DOMAIN WINTER CAMP

(Department of Computer Science and Engineering)

Name: Ankit Vashisth UID: 22BCS13378 Section: KPIT 901-B

DAY-6

(Easy)

Q1 Given the root of a binary tree, return the inorder traversal of its nodes' values.

Example 1:

Input: root = [1,null,2,3]

Output: [1,3,2]

Program code:

```
#include <iostream>
#include <vector>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Helper function for inorder traversal
void inorderHelper(TreeNode* root, vector<int>& result) {
    if (root == nullptr) return;
    inorderHelper(root->left, result);  // Traverse the left subtree
```

```
    result.push_back(root->val);       // Visit the current node
    inorderHelper(root->right, result); // Traverse the right subtree
}

// Function to return inorder traversal of a binary tree
vector<int> inorderTraversal(TreeNode* root) {
    vector<int> result;
    inorderHelper(root, result);
    return result;
}

// Driver code for testing
int main() {
    // Example: Create a binary tree
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);

    // Get the inorder traversal
    vector<int> result = inorderTraversal(root);

    // Print the result
    for (int val : result) {
        cout << val << " ";
    }
    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting.cpp -o
sorting && ./sorting
1 3 2
```

Q 2 (Medium) Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Example 1:

Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

Output: [3,9,20,null,null,15,7]

Program code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Helper function to build the tree recursively
TreeNode* buildTreeHelper(vector<int>& preorder, vector<int>& inorder,
                          int preStart, int preEnd,
                          int inStart, int inEnd,
                          unordered_map<int, int>& inMap) {
    if (preStart > preEnd || inStart > inEnd) return nullptr;

    // Root value is the first element in preorder
    int rootVal = preorder[preStart];
    TreeNode* root = new TreeNode(rootVal);

    // Find the index of root in inorder array
    int inRoot = inMap[rootVal];
    int numsLeft = inRoot - inStart; // Number of nodes in the left subtree

    // Recursively build left and right subtrees
    root->left = buildTreeHelper(preorder, inorder,
                                 preStart + 1, preStart + numsLeft,
                                 inStart, inRoot - 1,
                                 inMap);
    root->right = buildTreeHelper(preorder, inorder,
                                  preStart + numsLeft + 1, preEnd,
                                  inRoot + 1, inEnd,
```

```
                        inMap);

    return root;
}

// Main function to build the tree
TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
    unordered_map<int, int> inMap; // Map to store index of each value in inorder
    for (int i = 0; i < inorder.size(); ++i) {
        inMap[inorder[i]] = i;
    }
    return buildTreeHelper(preorder, inorder,
                           0, preorder.size() - 1,
                           0, inorder.size() - 1,
                           inMap);
}

// Function to print the tree in level-order for verification
void printLevelOrder(TreeNode* root) {
    if (!root) return;
    vector<TreeNode*> level = {root};
    while (!level.empty()) {
        vector<TreeNode*> nextLevel;
        for (TreeNode* node : level) {
            if (node) {
                cout << node->val << " ";
                nextLevel.push_back(node->left);
                nextLevel.push_back(node->right);
            } else {
                cout << "null ";
            }
        }
        level = nextLevel;
    }
}

// Driver code
int main() {
    vector<int> preorder = {3, 9, 20, 15, 7};
    vector<int> inorder = {9, 3, 15, 20, 7};

    TreeNode* root = buildTree(preorder, inorder);
```

```
    // Print the tree in level-order
    printLevelOrder(root);
    return 0;
}
```

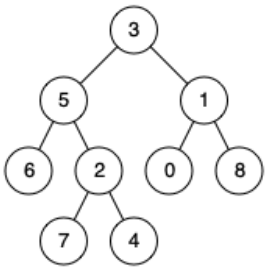
Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting2.cpp -o
 sorting2 && ./sorting2
3 9 20 null null 15 7 null null null null
```

Ques 3 Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).

Example 1:



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.

Program Code:

```
#include <iostream>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
```

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

// Function to find the lowest common ancestor
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
{
    if (root == nullptr || root == p || root == q) return root;

    // Recursively find LCA in left and right subtrees
    TreeNode* left = lowestCommonAncestor(root->left, p, q);
    TreeNode* right = lowestCommonAncestor(root->right, p, q);

    // If p and q are found in different subtrees, root is their LCA
    if (left != nullptr && right != nullptr) return root;

    // If one subtree returns null, LCA is in the other subtree
    return (left != nullptr) ? left : right;
}

// Helper function to test the solution
int main() {
    // Example binary tree
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(5);
    root->right = new TreeNode(1);
    root->left->left = new TreeNode(6);
    root->left->right = new TreeNode(2);
    root->right->left = new TreeNode(0);
    root->right->right = new TreeNode(8);
    root->left->right->left = new TreeNode(7);
    root->left->right->right = new TreeNode(4);

    TreeNode* p = root->left;  // Node 5
    TreeNode* q = root->right; // Node 1

    TreeNode* lca = lowestCommonAncestor(root, p, q);
    cout << "LCA of " << p->val << " and " << q->val << " is: " << lca->val <<
endl;

    return 0;
```

}

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting3.cpp -o
 sorting3 && ./sorting3
LCA of 5 and 1 is: 3
```

Ques 4. Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Example 1:

Input: root = [1,2,3,null,5,null,4]

Output: [1,3,4]

Program Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Function to return the right-side view of a binary tree
vector<int> rightSideView(TreeNode* root) {
    vector<int> result;
    if (root == nullptr) return result;

    queue<TreeNode*> q; // Queue for level-order traversal
    q.push(root);
```

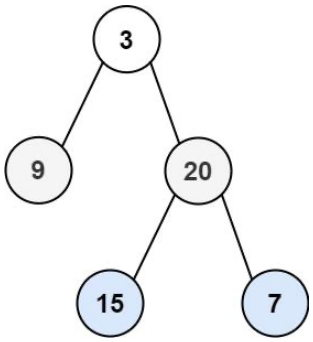
```
    while (!q.empty()) {
        int levelSize = q.size();
        for (int i = 0; i < levelSize; ++i) {
            TreeNode* current = q.front();
            q.pop();

            // If it's the last node at the current level, add it to the result
            if (i == levelSize - 1) {
                result.push_back(current->val);
            }

            // Add the child nodes to the queue
            if (current->left) q.push(current->left);
            if (current->right) q.push(current->right);
        }
    }

    return result;
}

// Driver code for testing
int main() {
    // Example: Create a binary tree
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->right = new TreeNode(5);
    root->right->right = new TreeNode(4);

    // Get the right-side view
    vector<int> result = rightSideView(root);

    // Print the result
    for (int val : result) {
        cout << val << " ";
    }
    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting4.cpp -o
 sorting4 && ./sorting4
1 3 4 2
```

Ques 5 Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

Example 1:



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

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

Program Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm> // For reverse function
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
// Function to return zigzag level order traversal of the binary tree
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>> result;
    if (root == nullptr) return result;

    queue<TreeNode*> q; // Queue for level-order traversal
    q.push(root);
    bool leftToRight = true; // Flag to determine the direction of traversal

    while (!q.empty()) {
        int levelSize = q.size();
        vector<int> level;

        for (int i = 0; i < levelSize; ++i) {
            TreeNode* current = q.front();
            q.pop();

            level.push_back(current->val);

            // Add child nodes to the queue
            if (current->left) q.push(current->left);
            if (current->right) q.push(current->right);
        }

        // Reverse the level order if the direction is right to left
        if (!leftToRight) {
            reverse(level.begin(), level.end());
        }
        result.push_back(level);

        // Toggle the direction for the next level
        leftToRight = !leftToRight;
    }

    return result;
}

// Driver code for testing
int main() {
    // Example: Create a binary tree
```

```
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);

    // Get the zigzag level order traversal
    vector<vector<int>> result = zigzagLevelOrder(root);

    // Print the result
    for (const auto& level : result) {
        for (int val : level) {
            cout << val << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting5.cpp -o
 sorting5 && ./sorting5
3
20 9
15 7
```