Chapter 2

Skyline Points (Maximal Points)

2.1 Introduction

Let P be a set of n points in a d-dimensional space. A point $p(p_1, \ldots, p_d)$ *dominates* a point $q(q_1, \ldots, q_d)$ if $p_i > q_i, \forall i \in [1, d]$. In a *skyline query*, the goal is to report points of P which are not dominated by any other points in P. In the computational geometry literature, skyline points are also known as *maximal or minimal* points. See Figure 2.1 for an example in the context of cricket (a popular sport in India). In cricket, for a batsman, higher the average (resp., strike rate), the better he/she is considered. In Figure 2.1, each point corresponds to a batsman with the x-coordinate (resp., y-coordinate) being the average (resp., strike rate). The skyline points represent *potentially* good batsmen.

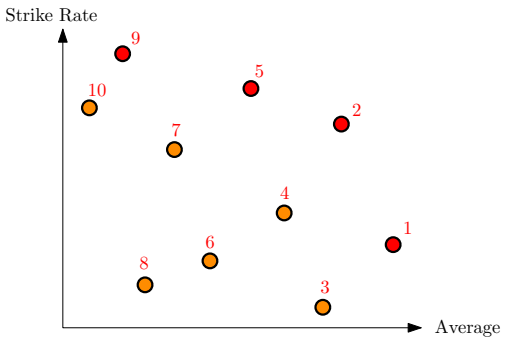

Figure 2.1: Points $1, 2, 5$ and 9 are skyline points.

It is an extremely popular *summarization* query in databases, since it filters out potentially useful points (other applications include finding "best" products or hotels based on some d parameters). As a result, there has been an extensive amount of work done on computing skyline points and its several variants in the database community.

2.2 Warm-up: Algorithm in 2D

A naive approach to compute the skyline points would be to compare each pair of points which would take $O(n^2)$ time. Instead, we can solve the problem in $O(n \log n)$ time.

Theorem 1. *Given a set P of n points in 2D, its skyline points can be computed in $O(n \log n)$ time.*

Let $p_1, \ldots, p_n$ be the sequence of points of P in decreasing order of their x-coordinate values. Initialize a variable $y_{max} \leftarrow -\infty$. The algorithm will scan the points in the order $p_1, p_2, \ldots, p_n$. When a point p_i is scanned, if the y-coordinate of p_i is greater than y_{max}, then p_i is declared as a skyline point and y_{max} is updated to the y-coordinate of p_i; otherwise, p_i is declared as a non-skyline point and y_{max} remains the same. See Figure 2.2 (a) and (b) for an illustration of these two cases. The running time is dominated by the sorting step which takes $O(n \log n)$ time.

Question. Prove the correctness of this algorithm.

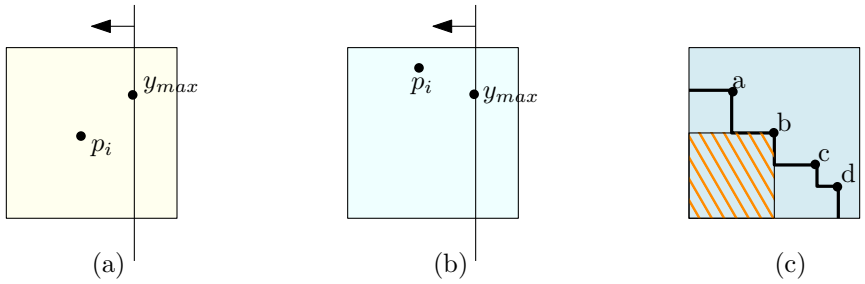

Figure 2.2: (a) p_i is a non-skyline point, (b) p_i is a skyline point, and (c) a 2D skyline can be visualized as a staircase.

It will be useful to visualize the 2D skyline as a "staircase" represented as a collection of horizontal and vertical segments shown in bold in Figure 2.2(c). Two useful properties of the staircase are the following:

1. If a point $p \notin P$ lies "below" the staircase, then p is dominated by one of the skyline points. On the other hand, if a point $p \notin P$ lies "above" the staircase, then p is not dominated by any of the skyline points.

2. As we walk from left-to-right on the staircase, the y-coordinates of the skyline points visited keeps decreasing.

2.3 Sweepline algorithm in 3D

In this section a *sweep-plane* based algorithm for computing skyline points in 3D will be discussed. Let $p_1, p_2, \ldots, p_n$ be the sequence of points in P in decreasing order of their z-coordinate values. Imagine a plane parallel to the xy-plane starting from $z = +\infty$ and sweeping towards $z = -\infty$. As the sweep proceeds, the plane will intersect points in P (in decreasing order of their z-coordinate values). Let S_i be the skyline points found by the algorithm after processing $p_1, \ldots, p_i$. Note that p_1 will always be a skyline point of P, and hence, initialize $S_1 \leftarrow \{p_1\}$. Let S_i^{xy} be the skyline points of S_i where the domination is defined w.r.t. x and y coordinates only (i.e., the z coordinate is ignored).

Question. Come up with a small example where S_i is not equal to S_i^{xy}.

When the sweep-plane reaches a point p_i, then the algorithm performs the following test:

Does any point in S_{i-1}^{xy} dominate p_i?

If the answer is yes (Figure 2.3(b)), then the algorithm declares that p_i is not a skyline point of P. Update $S_i \leftarrow S_{i-1}$ and $S_i^{xy} \leftarrow S_{i-1}^{xy}$. On the other hand, if the answer is no (Figure 2.3(a)), then the algorithm declares that p_i is a skyline point of P. Update $S_i \leftarrow S_{i-1} \cup \{p_i\}$ and S_i^{xy} is obtained from S_{i-1}^{xy} by adding p_i and removing points of S_{i-1}^{xy} dominated by p_i.

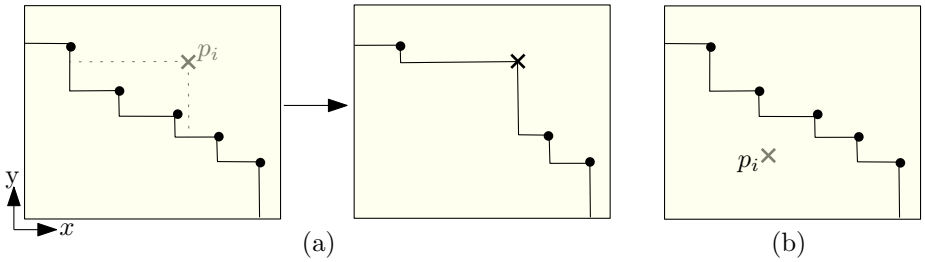

Figure 2.3: S_{i-1}^{xy} is the set of black points. (a) No point in S_{i-1}^{xy} dominates p_i, (b) p_i is dominated by a point in S_{i-1}^{xy}.

The final output of the algorithm will be S_n. Answering the following questions proves the correctness of the algorithm.

Question. While testing if p_i is a skyline point or not, why did the algorithm not consider points $p_{i+1}, \ldots, p_n$?

Question. If a point in S_{i-1}^{xy} dominates p_i, then why is p_i not a skyline point of P?

Question. If no point in S_{i-1}^{xy} dominates p_i, then why is p_i a skyline point of P?

Efficient implementation. An efficient data structure is needed to ensure that the above algorithm runs in $O(n \log n)$ time. Even though the question of "Does any point in S_{i-1}^{xy} dominate p_i" looks like a 2D question, it is actually a 1D problem. We will build a binary search tree based on the points in S_{i-1}^{xy}. We will store the points at the leaf nodes ordered from left to right in increasing order of their x-coordinate values. By Property (2) of 2D skyline, this also implies that the leaf nodes are ordered in decreasing order of their y-coordinate values. Additionally, the leaf nodes are connected as a doubly linked list, so for any leaf node its predecessor and successor leaf node can be accessed in constant time.

Recall from previous section that a 2D skyline can be conceptually represented as a staircase. To answer "Does any point in S_{i-1}^{xy} dominate p_i" is equivalent to answering whether p_i lies "above" the staircase or "below" the staircase. We will perform a successor query on the binary search tree with the x-coordinate of p_i, and let p_j be the point corresponding to it.

- If p_j dominates p_i w.r.t. x and y coordinates, then p_i will lie below the staircase.
- If p_j does not dominate p_i w.r.t. x and y coordinates, then p_i will lie above the staircase (since none of the points on the staircase to the right of p_j can dominate p_i).

If p_i lies above the staircase, then we also need to update the staricase by inserting p_i and removing the points on the staircase dominated by p_i. The points on the updated staircase will be the set S_i^{xy}.

Question. Assume that k_i points on the staircase are dominated by p_i. Show that these k_i points can be identified and deleted from the binary search tree in $O((k_i+1)\log n)$ amortized time. You can assume that deletion and insertion of a point in the balanced binary search tree takes $O(\log n)$ amortized time.

Once a point gets deleted from the staircase (and the binary search tree), then it can never re-appear. Using this crucial observation, the total time taken to update the staircase can be bounded by

$$\sum_{i=1}^{n} O((k_i+1)\log n) = O\left(\log n\left(\sum_{i=1}^{n} k_i + \sum_{i=1}^{n} 1\right)\right) = O(n\log n), \text{ since } \sum_{i=1}^{n} k_i \leq n.$$

Before the sweep-plane starts, the algorithm sort the points based on their z-coordinate values which takes $O(n\log n)$ time. For each point visited, it takes $O(\log n)$ amortized time to answer a successor query. Finally, we showed above that updating the staircase (and hence, S_i^{xy}) through out the algorithm takes $O(n\log n)$ time. Therefore, the overall running time of the algorithm is $O(n\log n)$.

Theorem 2. *Given a set P of n points in 3D, its skyline points can be computed in $O(n\log n)$ time.*

2.4 Output-sensitive algorithm in 2D

In the previous sections, we have seen an $O(n\log n)$ time algorithms to solve the skyline problem in 2D and 3D. In the *comparison model* of computation, a lower bound of $\Omega(n\log n)$ can be shown for the skyline problem. Therefore, the $O(n\log n)$ time algorithms are optimal in the comparison model.

The goal of this section is to design algorithms which will perform better than $O(n\log n)$ when the size of the skyline is "small", and when the size of the skyline is "large", then the running time remains $O(n\log n)$. Such algorithms are known as *output-sensitive* algorithms since their running time is proportional to the size of the output. If the algorithm designer is told that $k=1$ for the given input, then it is trivial to design an $O(n)$ time algorithm. This hints that it might be possible to beat the $n\log n$ barrier when k is small.

Slow algorithm. Let P be a set of n points in 2D and let k be the number of skyline points in P. Consider the following simple algorithm. Let p_{max} be the point in P with the largest x-coordinate, and let $D(p_{max}) \subset P$ be the points dominated by p_{max}. Let $P \leftarrow P \setminus (\{p_{max} \cup D(p_{max})\})$. Repeat the above step recursively on P. The algorithm stops when P becomes empty.

Exercise. Argue that the above algorithm reports the k skyline points correctly. Also, prove that the running time of the algorithm is $O(nk)$.

2.4.1 Chan's algorithm.

Now we will describe an interesting $O(n\log k)$ time algorithm designed by Timothy M. Chan (originally for the convex hull problem). Let us first dive right into the details of the algorithm. Later we will recap the key properties used in the algorithms.

Sub-problem. Given P and integer parameter $\hat{k}$, design an $O(n\log\hat{k})$ time algorithm which does the following:

- If $\hat{k} \geq k$, then report the skyline points of P,

- else if $\hat{k} < k$, then report *failure*.

The algorithm consists of the following three steps (See Figure 2.4):

1. Construct $\hat{k}+1$ vertical lines $\ell_1, \ell_2, \ldots, \ell_{\hat{k}+1}$, such that
   - $+\infty = x(\ell_1) > x(\ell_2) > \ldots > x(\ell_{\hat{k}+1}) = -\infty$, where $x(\ell_i)$ is the x-coordinate of ℓ_i, and
   - for all $1 \leq i \leq \hat{k}$, the number of points of P whose x-coordinate lies between $x(\ell_i)$ and $x(\ell_{i+1})$ is $n/\hat{k}$.

   For all $1 \leq i \leq \hat{k}$, define $P_i \subseteq P$ to be the points lying in $[x(\ell_{i+1}), x(\ell_i)]$. Also, initialize $S \leftarrow \emptyset$ and $y(p^*) \leftarrow -\infty$.
2. For $i = 1$ till $\hat{k}$,
   - if $|S| \geq \hat{k}+1$, then report *failure*.
   - Otherwise, let $P_i' \subseteq P_i$ be the points with y-coordinate greater than $y(p^*)$. Run the "slow algorithm" on P_i'. Let $S(P_i)$ be the skyline points reported. Set $S \leftarrow S \cup S(P_i)$. Define $y(p^*)$ to be the y-coordinate of the point in S with the largest y-coordinate.
3. Report S.

Exercise. Prove that the above algorithm solves the sub-problem correctly.

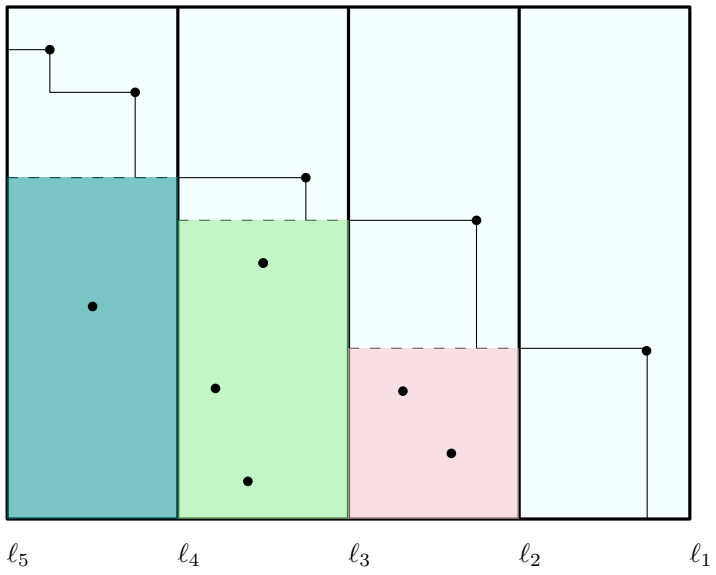

Figure 2.4: Execution of the algorithm for $\hat{k} = 4$. In this example, it will report a *failure*.

Exercise. Construct the $\hat{k}+1$ vertical lines in $O(n \log \hat{k})$ time. (*Hint: recursively use the linear-time median finding algorithm.*)

Step (1) of the algorithm can be performed in $O(n \log \hat{k})$ time. Running the slow algorithm on $|P_i'|$ takes $O((n/\hat{k}) \times |S(P_i)|)$ time. Summing over all the iterations, it takes $O((n/\hat{k}) \sum_{i=1}^{\hat{k}} |S(P_i)|) = O((n/\hat{k}) \cdot \hat{k}) = O(n)$ time. Therefore, the overall running time of the sub-problem is $O(n \log \hat{k})$.

Guessing $\hat{k}$. Now comes the mysterious part where we need to *guess* the value of k. A poor guess would be to set $\hat{k}$ to $1, 2, 3 \ldots$ until $\hat{k} \leftarrow k$, and run the sub-problem for each guess. This would lead to an overall running time of $O(n \sum_{i=1}^{k} \log i) = O(nk \log k)$. Another poor choice would be to do binary search: here the initial guess of k would be $n/2$, which implies that running time of the sub-problem on $\hat{k} \leftarrow n/2$ will be $\Omega(n \log n)$, whereas it could happen that $k \ll n/2$.

A decent choice would be the *doubling trick*, in which the algorithm sets $\hat{k} \leftarrow 2^i$ in the i-th iteration ($i \geq 0$). Then the running time of the algorithm will be:

$$\sum_{i=0}^{\lceil \log k \rceil} O(n \log 2^i) = O(n \sum_{i=0}^{\lceil \log k \rceil} i) = O(n \log^2 k).$$

The cleverness of Chan's algorithm lies in guessing $\hat{k} \leftarrow 2^{2^i}$ in the i-th iteration. Then the running time of the algorithm will be:

$$\sum_{i=0}^{\lceil \log \log k \rceil} O(n \log 2^{2^i}) = O(n \sum_{i=0}^{\lceil \log \log k \rceil} 2^i) = O(n \log k).$$

Exercise. Prove that a more *aggressive* choice of guessing $\hat{k} \leftarrow 2^{2^{2^i}}$ will not help! Specifically, prove that the running time will be $O(n \log^2 k)$.

Comments. We conclude by making a few observations about the algorithm. Notice that unlike the $O(n \log n)$ time algorithms, here the algorithm cannot afford to sort the input points. Step (1) of the algorithm tackles this issue by doing "partial sorting". For a given P_i, the points within P_i are not sorted among themselves. However, given $i < j$, all the points in P_i have a larger x-coordinate value than any point in P_j. Secondly, the algorithm nicely combines the sweepline approach of the $O(n \log n)$ time algorithm with the "slow algorithm". Unlike the traditional sweepline algorithms, in this algorithm, the sweepline sweeps $n/\hat{k}$ points at a time and runs the slow algorithm on the points swept at that time. Thirdly, this algorithm inherently requires knowing the value of k. However, we do not know it upfront. Chan's guessing technique hits the "sweet spot": it leads to a geometric series, where the largest term dominates the summation of the remaining terms.

2.5 Exercises

1 (Divide and conquer) We will try to come up with a divide and conquer algorithm to compute the skyline points in 3D. Let P be a set of n points in 3D. The algorithm computes a median plane ($x = x_{med}$) such that there are equal number of points of P on either side of the median plane; call these sets P_ℓ and P_r, respectively. Recursively, compute the skyline points of P_ℓ and P_r.

- Let S_ℓ and S_r be the skyline points of P_ℓ and P_r, respectively. Design a *merge* procedure which takes as input S_ℓ and S_r, and in $O(n \log n)$ time computes the skyline points of $S_\ell \cup S_r$.
- Prove that the set of skyline points of $S_\ell \cup S_r$ is equal to the set of skyline points of P.
- What is the overall running time of such an algorithm?

2. Now improve the running time of the above algorithm to $O(n \log n)$. Hint 1: Merge operation should happen in $O(n)$ time. Hint 2: Before the algorithm starts, pre-sort P based on their z-coordinate values. Hint 3: Maintain the invariant that the skyline points of any subproblem (in the divide and conquer algorithm) are *reported* in sorted order based on their z-coordinate values.

3 (Output sensitive algorithm-I) Let P be a set of n points lying in a d-dimensional space, where d is a constant independent of n such as 5 or 10. Design an *output-sensitive* algorithm for computing skyline points of P in $O(nh)$ time, where h is the number of skyline points.

4 (Output sensitive algorithm-II) Consider the following algorithm to compute the skyline points in 2D.

skyline(P):
1. if $|P| = 1$, then return P.
2. divide P into the left and right halves P_ℓ and P_r by the median x-coordinate.
3. *discover* the point p with the maximum y-coordinate in P_r.
4. *prune* all points from P_ℓ that are dominated by p.
5. return the concatenation of skyline(P_ℓ) and skyline(P_r).

Prove that the above algorithm runs in $O(n + n \log h)$ time, where h is the number of skyline points in P. (*Hint: Define $T(n, h)$ to be the running time of the algorithm on an input of size n and output size h. Write a recurrence in terms of $T(n, h)$ and establish an important base case of $T(n, 1) \leq cn$, where c is a constant independent of n.*)

5 (Output sensitive algorithm-III) In this problem, the goal is to design a fast *output-sensitive* algorithm in 3D. Specifically, the goal is to design an $O(n \log k)$ time algorithm, where k is the size of the output. To solve this problem, first read this article(`https://www.cse.cuhk.edu.hk/~taoyf/course/5010/notes/maxima.pdf`) which presents an alternate method to design an $O(n \log k)$ time algorithm in 2D. The idea of guessing the output size is extremely clever! Next, adapt this algorithm for 3D.

6 (Skyline in $\mathbb{R}^d$) Let $T(n, d)$ be the time taken to compute skyline points in $\mathbb{R}^d$. Then desgin an $O(T(n, d) \cdot \log n)$ time algorithm to compute skyline points in $\mathbb{R}^{d+1}$. (*Hints: One approach is via range trees.*)

7 (ε-skyline in 2d) In many instances the number of skyline points reported could be very large which can make it hard for the user to get an "informative summary" of the underlying data. Therefore, in some scenarios, it would be ideal to guarantee a small output size to the user. This motivates the study of *ε-skyline queries*. Here we will relax the notion of domination and tolerate a buffer in each dimension which checking for domination.

Formally, let P be lying inside a unit cube $[0, 1]^d$. Now a point p *ε-dominates* another point q if $p_i + \varepsilon > q_i, \forall i \in [1, d]$, where ε is a fixed real number in the range $(0, 1)$. The ε-skyline is a set of points in P such that they ε-dominate all the points in P.

- Construct an example where the size of ε-skyline almost $1/\varepsilon$.
- Construct an example where the size of the skyline is large, but the size of ε-skyline is $O(1)$ (independent of n and ε).

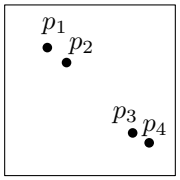

Figure 2.5: An input consisting of four points $p_1 = (0.1, 0.9)$, $p_2 = (0.15, 0.81)$, $p_3 = (0.8, 0.3)$, and $p_4 = (0.85, 0.21)$. The value of $\varepsilon = 0.1$. The skyline would consist of $\{p_1, p_2, p_3, p_4\}$. However, one possible ε-skyline would be $\{p_1, p_3\}$. The other possible ε-skyline would be $\{p_2, p_4\}$.

- Prove that in 2D one can construct an ε-skyline of size $O(1/\varepsilon)$.
- Describe an algorithm to compute the ε-skyline in $O(n \log \frac{1}{\varepsilon})$ time.

2.6 Open problems

(ε-skyline in higher dimensions) The size of the ε-skyline in $\mathbb{R}^d (d \geq 3)$ is not known. Is the size of the ε-skyline $O(1/\varepsilon)$ in any $\mathbb{R}^d$? Assume d is a constant and the dependency on d is hidden in the big-Oh. Or, is there an input instance on which it can be shown that the size of any ε-skyline is $\omega(1/\varepsilon)$?

TODO: Afshani's zig-zag sweep?, Instance-optimal algorithms in 2d and 3d, Open problems.