**Name:** Aditya Singh

**UID:** 22BCS15063

**Section:** FL\_IOT\_601 - A

**Assignment – 6 Solutions:-**

1. **Convert Sorted Array to Binary Search Tree:**

/\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

TreeNode\* BST(vector<int> &nums,int s, int e){

if(s>e) return NULL;

int mid=(s+e)/2;

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

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

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

return node;

}

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

if(nums.size()==0) return NULL;

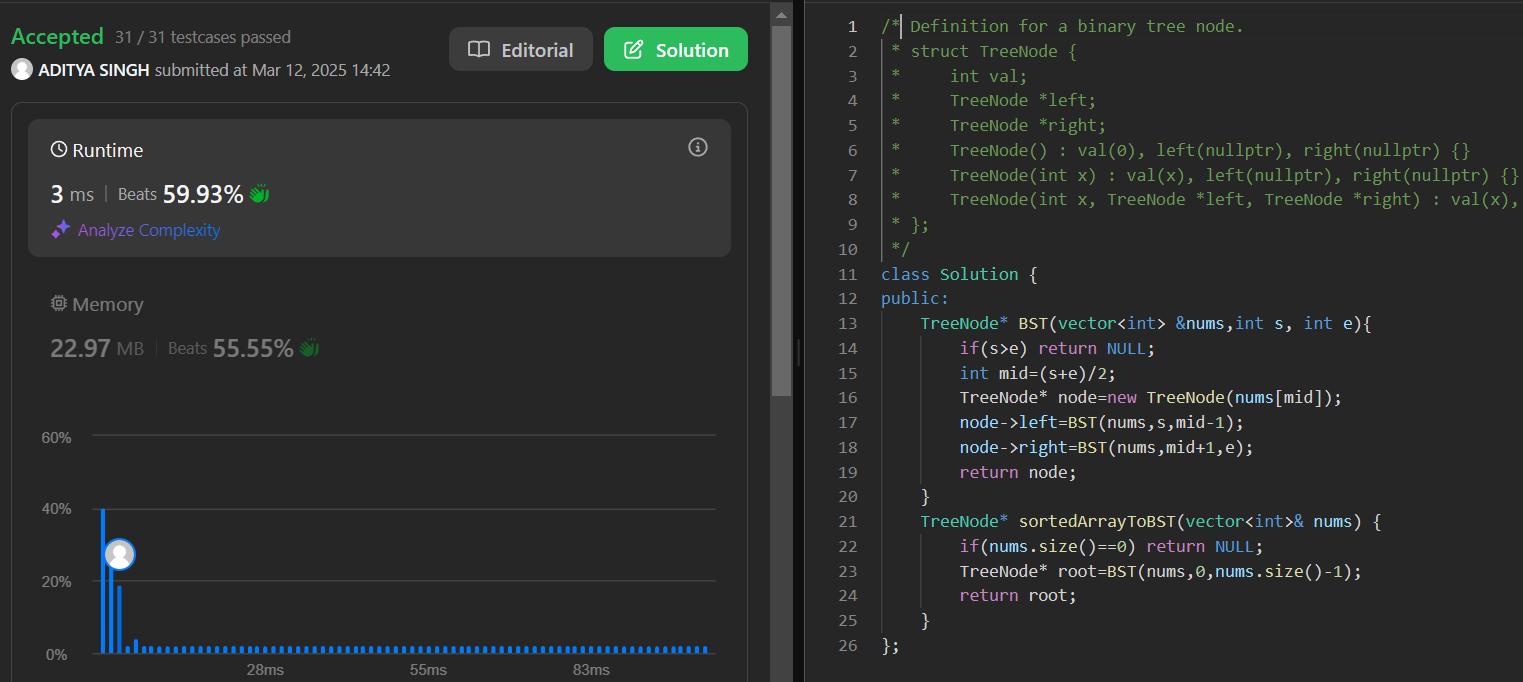
TreeNode\* root=BST(nums,0,nums.size()-1);

return root;

}

};

Result:-



1. **Number of 1 Bits:**

class Solution {

public int hammingWeight(int n) {

int count = 0;

while(n != 0){

n &= (n-1);

count++;

}

return count;

}

}

Result:

A screenshot of a computer

AI-generated content may be incorrect.

1. **Sort an Array:**

class Solution {

public:

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

int mini = \*min\_element(nums.begin(),nums.end());

int maxi = \*max\_element(nums.begin(),nums.end());

int range = maxi-mini+1;

vector<int> cnt(range,0);

for(auto ele : nums){

cnt[ele-mini]++;

}

int curr = 0;

for(int i=0;i<range;i++){

while(cnt[i] > 0){

nums[curr++] = i+mini;

cnt[i]--;

}

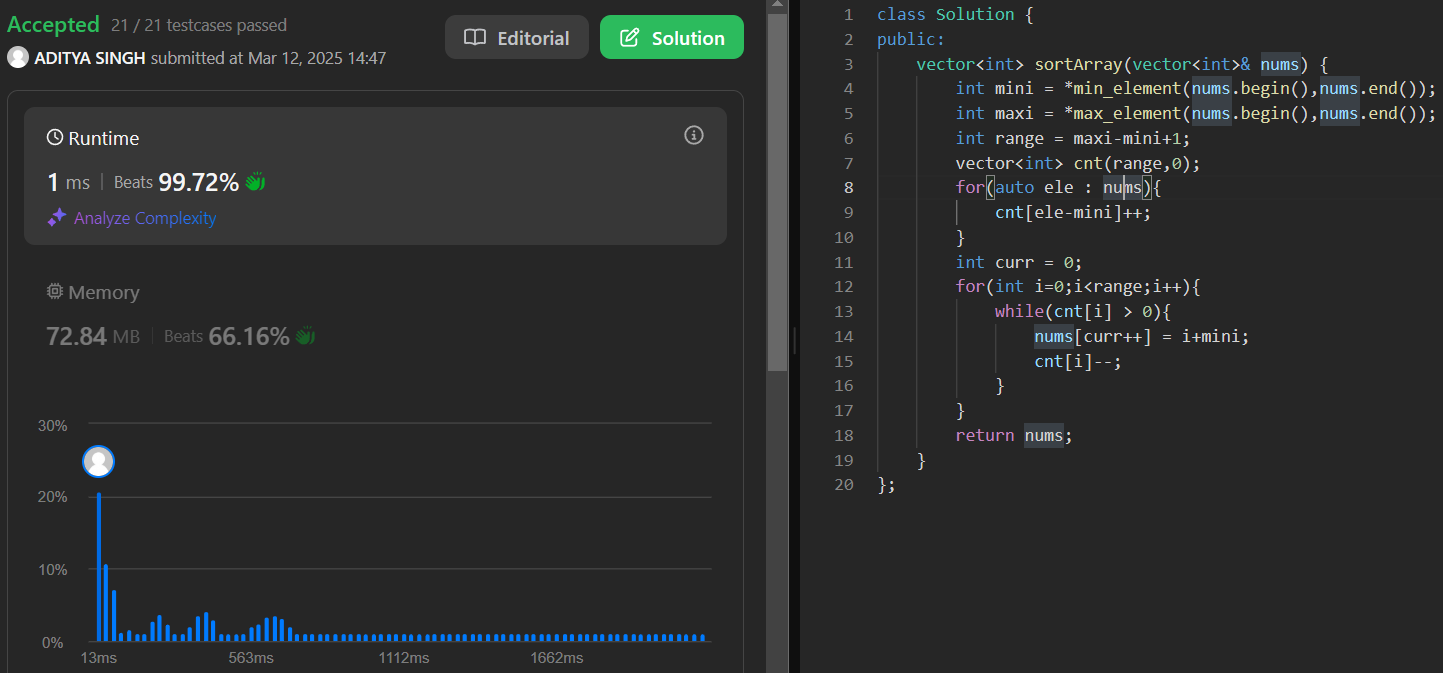
}

return nums;

}

};

Result:



1. **Maximum Subarray:**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int ans = nums[0], f = nums[0];

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

f = max(f, 0) + nums[i];

ans = max(ans, f);

}

return ans;

}

};

Result:

A screenshot of a computer

AI-generated content may be incorrect.

1. **Beautiful Array:**

class Solution {

public:

vector<int> beautifulArray(int n) {

if (n == 1) return {1};

vector<int> left = beautifulArray((n + 1) >> 1);

vector<int> right = beautifulArray(n >> 1);

vector<int> ans(n);

int i = 0;

for (int& x : left) ans[i++] = x \* 2 - 1;

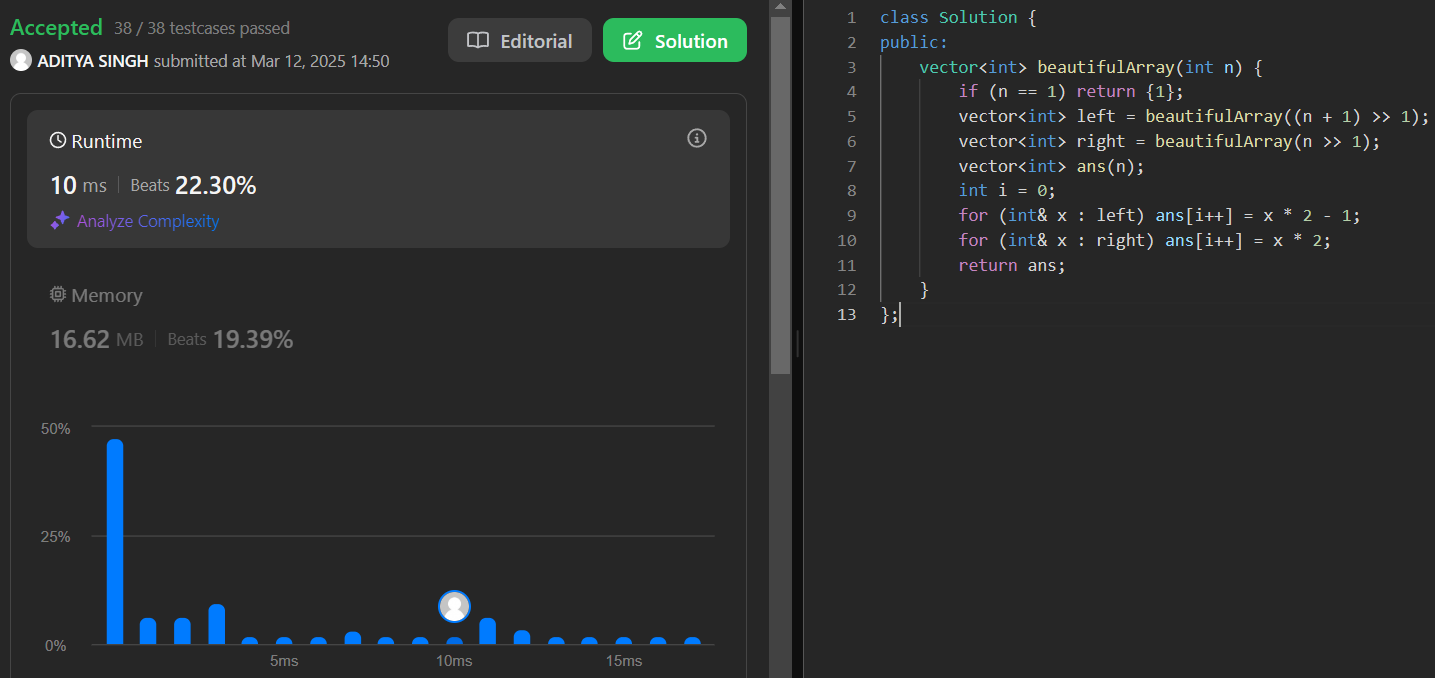
for (int& x : right) ans[i++] = x \* 2;

return ans;

}

};

Result:



1. **Super Pow:**

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;

}

};

Result:

A screenshot of a computer

AI-generated content may be incorrect.

1. **The Skyline Problem:**

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);

}

std::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;

}

};

Result:

