Introduction to Skylines

Yufei Tao

Department of Computer Science and Technology
Chinese University of Hong Kong

Definition (Monotonic function)

Let p be a d-dimensional point in $\mathbb{R}^d$. Let $f : \mathbb{R}^d \to \mathbb{R}$ a function that calculates a score $f(p)$ for p. We say that f is *monotonic* if the score never decreases when any coordinate of p increases.

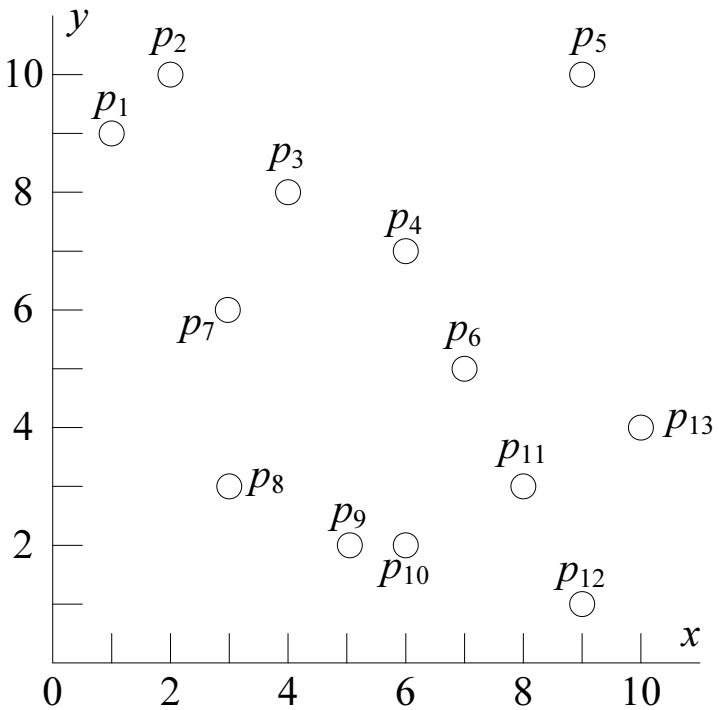
For example, $f(x, y) = x + y$ is monotonic but $f(x, y) = x - y$ is not.

Definition (Top-1 search)

Let P be a set of d-dimensional points in $\mathbb{R}^d$. Given a monotonic function f, a *top-1 query* finds the point in P that has the smallest score.

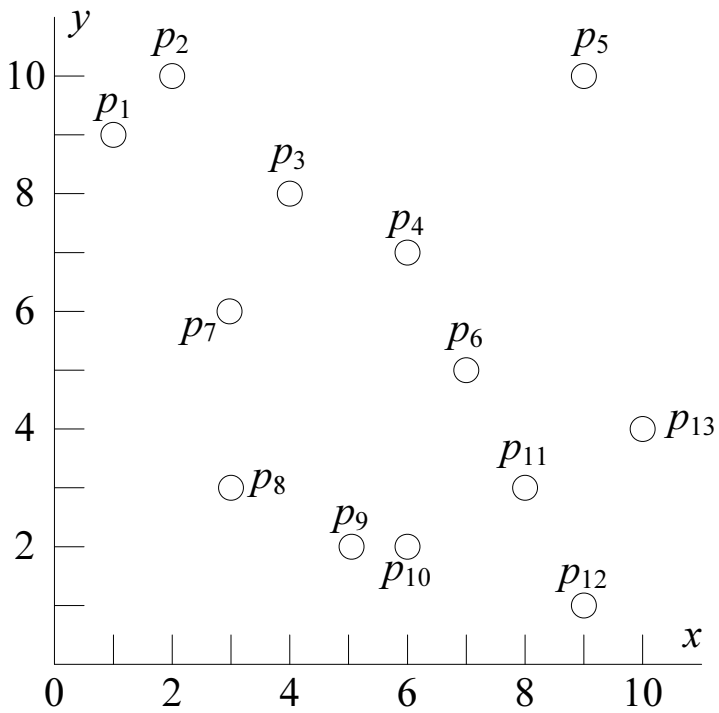
Example

If $f(x, y) = x + y$, then the top-1 is p_8.



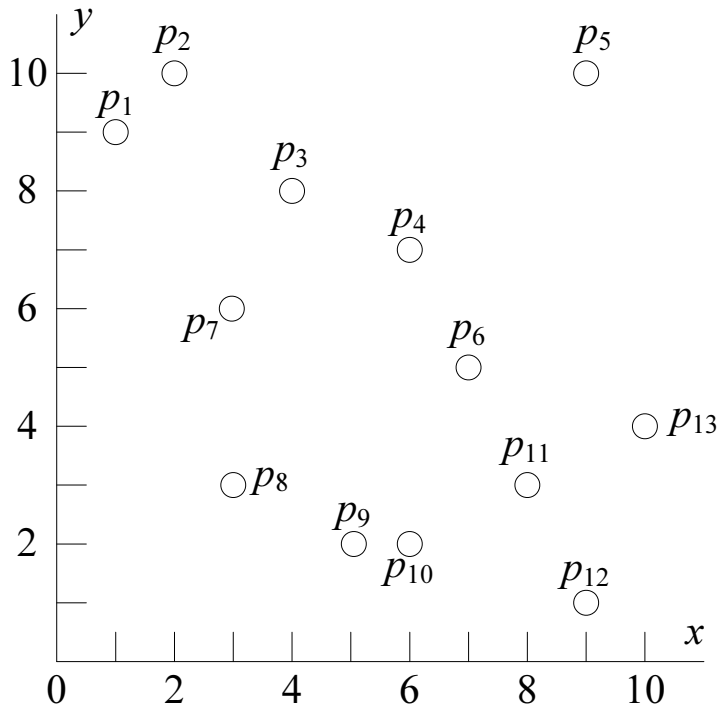
Applications

- Find the best NBA player according to *point* + *rebound*.
- Find the best hotel according to *price* + *distance* (say to the town center).
- Find the best laptop according to $-$*CPU-speed* $-$*memory* $-$*disk-volume* + *price*.



Drawback of top-1 search

In general, it is difficult to decide which distance function f should be used. For example, assume that the x-dimension corresponds to the price of a hotel and the y-dimension to its distance to the town center. Why is $f(x, y) = x + y$ a good function to use? Why not $2x + y$, or something more complex like $\sqrt{x} + y^2$?



The skyline operator remedies the drawback of top-1 search with an interesting idea. Instead of reporting only 1 object, the operator reports a set of objects that are guaranteed to cover the result of any top-1 query (i.e., regardless of the query function, as long as it is monotonic!).

Definition (Dominance)

A point p_1 *dominates* p_2 if the coordinate of p_1 is smaller than or equal to p_2 in all dimensions, and strictly smaller in one dimension.

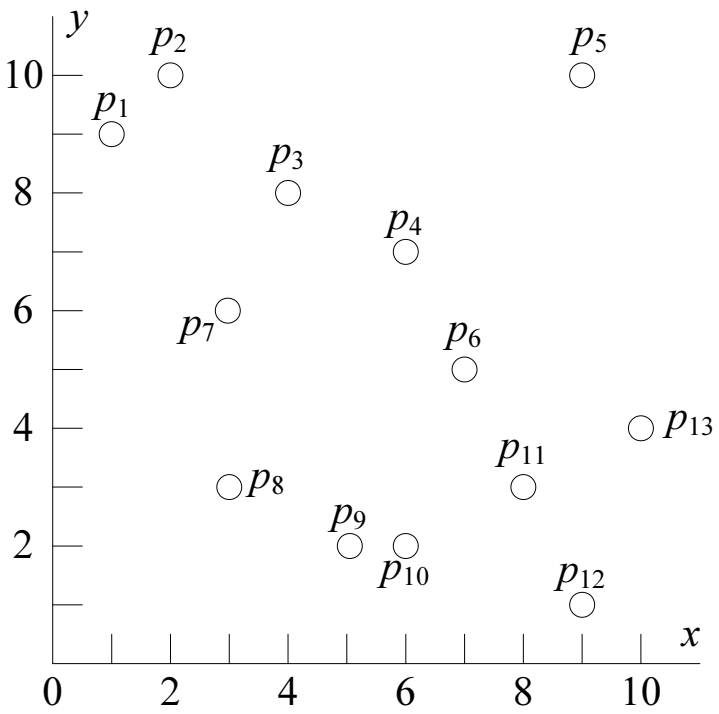
Note that p_1 has a smaller score than p_2 with respect to all monotonic function.

Definition (Skyline)

Let P be a set of d-dimensional points in $\mathbb{R}^d$ such that no two points coincide with each other. The *skyline* of P contains all the points that are not dominated by others.

The skyline is also known as *pareto set*.

The skyline is $\{p_1, p_8, p_9, p_{12}\}$.



Real example (by courtesy of Donghui Zhang)

name	point	rebound	assist	steal
Tracy McGrady	2003	484	448	135
Kobe Bryant	1819	392	398	86
Shaquille O'Neal	1669	760	200	36
Ming Yao	1465	669	61	34
Dwyane Wade	1854	397	520	121
Steve Nash	1165	249	861	74

Real example

4D skyline (negate all numbers to be consistent with our earlier definitions):

name	point	rebound	assist	steal
Tracy McGrady	2003	484	448	135
Kobe Bryant	1819	392	398	86
Shaquille O'Neal	1669	760	200	36
Ming Yao	1465	669	61	34
Dwyane Wade	1854	397	520	121
Steve Nash	1165	249	861	74

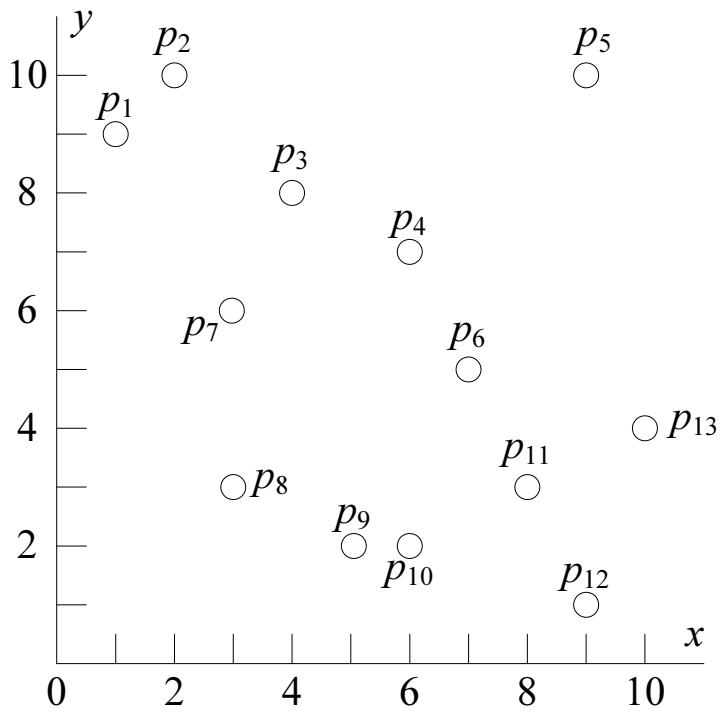
Real example

2D skyline by focusing on the first 2 attributes

name	point	rebound	~~assist~~	~~steal~~
Tracy McGrady	2003	484	~~448~~	~~135~~
Kobe Bryant	1819	392	~~398~~	~~86~~
Shaquille O'Neal	1669	760	~~200~~	~~36~~
Ming Yao	1465	669	~~61~~	~~34~~
Dwyane Wade	1854	397	~~520~~	~~121~~
Steve Nash	1165	249	~~861~~	~~74~~

Theorem

For any monotonic function, the top-1 point is definitely in the skyline. Conversely, every point in the skyline is definitely the top-1 of some monotonic function.



Next we discuss how to find the skyline of a set P of points efficiently.

Naive algorithm

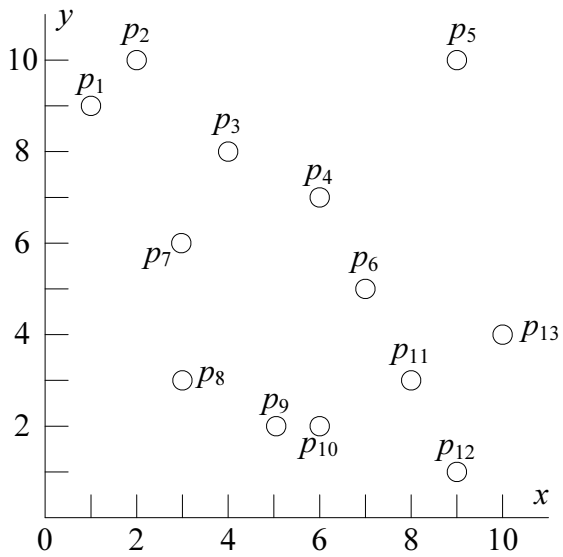
algorithm `naive`

1. $SKY \leftarrow \emptyset$
2. **for** each point $p \in P$
3. compare p to all other points in P
4. **if** p is not dominated by any other point
5. add p to the skyline set SKY
6. **return** SKY

Next, we will describe an algorithm called *sort first skyline (SFS)* that is fairly efficient on many practical datasets. It works in any dimensionality. The algorithm, however, is heuristic in nature (i.e., it does not have an attractable worst-case performance bound).

SFS example

We will use the following dataset to illustrate the algorithm.

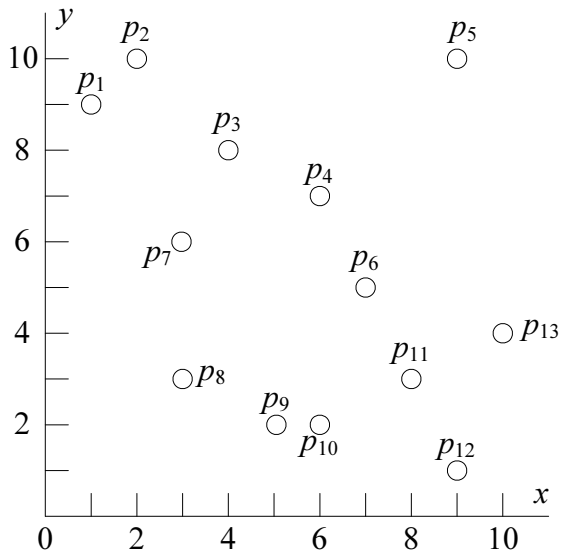


SFS example (cont.)

First, sort all the points according to an arbitrary monotonic function, e.g., $f(x, y) = x + y$. In case of a tie, the point with a smaller x-coordinate goes first.

Note that a point can only be dominated by points that rank before it.

$P = \{(p_8, 6), (p_9, 7), (p_{10}, 8),$
$(p_1, 9), (p_7, 9), (p_{12}, 10), (p_{11}, 11),$
$(p_2, 12), (p_3, 12), (p_6, 12), (p_4, 13),$
$(p_{13}, 14), (p_5, 19)\}$

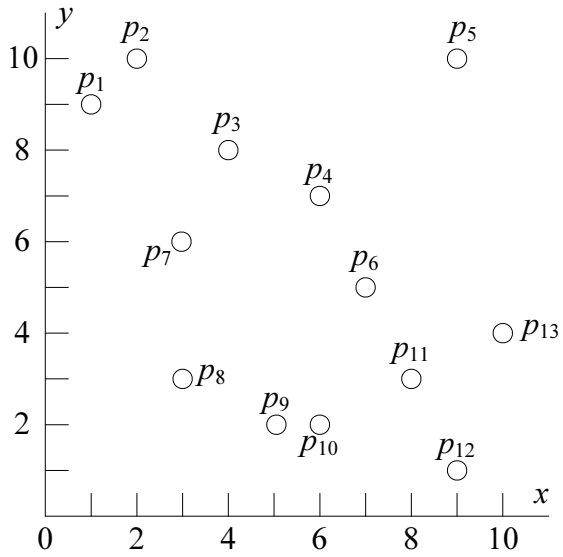


SFS example (cont.)

p_8 is definitely in the skyline.

$P = \{(p_8, 6), (p_9, 7), (p_{10}, 8), (p_1, 9), (p_7, 9), (p_{12}, 10), (p_{11}, 11), (p_2, 12), (p_3, 12), (p_6, 12), (p_4, 13), (p_{13}, 14), (p_5, 19)\}$

$SKY = \{p_8\}$



SFS example (cont.)

Next, we scan the rest of the points in the sorted order. For each point p:

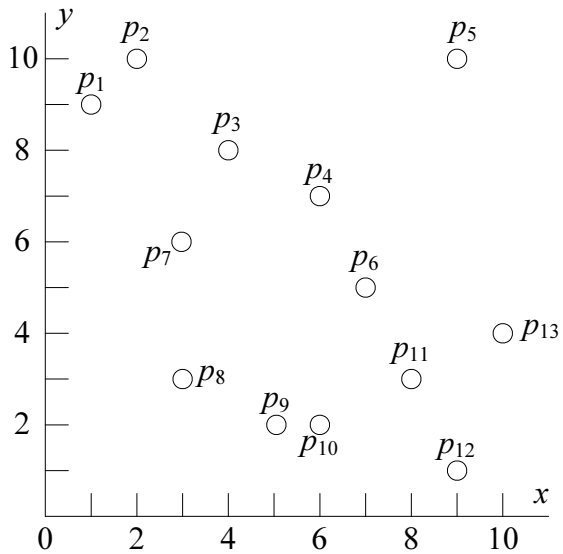
- compare it only to the points in SKY;
- if it is not dominated by any point in SKY, add it to SKY.

SFS example (cont.)

p_9 is not dominated by p_8 (i.e., the only point in SKY). Hence, it is added to SKY.

$P = \{(p_8, 6), (p_9, 7), (p_{10}, 8), (p_1, 9), (p_7, 9), (p_{12}, 10), (p_{11}, 11), (p_2, 12), (p_3, 12), (p_6, 12), (p_4, 13), (p_{13}, 14), (p_5, 19)\}$

$SKY = \{p_8, p_9\}$

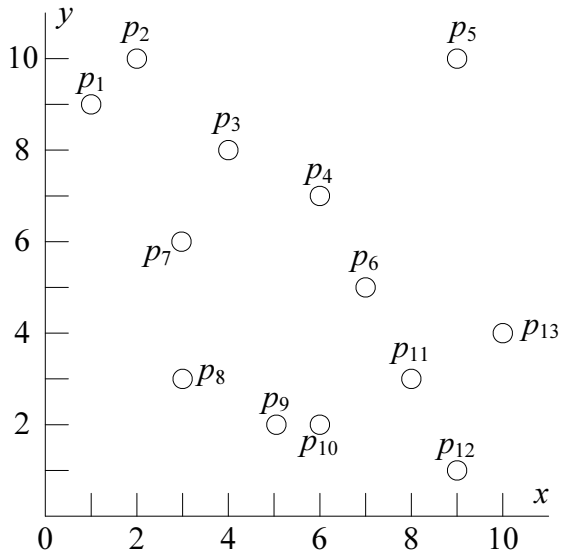


SFS example (cont.)

p_{10} is dominated by p_9, and is therefore discarded.

$P = \{(p_8, 6), (p_9, 7), (p_{10}, 8), (p_1, 9), (p_7, 9), (p_{12}, 10), (p_{11}, 11), (p_2, 12), (p_3, 12), (p_6, 12), (p_4, 13), (p_{13}, 14), (p_5, 19)\}$

$SKY = \{p_8, p_9\}$

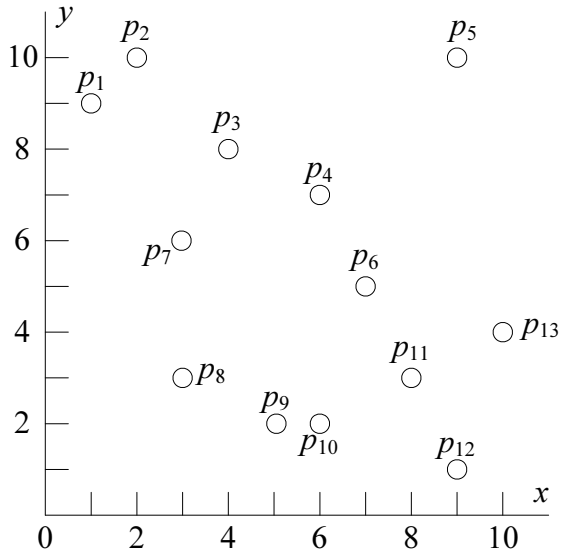


SFS example (cont.)

p_1 is added to SKY.

$P = \{(p_8, 6), (p_9, 7), (p_{10}, 8),$
$(p_1, 9), (p_7, 9), (p_{12}, 10), (p_{11}, 11),$
$(p_2, 12), (p_3, 12), (p_6, 12), (p_4, 13),$
$(p_{13}, 14), (p_5, 19)\}$

$SKY = \{p_8, p_9, p_1\}$

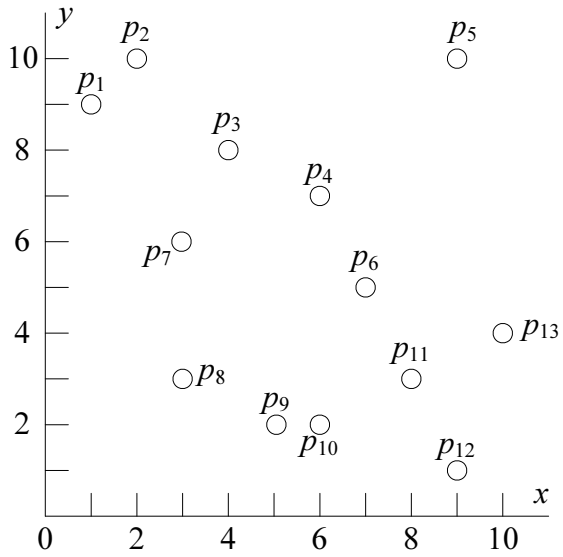


SFS example (cont.)

p_7 discarded and p_{12} added to SKY.

$P = \{(p_8, 6), (p_9, 7), (p_{10}, 8), (p_1, 9), (p_7, 9), (p_{12}, 10), (p_{11}, 11), (p_2, 12), (p_3, 12), (p_6, 12), (p_4, 13), (p_{13}, 14), (p_5, 19)\}$

$SKY = \{p_8, p_9, p_1, p_{12}\}$

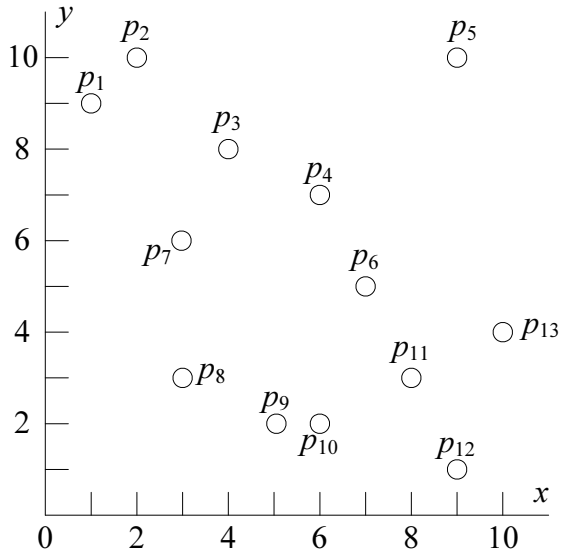


SFS example (cont.)

All other points discarded.

$P = \{(p_8, 6), (p_9, 7), (p_{10}, 8), (p_1, 9), (p_7, 9), (p_{12}, 10), (p_{11}, 11), (p_2, 12), (p_3, 12), (p_6, 12), (p_4, 13), (p_{13}, 14), (p_5, 19)\}$

$SKY = \{p_8, p_9, p_1, p_{12}\}$



SFS running time

- Let k be the number of points in the skyline.
- Each point is compared to at most k points.
- There are n points in total.
- Hence the total time is $O(nk)$.
- The initial sorting takes $O(n \lg n)$ time.

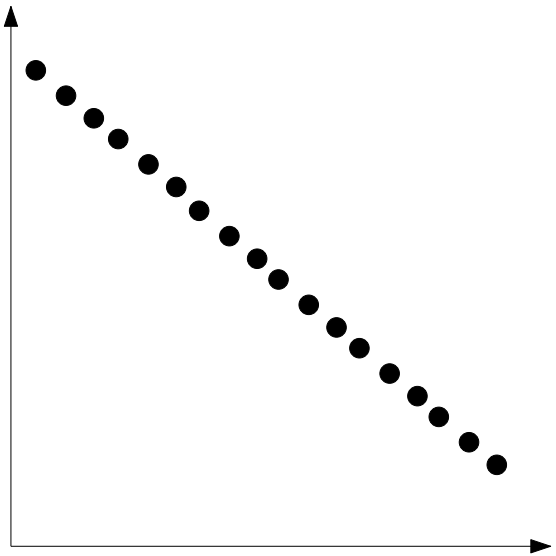
Total: $O(n \lg n + nk)$.

SFS running time

- From the previous slide: $O(n \lg n + nk)$.
- Efficient if k is small (true in practice when the dimensionality is low).
- How about the worst case?

SFS worst case

$k = n$



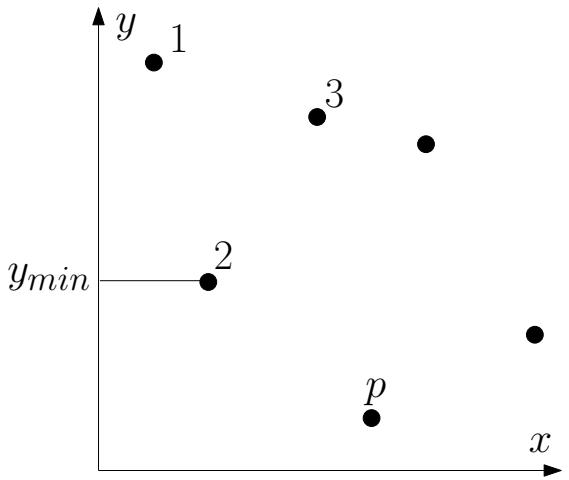
Running time $= O(n \lg n + n^2) = O(n^2)$.

Next, we will present two faster algorithms for solving the skyline problem in 2d and 3d, respectively. Both algorithm terminate in $O(n \log n)$ time.

Assume that P has been sorted in ascending order of their x-coordinates (which can be done in $O(n \log n)$ time). If two points have the same x-coordinate, rank the one with a smaller y-coordinate first. Consider any point $p \in P$. Let S be the set of points that rank before P. Observe:

- No point that ranks after p can possibly dominate p.
- Some point in S dominates p, if and only if the smallest y-coordinate of the points in S is no greater than the y-coordinate of p.

2-d (cont.)



Pseudocode of the 2-d algorithm

algorithm `2d-skyline`

1. sort the dataset P as described in Slide 29
2. $SKY = \emptyset, y_{min} = \infty$
2. **for** each point $p \in P$ in the sorted order
3. **if** the y-coordinate $p[y]$ of p is smaller than y_{min}
4. add p to SKY, and $y_{min} = p[y]$
5. **return** SKY

Line 1 takes $O(n \log n)$ time, whereas Lines 2-4 essentially scan the entire P only once in $O(n)$ time. Hence, the overall cost is $O(n \log n)$.

3-d

Again, sort P in ascending order of their x-coordinates. Break ties by putting the point with a smaller y-coordinate first, and if there is still a tie, the point with a smaller z-coordinate ranks first.

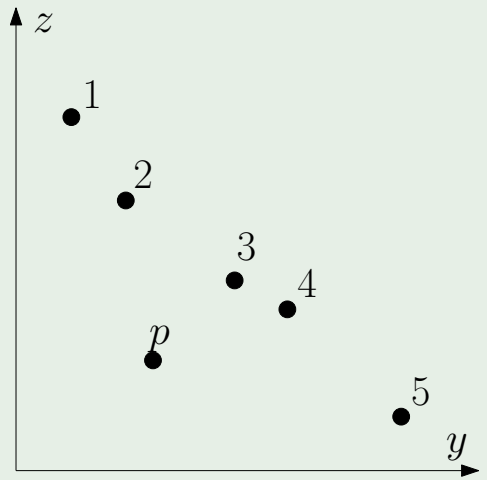
Consider any point $p \in P$. Let S be the set of points that rank before P. Observe:

- (Same as 2-d) no point that ranks *after* p can possibly dominate p.
- Let $SKY_{yz}(S)$ be the skyline of the projections of (the points of) S in the y-z plane. Some point in P dominates p in the x-y-z space, if and only if a point of $SKY_{yz}(S)$ dominates p in the y-z plane.

3-d (cont.)

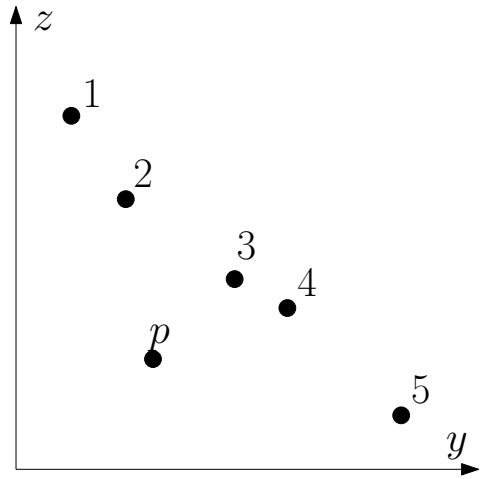
Example

Assume $SKY_{yz}(S)$ includes points 1, 2, 3, 4, 5. As no point of $SKY_{yz}(S)$ dominates p in the y-z plane, we can assert that p is definitely in the skyline (of the original space).



3-d (cont.)

$SKY_{yz}(S)$ is a 2-d skyline. In general, a 2-d skyline is a staircase. Namely, if we walk along the skyline towards the direction of ascending x-coordinates, the y-coordinates of the points keep decreasing.



3-d (cont.)

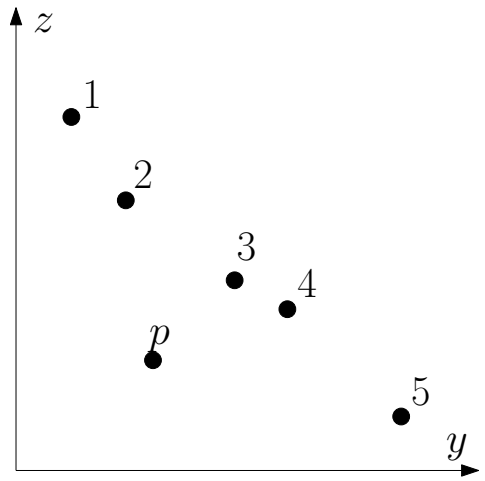
Let us index the points of $SKY_{yz}(S)$ by their y-coordinates using a binary tree (or a B-tree with a constant $B \geq 4$). Two operations can be done efficiently:

- Detect if a point p is dominated by any point in $SKY_{yz}(S)$ (in the y-z plane).
- Remove all points of $SKY_{yz}(S)$ dominated by p (in the y-z plane).

We will show that each detection can be done in $O(\log n)$ time, while removal in $O(k \log n)$ time, where k is the number of points removed.

3-d (cont.)

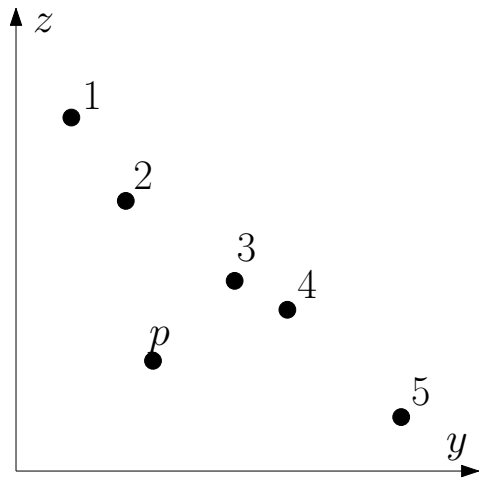
Detection is based on the observation that p is dominated by some point in $SKY_{yz}(S)$ if and only if p is dominated by the predecessor of p in $SKY_{yz}(S)$ on the y-dimension. For example, the predecessor is point 2 in the figure below.



Finding the predecessor takes $O(\log n)$ time using the binary tree.

3-d (cont.)

To remove the points of $SKY_{yz}(S)$ dominated by p (in the y-z plane), we first find the successor, say p', of p in $SKY_{yz}(S)$ on the y-dimension. If p dominates p', remove p' and set p' to its own successor. Repeat this until p no longer dominates p'. In the figure below, p' iterates through points 3 and 4.



Finding a successor and removing a point take $O(\log n)$ time.

Pseudocode of the 3-d algorithm

algorithm `3d-skyline`

1. sort the dataset P as described in Slide 32
2. $SKY = \emptyset$
3. let T be the binary tree as mentioned in Slide 35
4. **for** each point $p \in P$ in the sorted order
5. **if** p is not dominated by any point of T in the y-z plane **then**
6. add p to SKY
7. remove from T all points dominated by p in the y-z plane
8. **return** SKY

The detection at Line 5 is performed n times, and thus, requires $O(n \log n)$ time in total. On the other hand, each point is inserted and removed in T at most once. Hence, all the insertions and deletions entail $O(n \log n)$ time.

Remark

In general, the skyline problem can be settled in $O(n \log^{d-2} n)$ when the dimensionality d is at least 3. The algorithm for $d \geq 4$, however, is quite theoretical, and may not be as efficient as the heuristic algorithm SFS in practice.