

AP Assignment 6

**aliza asif**

22BCS50175

22BCS\_FL\_IOT\_601\_A

**AP ASSIGNMENT 6**

**Q1.** **Convert sorted array to binary search tree**

Implementation Code:

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\* node = new TreeNode(nums[mid]);

node->left = helper(nums, left, mid - 1);

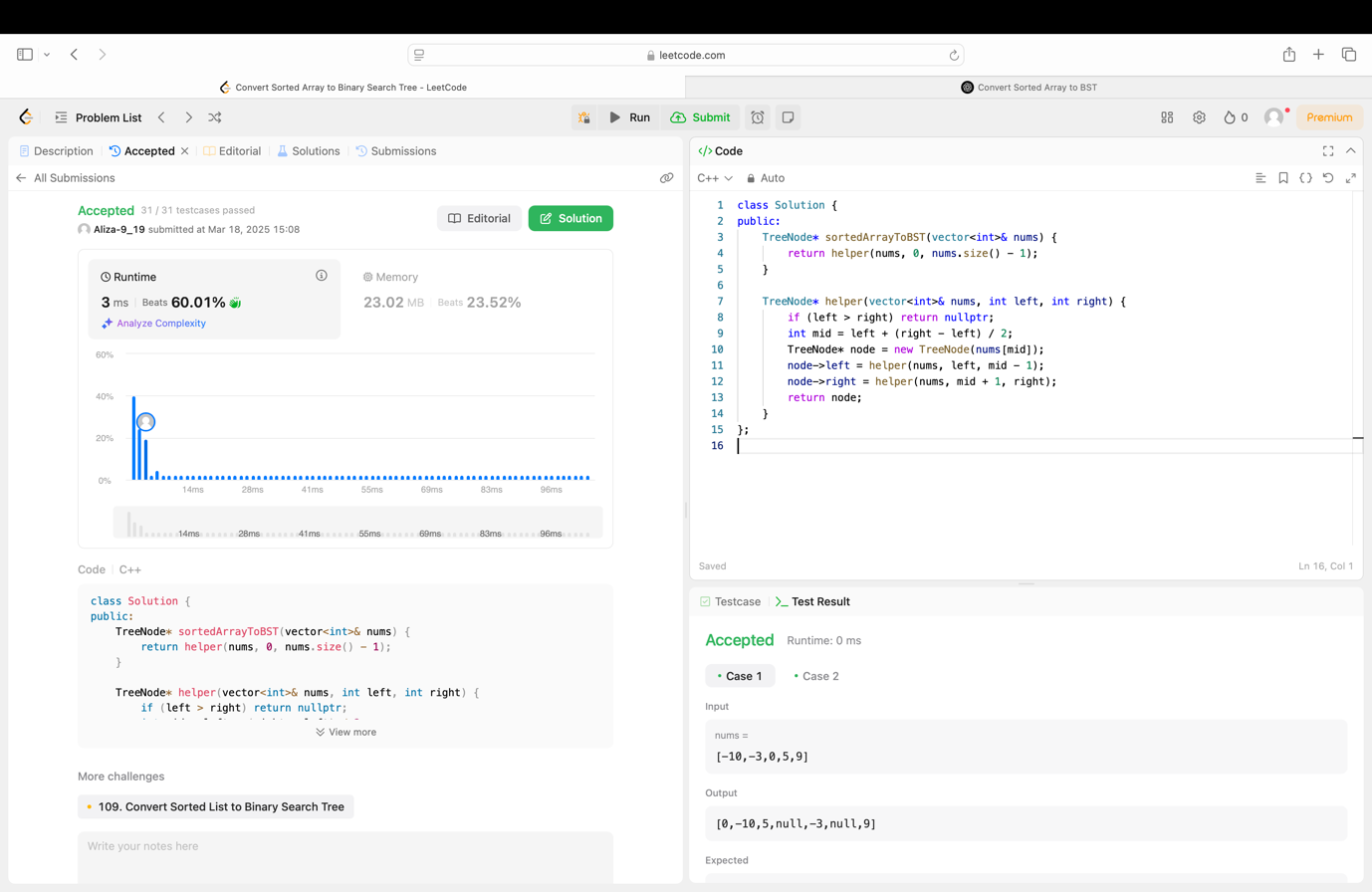
node->right = helper(nums, mid + 1, right);

return node;

}

};

Output:



**Q2. Number of 1 Bits**

Implementation Code:

class Solution {

public:

int hammingWeight(int n) {

int count = 0;

while (n != 0) {

count += n & 1;

n >>= 1;

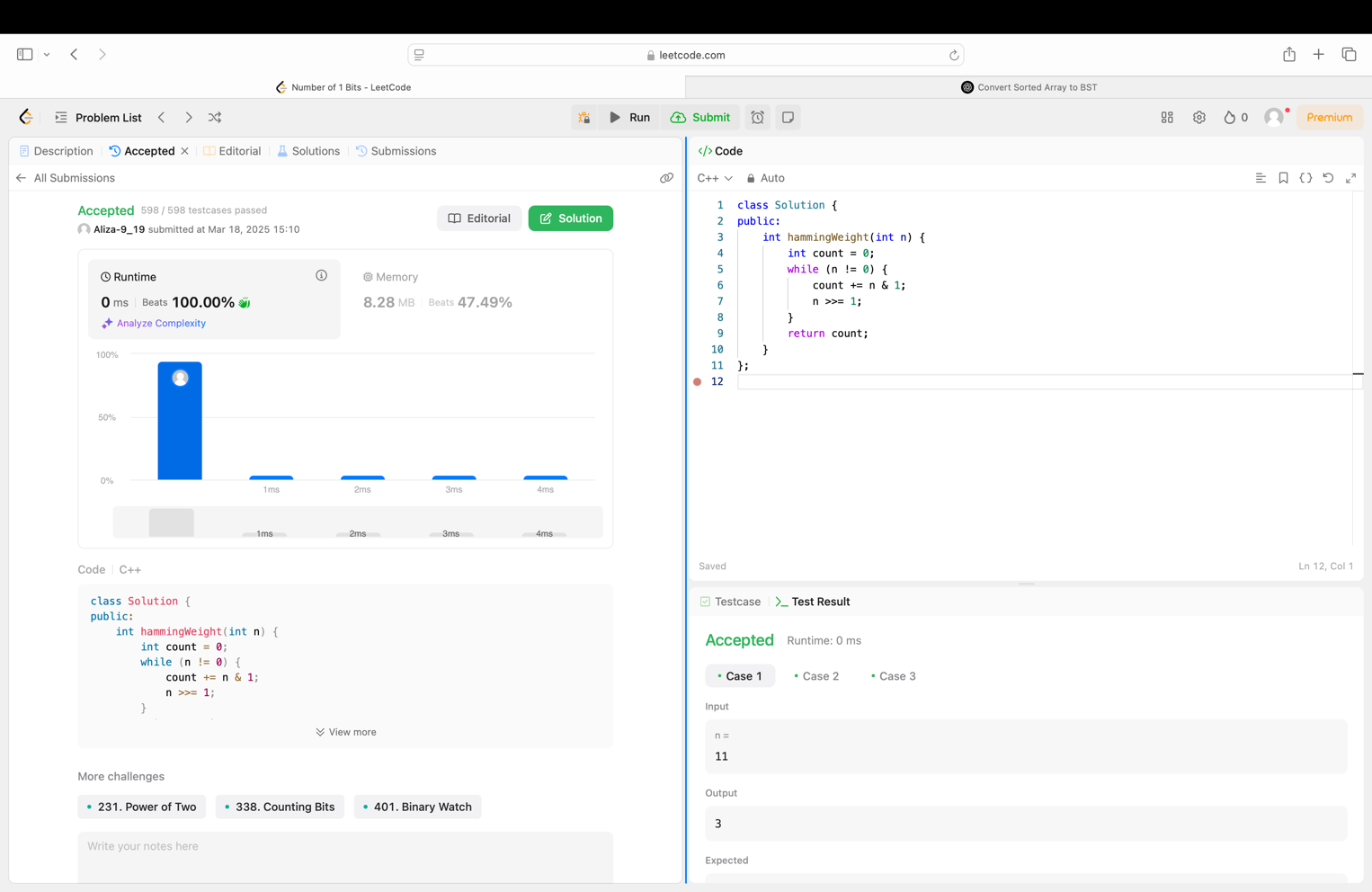
}

return count;

}

};

Output:



**Q3. Sort an array**

Implementation Code:

class Solution {

public:

vector<int> sortArray(vector<int>& nums) {

mergeSort(nums, 0, nums.size() - 1);

return nums;

}

void mergeSort(vector<int>& nums, int left, int right) {

if (left >= right) return;

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

mergeSort(nums, left, mid);

mergeSort(nums, mid + 1, right);

merge(nums, left, mid, right);

}

void merge(vector<int>& nums, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> leftArr(n1), rightArr(n2);

for (int i = 0; i < n1; i++) leftArr[i] = nums[left + i];

for (int i = 0; i < n2; i++) rightArr[i] = nums[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (leftArr[i] <= rightArr[j]) {

nums[k++] = leftArr[i++];

} else {

nums[k++] = rightArr[j++];

}

}

while (i < n1) {

nums[k++] = leftArr[i++];

}

while (j < n2) {

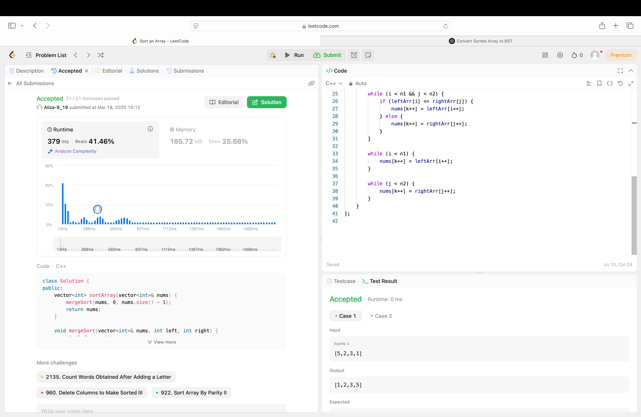
nums[k++] = rightArr[j++];

}

}

};

Output:



**Q4. Maximum Subarray**

Implementation 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);

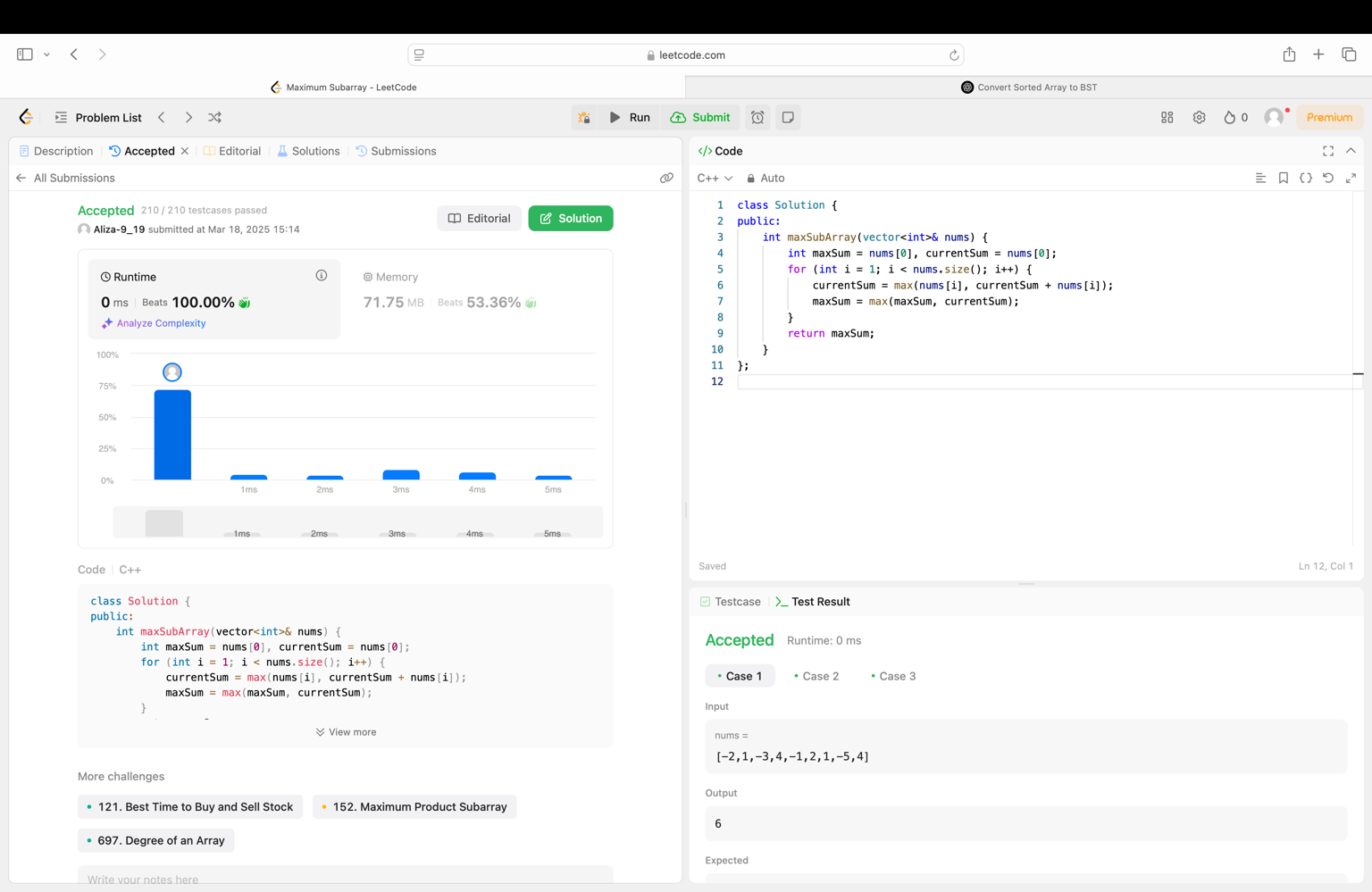
}

return maxSum;

}

};

Output:



**Q5. Beautiful array**

Implementation Code:

class Solution {

public:

vector<int> beautifulArray(int n) {

vector<int> result = {1};

while (result.size() < n) {

vector<int> temp;

for (int x : result) {

if (x \* 2 - 1 <= n) temp.push\_back(x \* 2 - 1);

}

for (int x : result) {

if (x \* 2 <= n) temp.push\_back(x \* 2);

}

result = temp;

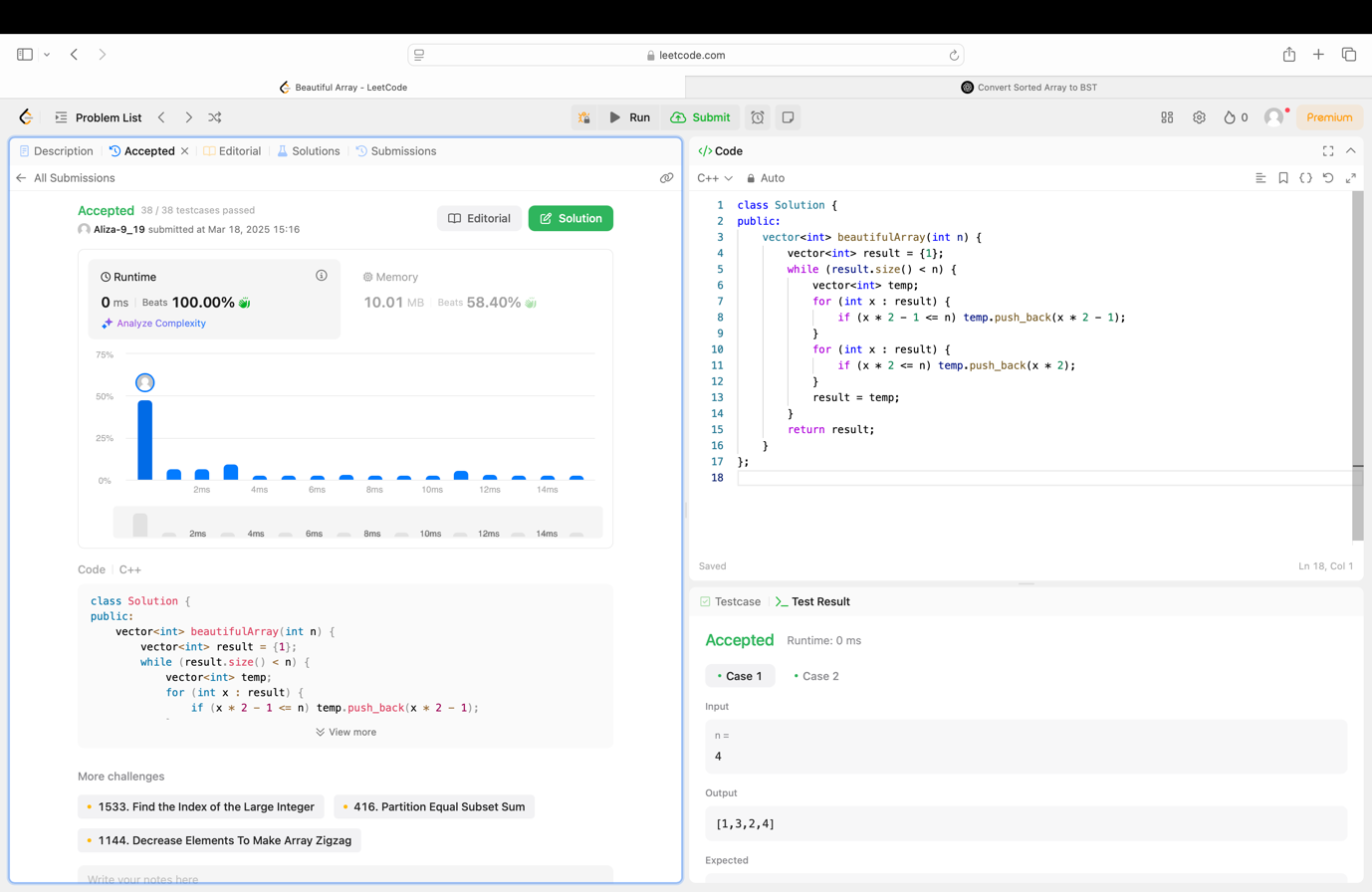
}

return result;

}

};

Output:



**Q6. Super Pow**

Implementation Code:

class Solution {

public:

int modPow(int a, int b, int mod) {

int result = 1;

a %= mod;

while (b > 0) {

if (b % 2 == 1)

result = (result \* a) % mod;

a = (a \* a) % mod;

b /= 2;

}

return result;

}

int superPow(int a, vector<int>& b) {

int mod = 1337;

int result = 1;

for (int i = 0; i < b.size(); i++) {

result = modPow(result, 10, mod) \* modPow(a, b[i], mod) % mod;

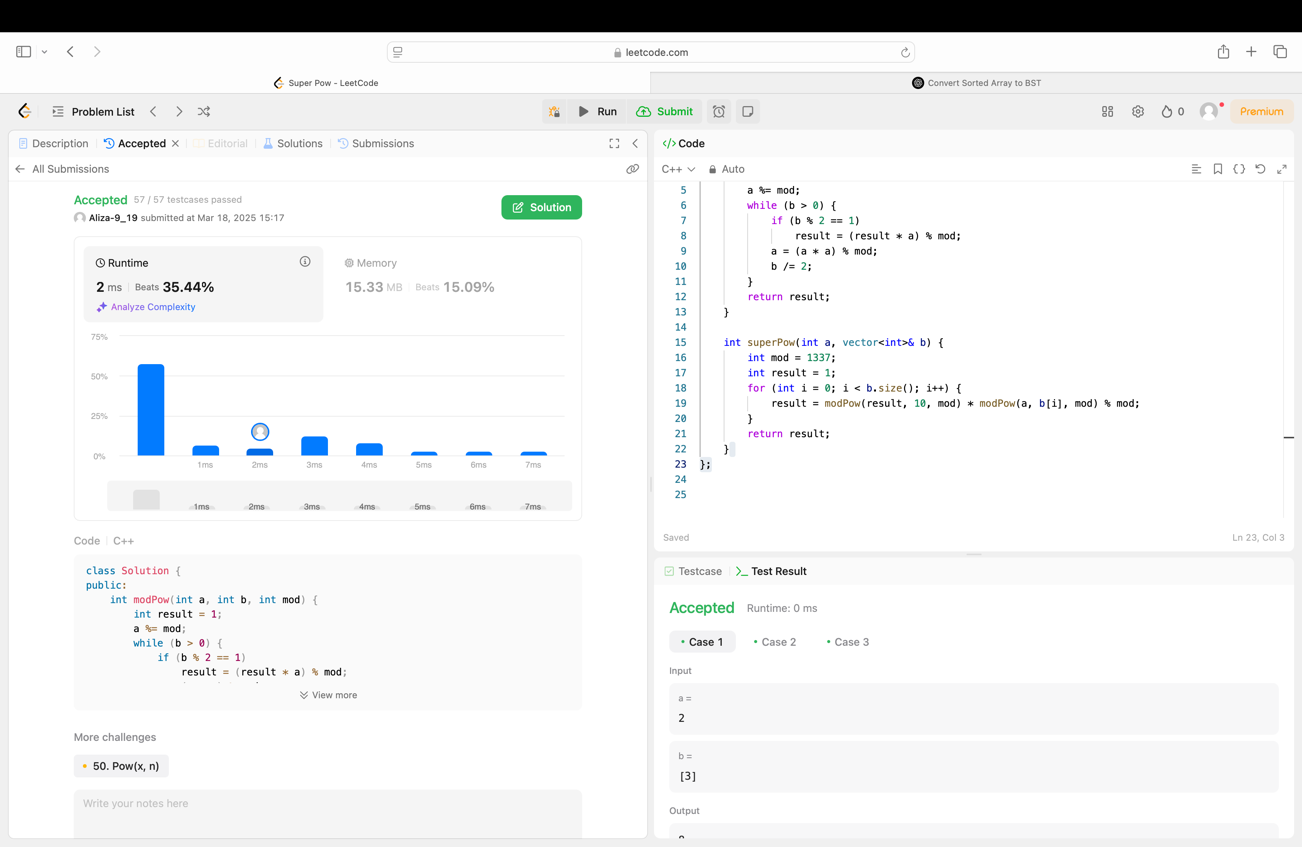
}

return result;

}

};

Output:



**Q7. The Skyline Problem**

Implementation Code:

class Solution {

public:

vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

vector<pair<int, int>> events;

vector<vector<int>> result;

for (auto& b : buildings) {

events.push\_back({b[0], -b[2]});

events.push\_back({b[1], b[2]});

}

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

multiset<int> heights({0});

int prevMaxHeight = 0;

for (auto& event : events) {

int x = event.first;

int h = event.second;

if (h < 0) {

heights.insert(-h);

} else {

heights.erase(heights.find(h));

}

int currMaxHeight = \*heights.rbegin();

if (currMaxHeight != prevMaxHeight) {

result.push\_back({x, currMaxHeight});

prevMaxHeight = currMaxHeight;

}

}

return result;

}

};

Output:

