**ADVANCED PROGRAMMING LAB-2 ASSIGNMENT**

**Submitted By:**

**Name- Anshika Yadav**

**UID- 22BCS10051**

**Section-22BCS\_IOT\_605-B**

1. **Convert sorted array to 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]);

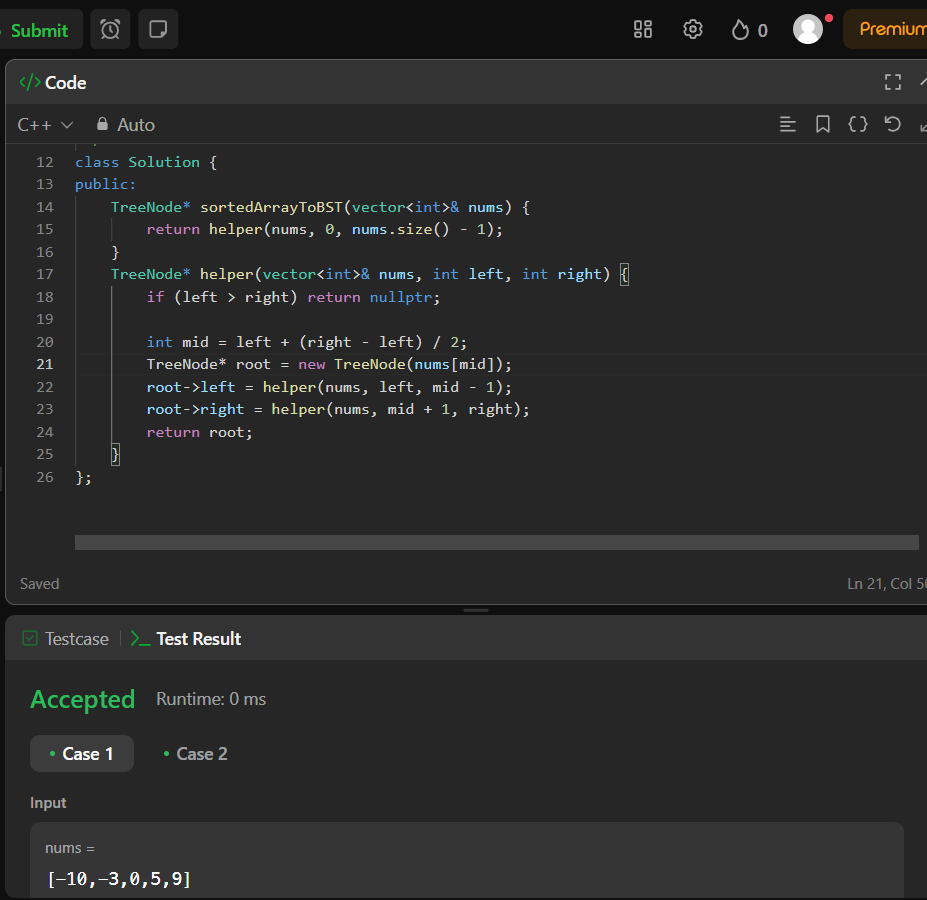
root->left = helper(nums, left, mid - 1);

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

return root;

}

};



1. **Number of 1 bits**

class Solution {

public:

int hammingWeight(int n) {

int count = 0;

while (n) {

count += (n & 1);

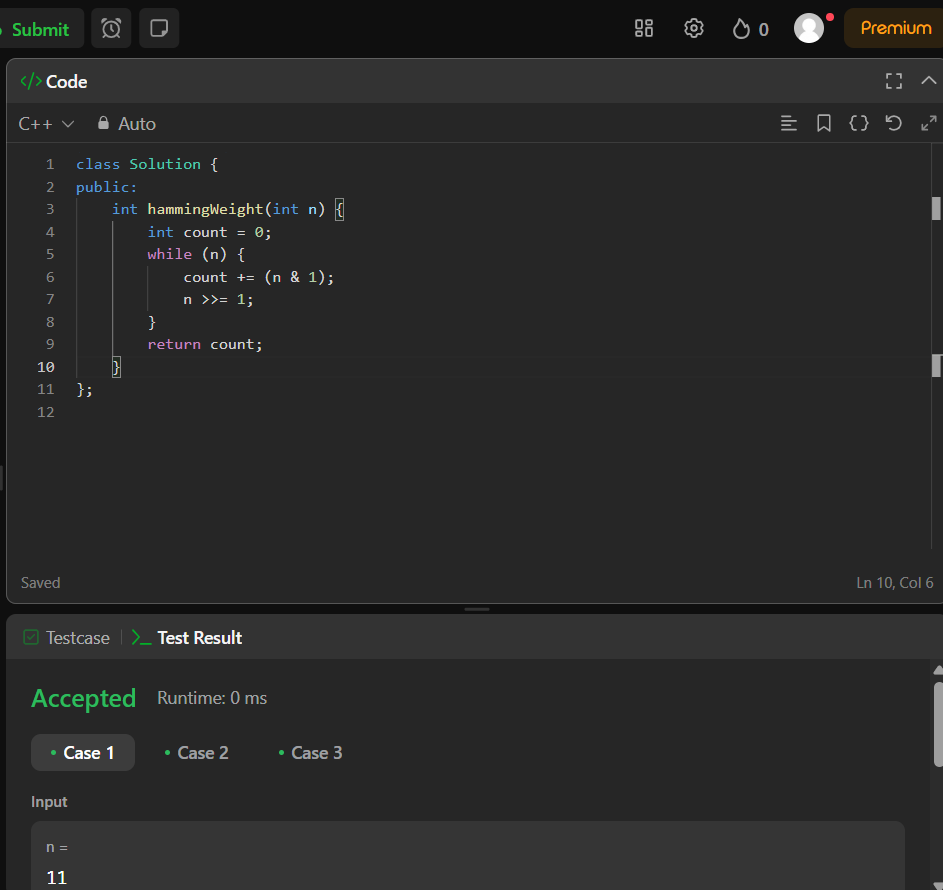
n >>= 1;

}

return count;

}

};



1. **Sort an array**

**class Solution {**

**public:**

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

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

**return nums;**

**}**

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

**vector<int> temp;**

**int i = left, j = mid + 1;**

**while (i <= mid && j <= right) {**

**if (nums[i] <= nums[j]) {**

**temp.push\_back(nums[i++]);**

**} else {**

**temp.push\_back(nums[j++]);**

**}**

**}**

**while (i <= mid) temp.push\_back(nums[i++]);**

**while (j <= right) temp.push\_back(nums[j++]);**

**for (int k = left; k <= right; ++k) {**

**nums[k] = temp[k - left];**

**}**

**}**

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

**if (left >= right) return;**

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

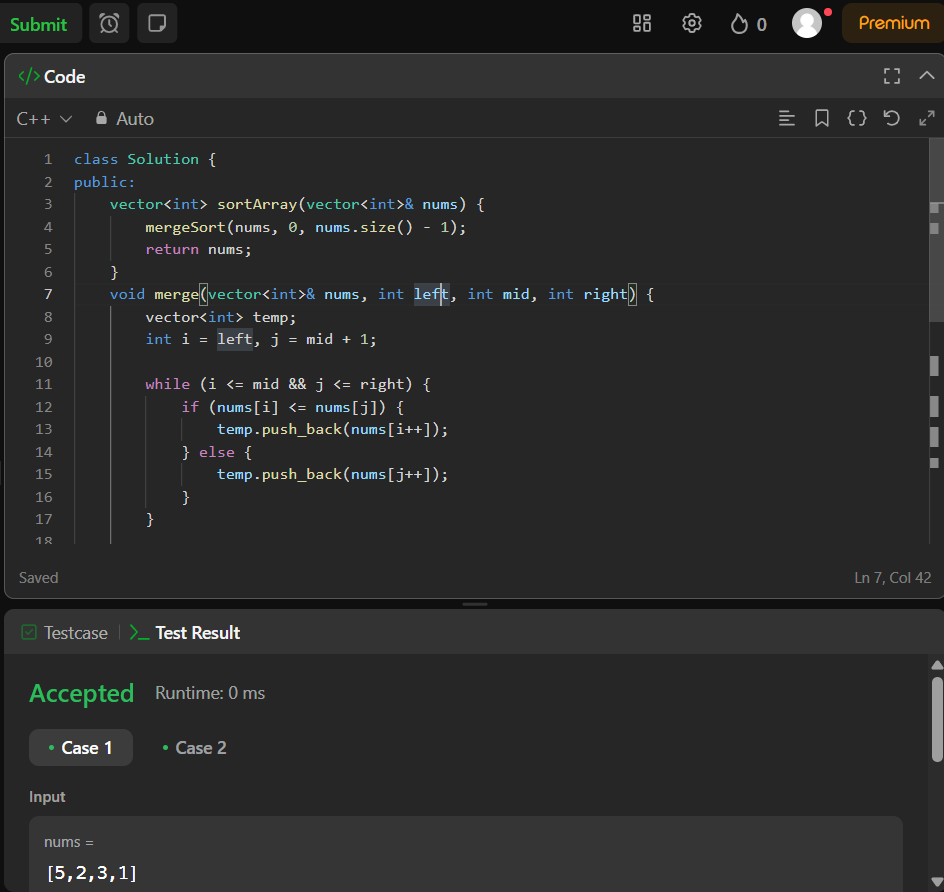
**mergeSort(nums, left, mid);**

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

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

**}**

**};**

****

1. **Maximum subarray**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int currentSum = nums[0];

int maxSum = nums[0];

for (int i = 1; i < nums.size(); i++) {

currentSum = max(nums[i], currentSum + nums[i]);

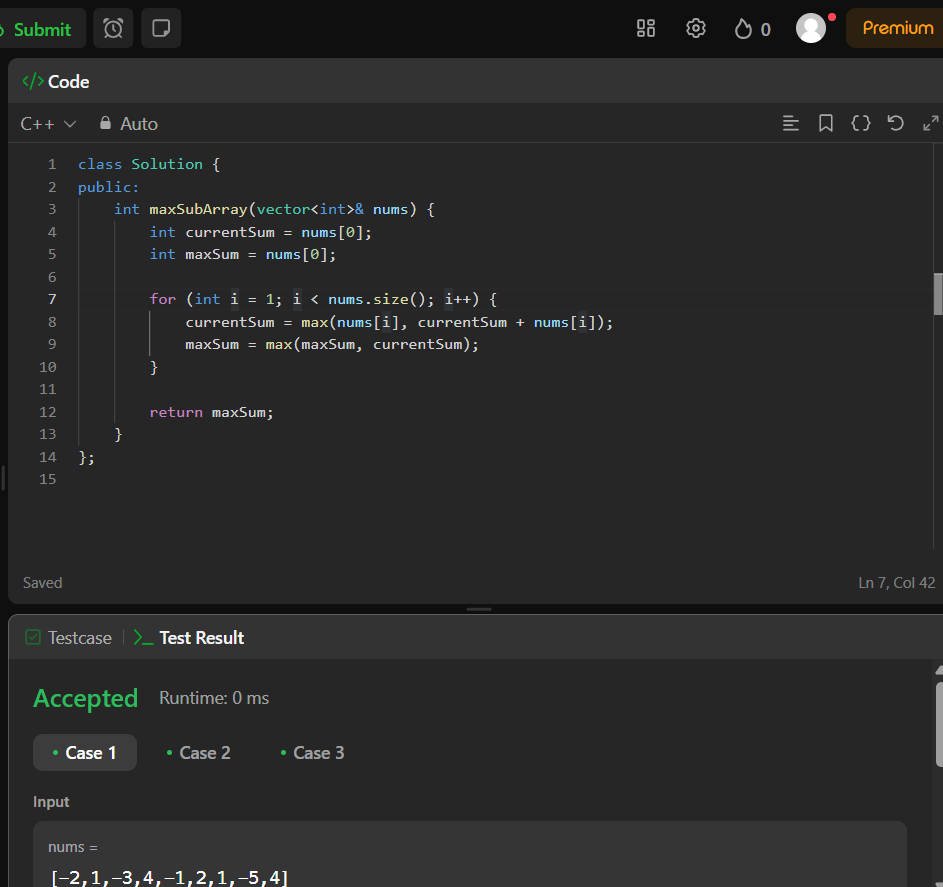
maxSum = max(maxSum, currentSum);

}

return maxSum;

}

};



1. **Beautiful array**

class Solution {

public:

vector<int> beautifulArray(int n) {

vector<int> result = {1};

while (result.size() < n) {

vector<int> next;

for (int num : result) {

if (num \* 2 - 1 <= n) {

next.push\_back(num \* 2 - 1);

}

}

for (int num : result) {

if (num \* 2 <= n) {

next.push\_back(num \* 2);

}

}

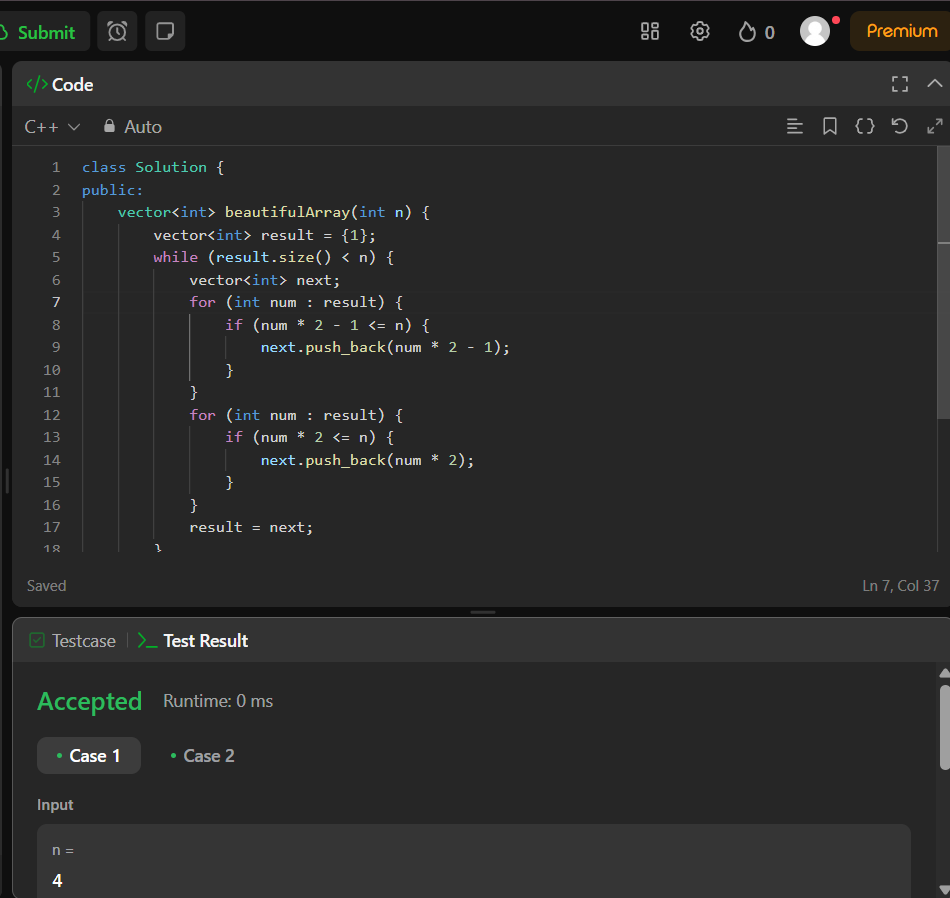
result = next;

}

return result;

}

};



1. **Super pow**

class Solution {

public:

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

int mod = 1337;

a = a % mod;

int result = 1;

for (int i = b.size() - 1; i >= 0; --i) {

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

a = modPow(a, 10, mod);

}

return result;

}

int modPow(int x, int y, int mod) {

int res = 1;

x = x % mod;

while (y > 0) {

if (y % 2 == 1) {

res = (res \* x) % mod;

}

x = (x \* x) % mod;

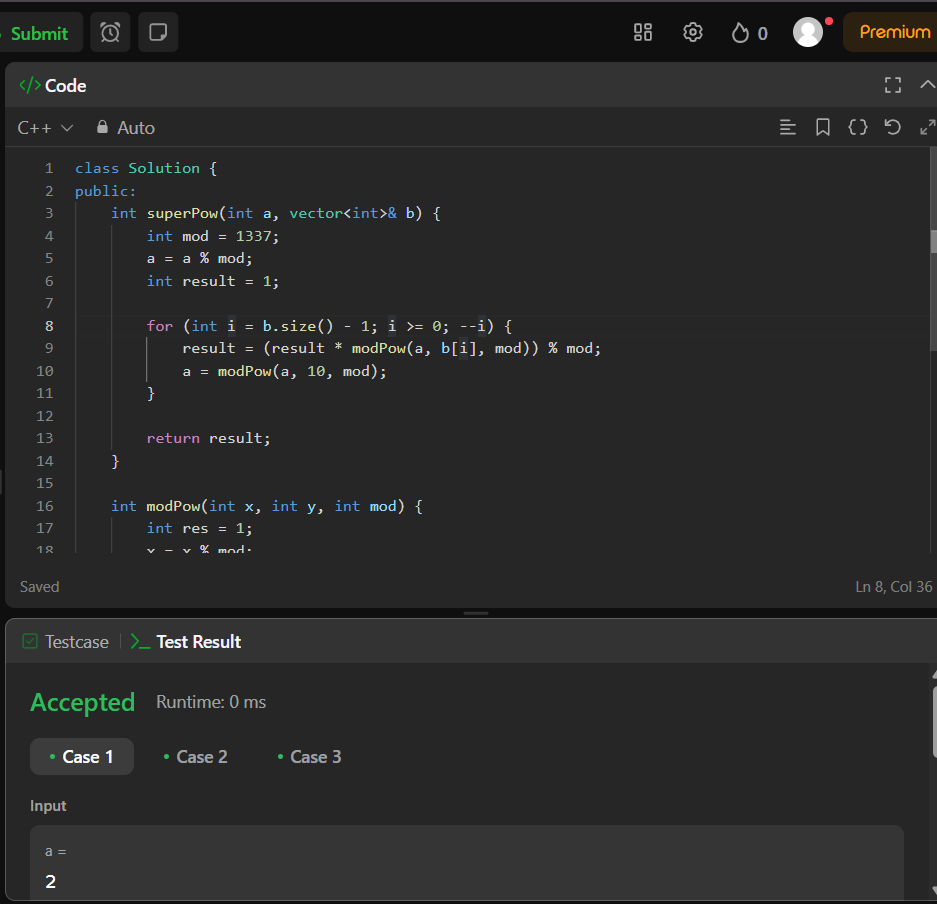
y = y / 2;

}

return res;

}

};



1. **The skyline problem**

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>> skyline;

int prevMaxHeight = 0;

for (auto& [x, h] : events) {

if (h < 0) {

heights.insert(-h);

} else {

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

}

int currentMaxHeight = \*heights.rbegin();

if (currentMaxHeight != prevMaxHeight) {

skyline.push\_back({x, currentMaxHeight});

prevMaxHeight = currentMaxHeight;

}

}

return skyline;

}

};

