Name: Archi bansal **UID: 22BCS15264**
Section: 620/A **Date: 27/12/2024**

DOMAIN WINTER WINNING CAMP

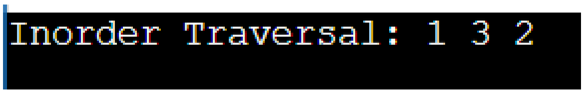
1. **Binary Tree Inorder Traversal. Given the root of a binary tree, return the inorder traversal of its nodes' values.**

Code:

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

// 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:
   vector<int> inorderTraversal(TreeNode* root) {
      vector<int> result;
      inorderHelper(root,
      result); return result;
   }

private:
   void inorderHelper(TreeNode* node, vector<int>&
      result) { if (!node) return;
      inorderHelper(node->left, result);  // Visit left
      subtree result.push_back(node->val);         // Visit
      root node inorderHelper(node->right, result); //
      Visit right subtree
   }
};

int main() {
   // Example usage:
    TreeNode* root = new
```

```
TreeNode(1); root->right = new
TreeNode(2);
root->right->left = new TreeNode(3);

Solution solution;
vector<int> inorder = solution.inorderTraversal(root);

cout << "Inorder Traversal:
"; for (int val : inorder) {
cout << val << " ";
}

return 0;
}
```

Output:



2. **Count Complete Tree Nodes. Given the root of a complete binary tree, return the number of the nodes in the tree.**

Code:

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

// 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:
int countNodes(TreeNode* root) { if
(!root) return 0;

int leftHeight = getHeight(root->left);  int
rightHeight = getHeight(root->right);

if (leftHeight == rightHeight) {
// Left subtree is a perfect binary tree
return (1 << leftHeight) + countNodes(root->right);
```

```
} else {
// Right subtree is a perfect binary tree
return (1 << rightHeight) + countNodes(root->left);
}
}

private:
int getHeight(TreeNode* node) {

  int height = 0;
  while (node) {
     height++;
         node = node->left; // Move down the leftmost path
}
return height;
}
};

int main() {
// Example usage:
TreeNode* root = new
TreeNode(1); root->left = new
TreeNode(2);
root->right = new TreeNode(3);
root->left->left = new
TreeNode(4);
root->left->right = new
TreeNode(5); root->right->left =
new TreeNode(6);

Solution solution;
int nodeCount = solution.countNodes(root);

cout << "Total nodes in the tree: " << nodeCount << endl;

return 0;
}
```

Output:

```
Total nodes in the tree: 6
```

3. **Binary Tree - Find Maximum Depth. A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.**

Code:

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

```
// 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:

int maxDepth(TreeNode* root) {
if (!root) return 0; // Base case: empty tree has depth 0

// Recursive case: find depth of left and right subtrees
int leftDepth = maxDepth(root->left);
int rightDepth = maxDepth(root->right);

// Return the larger depth + 1 for the current node
return max(leftDepth, rightDepth) + 1;
}
};

int main() {
// Example usage:
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);

Solution solution;
int depth = solution.maxDepth(root);

cout << "Maximum depth of the binary tree: " << depth << endl;

return 0;
}
```

```
Maximum depth of the binary tree: 3
```

Output:

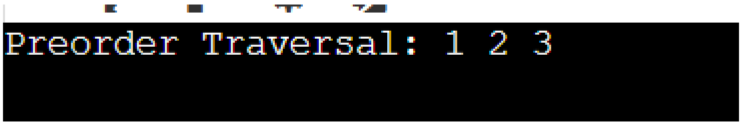
4. **Binary Tree Preorder Traversal. Given the root of a binary tree, return the preorder traversal of its nodes' values.**

Code:

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

// 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:
vector<int> preorderTraversal(TreeNode* root) {
vector<int> result;
preorderHelper(root, result);
return result;
}

private:
void preorderHelper(TreeNode* node, vector<int>& result) {
if (!node) return;
result.push_back(node->val);          // Visit root node
preorderHelper(node->left, result);  // Visit left subtree
preorderHelper(node->right, result); // Visit right subtree
}
};

int main() {
// Example usage:
TreeNode* root = new
TreeNode(1); root->right = new
TreeNode(2);
root->right->left = new TreeNode(3);

Solution solution;
vector<int> preorder = solution.preorderTraversal(root);

cout << "Preorder Traversal: ";
for (int val : preorder) {
cout << val << " ";
```

```
}

return 0;
}
```

Output:

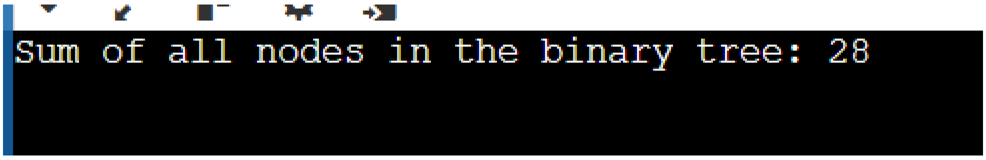


5. **Binary Tree - Sum of All Nodes. Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.**
   Code:

```
#include <iostream>
#include <queue> using

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:
int sumOfNodes(TreeNode* root) {
if (!root) return 0; // Base case: empty tree has a sum of 0
int sum = 0;
queue<TreeNode*> q;
q.push(root);
while (!q.empty()) { TreeNode*
node = q.front(); q.pop();
sum += node->val; // Add current node value to sum
// Push left and right children to the queue if they exist if
(node->left) q.push(node->left);
if (node->right) q.push(node->right);
}
return sum;
}};
int main() {
// Example usage:
TreeNode* root = new
TreeNode(1); root->left = new
TreeNode(2);
root->right = new TreeNode(3);
root->left->left = new
TreeNode(4);
```

```
root->left->right = new TreeNode(5);
root->right->left = new TreeNode(6);
root->right->right          =          new
TreeNode(7);

Solution solution;
int sum = solution.sumOfNodes(root);

cout << "Sum of all nodes in the binary tree: " << sum << endl;

return 0;
}
```

Output:



6. **Same Tree. Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.**

Code:

```
#include<iostream>
using namespace std;
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:
bool isSameTree(TreeNode* p, TreeNode* q) {
// Base cases:
if (!p && !q) return true;  // Both trees are empty
if (!p || !q) return false; // One tree is empty
return (p->val == q->val) && isSameTree(p->left, q->left) && isSameTree(p->right, q-
>right);
}};
int main() {
// Example usage:
TreeNode* root1 = new
TreeNode(1); root1->left = new
TreeNode(2);
root1->right = new TreeNode(3);
TreeNode* root2 = new
```

```
TreeNode(1); root2->left = new
TreeNode(2);
root2->right = new TreeNode(3);
Solution solution;
if (solution.isSameTree(root1, root2)) {
cout << "The two binary trees are the same." << endl;
} else {
cout << "The two binary trees are not the same." << endl;
}
return 0;
}
```

Output:

```
The two binary trees are the same.
```

7. **Invert Binary Tree. Given the root of a binary tree, invert the tree, and return its root.**
Code:

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

// 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* invertTree(TreeNode* root) {
if (!root) return nullptr; // Base case: empty tree

// Swap the left and right subtrees
TreeNode* temp = root->left; root-
>left = root->right;
root->right = temp;

// Recursively invert the left and right subtrees
invertTree(root->left);
invertTree(root->right);

return root;
```

```
}
};

void printTree(TreeNode* root) {
if (!root) return;
printTree(root->left);
cout << root->val << " ";
printTree(root->right);
}

int main() {
// Example usage:
TreeNode* root = new
TreeNode(4); root->left = new
TreeNode(2);
root->right = new TreeNode(7);
root->left->left = new
TreeNode(1);
root->left->right = new TreeNode(3);
root->right->left = new TreeNode(6);
root->right->right = new
TreeNode(9);

Solution solution;


cout << "Original Tree (Inorder Traversal): ";
printTree(root);
cout << endl;

TreeNode* invertedRoot = solution.invertTree(root);

cout << "Inverted Tree (Inorder Traversal): ";
printTree(invertedRoot);
cout << endl;

return 0;
}
```

Output:

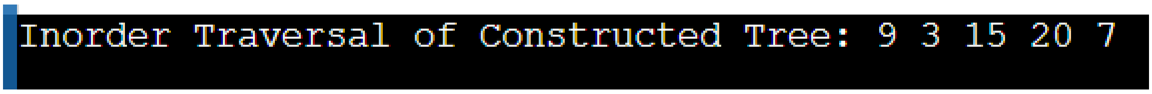
```
Original Tree (Inorder Traversal): 1 2 3 4 6 7 9
Inverted Tree (Inorder Traversal): 9 7 6 4 3 2 1
```

8. **Construct Binary Tree from Preorder and Inorder Traversal. 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.
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() : 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* buildTree(vector<int>& preorder, vector<int>& inorder) {
unordered_map<int, int> inorderMap;
for (int i = 0; i < inorder.size(); ++i) {
inorderMap[inorder[i]] = i;
}
    return buildTreeHelper(preorder, 0, preorder.size() - 1, inorder, 0, inorder.size() - 1,
inorderMap);
}

private:
TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int preEnd, vector<int>&
inorder, int inStart, int inEnd, unordered_map<int, int>& inorderMap) {
if (preStart > preEnd || inStart > inEnd) return nullptr;

// The first element of preorder is the root node
int rootVal = preorder[preStart];
TreeNode* root = new TreeNode(rootVal);

// Find the index of the root in the inorder array
int inRootIndex = inorderMap[rootVal];
int leftSubtreeSize = inRootIndex - inStart;

// Recursively build the left and right subtrees
ot->left = buildTreeHelper(preorder, preStart + 1, preStart + leftSubtreeSize, inorder, inStart,
inRootIndex - 1, inorderMap);
ot->right = buildTreeHelper(preorder, preStart + leftSubtreeSize + 1, preEnd, inorder,
inRootIndex + 1, inEnd, inorderMap);
```

```
    return root;
    }
    };
  void printInorder(TreeNode* root) {
    if (!root) return;
    printInorder(root->left);
    cout << root->val << " ";
    printInorder(root->right);
    }

    int main() {
    // Example usage:
    vector<int> preorder = {3, 9, 20, 15, 7};
    vector<int> inorder = {9, 3, 15, 20, 7};
    Solution solution;
    TreeNode* root = solution.buildTree(preorder, inorder);
    cout << "Inorder Traversal of Constructed Tree: ";
    printInorder(root);
    cout << endl;
    return 0;
    }
```



Output:

9. **Binary Tree Maximum Path Sum. A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root. The path sum of a path is the sum of the node's values in the path. Given the root of a binary tree, return the maximum path sum of any non-empty path.**

Code:

```
   #include <iostream>
#include <algorithm>
#include <climits> // Include this header for INT_MIN
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) {}
};

class Solution {
public:
  int maxPathSum(TreeNode* root) {
    int max_sum = INT_MIN; // Initialize the maximum path sum
```

```
        // Helper function to calculate the maximum path sum recursively
        helper(root, max_sum);

        return max_sum;
    }

private:
    // Helper function to compute the maximum path sum from each node
    int helper(TreeNode* node, int &max_sum) {
        if (node == nullptr) {
            return 0;
        }

        // Calculate the maximum path sum of the left and right subtrees
        int left = max(helper(node->left, max_sum), 0); // We discard negative sums
        int right = max(helper(node->right, max_sum), 0); // We discard negative sums

        // Calculate the path sum passing through the current node
        int current_sum = node->val + left + right;

        // Update the global maximum sum if the current path sum is greater
        max_sum = max(max_sum, current_sum);

        // Return the maximum sum for paths starting from the current node
        return node->val + max(left, right);
    }
};

int main() {
    // Example: Constructing a binary tree
    TreeNode* root = new TreeNode(-10);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);

    Solution solution;
    int result = solution.maxPathSum(root);

    cout << "Maximum path sum: " << result << endl;

    return 0;
}
```

Output:

```
Maximum path sum: 42
```

10. **Kth Smallest Element in a BST (Binary Search Tree). Given a binary search tree (BST), write a function to find the kth smallest element in the tree.**

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) {}
};

class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        int count = 0; // To count the number of nodes visited
        int result = -1; // To store the k-th smallest element
        inOrderTraversal(root, k, count, result);
        return result;
    }

private:
    void inOrderTraversal(TreeNode* node, int k, int& count, int& result) {
        if (!node || count >= k) return; // Stop if we reach the end or find the k-th smallest

        // Traverse the left subtree
        inOrderTraversal(node->left, k, count, result);

        // Visit the current node
        count++;
        if (count == k) {
            result = node->val;
            return;
        }

        // Traverse the right subtree
        inOrderTraversal(node->right, k, count, result);
    }
};

int main() {
    // Example: Constructing a binary search tree
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(1);
    root->right = new TreeNode(4);
    root->left->right = new TreeNode(2);

    Solution solution;
    int k = 2; // Example: Find the 2nd smallest element
    int result = solution.kthSmallest(root, k);

    cout << "The " << k << "-th smallest element is: " << result << endl;

    return 0;
```

```
}
```

Output: