



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## EXPERIMENT - 6

**Student Name: JEET BHARTI**

**Branch: BE-CSE**

**Semester: 6<sup>th</sup>**

**Subject Name: Advanced Programming - II**

**UID: 22BCS14804**

**Section/Group: 602/A**

**Date of Performance: 19-03-25**

**Subject Code: 22CSP-351**

**Problems Solved –**

**108. Convert Sorted Array to Binary Search Tree**

**191. Number of 1 Bits**

**912. Sort an Array**

**53. Maximum Subarray**

**932. Beautiful Array**

**372. Super Pow**

**218. The Skyline Problem**

**108. Convert Sorted Array to Binary Search Tree**

**Aim – Convert a sorted array into a height-balanced binary search tree.**

```
class Solution {
```

```
public:
```

```
    TreeNode* sortedArrayToBST(vector<int>& nums) {
```

```
        return helper(nums, 0, nums.size() - 1);
```

```
    }
```

```
    TreeNode* helper(vector<int>& nums, int left, int right) {
```

```
        if (left > right) return nullptr;
```

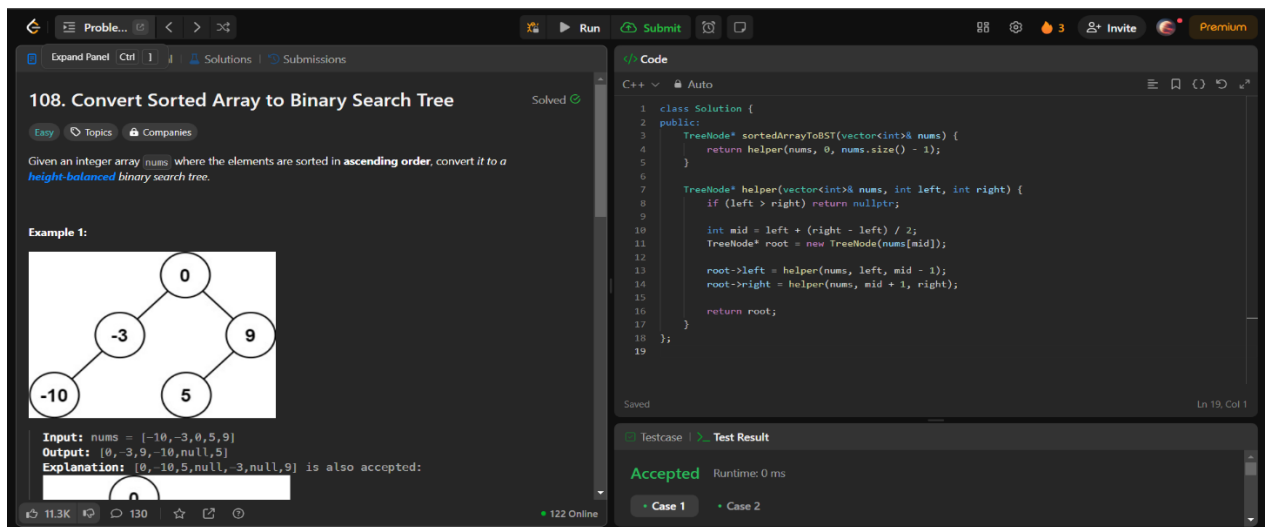
```
        int mid = left + (right - left) / 2;
```

```
        TreeNode* root = new TreeNode(nums[mid]);
```

```

root->left = helper(nums, left, mid - 1);
root->right = helper(nums, mid + 1, right);
return root;
}
};

```

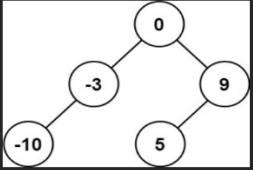


**108. Convert Sorted Array to Binary Search Tree** Solved

Easy Topics Companies

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

**Example 1:**



**Input:** `nums = [-10,-3,0,5,9]`  
**Output:** `[0,-3,9,-10,null,5]`  
**Explanation:** `[0,-10,5,null,-3,null,9]` is also accepted:

```

class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return helper(nums, 0, nums.size() - 1);
    }

    TreeNode* helper(vector<int>& nums, int left, int right) {
        if (left > right) return nullptr;

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

        root->left = helper(nums, left, mid - 1);
        root->right = helper(nums, mid + 1, right);

        return root;
    }
};

```

Accepted Runtime: 0 ms

Case 1 Case 2

## 191. Number of 1 Bits

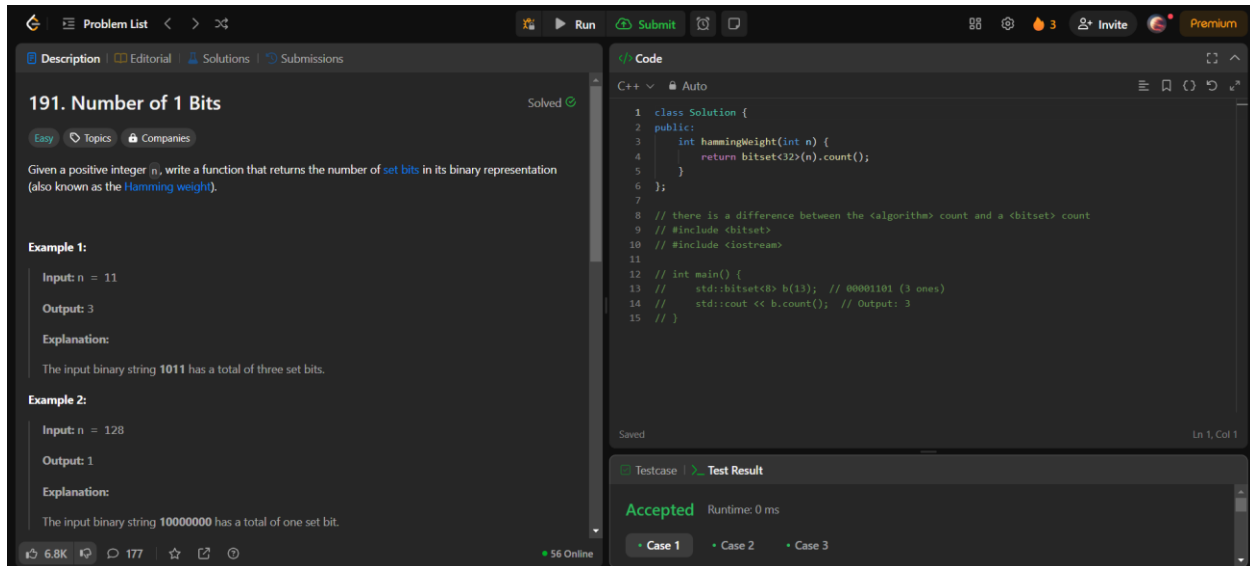
**Aim – Count the number of 1 bits in the binary representation of an integer.**

**CODE:-**

```

class Solution {
public:
    int hammingWeight(int n) {
        return bitset<32>(n).count();
    }
};

```



The screenshot shows a coding platform interface. On the left, the problem description for "191. Number of 1 Bits" is visible, including examples and explanations. On the right, a C++ code editor shows a solution using the `bitset` library. The code defines a `Solution` class with a `hammingWeight` method that returns the count of set bits in the binary representation of `n`. The code is saved and the test result is "Accepted".

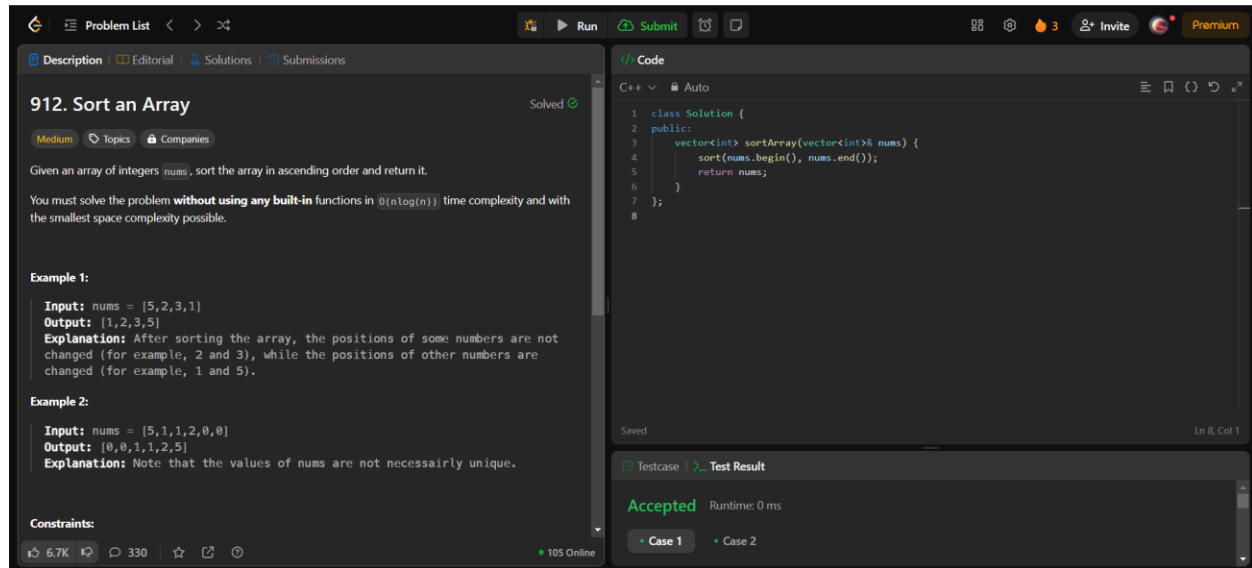
```
1 class Solution {
2 public:
3     int hammingWeight(int n) {
4         return bitset<32>(n).count();
5     }
6 };
7
8 // there is a difference between the <algorithm> count and a <bitset> count
9 // #include <bitset>
10 // #include <iostream>
11
12 // int main() {
13 //     std::bitset<8> b(13); // 00001101 (3 ones)
14 //     std::cout << b.count(); // Output: 3
15 // }
```

## 912. Sort an Array

**Aim – Sort an array in ascending order.**

**CODE:-**

```
class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        return nums;
    }
};
```

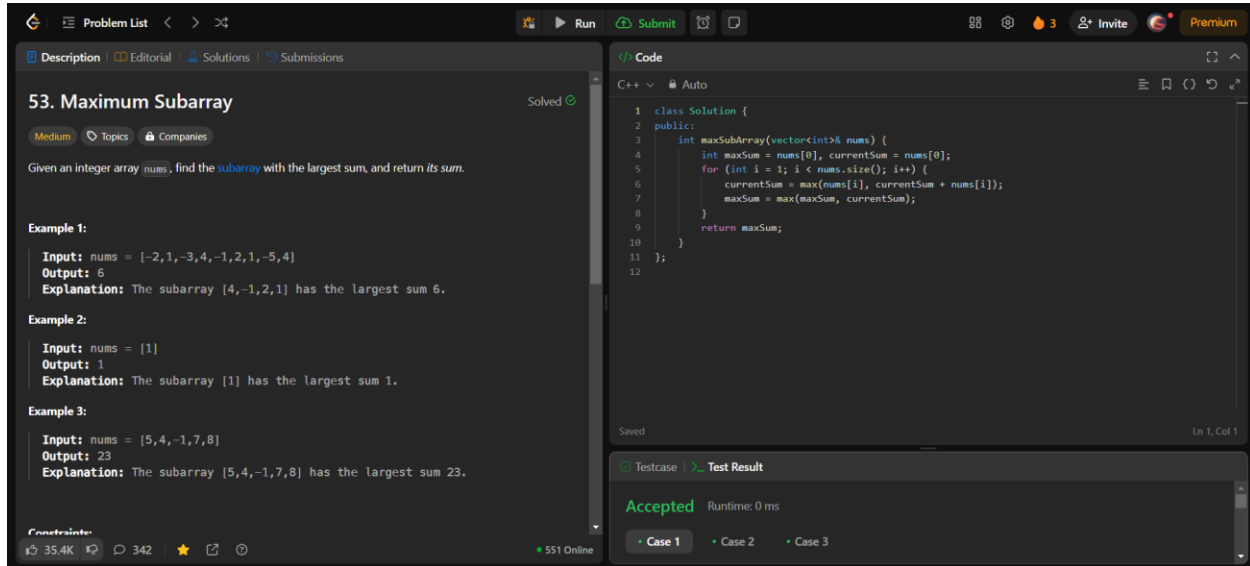


## 53. Maximum Subarray

**Aim – Find the contiguous subarray with the largest sum.**

**CODE:-**

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};
```



The screenshot shows the LeetCode interface for problem 53, "Maximum Subarray". The problem description states: "Given an integer array `nums`, find the subarray with the largest sum, and return its sum." It includes three examples with their inputs, outputs, and explanations. The C++ code in the editor implements a Kadane's algorithm solution. The test results at the bottom show "Accepted" with a runtime of 0 ms.

```

class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};

```

## 932. Beautiful Array

**Aim – Construct a "beautiful" array of integers from 1 to n.**

```
class Solution {
```

```
public:
```

```
    vector<int> beautifulArray(int n) {
```

```
        vector<int> res = {1};
```

```
        while (res.size() < n) {
```

```
            vector<int> temp;
```

```
            for (int num : res) if (num * 2 - 1 <= n) temp.push_back(num * 2 - 1);
```

```
            for (int num : res) if (num * 2 <= n) temp.push_back(num * 2);
```

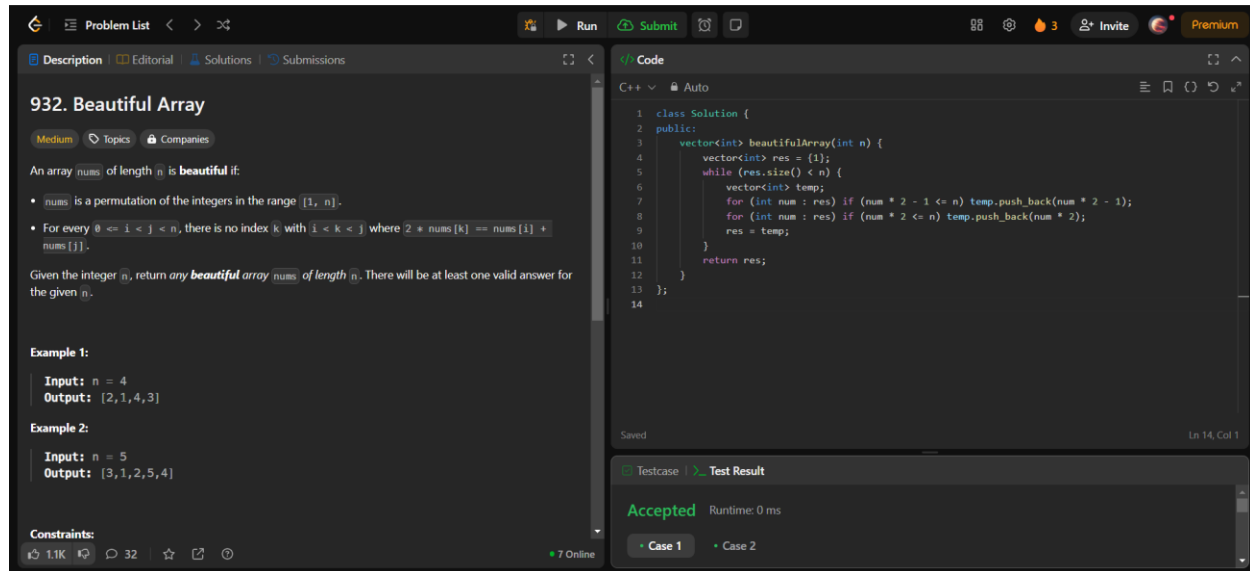
```
            res = temp;
```

```
        }
```

```
        return res;
```

```
    }
```

```
};
```



**932. Beautiful Array**

**Medium** Topics Companies

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

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

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]`

**Constraints:**  
1 ≤ `n` ≤ 1000

7 Online

```

1 class Solution {
2 public:
3     vector<int> beautifulArray(int n) {
4         vector<int> res = {1};
5         while (res.size() < n) {
6             vector<int> temp;
7             for (int num : res) if (num * 2 - 1 <= n) temp.push_back(num * 2 - 1);
8             for (int num : res) if (num * 2 <= n) temp.push_back(num * 2);
9             res = temp;
10        }
11        return res;
12    }
13 };
14

```

Accepted Runtime: 0 ms

Case 1 Case 2

## 372. Super Pow

**Aim – Compute  $a^b \bmod 1337$ , where `b` is an array of digits.**

**CODE:-**

```

class Solution {
public:
    static const int MOD = 1337;

    int modPow(int a, int k) {
        a %= MOD;
        int result = 1;
        for (int i = 0; i < k; ++i) {
            result = (result * a) % MOD;
        }
        return result;
    }
};

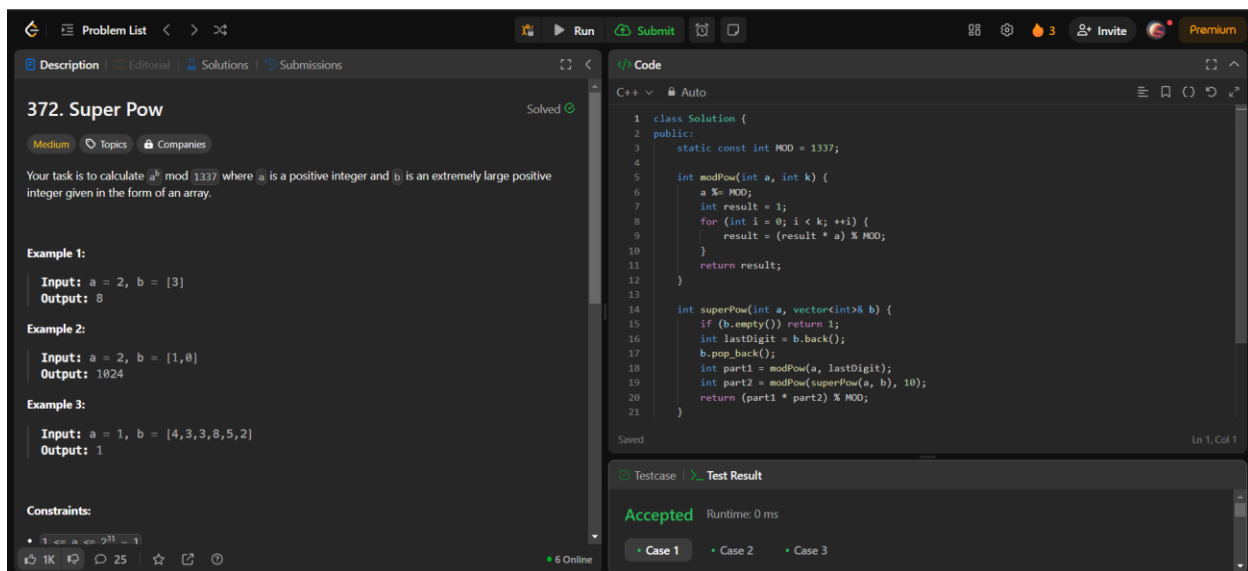
```

```

    }

    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1;
        int lastDigit = b.back();
        b.pop_back();
        int part1 = modPow(a, lastDigit);
        int part2 = modPow(superPow(a, b), 10);
        return (part1 * part2) % MOD;
    }
};

```



**372. Super Pow** Solved

Medium Topics Companies

Your task is to calculate  $a^b \bmod 1337$  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

**Constraints:**  
1 ≤ a ≤ 10000  
1 ≤ b.length ≤ 1000  
0 ≤ b[i] ≤ 9  
a and b are not zero.

```

1 class Solution {
2 public:
3     static const int MOD = 1337;
4
5     int modPow(int a, int k) {
6         a %= MOD;
7         int result = 1;
8         for (int i = 0; i < k; ++i) {
9             result = (result * a) % MOD;
10        }
11        return result;
12    }
13
14    int superPow(int a, vector<int>& b) {
15        if (b.empty()) return 1;
16        int lastDigit = b.back();
17        b.pop_back();
18        int part1 = modPow(a, lastDigit);
19        int part2 = modPow(superPow(a, b), 10);
20        return (part1 * part2) % MOD;
21    }
22 }

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

## 218. The Skyline Problem

**Aim –** Given a list of buildings, return the key points that form the skyline.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## CODE:-

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]);
            events.emplace_back(b[1], b[2]);
        }

        sort(events.begin(), events.end());
        multiset<int> heights = {0};
        vector<vector<int>> result;
        int prevHeight = 0;

        for (auto& e : events) {
            if (e.second < 0) heights.insert(-e.second);
            else heights.erase(heights.find(e.second));

            int currHeight = *heights.rbegin();
            if (currHeight != prevHeight) {
                result.push_back({e.first, currHeight});
                prevHeight = currHeight;
            }
        }
    }
}
```

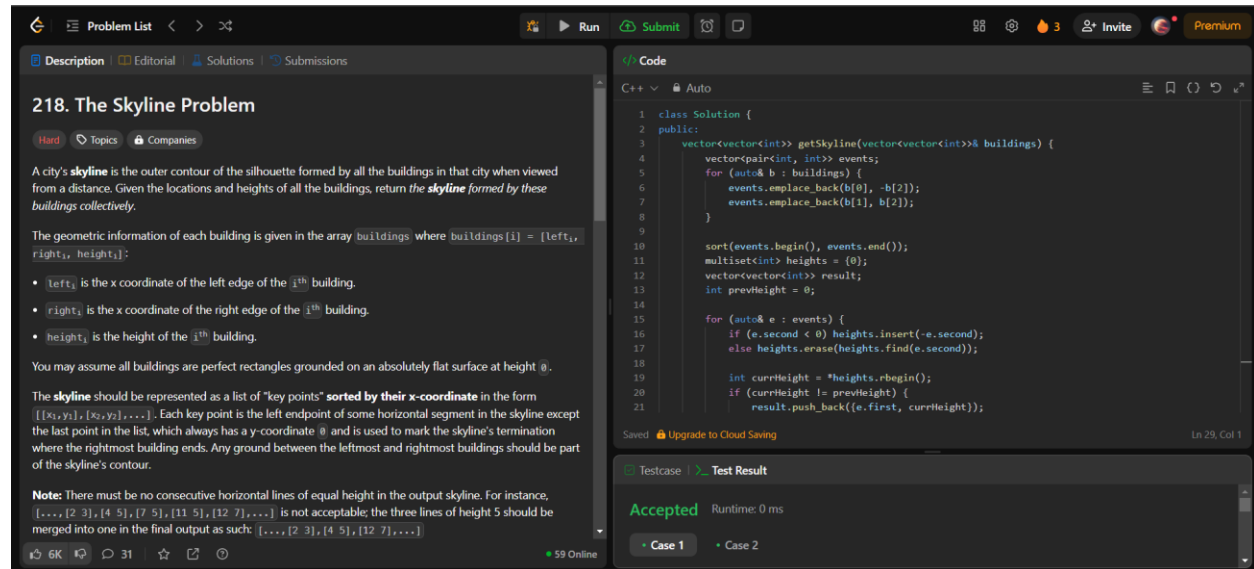


```

        return result;
    }

};

```



**218. The Skyline Problem**

**Hard** | Topics: Arrays, Sorting, Greedy | Companies: Google, Facebook, Microsoft, Amazon, Apple, LinkedIn, Uber, Lyft, Twitter, Bloomberg, Facebook, Microsoft, Amazon, Apple, LinkedIn, Uber, Lyft, Twitter, Bloomberg

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 *i*<sup>th</sup> building.
- `righti` is the x coordinate of the right edge of the *i*<sup>th</sup> building.
- `heighti` is the height of the *i*<sup>th</sup> 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], ...]`

```

1 class Solution {
2 public:
3     vector<vector<int>>> getSkyline(vector<vector<int>>>& buildings) {
4         vector<pair<int, int>> events;
5         for (auto& b : buildings) {
6             events.emplace_back(b[0], -b[2]);
7             events.emplace_back(b[1], b[2]);
8         }
9
10        sort(events.begin(), events.end());
11        multiset<int> heights = {0};
12        vector<vector<int>>> result;
13        int prevHeight = 0;
14
15        for (auto& e : events) {
16            if (e.second < 0) heights.insert(-e.second);
17            else heights.erase(heights.find(e.second));
18
19            int currHeight = *heights.rbegin();
20            if (currHeight != prevHeight) {
21                result.push_back({e.first, currHeight});
22            }
23        }
24        return result;
25    }
26 };

```

Accepted | Runtime: 0 ms

Case 1 | Case 2