A city's **skyline** is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return *the****skyline****formed by these buildings collectively*.

The geometric information of each building is given in the array buildings where buildings[i] = [lefti, righti, heighti]:

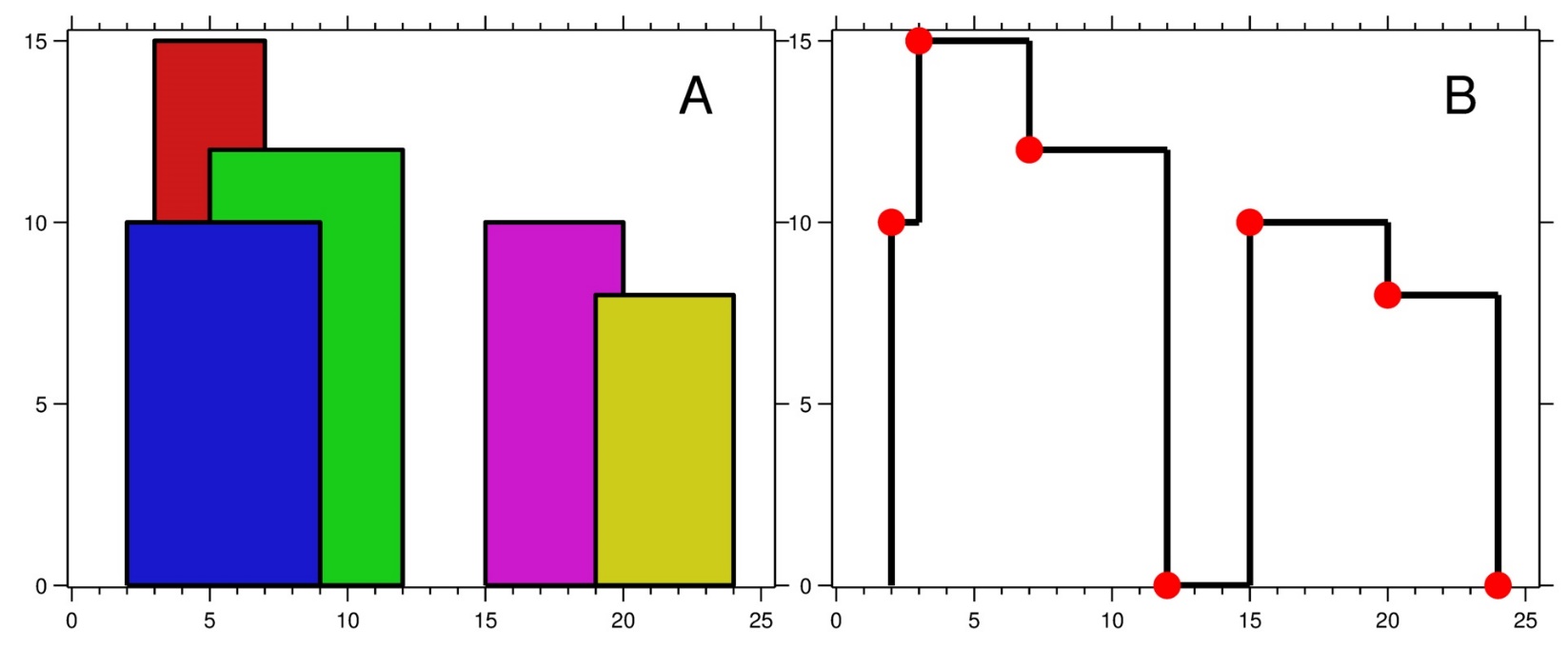
* lefti is the x coordinate of the left edge of the ith building.
* righti is the x coordinate of the right edge of the ith building.
* heighti is the height of the ith building.

You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

The **skyline** should be represented as a list of "key points" **sorted by their x-coordinate** in the form [[x1,y1],[x2,y2],...]. Each key point is the left endpoint of some horizontal segment in the skyline except the last point in the list, which always has a y-coordinate 0 and is used to mark the skyline's termination where the rightmost building ends. Any ground between the leftmost and rightmost buildings should be part of the skyline's contour.

**Note:** 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],...]

**Example 1:**



**Input:** buildings = [[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]

**Output:** [[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]

**Explanation:**

Figure A shows the buildings of the input.

Figure B shows the skyline formed by those buildings. The red points in figure B represent the key points in the output list.

**Example 2:**

**Input:** buildings = [[0,2,3],[2,5,3]]

**Output:** [[0,3],[5,0]]

**Solution:**

**Using TreeMap: (Faster)**

class Solution {

static class BuildingPoint implements Comparable<BuildingPoint> {

int x;

boolean isStart;

int height;

@Override

public int compareTo(BuildingPoint o) {

//first compare by x.

//If they are same then use this logic

//if two starts are compared then higher height building should be picked first

//if two ends are compared then lower height building should be picked first

//if one start and end is compared then start should appear before end

if (this.x != o.x) {

return this.x - o.x;

} else {

return (this.isStart ? -this.height : this.height) - (o.isStart ? -o.height : o.height);

}

}

}

public List<List<Integer>> getSkyline(int[][] buildings) {

//for all start and end of building put them into List of BuildingPoint

BuildingPoint[] buildingPoints = new BuildingPoint[buildings.length\*2];

int index = 0;

for(int building[] : buildings) {

buildingPoints[index] = new BuildingPoint();

buildingPoints[index].x = building[0];

buildingPoints[index].isStart = true;

buildingPoints[index].height = building[2];

buildingPoints[index + 1] = new BuildingPoint();

buildingPoints[index + 1].x = building[1];

buildingPoints[index + 1].isStart = false;

buildingPoints[index + 1].height = building[2];

index += 2;

}

Arrays.sort(buildingPoints);

//using TreeMap because it gives log time performance.

//PriorityQueue in java does not support remove(object) operation in log time.

TreeMap<Integer, Integer> queue = new TreeMap<>();

queue.put(0, 1);

int prevMaxHeight = 0;

List<List<Integer>> result = new ArrayList<>();

for(BuildingPoint buildingPoint : buildingPoints) {

//if it is start of building then add the height to map. If height already exists then increment

//the value

if (buildingPoint.isStart) {

queue.compute(buildingPoint.height, (key, value) -> {

if (value != null) {

return value + 1;

}

return 1;

});

} else { //if it is end of building then decrement or remove the height from map.

queue.compute(buildingPoint.height, (key, value) -> {

if (value == 1) {

return null;

}

return value - 1;

});

}

//peek the current height after addition or removal of building x.

int currentMaxHeight = queue.lastKey();

//if height changes from previous height then this building x becomes critcal x.

// So add it to the result.

if (prevMaxHeight != currentMaxHeight) {

List<Integer> list = new ArrayList<>();

list.add(buildingPoint.x);

list.add(currentMaxHeight);

result.add(list);

prevMaxHeight = currentMaxHeight;

}

}

return result;

}

}

Using Priority Queue(Slower)

class Solution {

static class BuildingPoint implements Comparable<BuildingPoint> {

int x;

boolean isStart;

int height;

@Override

public int compareTo(BuildingPoint o) {

//first compare by x.

//If they are same then use this logic

//if two starts are compared then higher height building should be picked first

//if two ends are compared then lower height building should be picked first

//if one start and end is compared then start should appear before end

if (this.x != o.x) {

return this.x - o.x;

} else {

return (this.isStart ? -this.height : this.height) - (o.isStart ? -o.height : o.height);

}

}

}

public List<List<Integer>> getSkyline(int[][] buildings) {

//for all start and end of building put them into List of BuildingPoint

BuildingPoint[] buildingPoints = new BuildingPoint[buildings.length\*2];

int index = 0;

for(int building[] : buildings) {

buildingPoints[index] = new BuildingPoint();

buildingPoints[index].x = building[0];

buildingPoints[index].isStart = true;

buildingPoints[index].height = building[2];

buildingPoints[index + 1] = new BuildingPoint();

buildingPoints[index + 1].x = building[1];

buildingPoints[index + 1].isStart = false;

buildingPoints[index + 1].height = building[2];

index += 2;

}

Arrays.sort(buildingPoints);

PriorityQueue<Integer> queue1 = new PriorityQueue<>(Collections.reverseOrder());

queue1.add(0);

int prevMaxHeight = 0;

List<List<Integer>> result = new ArrayList<>();

for(BuildingPoint buildingPoint : buildingPoints) {

//if it is start of building then add the height to map. If height already exists then increment

//the value

if (buildingPoint.isStart) {

queue1.add(buildingPoint.height);

} else { //if it is end of building then decrement or remove the height from map.

queue1.remove(buildingPoint.height);

}

//peek the current height after addition or removal of building x.

int currentMaxHeight = queue1.peek();

//if height changes from previous height then this building x becomes critcal x.

// So add it to the result.

if (prevMaxHeight != currentMaxHeight) {

List<Integer> list = new ArrayList<>();

list.add(buildingPoint.x);

list.add(currentMaxHeight);

result.add(list);

prevMaxHeight = currentMaxHeight;

}

}

return result;

}

}