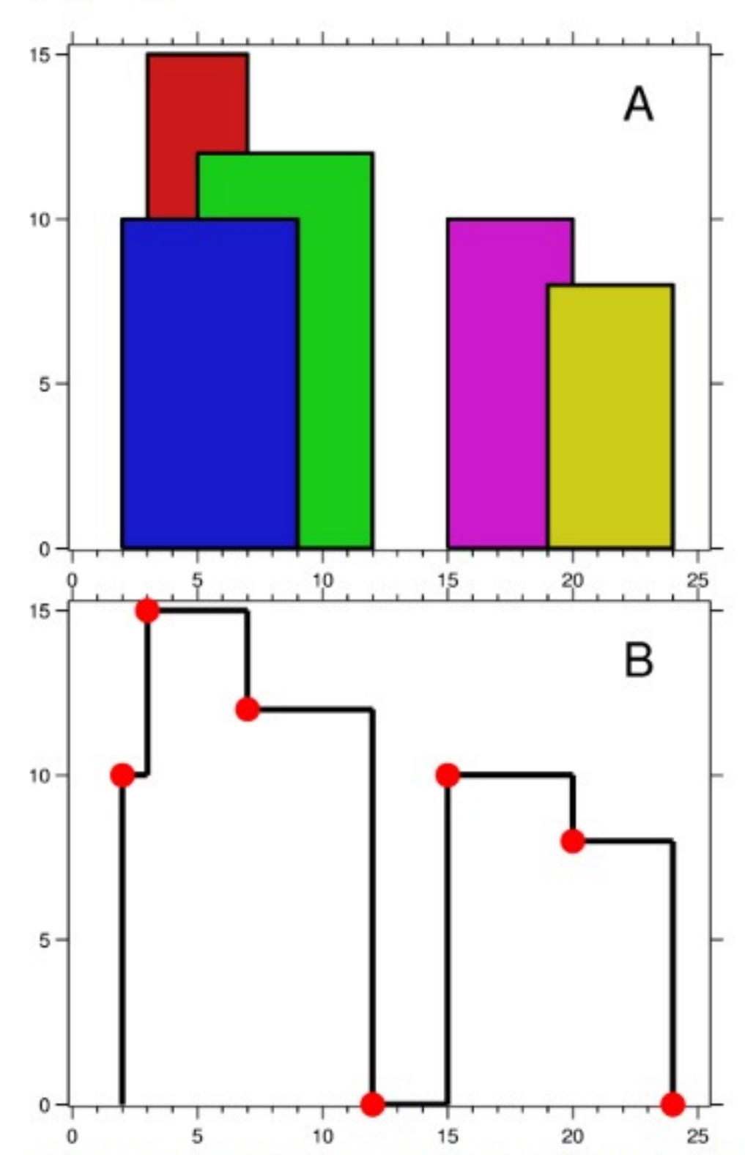
Previous Next

218. Skyline Problem

April 5, 2019 | 42.9K views

★★★★★
Average Rating: 4.18 (57 votes)

A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Now suppose you are **given the locations and height of all the buildings** as shown on a cityscape photo (Figure A), write a program to **output the skyline** formed by these buildings collectively (Figure B).



The geometric information of each building is represented by a triplet of integers `[Li, Ri, Hi]`, where `Li` and `Ri` are the x coordinates of the left and right edge of the ith building, respectively, and `Hi` is its height. It is guaranteed that `0 ≤ Li, Ri ≤ INT_MAX`, `0 < Hi ≤ INT_MAX`, and `Ri - Li > 0`. You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

For instance, the dimensions of all buildings in Figure A are recorded as: `[ [2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8] ]` .

The output is a list of "**key points**" (red dots in Figure B) in the format of `[ [x1,y1], [x2, y2], [x3, y3], ... ]` that uniquely defines a skyline. **A key point is the left endpoint of a horizontal line segment**. Note that the last key point, where the rightmost building ends, is merely used to mark the termination of the skyline, and always has zero height. Also, the ground in between any two adjacent buildings should be considered part of the skyline contour.

For instance, the skyline in Figure B should be represented as: `[ [2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0] ]`.

Notes:

- The number of buildings in any input list is guaranteed to be in the range `[0, 10000]`.
- The input list is already sorted in ascending order by the left x position `Li`.
- The output list must be sorted by the x position.
- There must be no consecutive horizontal lines of equal height in the output skyline. For instance, `[...[2 3], [4 5], [7 5], [11 5], [12 7]...]` is not acceptable; the three lines of height 5 should be merged into one in the final output as such: `[...[2 3], [4 5], [12 7], ...]`

Solution

Approach 1: Divide and Conquer

Solution template

The problem is a classical example of divide and conquer approach, and typically implemented exactly the same way as merge sort algorithm.

Let's follow here a solution template for divide and conquer problems :

- Define the base case(s).
- Split the problem into subproblems and solve them recursively.
- Merge the subproblems solutions into the problem solution.

Algorithm

getSkyline for `n` buildings :

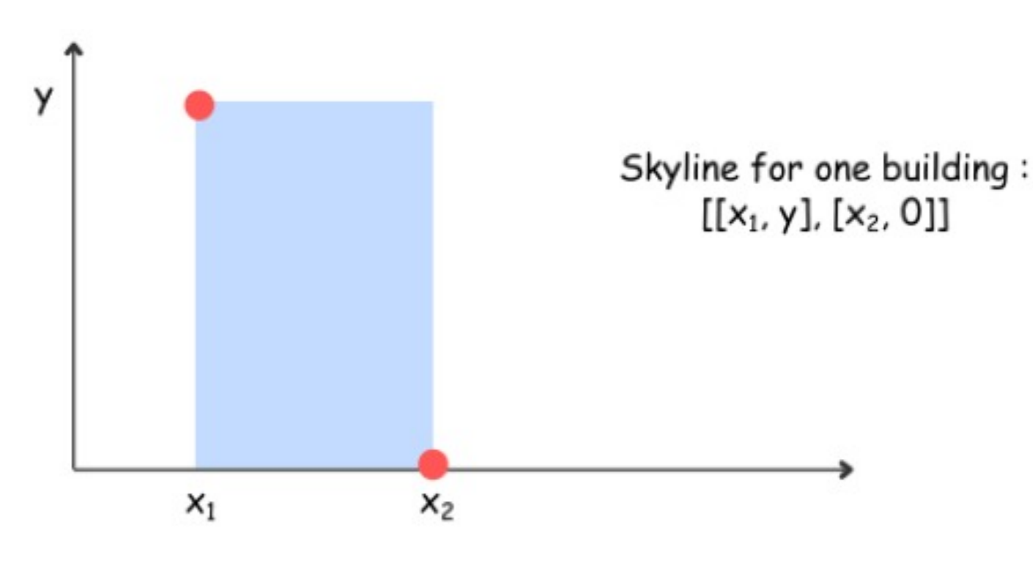
- If `n == 0` : return an empty list.
- If `n == 1` : return the skyline for one building (it's straightforward).
- `leftSkyline` = getSkyline for the first n/2 buildings.
- `rightSkyline` = getSkyline for the last n/2 buildings.
- Merge `leftSkyline` and `rightSkyline`.

Now let's discuss each step in more details.

Base cases

The first base case is an empty `buildings` list. Then the skyline is an empty list, too.

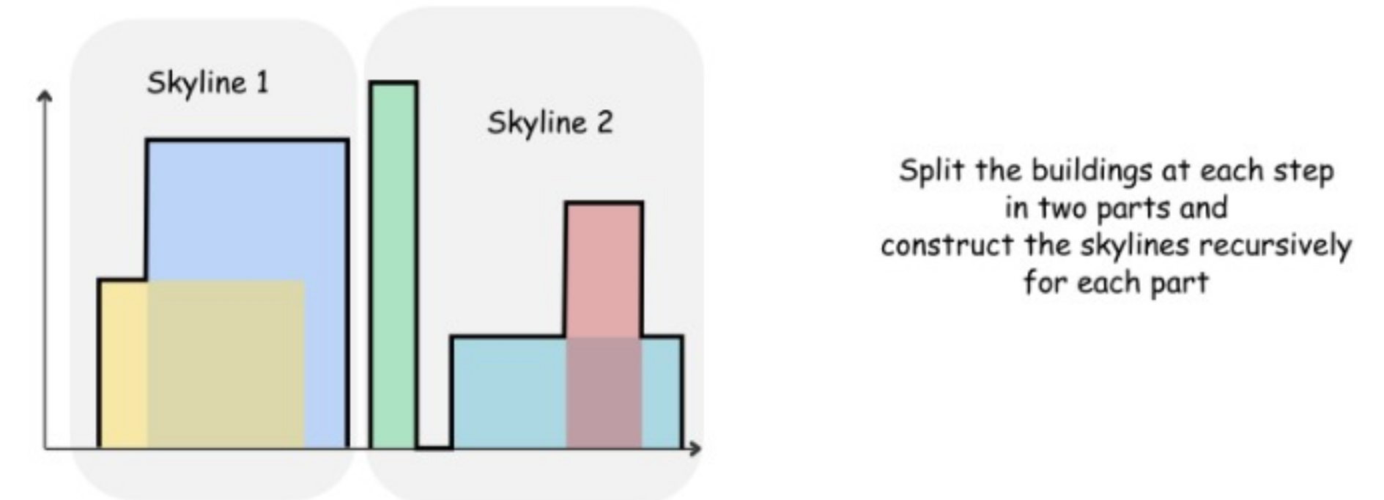
The second base case is the only one building in the list, when the skyline construction is quite straightforward.



Skyline for one building :
$[[x_1, y], [x_2, 0]]$

How to split the problem

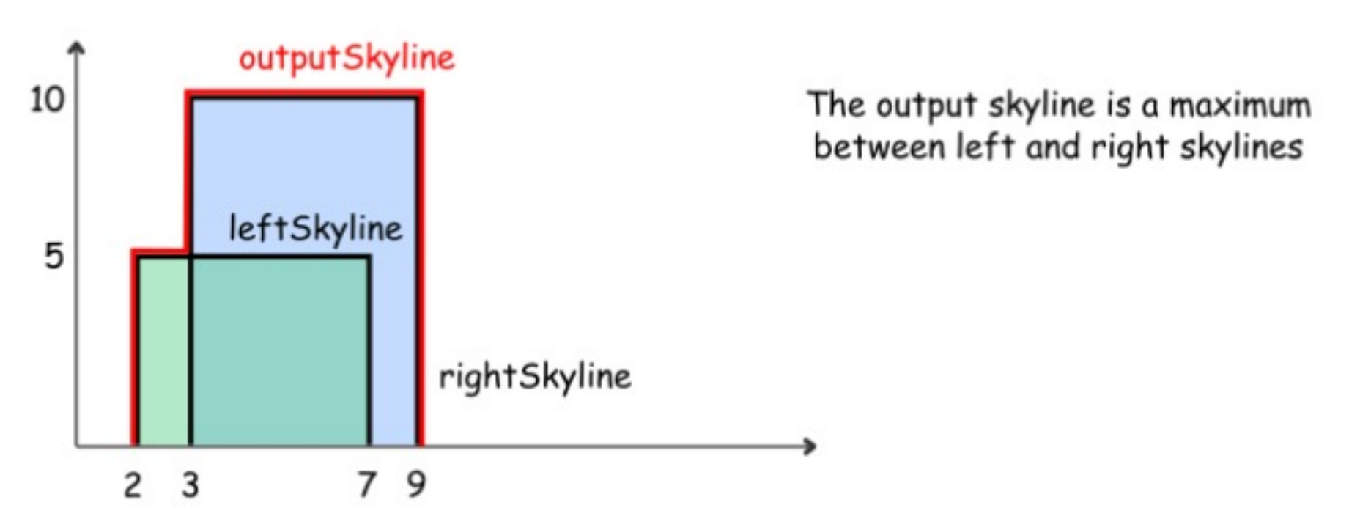
The idea is the same as for merge sort : at each step split the list exactly in two parts : from `0` to `n/2` and from `n/2` to `n`, and then construct the skylines recursively for each part.



Split the buildings at each step
in two parts and
construct the skylines recursively
for each part

How to merge two skylines

The algorithm for merge function is quite straightforward and based on the same merge sort logic : the height of an output skyline is always a maximum between the left and right skylines.



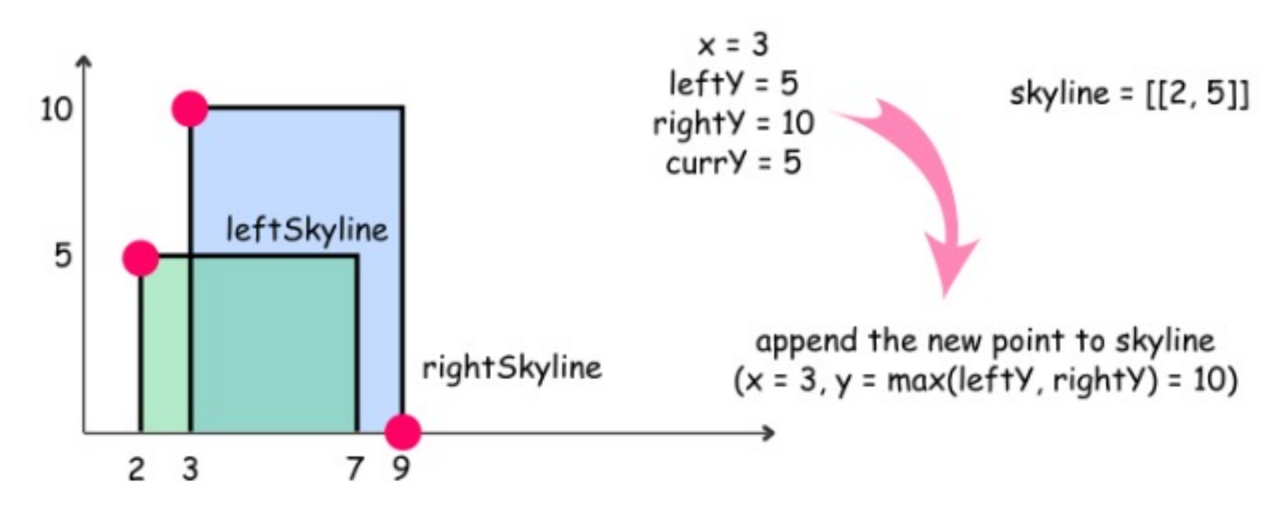
The output skyline is a maximum
between left and right skylines

Let's use here two pointers `pR` and `pL` to track the current element index in both skylines, and three integers `leftY`, `rightY`, and `currY` to track the current height for the `left` skyline, `right` skyline and the merged skyline.

mergeSkylines (left, right) :

- currY = leftY = rightY = 0
- While we're in the region where both skylines are present (`pR < nR` and `pL < nL`) :
  - Pick up the element with the smallest `x` coordinate. If it's an element from the left skyline, move `pL` and update `leftY`. If it's an element from the right skyline, move `pR` and update `rightY`.
  - Compute the largest height at the current point : `maxY = max(leftY, rightY)`.
  - Update an output skyline by `(x, maxY)` point, if `maxY` is not equal to `currY`.
- While there are still elements in the left skyline (`pL < nL`), process them following the same logic as above.
- While there are still elements in the right skyline (`pR < nR`), process them following the same logic as above.
- Return output skyline.

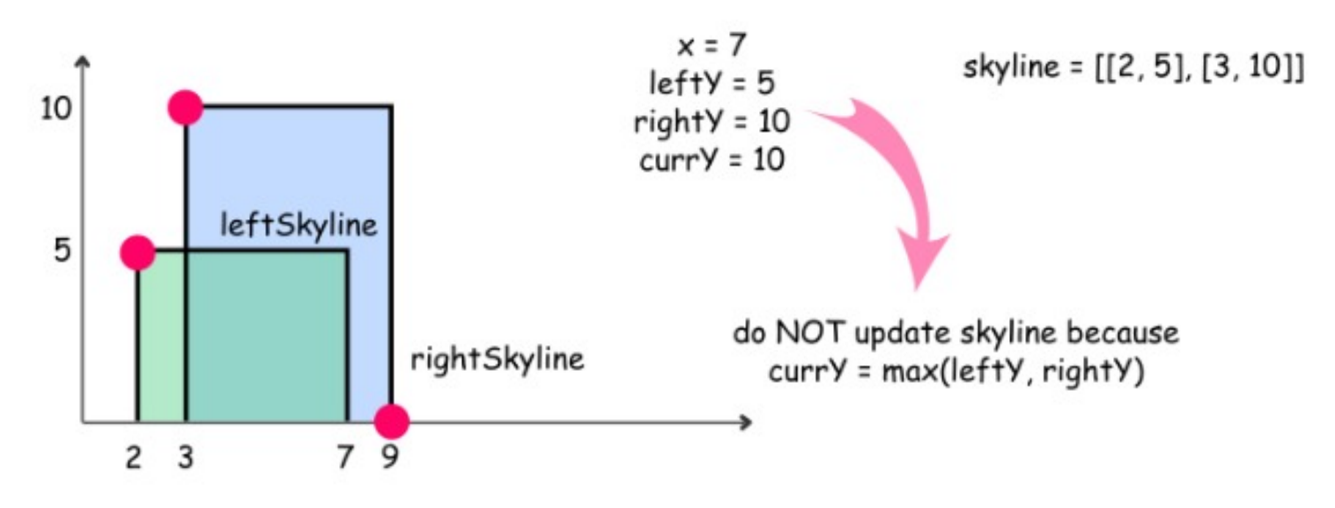
> Here are three usecases to illustrate the merge algorithm execution



x = 3
leftY = 5
rightY = 10
currY = 5

skyline = [[2, 5]]

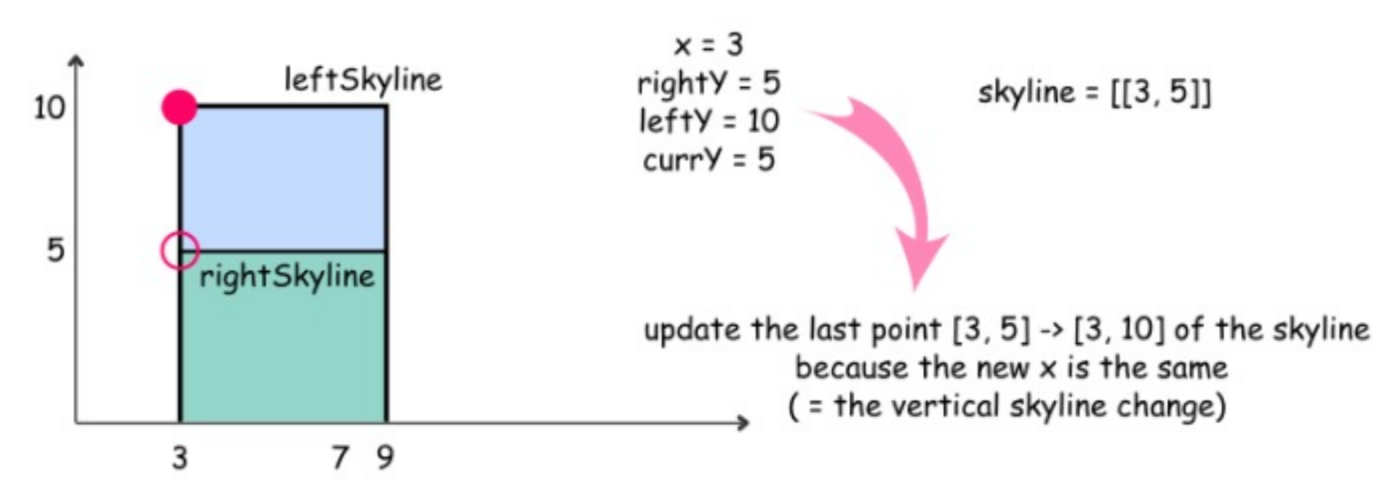
append the new point to skyline
(x = 3, y = max(leftY, rightY) = 10)



x = 7
leftY = 5
rightY = 10
currY = 10

skyline = [[2, 5], [3, 10]]

do NOT update skyline because
currY = max(leftY, rightY)



x = 3
rightY = 5
leftY = 10
currY = 5

skyline = [[3, 5]]

update the last point [3, 5] -> [3, 10] of the skyline
because the new x is the same
(= the vertical skyline change)

Implementation

Java Python Copy

```
class Solution:
    def getSkyline(self, buildings: 'List[List[int]]') -> 'List[List[int]]':
        """
        Divide-and-conquer algorithm to solve skyline problem,
        which is similar with the merge sort algorithm.
        """
        n = len(buildings)
        # The base cases
        if n == 0:
            return []
        if n == 1:
            x_start, x_end, y = buildings[0]
            return [[x_start, y], [x_end, 0]]

        # If there is more than one building,
        # recursively divide the input into two subproblems.
        left_skyline = self.getSkyline(buildings[: n // 2])
        right_skyline = self.getSkyline(buildings[n // 2 :])

        # Merge the results of subproblem together.
        return self.merge_skylines(left_skyline, right_skyline)

    def merge_skylines(self, left, right):
        """
        Merge two skylines together.
        """
        def update_output(x, y):
```

- Time complexity : $\mathcal{O}(N \log N)$, where N is number of buildings. The problem is an example of Master Theorem case II : $T(N) = 2T(\frac{N}{2}) + 2N$, that results in $\mathcal{O}(N \log N)$ time complexity.
- Space complexity : $\mathcal{O}(N)$ to keep the output.

Rate this article: ★★★★★

Previous Next

Comments: 15 Sort By

Type comment here... (Markdown is supported)

Preview Post

solutionsberkeley ★ 18 April 14, 2019 6:22 PM

what is the definition for nL and nR ?

18 Share Reply

SHOW 3 REPLIES

dovermore ★ 9 August 13, 2019 7:22 PM

I was thinking about using three heaps for left right height and do a sweep through the buildings.

9 Share Reply

SHOW 1 REPLY

tapanvaishnav17 ★ 18 June 5, 2020 5:26 AM

There were so many `straightforward` s in the article, I'm wondering if it was really that straightforward!?

8 Share Reply

ceek ★ 20 February 21, 2020 6:51 PM

In my opinion a solution that uses each building start and end point in an event stream updating a max heap and outputting the skyline while consuming events from the steam is much more intuitive. Runtime would be O(m log m) where m is the count of buildings. Space complexity O(m)

8 Share Reply

ande_ka_funda ★ 89 May 27, 2020 12:43 PM

This approach is way too complicated. No way one can get is all right in an interview without messing up some edge cases.

See the tushar roy explanation on youtube. The code is intutive and doesn't have many edge cases.

3 Share Reply

SHOW 1 REPLY

chosun1 ★ 94 April 9, 2019 2:32 AM

Very cool. Here is a solution to another problem Rectangle Area II: https://leetcode.com/articles/rectangle-area-ii/# written up by awice. The problems are conceptually similar (rectangle union) but his solution uses segment trees and sweep line.

3 Share Reply

ajithesh2 ★ 3 January 13, 2020 12:02 AM

Can't we do it simply, by storing low to high indices(horizontal distance) in a map, and reverse sort height and fill in the ranges in descending order. The solutions discussed here are pretty complicated.

1 Share Reply

SHOW 1 REPLY

kremebrulee ★ 52 June 15, 2019 8:42 AM

Also pretty confused why we need currY in the last leg of the algorithm when we're in appendSkyline. Can't think of a counter example of when currY would come in handy? the remaining skyline would never overlap with itself. and the other skyline would be at height 0.

1 Share Reply

SHOW 1 REPLY

kremebrulee ★ 52 June 15, 2019 9:59 PM

How come in some problems we include the space of the output in the space complexity (like this problem) and in other problems we don't count it?

1 Share Reply

SHOW 1 REPLY

kremebrulee ★ 52 June 15, 2019 8:25 AM

Why does appendSkyline return currY?

1 Share Reply

SHOW 2 REPLIES

‹ 1 2 ›