1. Binary tree Inorder traversal

class Solution {

public:

void inorder(TreeNode\* root,vector<int> &temp){

if(root==NULL)

return;

inorder(root->left,temp);

temp.push\_back(root->val);

inorder(root->right,temp);

}

vector<int> inorderTraversal(TreeNode\* root) {

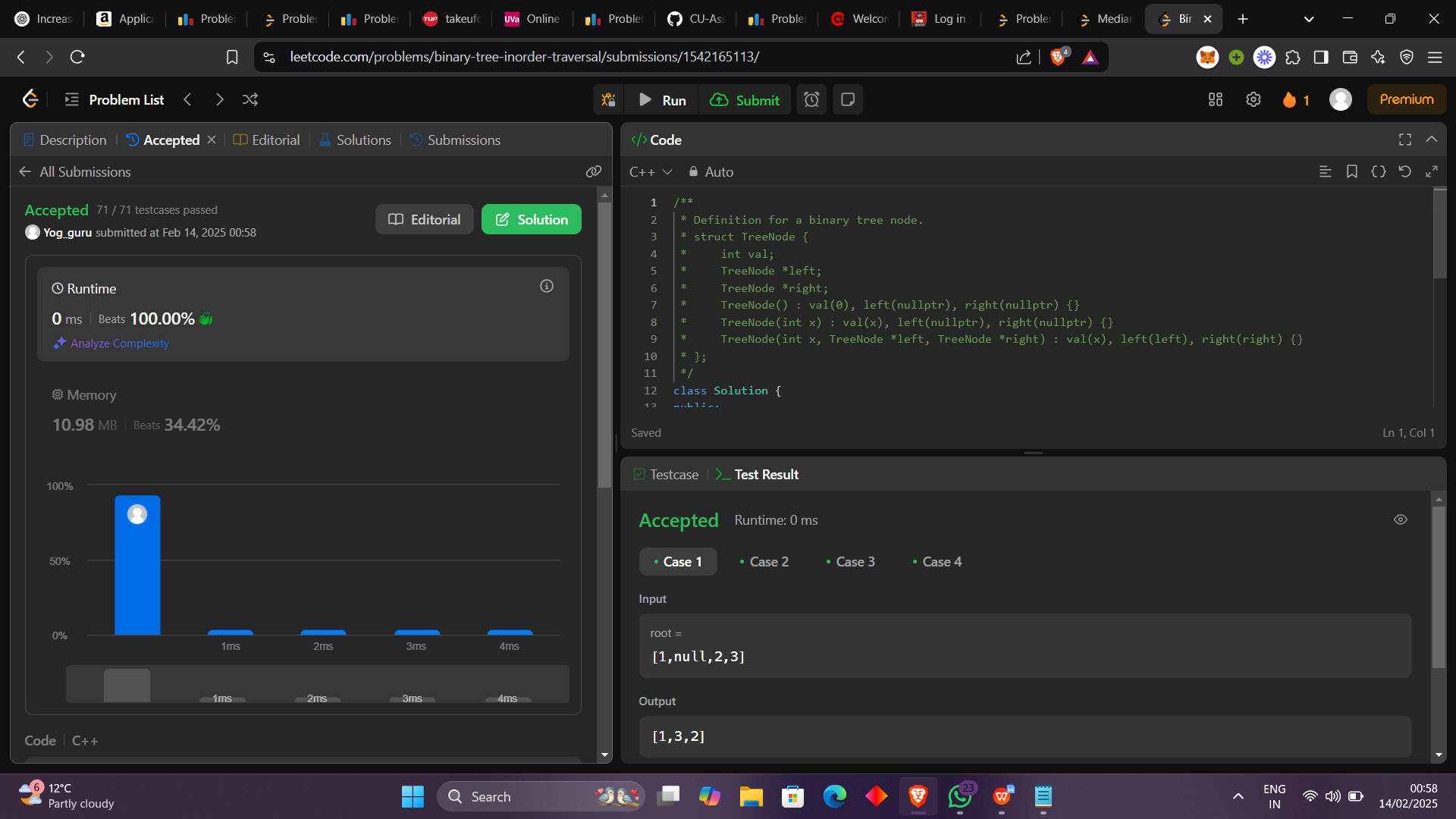
vector<int> temp;

inorder(root,temp);

return temp;

}

};



1. Symmetric Tree

class Solution {

public:

bool isSameTree(TreeNode\* p, TreeNode\* q) {

if(p==NULL && q==NULL)

return true;

if(p==NULL || q==NULL)

return false;

return isSameTree(p->left,q->right)&isSameTree(p->right,q->left) & (p->val==q->val);

}

bool isSymmetric(TreeNode\* root) {

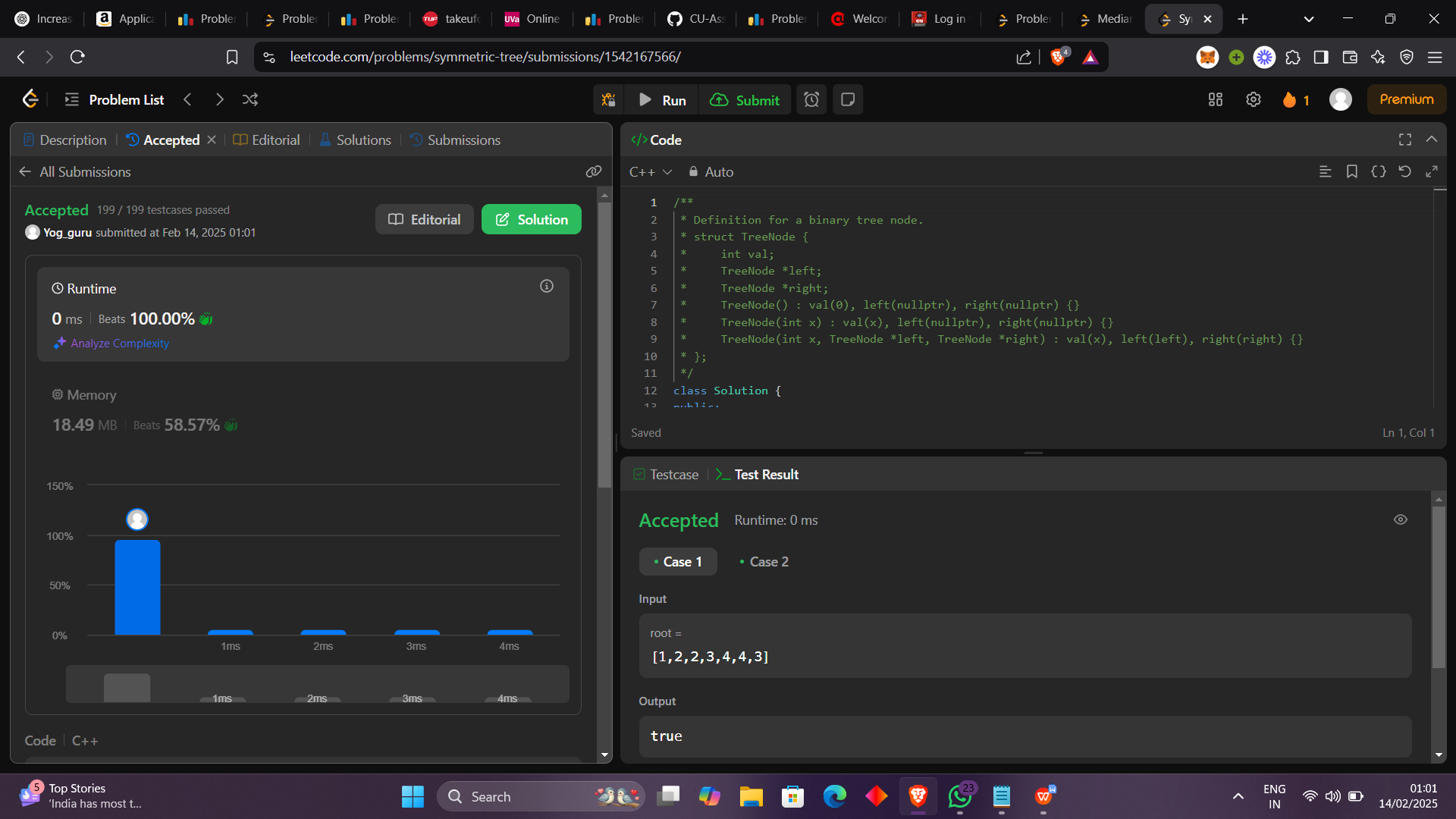
TreeNode\* p=root->left;

TreeNode\* q=root->right;

return isSameTree(p,q);

}

};



1. Maximum Deapth of a Binary Tree

class Solution {

public:

int deapth(TreeNode\* root){

if(root==NULL)

return 0;

int lheight=(deapth(root->left)+1);

int rheight=deapth(root->right)+1;

return max(lheight,rheight);

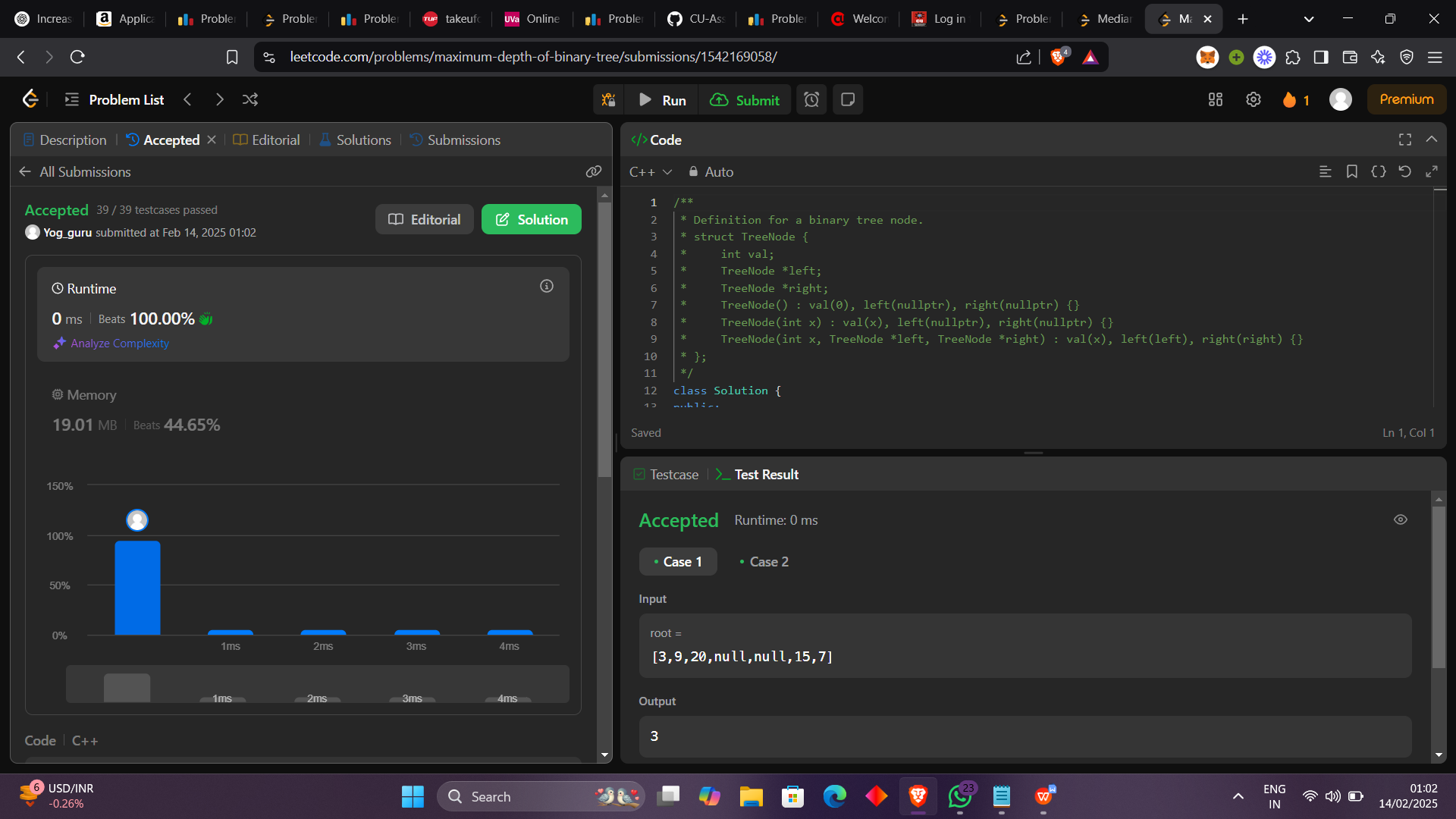
}

int maxDepth(TreeNode\* root) {

return deapth(root);

}

};



1. Validate Binary Search Tree

class Solution {

public:

void inorder(TreeNode\* root,vector<int> & tree){

if(root==NULL)

return;

inorder(root->left,tree);

tree.push\_back(root->val);

inorder(root->right,tree);

}

bool isValidBST(TreeNode\* root) {

vector<int> tree;

inorder(root,tree);

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

if(tree[i]<=tree[i-1])

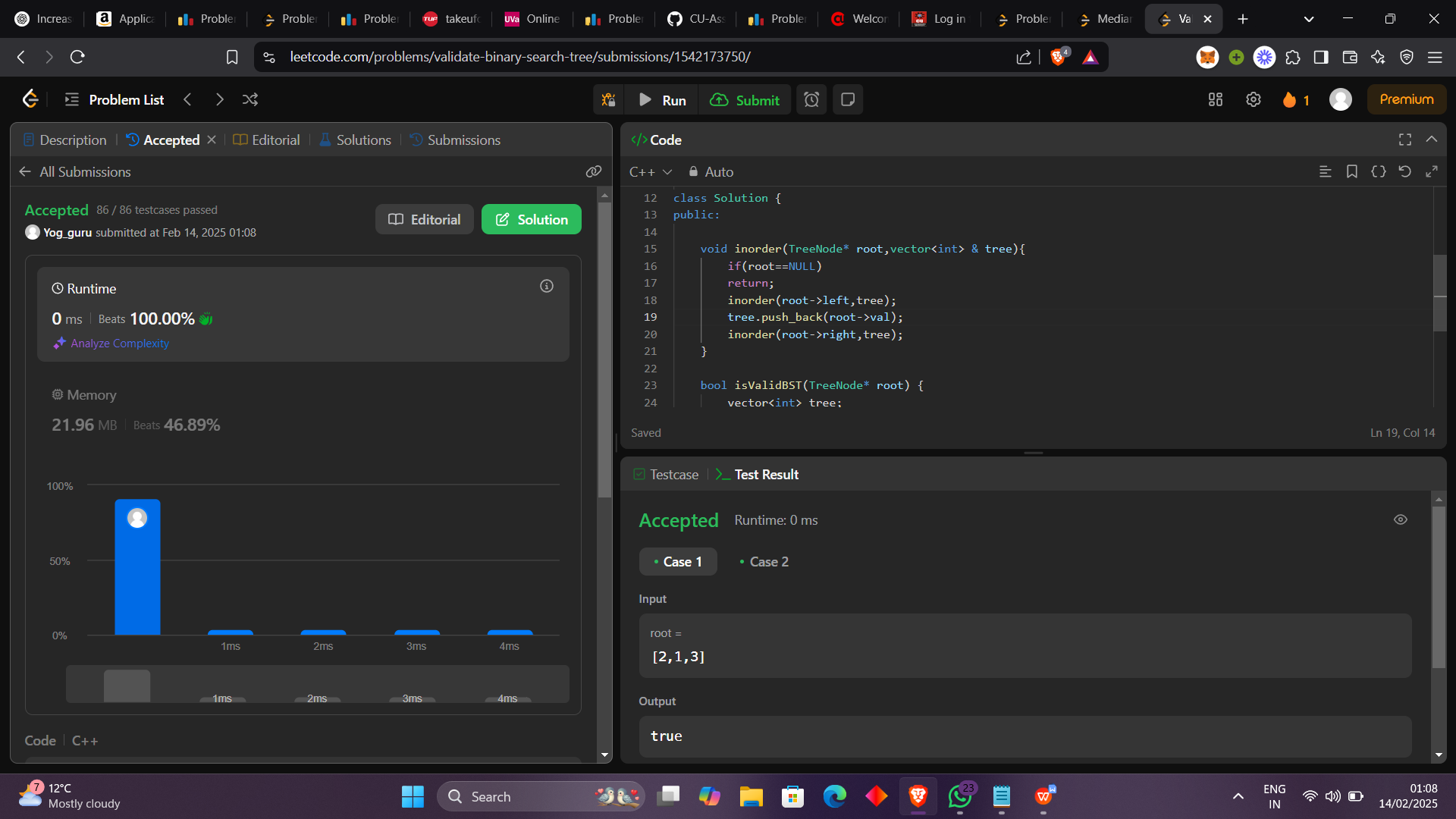
return 0;

}

return 1;

}

};



1. Kth smallest Element in a Binary Search Tree

class Solution {

public:

void inorder(TreeNode\* root,vector<int> & tree){

if(root==NULL)

return;

inorder(root->left,tree);

tree.push\_back(root->val);

inorder(root->right,tree);

}

int kthSmallest(TreeNode\* root, int k) {

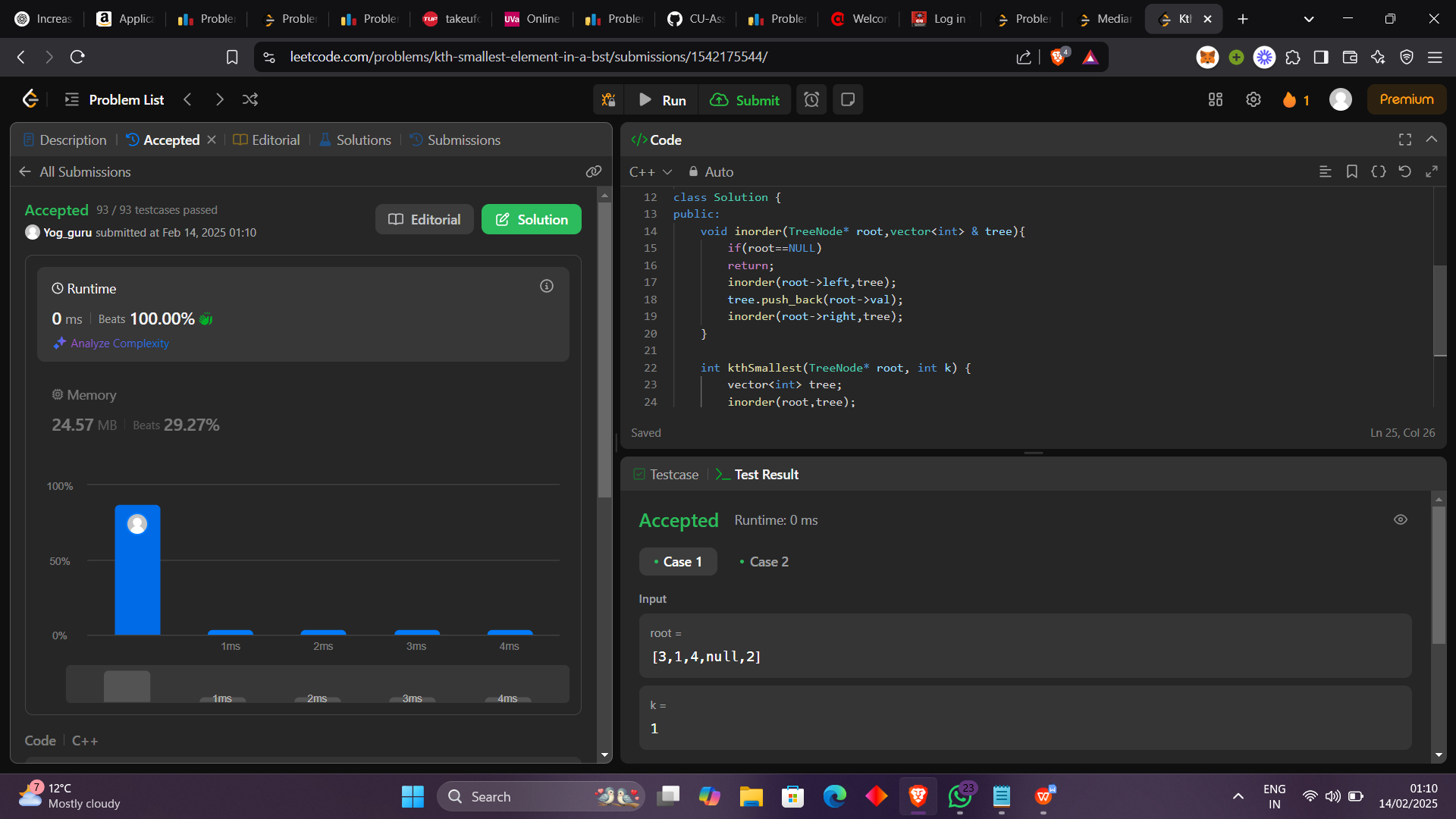
vector<int> tree;

inorder(root,tree);

return tree[k-1];

}

};



1. Binary Tree Level Order Traversal

class Solution {

public:

vector<vector<int>> levelOrder(TreeNode\* root) {

queue<TreeNode\*> temp;

temp.push(root);

temp.push(NULL);

vector<vector<int>> ans;

vector<int> tempo;

if(root==NULL)

return ans;

while(temp.size()!=1){

if(temp.front()==NULL){

ans.push\_back(tempo);

tempo.clear();

temp.push(NULL);

}

else{

tempo.push\_back(temp.front()->val);

if(temp.front()->left!=NULL)

temp.push(temp.front()->left);

if(temp.front()->right!=NULL)

temp.push(temp.front()->right);

}

temp.pop();

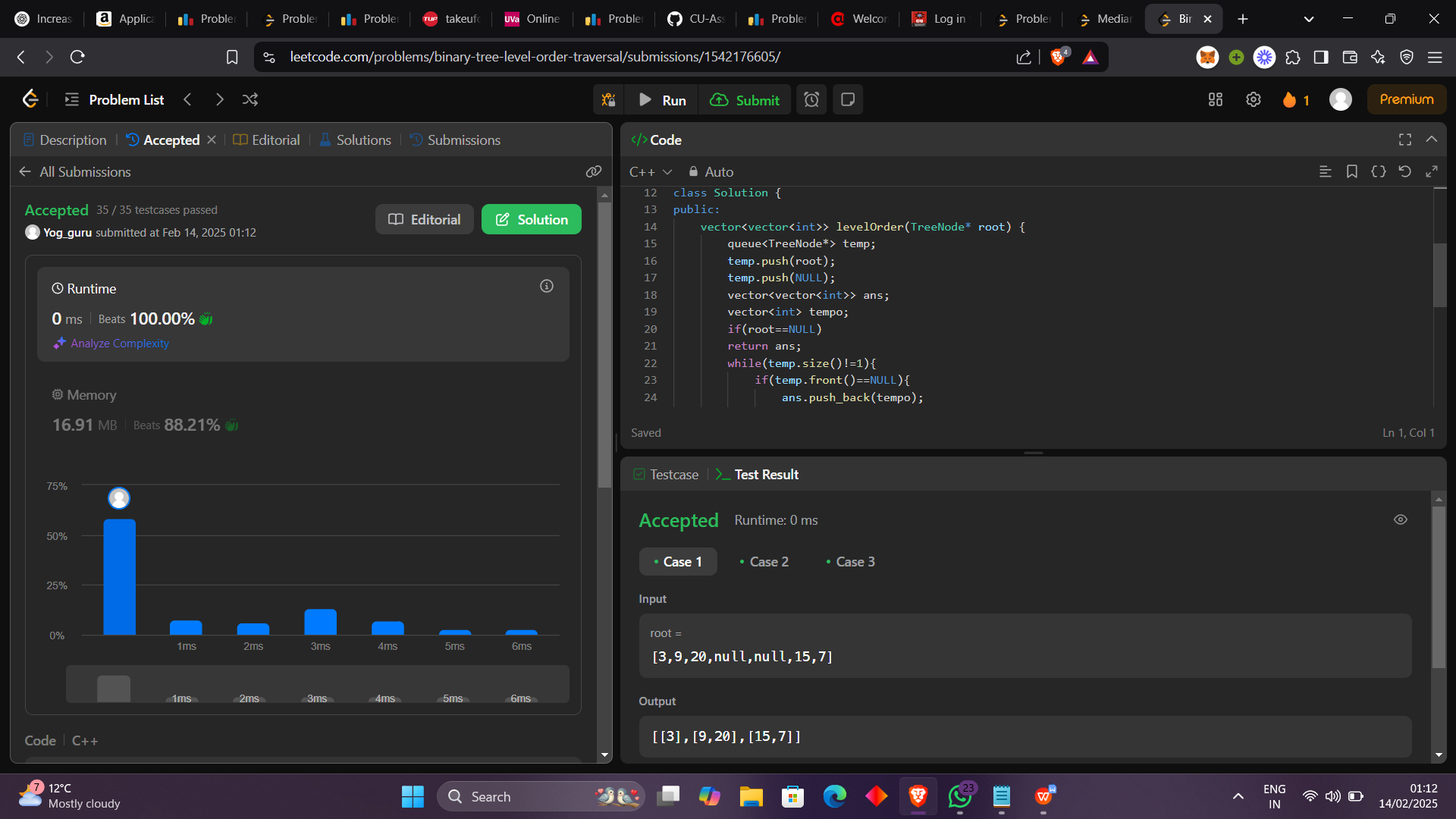
}

ans.push\_back(tempo);

return ans;

}

};



1. Binary Tree Level Order Traveral II

class Solution {

public:

vector<vector<int>> levelOrderBottom(TreeNode\* root) {

queue<TreeNode\*> temp;

temp.push(root);

temp.push(NULL);

vector<vector<int>> ans;

vector<int> tempo;

if(root==NULL)

return ans;

while(temp.size()!=1){

if(temp.front()==NULL){

ans.push\_back(tempo);

tempo.clear();

temp.push(NULL);

}

else{

tempo.push\_back(temp.front()->val);

if(temp.front()->left!=NULL)

temp.push(temp.front()->left);

if(temp.front()->right!=NULL)

temp.push(temp.front()->right);

}

temp.pop();

}

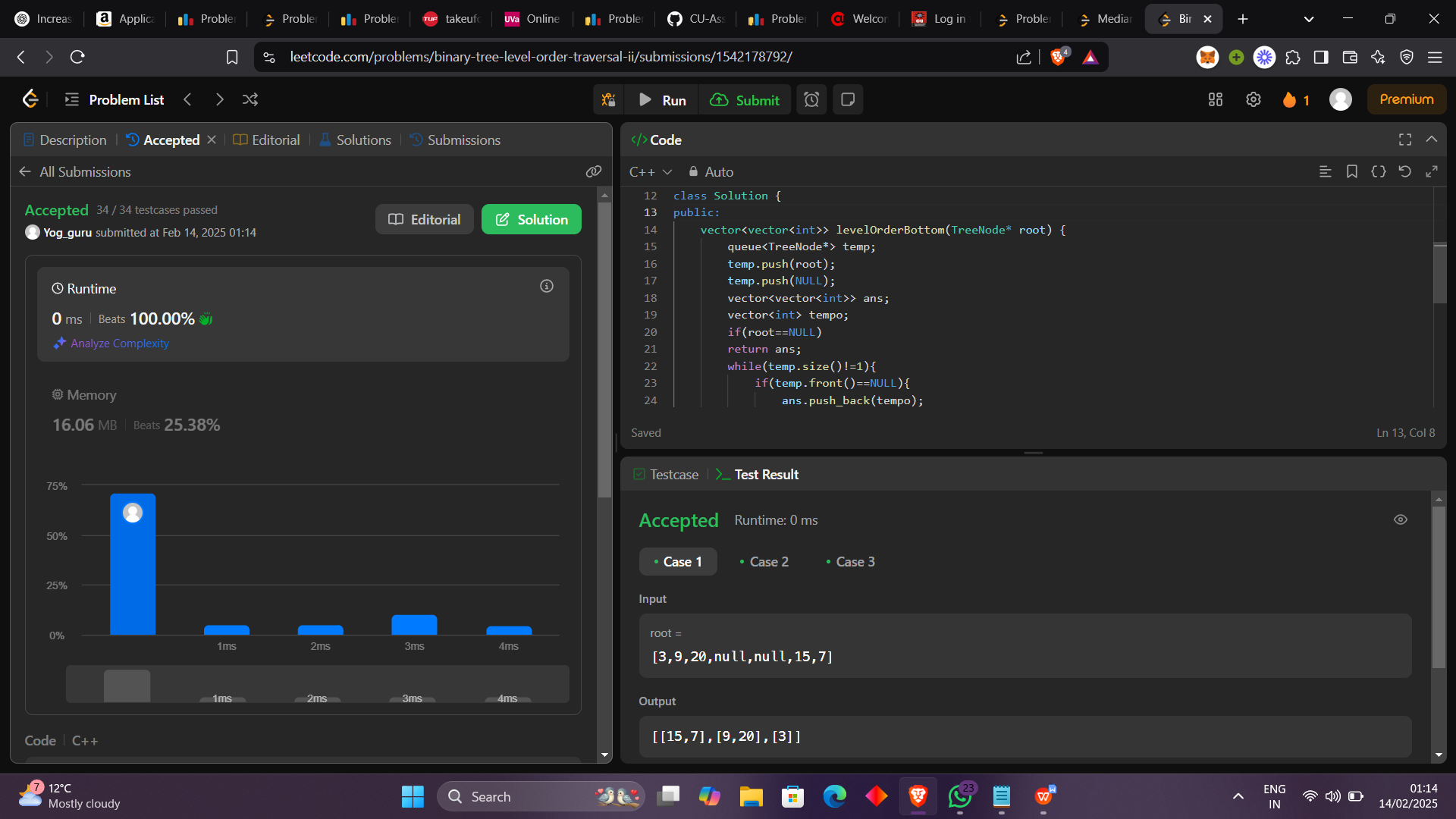
ans.push\_back(tempo);

reverse(ans.begin(),ans.end());

return ans;

}

};



1. Binary Tree Zigzag Level Order Traversal

class Solution {

public:

stack<TreeNode\*> stack1;

stack<TreeNode\*> stack2;

bool flag=1;

void spiral(TreeNode\* root){

if(flag==1){

if(root->left!=NULL)

stack1.push(root->left);

if(root->right!=NULL)

stack1.push(root->right);

}

else{

if(root->right!=NULL)

stack2.push(root->right);

if(root->left!=NULL)

stack2.push(root->left);

}

}

vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

vector<vector<int>> ans;

if(root==NULL)

return ans;

stack2.push(root);

while(stack2.empty()==0 || stack1.empty()==0){

vector<int> temp;

while(stack2.empty()==0){

spiral(stack2.top());

temp.push\_back(stack2.top()->val);

stack2.pop();

}

if(temp.size()!=0)

ans.push\_back(temp);

temp.clear();

flag=flag^1;

while(stack1.empty()==0){

spiral(stack1.top());

temp.push\_back(stack1.top()->val);

stack1.pop();

}

if(temp.size()!=0)

ans.push\_back(temp);

flag=flag^1;

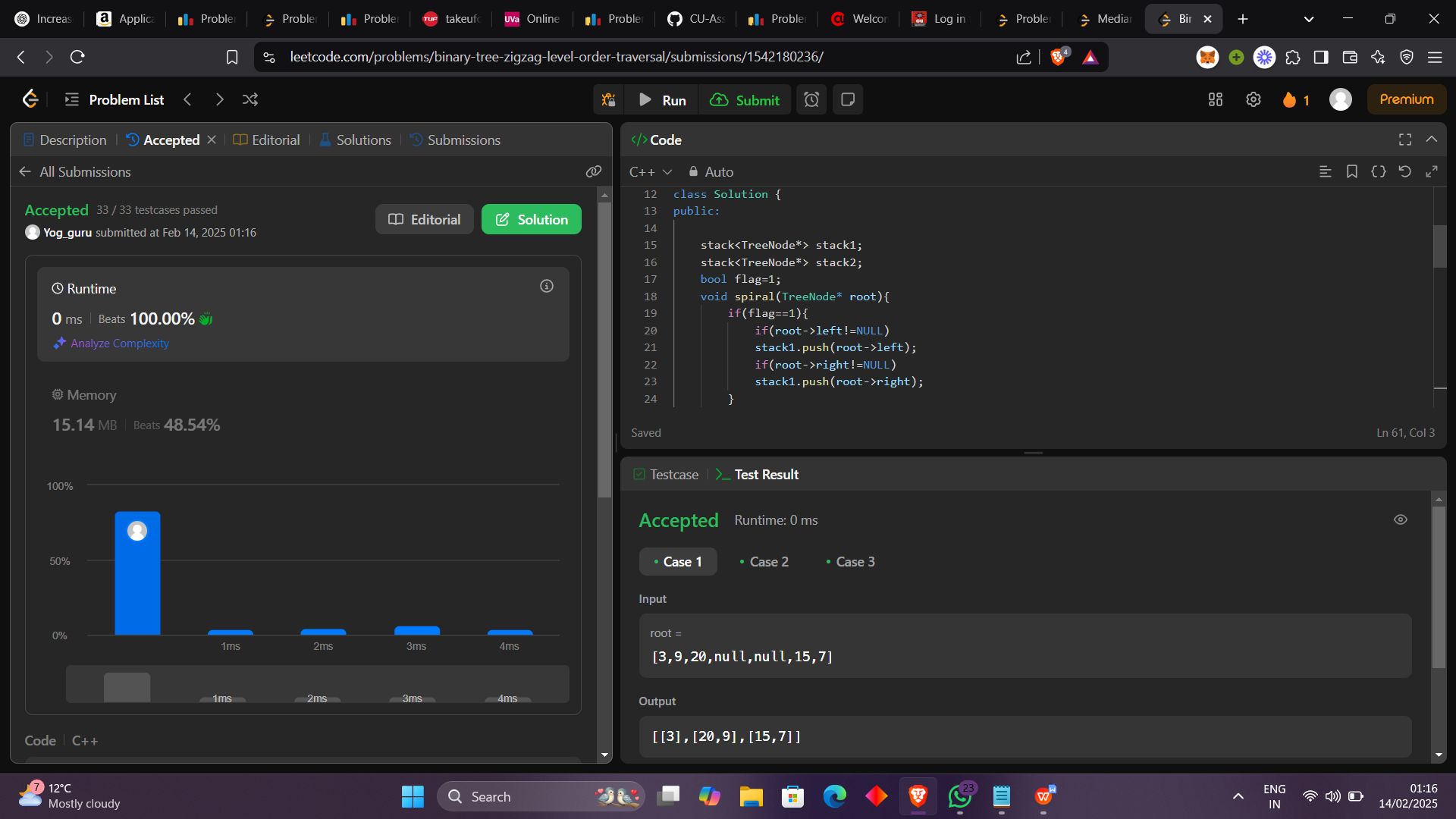
temp.clear();

}

return ans;

}

};



1. Vertical Order Traveral of a Binary Tree

class Solution {

public:

vector<vector<int>> verticalTraversal(TreeNode\* root) {

queue<pair<pair<int,int>,TreeNode\*>> level;

map<int,map<int,vector<int>>>temp;

level.push(make\_pair(make\_pair(0,0),root));

int lvl=1;

while(level.empty()==0){

if(level.front().second->left!=NULL){

level.push(make\_pair(make\_pair(lvl,level.front().first.second-1),level.front().second->left));

}

if(level.front().second->right!=NULL){

level.push(make\_pair(make\_pair(lvl,level.front().first.second+1),level.front().second->right));

}

temp[level.front().first.second][level.front().first.first].push\_back(level.front().second->val);

level.pop();

if(level.empty()!=1 && level.front().first.first==lvl)

lvl++;

}

vector<vector<int>> ans;

for(auto it:temp){

vector<int> temporary;

for(auto i:it.second){

sort(i.second.begin(),i.second.end());

for(auto a:i.second){

temporary.push\_back(a);

}

}

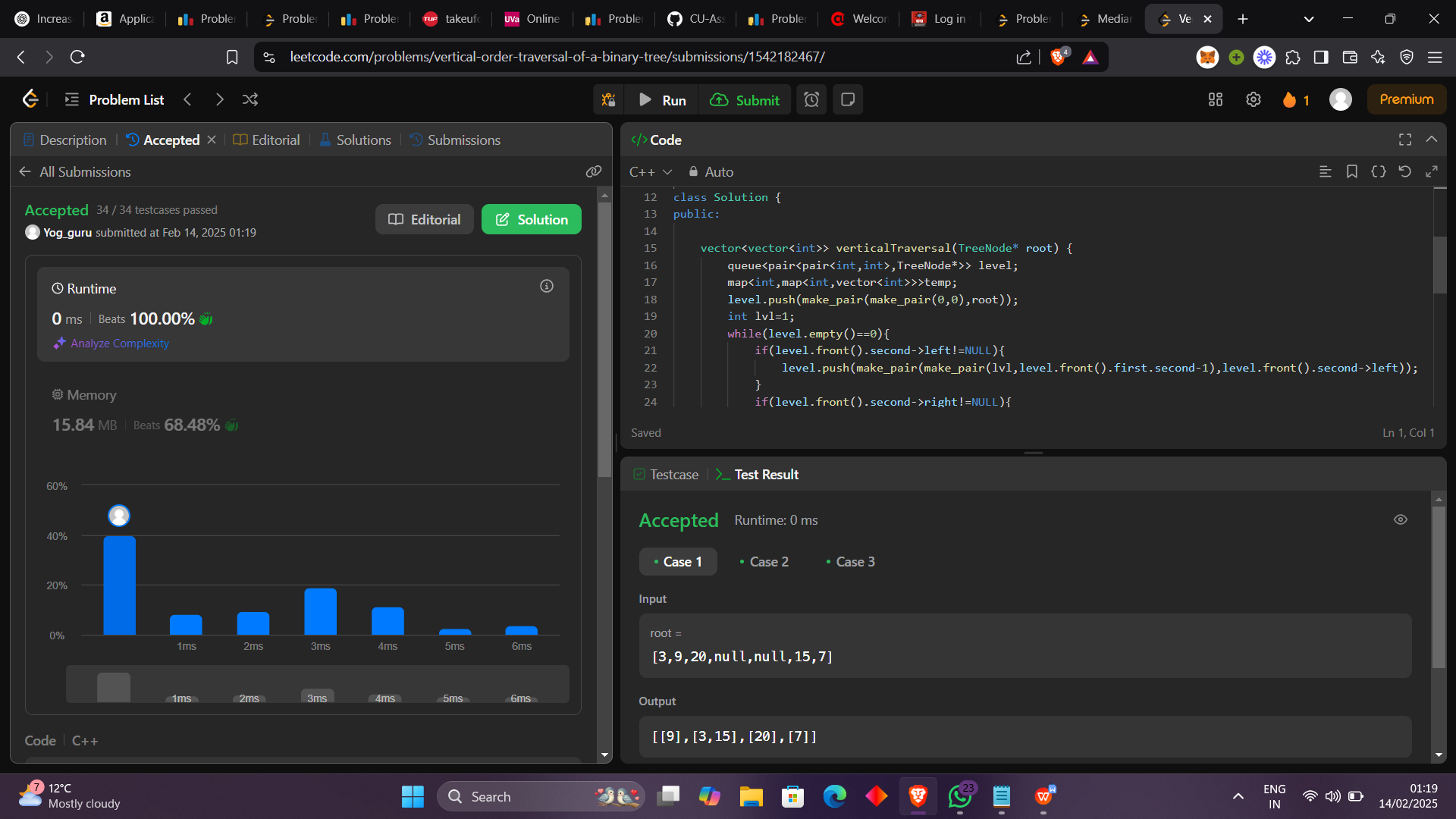
ans.push\_back(temporary);

}

return ans;

}

};



1. Binary Tree Right Side View

class Solution {

public:

void postorder(TreeNode\* root, vector<int> &ans, int count){

if(root==NULL)

return;

if(count==ans.size())

ans.push\_back(root->val);

postorder(root->right,ans,count+1);

postorder(root->left,ans,count+1);

}

vector<int> rightSideView(TreeNode\* root) {

vector<int> ans;

int count=0;

postorder(root,ans,count);

return ans;

}

};

