Output-sensitive Skyline Algorithms in External Memory*

Xiaocheng Hu† Cheng Sheng† Yufei Tao†‡ Yi Yang§ Shuigeng Zhou§

Abstract

This paper presents new results in external memory for finding the skyline (a.k.a. maxima) of N points in d-dimensional space. The state of the art uses $O((N/B)\log^{d-2}_{M/B}(N/B))$ I/Os for fixed $d \geq 3$, and $O((N/B)\log_{M/B}(N/B))$ I/Os for $d = 2$, where M and B are the sizes (in words) of memory and a disk block, respectively. We give algorithms whose running time depends on the number K of points in the skyline. Specifically, we achieve $O((N/B)\log^{d-2}_{M/B}(K/B))$ expected cost for fixed $d \geq 3$, and $O((N/B)\log_{M/B}(K/B))$ worst-case cost for $d = 2$.

As a side product, we solve two problems both of independent interest. The first one, the *M-skyline problem*, aims at reporting M *arbitrary* skyline points, or the entire skyline if its size is at most M. We settle this problem in $O(N/B)$ expected time in any fixed dimensionality d. The second one, the *M-pivot problem*, is more fundamental: given a set S of N elements drawn from an ordered domain, it outputs M *evenly scattered* elements (called pivots) from S, namely, S has asymptotically the same number of elements between each pair of consecutive pivots. We give a deterministic algorithm for solving the problem in $O(N/B)$ I/Os.

1 Introduction

Let p, p' be two different points in $\mathbb{R}^d$, where $\mathbb{R}$ represents the real domain. We say that p *dominates* p' if $p[i] \leq p'[i]$ for every $i \in [1, d]$, where $p[i]$ is the i-th coordinate of p. In the *skyline problem*, we are given a set P of N points in $\mathbb{R}^d$, and want to compute the *skyline* of P, or formally:

$$SKY(P) = \{p \in P \mid \nexists p' \in P \text{ s.t. } p' \text{ dominates } p\}$$

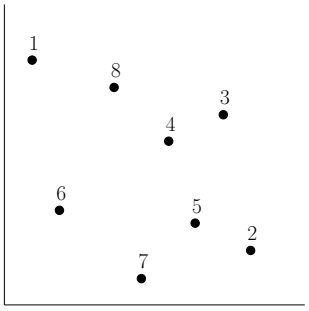

Figure 1: Skyline = {(point) 1, 6, 7}

See Figure 1 for an example. This is also known as the *maxima problem* in computational geometry or the *pareto set problem* in operations research. Let $K = |SKY(P)|$.

We study the problem in the *external memory* (EM) computation model [2]. A machine has M words of memory, and a disk of unbounded size that is formatted into disjoint *blocks*, each of which has B words. The value of M is at least $2B$. An I/O exchanges a block between the disk and memory. The running time of an algorithm is measured as the number of I/Os performed. A *linear complexity* refers to $O(N/B)$.

Mathematical conventions. Two positive integers are *roughly the same* if they are within a constant factor that is independent on N, K and B. Every logarithm $\log_b x$ should be interpreted as $\max\{1, \log_b x\}$. All logarithms by default have base $b = 2$. For $b > 1$, define $\log^{(1)}_b x = \log_b x$ and $\log^{(i)}_b x = \log_b \log^{(i-1)}_b x$ for any integer $i \geq 2$. Furthermore, $\log^*_b x$ is the smallest i such that $\log^{(i)}_b x < 16$.

Previous results. The skyline problem has been extensively studied due to its relevance to a large variety of applications (see [14] for a survey). Most solutions are heuristic, that is, they perform well on certain real datasets, but on a bad dataset, can be as slow as a naive *blocked nested loop* (*BNL*) that simply checks each pair of points in P using $O(N^2/(MB))$ I/Os. In the sequel, we focus on the existing algorithms beating the time complexity of *BNL*.

In 2d space, the skyline can be easily computed in

*This work was supported in part by (i) GRF grants 4164/12, 4165/11, and 4166/10 from the HKRGC, (ii) the WCU (World Class University) program under the National Research Foundation of Korea, which is funded by the Ministry of Education, Science and Technology of Korea (Project No: R31-30007), and (iii) the Research Innovation Program of Shanghai Municipal Education Committee under grant No. 13ZZ003.

†Department of Computer Science and Engineering, Chinese University of Hong Kong

‡Division of Web Science and Technology, Korea Advanced Institute of Science and Technology

§School of Computer Science, Fudan University

$O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os by sorting the dataset followed by a single scan [12]. Goodrich, Tsay, Vengroff and Vitter [9] were the first to observe that, in 3d space, the problem can also be settled in $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os, using a technique called *distribution sweep*. Sheng and Tao [14] extended the result to arbitrary fixed dimensionality $d \geq 3$, where their algorithm terminates in $O(\frac{N}{B}\log^{d-2}_{M/B}\frac{N}{B})$ I/Os.

Under many data distributions, the number K of skyline points is significantly smaller than the cardinality N of the dataset (see [15] and the references therein). An intriguing question is whether we can improve the computation time in such a scenario. Ideally, an algorithm should be *output-sensitive*: its cost ought to increase slowly along with K, whereas even at $K = \Omega(N)$, it is still as efficient as the best algorithm whose running time is *insensitive* to K (i.e., the solutions in [9, 12, 14]).

Motivated by this, Sarma, Lall, Nanongkai and Xu [13] developed a randomized algorithm, which we call *3-pass*, to solve the skyline problem of any fixed d in $O(\frac{N}{B} + \frac{NK}{MB})$ expected I/Os. When $K = O(M)$, their algorithm performs only linear I/Os in expectation. However, as K increases, the efficiency deteriorates rapidly such that when $K = \Omega(N)$, the algorithm loses its advantage even over *BNL*. In 2d space, the authors of [13] presented a deterministic algorithm, which runs in $O(\frac{N}{B} + \frac{NK}{MB}\log N)$ I/Os, provided that the memory has $\Omega(B \log N)$ words[1].

In internal memory, a classic result by Kung, Luccio and Preparata [12] computes the skyline in $O(N \log^{d-2} N)$ time for fixed $d \geq 3$ and in $O(N \log N)$ time for $d = 2$. Bentley [4] described how to achieve the same performance via different algorithms. Departing from the class of comparison-based algorithms and leveraging features of the word-RAM model, Gabow, Bentley and Tarjan [8] reduced the computation time to $O(N \log^{d-3} N \cdot \log\log N)$ for fixed $d \geq 4$. Chan, Larsen and Patrascu [7] recently further improved the cost to $O(N \log^{d-3} N)$. The only output-sensitive algorithm is due to Kirkpatrick and Seidel [11]. Their algorithm runs in $O(N \log^{d-2} K)$ time for fixed $d \geq 3$, and $O(N \log K)$ time for $d = 2$.

Our results. This paper develops the first output-sensitive skyline algorithms in external memory that is efficient for the whole range of K. For any fixed $d \geq 3$, we can find the skyline in $O((N/B)\log^{d-2}_{M/B}(K/B))$ expected I/Os, whereas for $d = 2$, the running time is $O((N/B)\log_{M/B}(K/B))$ I/Os in the worst case (Theorem 4.2). As an interesting corollary, when $K = \text{polylog}(M/B)$ – a situation likely to occur in practice – the skyline problem can be settled in linear I/Os. No previous algorithm is known to have this property. For large K, the cost of our algorithms gracefully aligns with the output-*insensitive* solutions in [9, 14]. At $d = 2$, our algorithm is optimal in the comparison class for all values of K (see the remark in Section 4.4).

Our techniques differ significantly from those of [11]. In particular, we have to settle two separate problems, both of which are specialized to external memory, and are of independent interest. In the first, called the *M-skyline problem*, the goal is to report M arbitrary points in $SKY(P)$ if $|SKY(P)| \geq M$, or the entire $SKY(P)$, otherwise. We prove that this problem can be solved in $O(N/B)$ expected I/Os for *any* fixed dimensionality d (Theorem 2.1). The running time's independence from d is somewhat surprising, given that d plays a major role in the cost of computing the full skyline. In 2d space, the running time can be made worst case (Theorem 4.1).

The second problem, called the *M-pivot problem*, is more fundamental. The input is a set S of N elements drawn from an ordered domain. The goal is to find $v_1, ..., v_{M-1}$ from S as *pivots*. Assuming $v_1 < ... < v_{M-1}$, the pivots should have the property that S has $\Theta(N/M)$ elements between v_{i-1} and v_i for each $i \in [1, M]$, defining v_0 (v_M) to be the minimum (maximum) element in the ordered domain.

The problem can be easily solved by sorting, which necessitates super-linear I/Os. If randomization is allowed, one can resort to random sampling to obtain a linear time algorithm easily [1]. What is non-trivial is to do so deterministically. Previously, the best linear-time algorithm [2] produces only $O(\sqrt{M/B})$ evenly scattered pivots[2] (i.e., S has roughly the same number of elements between two consecutive pivots). In this paper, we settle the M-pivot problem in linear I/Os (Theorem 3.1).

The M-pivot problem should not be confused with the *L-partition problem*, where the goal is to partition S into disjoint subsets $S_1, ..., S_L$ of roughly the same size, such that all elements in S_i are smaller than those in S_j for any $i < j$. Note that it is *impossible* to solve even the M^ϵ-partition problem (for any constant $\epsilon \in (0, 1]$) in linear time. One can prove this easily by setting $M = 2B$, and arguing that the ability of doing so would imply sorting N elements in $O((N/B)\log_B(N/B))$ I/Os, violating the lower bound $\Omega((N/B)\log_2(N/B))$ [2]. Our result, therefore, separates the M-pivot problem from the harder M^ϵ-partition problem.

[1] The memory requirement arises from the deployment of a *GK-sketch* [10].

[2] By repeating the algorithm c times where $c \geq 1$ is a constant integer, $O((M/B)^{c/2})$ evenly scattered pivots can be obtained.

2 The M-skyline Problem

This section concentrates on the M-skyline problem as defined in Section 1. We observe that the *3-pass* algorithm of [13] can be utilized to solve this problem in linear time. Our contribution is the analysis: we prove a new, crucial, property of this algorithm that has escaped the previous studies.

2.1 3-pass Let m be the number of points that can be stored in memory, namely, $m = M/d = \Theta(M)$. This algorithm scans the input set P three times, each of which performs $O(N/B)$ I/Os. The first scan samples (with replacement) m points uniformly and independently from P. Let Σ be the sample set. Note that $|\Sigma|$ may be less than m because a point can be sampled multiple times.

Σ is kept in memory during the second scan. Certain points may be inserted to Σ during the scan, but each insertion is always accompanied by removing at least one point in Σ. Hence, $|\Sigma|$ never increases, but may decrease significantly. Specifically, this scan inspects every point $p \in P$. If p dominates at least a point in Σ, p is added to Σ (if not already there) while those points dominated by p are expunged from Σ. Note that, at this moment, p may possibly be dominated by a sample $p' \in \Sigma$ (i.e., p and p' co-exist in Σ). However, such a situation can happen only if the (second) scan has not reached p' yet, and p' was acquired from the first scan. Later, p will be removed from Σ when p' is encountered, if not earlier. When the second scan finishes, all the points of Σ must be in the skyline.

To illustrate, assume that the points in Figure 1 are scanned in ascending order of their ids, and that the first scan returns $\Sigma = \{(\text{point})\ 3, 7\}$ (i.e., $m = 2$). In the second scan, point 1 has no effect on Σ because it does not dominate any point in Σ. The first change of Σ takes place when inspecting point 4 – since it dominates point 3, we insert the former and delete the latter in Σ, which becomes $\{4, 7\}$. Notice that even though point 7 dominates point 4, they are co-existing in Σ. Later, point 4 is replaced by point 6, which remains in Σ together with point 7 till the end of this scan.

Again retaining Σ in memory, the last scan of *3-pass* eliminates the points of P dominated by at least one point in Σ. All the (skyline) points in Σ are also removed from P. In other words, at the end of *3-pass*, P contains points that have not been confirmed as a skyline or non-skyline point. Continuing our earlier example, with $\Sigma = \{6, 7\}$ from the second scan, the third scan removes all the points from P except point 1.

To solve the M-skyline problem, we keep running *3-pass* until either at least M skyline points have been reported, or P has become empty.

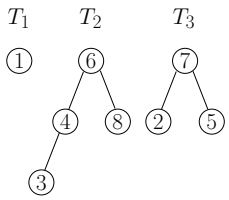

Figure 2: T_1, T_2, T_3 for the example in Figure 1

2.2 Analysis This subsection serves as the proof of:

THEOREM 2.1. *The M-skyline problem on N points in any fixed dimensionality can be solved in $O(N/B)$ expected I/Os.*

We will show that the algorithm of the previous section has the desired performance guarantee. Towards this purpose, next we first review some facts observed in [13] about *3-pass*, and then prove a new imperative property.

Facts from [13]. Regard the input set P as a sequence, arranging the points in the same order as they are encountered by a scan. For each point $p \in P$, let *first*(p) be the earliest point $p' \in P$ dominating p. If p is in the skyline, *first*$(p) = \emptyset$. Note that, for a non-skyline point p, *first*(p) can be before or after p in the sequence. We define K trees $T_1, ..., T_K$, where K is the number of skyline points, by making *first*(p) the parent of p for every $p \in P$. If p is a skyline point, it has no parent, and therefore, is the root of a tree. Figure 2 shows the trees for the dataset in Figure 1, assuming that points are scanned in ascending order of their ids. The parent of point 3, for instance, is point 4 because points 1 and 2 do not dominate point 3. *3-pass* has the following property:

LEMMA 2.1. ([13]) *The following are true at the end of 3-pass. If a point in T_i ($1 \leq i \leq K$) is sampled in the first scan, the entire T_i is eliminated, and the point at its root is reported. Otherwise, the entire T_i remains in P and its root is not reported.*

A new property. Set $c = 0.7$ in the rest of the section. Denote by N_i ($1 \leq i \leq K$) the number of points in T_i. We say that T_i is *big* if $N_i \geq cN/m$; otherwise, T_i is *small*. We will prove:

LEMMA 2.2. *At the end of* 3-pass*, P has at most $(0.5 + e^{-c})N < 0.997N$ points in expectation if*

$$\sum_{\forall\ small\ T_i} N_i \leq N/2. \tag{2.1}$$

Otherwise, 3-pass returns more than $m/4$ points in expectation.

In other words, *3-pass* either prunes a constant fraction of the dataset, or reports $\Omega(M)$ skyline points. This will be the key to establishing Theorem 2.1 later.

Proof. We say that T_i $(1 \le i \le K)$ is *sampled* if the first scan of *3-pass* samples at least one point from T_i. Otherwise, T_i is not sampled. Define a random variable x_i to be 1 if T_i is sampled, or 0 otherwise. Clearly, $\mathbf{Pr}[x_i = 0] = (1 - N_i/N)^m$, and hence, $\mathbf{E}[x_i] = 1 - (1 - N_i/N)^m$.

First consider the scenario where (2.1) holds. By Lemma 2.1, either every or no point of T_i remains in P when *3-pass* finishes. The former situation happens if and only if $x_i = 0$. Hence, the number of points in P after *3-pass* equals $\sum_{i=0}^{K} N_i(1 - x_i)$ whose expectation is:

$$\begin{aligned}
\sum_{i=0}^{K} N_i(1 - \mathbf{E}[x_i]) &= \sum_{i=1}^{K} N_i \cdot (1 - N_i/N)^m \\
&\le \sum_{\forall \text{ small } T_i} N_i + \sum_{\forall \text{ big } T_i} N_i(1 - N_i/N)^m \\
\text{(by (2.1))} &\le \frac{N}{2} + \sum_{\forall \text{ big } T_i} N_i \left(1 - \frac{N_i}{N}\right)^m \\
(1 - x \le e^{-x}) &\le \frac{N}{2} + \sum_{\forall \text{ big } T_i} \frac{N_i}{(e^{N_i/N})^m} \\
\text{(by the def. of big } T_i) &\le \frac{N}{2} + \sum_{\forall \text{ big } T_i} \frac{N_i}{e^{\frac{cN}{m} \cdot \frac{m}{N}}} \\
&\le \frac{N}{2} + \frac{N}{e^c}
\end{aligned}$$

which proves the first part of Lemma 2.2.

Now consider that (2.1) does not hold. By Lemma 2.1, the root of T_i is returned by *3-pass* if and only if $x_i = 1$. Hence, the number of skyline points returned is $\sum_{i=1}^{K} x_i$ whose expectation is:

$$\begin{aligned}
\sum_{i=1}^{K} \mathbf{E}[x_i] &= \sum_{i=1}^{K} 1 - \left(1 - \frac{N_i}{N}\right)^m \\
(2.2) \qquad &\ge \sum_{\forall \text{ small } T_i} 1 - \left(1 - \frac{N_i}{N}\right)^m.
\end{aligned}$$

In the appendix, we prove the following fact: *for any real x and integer y satisfying $0 < x < 1$, $y \ge 1$, and $xy \le 3/4$, it holds that $(1 - x)^y < 1 - xy/2$.* Now we apply the fact to every small T_i. Treat $x = N_i/N$ and $y = m$. Since T_i is small, $N_i < cN/m$, which means $(N_i/N)m < c$ and hence, $xy < c = 0.7 < 3/4$. Therefore, $(1 - N_i/N)^m < 1 - N_i m/(2N)$. We now have:

$$\begin{aligned}
(2.2) &> \sum_{\forall \text{ small } T_i} 1 - \left(1 - \frac{N_i m}{2N}\right) \\
&= \frac{m}{2N} \sum_{\forall \text{ small } T_i} N_i \\
\text{(as (2.1) does not hold)} &> \frac{m}{2N} \frac{N}{2} = \frac{m}{4}.
\end{aligned}$$

as claimed. □

We remark that Lemma 2.2 applies to not only the first run of *3-pass* in our M-skyline algorithm, but also the subsequent runs. The only change is that N should be replaced by the size of P at the beginning of a run.

Proof of Theorem 2.1. Equipped with Lemma 2.2, we now present the proof of Theorem 2.1. Set $\rho = 0.997$. We first give a corollary of Lemma 2.2:

COROLLARY 2.1. *The following are true at the end of 3-pass. If* (2.1) *holds, with probability at least $1 - \rho$, P has less than $0.9996N$ points left. Otherwise, with probability at least $1 - \rho$, 3-pass returns more than $0.247m$ skyline points.*

All the omitted proofs (such as the above one) can be found in the appendix. Let $c_1 = 0.9996$ and $c_2 = 0.247$. We are ready to bound the expected cost of the M-skyline algorithm. Recall that the algorithm runs *3-pass* multiple times. We say that a run is (i) *downscaling*, if P has at most $c_1 x$ points left after the run, where x is the size of P when the run started; (ii) *productive*, if it finds at least $c_2 m$ skyline points; (iii) *futile*, if it is neither downscaling nor productive.

There can be at most $M/(c_2 m) = O(1)$ productive runs because we aim at reporting M skyline points only. As for downscaling runs, notice that when the i-th $(i \ge 1)$ such run is launched, P has at most Nc_1^{i-1} points left. Therefore, the total cost of all these runs is at most $\sum_{i=1}^{\infty} \frac{Nc_1^{i-1}}{B} = O(N/B)$.

It remains to bound the cost of futile runs. For this purpose, define a *streak* as a sequence of consecutive futile runs, preceded (resp. followed) by either a downscaling/productive run, or the beginning (resp. end) of the algorithm. Let n_{futile} be the number of runs in a streak. Corollary 2.1 shows that a run is futile with probability at most ρ. Hence, $\mathbf{Pr}[n_{futile} = i] \le \rho^i$ such that $\mathbf{E}[n_{futile}] \le \sum_{i=1}^{\infty} i \cdot \rho^i = \frac{\rho}{(1-\rho)^2} = O(1)$. Hence, if P has x points at the beginning of a streak, the total cost of the streak is $O(x/B)$ in expectation because each run of the streak performs at most $O(x/B)$ I/Os.

It follows that, if a streak is preceded by a downscaling or productive run, in expectation, the streak incurs asymptotically at most the same cost as that run. Otherwise (i.e., the algorithm goes into a streak right from the beginning), the streak entails $O(N/B)$ I/Os in expectation. Therefore, the total overhead of all the streaks cannot be more expensive than that of downscaling and productive runs by more than a constant factor. We thus conclude the proof of Theorem 2.1.

3 The M-pivot problem

We now proceed to explain how to settle the M-pivot problem (defined in Section 1) deterministically in $O(N/B)$ I/Os. Set $n = N/M$. For simplicity, we will assume that all elements are drawn from the real domain, but our discussion extends to any ordered domain straightforwardly.

3.1 Even sampling We first describe an algorithm *even-sample*, which extends a sub-routine in *distribution sort* [2]. The input of *even-sample* is a set S of N values in $\mathbb{R}$, and an integer parameter s called *reduction ratio* satisfying $2 \leq s \leq \sqrt{n}$. It outputs a sample set Σ of S such that $N/s - sM < |\Sigma| \leq N/s + sM$.

Even-sample starts by dividing S arbitrarily into $g = \lceil N/(s^2M) \rceil$ *groups*, each of which has size s^2M except possibly the last group, whose size can be smaller. Denote these groups as $G_1, ..., G_g$ respectively. For each G_i $(1 \leq i \leq g)$, sort its values, and divide the sorted list into sM *sub-groups* of size s. The only exception is the last sub-group of G_g, which may have less than s values. Each of $G_1, ..., G_{g-1}$ must contain exactly sM sub-groups, whereas the number of sub-groups in G_g can be anywhere from 1 to sM. The total number of sub-groups from all groups is therefore between $s(g-1)M + 1$ and sgM. Finally, Σ takes the maximum value in each sub-group as a sample.

Partitioning S into (arbitrary) groups requires only $O(N/B)$ I/Os. As each group has size at most s^2M, sorting the group takes $O(\frac{s^2M}{B} \log_{M/B} \frac{s^2M}{B}) = O(\frac{s^2M}{B} \log_{M/B} s)$ I/Os. Hence, the sorting of all g groups finishes in $O(\frac{gs^2M}{B} \log_{M/B} s) = O(\frac{N}{B} \log_{M/B} s)$ I/Os. Hence, *even-sample* runs in $O(\frac{N}{B} \log_{M/B} s)$ I/Os overall.

We have the following facts about Σ:

LEMMA 3.1. *$N/s - sM < |\Sigma| \leq N/s + sM$.*

LEMMA 3.2. *Let $(v_1, v_2]$ be any interval in $\mathbb{R}$. If Σ has x values in $(v_1, v_2]$, S has more than $sx - 2n/s$, but less than $sx + 2n/s$ values in $(v_1, v_2]$.*

3.2 Solving the M-pivot problem We consider $n > 65536$ so that $\log^4_{M/B} n \leq \log^4_2 n < n$. For $n \leq 65536$, the M-pivot problem can be solved in linear time by sorting.

Algorithm. Define $h = \log^*_{M/B} n$. In other words, $\log^{(h-1)}_{M/B} n \geq 16$ (recall our definition of $\log^*$ in Section 1). Our algorithm has two steps. The first step runs *even-sample* $h-1$ times with rapidly increasing reduction ratios. The input to each run is the output of the previous run, except the first run whose input is the dataset S itself. Let S_i be the input of the i-th $(1 \leq i \leq h-1)$ run, and Σ_i its output. Hence, $S_1 = S$, and $S_i = \Sigma_{i-1}$ for $i \geq 2$. Denote by s_i the parameter s (i.e., reduction ratio) of the i-th run. We set $s_i = \lfloor \log^{(h-i)}_{M/B} n \rfloor$. It is easy to verify that $s_i \geq \frac{1}{2}(M/B)^{s_{i-1}}$ for $i \geq 2$.

The second step sorts Σ_{h-1} in ascending order, and divides the sorted list into M segments as evenly as possible. Namely, each segment has size either $\lfloor |\Sigma_{h-1}|/M \rfloor$ or $\lfloor |\Sigma_{h-1}|/M \rfloor + 1$. The greatest value in each of the first $M-1$ segments is taken as a final pivot, which is returned.

Analysis. For each $i \in [1, h-1]$, define $N_i = |S_i|$ and $n_i = N_i/M$. Remember $N_1 = N$ and $N_i = |\Sigma_{i-1}|$ for $i \geq 2$. The following fact is elementary:

LEMMA 3.3. *For $n > 65536$, $\prod_{i=1}^{h-1} s_i < \sqrt{n} < \frac{n}{4 \log_{M/B} n}$.*

We now establish a crucial property of our algorithm:

LEMMA 3.4. *For each $i \in [1, h-1]$, $\frac{N}{2s_i s_{i-1} ... s_1} < |\Sigma_i| < \frac{3N}{2s_i s_{i-1} ... s_1}$.*

Proof. Regarding the i-th run of *even-sample*, setting the N of Lemma 3.1 to N_i shows:

$$
\begin{aligned}
|\Sigma_i| &> \frac{N_i}{s_i} - s_i M = \frac{|\Sigma_{i-1}|}{s_i} - s_i M \\
\text{(applying Lemma 3.1)} \quad &> \frac{\frac{N_{i-1}}{s_{i-1}} - s_{i-1} M}{s_i} - s_i M \\
\text{(as } s_i > 1\text{)} \quad &> \frac{N_{i-1}}{s_i s_{i-1}} - M \sum_{j=i-1}^{i} s_j \\
&\cdots \\
(3.3) \qquad &> \frac{N_1}{s_i ... s_1} - M \sum_{j=1}^{i} s_j.
\end{aligned}
$$

The above used only the lower bound of $|\Sigma|$ in Lemma 3.1. From its upper bound (i.e., $N/s + sM$),

after similar derivation, we get:

$$|\Sigma_i| \leq \frac{N_1}{s_i...s_1} + M\sum_{j=1}^{i} s_j. \tag{3.4}$$

The rest of the proof will show: $M\sum_{j=1}^{i} s_j < \frac{N_1}{2s_i...s_1}$, which will establish the lemma together with (3.3) and (3.4) (recall that $N_1 = N$). In fact:

$$\begin{aligned} M\sum_{j=1}^{i} s_j &\leq M\sum_{j=1}^{i} \log_{M/B}^{(h-j)} n \\ &< 2M\log_{M/B} n \\ \text{(applying Lemma 3.3)} &< \frac{nM}{2s_{h-1}...s_1} \\ &\leq \frac{nM}{2s_i...s_1} = \frac{N}{2s_i...s_1} \end{aligned}$$

thus completing the proof. □

COROLLARY 3.1. *For each $i \in [2, h-1]$, $\frac{n}{2s_{i-1}s_{i-2}...s_1} < n_i < \frac{3n}{2s_{i-1}s_{i-2}...s_1}$.*

Proof. It follows from the previous lemma and the fact that $n_i = N_i/M = |\Sigma_{i-1}|/M$. □

COROLLARY 3.2. *For each $i \in [2, h-1]$, $s_i < \sqrt{n_i}$.*

The above corollary shows that s_i falls in the legal range $[2, \sqrt{n_i}]$ as demanded by *even-sample* (recall that $s_1 \geq 16$). Next, we prove that the pivots computed by our algorithm satisfy the requirement of the M-pivot problem. Let $v_1, ..., v_{M-1}$ be the pivots in ascending order. Define dummy values $v_0 = -\infty$ and $v_M = \infty$.

LEMMA 3.5. *For each $j \in [1, M]$, the original dataset S has $\Theta(n)$ values in $(v_{j-1}, v_j]$.*

Proof. Let x_i $(1 \leq i \leq h-1)$ be the number of values in $\Sigma_i \cap (v_{j-1}, v_j]$, and x_0 the number of values in $S \cap (v_{j-1}, v_j]$. We will prove $x_0 = \Omega(n)$ only, because an analogous argument establishes $x_0 = O(n)$.

Recall that the second step of our algorithm divides Σ_{h-1} into M segments evenly. Exactly one segment falls in $(v_{j-1}, v_j]$. A segment has length $\lfloor |\Sigma_{h-1}|/M \rfloor$ or $\lfloor |\Sigma_{h-1}|/M \rfloor + 1$. Hence, x_{h-1} is at least $|\Sigma_{h-1}|/(2M)$. As for $|\Sigma_{h-1}|$, we have from Lemma 3.4: $|\Sigma_{h-1}| > \frac{N}{2s_{h-1}...s_1}$ which gives:

$$x_{h-1} > \frac{n}{4s_{h-1}...s_1}. \tag{3.5}$$

Now we turn attention to x_{h-2}. Recall that $\Sigma_{h-2} = S_{h-1}$. Hence, x_{h-2} is also the number of values in $S_{h-1} \cap (v_{j-1}, v_j]$. By Lemma 3.2, this number is greater than $s_{h-1}x_{h-1} - 2n_{h-1}/s_{h-1}$. Lower bounding x_{h-1} with (3.5) and upper bounding n_{h-1} with Corollary 3.1, we obtain:

$$\begin{aligned} x_{h-2} &> s_{h-1}\frac{n}{4s_{h-1}...s_1} - \frac{2}{s_{h-1}}\frac{3n}{2s_{h-2}...s_1} \\ &= \frac{n}{4s_{h-2}...s_1} - \frac{3n}{s_{h-1}...s_1} \\ &= \frac{n}{s_{h-2}...s_1}\left(\frac{1}{4} - \frac{3}{s_{h-1}}\right). \end{aligned} \tag{3.6}$$

With reasoning similar to what we applied to x_{h-2}, Lemma 3.2 shows that $x_{h-3} > s_{h-2}x_{h-2} - 2n_{h-2}/s_{h-2}$. Lower bounding x_{h-2} with (3.6) and upper bounding n_{h-2} with Corollary 3.1, we obtain:

$$\begin{aligned} x_{h-3} &> \frac{s_{h-2}\cdot n}{s_{h-2}...s_1}\left(\frac{1}{4} - \frac{3}{s_{h-1}}\right) - \frac{2}{s_{h-2}}\frac{3n}{2s_{h-3}..s_1} \\ &= \frac{n}{s_{h-3}...s_1}\left(\frac{1}{4} - \frac{3}{s_{h-1}}\right) - \frac{3n}{s_{h-2}...s_1} \\ &= \frac{n}{s_{h-3}...s_1}\left(\frac{1}{4} - \frac{3}{s_{h-2}} - \frac{3}{s_{h-1}}\right). \end{aligned}$$

Continuing in the same manner, eventually we obtain:

$$x_0 > n\left(\frac{1}{4} - \frac{3}{s_1} - \frac{3}{s_2} - ... - \frac{3}{s_{h-1}}\right). \tag{3.7}$$

We know: $s_1 \geq 16$, $s_2 \geq \frac{1}{2}(M/B)^{16} \geq 2^{15}$, $s_3 \geq \frac{1}{2}(M/B)^{2^{15}} \geq 2^{2^{15}-1}$, ... Therefore, from (3.7), we have $x_0 > n\left(\frac{1}{4} - \frac{3}{16} - 2\cdot\frac{3}{2^{15}}\right) = \Omega(n)$. □

We complete our analysis with:

THEOREM 3.1. *Our algorithm solves the M-pivot problem in $O(N/B)$ I/Os.*

Proof. The second step of our algorithm sorts Σ_{h-1}. With $|\Sigma_{h-1}|$ upper bounded in Lemma 3.4, we know that the sorting needs no more than $O(\frac{N}{Bs_{h-1}...s_1}\log_{M/B}\frac{N}{B})$ I/Os, which is $O(N/B)$ because $\log_{M/B}\frac{N}{B} = O(\log_{M/B} n) = O(s_{h-1})$.

The first step runs *even-sample* $h-1$ times. The first run takes $O(\frac{N}{B}\log_{M/B} s_1) = O(N/B)$ I/Os, because $\log_{M/B} s_1 \leq \log_{M/B}\log_{M/B}^{(h-1)} n = \log_{M/B}^{(h)} n = O(1)$. It remains to bound the cost of the 2nd, ..., $(h-1)$-st runs. Combining Corollary 3.1 and the discussion in Section 3.1, the i-th $(2 \leq i \leq h-1)$ run entails cost $O(\frac{N}{Bs_{i-1}...s_1}\log_{M/B} s_i)$, which is $O(\frac{N}{Bs_{i-2}...s_1})$ because $\log_{M/B} s_i \leq \log_{M/B}\log_{M/B}^{(h-i)} n = \log_{M/B}^{(h-(i-1))} n = O(s_{i-1})$. Therefore, the total cost of the 2nd, ..., $(h-1)$-st runs is $O(\sum_{i=2}^{h-1}\frac{N}{Bs_{i-2}...s_1})$, which is $O(N/B)$ because the terms decrease at least geometrically as i grows. To see this, notice that the ratio between the i-th and $(i-1)$-st term (for each $i \geq 3$) equals $1/s_{i-2} \leq 1/16$. □

4 Output-sensitive skyline algorithms

This section will develop our new skyline algorithms. Sections 4.1 and 4.2 clear two more obstacles before we can start presenting our final solutions. In Section 4.3, we explain an output-sensitive algorithm under the assumption that the skyline size K is (magically) known. In Section 4.4, we remove this assumption, and give the ultimate algorithm.

4.1 M-skyline problem in 2d space Section 2 has presented a randomized algorithm for solving the M-skyline problem in linear expected I/Os in any fixed dimensionality. Exploiting our M-pivot algorithm, we obtain a deterministic solution in 2d space:

THEOREM 4.1. *The M-skyline problem on N points in 2d space can be solved in $O(N/B)$ I/Os deterministically.*

4.2 Dominance screening In the *dominance screening problem*, we are given two sets, denoted as R and S, of points in $\mathbb{R}^d$. The goal is to report the points of S that are not dominated by any point of R. We prove in the appendix:

LEMMA 4.1. *The dominance screening problem can be solved in $O(\frac{|R|+|S|}{B} \log^{d-2}_{M/B} \frac{|R|}{B})$ I/Os in any fixed dimensionality $d \geq 3$.*

Note that the logarithmic term is unrelated to $|S|$. Also, the lemma implies that the problem in 2d space (i.e., a special case of 3d) can be solved in $O(\frac{|R|+|S|}{B} \log_{M/B} \frac{|R|}{B})$ I/Os.

4.3 Skyline algorithm when K is known If $K \leq M$, the problem can be settled by our M-skyline algorithm, which runs in $O(N/B)$ expected cost for $d \geq 3$ (Theorem 2.1) and in the worst case (Theorem 4.1). On the other hand, if $K \geq \sqrt{NM}$, it holds that $N \leq K^2/M$. For $d \geq 3$, we can simply apply the output-*insensitive* algorithm of [14] to solve the problem in $O(\frac{N}{B} \log^{d-2}_{M/B} \frac{N}{B}) = O(\frac{N}{B} \log^{d-2}_{M/B} \frac{K^2}{MB}) = O(\frac{N}{B} \log^{d-2}_{M/B} \frac{K}{B})$ I/Os. Similarly, for $d = 2$, we can solve the problem in $O(\frac{N}{B} \log_{M/B} \frac{N}{B}) = O(\frac{N}{B} \log_{M/B} \frac{K}{B})$ I/Os.

The subsequent discussion considers only $M < K < \sqrt{NM}$. Set $f = \Theta(K/M)$. We divide P into disjoint partitions $P_1, ..., P_f$ of roughly the same size, such that every point in P_i has a smaller x-coordinate than all points in P_j for any i, j satisfying $1 \leq i < j \leq f$.

LEMMA 4.2. *$P_1, ..., P_f$ can be produced in $O(\frac{N}{B} \log_{M/B} \frac{K}{B})$ I/Os.*

We inspect P_i in ascending order of i. When P_i is finished, we guarantee that the skyline R of $P_1 \cup ... \cup P_i$ should have been found. In other words, $R = SKY(P)$ after $P_1, ..., P_f$ have all been processed. The next lemma below bounds the cost of handling each P_i.

LEMMA 4.3. *For $d \geq 3$, the processing of P_i takes $O(\frac{N}{fB} \log^{d-2}_{M/B} \frac{K}{B} + \frac{NK_i}{fMB})$ expected I/Os, where K_i is the number of points of $SKY(P)$ that are found in P_i. For $d = 2$, the cost is $O(\frac{N}{fB} \log_{M/B} \frac{K}{B} + \frac{NK_i}{fMB})$ in the worst case.*

Therefore, for $d \geq 3$, the overall expected cost is bounded by

$$\begin{aligned} & O\left(\sum_{i=1}^{f} \left(\frac{N}{fB} \log^{d-2}_{M/B} \frac{K}{B} + \frac{NK_i}{fMB}\right)\right) \\ = \; & O\left(\frac{N}{B} \log^{d-2}_{M/B} \frac{K}{B} + \frac{NK}{fMB}\right) \\ = \; & O\left(\frac{N}{B} \log^{d-2}_{M/B} \frac{K}{B} + \frac{N}{B}\right). \end{aligned}$$

With a similar argument for $d = 2$, we arrive at:

LEMMA 4.4. *For fixed $d \geq 3$, when K is known, the skyline of N points can be found in $O(\frac{N}{B} \log^{d-2}_{M/B} \frac{K}{B})$ expected I/Os. In 2d space, when K is known, the skyline can be found in $O(\frac{N}{B} \log_{M/B} \frac{K}{B})$ I/Os.*

4.4 The ultimate algorithm Finally, we remove the assumption that K is known. The subsequent discussion concentrates on $d \geq 3$ because the 2d case can be handled in the same fashion. The main idea behind our approach is to guess K strategically. The idea is hardly new (see, e.g., [6]), but its application in our context requires some careful treatment, as shown below.

From now on, let $K^\star$ be the actual number of skyline points (which is unknown), and K be our current guess which will always satisfy $M < K < \sqrt{NM}$. We will refer to the algorithm of Lemma 4.4 as *rigid-size*. Imagine running *rigid-size* with K anyway (i.e., potentially feeding a wrong guess) with one modification: the algorithm terminates by reporting *failure* as soon as it finds $K + 1$ skyline points. From the previous subsection, we have:

COROLLARY 4.1. *For fixed $d \geq 3$, the following are true for some constant $c > 0$:*

- *When $K^\star \leq K < \sqrt{NM}$, rigid-size finds the entire skyline within $3^d c \frac{N}{B} \log^{d-2}_{M/B} \frac{K}{M}$ I/Os with probability at least $1 - 3^{-d}$.*

– *When $M < K < K^\star$ rigid-size always fails.*

We are ready to elaborate our final algorithm. First, start by using our M-skyline algorithm to check whether $K^\star \leq M$; if so, the problem has been solved. Otherwise, we run *rigid-size* with guess K (whose value will be given later), while counting how many I/Os have been performed. As soon as the I/O count has exceeded $3^d c\frac{N}{B}\log_{M/B}^{d-2}\frac{K}{M}$, we *abort* the algorithm (i.e., forcing it to stop). In other words, *rigid-size* has three possible outcomes: normally terminated, failed, and aborted. The entire algorithm finishes in the first outcome, whereas in the other two, we increase K and re-run *rigid-size*. In general, K equals $M \cdot (M/B)^{2^i}$ (so that $\log_{M/B}(K/M) = 2^i$) when running *rigid-size* for the i-th ($i \geq 1$) time. As soon as $K \geq \sqrt{NM}$, however, we stop using *rigid-size* and instead deploy an output-*insensitive* algorithm, as explained in Section 4.3.

Leaving the analysis to the appendix, we now present the final main result:

THEOREM 4.2. *For any fixed $d \geq 3$, we can find the skyline of N points in d-dimensional space using $O(\frac{N}{B}\log_{M/B}^{d-2}\frac{K}{B})$ expected I/Os, where K is the number of skyline points. For $d = 2$, the running time is $O(\frac{N}{B}\log_{M/B}\frac{K}{B})$ I/Os in the worst case.*

Remark. It is not difficult to show that our 2d algorithm is optimal in the class of comparison-based algorithms. For example, by the transformation in [11], we can show that a skyline algorithm can be used to remove duplicates in a multi-set of N elements where only K of them are distinct. The latter problem is known to have a lower bound of $\Omega(\frac{N}{B}\log_{M/B}\frac{K}{B})$ [3].

References

[1] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 23–34, 2012.

[2] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.

[3] L. Arge, M. Knudsen, and K. Larsen. A general lower bound on the I/O-complexity of comparison-based algorithms. In *Algorithms and Data Structures Workshop (WADS)*, pages 83–94, 1993.

[4] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM (CACM)*, 23(4):214–229, 1980.

[5] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences (JCSS)*, 7(4):4 48–461, 1973.

[6] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

[7] T. M. Chan, K. G. Larsen, and M. Patrascu. Orthogonal range searching on the ram, revisited. In *Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.

[8] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 135–143, 1984.

[9] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 1993.

[10] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 58–66, 2001.

[11] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Symposium on Computational Geometry (SoCG)*, pages 89–96, 1985.

[12] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM (JACM)*, 22(4):469–476, 1975.

[13] A. D. Sarma, A. Lall, D. Nanongkai, and J. Xu. Randomized multi-pass streaming skyline algorithms. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):85–96, 2009.

[14] C. Sheng and Y. Tao. Finding skylines in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 107–116, 2011.

[15] Z. Zhang, Y. Yang, R. Cai, D. Papadias, and A. K. H. Tung. Kernel-based skyline cardinality estimation. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 509–522, 2009.

Appendix 1: A mathematical fact

Next we show: for any real x and integer y satisfying $0 < x < 1$, $y \geq 1$, and $xy \leq 3/4$, it holds that $(1-x)^y < 1 - xy/2$. This is trivial for $y = 1$. If $y = 2$, then $(1-x)^y < 1 - x = 1 - xy/2$. The rest of the proof considers $y \geq 3$, for which we have:

$$\begin{aligned}(1-x)^y &= \sum_{i=0}^{y}\binom{y}{i}(-x)^i \\ &= 1 - yx + \sum_{i=2}^{y}\binom{y}{i}(-x)^i \\ &< 1 - yx + \sum_{i=2}^{y}\binom{y}{i}x^i\end{aligned}$$

Define $z_i = \binom{y}{i}x^i$ so that we can simplify the above to

$$(1-x)^y \quad < \quad 1 - yx + \sum_{i=2}^{y} z_i \tag{4.8}$$

For any $i \geq 3$:

$$\begin{aligned} \frac{z_i}{z_{i-1}} &= \frac{\binom{y}{i}x^i}{\binom{y}{i-1}x^{i-1}} \\ &= \frac{y-i+1}{i}x \\ (\text{as } i \geq 3,\ y-i+1 < y) &< \frac{yx}{3} \\ (\text{as } xy < 3/4) &< 1/4. \end{aligned}$$

Therefore, for any $i \geq 3$, it holds that

$$z_i < z_2 \cdot \left(\frac{1}{4}\right)^{i-2}.$$

It follows that

$$\begin{aligned} (4.8) &< 1 - yx + \sum_{i=2}^{y} z_2 \cdot \left(\frac{1}{4}\right)^{i-2} \\ &< 1 - yx + \frac{4}{3}z_2 \\ &= 1 - yx + \frac{4}{3} \cdot \frac{y(y-1)}{2} \cdot x^2 \end{aligned}$$

To prove the above is at most $1 - yx/2$, it remains to show

$$\begin{aligned} \frac{4}{3} \cdot \frac{y(y-1)}{2} \cdot x^2 &\leq \frac{xy}{2} \\ \Leftrightarrow \frac{4}{3}(y-1)x &\leq 1 \\ \Leftrightarrow \frac{4}{3}xy &\leq 1 + \frac{4}{3}x \\ (\text{as } xy < 3/4) \Leftarrow 1 &\leq 1 + \frac{4}{3}x \end{aligned}$$

which is true.

Appendix 2: Proof of Corollary 2.1

If (2.1) holds, at the end of *3-pass*, $|P|$ has at most $(\frac{1}{2} + \frac{1}{e^c})N$ points in expectation. Hence, by the Markov inequality[3]:

$$\mathbf{Pr}\left[|P| \geq \left(\frac{1}{2} + \frac{1}{e^c}\right)\frac{N}{\rho}\right] \quad \leq \quad \rho.$$

Remember $c = 0.7$. Hence, with probability at least $1-\rho$, $|P|$ is smaller than $(\frac{1}{2} + \frac{1}{e^c})\frac{N}{\rho} < 0.9996N$.

[3] If X is a non-negative random variable, the Markov inequality says that $\mathbf{Pr}[X \geq \mathbf{E}[X]/y] \leq y$ for any $y > 0$.

Now consider that (2.1) does not hold. At the end of *3-pass*, the sample set Σ has more than $m/4$ points in expectation. Set

$$z = m - |\Sigma|.$$

Note that $z \geq 0$ because Σ can have at most m samples. Also:

$$\mathbf{E}[z] = m - \mathbf{E}[|\Sigma|] < 3m/4.$$

By the Markov inequality:

$$\begin{aligned} \mathbf{Pr}\left[z \geq \frac{3m}{4\rho}\right] &\leq \rho. \\ \Rightarrow \mathbf{Pr}\left[m - |\Sigma| \geq \frac{3m}{4\rho}\right] &\leq \rho. \\ \Rightarrow \mathbf{Pr}\left[|\Sigma| \leq m - \frac{3m}{4\rho}\right] &\leq \rho. \end{aligned}$$

Hence, with probability at least $1-\rho$, $|\Sigma|$ is greater than $m - \frac{3m}{4\rho} > 0.247m$.

Appendix 3: Proof of Lemma 3.1

If N is a multiple of s^2M, Σ has exactly $gsM = N/s$ samples so the lemma obviously holds. Next, we consider that N is not a multiple of s^2M.

Write $N = \alpha s^2 M + \beta$ where α and β are integers satisfying $\alpha \geq 0$ and $\beta \in [1, s^2M - 1]$. Hence, $g = \alpha + 1$ and $\lfloor N/s \rfloor = \alpha sM + \lfloor \beta/s \rfloor$. Clearly:

$$\alpha sM \leq \lfloor N/s \rfloor \leq \alpha sM + sM - 1. \tag{4.9}$$

As mentioned before, there are altogether between $s(g-1)M + 1$ and sgM sub-groups, from each of which a sample is taken. Hence, $|\Sigma| \leq sgM = s(\alpha+1)M = \alpha sM + sM \leq \lfloor N/s \rfloor + sM \leq N/s + sM$. On the other hand, $|\Sigma| \geq s(g-1)M + 1 = \alpha sM + 1 > \alpha sM$, which by (4.9) is at least $\lfloor N/s \rfloor + 1 - sM > N/s - sM$.

Appendix 4: Proof of Lemma 3.2

Among the x values of Σ in $(v_1, v_2]$, suppose that x_i of them come from group G_i $(1 \leq i \leq g)$. Clearly, $\sum_{i=1}^{g} x_i = x$.

Let us focus on G_i of any $i \in [1, g]$. Denote by X_i the set of values (from S) in $G_i \cap (v_1, v_2]$. Among the sub-groups of G_i having at least one value in X_i (such a sub-group is said to *touch* X_i), X_i *entirely* covers all those sub-groups except possibly two. Figure 3 illustrates the idea with each box representing a sub-group. X_i fully covers 3 sub-groups, and contains a portion of the two boxes intersecting the vertical lines at v_1 and v_2, respectively. The dot in a box denotes the maximum value of the corresponding sub-group, i.e., the dot is a sample in Σ.

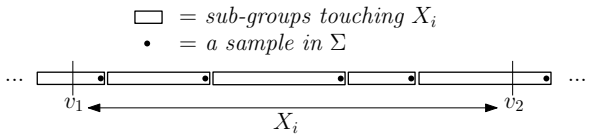

Figure 3: Proof of Lemma 3.2

X_i can touch at most $x_i + 1$ sub-groups. Otherwise (if X_i touches at least $x_i + 2$), the greatest values of at least $x_i + 1$ sub-groups would have fallen in $(v_1, v_2]$, contradicting the definition of x_i. Hence, $|X_i| \leq s(x_i + 1)$. On the other hand, X_i must fully cover at least $x_i - 1$ sub-groups. Otherwise (if X_i covers at most $x_i - 2$), the greatest values of at most $x_i - 1$ sub-groups could have fallen in $(v_1, v_2]$, also contradicting the definition of x_i. Hence, $s(x_i - 1) \leq |X_i|$.

It follows that

$$\sum_{i=1}^{g} s(x_i - 1) \leq \sum_{i=1}^{g} |X_i| \leq \sum_{i=1}^{g} s(x_i + 1)$$
$$\Rightarrow sx - sg \leq \sum_{i=1}^{g} |X_i| \leq sx + sg$$

$g = \lceil N/(s^2 M) \rceil < n/s^2 + 1$. Hence, $\sum_{i=1}^{g} |X_i|$ is more than $sx - n/s - s$ and less than $sx + n/s + s$. Recall that $s \leq \sqrt{n}$, which implies $s \leq n/s$. Therefore, $sx - 2n/s < \sum_{i=1}^{g} |X_i| < sx + 2n/s$.

Appendix 5: Proof of Lemma 3.3

$\sqrt{n} < \frac{n}{4 \log_2 n} < \frac{n}{4 \log_{M/B} n}$ is obvious so we focus on the first inequality. When $n > 65536$, $\log_2^4 n < n$. Hence:

$$\begin{aligned} \frac{n}{\prod_{i=1}^{h-1} \lfloor \log_{M/B}^{(i)} n \rfloor} &> \frac{\sqrt{n}\sqrt{n}}{\prod_{i=1}^{h-1} \log_2^{(i)} n} \\ &> \frac{\sqrt{n} \log_2^2 n}{\prod_{i=1}^{h-1} \log_2^{(i)} n} \\ &= \sqrt{n} \frac{\log_2 n}{\prod_{i=2}^{h-1} \log_2^{(i)} n} > \sqrt{n}. \end{aligned}$$

Therefore, $\prod_{i=1}^{h-1} s_i < \sqrt{n}$.

Appendix 6: Proof of Corollary 3.2

Since $2s_{i-1}...s_1 > s_1 \geq 16$, we know $\sqrt{2s_{i-1}...s_1} < s_{i-1}...s_1$. Hence:

$$\begin{aligned} s_i &\leq \frac{s_{h-1}...s_1}{s_{i-1}...s_1} \\ \text{(by Lemma 3.3)} &< \frac{\sqrt{n}}{s_{i-1}...s_1} \\ &< \frac{\sqrt{n}}{\sqrt{2s_{i-1}...s_1}} \\ \text{(by Corollary 3.1)} &< \sqrt{n_i}. \end{aligned}$$

Appendix 7: Proof of Theorem 4.1

The M-skyline algorithm described next is due to [13]; we sketch it here for completeness. Let P be a set of 2d points. We run our M-pivot algorithm on the x-coordinates of the points in P. In linear I/Os, the algorithm reports pivots, in ascending order, $v_1, ..., v_{M-1}$. They determine M vertical slabs partitioning the data space $\mathbb{R}^2$, where the i-th ($1 \leq i \leq M$) slab σ_i has an x-range set to $(v_{i-1}, v_i]$, defining $v_0 = -\infty$ and $v_M = \infty$. Note that P has $\Theta(N/M)$ points in each σ_i.

Next, we scan P once to obtain, for each $i \in [1, M]$, the lowest point p_i in $P \cap \sigma_i$, i.e., p_i has the smallest y-coordinate among the points in σ_i. Now, compute in memory the skyline Π of $\{p_1, ..., p_M\}$, namely, the skyline of just M points. It is easy to see that all the points in Π are also in the skyline of P. Furthermore, if p_i for some i is *not* in Π, the entire slab σ_i can be pruned, because whatever point dominating p_i also dominates all the other points in $P \cap \sigma_i$. It is easy to verify that, we either have reported at least $M/2$ skyline points, or have pruned $(M/2) \cdot \Theta(N/M) = \Theta(N)$ points from P.

If less than M skyline points have been found, we re-run the above algorithm on the set of points in P that are not dominated by any point in Π. *Given* Theorem 3.1, we can now apply the analysis in [13] to show that the total cost is $O(N/B)$.

Appendix 8: Proof of Lemma 4.1

Partial sorting. To prove Lemma 4.1, we cannot afford to sort S as this will bring about a term $\log_{M/B} \frac{|S|}{B}$ in the running time. Nevertheless, we can still sort S *partially*, by the definition below.

DEFINITION 1. *Let R and S be two sets of points in $\mathbb{R}^d$. Let U be a sequence including all the points in R and S. In this sequence, S is **partially sorted against R** if the following are true:*

- *Each point p is assigned an integer label l_p.*
- *For any two points p_1 and p_2 with $l_{p_1} < l_{p_2}$, it must*

hold that (i) p_1 precedes p_2 in U, (ii) $p_1[d-1] \leq p_2[d-1]$, and (iii) in case $p_1[d-1] = p_2[d-1]$, p_1 is lexicographically smaller than p_2.[4]

- *At most $m = M/d$ points from R have an identical label.*

Viewed differently, U can be partitioned into *trunks* such that all points in the same trunk carry an identical label. For this reason, we may sometimes refer to a label as a *trunk id*. The above definition essentially states that points in different trunks must have been properly ordered (in the sense of the second bullet), whereas we do not care about the ordering within a trunk.

LEMMA 4.5. *The sequence U in Definition 1 can be obtained in $O(\frac{|R|+|S|}{B} \log_{M/B} \frac{|R|}{B})$ I/Os.*

Proof. Using a subroutine of the *distribution sort* [2], we can divide R (obeying the ordering requirement) into $\sqrt{M/B}$ partitions of roughly the same size, and distribute S accordingly. This requires $O(\frac{|R|+|S|}{B})$ I/Os. Then, we can focus on each partition recursively. The recursion terminates when the number of points from R is at most m. The depth of the recursion is $O(\log_{M/B} \frac{|R|}{M}) = O(\log_{M/B} \frac{|R|}{B})$. □

The h-screening problem. Let us tackle a related problem called *h-screening*. The input is:

- A parameter h satisfying $0 \leq h \leq d-2$.
- For each dimension $i \in [1, h]$, a set of $f = \Theta((M/B)^{\frac{1}{d-2}})$ slabs $\sigma_i(1), ..., \sigma_i(f)$. If $h = 0$, no slab needs to be given. Each set of $\sigma_i(1), ..., \sigma_i(f)$ is obtained by cutting the data space $\mathbb{R}^d$ with $f-1$ hyperplanes orthogonal to dimension i, so that for any point $p \in \sigma_i(j)$ and $p' \in \sigma_i(j')$, we have $p[i] < p'[i]$ if $1 \leq j < j' \leq f$.
- A sequence of the points in R sorted on dimension $h+1$.
- A sequence U of $R \cup S$ in which S is partially sorted against R. In addition, U has the property that a point p of S is marked if (i) there exists some point of R dominating p, and (ii) both points fall in the same slab $\sigma_i(j)$ for some $i \in [1, h]$ and $j \in [1, f]$.

If a point $p \in S$ is dominated by some point in R, we say that p is *screened*. The output of the problem has one or two parts:

- A sequence of R sorted on dimension $h+2$ if $h < d-2$. We do not need this part if $h = d-2$;
- A sequence U of $R \cup S$ where S is partially sorted against R, and all the screened points of S are marked.

LEMMA 4.6. *The h-screening problem can be solved in $O(\frac{|R|+|S|}{B} \log_{M/B}^{d-h-2} \frac{|R|}{B})$ I/Os.*

Proof. We handle $h = d-2$ and $h < d-2$ separately.

Case 1: $h = d-2$. In this case, the problem can be settled by scanning the trunks of U in ascending order of the trunk ids. During the scan, we maintain a set of f^h values $\lambda(i_1, ..., i_h)$, where $1 \leq i_j \leq f$ for each $j \in [1, h]$. Each $\lambda(i_1, ..., i_h)$ records the minimum coordinate on dimension d of all the points of R that (i) fall in $\sigma_1(i_1) \cap ... \cap \sigma_h(i_h)$ and (ii) belong to trunks that have already been fully scanned. If no point satisfies both conditions, $\lambda(i_1, ..., i_h) = \infty$. With the choice of f, these f^h values fit in memory.

Now let us focus on a trunk. We first read the trunk once to buffer all points from R in memory (recall that there can be at most m such points). Then, scan the trunk again, and mark all points from S screened by a point of R in memory. Next, we scan the trunk one more time, and mark all points from S screened by a point of R in trunks already scanned. Specifically, let $p \in S$ be in this trunk, and $\sigma_1(i_1), ..., \sigma_h(i_h)$ be the slabs that p falls in. If

$$p[d] \geq \min_{j_1 < i_1 ... j_h < i_h} \lambda(j_1, ..., j_h),$$

p must be screened, and hence, is marked.

As each trunk is processed only once, $(d-2)$-screening can be settled in $O(|U|/B)$ I/Os.

Case 2: $h < d-2$. If $|R| \leq m$, the problem can easily be solved in $O(|U|/B)$ I/Os by keeping R in memory. The subsequent discussion assumes that R does not fit in memory. We will convert the problem to $(h+1)$-screening.

First, divide U into disjoint sequences $U_1, ..., U_f$, such that:

- the points in U_i have smaller coordinates on dimension $h+1$ than those in U_j for any i, j with $1 \leq i < j \leq f$;
- for each $i \in [1, f]$, U_i has $\Theta(|R|/f)$ points from R;
- for each $i \in [1, f]$, S_i is partially sorted against R_i, where $R_i = U_i \cap R$ and $S_i = U_i \cap S$.

Sequences $U_1, ..., U_f$ can be generated in $O(|U|/B)$ I/Os. Recall that we have at our disposal a list of R

[4] Namely, there is an $i \in [1, d]$ such that $p_1[i] < p_2[i]$ while $p_1[j] = p_2[j]$ for all $j < i$.

sorted on dimension $h+1$. Scanning the list once, we can partition the data space along this dimension into f disjoint vertical slabs $\sigma_{h+1}(1), ..., \sigma_{h+1}(f)$ such that R has roughly the same number of points in each slab. Keeping the slab definitions in memory, in linear time, we can divide U into $U_1, ..., U_f$, while ensuring that the points in each U_i retain their mutual ordering in U.

For each $i \in [1, f]$, perform dominance screening on (R_i, S_i) recursively, which is another h-screening problem. On return, we have obtained:

- a sequence of R_i sorted on dimension $h+2$;
- a sequence U_i of $R_i \cup S_i$ where S_i is partially sorted against R_i, and every point of S_i screened by R_i is marked.

In $O(|U|/B)$ I/Os, we can merge $U_1, ..., U_f$ into a sequence U of $R \cup S$ where S is partially sorted against R (replacing the original U), and merge the sequences of $R_1, ..., R_f$ into another sequence of R sorted on dimension $h+2$. We have obtained all the inputs to an $(h+1)$-screening problem, which is therefore recursively solved.

Running time of h-screening. Set $T = |R|$ and $N = |S|$. Let $G_h(N, T)$ be the number of I/Os required to solve h-screening, and $H_h(N, T)$ the cost of converting h-screening to $(h+1)$-screening. We have:

$$G_h(N,T) = \begin{cases} O((N+T)/B) & \text{if } h = d-2 \text{ or } T = O(M) \\ H_h(N,T) + G_{h+1}(N,T) & \text{otherwise} \end{cases}$$

with

$$H_h(N,T) = \sum_{i=1}^{f} G_h(N_i, T/f) + O((N+T)/B)$$

where N_i is the number of points in S_i for each $i \in [1, f]$. Solving the recurrence yields $G_h(N,T) = O(\frac{N+T}{B} \log_{M/B}^{d-h-2} \frac{T}{B})$, which proves Lemma 4.6. □

Dominance screening. We are now ready to solve the dominance screening problem. Sort R on the first dimension. Generate a sequence U of $R \cup S$ in which S is partially sorted against R, which requires $O(\frac{|R|+|S|}{B} \log_{M/B} \frac{|R|}{B})$ I/Os (Lemma 4.5). Now, solve a 0-screening problem with R and U as the input. On return, all the screened points are marked in U. We can extract those points with another scan of U. By Lemma 4.6, the entire process takes $O(\frac{|R|+|S|}{B} \log_{M/B}^{d-2} \frac{|R|}{B})$ I/Os.

Appendix 9: Proof of Lemma 4.2

Assume first that N is a power of 2; we will remove the assumption at end of the proof. Our algorithm is based on *distribution sort* [2]. Let u be a power of 2 that is $\Theta(\sqrt{M/B})$. Let f' be the largest power of u that is at most f. We will first explain how to divide P into f' partitions of roughly the same size, and then, further divide it into f partitions.

The core of *distribution sort* is a sub-routine that uses linear I/Os to partition P along the x-dimension into $\Pi_1, ..., \Pi_u$ of roughly the same size. After this, we can recursively apply the sub-routine on each Π_i $(1 \leq i \leq u)$, until there are f' partitions in total. As the number of partitions increases by a factor of u after each level of recursion, the number of levels is $O(\log_u f')$. The overall cost is therefore $O(\frac{N}{B} \log_u f') = O(\frac{N}{B} \log_{M/B} \frac{K}{B})$.

There is, however, a problem with the above strategy. The sizes of $\Pi_1, ..., \Pi_u$ are only *roughly* the same, i.e., two partitions can differ in their sizes by a constant factor. However, as there are $\log_{M/B} \frac{K}{B}$ recursion levels, the size ratio of two partitions can be multiplied non-constant times, such that the sizes of two final partitions can differ by a non-constant factor, thus beating our objective. One way to remedy the problem is to make the sizes of $\Pi_1, ..., \Pi_u$ *perfectly* the same. Fortunately, given $\Pi_1, ..., \Pi_u$, we can create such a perfect partition $\Pi'_1, ..., \Pi'_u$ of P in $O(N/B)$ I/Os.

The main idea is to run the *k-selection* algorithm [5] $O(u)$ times, each of which is carried out on some Π_i. As $|\Pi_i| = O(N/u)$, each run incurs $O(N/(uB))$ I/Os so the overall cost is still $O(N/B)$. To understand the details, first note that each Π'_i $(1 \leq i \leq u)$ should have exactly N/u points. We thus aim at deciding $u-1$ pivots $v_1, ..., v_{u-1}$ (in ascending order) such that exactly N/u points of P have x-coordinates in $(v_{i-1}, v_i]$ for each $i \in [1, u]$ (define dummy $v_0 = -\infty$ and $v_u = \infty$). To find v_1, we run *k-selection* to retrieve from Π_1 the point with the (N/u)-th smallest x-coordinate. Note that the retrieval may *fail* by returning nothing if $|\Pi_1| < N/u$. However, in that case, we finish with Π_1 and move on to retrieve the point from Π_2 having the $(N/u - |\Pi_1|)$-th smallest x-coordinate. In general, we may fail only u times, whereas a new pivot is decided every time we do not. Hence, *k-selection* is executed at most $2u$ times.

We now have obtained f' partitions $P_1, ..., P_{f'}$ of P. To obtain f partitions, we divide each P_i into $\lceil f/f' \rceil$ sub-partitions of roughly the same size. As $\lceil f/f' \rceil \leq u$, the division of P_i can be accomplished by invoking the aforementioned sub-routine of *distribution sort* only once. After having divided all of $P_1, ..., P_{f'}$, we may have ended up with more than f partitions. In this case, pair-wisely merge some partitions to make the number

exactly f. It is easy to verify that the final partitions have roughly the same size.

Finally, let us deal with the case where N is not a power of 2. Let N' be the smallest power of 2 greater than N. Add $N'-N$ dummy points to P, ensuring that the dummy points have x-coordinates smaller than all the points in P. Denote by P' the resulting dataset. Use our earlier algorithm to partition P' into $P'_1, ..., P'_{f'}$ with exactly the same size. Let $P'_1, ..., P'_x$ be the partitions containing dummy points. Note that $x \leq f'/2$. Discard $P'_1, ..., P'_{x-1}$, move all the non-dummy points of P'_x into P'_{x+1}.

At this point, P has been divided into at least $f'/2$ but at most f' partitions of roughly the same size. Similar to the method explained before, we further divide them into f partitions of roughly the same size by invoking the sub-routine of *distribution sort* at most twice.

Appendix 10: Proof of Lemma 4.3

Next we describe how to handle P_i, given the skyline R of $P_1 \cup ... \cup P_{i-1}$. Let us first consider $d \geq 3$. We start by eliminating from P_i those points dominated by any point in R. This can be done with the dominance screening algorithm of Lemma 4.1, whose cost is

$$\begin{aligned} & O\left(\frac{|P_i|+|R|}{B}\log^{d-2}_{M/B}\frac{|R|}{B}\right) \\ = \; & O\left(\left(\frac{N}{fB}+\frac{K}{B}\right)\log^{d-2}_{M/B}\frac{K}{B}\right) \\ & (\text{as } |P_i| = O(N/f) \text{ and } |R| \leq K) \\ = \; & O\left(\frac{N}{fB}\log^{d-2}_{M/B}\frac{K}{B}\right) \\ & (\text{as } f = \Theta(K/M) \text{ and } K < \sqrt{NM}, \text{ thus} \\ & K/B < \frac{NM}{KB} = O\left(\frac{N}{fB}\right)). \end{aligned}$$

After the screening, we can concentrate on finding the skyline S of the *remaining* P_i. Every point of S is definitely in $SKY(P)$, i.e., $K_i = |S|$.

The entire S can be retrieved in $O(\frac{N}{fB} + \frac{NK_i}{fMB})$ expected I/Os as follows. Execute the M-skyline algorithm of Theorem 2.1 on P_i, which fetches a set S' of skyline points in $O(N/(fB))$ expected I/Os. If S' has less than M points, we are done because the entire S has been extracted. Otherwise, scan P_i once to remove all the points dominated by any point in S', which takes only linear I/Os because S' can be kept in memory. Now, repeat the above by executing the M-skyline algorithm again. There are at most K_i/M repeats because each repeat discovers M new skyline points except perhaps the last one. Hence, the total cost is $O(\frac{N}{fB}+\frac{N}{fB}\frac{K_i}{M})$ expected.

The above discussion also applies to $d = 2$ by using instead the M-skyline algorithm of Theorem 4.1. All the I/O cost is thus in the worst case. Also note that dominance screening is now completed in $O(\frac{N}{fB}\log_{M/B}\frac{K}{B})$ I/Os.

Appendix 11: Proof of Corollary 4.1

We focus on $K^\star \leq K < \sqrt{NM}$ because the other case of the corollary is obvious. We will show that *rigid-size* reports the entire skyline in $O(\frac{N}{B}\log^{d-2}_{M/B}\frac{K}{B}) = O(\frac{N}{B}\log^{d-2}_{M/B}\frac{K}{M})$ expected I/Os, from which the corollary follows with a simple application of the Markov inequality. In fact, it is easy to verify that both Lemmas 4.2 and 4.3 still hold. The total cost of *rigid-size* is thus $O(\frac{N}{B}\log^{d-2}_{M/B}\frac{K}{B} + \sum_{i=1}^{f}\frac{NK_i}{fMB}) = O(\frac{N}{B}\log^{d-2}_{M/B}\frac{K}{B} + \frac{NK^\star}{fMB}) = O(\frac{N}{B}\log^{d-2}_{M/B}\frac{K}{B})$, noticing that $f = \Theta(K/M) = \Omega(K^\star/M)$.

Appendix 12: Proof of Theorem 4.2

We concentrate on $d \geq 3$ because the argument for $d = 2$ is analogous. We assume $K^\star > M$ because otherwise the expected cost is obviously $O(N/B)$.

First consider that our guess K of the skyline size never reaches $\sqrt{NM}$. Set $K_i = M \cdot (M/B)^{2^i}$ for $i \geq 1$. The i-th run of *rigid-size* incurs at most

$$\begin{aligned} 3^d c \cdot (N/B) \cdot \log^{d-2}_{M/B}(K_i/M) &= 3^d c \cdot (N/B) \cdot 2^{i(d-2)} \\ &= O(N/B) \cdot 2^{i(d-2)} \end{aligned}$$

I/Os. Let z be the smallest value satisfying $K_z \geq K^\star$. The total cost of the first z runs of *rigid-size* is at most

$$\begin{aligned} & O(N/B) \cdot (2^{d-2} + 2^{2(d-2)} + ... + 2^{z(d-2)}) \\ (4.10) = \; & O(N/B) \cdot 2^{z(d-2)}. \end{aligned}$$

Since $K_{z-1} < K^\star$, we know:

$$\begin{aligned} M \cdot (M/B)^{2^{z-1}} &< K^\star \\ \Rightarrow 2^{z-1} &< \log_{M/B}(K^\star/M) \\ \Rightarrow 2^z &< 2\log_{M/B}(K^\star/M). \end{aligned}$$

Therefore, we continue (4.10) with

$$\begin{aligned} O(N/B) \cdot 2^{z(d-2)} &= O(N/B) \cdot 2^{d-2} \cdot \log^{d-2}_{M/B}\frac{K^\star}{M} \\ (4.11) \qquad &= O\left(\frac{N}{B}\log^{d-2}_{M/B}\frac{K^\star}{B}\right). \end{aligned}$$

Next we will show that, the cost of *rigid-size* beyond its z-th run is $O(\frac{N}{B}\log^{d-2}_{M/B}\frac{K^\star}{B})$ in expectation. Assume that the algorithm eventually performs $x > z$ runs of *rigid-size*. With derivation analogous to (4.10), we know

that the total running time is dominated by the cost of the last run, i.e., $O(N/B) \cdot 2^{x(d-2)}$. By Corollary 4.1, for any $i \geq z$, the probability that i-th run does not complete on itself is at most $1/3^d$ (namely, the run does not find the entire skyline). To necessitate the x-th run, self-completion happens to none of the z-th, $(z+1)$-st, ..., $(x-1)$-th runs. Hence, the probability of terminating at the x-th run is at most $(1/3^d)^{x-z}$. It follows that the expected cost the algorithm incurs after the z-th run is bounded by:

$$\begin{aligned}
& \sum_{x=z+1}^{\infty} \left(O(N/B) \cdot \frac{2^{x(d-2)}}{3^{d(x-z)}} \right) \\
\leq & \sum_{x=z+1}^{\infty} \left(O(N/B) \cdot 2^{z(d-2)} \cdot \frac{2^{(x-z)(d-2)}}{3^{(x-z)(d-2)}} \right) \\
= & \ O(N/B) \cdot 2^{z(d-2)} \sum_{x=z+1}^{\infty} \left((2/3)^{d-2} \right)^{x-z} \\
= & \ O(N/B) \cdot 2^{z(d-2)}
\end{aligned}$$

which is $O(\frac{N}{B} \log_{M/B}^{d-2} \frac{K^\star}{B})$ as shown in (4.11).

Finally, consider that our final guess K reaches $\sqrt{NM}$ eventually. In this case, our algorithm invokes an output-insensitive algorithm that performs $O(\frac{N}{B} \log_{M/B}^{d-2} \frac{N}{B}) = O(\frac{N}{B} \log_{M/B}^{d-2} \frac{K}{B})$ I/Os. Note that this can be more expensive than $3^d c(N/B) \log_{M/B}^{d-2}(K/M)$ by at most a constant factor. Therefore, our earlier analysis shows that the overall expected cost is still $O(\frac{N}{B} \log_{M/B}^{d-2} \frac{K^\star}{B})$.