CSCI 432: The Skyline Algorithm

Name: _____

Collaborators: _____

April 2026

The Problem

A city's **skyline** is the outer silhouette formed by all its buildings when viewed from a distance. Given n buildings, each described as $[\ell_i, r_i, h_i]$ (ℓ_i = left edge, r_i = right edge, h_i = height), compute the skyline as a list of **key points** $[x, \text{height}]$ marking every location where the visible height changes.

Output rules: (1) No two consecutive key points share the same height. (2) The last key point always has height 0. (3) Key points are sorted by x.

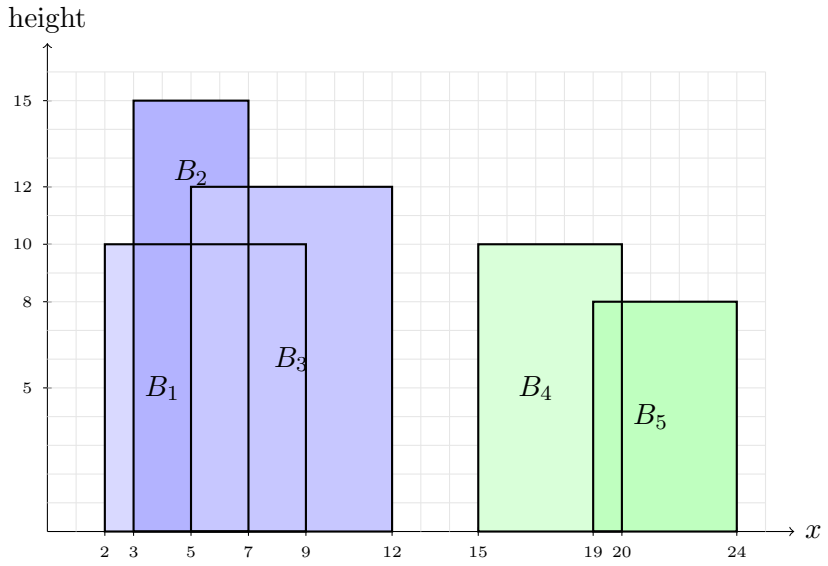
Three approaches we will study today:

1. **Brute Force**
2. **Priority Queue**
3. **Divide & Conquer**

Warm-Up: Understanding the Output

Consider the following five buildings:

$$B_1 = [2, 9, 10], \quad B_2 = [3, 7, 15], \quad B_3 = [5, 12, 12], \quad B_4 = [15, 20, 10], \quad B_5 = [19, 24, 8]$$



1. On the figure above, **draw the skyline** by tracing the outer silhouette with a bold line. Then list the key points you identified.

Key points: _____

2. For each key point record the height change and explain *why* a new key point is recorded there.

Key point	**Height change**	**Reason**

Part I: Brute Force

Idea: Collect every left and right x-coordinate as a critical point. For each critical point, scan *all* buildings to find the tallest active one. A building $[\ell, r, h]$ is *active* at x if $\ell \leq x < r$. Record a key point whenever the maximum height changes.

Algorithm 1: Brute Force Skyline

Input : buildings: an array of n triples $[\ell_i, r_i, h_i]$
Output: result: a list of key points $[x, \text{height}]$ forming the skyline

```
Collect all left and right x-coordinates → points[]
Sort points[]
result ← []
prev ← 0
for each x in points do
    maxH ← 0
    for each building [ℓ, r, h] in buildings do
        if ℓ ≤ x < r then
            maxH ← max(maxH, h)
    if maxH ≠ prev then
        result.add([x, maxH])
        prev ← maxH
return result
```

3. Using the five buildings from the warm-up, complete the brute-force table. The critical x-coordinates are $\{2, 3, 5, 7, 9, 12, 15, 19, 20, 24\}$.

x	**Active buildings (list all)**	**Max height**	**Record key point?**
2			
3			
5			
7			
9			
12			
15			
19			
20			
24			

Output: _____

4. What is the runtime of this brute force approach? Could we do better?

Part II: Priority Queue

Idea: This algorithm follows a very similar logic to the brute force method, however, a priority queue is used to keep track of only the buildings we need to pay attention to.

Algorithm 2: Priority Queue Skyline

Input : buildings: an array of n triples $[\ell_i, r_i, h_i]$
Output: result: a list of key points $[x, \text{height}]$ forming the skyline

```
1  Collect all left and right x-coordinates → points[]
2  Sort points[]
3  result ← []
4  active ← MaxPriorityQueue()
5  i ← 0
6  for each x in points do
7  |   while i < n AND buildings[i][ℓ_i] ≤ x do
8  |   |   active.add([buildings[i][h_i], buildings[i][ℓ_i], buildings[i][r_i]])
9  |   |   i++
10 |   while active !empty AND active[0][r_i] ≤ x do
11 |   |   active.pop()
12 |   h ← 0
13 |   if active !empty then
14 |   |   h ← −active[0][h_i]
15 |   if result !empty AND result[−1][h_i] ≠ h then
16 |   |   result.add([x, h])
17 return result
```

5. Walk through a small example: buildings = [[1, 5, 10], [4, 7, 5], [7, 9, 5]]

6. Why might it be important for us to have the two while loops on lines 7 and 10?

7. What is the if statement on line 15 doing for us? (Hint: Watch how it behaves when dealing with the last two buildings in our walk through)

8. Is the asymptotic runtime of this algorithm faster than the naive approach? Why/why not.

9. Write out the following conditions based on the above algorithm

Precondition (p) State the necessary conditions that must be true before the algorithm starts.

Loop Guard (g) State the condition that controls the execution of the main loop.

Postcondition (q) State the conditions that must be true after the algorithm terminates.

Loop Invariant ($L = L_i$) State an invariant that holds before and after each iteration of the loop.

9.Invariant Proof

Initialization Show that the loop invariant holds before the first iteration of the loop. $(P \wedge G \implies L)$

Maintenance Assuming the loop invariant holds at the beginning of an iteration, show that it still holds after the iteration completes. $(G \wedge L_i \implies L_{i+1})$

Termination Explain how the loop invariant and the negation of the loop guard imply the correctness of the algorithm upon termination.$(L \wedge \neg G \implies P)$

Part III: Divide & Conquer

Consider the following algorithm which will merge two existing skylines

Algorithm 3: Merge Skylines

Input : skyline1, skyline2: two skylines to be merged
Output: mergedSkyline: a merged list of key points $[x, \text{height}]$ forming the skyline

```
1  mergedSkyline ← []
2  x, h1, h2, h_max, h_cur ← 0
3  while skyline1 ∧ skyline2 do
4      if skyline1[0][0] < skyline2[0][0] then
5          x, h1 ← skyline1.pop()
6      else if skyline2[0][0] < skyline1[0][0] then
7          x, h2 ← skyline2.pop()
8      else
9          x, h1 ← skyline1.pop()
10         h2 ← skyline2.pop()[1]
11     h_max ← max(h1, h2)
12     if h_cur ≠ h_max then
13         h_cur ← h_max
14         mergedSkyline.add([x, h_cur])
15 return mergedSkyline + skyline1 + skyline2
```

10. Walk through merging the following skylines

$s_1 = [[0, 0], [1, 10], [5, 0]]$ $s_2 = [[0, 0], [4, 5], [7, 0]]$

11. What is the asymptotic runtime of merging skylines this way?

Algorithm 4: Recursive Skyline

Input : skylines: a list of separate skylines
Output: result: a list of key points $[x, \text{height}]$ forming the combined skyline

```
if len(skylines) = 1 then
    return skylines[0]
mid ← ⌊len(skylines)/2⌋
left ← getSkyline(skylines[0 ... mid])
right ← getSkyline(skylines[mid+1 ... n])
merged ← MergedSkylines(left,right)
return merged
```

12. Based on your answer to (11.) write the recurrence relation for the above algorithm

13. Given the recurrence relation, what is the asymptotic runtime of the algorithm? (hint: it's the same as another well known "merging" algorithm)

14. Algorithm 4 does not accept arrays of buildings directly and instead needs an array of skylines as input. Provide a linear time algorithm which can convert an array of buildings B (of the form $[\ell, r, h]$)) to an array of skylines S (key points of the form $[x, h]$). (hint: consider the two example skylines given for question 10)

15. Recursion Invariant Proof

Invariant what is the recursion invariant R for the above algorithm?

Initialization (base case) show that the recurion invariant holds for the smallest input (i.e. when len(skylines) = 1)

Maintenance (inductive step) Assuming that the recursion invariant holds after a call to RecursiveSkyline(K) (where K is an array of skylines of size $k < n$), show that it still holds after a call to RecursiveSkylines(N) (where N is an array of skylines of size n).

Termination Show that if the preconditions are met, the recursion invariant holds, and the algorithm terminates then the algorithm will produce the correct output. ($P \wedge R \wedge T \implies Q$)