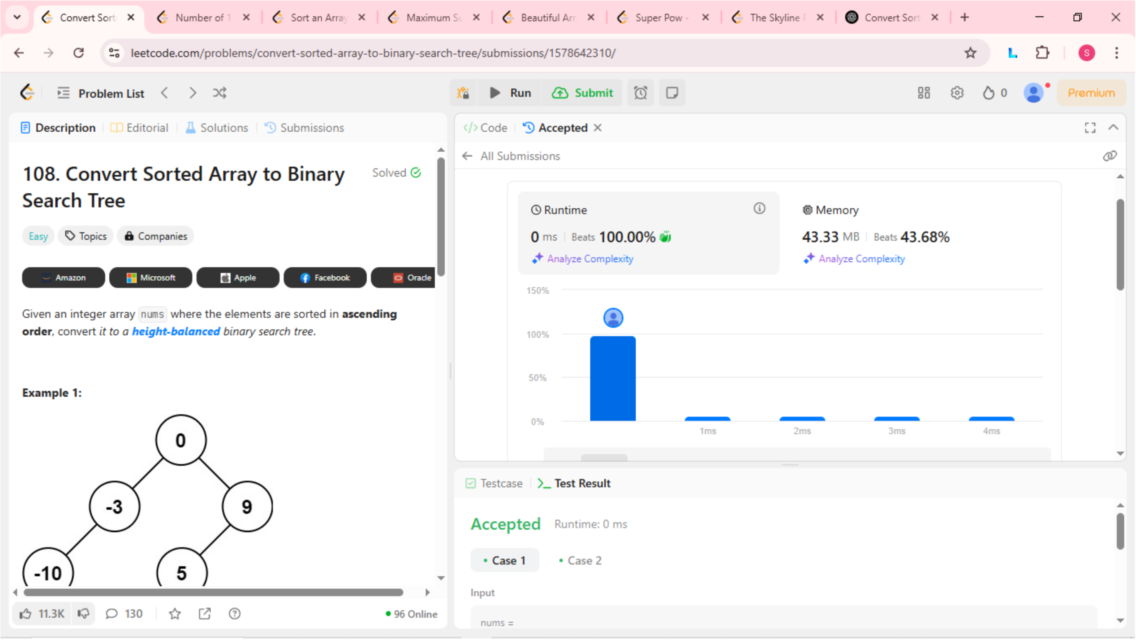
Shaurya Sharma 22BCS15079

108. Convert Sorted Array to Binary Search Tree



108. Convert Sorted Array to Binary Search Tree

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree.*

Example 1:

Runtime
0 ms Beats 100.00%

Memory
43.33 MB Beats 43.68%

Accepted Runtime: 0 ms

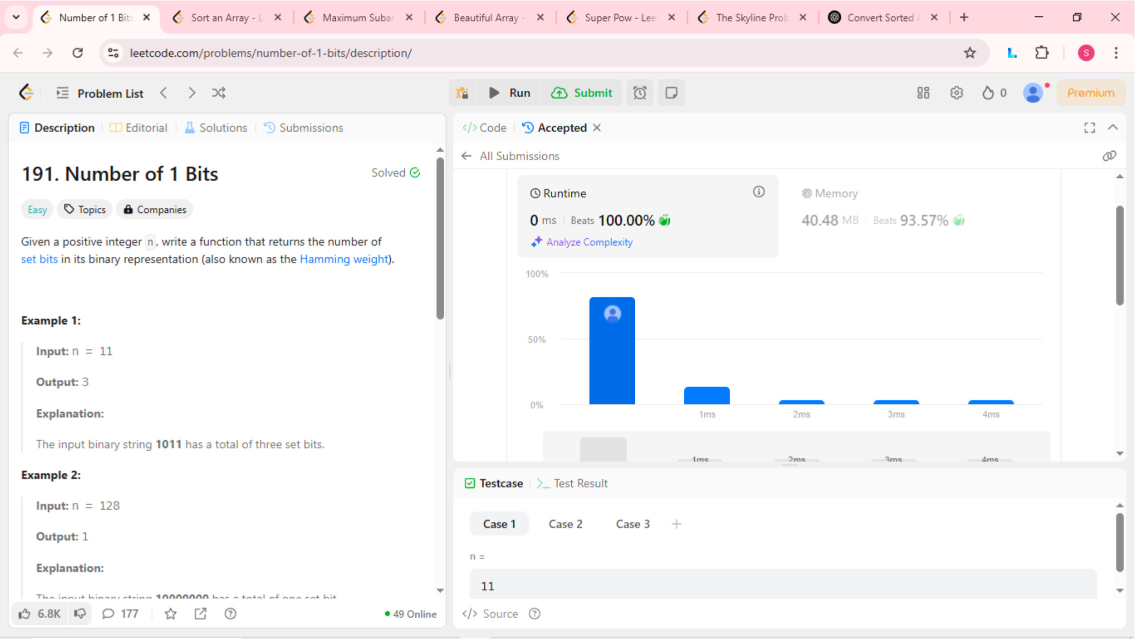
Case 1 Case 2

Input

nums =

```
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return sortedArrayToBSTHelper(nums, 0, nums.length - 1);
    }

    private TreeNode sortedArrayToBSTHelper(int[] nums, int left, int right) {
        if (left > right) {
            return null;
        }

        int mid = left + (right - left) / 2;
        TreeNode root = new TreeNode(nums[mid]);

        root.left = sortedArrayToBSTHelper(nums, left, mid - 1);
        root.right = sortedArrayToBSTHelper(nums, mid + 1, right);

        return root;
    }
}
```

191. Number of 1 Bits



191. Number of 1 Bits

Given a positive integer `n`, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

Example 1:

Input: n = 11

Output: 3

Explanation:

The input binary string **1011** has a total of three set bits.

Example 2:

Input: n = 128

Output: 1

Explanation:

The input binary string **10000000** has a total of one set bit.

Runtime

0 ms | Beats 100.00%

Memory

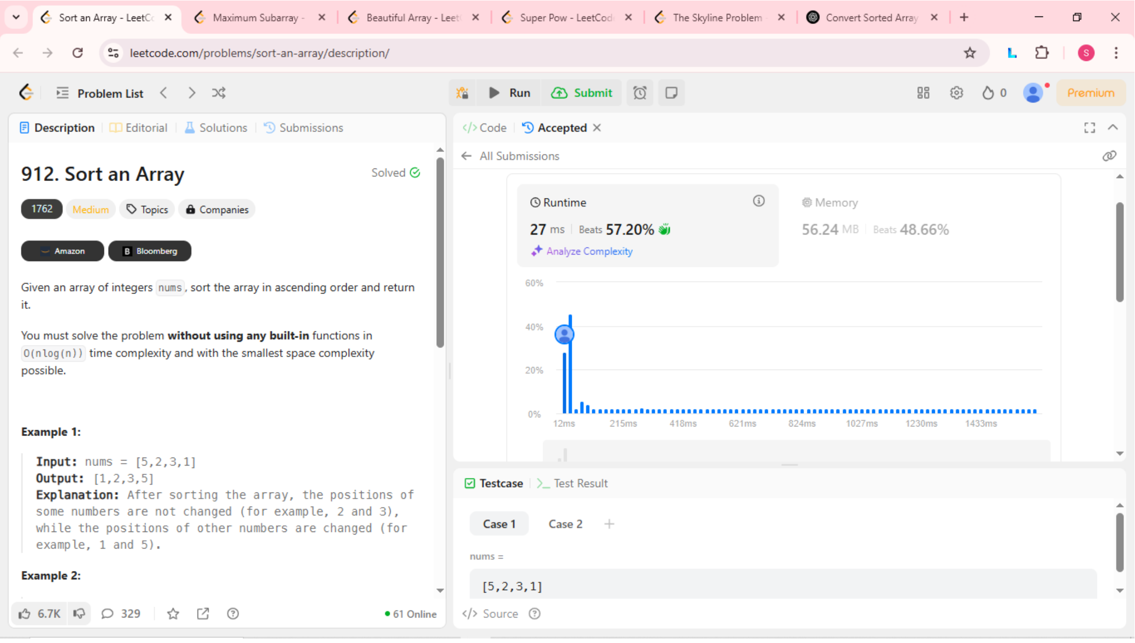
40.48 MB | Beats 93.57%

n =

11

```
class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += n & 1;
            n >>= 1;
        }
        return count;
    }
}
```

912. Sort an Array



912. Sort an Array

Given an array of integers `nums`, sort the array in ascending order and return it.

You must solve the problem **without using any built-in** functions in `O(nlog(n))` time complexity and with the smallest space complexity possible.

Example 1:

```
Input: nums = [5,2,3,1]
Output: [1,2,3,5]
Explanation: After sorting the array, the positions of
some numbers are not changed (for example, 2 and 3),
while the positions of other numbers are changed (for
example, 1 and 5).
```

Example 2:

Runtime

27 ms | Beats 57.20%

Memory

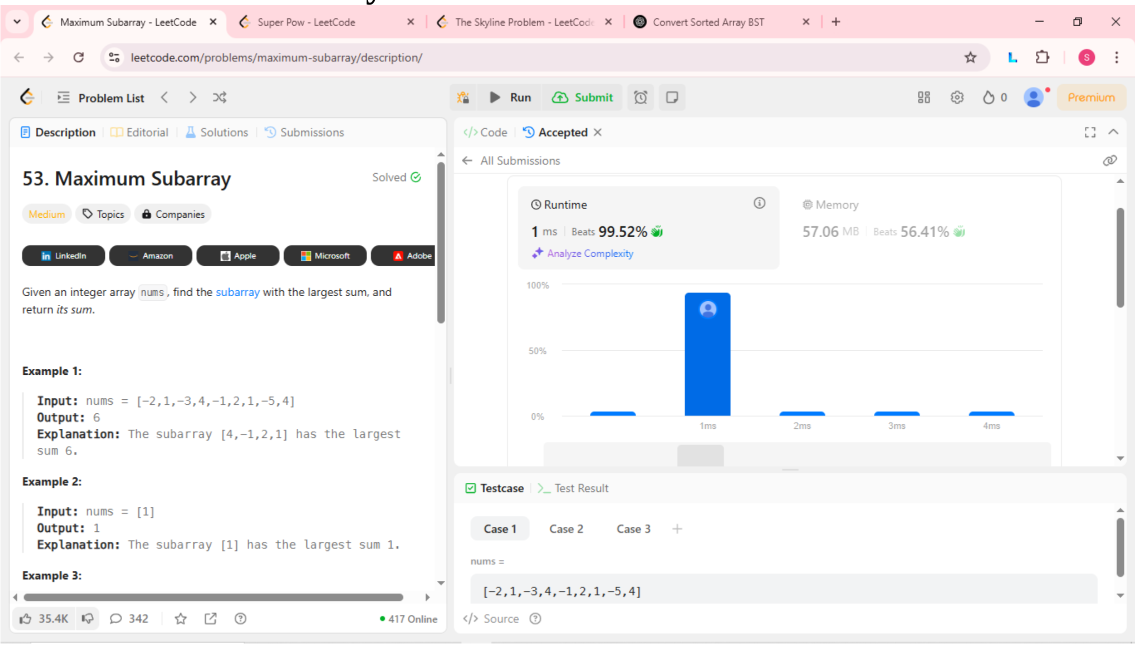
56.24 MB | Beats 48.66%

nums =

[5,2,3,1]

```java
class Solution {
    public int[] sortArray(int[] nums) {
        if (nums == null || nums.length < 2) {
            return nums;  // No need to sort if the array has fewer than 2 elements.
        }
        mergeSort(nums, 0, nums.length - 1);
        return nums;
    }

    private void mergeSort(int[] nums, int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;

            mergeSort(nums, left, mid);
            mergeSort(nums, mid + 1, right);

            merge(nums, left, mid, right);
        }
    }

    private void merge(int[] nums, int left, int mid, int right) {
        int[] leftArray = new int[mid - left + 1];
        int[] rightArray = new int[right - mid];

        System.arraycopy(nums, left, leftArray, 0, leftArray.length);
        System.arraycopy(nums, mid + 1, rightArray, 0, rightArray.length);

        int i = 0, j = 0, k = left;
        while (i < leftArray.length && j < rightArray.length) {
            if (leftArray[i] <= rightArray[j]) {
                nums[k++] = leftArray[i++];
            } else {
                nums[k++] = rightArray[j++];
            }
        }

        while (i < leftArray.length) {
            nums[k++] = leftArray[i++];
        }

        while (j < rightArray.length) {
            nums[k++] = rightArray[j++];
        }
    }
}
```

53. Maximum Subarray



53. Maximum Subarray

Solved

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest
sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

Example 3:

Runtime
1 ms | Beats 99.52%

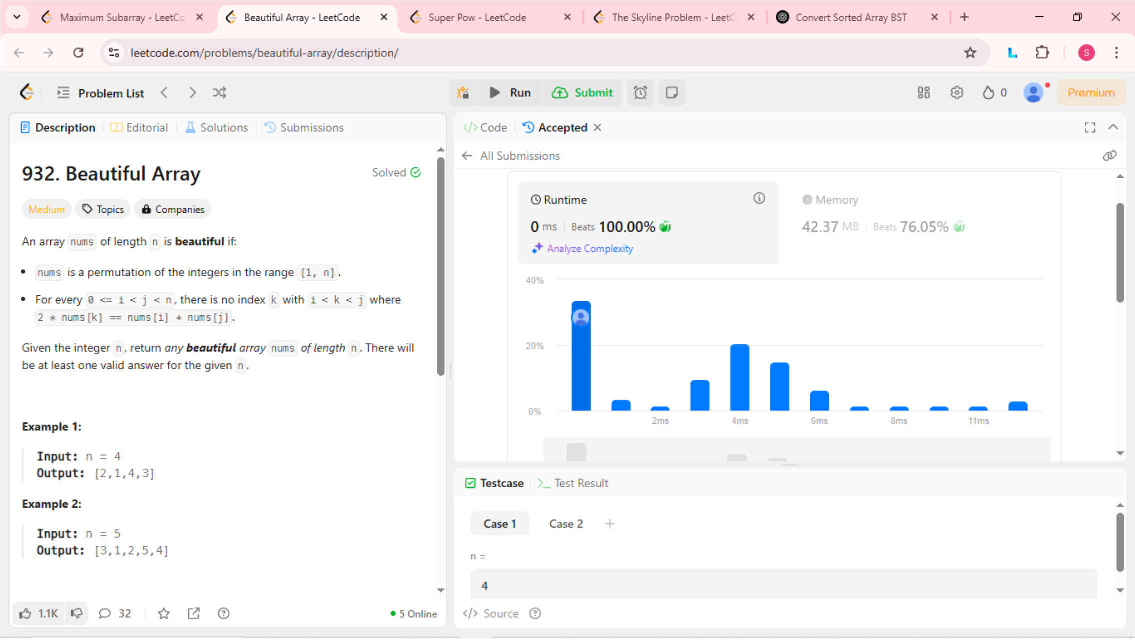
Memory
57.06 MB | Beats 56.41%

nums =

[-2,1,-3,4,-1,2,1,-5,4]

```
class Solution {
    public int maxSubArray(int[] nums) {
        int currentSum = nums[0];
        int maxSum = nums[0];

        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);

            maxSum = Math.max(maxSum, currentSum);
        }

        return maxSum;
    }
}
```

932. Beautiful Array



932. Beautiful Array

Solved

An array `nums` of length `n` is **beautiful** if:

- `nums` is a permutation of the integers in the range `[1, n]`.
- For every `0 <= i < j < n`, there is no index `k` with `i < k < j` where `2 * nums[k] == nums[i] + nums[j]`.

Given the integer `n`, return *any* ***beautiful*** *array* `nums` *of length* `n`. There will be at least one valid answer for the given `n`.

Example 1:

```
Input: n = 4
Output: [2,1,4,3]
```

Example 2:

```
Input: n = 5
Output: [3,1,2,5,4]
```

Runtime
0 ms | Beats 100.00%

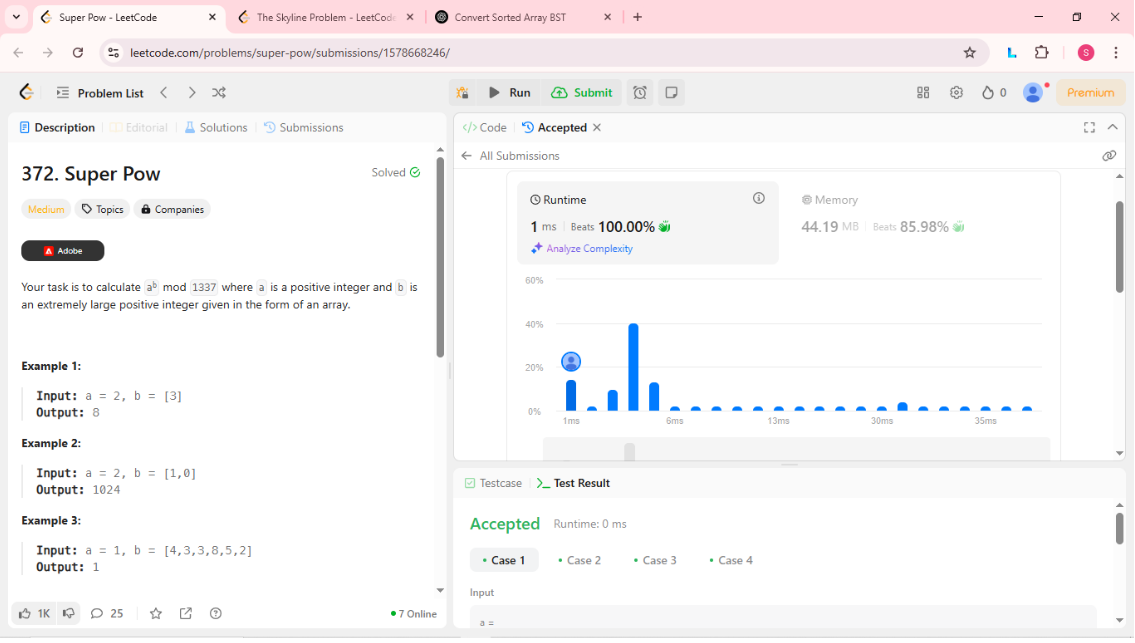
Memory
42.37 MB | Beats 76.05%

n =

4

```
class Solution {
    public int[] beautifulArray(int n) {
        if (n == 1) {
            return new int[] {1};
        }
        int[] left = beautifulArray((n + 1) >> 1);
        int[] right = beautifulArray(n >> 1);
        int[] ans = new int[n];
        int i = 0;
        for (int x : left) {
            ans[i++] = x * 2 - 1;
        }
        for (int x : right) {
            ans[i++] = x * 2;
        }
        return ans;

    }
}
```

372. Super Pow



372. Super Pow

Your task is to calculate a^b mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Example 1:

```
Input: a = 2, b = [3]
Output: 8
```

Example 2:

```
Input: a = 2, b = [1,0]
Output: 1024
```

Example 3:

```
Input: a = 1, b = [4,3,3,8,5,2]
Output: 1
```

Accepted Runtime: 0 ms

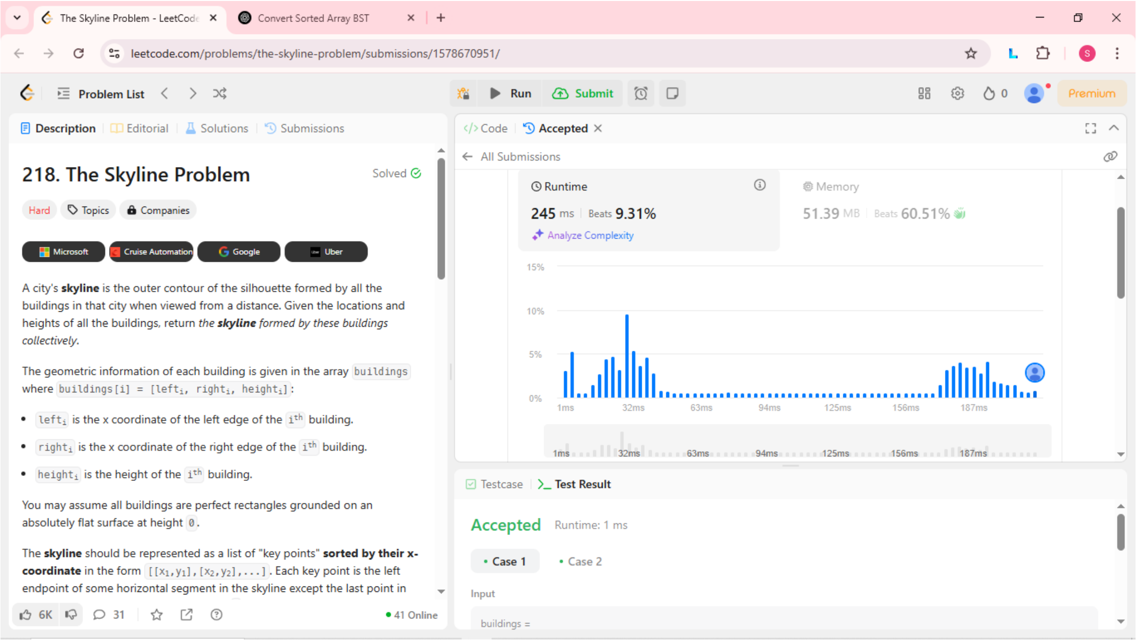
• Case 1 • Case 2 • Case 3 • Case 4

Input

a =

```
class Solution {
    private int modPow(int x, int y, int mod) {
        int result = 1;
        x = x % mod;
        while (y > 0) {
            if (y % 2 == 1) {
                result = (result * x) % mod;
            }
            x = (x * x) % mod;
            y /= 2;
        }
        return result;
    }

    public int superPow(int a, int[] b) {
        final int MOD = 1337;
        final int PHI_MOD = 1140;
        int exp = 0;
        for (int digit : b) {
            exp = (exp * 10 + digit) % PHI_MOD;
        }

        if (exp == 0) {
            exp = PHI_MOD;
        }

        return modPow(a, exp, MOD);
    }
}
```

218. The Skyline Problem



218. The Skyline Problem

Solved

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] = [left_i, right_i, height_i]`:

- `left_i` is the x coordinate of the left edge of the `i^th` building.
- `right_i` is the x coordinate of the right edge of the `i^th` building.
- `height_i` is the height of the `i^th` 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

Runtime

245 ms | Beats 9.31%

Analyze Complexity

Memory

51.39 MB | Beats 60.51%

Accepted Runtime: 1 ms

Case 1 Case 2

Input

buildings =

```
import java.util.*;

class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        // Step 1: Create events
        List<int[]> events = new ArrayList<>();
        for (int[] building : buildings) {
            int left = building[0], right = building[1], height = building[2];
            events.add(new int[]{left, -height}); // Start event (negative height)
            events.add(new int[]{right, height});  // End event (positive height)
        }

        // Step 2: Sort events
        Collections.sort(events, (a, b) -> {
            if (a[0] != b[0]) {
                return a[0] - b[0]; // Sort by x-coordinate
            } else {
                return a[1] - b[1]; // If same x, prioritize start events
            }
        });

        // Step 3: Process events
        List<List<Integer>> result = new ArrayList<>();
        PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
        maxHeap.offer(0); // Initialize with ground level
        int prevMax = 0;

        for (int[] event : events) {
            int x = event[0], height = event[1];
            if (height < 0) {
                // Start event: add height to the heap
                maxHeap.offer(-height);
            } else {
                // End event: remove height from the heap
                maxHeap.remove(height);
            }

            // Get the current maximum height
            int currMax = maxHeap.peek();
            if (currMax != prevMax) {
                // If the maximum height changes, add the key point to the result
                result.add(Arrays.asList(x, currMax));
                prevMax = currMax;
            }
        }

        return result;
    }
}
```