the top-\((B \cdot Q_{\text{max}}(n))\) query.

Next, we focus on top-k queries with \(k \geq B \cdot Q_{\text{max}}(n)\). If \(k > K_h\), the query is answered naively by performing \(k\)-selection on \(D\) in \(O(n/B)\) I/Os, which is \(O(k/B)\) because \(k > K_h \geq n/(4(1 + \sigma)) = \Omega(n)\).

Consider now \(k \leq K_h\). Let \(i^*\) be the smallest \(i\) with \(K_i \geq k\); note that \(K_i^* = \Theta(k)\). Starting with \(j = i^*\), we carry out a round as follows:

1. If \(|q(D)| \leq 4K_j\), solve the query with the prioritized structure on \(D\) in the cost-monitoring manner (Section 3.2) with cost \(Q_{\text{pri}}(n) + O(K_j/B)\). The algorithm then declares the round successful and terminates.

2. Otherwise, identify the object \(e \in q(R_j)\) with the maximum \(w\)-weight from the max structure on \(R_j\) in \(Q_{\text{max}}(n)\) I/Os. In the special case where \(q(R_j)\) is empty, treat \(e\) as a dummy object with \(w(e) = -\infty\).

3. Perform a prioritized query on \(D\) with \(q\) and threshold \(\tau = w(e)\) in a cost-monitoring manner:
   (a) either the query terminates by itself, in which case we define \(S\) as the set of objects retrieved,
   (b) or we terminate it as soon as \(4K_j + 1\) objects have been reported.

   The cost is \(Q_{\text{pri}}(n) + O(K_j/B)\) in both cases.

4. Declare this round failed in either situation below:
   - Case 3(a) occurred but \(|S| \leq K_j\).
   - Case 3(b) occurred.
Therefore, with probability at least 1 − which is O(K) bound (13), we have:

\[ n \text{ independent Bernoulli variables each taking the value of 1 with probability } 1 - \frac{O}{K} \] 

occupy \( n \) \( O(S) \) as \( S \) space.

Q succeeds, performs \( Q_{\text{pri}}(n) + Q_{\max}(n) + O(K_j/B) \) I/Os. Thus, the algorithm’s expected cost is

\[
\sum_{j=i^*}^{h} O \left( \left( Q_{\text{pri}}(n) + Q_{\max}(n) + \frac{K_j}{B} \right) \cdot 0.91^{j-i^*} \right) = O \left( Q_{\text{pri}}(n) + Q_{\max}(n) + \sum_{j=i^*}^{h} \frac{K_j}{B} 0.91^{j-i^*} \right). \tag{10}
\]

Plugging in \( K_j = K_i \cdot (1 + \sigma)^{j-i^*} = O(k) \cdot (1 + \sigma)^{j-i^*} \) gives

\[
(10) = O \left( Q_{\text{pri}}(n) + Q_{\max}(n) + \frac{k}{B} \sum_{j=i^*}^{h} (1 + \sigma) \cdot 0.91^{j-i^*} \right)
\]

which is \( O(Q_{\text{pri}}(n) + Q_{\max}(n) + k/B) \) because \( (1 + \sigma) \cdot 0.91 < 1 \).

**Space.** The prioritized structure on \( D \) obviously takes up \( S_{\text{pri}}(n) \) space. We claim that all the max structures occupy \( o(n/B) + O(S_{\max}(\frac{6n}{BQ_{\max}(n)}) \) expected space in total, which implies the space complexity in Theorem 2 (because \( S_{\text{pri}}(n) = \Omega(n/B) \)).

\[
E[|R_i|] = n/K_i \text{ geometrically decreases as } i \text{ increases. This seems to yield the claim immediately as } S_{\max}(n) \text{ is geometrically converging. The complication, however, is that } E[S_{\max}(|R_i|)] \text{ is not necessarily } O(S_{\max}(E[|R_i|])). To circumvent the issue, we will prove that all the max structures occupy } o(n/B) + O(S_{\max}(\frac{6n}{BQ_{\max}(n)}) \text{ space in total with probability at least } 1 - 1/n^2. \text{ This will establish our claim on the expected space consumption because all these structures obviously demand no more than } O(S_{\max}(n \cdot h) = O((n^2/B) \cdot \log(n/B)) \text{ space (Theorem 2 has the condition that } S_{\max}(n) = O(1 + n^2/B)) \text{.}

Let \( i' \) be the largest \( i \) such that \( K_i \leq n/(3\ln n) \). Consider an \( i \in [1, i'] \). Since \( |R_i| \) is the sum of \( n \) independent Bernoulli variables each taking the value of 1 with probability \( 1/K_i \), by Chernoff bound (13), we have:

\[
\Pr[|R_i| \geq 6 \cdot E[|R_i|]] \leq \exp(-E[|R_i|]) = \exp(-n/K_i) \leq 1/n^3.
\]

Therefore, with probability at least \( 1 - h/n^3 \), the max structures on \( R_1, R_2, ..., R_{i'} \) use at most

\[
\sum_{i=1}^{i'} O \left( S_{\max} \left( \frac{6n}{B \cdot Q_{\max}(n)} \right) \right) = O \left( h + S_{\max} \left( \frac{6n}{B \cdot Q_{\max}(n)} \right) \right)
\]

space overall because \( S_{\max}(\cdot) \) is geometrically converging.
Let us now concentrate on $i \in [i' + 1, h]$. Notice that there are only $O(\log \log n)$ such values of $i$. Also, by definition of $i'$, we know that $E[|R_i|] = n/K_i$ is $O(\log n)$ but at least 4 (recall that $K_i \leq n/4$). Again, by Chernoff bound (13), we have:

$$\Pr[|R_i| \geq (\ln n^4) \cdot E[|R_i|]] \leq \exp(-((\ln n^4) \cdot E[|R_i|]/6)$$

(by $E[|R_i|] \geq 4$) \leq \exp((-\ln n^{12/3})) = 1/n^{8/3}.

Hence, with probability at least $1 - O(\log \log n)/n^{8/3}$, $|R_i| \leq 4 \ln n \cdot E[|R_i|] = O(\log^2 n)$ holds for all $i \in [i' + 1, h]$. As $S_{\max}(n) = O(1 + n^2/B)$, the max structures on $R_{i'}, R_{i'+1}, \ldots, R_h$ together consume $O(h + (\log \log n \cdot \log^4 n)/B) = o(n/B)$ space, applying the fact that $S_{\max}(n) = O(n^2/B)$. We now conclude that, with probability at least $1 - h/n^3 - O(\log \log n)/n^{8/3} > 1 - 1/n^2$, all the max structures use $o(n/B) + O(S_{\max}(\frac{6n}{B \cdot Q_{\max}(n)}))$ space.

**Update.** In expectation, each object $e \in D$ has only $O(1)$ copies in the entire structure (the sampling rate $1/K_i$ of $R_i$ geometrically decreases as $i$ increases). Hence, the insertion of $e$ triggers one insertion into the prioritized structure and one insertion into $O(1)$ max structures in expectation. The total cost is thus $O(U_{\text{pri}} + U_{\text{max}})$ expected. We can record in $O(1)$ expected words which max structures include $e$. By hashing, such bookkeeping information can be maintained in $O(1)$ expected I/Os as $e$ is inserted/deleted, without increasing the overall space complexity. In this way, a deletion of $e$ can also be supported in $O(U_{\text{pri}} + U_{\text{max}})$ expected I/Os. The above argument still works even if one or both of $U_{\text{pri}}$ and $U_{\text{max}}$ are amortized. This completes the whole proof of Theorem 2.

**Remarks.** The above reduction may be reminiscent of a method by Aronov and Har-Peled [14] that reduces approximate counting to emptiness detection. Our approach differs in both algorithmic and technical details, a quick proof of which is the following fact: the counting structure of [14] suffers from performance degradation by a logarithmic factor compared to the emptiness structure, while Theorem 2 incurs no performance degradation as explained in Section 1.3.

Another remark concerns relaxing the assumption $S_{\max}(n) = O(n^2/B)$. The constant 2 can be replaced with any constant $\gamma > 2$. We needed the assumption only because the space of all the max structures was shown to be $o(n/B) + O(S_{\max}(\frac{6n}{B \cdot Q_{\max}(n)}))$ with probability at least $1 - 1/n^2$. If $S_{\max}(n) = O(n^\gamma/B)$, all we have to do is to reduce the failure probability from $1/n^2$ to $1/n^\gamma$. The details are standard and omitted.

5 “Easy” Top-$k$ Structures from Our Reductions

Theorems 1 and 2 are powerful tools for designing top-$k$ structures with attractive performance guarantees. They allow us to focus on solving the corresponding prioritized queries and perhaps also the max queries. Next, we will demonstrate how “enjoyable” it is to obtain most of the new top-$k$ structures claimed in Section 1.4. Note that the top-$k$ problems discussed below appear “easy” only because of our reductions, without which they would require (much) more effort.

The following subsections are arranged in ascending order of sophistication. We will, however, not prove the first bullet of Theorem 5 in this section. 2D top-$k$ halfspace reporting requires additional non-trivial ideas to be developed in the next section.

5.1 Top-$k$ Interval Stabbing (Theorem 3)

The problem’s prioritized version is called my stabbing, for which Tao [45] gave a structure of $O(n/B)$ size answering a query in $O(\log_B n + t/B)$ I/Os and supporting an update in $O(\log_B n)$ amortized
I/Os. The problem’s max version has been studied by Agarwal et al. [7], who gave a structure of $O(n/B)$ size answering a query in $O(\log_B n)$ I/Os and supporting an update in $O(\log_B n)$ amortized I/Os. Theorem 3 now becomes a corollary of Theorem 2.

5.2 Linear Ranking (Theorem 6)

The problem’s prioritized version is simply *halfspace reporting* (see Section 1.4), while the corresponding max queries are known as *extreme-point queries*.

For $d = 2$, Agarwal et al. [6] gave a structure of $O(n/B)$ size that supports halfspace reporting in $O(\log_B n + t/B)$ I/Os. It is well-known that an extreme-point query can be answered in $O(\log_B n)$ I/Os by a B-tree storing the boundary of the convex hull of the input $D$. The first bullet of Theorem 6 is now a corollary of Theorem 2.

Consider now $d \geq 4$. In the RAM model, Afshani and Chan [4] gave a structure of $O(n)$ size that answers a halfspace reporting query in $O(n^{1-1/\lfloor d/2 \rfloor} + t/B)$ time. In the EM model, Agarwal et al. [6] gave a structure of $O((n/B)^{1-1/\lfloor d/2 \rfloor} + t/B)$ I/Os. Plugging these into Theorem 1 gives the second bullet of Theorem 6.

5.3 $L_\infty k$-Nearest Neighbor (Theorem 7)

The problem’s prioritized version is *orthogonal range reporting* where a query specifies an axis-parallel *square* $q$ in $\mathbb{R}^d$ and reports all the points in $D \cap q$. In the RAM, Chazelle and Edelsbrunner [20] gave a structure of $O(n)$ space that answers an orthogonal range reporting query in $O(\log n + t)$ time. Their solution essentially reduces the problem to *3D dominance reporting*, for which Afshani [1] gave an EM structure of $O(n/B)$ space and $O(\log_B n + t/B)$ query time. The problem’s max version is the standard $L_\infty$ nearest neighbor search. It is well-known that we can store $D$ in an EM structure of $O(n/B)$ space that answers an $L_\infty$ nearest neighbor query in $O(\log_B n)$ time [13, 27, 30]. Theorem 7 then becomes a corollary of Theorem 2.

5.4 Top-$k$ 3D Dominance (Theorem 4)

The problem’s prioritized version of the problem is called *4D dominance reporting*. Afshani et al. [2] gave a RAM structure with size $O(n \log n / \log \log n)$ and query time $O(\log^{1.5} n + t)$. In the max version, $D$ is a set of $n$ points in $\mathbb{R}^3$ each carrying a real-valued weight. Given a point $q = (x, y, z) \in \mathbb{R}^3$, a max query reports the maximum weight of the points $e = (e_x, e_y, e_z) \in D$ satisfying $e_x \leq x$, $e_y \leq y$, and $e_z \leq z$. Next, we will develop a structure for this problem that uses $O(n)$ space and answers a query in $O(\log n)$ time, after which Theorem 4 will follow from Theorem 2.

![Figure 2](image_url)

Figure 2: (a) a set of five points, (b) region $\rho_i$ associated with each point $e_i$, and (c) vertical decomposition.
Let $e_1, e_2, \ldots, e_n$ be the sequence of points in descending order of weight. With each point $e_i$, we associate a region $\rho_i$ in $\mathbb{R}^3$ satisfying the following constraint: a point $q = (x, y, z) \in \mathbb{R}^3$ belongs to $\rho_i$ if and only if $e_i$ is the point with the maximum weight in $(-\infty, x] \times (-\infty, y] \times (-\infty, z]$. The following design of regions satisfies the constraint for all points $e_i = (e_{ix}, e_{iy}, e_{iz})$:

- $\rho_1 = \left[e_{1x}, \infty\right) \times \left[e_{1y}, \infty\right) \times \left[e_{1z}, \infty\right)$;
- for $i \in [2, n]$, $\rho_i = \left[e_{ix}, \infty\right) \times \left[e_{iy}, \infty\right) \times \left[e_{iz}, \infty\right) \setminus \bigcup_{j=1}^{i-1} \rho_j$.

Each non-empty region $\rho_i$ is decomposed into axis-parallel disjoint cuboids by performing a vertical decomposition. If $\rho_i$ has $n_i$ vertices, then the number of cuboids in the decomposition of $\rho_i$ will be $O(n_i)$. It can be verified [1] that $\sum_{i=1}^{n} n_i = O(n)$. See Figure 2 for an illustration of these ideas.

Therefore, the max reporting problem can be transformed to a point location problem: given a query point $q \in \mathbb{R}^3$, find the cuboid (if any) containing $q$ from a set of $O(n)$ disjoint axis-parallel cuboids. Chan et al. [19] presented a structure of size $O(n)$ to answer such a query in $O(\log n)$ time.

### 5.5 Top-$k$ Halfspace Reporting for $d \geq 4$ (2nd Bullet of Theorem 5)

All the aforementioned structures are based on Theorem 2. In this subsection, we will leverage Theorem 1 to obtain a structure for top-$k$ halfspace structure with good worst-case guarantees. In the problem’s prioritized version, the input is a set $D$ of $n$ points in $\mathbb{R}^d$, where each point $e$ carries a real-valued weight $w(e)$. Given a halfspace $q$ in $\mathbb{R}^d$ and a real value $\tau$, a query reports all the points $e \in D \cap q$ with $w(e) \geq \tau$.

**RAM.** We will show that the prioritized version admits a structure of $O(n \log n)$ size that answers a query in $\tilde{O}(n^{1-1/[d/2]}) + O(t)$ time. Plugging this into Theorem 1 yields the RAM result in second bullet of Theorem 5.

For halfspace reporting, Afshani and Chan [4] gave a structure of $O(n)$ space and query time $\tilde{O}(n^{1-1/[d/2]}) + O(t)$. We leverage this structure to support prioritized queries. Create a balanced binary search tree $T$ on the weights of the points in $D$. Each weight is stored at a leaf, together with the point carrying the weight. For each node $u$ of $T$, denote by $D_u$ the set of points in the subtree of $u$. Create a halfspace reporting structure of $[4]$ on $D_u$. The total space is $O(n \log n)$.

We answer a prioritized query parameterized by halfspace $q$ and threshold $\tau$ as follows. First, pay $O(\log n)$ time to collect the canonical set of nodes $u_1, u_2, \ldots, u_m$ such that $m = O(\log n)$. $D_{u_1}, \ldots, D_{u_m}$ are disjoint, and their union equals $\{e \in D \mid w(e) \geq \tau\}$. Then, perform a halfspace reporting query on $D_{u_i}$ with halfspace $q$ for each $i \in [1, m]$. The final answer is the union of all these $m$ queries’ outputs. The query time is $\tilde{O}(n^{1-1/[d/2]}) + O(t)$.

**EM.** We will show that the prioritized version admits a structure of $O(n/B)$ size that answers a query in $O((n/B)^{1-1/[d/2]+\epsilon} + t/B)$ I/Os. Plugging this into Theorem 1 yields the EM result in the second bullet of Theorem 5.

For halfspace reporting, Agarwal et al. [6] gave a structure of $O(n/B)$ space and query time $O((n/B)^{1-1/[d/2]+\epsilon'} + t/B)$ where the constant $\epsilon' > 0$ can be arbitrarily small. We will utilize this structure to design our prioritized structure. Build a B-tree $T$ on the weights of the points in $D$; $T$ has leaf capacity $B$ and internal fanout $f = (n/B)^{1/2}$. All the weights are stored at the leaf level; furthermore, we place each point $e \in D$ in the leaf node containing $w(e)$. For each node $u$ of $T$, denote by $D_u$ the set of points in the subtree of $u$. Create a structure of $[6]$ on $D_u$ with $\epsilon' = \epsilon/2$. The overall space consumption is $O(n/B)$, noticing that $T$ has $O(1)$ levels.
To answer a prioritized query with halfspace $q$ and threshold $\tau$, we collect in the canonical set of nodes $u_1, u_2, ..., u_m$ such that $m = O(f)$, $D_{u_1}, ..., D_{u_m}$ are disjoint, and their union equals $\{ e \in D \mid w(e) \geq \tau \}$. It is rudimentary to find these nodes in $O(1 + f/B)$ I/Os. We perform a halfspace reporting query on $D_{u_i}$ with halfspace $q$ for each $i \in [1, m]$. The final answer is the union of all these $m$ queries' outputs. The query cost is

$$O \left( m \cdot (n/B)^{1-1/\lceil d/2 \rceil + \epsilon} + t/B \right) = O \left( (n/B)^{1-1/\lceil d/2 \rceil + \epsilon} + t/B \right).$$

6 2D Top-$k$ Halfspace Reporting (1st Bullet of Theorem 5)

This section serves as a proof of the first bullet of Theorem 5 on the 2D top-$k$ halfspace reporting problem. We will give a max structure of $O(n)$ size answering a max query in $O(\log n)$ time, and a prioritized structure of $O(n)$ size answering a prioritized query in $O(\log n + t)$ time. Plugging in these results into Theorem 2 completes the mission.

6.1 Max Reporting

In the max version, we have a set $D$ of $n$ points in $\mathbb{R}^2$, each associated with a real-valued weight. Given a halfplane $q$ (i.e., a 2D halfspace), a query returns the maximum weight of the points in $D \cap q$. By standard duality [24], this is equivalent to the stabbing max problem, where the input is a set $D'$ of $n$ halfplanes in $\mathbb{R}^2$, each carrying a real-valued weight. Given a point $q'$ in $\mathbb{R}^2$, a query reports the maximum weight of the halfplanes of $D'$ containing $q'$. The goal is to preprocess $D'$ in a data structure to answer queries efficiently.

Using the ideas in Section 5.4, we can transform the latter problem into a point location problem on a planar subdivision. Let $e'_1, ..., e'_n$ be the halfspaces of $D'$ in descending order of weight. For each $e'_i$, define a region $\rho_i$ in $\mathbb{R}^2$ satisfying the following constraint: a point $q \in \mathbb{R}^2$ belongs to $\rho_i$ if and only if $e'_i$ is the halfplane with the maximum weight among all the halfspaces containing $q$. The following region assignment satisfies the constraint: $\rho_1 = e'_1$, and $\rho_i = e'_i \setminus \bigcup_{j=1}^{i-1} \rho_j$ for $i \in [2, n]$.

$r_1, r_2, ..., r_n$ are disjoint polygons inducing a planar subdivision with $O(n)$ vertices. To see this, imagine generating $\rho_i$ in ascending order of $i$ as follows. If $e'_i$ falls entirely in $\bigcup_{j=1}^{i-1} \rho_j$, then $e'_i$ introduces no vertices on the subdivision. Otherwise, at least one point $p$ on the boundary of $e'_i$ must be outside $\bigcup_{j=1}^{i-1} \rho_j$. Start walking from $p$ in both directions along the boundary of $e'_i$ and stop as soon as hitting the boundary line of any of $\rho_1, ..., \rho_{i-1}$ or the (conceptual) boundary of the data space $\mathbb{R}^2$. The stopping point of each direction is a new vertex on the subdivision. Hence, $e'_i$ can add at most 2 vertices to the subdivision.

Given a query point $q'$, we answer it by finding the polygon of the subdivision containing $q'$. This point location query can be done in $O(\log n)$ time with an $O(n)$-size structure [42].

6.2 Prioritized Reporting

In the prioritized version, we have a set $D$ of $n$ points in $\mathbb{R}^2$, where each point $e \in D$ is associated with a real-valued weight $w(e)$. Given a halfplane $q$ and a real-valued threshold $\tau$, a query returns all the points in $D \cap q$ with weights at least $\tau$. We will present our data structure in three steps (Sections 6.2.1-6.2.3).

6.2.1 When All Points are on the Convex Hull Boundary

We use $CH(D)$ to represent the set of vertices of the convex hull of $D$. This subsection will prove:
Lemma 12. If all the points of $D$ are in $CH(D)$, then there is a data structure of size $O(n)$ that answers a prioritized query in $O(\log n + t)$ time, where $t$ is the number of points reported.

We assume, without loss of generality, that the query halfplane $q$ is the portion of $\mathbb{R}^2$ on and above a line with a negative slope. Label the points of $D$ in clockwise order as $e_1, e_2, ..., e_n$ with $e_1$ being the highest point (with the largest y-coordinate). Let $b$ be index such that $e_b$ is the lowest point, and $l$ such that $e_l$ is the leftmost point (with the smallest x-coordinate). $D$ can be broken into disjoint chains: $D_1 = (e_1, ..., e_b)$, $D_2 = (e_{b+1}, ..., e_l)$, and $D_3 = (e_{l+1}, ..., e_n)$. We observe:

- If $D_1 \cap q \neq \emptyset$, the points in $D_1 \cap q$ form a sequence $(e_{i^*}, e_{i^*+1}, ..., e_{j^*})$, where $1 \leq i^* \leq j^* \leq b$; see Figure 3(a). The values of $i^*$ and $j^*$ can be found using binary search in $O(\log n)$ time.

- If $D_2 \cap q \neq \emptyset$, the points in $D_2 \cap q$ form two sequences: $(e_{b+1}, e_{b+2}, ..., e_{i^*})$ and $(e_{j^*}, e_{j^*+1}, ..., e_{l})$, where $b + 1 \leq i^* < j^* \leq l$; see Figure 3(b). The values of $i^*$ and $j^*$ can be found using binary search in $O(\log n)$ time. Note that one of the two sequences may be empty, in which case either $i^*$ or $j^*$ does not exist (in the example of Figure 3(b), $i^*$ does not exist).

- If $D_3 \cap q \neq \emptyset$, the points in $D_3 \cap q$ form a sequence $(e_{i^*}, e_{i^*+1}, ..., e_n)$ where $l + 1 \leq i^* \leq n$; see Figure 3(c). The value of $i^*$ can be found using binary search in $O(\log n)$ time.

We use separate structures and algorithms for $D_1$, $D_2$, and $D_3$:

- Reporting $D_1 \cap q$: Create a 2D point set $P_1 = \{(i, w(e_i)) \mid 1 \leq i \leq b\}$. Given a halfplane $q$, the points in $D_1 \cap q$ must have their “image points” in $P_1$ covered by the 3-sided rectangle $[i^*, j^*] \times [\tau, \infty)$ (where $i^*$ and $j^*$ are as defined earlier). We build a priority search tree [33] on $P_1$, which consumes $O(|P_1|)$ space and finds all the points in a 3-sided rectangle in $O(\log n + t')$ time, where $t'$ is the number of points reported.

- Reporting $D_2 \cap q$: We explain only the retrieval of $(e_{b+1}, e_{b+2}, ..., e_{i^*})$ because a similar method works for $(e_{j^*}, e_{j^*+1}, ..., e_{l})$. Create a 2D point set $P_2 = \{(i, w(e_i)) \mid b + 1 \leq i \leq l\}$. Given a halfplane $q$, the points in $D_2 \cap q$ have their image points in $P_2$ covered by $[b, i^*] \times [\tau, \infty)$. We can build a priority search tree on $P_2$ to find those image points.

- Reporting $D_3 \cap q$: Create a 2D point set $P_3 = \{(i, w(e_i)) \mid l + 1 \leq i \leq n\}$. Given a halfplane $q$, the points in $D_3 \cap q$ have their image points in $P_3$ covered by $[i^*, n] \times [\tau, \infty)$. A priority search tree on $P_3$ serves our purposes in the same way as discussed before.

All the priority search trees consume $O(n)$ space and allow us to find $D \cap q$ in $O(\log n + t)$ time.
6.2.2 A Structure of $O(\log^2 n + t)$ Query Time

In this subsection, we will prove:

**Lemma 13.** There is a data structure of size $O(n)$ that answers a prioritized halfplane reporting query in $O(\log^2 n + t)$ query time.

Recall that a halfplane reporting query designates a halfplane $q$ and reports the points in $D \cap q$. The following result is due to Chazelle et al. [21]:

**Lemma 14.** ([21]) Suppose that all the points of $D$ are in $\text{CH}(D)$. For any halfplane $q$, we can report all the points of $D \cap q$ in $O(\log n + |D \cap q|)$ time. If we are given an arbitrary point in $D \cap q$, the query time can be reduced to $O(1 + |D \cap q|)$ time.

**Structure.** We build a binary search tree $T$ as follows. Find $D' = D \setminus \text{CH}(D)$ and sort the points of $D'$ in ascending order of weight. Let $D_l$ be the first $|D'|/2$ points in the sorted list, and $D_r = D' \setminus D_l$. We associate the root $u$ of $T$ with the set $D_u = \text{CH}(D)$, and build a structure of Lemma 12 and another structure of Lemma 14 on $D_u$. Then, recursively build the left (resp., right) subtree of $u$ on $D_l$ (resp., $D_r$), if $D_l$ (resp., $D_r$) is not empty. The smallest weight (of the points) in $D_r$ will be referred to as the split weight of $u$ (undefined if $u$ is a leaf node).

For every internal node $u$ that has a right child $v$, we store fractional cascading pointers such that, having reported the points in $D_u \cap q$, we can decide in $O(1)$ time whether $D_v \cap q$ is empty and, if not, also obtain a point in $D_v \cap q$. As discussed in [21] (see also [9]), such fractional cascading pointers can increase the space by only a constant factor. The overall space consumption is $O(n)$.

**Query.** Given a prioritized query parameterized by halfplane $q$ and weight threshold $\tau$, we first identify a root-to-leaf path $\pi$ in $T$ as follows. First, add the root to $\pi$. In general, after appending $u$ to $\pi$, we add the left child of $u$ if $\tau$ is less than the split weight of $u$, or the right child of $u$ otherwise. After $\pi$ is ready, we proceed as follows:

1. For each node $u \in \pi$, search the structure of Lemma 12 at $u$ to find all the qualifying points in $D_u$.
2. For each node $u \in \pi$, mark its right child, if the right child is not on $\pi$.
3. While there is still a marked node $v$, search the structure of Lemma 14 at $v$ to find the points in $D_v \cap q$ (note that these points must have weights at least $\tau$). If at least one point is reported, mark both children of $v$.

**Analysis.** Step 1 obviously incurs $O(\log^2 n + t)$ time. Next, we will focus on the nodes $v$ processed at Step 3. Such nodes can be divided into disjoint sets $S_1$ and $S_2$: every $v \in S_1$ has its parent on $\pi$, while $S_2$ includes all the other nodes. The total cost paid at the nodes in $S_1$ is clearly $O(\log^2 n + t)$. Every node $v \in S_2$ must have a parent $u$; furthermore, we must have reported at least one point from $D_u$. By leveraging the fractional cascading pointers at $u$ and the second sentence of Lemma 14, we spend only $O(1 + t')$ time on $v$, where $t'$ is the number of points reported from $D_u$. We can charge the $O(1)$ term on an arbitrary point reported from $D_u$. Every reported point is charged at most twice. The total query cost is, therefore, $O(\log^2 n + t)$. This completes the proof of Lemma 13.

6.2.3 The Final Structure

We will improve the query time to $O(\log n + t)$ by resorting to shallow cuttings [10,32] and a framework introduced by Afshani [1].
**Equivalent problems.** By standard duality, we can transform the prioritized reporting problem to the following. Define an upper halfplane in $\mathbb{R}^2$ as the set of points $(x, y) \in \mathbb{R}^2$ satisfying $y \geq c_1 x + c_2$ where $c_1$ and $c_2$ are constants. We will drop the word upper with the understanding that all the halfplanes in this subsection are upper ones. Let $D$ be a set of $n$ halfplanes, where each halfplane $e \in D$ is associated with a real-valued weight $w(e)$. Given a point $q \in \mathbb{R}^2$ and a real-valued threshold $\tau$, a query reports all the halfplanes $e \in D$ such that $q \in e$ and $w(e) \geq \tau$. By Lemma 13, we can preprocess $D$ into a structure of $O(n)$ space that answers a query in $O(\log^2 n + t)$ time.

With one more transformation, we will work with yet another equivalent problem in 3D space. For every halfplane $e \in D$, introduce a partially-defined bivariate function:

$$f(x, y) = \begin{cases} w(e) & \text{if } (x, y) \in e \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{11}$$

We will refer to $w(e)$ as the weight of $f$. Figure 4 shows the graph of such a function (the gray portion). Let $F$ be the set of functions obtained from the halfplanes in $D$ (i.e., $|F| = n$). Given a 3D point $q' = (x, y, z)$, we say that a function $f$ (i) is below $q'$ if $f$ is defined at $(x, y)$ and $z > f(x, y)$, or (ii) passes $q$ if $z = f(x, y)$. A query reports all the functions in $F$ that either pass or are below $q'$. Lemma 13 allows us to build a structure on $F$ that consumes $O(n)$ space and answers a query in $O(\log^2 n + t)$ time. We want to improve the query time to $O(\log n + t)$.

**Shallow cutting.** The level of a point $p \in \mathbb{R}^3$ is the number of functions $f$ below $p$. The $(\leq l)$-level of $F$ is the (infinite) set of points in $\mathbb{R}^3$ whose levels are at most $l$. We define the “boundary” of the $(\leq l)$-level of $F$ as the $l$-level of $F$; specifically, a point $p$ is on the boundary if an infinitesimally-small ball (with a positive radius) centering at $p$ covers a point whose level is strictly larger than $l$.

The lower envelope of $F$ is defined as the 0-level of $F$. We observe:

**Lemma 15.** The lower envelope of $F$ has complexity $O(n)$.

**Proof.** The projection of each function $f \in F$ onto the xy-plane is a halfplane. Given a real value $z$, denote by $F_{\leq z}$ the set of functions in $F$ whose weights are at most $z$, and by $H$ the set of halfplanes obtained by projecting the functions in $F_{\leq z}$ onto the xy-plane. As $z$ increases from $-\infty$ to $\infty$, the lower envelope of $H$ (the lower envelope of a set of upper halfplanes is a standard concept; see, e.g., [24]) may change every time $z$ reaches the weight of a function in $F$. Specifically, a change is in the form of a segment appearing or disappearing. The complexity of $F$ is proportional to the total number of such changes as $z$ goes from $-\infty$ to $\infty$. Next, we will show that there can be $O(n)$ such changes.
Consider the moment when \( z \) reaches the weight of a function \( f \in F \). If the graph of \( F \) does not intersect with the 0-level of \( F_{\leq z} \), \( f \) introduces no changes to the lower envelope of \( H \). Otherwise, \( f \) can create at most two points on the lower envelope of \( H \). Figure 5 shows an example where the lower envelope of \( H \) was the chain of segments \( p_1p_2, p_2p_3, p_3p_4, p_4p_5, p_5p_6, p_6p_7 \) before processing \( f \), and becomes a new chain \( p_1A, AB, Bp_5, p_5p_6, p_6p_7 \) after processing \( f \). In general, the number of changes is proportional to the total number of disappearing segments plus a constant \( O(1) \) (which is the number of new segments). As each segment disappears only once, we conclude that the total number of changes is \( O(n) \).

Given a simplex \( \Delta \) in \( \mathbb{R}^3 \), define its conflict list \( F_{\Delta} \) as the set of functions \( f \in F \) whose graphs intersect with \( \Delta \). Given an integer \( r \in [1, n] \), a \((1/r)\)-shallow cutting of \( F \) is a collection \( \Xi \) of non-overlapping simplices (except at the boundaries) satisfying:

- every simplex in \( \Xi \) has a conflict list of size \( O(r) \);
- the union of all the simplices in \( \Xi \) covers the \((\leq r)\)-level of \( F \).

A prism is a special simplex whose bottom vertex has z-coordinate \(-\infty\). There is an established theory \([10,18,22,23,32]\) on shallow-cuttings for general bi-variate functions. In our context, the theory yields:

**Lemma 16.** \( F \) has a shallow-cutting of size \( O(n/r) \) where every simplex is a prism.

**Proof.** Given Lemma 15, Theorem 3.1\(^3\) of [10] proves the existence of a \((1/r)\)-shallow cutting \( \Xi \) of size \( O(n/r) \). By applying a technique in Lemma 2.1 of [18], we can obtain from \( \Xi \) another \((1/r)\)-shallow cutting satisfying our requirements. \( \square \)

**Structure.** We are now ready to present our final structure for prioritized reporting on \( F \) (which, as shown earlier, is equivalent to the original prioritized reporting problem). First, build a structure \( T \) of Lemma 13 on \( F \). Then, apply Lemma 16 to obtain a \((1/\log^2 n)\)-shallow cutting \( \Xi \) of \( F \). For each prism \( \Delta \in \Xi \), build a structure \( T_\Delta \) of Lemma 13 on \( F_\Delta \) (the conflict list of \( \Delta \)). The projection of the prisms in \( \Xi \) onto the xy-plane is a subdivision of \( \mathbb{R}^2 \). Build a point location structure [42] on the cells of that subdivision.

---

\(^3\)The theorem was proved for totally-defined functions. However, as remarked in [10], it also holds on partially-defined functions whose boundaries can be described as constant-degree polynomials of \( x \) and \( y \). This is true in our context, where the boundary of each function in \( F \) is a line. The idea is to convert every function \( f \in F \) into a totally-defined function by computing its Minkowski sum with a cone \( z = c\sqrt{x^2 + y^2} \) for an arbitrarily large constant \( c \).
T uses $O(n)$ space. As $|\Xi| = O(n/\log^2 n)$, the point location structure occupies $O(n/\log^2 n)$ space. As each conflict list $F_\Delta$ has size $O(\log^2 n)$, the total space of all the $T_\Delta$’s is $|\Xi| \cdot O(\log^2 n) = O(n)$. The overall space of our structure is therefore $O(n)$.

**Query.** Recall that, given a point $q \in \mathbb{R}^3$, a query reports all the functions in $F$ that either pass through $q$ or are below $q$. By searching the point location structure, we can determine in $O(\log n)$ time whether a prism $\Delta \in \Xi$ covers $q$. If so, all the qualifying functions must be in the conflict list $F_\Delta$; we can report all of them using $T_\Delta$ in $O((\log \log n)^2 + t)$ time, using the fact $|F_\Delta| = O(\log^2 n)$. Otherwise, the level of $q$ must be at least $\log^2 n$, implying that the number $t$ of qualifying functions is at least $\log^2 n$. From $T$, we can find all of them in $O(\log^2 n + t) = O(t)$ time. In any case, the query time is $O(\log n + t)$.

### 7 Concluding Remarks

We conclude the paper with several directions for future research:

- Theorem 1 shows an $O(\log_B n)$ gap in query cost between top-$k$ search and prioritized reporting. Closing the gap would make an exciting result because it would imply the two problems’ asymptotic equivalence.
- The space and query bounds in Theorem 2 hold in expectation. Can we prove high-probability or even worst-case bounds?
- Currently, the core-set construction in Theorem 1 takes $n^{O(1)}$ expected time. Reducing the construction time significantly remains as a major challenge.
- Some top-$k$ problems still remain difficult even with our reductions. For example, is there a RAM structure of $O(n)$ space and $O(\log n + k)$ query time for 3D top-$k$ halfspace reporting?
- Our reductions appear to be simple enough for practical implementation. How do they compete with the top-$k$ solutions in the existing systems [28]?

### Acknowledgements

The research of Rahul Saladi was supported by a start-up grant from the Indian Institute of Science. The research of Yufei Tao was supported by GRF projects 14203421 and 14207820 from HKRGC.

### Appendix: Chernoff Bounds

Let $X_1, \ldots, X_n$ be independent Bernoulli variables such that $\Pr[X_i = 1] = p_i$ for all $i \in [1, n]$. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X] = \sum_{i=1}^n p_i$. It holds for any $\alpha \in (0, 1)$ that

$$\Pr[X \leq (1 - \alpha)\mu] \leq e^{-\alpha^2 \mu/3}. \quad (12)$$

For any $\alpha \geq 2$, it holds that

$$\Pr[X \geq \alpha\mu] \leq e^{-\alpha\mu/6}. \quad (13)$$

The above inequalities can be found in many papers and textbooks, e.g., [25, 34].
References


