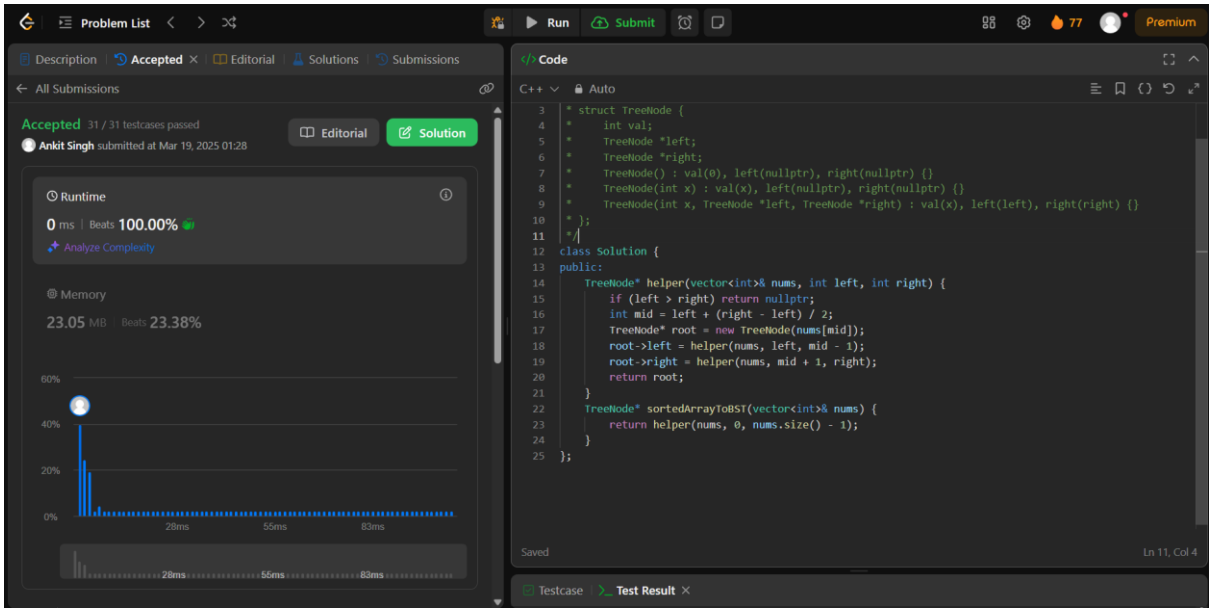
ASSIGNMENT -6 (ADVANCED PROGRAMMING)

Profile: https://leetcode.com/u/AnkitSingh101/

1. **Problem 1:** Convert Sorted Array to Binary Search Tree (108)
2. **Code:**

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

3. **Output:**

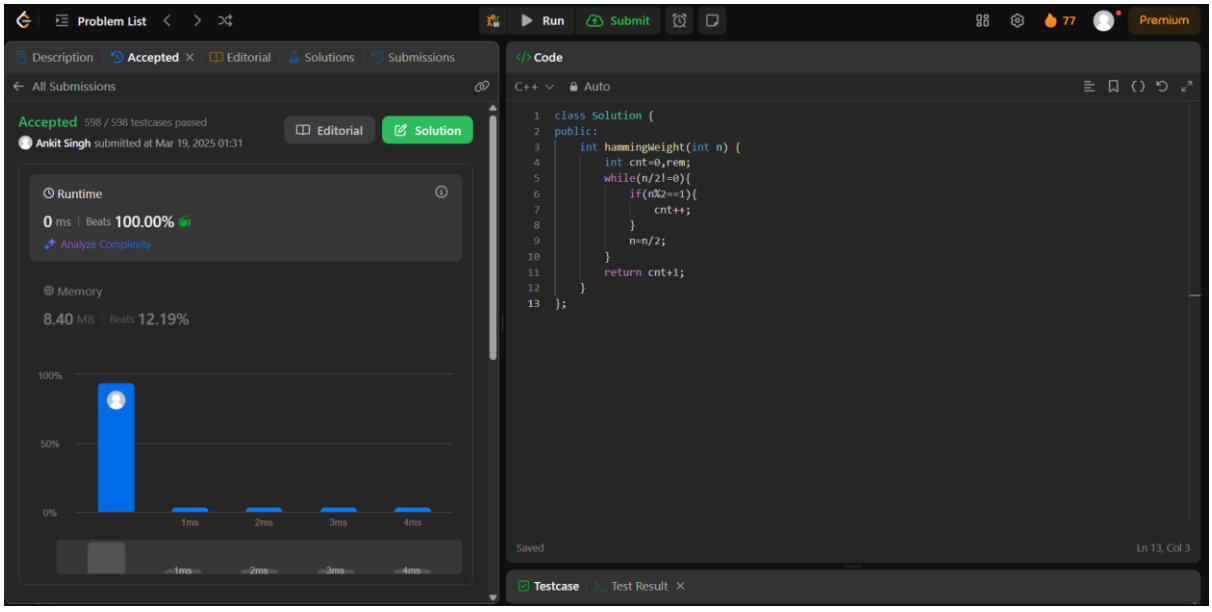


1. **Problem 2: Number of 1 Bits (191)**

2. **Code:**

```
class Solution {
public:
    int hammingWeight(int n) {
        int cnt=0,rem;
        while(n/2!=0){
            if(n%2==1){
                cnt++;
            }
            n=n/2;
        }
        return cnt+1;
    }
};
```

3. **Output:**

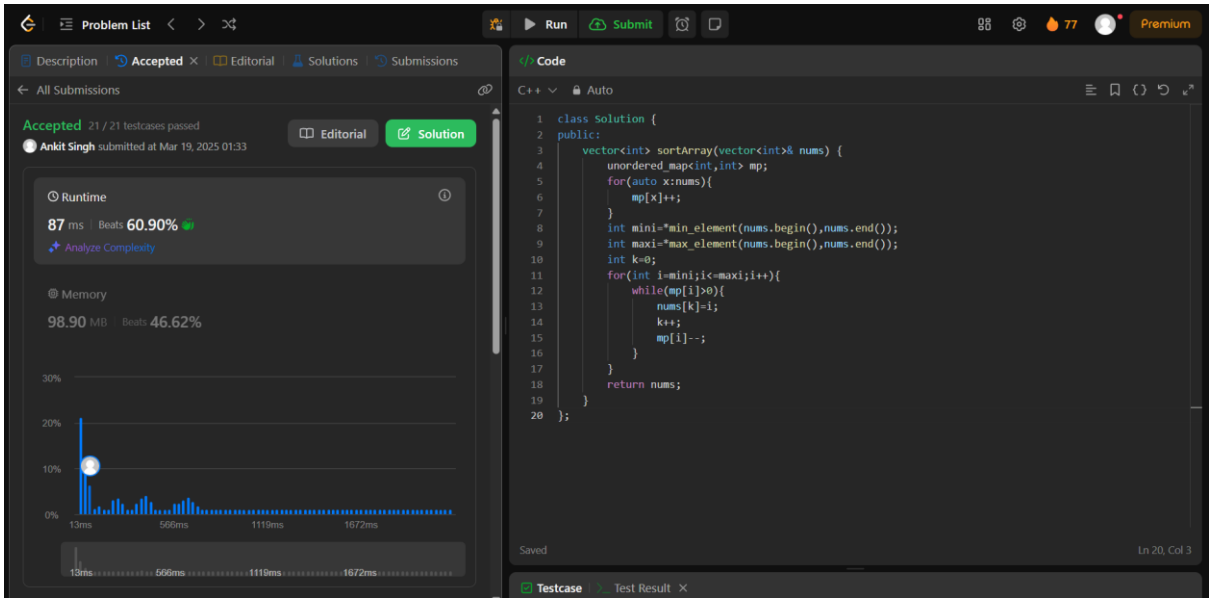


1. **Problem 3: Sort an Array** (912)

2. **Code:**

```
class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        unordered_map<int,int> mp;
        for(auto x:nums){
            mp[x]++;
        }
        int mini=*min_element(nums.begin(),nums.end());
        int maxi=*max_element(nums.begin(),nums.end());
        int k=0;
        for(int i=mini;i<=maxi;i++){
            while(mp[i]>0){
                nums[k]=i;
                k++;
                mp[i]--;
            }
        }
        return nums;
    }
};
```

3. **Output:**

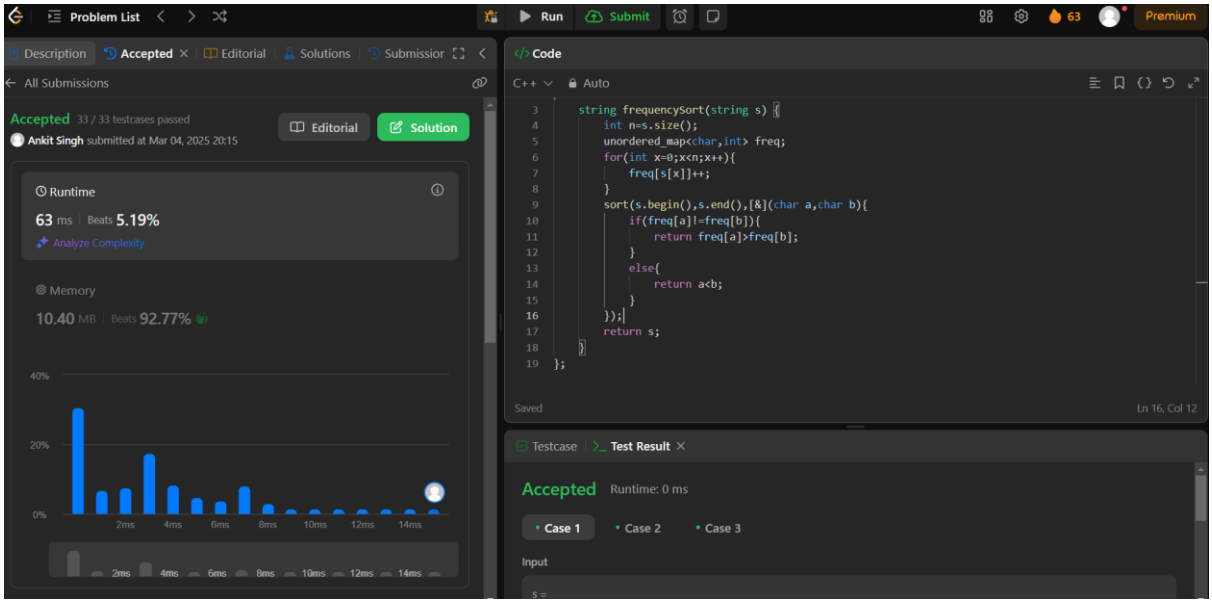


1. **Problem 4: [Maximum Subarray](#) (53)**

2. **Code:**

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

};
```

3. **Output:**

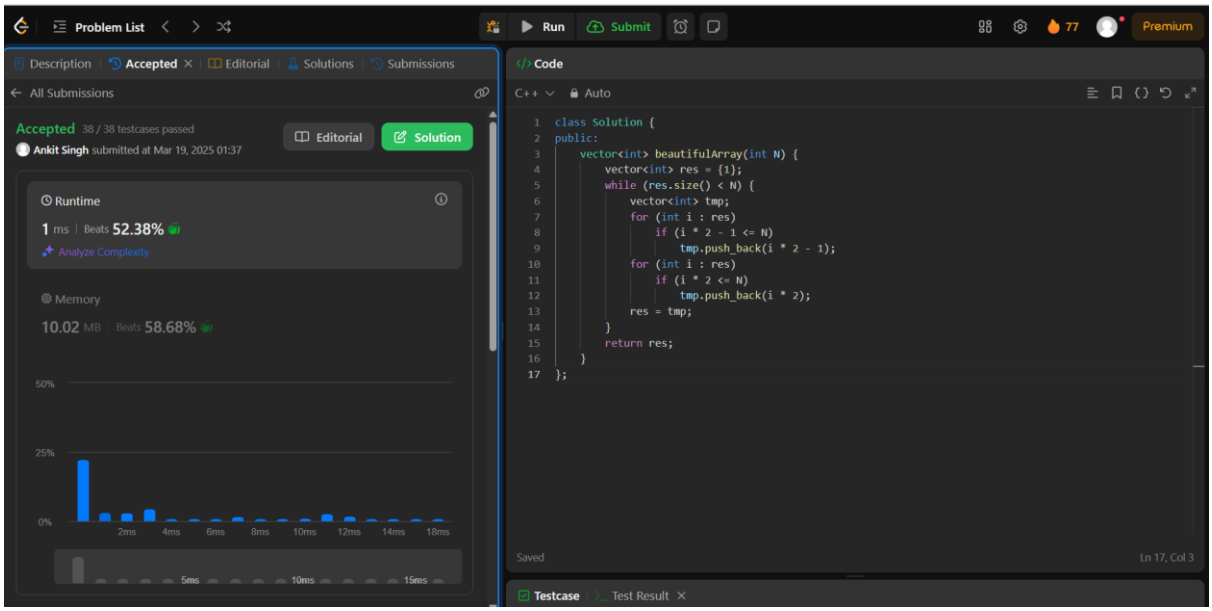


1.Problem 5: Beautiful Array (932)

2. Code:

```
class Solution {
public:
    vector<int> beautifulArray(int N) {
        vector<int> res = {1};
        while (res.size() < N) {
            vector<int> tmp;
            for (int i : res)
                if (i * 2 - 1 <= N)
                    tmp.push_back(i * 2 - 1);
            for (int i : res)
                if (i * 2 <= N)
                    tmp.push_back(i * 2);
            res = tmp;
        }
        return res;
    }
};
```

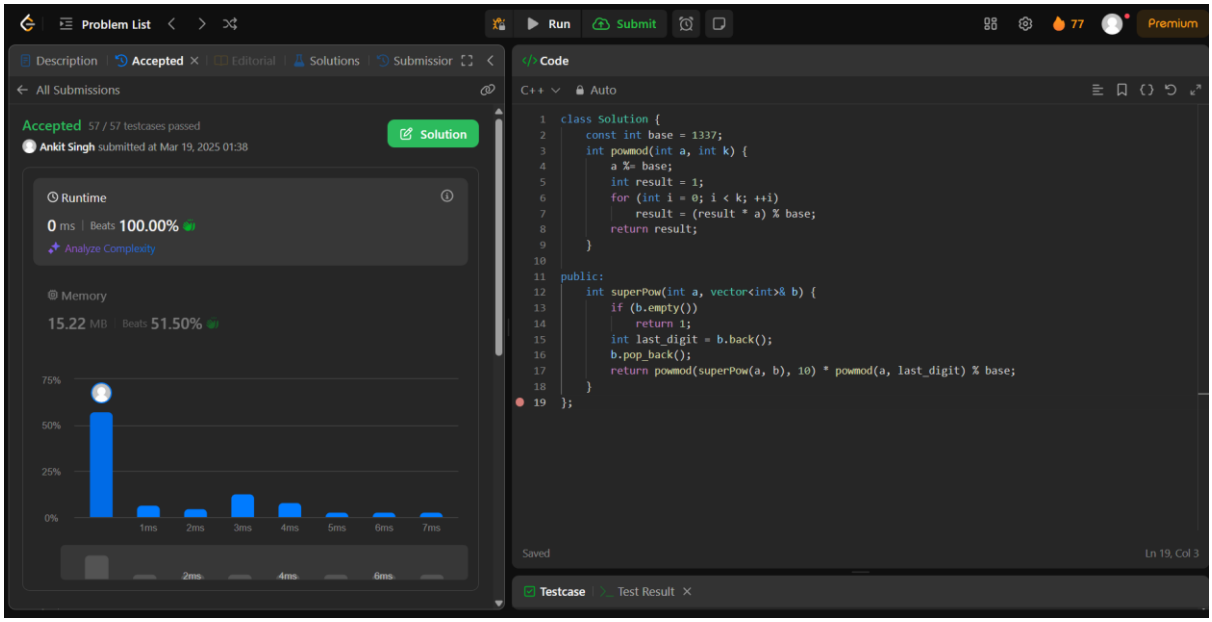
3. Output:



1. **Problem 6: Super Pow (372)**
2. **Code:**

```
class Solution {
    const int base = 1337;
    int powmod(int a, int k) {
        a %= base;
        int result = 1;
        for (int i = 0; i < k; ++i)
            result = (result * a) % base;
        return result;
    }

public:
    int superPow(int a, vector<int>& b) {
        if (b.empty())
            return 1;
        int last_digit = b.back();
        b.pop_back();
        return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;
    }
};
```

3. **Output:**

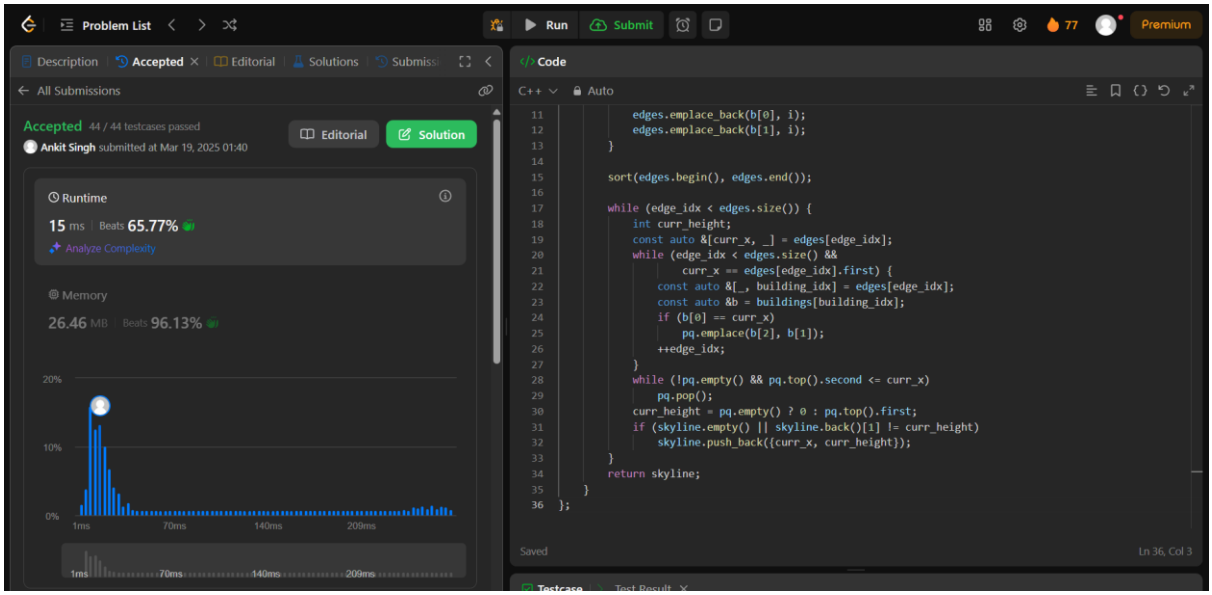


1. **Problem 7: The Skyline Problem (218)**

2. **Code:**

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        int edge_idx = 0;
        vector<pair<int, int>> edges;
        priority_queue<pair<int, int>> pq;
        vector<vector<int>> skyline;

        for (int i = 0; i < buildings.size(); ++i) {
            const auto &b = buildings[i];
            edges.emplace_back(b[0], i);
            edges.emplace_back(b[1], i);
        }

        sort(edges.begin(), edges.end());

        while (edge_idx < edges.size()) {
            int curr_height;
            const auto &[curr_x, _] = edges[edge_idx];

            while (edge_idx < edges.size() &&
                    curr_x == edges[edge_idx].first) {
                const auto &[_, building_idx] = edges[edge_idx];
                const auto &b = buildings[building_idx];
                if (b[0] == curr_x)
                    pq.emplace(b[2], b[1]);
                ++edge_idx;
            }

            while (!pq.empty() && pq.top().second <= curr_x)
                pq.pop();
            curr_height = pq.empty() ? 0 : pq.top().first;

            if (skyline.empty() || skyline.back()[1] != curr_height)
                skyline.push_back({curr_x, curr_height});
        }
        return skyline;
    }
};
```

3. Output:


Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
All Submissions
Accepted 44 / 44 testcases passed
Ankit Singh submitted at Mar 19, 2025 01:40
Editorial
Solution
Runtime
15 ms | Beats 65.77%
Analyze Complexity
Memory
26.46 MB | Beats 96.13%
20%
10%
0%
1ms
70ms
140ms
209ms
Code
C++
Auto
edges.emplace_back(b[0], i);
edges.emplace_back(b[1], i);
}
sort(edges.begin(), edges.end());
while (edge_idx < edges.size()) {
int curr_height;
const auto &[curr_x, _] = edges[edge_idx];
while (edge_idx < edges.size() &&
curr_x == edges[edge_idx].first) {
const auto &[_, building_idx] = edges[edge_idx];
const auto &b = buildings[building_idx];
if (b[0] == curr_x)
pq.emplace(b[2], b[1]);
++edge_idx;
}
while (!pq.empty() && pq.top().second <= curr_x)
pq.pop();
curr_height = pq.empty() ? 0 : pq.top().first;
if (skyline.empty() || skyline.back()[1] != curr_height)
skyline.push_back({curr_x, curr_height});
}
return skyline;
}
};
Saved
Ln 36, Col 3