2. Divide and Conquer

a) Because the x cordinates in a skyline are already sorted, We can pick x cordinates from smallest to biggest in order by comparing the leftmost element in each skyline. We also keep track of the current height(h_1) in skyline A and current height(h_2) in skyline B. Then pick the bigger one and update it to resulting (This step will give us the upper skyline and eliminate lower ones) lists.

Here is a overall pickture of my algorithm: (set $h_1=0$, $h_2=0$ at begining)

step1: ① Compare the leftmost x cordinates in skyline A and skyline B.
(repeat) ② remove it from skyline A or B for which it belongs to. (x, h)
③ set h_1 or h_2 to height that is picked
④ put (x, max(h_1, h_2)) to the result list.

step2: ① If either skyline A or skyline B is empty, put the rest of elements directly to the result list.
② return result.

Here is a brief example of running my algorithm.

Skyline A = {(1,11), (3,12), (6,0)}. B = {(2,5), (5,0)}

Pick (1,11)
$h_1 = 11$, $h_2 = 0$
max(h_1, h_2) = 11
result = [(1,11)]

Pick (2,5)
$h_1 = 11$, $h_2 = 5$
max(h_1, h_2) = 11
result = [(1,11), (2,11)]

Pick (3,12)
$h_1 = 12$, $h_2 = 5$
max(h_1, h_2) = 12
result = [(1,11), (2,11), (3,12)]

Pick (5,0)
$h_1 = 12$, $h_2 = 0$
max(h_1, h_2) = 12
result = (1,11)(2,11)(3,12), (5,12)

Done! because B is empty, Simply put (6,0) at the end.

A problem here is that (2,11) and (5,12) is useless since it has the same height as the previous element. To fix this problem, simply add an if statement between step1 ③ and step1 ④: [If h == previous_element_ height, Don't add this element]
Note that the height of previous element is also easy to track.

Time complexity is on next page ☆
the final output for my alogrithm and example is: [(1,11), (3,12), (6,0)]
which is in right logic and correct.

Step 1

Time complexity: Only one for loop is used and each element in skyline A and B is only went through once. As a result, if we want to combine skyline A of size n_1 and skyline of size n_2, the running time is $O(n_1 + n_2)$.

b) We can apply divide and conquer method to build this algorithm. From previous question, we know it takes O(n) time to merge two skyline

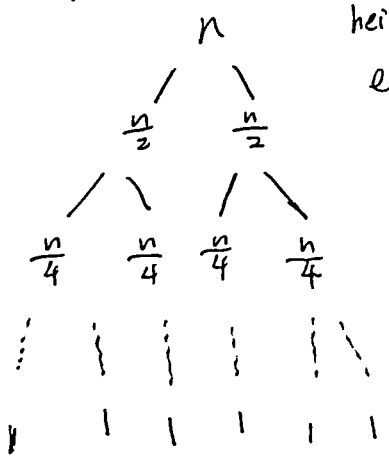
Here is a short picture of my algorithm:

```
find_skyline (list of n buildings) {
  if len(list of n buildings) == 1:
      return this building  (Base case for my recursion)
  result = merge_skyline (half list of n buildings, half list of n buildings)
  return result }.
```

→ function from last question

split the input list in half

Here is a picture of the recursion tree: (time complexity)



height = $\log n$

each step takes $O(n)$ time.

As a result, the time complexity of this algorithm is $O(n \log n)$.

We can also solve $T(n) \leq 2T(\frac{n}{2}) + O(n)$ which also gives $O(n \log n)$.