Skyline Computation with Noisy Comparisons

Frederik Mallmann-Trenn\textsuperscript{1,2}, Claire Mathieu\textsuperscript{1,3}, and Victor Verdugo\textsuperscript{1,4}

\textsuperscript{1}École normale supérieure, CNRS, PSL Research University, Paris, France, 
\{mallmann, cmathieu, victor.verdugo\}@di.ens.fr
\textsuperscript{2}Simon Fraser University
\textsuperscript{3}Collège de France; work partly done while visiting IRIF (Université Paris-Diderot)
\textsuperscript{4}Universidad de Chile

October 6, 2017

Abstract

Given a set of \( n \) points in a \( d \)-dimensional space, we seek to compute the skyline, i.e., those points that are not strictly dominated by any other point, using few comparisons between elements. We study the crowdsourcing-inspired setting ([FRPU94]) where comparisons fail with constant probability. In this model, Groz & Milo [GM15] show three bounds on the query complexity for the skyline problem.

We provide two output-sensitive algorithms computing the skyline with query complexity \( O(nd \log(dk)) \) and \( O(ndk \log(k)) \), where \( k \) is the size of the skyline. These results improve significantly on the state-of-the-art and are tight for low dimensions.

1 Introduction

Skylines have been studied extensively, since the 1960s in statistics [BS66], then in algorithms and computational geometry [KLP75] and in databases [BKS01; CCM13; GSG07; KRR02]. Depending on the field of research, the skyline is also known as the set of maximum vectors, the dominance frontier, admissible points, or Pareto frontier. The skyline of a set of points consists of those points which are not strictly dominated by any other point. A point \( p \) is dominated by another point \( q \) if \( p_i \leq q_i \) for every coordinate (attribute or dimension) \( i \). It is strictly dominated if in addition the inequality is strict for at least one coordinate; see Figure 1 (from [GM15]) for an example.

In many contexts, comparing attributes is not straightforward. Consider the example of finding optimal cities from [GM15].

To compute the skyline with the help of the crowd we can ask people questions of the form “is the education system superior in city \( x \) or city \( y \)” or “can I
Figure 1: Given a set of points $X$, the goal is to find the set of skyline points, i.e., points are not dominated by any other points.

"expect a better salary in city x or city y". Of course, people are likely to make mistakes, and so each question is typically posed to multiple people. Our objective is to minimize the number of questions that need to be issued to the crowd, while returning the correct skyline with high probability.

Thus, much attention has recently been given to computing the skyline when information about the underlying data is uncertain [MWK+11], and comparisons may give erroneous answers. One may consider that the location of each point is determined by a probability distribution over a set of locations, or that data is incomplete [KML08; LEB13]. Some previous work [PJLY07; AAA+11] model uncertainty about the output by computing a $\rho$-skyline: points having probability at least $\rho$ to be in the skyline. In this paper, we work in the noisy comparison model, which was introduced in the seminal paper [FRPU94] and has been studied in [GM15; BMW16]: We assume queries are of the type is the $i$-th coordinate of point $p$ (strictly) smaller than that of point $q$?, and the outcome of each such query is independently correct with probability greater than some constant better than $1/2$ (for definiteness we assume probability $2/3$). Our goal is to recover the exact skyline, with error probability at most $\delta$. In the context of crowdsourcing this model has been considered in order to capture the fact that people might incur in errors when comparing elements. We refer to [AZH+15] about skyline computation using the crowd and [LWZF16] for a survey in crowdsourced data management.

**Results** In many settings the skyline consists of very few points compared to the input size, motivating the study of output-sensitive algorithms. Our measure of complexity is the number of queries. This is expressed as a function of three parameters: $n = |X|$, the number of data items (points); $d$, the number of attributes (dimensions); and $k = |\text{sky}(X)|$, the size of the skyline (output).
Theorem 1.1. Given $\delta \in \left(0, \frac{1}{2}\right)$ and a set $X$ of data items, SkyLowDim-Search$(X, \delta)$ outputs a subset of $X$ which, with probability at least $1 - \delta$, is the skyline of $X$. The expected number of queries is $O(nd \log(dk/\delta))$.

Theorem 1.2. Given $\delta \in \left(0, \frac{1}{2}\right)$ and a set $X$ of data items, SkyHighDim-Search$(X, \delta)$ outputs a subset of $X$ which, with probability at least $1 - \delta$, is the skyline of $X$. The expected number of queries is $O(ndk \log(k/\delta) + dk^2 \log(kn/\delta))$.

Additionally, we prove that the bound of Theorem 1.1 is (up to constant factors) essentially tight whenever $d \leq k^c$ for any constant $c$:

Theorem 1.3. Let $A$ be an algorithm that computes the skyline with error probability less than $1/10$. Then the expected number of queries of $A$ is $\Omega(nd \log k)$.

Techniques In previous work, Groz and Milo [GM15] give three algorithms. Of the three algorithms, the third (iii) algorithm is simply based on sorting all the input points in each dimension and thus reduces the problem to the case of computing the skyline in the noiseless setting. Our algorithm SkylineLowDim uses a natural but quite different idea: it is to use discretization, by sampling, sorting all the sample points in each dimension to define buckets, then placing points into buckets, and identifying “skyline buckets”. Eliminating points in dominated buckets, we reduce the input size significantly allowing us to apply a cruder algorithm to solve the problem on the smaller input. One interesting aspect of our discretization is that a fraction of the input will be, due to the low query complexity, incorrectly discretized yet we are able to recover the correct skyline.

Algorithms (i) and (ii) from [GM15], recover the skyline points one by one. They iteratively compute the maximum point, in lexicographic order, among those not dominated by the skyline points already found. The idea behind our algorithm SkylineHighDim is that it is more efficient to separate the two tasks: finding a point $p$ not dominated by the skyline points already found, on the one hand, and computing a maximum point (in lexicographic order) among those dominating $p$, on the other hand; we optimize queries carefully for the latter of the two tasks (Algorithm MaxLex).

Our lower bound constructs a technical reduction from the problem of identifying null vectors among a collection of vectors, each having at most one non-zero coordinate. That problem can be studied using a two-phase process inspired from [FRPU94].

Context Groz and Milo [GM15] start the research line of computing the skyline in the noisy setting and they give three algorithms showing three upper bounds on the query complexity: (i) $O(ndk \log(dk))$, (ii) $O(ndk^2 \log(k))$, and (iii) $O(nd \log(dn))$. Here, we improve the first two bounds by a factor of $k$ and we improve on the third bound.

When do our bounds improve asymptotically on the existing bounds? We are focused on settings where the output size $k$ is quite small compared to $n$: assume that $k = n^o(1)$ (If $k = n^{\Omega(1)}$ then the simple bound (iii) from [GM15] is best.) Moreover,

\footnote{The difference between (i) and (ii) is due to different subroutines to check dominance.}
• If the dimension is relatively low: \( d = k^{O(1)} \), then Theorem 1.1 is best, beating (i) by a factor of \( k \), and is in fact optimal by Theorem 1.3. This comprises in particular the constant dimension setting: Our bound \( O(n \log k) \) is tight and improves on \( O(n \log n) \) ([GM15]) for small enough \( k \) (e.g. poly-log \( n \)).

• If the dimension is quite high: \( d = 2^{\omega(k)} \), then Theorem 1.2 is best, beating (ii) by a factor of \( k \).

Roadmap In Section 2, we provide the preliminaries, which the reader might wish to skip on first reading. Section 3 introduces our algorithm for low dimensions (Theorem 1.1) and Section 4.2 introduces the counterpart for high dimensions (Theorem 1.2). Section 5, contains our lower bound (Theorem 1.3).

2 Preliminaries

Algorithm \texttt{SkyLowDim-Search}(X, \delta) guesses an upper bound \( k \) for \(|\text{skyline}(X)|\) by a super-exponentially increasing sequence of guesses (similarly to [Cha96; GM15]). This reduces the problem to that of computing \text{skyline}(X) given a rough upper bound \( k \) on its cardinality, a problem solved by our Algorithm \texttt{SkylineLowDim}(k, X, \delta).

\begin{algorithm}
\textbf{Algorithm \texttt{SkyLowDim-Search}(X, \delta)} (see Theorem 1.1)
\begin{algorithmic}[1]
\State \textbf{input:} \( X \) set of points, \( \delta \) error probability
\State \textbf{output:} \text{skyline}(X)
\State \textbf{error probability:} \( \delta \)
\State 1: \( k \leftarrow ([d/\delta])^2 \)
\State 2: \textbf{repeat}
\State 3: \( \delta \leftarrow \delta/2 \); \( k \leftarrow k^2 \); \( S \leftarrow \text{SkylineLowDim}(k, X, \delta) \)
\State 4: \textbf{until} \( |S| < k \)
\State 5: Output \( S \)
\end{algorithmic}
\end{algorithm}

Similarly, algorithm \texttt{SkyHighDim-Search}(X, \delta) guesses an upper bound \( k \) for \(|\text{skyline}(X)|\) by an exponentially increasing sequence of guesses. This reduces the problem to that of computing \text{skyline}(X) given an upper bound \( k \) on its cardinality, a problem which will be solved by Algorithm \texttt{SkylineHighDim}(k, X, \delta).

2.1 Subroutines used by our algorithms

\textit{Sorting, searching and skyline.} Before we state our Algorithm \texttt{SkylineLowDim}, we introduce the subroutines it builds on, namely, \texttt{SkyGM} from [GM15], and \texttt{NoisySearch} and \texttt{NoisySort} from [FRPU94]. In particular, the algorithm \texttt{SkyGM} yields the complexity in (i) and (ii). The pseudocode of these three routines is provided in the Appendix.

4
Algorithm SkyHighDim-Search\((X, \delta)\) (see Theorem 1.2)

**input:** \(X\) set of points, \(\delta\) error probability

**output:** skyline\((X)\)

**error probability:** \(\delta\).

1: Initialize \(j \leftarrow 0, k \leftarrow 1\)
2: repeat
3: \(j \leftarrow j + 1; k \leftarrow 2k; S \leftarrow \text{SkylineHighDim}(k, X, \delta/8^j)\)
4: until \(|S| < k\)
5: Output \(S\)

**Theorem 2.1** ([GM15]). Given \(\delta \in (0, 1/2)\) and a set \(X\) of data items, algorithm SkyGM\((X, \delta)\) outputs a subset of \(X\) which, with probability at least \(1 - \delta\), is the skyline of \(X\). The expected number of queries is (i) \(O(ndk^2 \log(k/\delta))\) or (ii) \(O(ndk \log(dk))\). The complexity depends on the dominance test used.

In the **noisy binary search problem** the input is the following: an element \(y\), and an ordered list \((y_1, y_2, \ldots, y_{m-1})\), accessible by comparisons that each have error probability at most \(p\), and a parameter \(\delta\); the goal is to output the interval \(I = (y_{i-1}, y_i]\) such that \(y \in I\).

**Theorem 2.2** ([FRPU94]). There exists an algorithm, NoisySearch, that solves the noisy binary search problem with success probability \(1 - \delta\) and expected number of comparisons \(O(\log(m/\delta))\).

In the **noisy sort problem** the input correspond to an unordered set \(Y = \{y_1, y_2, \ldots, y_m\}\), whose elements are accessible by comparisons that each have error probability at most \(p\), and a parameter \(\delta\). The goal is to output an ordering of \(Y\) that is the correct non-decreasing sorted order.

**Theorem 2.3** ([FRPU94]). There exists an algorithm, NoisySort, that solves the noisy sorting problem with success probability \(1 - \delta\) and expected number of comparisons \(O(m \log(m/\delta))\).

**Boosting.** A folklore approach to deal with noise is to take an algorithm for the noiseless setting and repeat each noisy operation enough times to reduce noise and boost the success probability; this increases the query complexity by logarithmic factors. We use this approach in a variety of settings, so we formalize it with a (higher order) algorithm which we call BoostProb. See Section B.2.

**Proposition 2.4.** Algorithm BoostProb\((\otimes, \delta_1, \delta_2)\) takes as input two parameters \(\delta_1\) and \(\delta_2\) that are the desired two-sided errors, and a test \(\otimes\) (query or algorithm) that returns either true or false with error probability at most \(1/3\). BoostProb incorrectly outputs true (false positive) w.p. \(\delta_1\) and incorrectly outputs false (false negative) w.p. \(\delta_2\).

Let \(T_{\otimes}\) be the expected query time of test \(\otimes\). BoostProb has expected query complexity \(O(\log(1/\delta_1))T_{\otimes}\) if \(\otimes\) should return true (YES instances) and \(O(\log(1/\delta_2))T_{\otimes}\) otherwise (NO instances).
For our algorithm **SkylineLowDim** to be able to eliminate dominated buckets, we design an efficient subroutine to test whether a bucket is empty of points.

In the **Bucket-emptiness** problem the input corresponds to a bucket \( B \), a set of points \( Y \) and two error parameters \( \delta_1 \) and \( \delta_2 \). The goal is to decide whether \( Y \cap B = \emptyset \); the algorithm incorrectly outputs true w.p. at most \( \delta_1 \), and incorrectly outputs false w.p. at most \( \delta_2 \). (The two-sided error is used to improve the query complexity). We leave the description of the algorithm **IsEmpty** and it analysis to Section B.2.

**Lemma 2.5.** Algorithm **IsEmpty** \((B, Y, \delta_1, \delta_2)\) solves Bucket-emptiness with expected query complexity \( O(d|Y| \log(1/\delta_1)) + O(d \log(d|Y|/\delta_2)) \).

### 3 Skyline computation in low dimension: Theorem 1.1

#### 3.1 Overview

**Algorithm SkylineLowDim** \((k, X, \delta)\)

*input*: \( k \) integer, \( X \) set of points, \( \delta \) error probability  
*output*: \( \min\{k, |\text{skyline}(X)|\} \) points of skyline \((X)\)  
*error probability*: \( \delta \)

1. \( \delta' \leftarrow \delta/(2dk)^5 \) and \( s \leftarrow dk^2 \log(dk^2/\delta') \)
2. if \( k^5 \geq n \) or \( d^5 \geq n \) then
3. Output **SkyGM** \((X, \delta')\)
   
   **{Phase (i): bucketing}**
4. for each dimension \( i \in \{1, 2, \ldots, d\} \) do
5. \( S_i \leftarrow \text{NoisySort}(\text{sample of } X \text{ of size } s, i, \delta'/d) \)
6. Remove duplicates so that, with prob. \( 1 - \delta'/d \), the values in \( S_i \) are all distinct
7. for each point \( p \in X \) do
8. Place \( p \) in set \( X_B \) associated to \( B = \prod_{i=1}^d I_i \), with \( I_i = \text{NoisySearch}(p_i, S_i, \delta'/(dk)) \).
9. for each bucket \( B \) do
10.\( \text{empty}_B \leftarrow \begin{cases} 
    \text{true} & \text{if } X_B = \emptyset, \\
    \text{unknown} & \text{otherwise.}
\end{cases} \)
   
   **{Phase (ii): elimination}**
11. while \( \exists B \) with \( \text{empty}_B = \text{unknown} \) and \( \forall B' \) dominating \( B \): \( \text{empty}_{B'} = \text{true} \) do
12.\( \text{empty}_B \leftarrow \text{IsEmpty}(B, X_B, \delta'/k, \delta'/n) \)
13. if more than \( |X|/\log(|X|) \) buckets \( B \) have \( \text{empty}_B = \text{false} \) then Output **SkylineLowDim** \((k, X, \delta)\)
14. \( X' \leftarrow \cup\{X_B : \text{empty}_B = \text{false} \) and \( \forall B' \) dominating \( B \): \( \text{empty}_{B'} = \text{true} \} \)
   
   **{Phase (iii): solve reduced problem}**
15. Output **SkylineHighDim** \((k, X', \delta')\).
Figure 2: An illustration of the bucket dominance. We say a bucket $i$ dominates another bucket $j$ if it is non-empty and has strictly larger coordinates. Dominated buckets are gray-striped and dominating buckets are orange-striped, and boxed. Here bucket $b$ dominates $c$ and $f$ but not $a$, $d$, $e$ or $g$. Bucket $b$ is undominated because all buckets to the Northeast are empty. All points in all undominated buckets are handed to the third phase, ensuring that the skyline points $p_1$, $p_2$, and $p_3$ are among these points.
Algorithm SkylineLowDim \((k, X, \delta)\) uses phases. In the first phase, bucketing: we sort the \(i\)-th coordinate of a random sample to define \(s + 1\) intervals in each dimension \(i \in [d]\), hence \((s + 1)^d\) buckets, where each bucket is a product of intervals of the form \(\prod_i I_i\); then we place each point \(p\) of \(X\) in those buckets by searching for each dimension for the interval \(I_i\) containing \(p_i\).

In the second phase, Algorithm SkylineLowDim \((k, X, \delta)\) uses elimination: Say that a bucket \(B = \prod_i I_i\) is dominated by a different bucket \(B' = \prod_i I'_i\) if in every dimension \(\max I_i \leq \min I'_i\). Then every point in \(B'\) dominates every point in \(B\), so no skyline point belongs to a bucket dominated by a non-empty bucket. We test buckets for emptiness and eliminate points placed in buckets that are dominated by non-empty buckets. See Figure 2 for an illustration.

In the third phase Algorithm SkylineLowDim \((k, X, \delta)\) simply calls the Algorithm SkylineHighDim to find the skyline of the remaining points.

### 3.2 Algorithm SkylineLowDim

We can now formally define the Algorithm SkylineLowDim that was outlined in section 3.1. In the third phase we have eliminated enough points that we can call our other (less efficient) algorithm SkylineHighDim (see Section 4.2) on the residual instance.

### 3.3 Analysis: Proof of Theorem 1.1

The error probability analysis follows by carefully considering the operations made by the algorithm. We leave that analysis to the Appendix, and focus on the query complexity analysis.

To study the query complexity of algorithm SkylineLowDim we need some structure about the bucketing. We say the assignment performed in line 8 is decent if the following holds: For every \(j \in \{1, \ldots, d\}\) and for every interval \(I_j\) in \(S_j\), at most \(4|X|/(dk^2)\) points with distinct \(j\) coordinate are placed in \(I\), that is,

\[
|\{p_j : I = \text{NoisySearch}(p_j, S_j, \delta')/(dk)\}| \leq \frac{4|X|}{dk^2}.
\]

Otherwise, if the condition above does not hold for an interval we say that \(I\) is dense.

**Lemma 3.1.** With probability at least \(1 - 1/k\), the assignment performed in line 8 is decent.

We first study the expected number of queries performed during the execution of SkylineLowDim \((k_i, X, \delta)\), where \(k_i = [d/\delta]^{2i}\). Throughout this section we assume \(k_i^5, d^5 \leq n = |X|\) since otherwise the algorithm just calls SkyGM and thus the correctness simply follows from the correctness of SkyGM. We now study the query complexity of each phase in SkylineLowDim.

---

2Alternatively, one could use an algorithm provided by Groz and Milo [GM15], it is only important that the size of the input set is reduced to \(n/k\) to cope with the larger runtime of the mentioned algorithms.
• **Phase (i) Bucketing.** On line 5: by Theorem 2.3 (noisy sorting) the expected number of queries performed is \(d \cdot O(dk_i^2 \log(16d^2k_i^2/\delta)) = O(n\log(k_i d/\delta))\), by the assumption over \(k_i\) and \(d\). On line 8: by Theorem 2.2 (noisy search) the expected number of queries is \(nd \cdot O(\log(16d^2k_i^2/\delta)) = O(nd\log(dk_i/\delta))\).

• **Phase (ii) Elimination.** On line 12: by Lemma 2.5, since (conditioned on an execution where NoisySort and NoisySearch were both correct) at most \(\gamma = k_i^2d\log(16d^2k_i^2/\delta)\) buckets tested are non-empty. Observe that \(\gamma < |X|/\log(|X|)\) due to line 2. The resulting conditional expected query complexity is \(O(|X|d\log(dk_i/\delta))\). An incorrect execution of algorithms NoisySort and NoisySearch can cause at most \(|X|/\log(|X|)\) buckets to be verified, however, the probability of such an error is at most \(1/k\). The probability of failing \(i\) times due to line 13 is at most \(1/k^i\). Thus, resulting conditional expected query complexity is \(O(|X|d\log(dk_i/\delta))\). Hence, by law of total expectation, and since the probability of executing the expected number of queries is at most \(O(|X|d\log(dk_i/\delta))\).

• **Phase (iii) Recovering the skyline.** On line 15: We claim that on expectation \(|X'| = O(n/k_i)\). which gives, by Theorem 4.3, \(O(|X'|d\log(dk_i/\delta))\). We proceed by proving the claim. Assume that the point assignment is decent, which is, by Lemma 3.1, w.p. \(1 - 1/k\) the case. Let \(Q_p\) be the set of points which were strictly dominated by \(p\) and in some bucket \(B\) which was not dominated resulting in \(p \in X'\). Let \(B_p\) be the bucket of \(p\). Consider every dimension \(i\). If \(I_i\) of \(B_p\) is of type \((\ell, r]\), then because the assignment was decent there are at most \(L = 4|X'|(dk_i^2/\delta)\) points in \(Q_p\) w.r.t. to dimension \(i\). If \(I_i\) of \(B_p\) is of type \(I = [x], x \in \mathbb{R}\), then for every \(q \in Q_p\) correctly assigned to \(I\) we see that \(p\) dominates \(q\). Thus summing over all dimensions we have \(|Q_p| \leq |X|/k_i^2\) and

\[
|X'| \leq \left| \bigcup_{p \subseteq \text{skyline}(X)} Q_p \right| \leq |X|/k_i.
\]

Suppose now that an error occurred (which happens w.p. at most \(1/k\)) we derive, as before, \(|X'| \leq |X|\). Thus, similarly as before, the expected size of \(|X'|\) is \(|X'| = O(n/k_i)\), yielding the claim. Hence, the expected number of queries of that last line is \(O(nd\log(dk_i/\delta))\).

Let \(O_i\) be the output of SkyLowDim\((k_i, X, \delta/2^i)\). Recall that SkyLowDim-Search\((X, \delta)\) terminates as soon as \(|O_i| \neq k_i\). Whenever \(k_i > |\text{skyline}(X)|\) such an output is returned w.p. at least \(1 - \delta/2^i\). Let \(Z\) denote the variable denoting the number of extra iterations due to failure, Set \(\gamma = \log(\log(1/d|\text{skyline}(X)|))\) and recall that \(\log(\cdot) = \log_2(\cdot)\). Since \(k_i \leq (d/\delta)^{2^i}\), the number of iterations is bounded by \([\gamma] + 1 + Z\). Therefore, the runtime
of SkylineLowDim($k_i, X, \delta/2^i$) is bounded by $O(nd \log(2^i dk_i/\delta))$. This implies that the total expected query complexity is upper bounded by

$$\sum_{i=1}^{[\gamma]+1+Z} O(nd \log(2^i dk_i/\delta)).$$

To obtain the claimed bound, we first show that

$$\sum_{i=1}^{[\gamma]+1+Z} \log(2^i dk_i/\delta) \leq 32 \cdot 2^Z \log |\text{skyline}(X)|.$$

By using this inequality and that $\mathbb{P}(Z = i) \leq \delta^i$ and $\delta < 1/2$ is enough to conclude, since

$$\sum_{i=0}^{\infty} O\left(nd 2^i \log \left(\frac{|\text{skyline}(X)|d}{\delta}\right)\right) \mathbb{P}(Z = i) = O\left(nd \log \left(\frac{|\text{skyline}(X)|d}{\delta}\right)\right).$$

We now check that (2) holds:

$$\sum_{i=1}^{[\gamma]+1+Z} \log(2^i dk_i/\delta) \leq 2 \sum_{i=1}^{[\gamma]+1+Z} \log(dk_i) \leq 2 \sum_{i=1}^{[\gamma]+1+Z} \log((d/\delta)^{2^i+1})$$

$$\leq 4 \log(d/\delta) \cdot 2^{[\gamma]+2+Z} \leq 32 \cdot 2^Z \log(d/\delta) \log d/\delta |\text{skyline}(X)|$$

$$= 32 \cdot 2^Z \log |\text{skyline}(X)|.$$

4 Skyline computation in high dimension: Theorem 1.2

Algorithm SkylineHighDim builds the skyline incrementally by discovering skyline points one by one. We obtain an additional skyline point by first looking for a point $p$ which is not dominated by the current set of skyline points; $p$ itself is not necessarily a skyline point. Then we find a skyline point $p^*$ dominating $p$: it has to be a new, additional skyline point and can thus be added to the set.

We note that an algorithm from [GM15] also builds the skyline incrementally, but it looks for a point that is simultaneously not dominated by the current skyline and is itself a skyline point: by separating those two tasks and optimizing each, we gain a factor of $k$ in complexity.

4.1 Subroutines: Domination and Lexicographic Maximum

Algorithm SkylineHighDim uses a subroutine for the Domination Problem: the input is a set of points $S$, a point $q \in S$, and two error parameters $\delta_1$ and $\delta_2$. The goal is to check whether the point $q$ is not dominated by $S$, that is, there is no point in $S$ that dominates $q$ on every dimension. Our subroutine is a more efficient variant of an approach already taken in [GM15, Lemma 5]. The algorithm and analysis can be found in Section B.4.
Lemma 4.1. There is an algorithm for the domination problem, \text{SetDominates}(S, q, \delta_1, \delta_2), that is correct w.r.t. to its specification and has expected query complexity $O(\lvert S \rvert d \log(1/\delta_2) + d \log(\lvert S \rvert/\delta_1))$.

Algorithm \text{SkylineHighDim} also uses a subroutine for \text{Lex-Maximum}: the input is a point $p$, a set of points $S$ and an error probability $\delta$. This algorithm computes the maximum point in the lexicographic order\(^3\) in $S$ that is not dominated by $p$. In [GM15, Proposition 2], the authors sketch an algorithm doing the same.

Algorithm \text{MaxLex} aims to find the maximum point in lexicographic order, among the ones that dominate $p$, by keeping a counter for each point, that is increased when that point is found to dominate $p$ and decreased when it is either less than another point in lexicographic order or found not to dominate $p$. The increments are chosen in such a way that the counter of the unknown desired output point performs a random walk biased upwards with non-uniform step size, the others perform random walks biased downwards, and we always compare (and compare to $p$) the points with the largest current counters. We leave the full description of the algorithm and its analysis to Section B.4.

Lemma 4.2. There is an algorithm for lex-maximum, \text{MaxLex}(p, S, \delta), that is correct w.r.t. to its specification and has expected query complexity $O(\lvert S \rvert d \log(1/\delta))$.

4.2 Algorithm SkylineHighDim

We are now ready to give the full description of the algorithm \text{SkylineHighDim}.

Theorem 4.3. Suppose $k \geq \lvert \text{skyline}(X) \rvert$. Then, \text{SkylineHighDim}(k, X, \delta)$ outputs $\text{skyline}(X)$ w.p. at least $1 - \delta$ with expected query complexity $O(dk n \log(k/\delta))$.

The proof of Theorem 1.2 follows the same lines as the proof of Theorem 1.1, so we leave it to the Appendix.

5 Lower Bound-Theorem 1.3

In this section, we exhibit an $\Omega(dn \log k)$ lower bound on the query complexity in the noisy skyline problem, denoted \text{Skyline}. To that end, we define a noisy vector problem, in which one is given $k$ vectors each of length $\ell$ and needs to decide for each vector whether it is the all-zero vector. We prove a lower bound for this problem and reduce it to \text{Skyline} yielding the desired result.

---

\(^3\)Given two points, $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ the first one is lexicographically smaller than the second one, denoted by $p \leq_{\text{lex}} q$, if $p_i < q_i$ for the first $i$ where $p_i$ and $q_i$ differ. If there is no such $i$ meaning that the points are identical we use the id of the points in the input as a tie-breaker, ensuring that we obtain a total order.
Algorithm SkylineHighDim\((k, X, \delta)\)  
(see Theorem 4.3)

**input:** \(X\) set of points, \(k\) upper bound on skyline size, \(\delta\) error probability

**output:** \(\min\{k, \text{skyline}(X)\}\) skyline points w.p. \(1 - \delta\)

1: Initialize \(S_0 \leftarrow \emptyset\), \(C \leftarrow X\)
2: \textbf{for} \(i = 1\) to \(k\) \textbf{do}
3: \{Find a point \(p\) not dominated current skyline points\}
4: \textbf{while} \(C\) not empty \textbf{do}
5: Pick an arbitrary \(p \in C\)
6: if not \textbf{SetDominates}\((S_i, p, \delta/4k, \delta/(4k|X|))\) (check if \(p\) is already dominated) then
7: \textbf{break}
8: \(C \leftarrow C \setminus \{p\}\)
9: \{Find a skyline point dominating \(p\)\}
10: Compute \(p^* \leftarrow \text{MaxLex}(C, p, \delta/2k)\)
11: \(S_i \leftarrow S_{i-1} \cup \{p^*\}\)
12: Output \(S_k\)

5.1 \((k, \ell)\)-Null-Vectors: definition and lower bound

**Definition 1.** In the \((k, \ell)\)-Null-Vectors the input \(S\) is a collection \(\{v^1, v^2, \ldots, v^k\}\) of \(k\) vectors such that for each \(i\), \(v^i \in \{0, 2\}^\ell\) and \(\sum_j v^i_j \leq 2\), and the output is a vector \((w_1, w_2, \ldots, w_k) \in \{0, 2\}^k\) such that for each \(i\), \(w_i = \sum_j v^i_j\).

We define the distribution \(\mu\) over vectors of \(\{0, 2\}^\ell\) as follows.

\[
\begin{align*}
\mathbf{v} = & \left\{ \\
(0,0,\ldots,0) & \text{ with probability } 1/2 \\
(1,0,\ldots,0) & \text{ with probability } 1/(2\ell) \\
\ldots & \\
(0,\ldots,0,1) & \text{ with probability } 1/(2\ell) \\
\right. 
\end{align*}
\]

For inputs to \((k, \ell)\)-Null-Vectors, we will consider the product distribution \(\mu^k\).

**Lemma 5.1.** For \((k, \ell)\)-Null-Vectors under the product distribution \(\mu^k\), if \(A\) is is a deterministic algorithm with success probability at least \(3/4\), then the worst case number of queries of \(A\) is \(\Omega(\ell k \log k)\).

The proof can be found in Section B.5.

5.2 Reduction from \((k, \ell)\)-Null-Vectors to Skyline

**Step 1.** Assume, for simplicity, that \(d-2\) divides \(k\). From an input \(S = \{u^1, u^2, \ldots, u^k\}\) to the \((k, \ell)\)-Null-Vectors, we first show how to construct an input \(I_S\) for Skyline with \(n\) points
in \(d\) dimensions and a skyline that is likely to be of size \(k\), where \(n = (\ell + d - 2)k/(d - 2)\). We first randomly permute the entries of each \(u^j\), by using \(k\) independent permutations, resulting in \(S_\pi = \{v^1, v^2, \ldots, v^k\}\). Partition \(S_\pi\) into \(k/(d - 2)\) blocks of \(d - 2\) vectors, where for \(j \in \{0, 1, \ldots, k/(d - 2) - 1\}\), block \(S_\pi^j = \{v^{j(d - 2) + q}: i \in [d - 2]\}\).

For each block, define \(\ell + d - 2\) points, as displayed (one point per row) on Figure 3, and the union over all blocks is the input \(\mathcal{I}_S\) to the Skyline. Formally, we define point \(p^{(t)}\) with \(t = j\ell + i\) as follows.

\[
p^{(t)}_d := \begin{cases} 
  j & \text{if } d' = d - 1, \\
  n - j & \text{if } d' = d, \\
  1 & \text{if } d' = i - \ell \text{ and } \ell \leq i, \\
  v^{j(d - 2) + i} & \text{if } d' = i \text{ and } i \in [d - 2], \\
  0 & \text{otherwise.}
\end{cases}
\]

Figure 3: Block \((j, n - j)\) of the reduction. The vectors of \(S_\pi^j\) placed in this block are \(v^{j(d - 2) + q}, v^{j(d - 2) + 2}, \ldots, v^{j(d - 2) + (d - 2)}\).

**Step 2.** Because of the non-domination implied by the last two coordinates of any point, the skyline of the set of points is the sum over all blocks of the skyline of each block. Fix an arbitrary block and focus on the first \(d - 2\) dimensions. For each dimension, the corresponding column (whose first \(\ell\) coordinates are those of some vector \(v^j\)) contains exactly one 1 (on the row of some point \(p\)) and possibly one 2, the remaining entries being all 0. Thus it is easy to verify that \(p\) is part of the skyline if and only if \(v^j = 0\).

From the output \(\text{sky}(I)\) it is now easy to construct the output of the \((k, \ell)\)-Null-Vectors: For all blocks, for all dimensions \(\leq d - 2\), if \(p \in \text{sky}(I)\) then \(w_i \leftarrow 0\) else \(w_i \leftarrow 2\). This yields the correct output \(w = (w_1, w_2, \ldots, w_k)\). Thus we derive the following observation.

**Observation 1.** Given the set of points \(\text{sky}(\mathcal{I}_S)\), one can recover the solution to the \((k, \ell)\)-Null-Vectors without further queries.
In the following we prove that the construction is likely to have \( k \) skyline points.

**Observation 2.** Let \( \mathcal{E} \) be the event that the input \( \mathcal{I}_S \) has exactly \( k \) skyline points. Then, \( \mathbb{P}(\mathcal{E}) \geq 1 - 1/k \) as long as \( k \leq \sqrt[3]{n} \).

**Proof.** First observe that, by construction, regardless of whether \( \mathcal{E} \) holds, every block contains at most \( d - 2 \) skyline points: Consider an arbitrary block. The last two dimensions are identical for each point belonging to that block and we focus thus on the first \( d - 2 \) dimensions.

There are exactly \( d - 2 \) points with one coordinate being 1 and all of these points are potential skyline points. In particular, take any such point \( p \) and assume that the \( i \)'th coordinate of \( p \) is 1. Then \( p \) is part of the skyline if and only if the vector \( v^i \) is the null vector. Moreover, every block can have at most \( d - 2 \) entries with value 2 and each such 2 eliminating one potential skyline point. Thus, there are at most \( d - 2 \) skyline points per block.

Consider the vertices \( v^{i_1}, v^{i_1+1}, \ldots, v^{i_2} \) of any block. We say they are collision free if the following holds: if \( v^j = 2 \) for \( j \in [i_1, i_2] \), then \( v^{j'} = 0 \) for all \( j' \in [i_1, i_2] \setminus \{j\} \). Observe that if the vertices of any block are collision free, then each of the first \( d - 2 \) dimensions is dominated by a distinct skyline point and thus there \( d - 2 \) skyline points in that block. Thus, if the vectors of every block are collision free, then there \( d - 2 \) skyline points per block and summing up over all \( k/(d - 2) \) blocks, we get that there are thus \( k \) skyline points in total.

Thus, in order to bound \( \mathbb{P}(\mathcal{E}) \) it suffices to bound the probability that all blocks are collision free. Recall that the random permutations \( \pi_1, \pi_2, \ldots, \pi_k \) permute each vector \( v^i \) independently. Since in a block at most \( k^2 \) pairs may collide, and each collision happens with probability \( 1/\ell \), the expected number of collisions per block is at most \( k^2/\ell \). The expected number of collisions over all blocks is thus, by the Union bound, at most \((k/(d - 2)) \cdot k^2/\ell \leq 1/k \), by assumption on \( k \). Thus, the claim follows by applying Markov inequality.

\[ \square \]

### 5.3 Proof of Theorem 1.3

Suppose for the sake of contradiction that there exists an algorithm \( \mathcal{A} \) recovering the skyline for any input with exactly \( k \) skyline points, with error probability at most \( 1/10 \), and using \( o(nd \log k) \) queries in expectation. By Markov’s inequality, the probability that the number of queries exceeds 5 times the expectation is at most \( 1/5 \), so truncating the execution at that point adds \( 1/5 \) to the error probability, transforming \( \mathcal{A} \) into an algorithm \( \mathcal{B} \) that recovers the skyline for any input with exactly \( k \) skyline points, with error probability at most \( 1/5 + 1/10 < 1/3 \), and using \( o(5nd \log k) \) queries in the worst case. We claim that this implies that one can solve the \((k, \ell)\)-**Null-Vectors** with \( o(nd \log k) \) w.p. at least \( 1/3 \) contradicting Lemma 5.1.
Let $S$ be the input of the $(k, \ell)$-Null-Vectors. We cast $S$ as an input $\mathcal{I}_S$ of $B$ as described in Section 5.2. By Observation 2, the event $\mathcal{E}$ holds w.p. at least $1 - 1/k$ and thus there are $k$ skyline points.

By assumption, $B$ can thus compute the skyline w.p. at least $1/2 - 1/k \geq 1/3$, where we used the Union bound. Thus, by Observation 1, one can obtain w.p. at least $1/3$ the solution to $(k, \ell)$-Null-Vectors using $o(\eta d \log k)$ queries, a contradiction.

**Future Work**

We show that that the query complexity is $\Theta(dn \log (dk))$ whenever $d \leq k^c$ for any constant $c$. The arising questions is thus what the query complexity is when $d = \omega(k^c)$. In the light of the upper bound $O(dkn \log k)$ of Theorem 1.2 we conjecture that there exists an algorithm achieving a query complexity of $\Theta(dn \log k)$.

Furthermore, we believe that for constant dimensions, the correct bound is $O(n \log k)$ regardless of the instance (assuming that there are exactly $k$-skyline points) even if the algorithm knows the instance up to a permutation—in other words, we believe that our algorithm is instance-optimal for constant dimensions.

**Acknowledgements**

We gratefully acknowledge Vincent Cohen-Addad for sharing his insights and ideas.

**References**

[AAA+11] P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips. “(Approximate) Uncertain Skylines”. In: Proceedings of the 14th International Conference on Database Theory. ICDT ’11. Uppsala, Sweden: ACM, 2011, pp. 186–196. URL: http://doi.acm.org/10.1145/1938551.1938576 (cit. on p. 2).

[AZH+15] A. Asudeh, G. Zhang, N. Hassan, C. Li, and G. V. Zaruba. “Crowdsourcing pareto-optimal object finding by pairwise comparisons”. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM. 2015, pp. 753–762 (cit. on p. 2).

[BS66] O. Barndorff-Nielsen and M. Sobel. “On the Distribution of the Number of Admissible Points in a Vector Random Sample”. English. In: Theory of Probability and its Applications 11.2 (1966), pp. 249–21. URL: http://search.proquest.com/docview/915869827?accountid=15867 (cit. on p. 1).
[BKS01] S. Börzsönyi, D. Kossmann, and K. Stocker. “The Skyline Operator”. In: *Proceedings of the 17th International Conference on Data Engineering*. IEEE Computer Society, 2001, pp. 421–430. URL: [http://dl.acm.org/citation.cfm?id=645484.656550](http://dl.acm.org/citation.cfm?id=645484.656550) (cit. on p. 1).

[BMW16] M. Braverman, J. Mao, and S. M. Weinberg. “Parallel Algorithms for Select and Partition with Noisy Comparisons”. In: *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*. STOC ’16. Cambridge, MA, USA, 2016, pp. 851–862 (cit. on p. 2).

[Cha96] T. M. Chan. “Optimal output-sensitive convex hull algorithms in two and three dimensions”. In: *Discrete & Computational Geometry* 16.4 (1996), pp. 361–368. URL: [http://dx.doi.org/10.1007/BF02712873](http://dx.doi.org/10.1007/BF02712873) (cit. on p. 4).

[CCM13] J. Chomicki, P. Ciaccia, and N. Meneghetti. “Skyline Queries, Front and Back”. In: *SIGMOD Rec.* 42.3 (2013), pp. 6–18. URL: [http://doi.acm.org/10.1145/2536669.2536671](http://doi.acm.org/10.1145/2536669.2536671) (cit. on p. 1).

[FRPU94] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. “Computing with Noisy Information”. In: *SIAM Journal on Computing* 23.5 (1994), pp. 1001–1018 (cit. on pp. 1–5, 27–29).

[Fel68] W. Feller. *An Introduction to Probability Theory and Its Applications*. Vol. 1. Wiley, 1968. URL: [http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20%7B%7Dpath=ASIN/0471257087](http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20%7B%7Dpath=ASIN/0471257087) (cit. on p. 17).

[GSG07] P. Godfrey, R. Shipley, and J. Gryz. “Algorithms and Analyses for Maximal Vector Computation”. In: *The VLDB Journal* 16.1 (2007), pp. 5–28. URL: [http://dx.doi.org/10.1007/s00778-006-0029-7](http://dx.doi.org/10.1007/s00778-006-0029-7) (cit. on p. 1).

[GM15] B. Groz and T. Milo. “Skyline Queries with Noisy Comparisons”. In: *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. PODS ’15. Melbourne, Victoria, Australia: ACM, 2015, pp. 185–198. URL: [http://doi.acm.org/10.1145/2745754.2745775](http://doi.acm.org/10.1145/2745754.2745775) (cit. on pp. 1–5, 8, 10, 11, 29).

[KML08] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. “Skyline query processing for incomplete data”. In: 2008 *IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 556–565 (cit. on p. 2).

[KRR02] D. Kossmann, F. Ramsak, and S. Rost. “Shooting Stars in the Sky: An Online Algorithm for Skyline Queries”. In: *Proceedings of the 28th International Conference on Very Large Data Bases*. VLDB ’02. Hong Kong, China: VLDB Endowment, 2002, pp. 275–286. URL: [http://dl.acm.org/citation.cfm?id=1287369.1287394](http://dl.acm.org/citation.cfm?id=1287369.1287394) (cit. on p. 1).
A Auxiliary Claims

**Proposition A.1** ([Fel68, Chapter XIV.2, XIV.3]). Let $p \in (0, 1) \setminus \{1/2\}$ and $b, s \in \mathbb{N}$. Consider a discrete time Markov chain $(Z_t)_{t \geq 0}$ with state space $\Omega = [0, b]$ where

- $Z_0 = s \in [0, b]$
- $\mathbb{P}(Z_t = i \mid Z_{t-1} = i - 1) = p$ for $i \in [1, b - 1], t \geq 1$
- $\mathbb{P}(Z_t = i \mid Z_{t-1} = i + 1) = 1 - p$ for $i \in [1, b - 1], t \geq 1$
- $\mathbb{P}(Z_t = i \mid Z_{t-1} = i) = 1$ for $i \in \{0, b\}, t \geq 1$

Let $T = \min\{t \geq 0 \mid Z_t \in \{0, b\}\}$. Then,

$$\mathbb{P}(Z_T = b) = \frac{(1-p)^{s-b} - 1}{(1-p)^b - 1}$$

and

$$\mathbb{P}(Z_T = 0) = \frac{(1-p)^b - (1-p)^s}{(1-p)^b - 1}.$$
Moreover,

\[ E(T) = \frac{s}{1 - 2p} - \frac{b}{1 - 2p} \cdot \frac{1 - \left(\frac{1-p}{p}\right)^s}{1 - \left(\frac{1-p}{p}\right)^b}. \]

**B Appendix**

**B.1 Error probability analysis of SkylineLowDim**

The probability that the output of SkylineLowDim-Search\((X, \delta)\) is incorrect is the probability that there exists an iteration such that SkylineLowDim\((k, X, \delta)\) is incorrect, which by the union bound \(\sum_{j \geq 1} \delta / 2^j \leq \delta\), hence the error probability in Theorem 1.1. In the following we show that

\[ P(\text{output of SkylineLowDim}(k, X, \delta) \text{ incorrect}) \leq \frac{\delta}{2(2dk)^5} \]  \hspace{1cm} (2)

The proof proceeds by examining possible sources of error during the execution of Algorithm SkylineLowDim\((k, X, \delta)\) and bounding the probability that they occur. Let \(\delta' = \delta / (2dk)^5\). Observe that if \(k^5 \geq n\) or \(d^5 \geq n\) and SkyGM returns an incorrect skyline, then by Theorem E.1 this happens w.p. at most \(\delta'\).

- **Phase (i) Bucketing.** Some sample is incorrectly sorted in line 5: by Theorem 2.3 and Union Bound over \(i \in [d]\), error probability is at most \(\delta'\). The samples are correctly sorted in line 5, but some points considered in line 6 are either deleted when they should have been kept, or kept when they should have been deleted: by Theorem 2.3 and Union Bound over \(i \in [d], j \in [s - 1]\), error probability at most \(\delta'\). The samples are sorted correctly, but some skyline point is placed in the wrong buckets: by Theorem 2.2 and Union Bound over \(i \in [d]\) and \(p \in \text{skyline}(X)\), error probability at most \(\delta'\).

- **Phase (ii) Deleting dominated buckets.** The samples are sorted and filtered correctly and the skyline points are placed in the correct buckets, but on line 12 some bucket that contains a skyline point is (incorrectly) tested as empty: Let \(p \in B\) be a skyline point. By assumption, \(p \in X_B\). By the specification of IsEmpty, the probability that \(B\) is (incorrectly) tested as empty is bounded by \(\delta' / k\). Applying the Union Bound on the at most \(\min\{k, |\text{skyline}(X)|\}\) buckets where this might happen, the error probability is at most \(\delta'\). (Note that this never happens more than \(k\) times because as soon as more than \(k\) skyline buckets are tested non-empty, the algorithm stops with an error message.)

The samples are sorted and filtered correctly and the skyline points are placed in the correct buckets, but on line 12 some bucket that dominates a bucket containing
a skyline point is (incorrectly) tested as empty: For each dominating bucket, by Lemma 2.5 this has probability at most \( \delta'/n \). There are at most \( n \) such buckets (since each has \( X_B \neq \emptyset \)), so by the Union Bound the probability overall in this case is at most \( \delta' \).

- **Phase (iii) Recovering the skyline.** Finally, if the call to `SkylineHighDim` on line 15 results in an error, that has probability at most \( \delta' \) by the specification of `SkylineHighDim`.

Overall, the error probability sum up to at most \( 16\delta' \) and thus (2) holds.

*Proof of Lemma 3.1.* Recall that \( \delta' = \delta/(2dk)^5 \). For the ease of presentation we will assume that the line 6 is executed after the points are assigned to buckets (line 8). Assume the points of \( X \) are ordered w.r.t. to their \( j \)'th dimension, breaking ties arbitrarily. Consider these ordered points to be divided into blocks, each one having \( \ell = |X|/(dk^2) \) consecutive points. In particular, the number of blocks is at most \( dk^2 \).

Consider now the samples after line 5. Each block contains one sample with probability at least \( 1 - (1 - \ell/|X|)^{s-1} \geq 1 - \delta'/dk^2 \). Thus, in this event, the distance between any two samples is at most \( 2\ell \) which implies that the number of distinct values is bounded by \( 2\ell \). Furthermore, with probability \( \delta'(dk^2) \) the number of points placed incorrectly between any two samples in line 5 is at most \( 2|X|/(dk^2) \) w.p. at least \( 1 - \delta' \), by Theorem 2.2 and Chernoff bounds. Moreover, recall that by assumption, line 6 was executed correctly. We thus get that that the number of points in \( I \) is bounded by \( 2\ell \) (maximum distance between two samples) plus \( 2|X|/(dk^2) \) (the incorrectly sorted points), and the points added through the removal of duplicates (line 6), but these have all the same \( j \)-th coordinate. Therefore,

\[
|\{p_j: p \text{ was sorted into } I \text{ in line 8}\}| \leq 2\ell + \frac{2|X|}{dk^2} + \frac{4|X|}{dk^2}.
\]

By taking the union bound over all \( d \) dimensions and over all \( s \) intervals, the error probability is at most \( 3\delta' \).

**B.2 Subroutines analysis for SkylineLowDim**

*Proof of Proposition 2.4.* By symmetry, assume that \( \otimes \) should return true. Every query \( \otimes \) returns true w.p. at least \( 2/3 \), by assumption, so the difference between the number of queries returning true and the number of queries returning false is a random walk with bias \( 2/3 \). The algorithm outputs false if and only if a biased random walk starting at \( s = \log(1/\delta_2) \) reaches 0 before reaching \( b = \log(1/\delta_2) + \log(1/\delta_1) \). Applying Proposition A.1 part 1, the probability of reaching 0 is bounded by \( 2^{-\log(1/\delta_2)} = \delta_2 \).

---

\(^4\text{we assume } k^5, d^5 \leq n \text{ since otherwise the algorithm just calls } \text{SkyGM}.\)
Algorithm BoostProb(⊗, δ₁, δ₂)

input: a test ⊗ (query or algorithm) returning either true or false w.p. at least 2/3
output: whether ⊗ should return true
error: incorrectly outputs true w.p. δ₁ and incorrectly outputs false w.p. δ₂.

Execute/query ⊗ until one of the following two cases.

1. The number of times the outcome was true exceeds the number of false by \log(1/δ₁). In this case output true.
2. The number of times the outcome was false exceeds the number of true by \log(1/δ₂). In this case output false.

Furthermore, using Proposition A.1 part 2 we derive the claimed bound on the number of queries it takes for the algorithm to outputs either true or false since this happens when the random walk reaches either 0 or b. Each query takes \(T(⊗)\) expected time, concluding the proof.

Algorithm for Bucket-emptiness. To prove Lemma 2.5 we require an algorithm for testing whether a given point belongs or not to a bucket. The following algorithm, InBucket, solves this problem with constant error probability.

Algorithm InBucket(p, B = \prod_{i=1}^{d} I_i)

input: p point, B bucket
output: whether p ∈ B
error: incorrect output w.p. 1/16

for each dimension i ∈ [d] do
    if BoostProb(p_i /∈ I_i, 1/(16d), 1/16) then
        Output false
    Output true

Lemma B.1. InBucket(p, B) is correct w.r.t. to its specification and has expected query complexity \(O(d)\).

Proof. Suppose p ∈ B. In all d dimensions i we have \(p_i \in I_i\). By the Union bound, the probability that BoostProb(p_i /∉ I_i, 1/(16d), 1/16) returns incorrectly true is at most \(d \times 1/(16d) = 1/16\). The query complexity is, by Proposition 2.4 and linearity of expectation, \(d \cdot O(1) = O(d)\).

Now suppose that p /∉ B. Let i be a dimension such that \(p_i /∉ I_i\). For that value of i, the probability that BoostProb(p_i /∉ I_i, 1/(16d), 1/16) incorrectly returns false is at most 1/16. The expected query complexity for that i is, by query complexity of BoostProb, \(O(\log(d))\). If the algorithm observes correctly that \(p_i /∉ I_i\), then it terminates without testing any further dimensions. Thus, the expected total query complexity is given by two terms: the
expected query complexity for all dimensions $i$ such that $p_i \in I_i$, and the expected query complexity for all dimensions $i$ such that $p_i \notin I_i$: $d \cdot O(1) + \sum_{j \geq 0} O(\log d) \cdot \frac{1}{16^j} = O(d)$. \hfill \Box

Now we can describe the algorithm **IsEmpty**, which solves the **Bucket-emptiness** problem. The idea is simple: we iterate over every point in $Y$, calling a boosted version of the **InBucket** subroutine, since we look for a two-sided error algorithm.

**Algorithm IsEmpty** $(B = \prod_i I_i, Y = \{y_1, \ldots, y_m\}, \delta_1, \delta_2)$

*input:* $B$ bucket, $Y$ set of points, $\delta_1, \delta_2$ error probabilities

*output:* whether $Y \cap B = \emptyset$

*error:* incorrectly outputs true w.p. $\delta_1$ and incorrectly outputs false w.p. $\delta_2$.

\begin{algorithm}
\begin{algorithmic}
\STATE for each point $p \in Y$ do
\STATE \quad if BoostProb(InBucket($p, B$), $\delta_2/m, \delta_1$) then
\STATE \quad \quad Output false
\STATE \quad Output true
\end{algorithmic}
\end{algorithm}

*Proof of Lemma 2.5.* Suppose $B$ is empty. For all $m$ points $p \in Y$ we have $p \notin B$. By the Union bound, the probability that BoostProb(InBucket($p, B$), $\delta_2/m, \delta_1$) returns incorrectly true is at most $m \cdot \delta_2/m = \delta_2$. The query complexity is, by Proposition 2.4 and linearity of expectation, $mT(\text{InBucket}(p, B)) \log(1/\delta_1)$, which is $O(dm \log(1/\delta_1))$.

Now suppose that $B$ is non-empty. Let $p$ be a point of $Y$ such that $p \in B$. For that value of $p$, the probability that BoostProb(InBucket($p, B$), $\delta_2/m, \delta_1$) incorrectly returns false is at most $\delta_1$. The expected query complexity for this test is $O(d \log(m/\delta_2))$. If the algorithm observes correctly that $p \in B$, then it terminates without testing any further points. Thus, the expected total query complexity is given by two terms: the expected query complexity for all points $p$ such that $p \notin B$, and the expected query complexity for all points $p$ such that $p \in B$:

$$m \cdot d \log(1/\delta_1) + \sum_{j \geq 0} O(d \log(m/\delta_2)) \cdot \delta_1^j = O(dm \log(1/\delta_1)) + O(d \log(m/\delta_2)),$$

hence the claimed bounds. \hfill \Box

**B.3 Proof of Theorem 1.2**

*Proof of Theorem 4.3.* We first study the correctness of algorithm **SkylineHighDim**($k$, $X$, $\delta$), that is, with error probability at most $\delta$ the algorithm returns $\min\{k, |\text{skyline}(X)|\}$ skyline points. We will check that at every iteration $i \in \{1, \ldots, k\}$ the probability error is at most $\delta/k$, that is, the probability of not recovering a skyline point is at most $\delta/k$. Then, the overall error probability by the algorithm is at most $\delta$. Let $S_i$ be the skyline
points identified after the first \(i\) iterations. Here are the possible error sources for Algorithm \textbf{SkylineHighDim}(\(k, X, \delta\)) at iteration \(i\):

- **Phase (i) testing domination.** An skyline point is incorrectly certified as dominated in line 6, or a dominated point is certified to be non dominated by \textbf{SetDominates}. By Lemma B.2, this happens with probability error at most \(\delta/(4k)\) for the first source of error. For the second source of error this happens w.p. at most \(|X| \cdot \delta/(4k|X|) = \delta/2k\), by the union bound.

- **Phase (ii) computing lex-maximum.** The dominance test was performed correctly, but the \textbf{MaxLex} computation is incorrect in line 10: by Proposition 4.2 this happens with probability at most \(\delta/2k\).

Overall, the probability that all iterations are correct is at least \(1 - k(\delta/2k + \delta/2k) = \delta\), by the union bound.

We now consider the query complexity of \textbf{SkylineHighDim}(\(k, X, \delta\)). Observe that every point in \(p \in X\) will consider at most twice In any call of \textbf{SetDominates}(\(S_i, p, \cdot, \cdot\)). Thus, by Lemma B.2, the expected query complexity is

\[
O(|S_i|d \log(4k|X|/\delta)) = O(dk \log(k|X|/\delta))
\]

per found skyline point and \(O(d \log(|S_i|k/\delta))\) for each non-skyline point. This gives a total expected query complexity of \(O(k^2d \log(k|X|/\delta)) + O(|X|d \log(k/\delta))\) as claimed. This finishes the proof.

**Proof of Theorem 1.2.** First consider the error probability of \textbf{SkylineHighDim}(\(k_i, X, \delta/8^i\)). This error probability is bounded by \(\delta/2^i\) for the \(i\)'th iteration. Thus, by the union bound, the probability that for no \(k_i\) an error occurred is at most \(\delta\).

We now consider the query complexity. Let \(O_i\) be the set of the output of Algorithm \textbf{SkylineHighDim}(\(k_i, X, \delta/8^i\)). Recall that \textbf{SkylineHighDim}(\(X, \delta\)) terminates as soon as \(|O_i| \neq k_i\). Whenever \(k_i > |\text{skyline}(X)|\) such an output is returned w.p. at least \(1 - \delta_i\). Let \(Z\) denote the variable denoting the number of ‘extra’ increase due to failure, i.e., \(k_i > |\text{skyline}(X)|\), but \(|O_i| \neq k_i\). Set \(\gamma = \log(|\text{skyline}(X)|)\) and recall that \(\log(\cdot) = \log_2(\cdot)\).

Since \(k_i = 2^i\), the number of iterations is bounded by \(\lceil \gamma \rceil + 1 + Z\). Therefore, the query complexity of \textbf{SkylineHighDim}(\(k_i, X, \delta/8^i\)) is bounded by \(O(ndk_i \cdot \log(2^i k_i/\delta)) + O(k_i^2 d \log(k_i n/\delta))\), which in turns implies that the total expected query complexity is bounded by

\[
\sum_{i=1}^{\lceil \gamma \rceil + 1 + Z} O(ndk_i \cdot \log(8^i k_i/\delta) + k_i^2 d \log(k_i n/\delta)).
\]
Let $k = |\text{skyline}(X)|$. We first show the following statement

$$\sum_{i=1}^{[\gamma]+1+Z} k_i \log (k_i/\delta) \leq O(2^Z Z \cdot k \log (k/\delta)).$$

We have

$$\sum_{i=1}^{[\gamma]+1+Z} 2^i \log (k_i^4/k_i^4/\delta) \leq \sum_{i=1}^{[\gamma]+1+Z} 2^i \log (k_i^4/k_i^4+1+Z/\delta) \leq \sum_{i=1}^{[\gamma]+1+Z} 2^i \leq O(2^Z \cdot k \log (k \cdot 2^Z/\delta)).$$

Note that $\mathbb{P}(Z = i) \leq \delta^i$ and $\delta \leq 1/8$. Therefore, the total expected number of queries is thus

$$O \left( \sum_{i=0}^{\infty} 2^Z Z \cdot k \log (k/\delta) \delta^i \right) = O(k \log (k/\delta)).$$

Similarly, we can show the following statement

$$\sum_{i=1}^{[\gamma]+1+Z} k_i^2 \log (k_i n/\delta) \leq O(4^Z Z \cdot k^2 \log (kn/\delta)).$$

Using $\mathbb{P}(Z = i) \leq \delta^i$ and $\delta \leq 1/8$ we get

$$\sum_{i=0}^{\infty} O(4^Z Z \cdot k^2 \log (kn/\delta)) \cdot \mathbb{P}(Z = i) = O(k^2 \log (kn/\delta)).$$

Overall we obtain the number of queries claimed in the theorem. \qed

### B.4 Subroutines analysis for SkylineHighDim

The algorithm **SetDominates** is based on a two-sided algorithm for the basic problem **Domination**: the input is two points $p$ and $q$ and two error parameters $\delta_1$ and $\delta_2$. The goal is to check whether $p$ dominates $q$. The following algorithm solves this problem.

With that at hand, the algorithm **SetDominates** runs over every point in $S$, and breaks with **true** as soon as it finds a point dominating $q$. 

---

23
Algorithm Dominates\((p, q)\)
\begin{itemize}
  \item \textbf{input:} points \(p, q\) and two error parameters \(\delta_1\) and \(\delta_2\)
  \item \textbf{output:} whether \(p\) dominates \(q\)
  \item \textbf{error probability:} 1/16
\end{itemize}

\begin{algorithmic}
  \FOR each dimension \(i\)
    \IF \textbf{BoostProb}(\(p_i < q_i, 1/(16d), 1/16\))
      \OUTPUT false
    \ELSE\OUTPUT true
  \ENDIF
\ENDFOR
\end{algorithmic}

Algorithm SetDominates\((S, q, \delta_1, \delta_2)\)
\begin{itemize}
  \item \textbf{input:} Set of points \(S\), a point \(q\), and two error parameters \(\delta_1\) and \(\delta_2\)
  \item \textbf{output:} whether there exists a \(p \in S\) that dominates \(q\)
  \item \textbf{error:} incorrectly outputs \textbf{true} w.p. \(\delta_1\) and incorrectly outputs \textbf{false} w.p. \(\delta_2\).
\end{itemize}

\begin{algorithmic}
  \FOR each \(p \in S\)
    \IF \textbf{BoostProb}(\textbf{Dominates}(p, q), \delta_1/|S|, \delta_2)\ (\text{i.e., test if } p \geq q)\)
      \OUTPUT true
    \ELSE \OUTPUT false
  \ENDIF
\ENDFOR
\end{algorithmic}

In the following lemma we study the algorithm \textbf{Dominates}. The proof of Lemma 4.1 follows directly, from the \textbf{if} line and the application of union bound.

\textbf{Lemma B.2.} The algorithm \textbf{Dominates}\((p, q)\) is correct w.r.t. to its specification and has expected query complexity \(O(d)\).

\textit{Proof.} Suppose \(p\) dominates \(q\). Thus in all dimensions \(i\) we have \(p_i \geq q_i\). Hence, for any dimensions, the probability that \textbf{BoostProb}(\(p_i < q_i, 1/(16d), 1/16\)) returns incorrectly \textbf{true} is at most 1/(16d). Taking the union bound over all \(d\) dimensions yields the desired bound on the probability of incorrectly returning \textbf{false}. The query complexity is, by Proposition 2.4 (query complexity of \textbf{BoostProb}) and linearity of expectation,

\[ d \cdot O(1) = O(d). \]

Now suppose that \(p\) does not dominate \(q\). The query complexity for dimension \(i\) with \(p_i \geq q_i\) is due to the fact that the query complexity of \textbf{BoostProb} is, in expectation, \(O(\log(1/16)) = O(1)\). Since \(p\) does not dominate \(q\), there exists a dimensions \(i\) such that \(p_i < q_i\). The expected query complexity is, by Proposition 2.4, \(O(\log(16d))\). If the algorithm observes correctly that \(p_i < q_i\), then it terminates without testing any further dimensions. The probability that \textbf{BoostProb}(\(p_i < q_i, 1/(16d), 1/16\)) returns incorrectly \textbf{false} is at most 1/16. Thus, the case that the algorithm does not observes that \(p_i < q_i\), happens only w.p.
and hence the expected total query complexity is given by

\[
d \cdot O(1) + \sum_{j=0}^{\left|\{i \in \{1,2,...,d\} : p_i < q_i\}\right|-1} O(\log d) \cdot \frac{1}{16^j} = O(d).
\]

Algorithm \text{Lex}(p,q)

\textbf{input:} points \(p,q\)
\textbf{output:} whether \(p >_{\text{lex}} q\)
\textbf{error probability:} 1/16

\begin{algorithmic}
\State \For {each dimension \(i\)}
\State \If {BoostProb\((p_i >_i q_i, 1/(32d), 1/32)\) then}
\State Output true
\Else
\State \If {BoostProb\((q_i >_i p_i, 1/(32d), 1/32)\) then}
\State Output false
\EndIf
\EndIf
\EndFor
\end{algorithmic}

Algorithm \text{MaxLex}(p,S,\delta)

\textbf{input:} Point \(p\), set \(S\) containing \(p\), error probability \(\delta\)
\textbf{output:} the point \(p^* \in S\) which has the maximum lexicographic order among those that dominate \(p\)
\textbf{error probability:} \(\delta\)

\begin{algorithmic}
\State For all \(q \in S\), \(c(q) \leftarrow \log(1/\delta)\)
\Repeat
\State Let \(q_1 \leftarrow \arg\max_q \in S c(q)\);
\State Let \(q_2 \leftarrow \arg\max_{q \in S \setminus \{q_1\}} c(q)\)
\State \If {Lex\((q_1, q_2)\) (i.e., \(q_1 >_{\text{lex}} q_2\) then}
\State \(x \leftarrow q_1, y \leftarrow q_2\)
\Else
\State \(x \leftarrow q_2, y \leftarrow q_1\)
\State \(c(y) \leftarrow c(y) - 1\)
\EndIf
\State \If {Dominates\((x, p)\) (i.e., \(x \geq p\) then}
\State \(c(x) \leftarrow c(x) + 1/2\)
\Else
\State \(c(x) \leftarrow c(x) - 1\)
\EndIf
\Until {\(c(q_2) \leq -2\)}
\State Output \(\arg\max_{q \in S} c(q)\).
\end{algorithmic}
Observation 3. Consider MaxLex. There exists a iteration $t \leq 10(|S|+3)(\log(1/\delta)+3)$ (of the repeat loop) in which all but first counter are smaller than $-1$, in symbols, if $q \neq \arg \max_{p \in S} c(p)$, then $c(q) \leq -1$.

Proof. Let $\tau = 10(|S|+3)(\log(1/\delta)+3)$. First observe that after any iteration all but the largest and second largest counter have a value which differs by at most 1. Furthermore, the sum of the counters at iteration $t'$ is at most $|S| \log(1/\delta) - t'/2$ implying that after $2\tau/5 = 4(|S|+3)(\log(1/\delta)+3)$ iterations all but the largest $c(q_1)$ and second largest counter $c(q_2)$ must be smaller than $-2$. At this iteration $2\tau/5$, the sum of the first two counters is at most $c(q_1) + c(q_2) \leq 2\log(1/\delta) + \tau/5$. Thus after further $3\tau/5$ iterations, at least one of the counters $c(q_1)$ or $c(q_2)$ must have been $-1$. Thus there exists an iteration where all but one counter are $-1$ or smaller.

Proof of Lemma 4.2. Let $q^*$ be the maximum in lexicographic order in $S$ that is not dominated by $p$. By Observation 3, there is an iteration where all but the first counter are smaller than $-2$. Thus, it suffices to show that, with probability error at most $\delta$, it holds that $c(q^*) > -1$, implying that $q^*$ must be the point returned by the algorithm.

Let $X_t$ denote the value of the counter $c(q^*)$ at the time where $q^*$ took part in $t$ comparisons and we will say that $X_t$ is the counter of $q^*$ after $t$ time steps (the time steps are thus the number of comparisons $q^*$ took part in). Let $Y_t$ denote the value of the counter $c(q^*)$ after point $q^*$ took part of $t$ rounds, which are defined as follows. Each round consists out of one or two time-steps. A round ends when either $X_t$ increased two times or decreased at least once. Hence, given the outcome of all time-steps one can group them into rounds. Furthermore, let $\tau(t)$ be the time-step at which round $t$ ends. Let $E_t$ be the event that $X_{\tau(t)+1} = X_{\tau(t)} + 1/2$. Note that $\neg E_t$ implies $X_{\tau(t)+1} = X_{\tau(t)} - 1$. Moreover, define $Y_t = X_{\tau(t)}$ and

$$Y_{t+1} = \begin{cases} X_{\tau(t)} - 1 & \text{if } \neg E_t, \\ X_{\tau(t)} - 1/2 & \text{if } E_t \text{ and } \neg E_{t+1} \\ X_{\tau(t)} + 1 & \text{if } E_t \text{ and } E_{t+1} \end{cases}$$

Let $T = \min\{t \geq 0 : X_t \in \{0, 20(|S|+3)(\log(1/\delta)+3)\}\}$. We will show that for all $t \in \mathbb{Z}$ we have $Y_t > 0$ which implies that for all $t \in \mathbb{Z}$ we have $X_t > 0 - 1 = -1$ since every round consists of at most two time-steps. Consider the random walk $(Z_t)_{t \geq 0}$ with state space $[0, b]$ in Proposition A.1, where $b = 2\tau = 20(|S|+3)(\log(1/\delta)+3)$. Let $Z_0 = Y_0 = X_0 = \log(1/\delta)$. Conditioning on $F_t$ and $Y_t = i - 1$ (the latter implying that $X_{\tau(t)} = i - 1$) we have for $i - 1 \in [1, b - 1]$

$$\mathbb{P}(Y_{t+1} = i) = \mathbb{P}(E_{t+1} | E_t) \cdot \mathbb{P}(E_t) \geq (1 - 2/12) \cdot (1 - 2/12) \geq 2/3 = \mathbb{P}(Z_t = i | Z_{t-1} = i - 1),$$

where we used the union bound. Thus

$$\mathbb{P}(Y_{t+1} = i | Y_t = i - 1) \geq \mathbb{P}(Z_{t+1} = i | Z_t = i - 1).$$
By Proposition A.1 parameters \( p = \frac{10}{12}, s = \log(1/\delta) \), \( b \) we have that

\[
P(Z_T = 0) = \frac{\left(\frac{1-p}{p}\right)^b - \left(\frac{1-p}{p}\right)^s}{\left(\frac{1-p}{p}\right)^b - 1} \leq 2^{-\log(1/\delta)} = \delta.
\]

Thus, w.p. \( 1 - \delta \) we have \((Z_t) > 0\). And hence

\[
P(q^* \text{ is not returned}) \leq P(X_T \leq -1) \leq P(Y_T \leq 0) \leq P(Z_0 \leq 0) = P(Z_T = 0) \leq \delta,
\]

which proves the correctness w.r.t. to the specification of MaxLex. By Lemma B.2, executing Lex and Dominates requires \( O(d \log(1/\delta^2) + \log(d/\delta_1)) = O(d) \) queries in expectation. Furthermore, by Observation 3, the total number of iterations is bounded by \(10(|S| + 3)(\log(1/\delta) + 3)\). This finishes the proof.

B.5 Missing Proofs Section 5

Proof of Lemma 5.1. The proof is by contradiction. Assume that \( A \) is an algorithm with success probability at least 3/4 and worst case number of queries \( T \leq (\ell k \log_3 k)/1000 \). We assume that the adversary is generous, i.e. the adversary tells the truth for every entry \((i, j)\) such that \( v_{ij} = 0 \), and that lies with probability \( 1/3 \) otherwise.

Generalizing the 2-phase computational model by Feige, Peleg, Raghavan and Upfal [FRPU94], we will give the algorithm more leeway and study a 4-phase computation model, defined as follows. In the first phase, the algorithm queries every entry \( v_{ij} \) \( (\log_3 k)/100 \) times. In the second phase, the adversary reveals to the algorithm all remaining hidden entries \((i, j)\) such that \( v_{ij} = 2 \), except for a single random one. In the third phase, the algorithm can strategically and adaptively choose \( k\ell/10 \) entries, and the adversary reveals their true value at no additional cost. Finally, in phase 4, the algorithm outputs \( w_i = 2 \) for every vector where it found an entry equal to 2, and \( w_i = 0 \) for the rest of the vectors.

To see how the two models are related, observe that since \( T \leq (\ell k \log_3 k)/20 \), by Markov’s inequality at most a set \( S \) of \( \ell k/10 \) entries are queried by algorithm \( A \) more than \( (\log_3 k)/2 \) times, so at the end of the first phase we have queried every entry at least as many times as \( A \), except for those \( \ell k/10 \) entries, and in the beginning of the third phase there is all the necessary information to simulate the execution of \( A \), adaptively finding \( S \) (and getting those values correctly), hence the success probability of the three-phase algorithm is greater than or equal to the success probability of \( A \). Also observe that, thanks to the definition of \( \mu \) and to the generosity of the adversary, any execution where all queries to a vector lead to 0 answers must lead to an output where \( w_i = 0 \)—else the algorithm would be incorrect when \( \mu \) selects the null vector.

We now sketch the analysis of the success probability of the three-phase algorithm. Due to the definition of \( \mu \), with probability at least 9/10 the ground-truth input drawn
from $\mu^k$ has $k/2 \pm O(\sqrt{k})$ vectors that contain an entry equal to 2. At the end of the first phase, and due to the fact that the adversary is generous, we have that at most of them have been identified. There remain $k/2 \pm O(\sqrt{k})$ vectors that appear to be all zeroes, and about $(k/2)(1/3)^{(\log_3 k)/2} = (1/2)\sqrt{k}$ of those vectors contain a still-hidden entry whose true value is 2. During the third phase, all of those hidden 2’s are revealed except for one. At that point, there still remain $k/2 \pm O(\sqrt{k})$ vectors whose entries appear to be all zeroes, there is a 2 hidden somewhere uniformly at random, but all entries have been queried an equal number of times, all in vain. To find that remaining hidden entry (and therefore decide which $w_j$ is equal to 2), the algorithm has no information to distinguish between the $\ell(k/2 \pm O(\sqrt{k}))$ remaining entries. Since, the algorithm may only select $\ell k/10$ elements to query further, the algorithm’s success probability after the fourth phase cannot be better than $(k\ell/10)/(\ell(k/2 \pm O(\sqrt{k}))) < 1/4$, a contradiction. 

\section{The NoisySearch algorithm}

\textbf{input:} a point $y$, and an ordered list $(y_1, y_2, \ldots, y_{m-1})$, accessible by comparisons that each have error probability at most $p$, and a parameter $\delta$

\textbf{output:} an interval $I = (y_{i-1}, y_i)$ or $[y_i]$

\textbf{error probability:} $\delta$

Here, we recall the algorithm from [FRPU94] whose performance is given in Theorem 2.2. It is best described by first rephrasing standard binary search to search for $y$ in an ordered set $Y = (y_1, y_2, \ldots, y_n)$

The binary search algorithm can be viewed as a downwards walk from the root in a search tree whose nodes represent intervals. Letting $y_0$ denote $-\infty$ and $y_n$ denote $+\infty$, each tree node represents an interval $[y_i, y_j]$, the root represents $(y_0, y_n]$, each leaf represents $(y_{i-1}, y_i]$ for some $i$, and the left and right child of the node representing $(y_i, y_j]$ represent $(y_i, y_k]$ and $(y_k, y_j]$ respectively, with $k = (i + j)/2$. When the walk is at a node representing $(y_i, y_j]$, if the node has two children $(y_i, y_k]$ and $(y_k, y_j]$ then the algorithm compares $y$ to $y_k$ and proceeds to the left or right child according to the result of the comparison. The algorithm stops when a leaf is reached, after $\log_2 n$ steps of the downwards walk, and returns the interval of the current node.

The noisy search algorithm can be viewed as a biased random walk from the root in a search tree that extends the noiseless search tree, each leaf $(y_{i-1}, y_i]$ being the parent of an infinite chain, each of whose nodes are also labeled $(y_{i-1}, y_i]$. When the walk is at a node representing $(y_i, y_j]$, the noisy search algorithm first performs two comparisons to check whether $y_i < y \leq y_j$. If the answer is negative, then the walk moves to the parent node; if the answer is positive, then, if the node has only one child then the walk proceeds to the child node, and if the node has two children $(y_i, y_k]$ and $(y_k, y_j]$ then the algorithm compares $y$ to $y_k$ and proceeds to the left or right child according to the result of the comparison. The algorithm stops after $c \log n/\delta$ steps of the random walk and returns the interval of the
current node.

D The NoisySort algorithm

Here, we recall the algorithm from [FRPU94] whose performance is given in Theorem 2.3.

| Algorithm D.1 NoisySort(Y, i, δ) [FRPU94] |
|---------------------------------------------|
| **input:** Set of points Y = (y₁, y₂, ..., yₙ), coordinate i, δ error prob. |
| **output:** Sorted list of i'th coordinates of points in Y |
| **error probability:** δ |
| 1: Z ← ∅ |
| 2: for ℓ = 1 to n do |
| 3: (zₗ₋₁, zₗ) ← NoisySearch(yₗ,i, Z, δ/n) |
| 4: Update list Z by inserting yₗ between zₗ₋₁ and zₗ in Z |
| 5: Output Z. |

E The SkyGM algorithm

The idea of SkyGM is to sort the points correctly on each dimension and to use any skyline algorithm in the noiseless setting to deduce the skyline.

**Theorem E.1** ([GM15, Theorem 3]). The algorithm SkyGM is correct w.r.t. to its specification and has an expected query complexity of $O(dn \log(dn/δ))$.

| Algorithm SkyGM(X, δ) ([GM15, Algorithm 3]) |
|-----------------------------------------------|
| **input:** X set of points, δ error probability |
| **output:** skyline(X) |
| **error probability:** δ. |
| 1: for dimension i ∈ {1, 2, ..., d} do |
| 2: $S_i$ ← NoisySort(X, i, δ/d) |
| 3: Deduce skyline(X) from $S_1, S_2, ..., S_d$ and output it. |