**AP ASSIGNMENT - 5**

**Name: Abhijeet | UID: 22bcs16832| Section: 612-“B”**

**maximum-depth-of-binary-tree**

class Solution {

public:

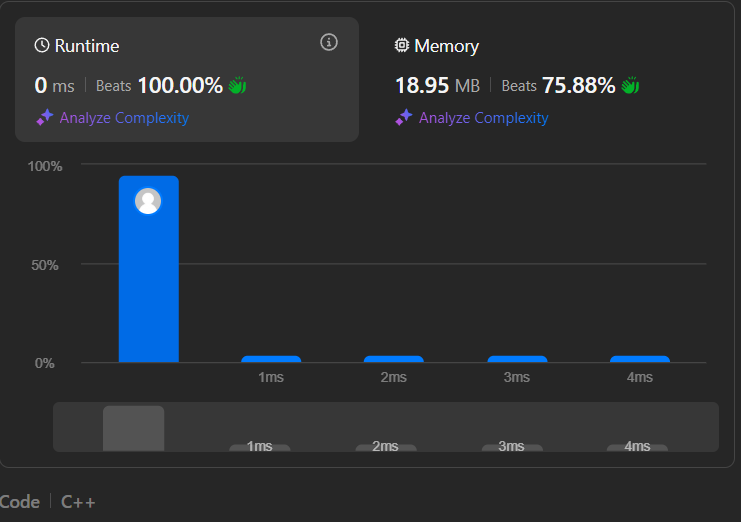
int maxDepth(TreeNode\* root) {

if (root == nullptr) return 0;

int leftDepth = maxDepth(root->left);

int rightDepth = maxDepth(root->right);

return max(leftDepth, rightDepth) + 1;}}



[**Validate Binary Search Tree**](https://leetcode.com/problems/validate-binary-search-tree/description/)

class Solution {

public:

    bool isValidBST(TreeNode\* root) {

          return isValidBSTHelper(root, LONG\_MIN, LONG\_MAX); }

private:

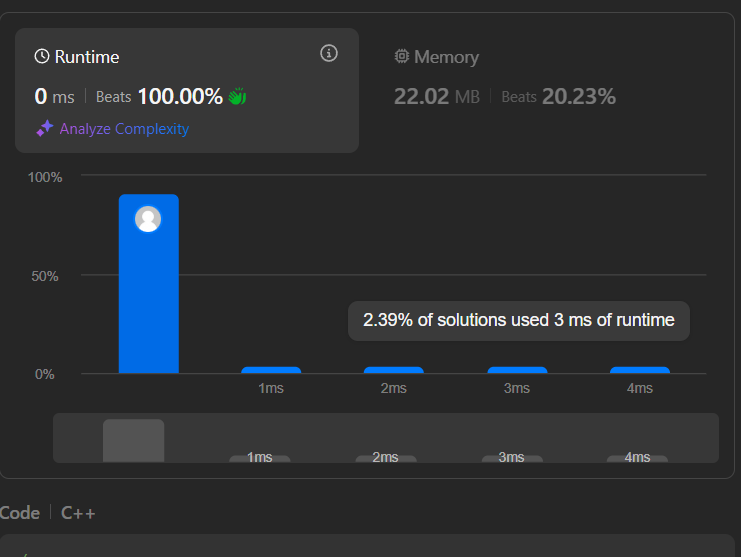
    bool isValidBSTHelper(TreeNode\* root, long minVal, long maxVal) {

        if (!root) return true;

        if (root->val <= minVal || root->val >= maxVal) return false;

        return isValidBSTHelper(root->left, minVal, root->val) &&

               isValidBSTHelper(root->right, root->val, maxVal);}};



**Symmetric Tree**

class Solution {

public:

    bool isSymmetric(TreeNode\* root) {

        return root == nullptr || isMirror(root->left, root->right);

    }

private:

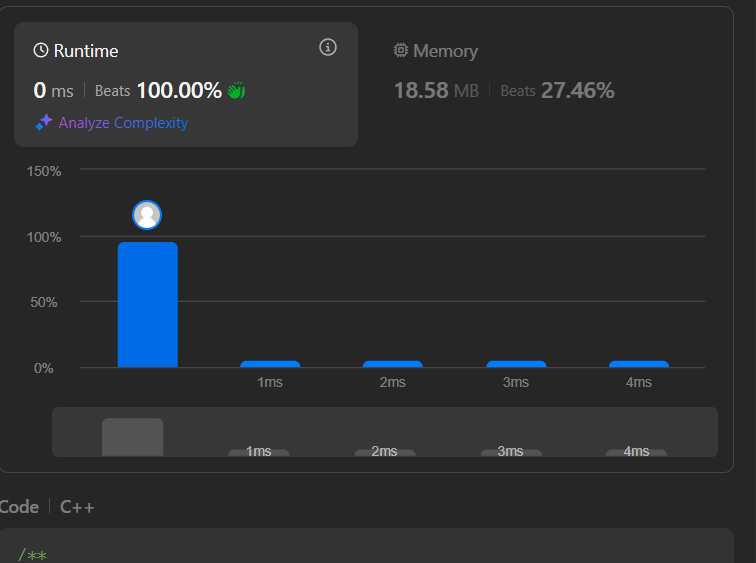
    bool isMirror(TreeNode\* a, TreeNode\* b) {

        if (!a || !b) return a == b;

        return (a->val == b->val) && isMirror(a->left, b->right) && isMirror(a->right, b->left);

    }

};



[**Binary Tree Zigzag Level Order Traversal**](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/)

class Solution {

public:

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

        vector<vector<int>> result;

        if (!root) return result;

        queue<TreeNode\*> q;

        q.push(root);

        bool leftToRight = true;

        while (!q.empty()) {

            int size = q.size();

            vector<int> level(size);

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

                TreeNode\* node = q.front();

                q.pop();

                int index = leftToRight ? i : (size - 1 - i);

                level[index] = node->val;

                if (node->left) q.push(node->left);

                if (node->right) q.push(node->right);

            }

            result.push\_back(level);

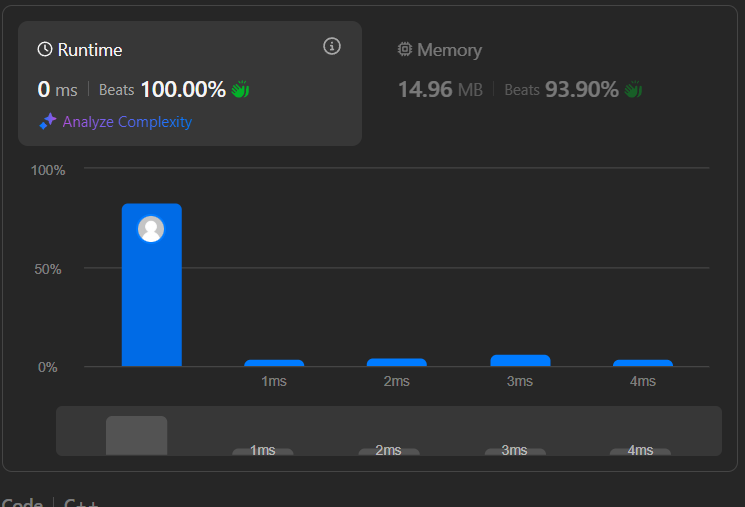
            leftToRight = !leftToRight;

        }

        return result;

    }

};



[**Lowest Common Ancestor of a Binary Tree**](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/)

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

         if (!root || root == p || root == q) return root;

        TreeNode\* left = lowestCommonAncestor(root->left, p, q);

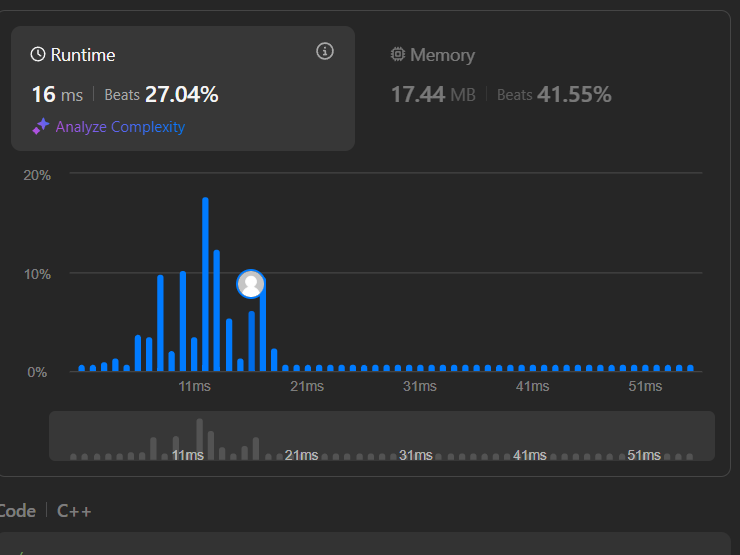
        TreeNode\* right = lowestCommonAncestor(root->right, p, q);

        if (left && right) return root;

        return left ? left : right;

    }

};



[**Binary Tree Inorder Traversal**](https://leetcode.com/problems/binary-tree-inorder-traversal/)

class Solution {

public:

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

        vector<int> result;

        inorder(root, result);

        return result;

    }

private:

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

        if (!root) return;

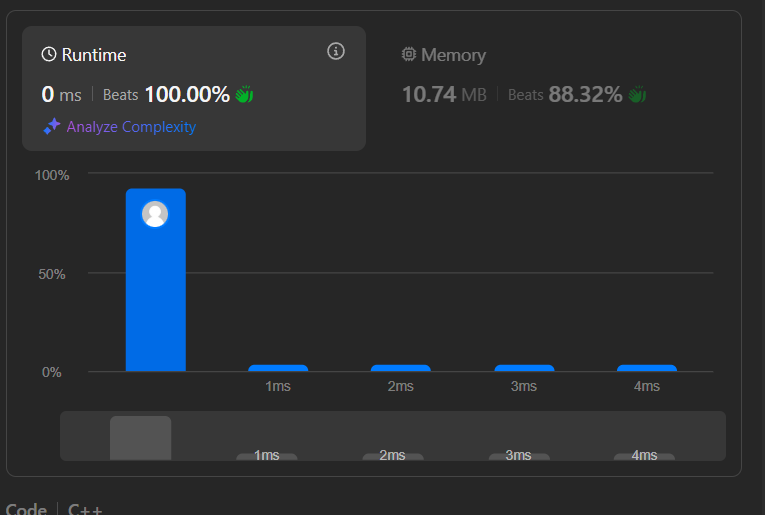
        inorder(root->left, result);

        result.push\_back(root->val);

        inorder(root->right, result);

    }

};



[**Binary Tree Level Order Traversal**](https://leetcode.com/problems/binary-tree-level-order-traversal/)

class Solution {

public:

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

         vector<vector<int>> result;

        if (!root) return result;

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            int size = q.size();

            vector<int> level;

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

                TreeNode\* node = q.front();

                q.pop();

                level.push\_back(node->val);

                if (node->left) q.push(node->left);

                if (node->right) q.push(node->right);

            }

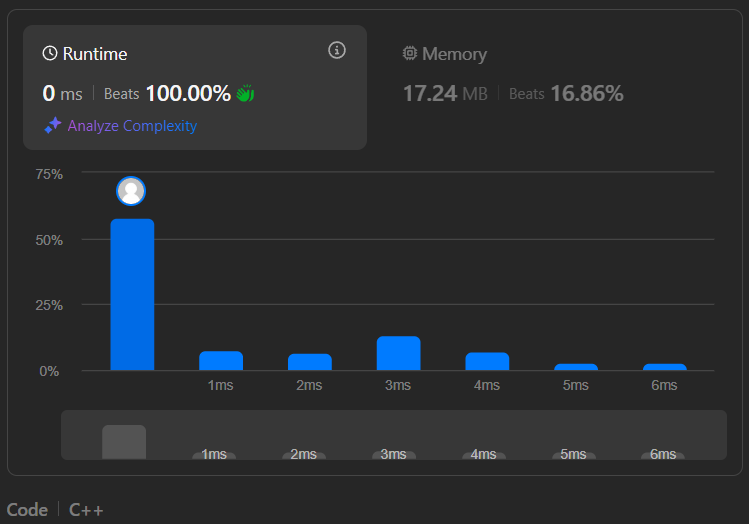
            result.push\_back(level);

        }

        return result;

    }

};

****

[**Kth Smallest Element in a BST**](https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

class Solution {

public:

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

        int count = 0, result = 0;

        inorder(root, k, count, result);

        return result;

    }

private:

    void inorder(TreeNode\* root, int k, int& count, int& result) {

        if (!root) return;

        inorder(root->left, k, count, result);

        count++;

        if (count == k) {

            result = root->val;

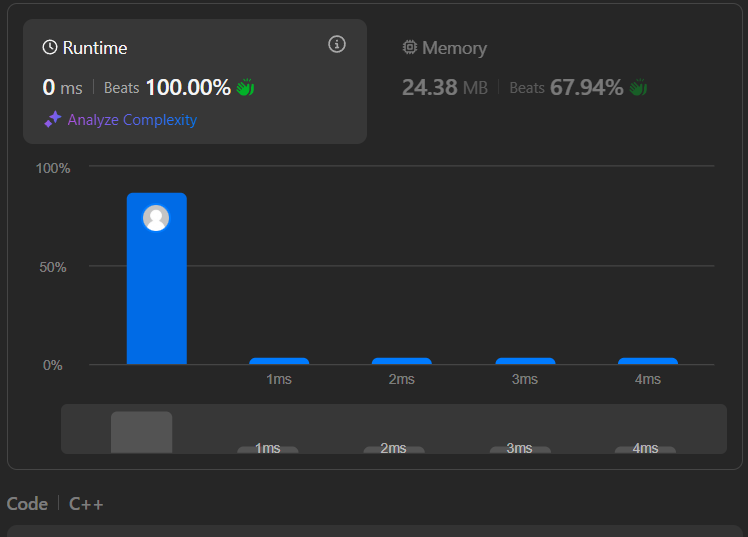
            return;

        }

        inorder(root->right, k, count, result);

    }

};



[**Populating Next Right Pointers in Each Node**](https://leetcode.com/problems/populating-next-right-pointers-in-each-node/)

class Solution {

public:

    Node\* connect(Node\* root) {

        if (!root) return nullptr;

        queue<Node\*> q;

        q.push(root);

        while (!q.empty()) {

            int size = q.size();

            Node\* prev = nullptr;

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

                Node\* node = q.front();

                q.pop();

                if (prev) prev->next = node;

                prev = node;

                if (node->left) q.push(node->left);

                if (node->right) q.push(node->right);

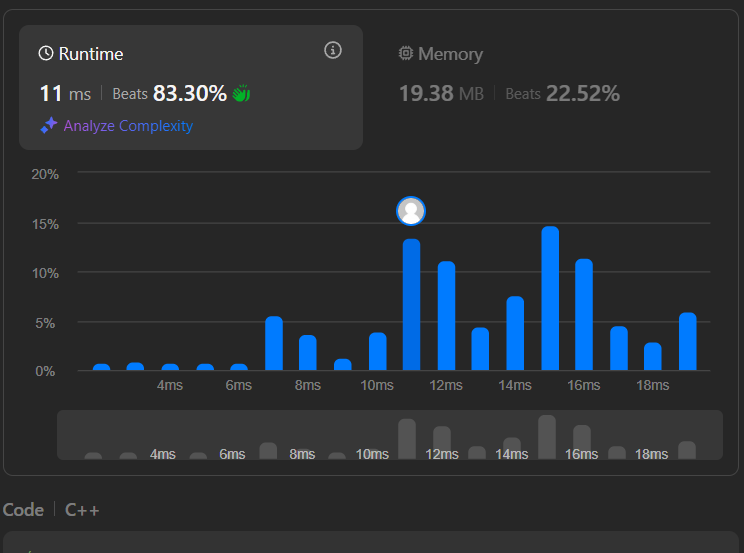
            }

        }

        return root;

    }

};



[**Sum of Left Leaves**](https://leetcode.com/problems/sum-of-left-leaves/)

class Solution {

public:

        int sumOfLeftLeaves(TreeNode\* root) {

        if (!root) return 0;

        int sum = 0;

        if (root->left && !root->left->left && !root->left->right) {

            sum += root->left->val;  // If left child is a leaf, add its value

        }

        return sum + sumOfLeftLeaves(root->left) + sumOfLeftLeaves(root->right);

    }

};

