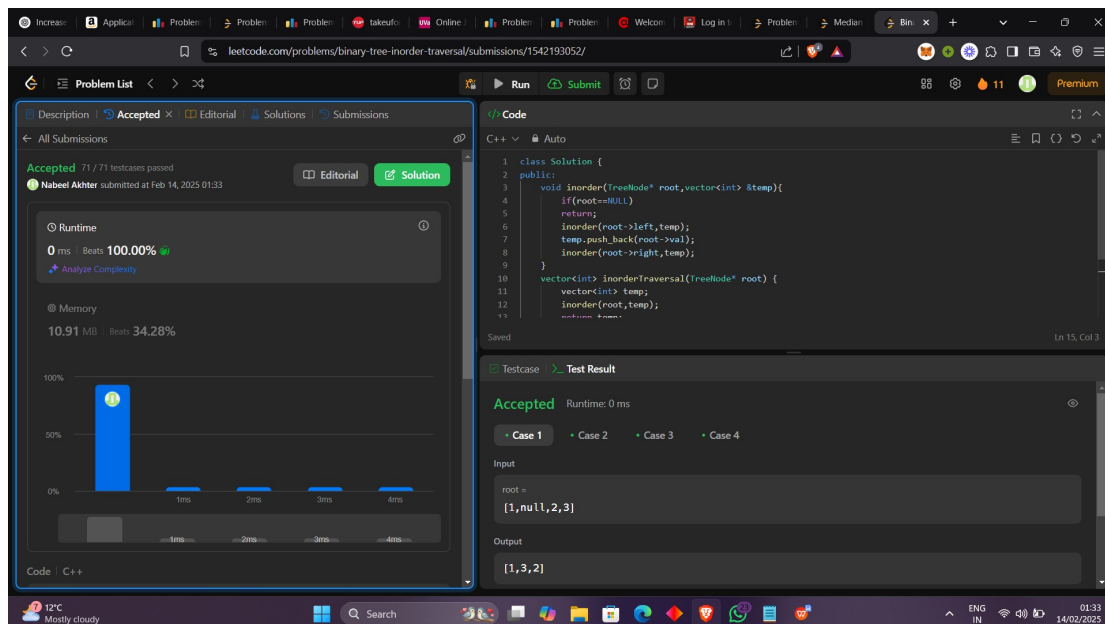


## 94. Binary tree Inorder traversal

```
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;
    }
};
```



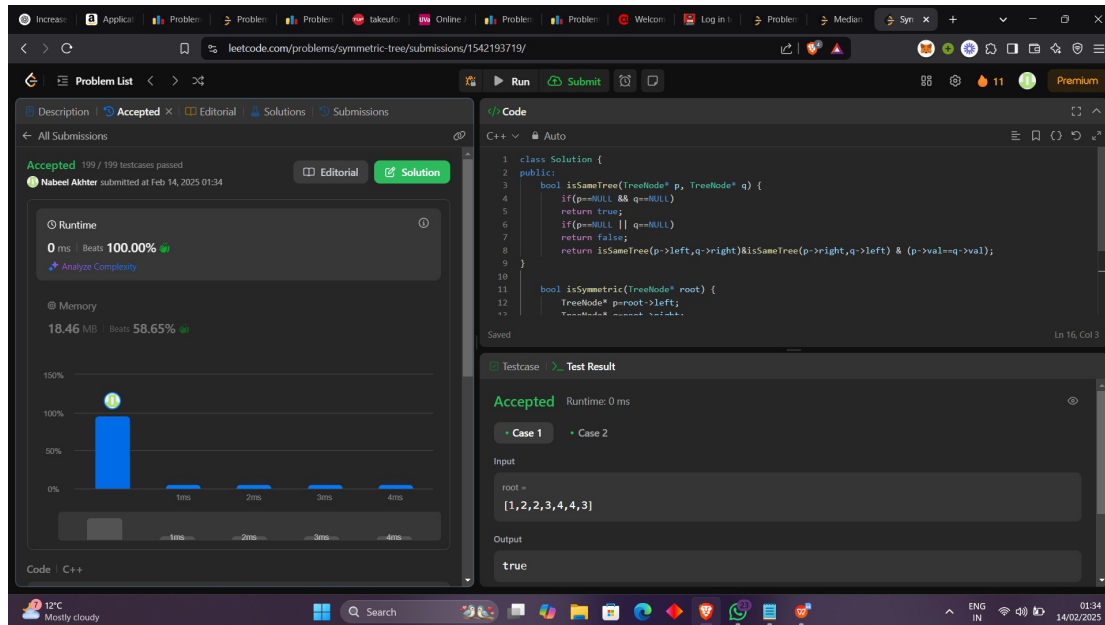
## 101. Symmetric Tree

```
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);
}
};

```



## 104. Maximum Depth 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);
    }
};

```

The screenshot shows a LeetCode submission for the problem "Maximum Depth of Binary Tree". The submission is accepted, with a runtime of 0 ms and memory usage of 19.09 MB. The code is in C++ and implements a recursive function to calculate the maximum depth of a binary tree. The test case shows a tree with root 3, left child 9, right child 20, and further children 15 and 7.

```

class Solution {
public:
    int depth(TreeNode* root){
        if(root==NULL)
            return 0;
        int lheight=depth(root->left)+1;
        int rheight=depth(root->right)+1;
        return max(lheight,rheight);
    }
    int maxDepth(TreeNode* root) {
        return depth(root);
    }
};

```

Testcase: Case 1, Case 2. Input: root = [3,9,20,null,null,15,7]. The tree structure is shown as a diagram with root 3, left child 9, right child 20, and further children 15 and 7.

## 98. Validate Binary Search Tree

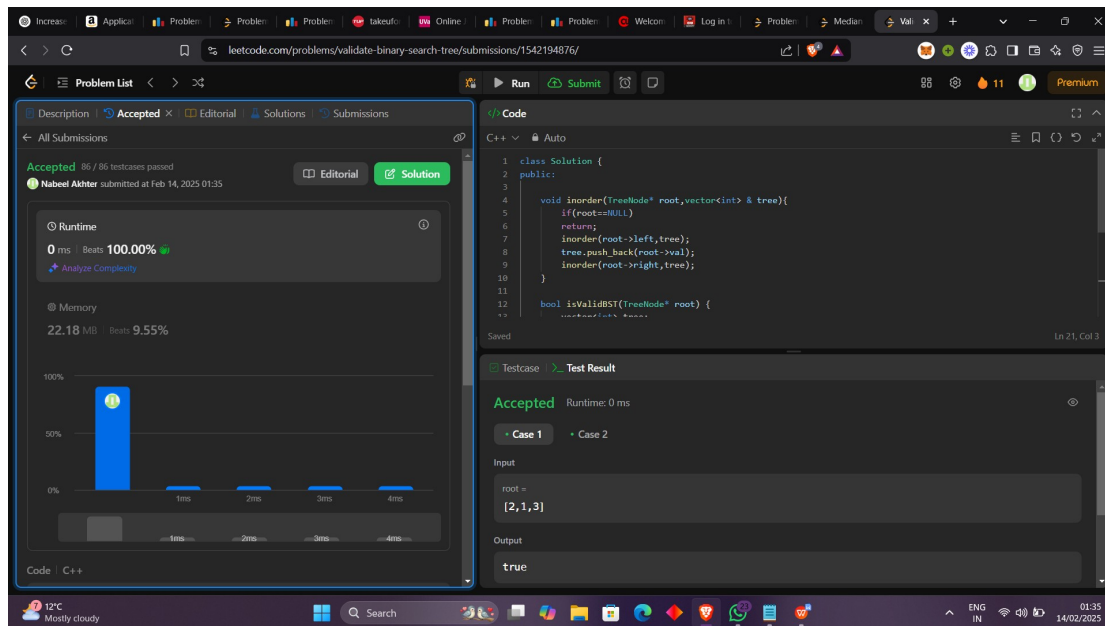
```

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;
    }
};

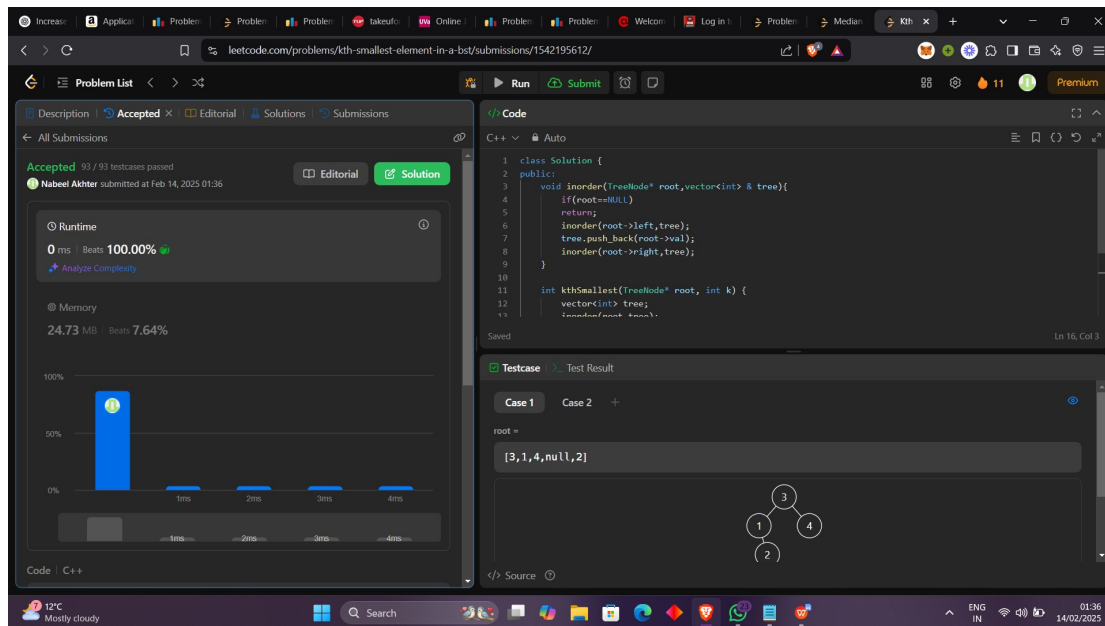
```



### 230. 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];
    }
};
```



## 102. 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;
    }
}
```

};

The screenshot displays a LeetCode submission interface. On the left, the 'Submissions' tab shows the submission is 'Accepted' with 35/35 testcases passed. The runtime is 3 ms (Beats 27.72%) and memory is 17.03 MB (Beats 70.68%). A bar chart shows the runtime distribution. On the right, the 'Code' tab shows the C++ code for the solution. Below the code, the 'Testcase' tab shows the input and output for Case 1.

```
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);
        reverse(ans.begin(),ans.end());
        return ans;
    }
};
```

Testcase: Accepted Runtime: 0 ms

Case 1: Input: root = [3, 9, 20, null, null, 15, 7] Output: [[3], [9, 20], [15, 7]]

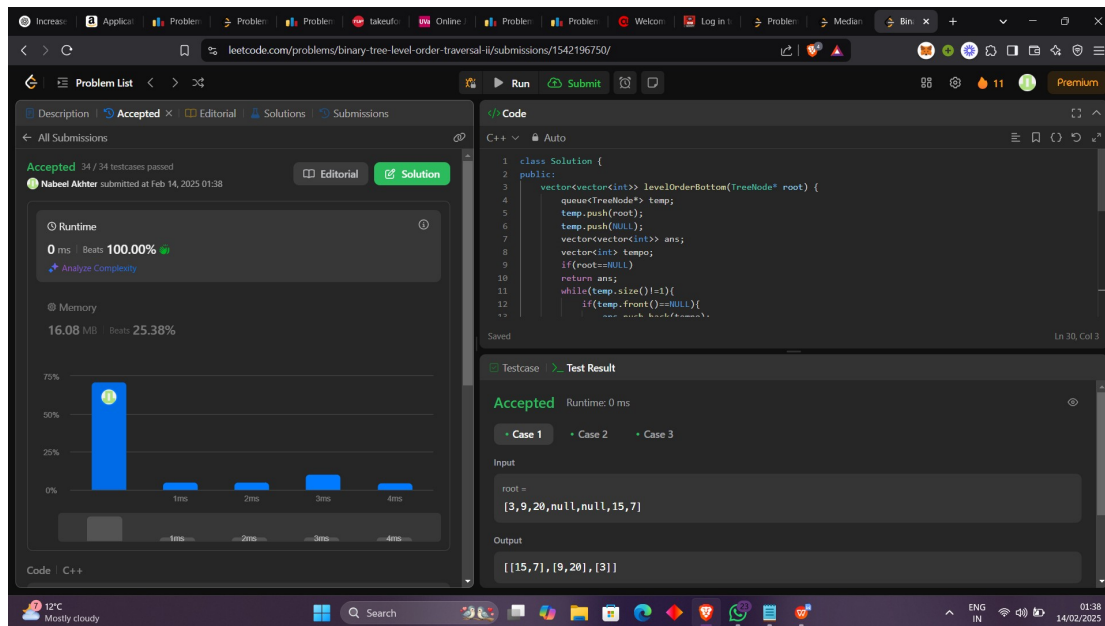
## 107. Binary Tree Level Order Traversal II

```
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;
    }
};
```

```

    }
};

```



### 103. 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);
        }
    }
}

```

```

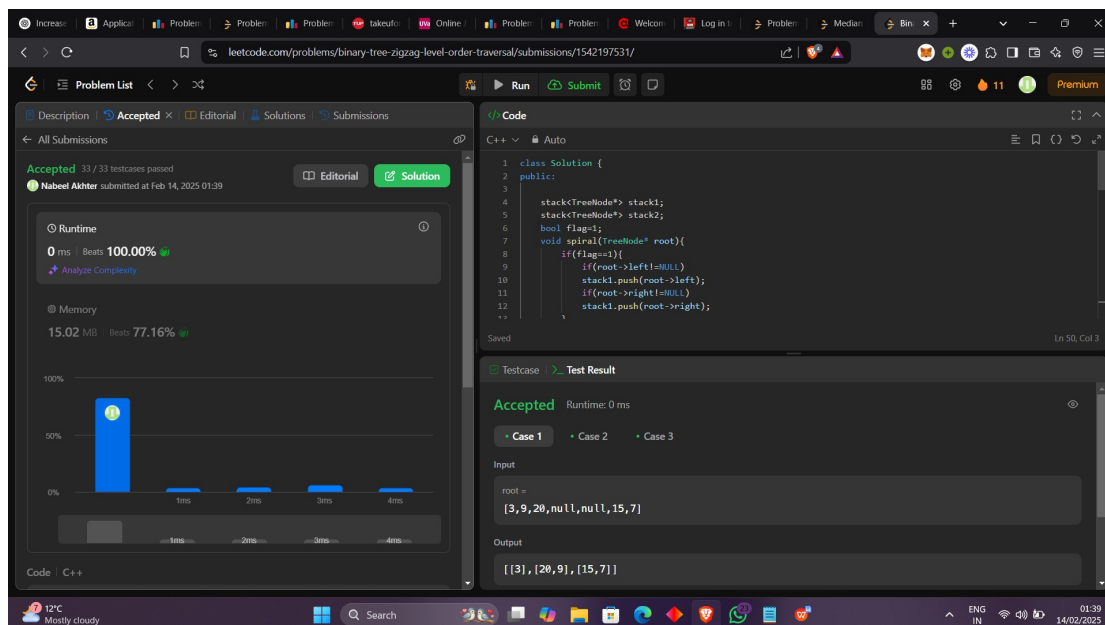
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>> ans;
    if(root==NULL)
        return ans;

```

```

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);
    temp.clear();
    flag=flag^1;
}
return ans;
}
};

```



## 987. Vertical Order Traversal 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;
}
};

```

leetcod.com/problems/vertical-order-traversal-of-a-binary-tree/submissions/1542198237/

Problem List > < > < >

Description | Accepted | Editorial | Solutions | Submissions

Accepted 34 / 34 testcases passed

Nabeel Akhter submitted at Feb 14, 2025 01:40

Editorial Solution

Runtime: 4 ms | Beats: 23.45%

Memory: 15.76 MB | Beats: 74.44%

Code C++

```

1 class Solution {
2 public:
3
4     vector<vector<int>> verticalTraversal(TreeNode* root) {
5         queue<pair<pair<int, int>, TreeNode*>> level;
6         map<int, map<int, vector<int>>>>temp;
7         level.push(make_pair(make_pair(0,0),root));
8         int lvl=1;
9         while(level.empty()==0){
10             if(level.front().second->left!=NULL){
11                 level.push(make_pair(make_pair(lvl,level.front().first.second-1),level.front().second->left));
12             }
13             //level.front().second->right!=NULL ?
14         }
15     }
16 };

```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

root = [3,9,20,null,null,15,7]

Output

[[9], [3,15], [20], [7]]

## 199. 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;
    }
};

```

The screenshot shows a C++ solution for the 'Binary Tree Right Side View' problem on LeetCode. The code uses a recursive approach to traverse the tree in postorder, storing the rightmost node value at each level in a vector. The solution is accepted, with a runtime of 0ms and memory usage of 14.88 MB, both of which are optimal (beats 100.00% and 84.63% respectively). A bar chart shows the runtime distribution, with the user's solution being the fastest. The test case input is [1,2,3,null,5,null,4] and the output is [1,3,4].

```
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;
        postorder(root,ans,0);
        return ans;
    }
};
```

Runtime: 0 ms, Beats 100.00%  
Memory: 14.88 MB, Beats 84.63%

Testcase: Test Result  
Accepted Runtime: 0 ms  
Case 1 Case 2 Case 3 Case 4  
Input: root = [1,2,3,null,5,null,4]  
Output: [1,3,4]