



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 6

Student Name: Nitesh Bamber

Branch: BE-CSE

Semester: 6th

Subject Name: Advanced Programming - II

UID: 22BCS13791

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](#)

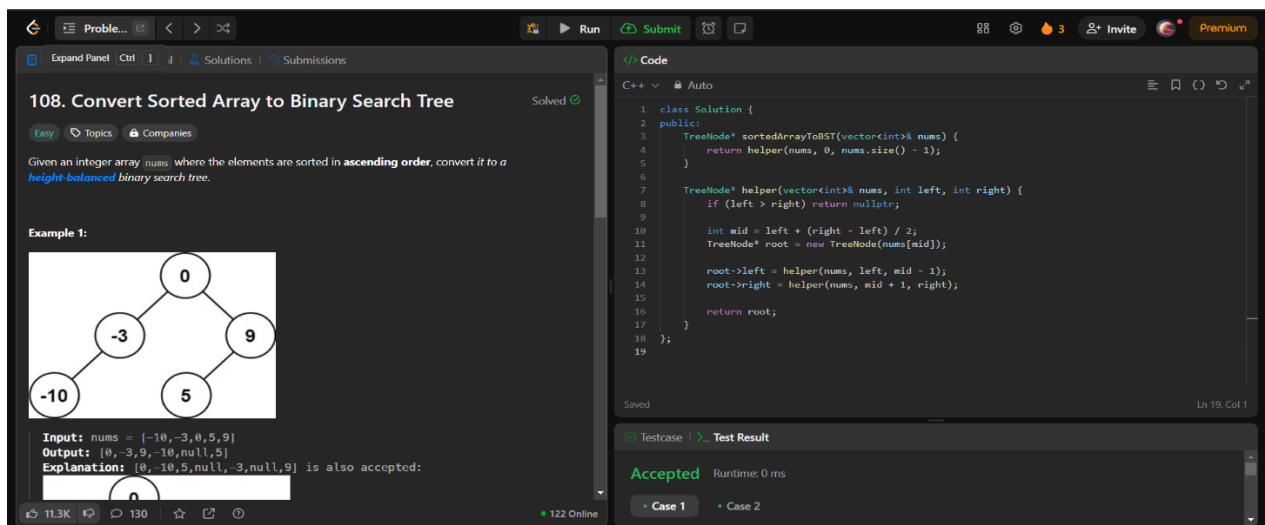
372. [Super Pow](#)

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;  
    }  
};
```



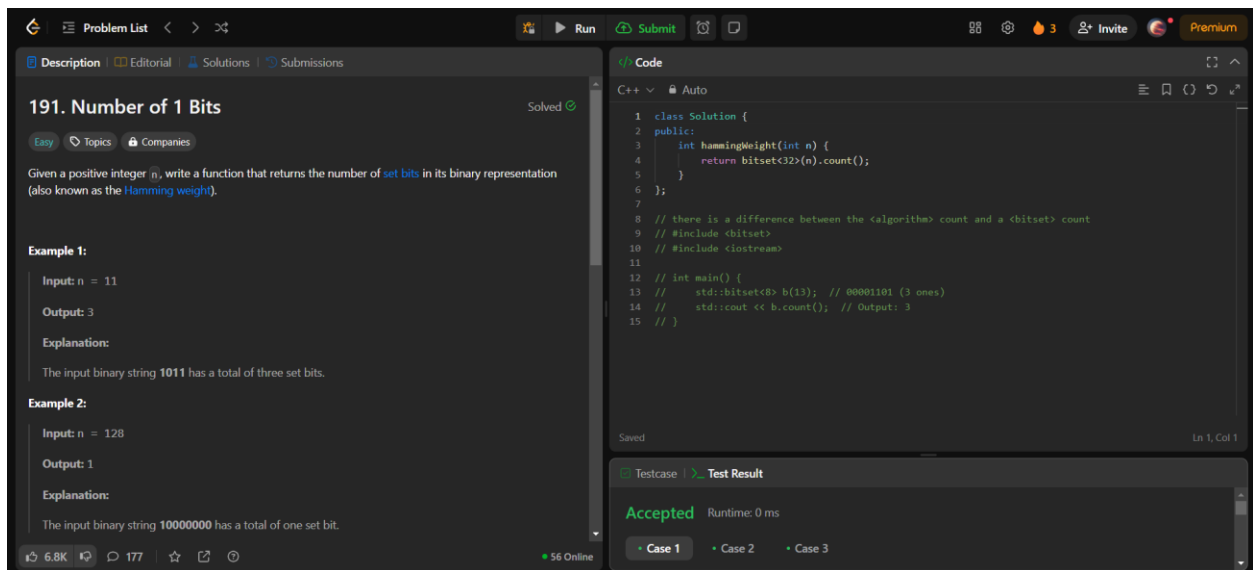
The screenshot displays a coding interface for the problem "108. Convert Sorted Array to Binary Search Tree". The problem description states: "Given an integer array `nums` where the elements are sorted in **ascending order**, convert it to a **height-balanced** binary search tree." An example is provided with input `nums = [-10, -3, 0, 5, 9]` and output `[0, -3, 9, -10, 5]`. The explanation notes that `[0, -10, 5, null, -3, null, 9]` is also accepted. A diagram shows a height-balanced BST with root 0, left child -3, right child 9, and further children -10 and 5. The code editor shows the C++ solution, and the test result indicates it is "Accepted" with a runtime of 0 ms.

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();  
  
    }  
  
};
```



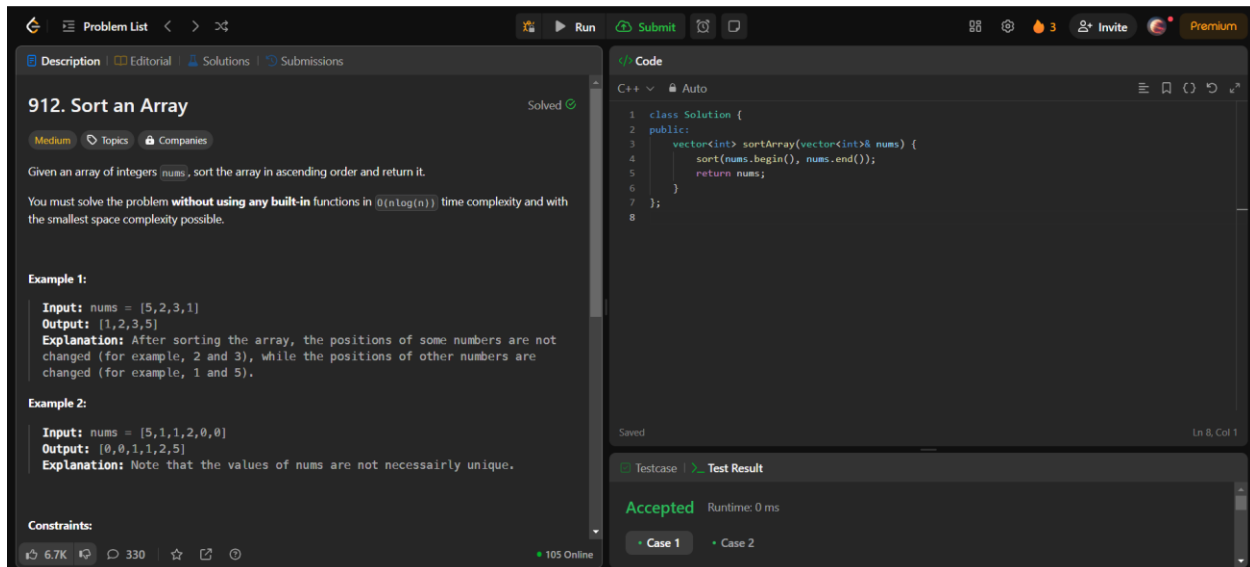
The screenshot displays a coding platform interface for the problem "191. Number of 1 Bits". The left panel shows the problem description, which asks for a function to count the number of set bits (Hamming weight) in the binary representation of a positive integer n . It includes two examples: Example 1 with input $n = 11$ and output 3, and Example 2 with input $n = 128$ and output 1. The right panel shows the C++ code solution, which defines a class `Solution` with a public method `hammingWeight` that returns `bitset<32>(n).count()`. Below the code, the test result is shown as "Accepted" with a runtime of 0 ms. The bottom of the interface shows the number of online users (56) and the number of cases (3).

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;  
  
    }  
  
};
```



The screenshot displays a coding platform interface. On the left, the problem description for '912. Sort an Array' is shown, including input/output examples and constraints. On the right, the C++ code for the solution is displayed, which uses the `sort` function from the `algorithm` library. The code is as follows:

```
1 class Solution {  
2 public:  
3     vector<int> sortArray(vector<int>& nums) {  
4         sort(nums.begin(), nums.end());  
5         return nums;  
6     }  
7 };  
8
```

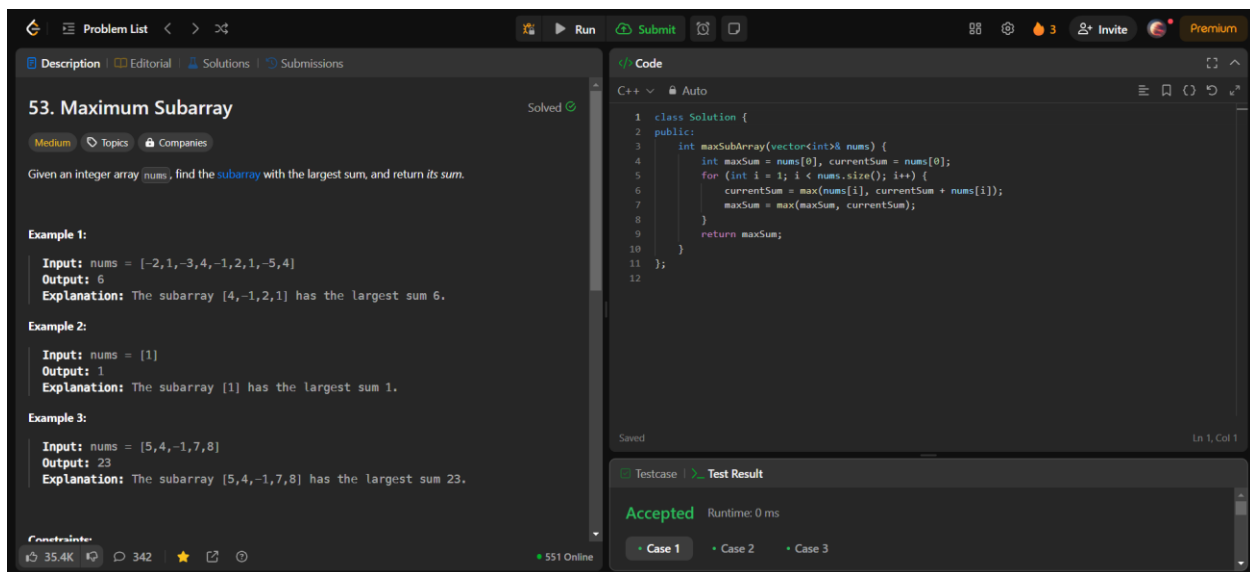
Below the code, the test results are shown, indicating that the solution is 'Accepted' with a runtime of 0 ms.

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 displays a coding interface for the problem "53. Maximum Subarray". The left panel shows the problem description, which asks to find the contiguous subarray with the largest sum. It includes three examples: Example 1 with input [-2,1,-3,4,-1,2,1,-5,4] and output 6; Example 2 with input [1] and output 1; and Example 3 with input [5,4,-1,7,8] and output 23. The right panel shows the C++ code for the solution, which implements the Kadane's algorithm. The code is as follows:

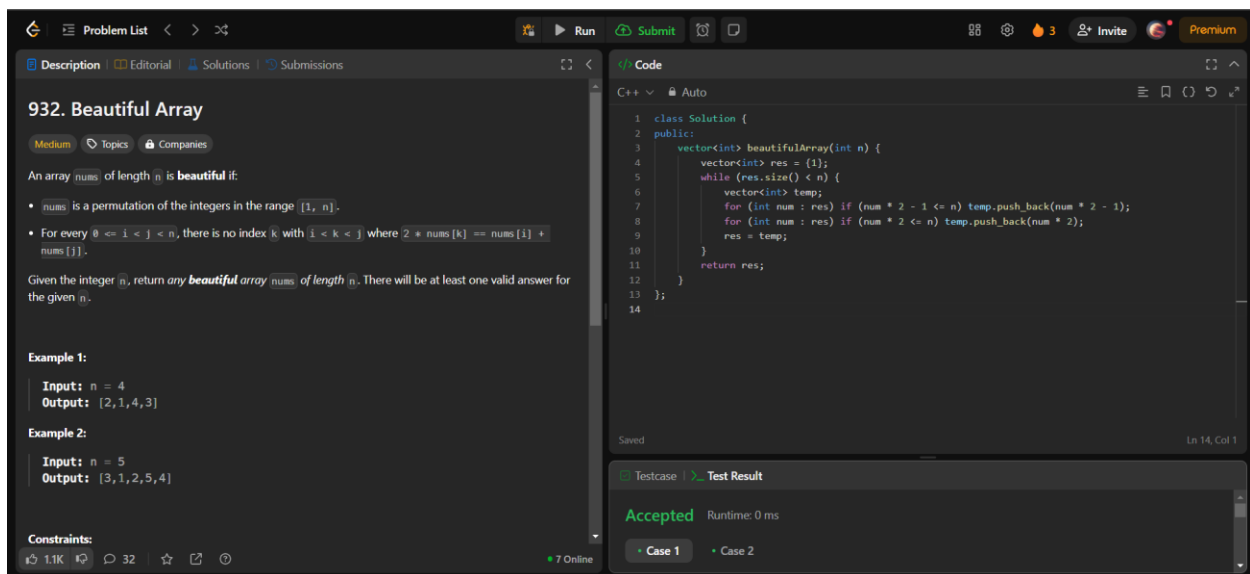
```
1 class Solution {  
2 public:  
3     int maxSubArray(vector<int>& nums) {  
4         int maxSum = nums[0], currentSum = nums[0];  
5         for (int i = 1; i < nums.size(); i++) {  
6             currentSum = max(nums[i], currentSum + nums[i]);  
7             maxSum = max(maxSum, currentSum);  
8         }  
9         return maxSum;  
10    }  
11 };  
12
```

The bottom right panel shows the test result, indicating that the solution is "Accepted" with a runtime of 0 ms. The interface also shows a "Problem List" tab, a "Run" button, and a "Submit" button.

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;  
  
    }  
  
};
```



The screenshot displays a coding interface for the problem "932. Beautiful Array". The left panel shows the problem description, which defines a "beautiful" array as a permutation of integers from 1 to n such that for any two elements, their sum is not equal to the sum of any two other elements. It includes two examples: Example 1 with n=4 and output [2,1,4,3], and Example 2 with n=5 and output [3,1,2,5,4]. The right panel shows the C++ code for the solution, which uses a recursive approach to build the array by interleaving odd and even multiples of the current elements. The bottom panel shows the test results, indicating that the solution is "Accepted" with a runtime of 0 ms.

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

372. Super Pow

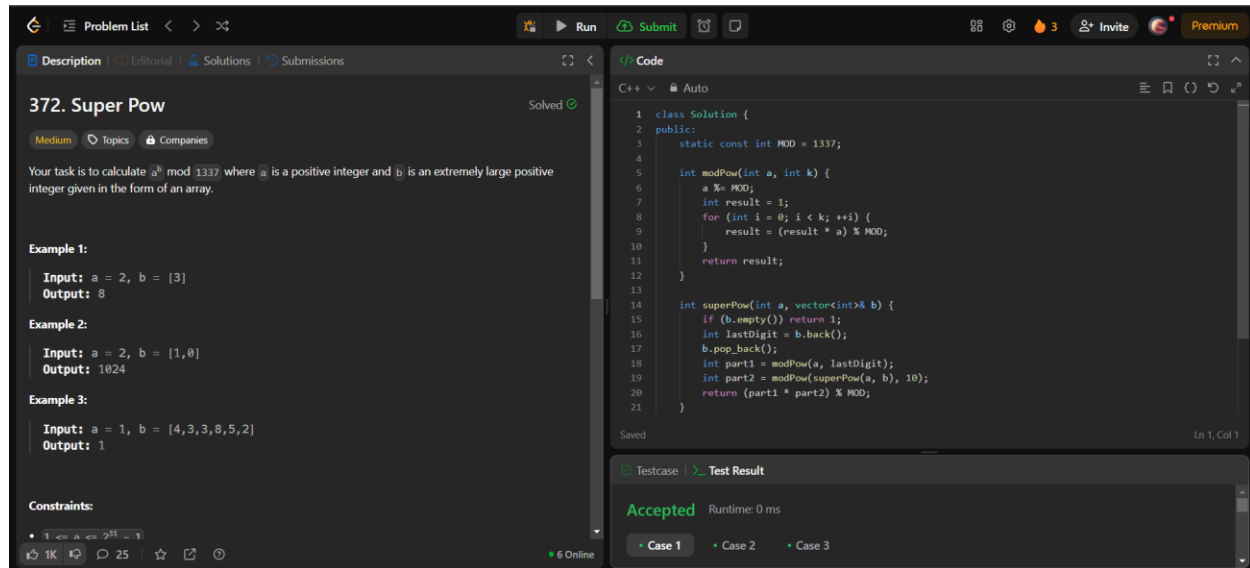
Aim – Compute $a^b \bmod 1337$, where b is an array of digits.

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:
• $1 \leq a < 2^{31}$
• $1 \leq b.length \leq 2000$
• $0 \leq b[i] < 10$
• $0 \leq b[0] < b[1] < \dots < b[b.length - 2] < b[b.length - 1]$

```

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.

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};
    }
}

```



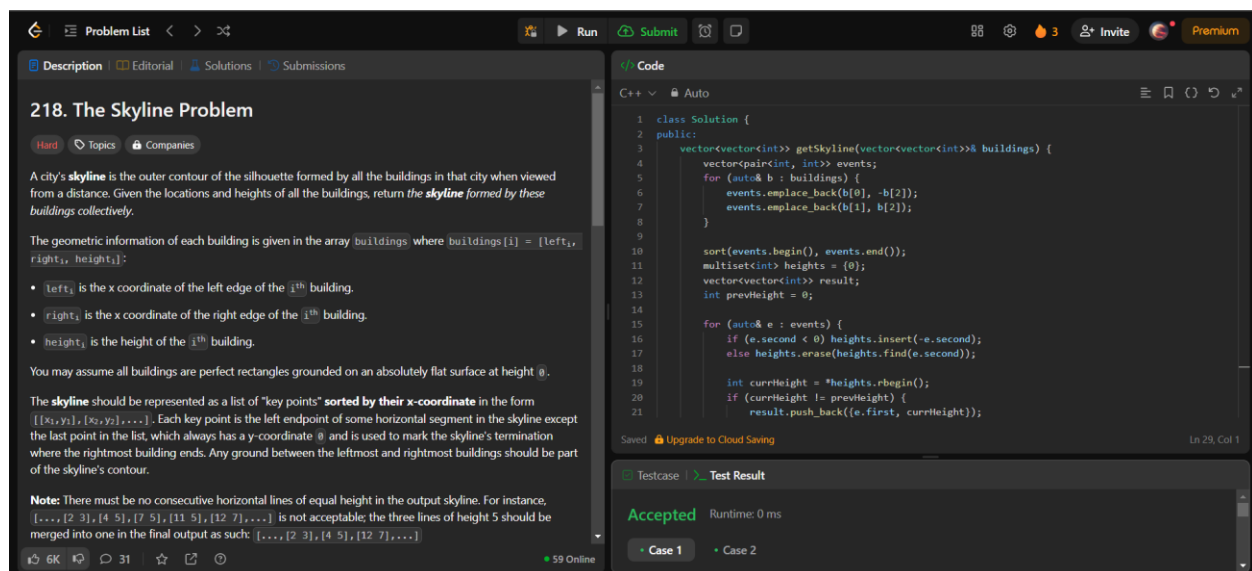
```
vector<vector<int>>> result;

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

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*th building.
- `righti` is the x coordinate of the right edge of the *i*th building.
- `heighti` 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 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], ...]`

```
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});
            }
        }

        return result;
    }
};
```

Accepted Runtime: 0 ms