`

DAY 3

Aryan **22BCS10190** **KPIT-901(A)**

Q1 . 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(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorder(root, result);
        return result;
    }

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

Q2. Count Complete Tree Nodes

```
#include <iostream>
#include <cmath>
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 countNodes(TreeNode* root) {
        if (!root) return 0;

        // Calculate left and right subtree heights
        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;
    }
};
```

Q3. Binary Tree - Find Maximum Depth

```
#include <iostream>
#include <algorithm> // For max 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) {}
};

class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (!root) return 0; // Base case: null node has depth 0
        // Calculate depths of left and right subtrees
        int leftDepth = maxDepth(root->left);
        int rightDepth = maxDepth(root->right);
        // Return the maximum of the two depths + 1 for the current node
        return 1 + max(leftDepth, rightDepth);
    }
};
```

Q4. Binary Tree Preorder Traversal

```
#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) {}
};

class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> result;
        preorder(root, result);
        return result;
    }

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

`

};

Q5. Binary Tree - Sum of All Nodes

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

class Solution {
public:
    int sumOfNodes(TreeNode* root) {
        if (!root) return 0;

        queue<TreeNode*> q;
        q.push(root);
        int totalSum = 0;

        while (!q.empty()) {
            TreeNode* node = q.front();
            q.pop();
            totalSum += node->val; // Add the current node value

            if (node->left) q.push(node->left); // Add left child
            if (node->right) q.push(node->right); // Add right child
        }

        return totalSum;
    }
};


int main() {
    // Construct the tree: [1, 2, 3, 4, 5, null, 6]
    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->right = new TreeNode(6);

    Solution sol;
    cout << "Sum of All Nodes: " << sol.sumOfNodes(root) << endl; // Output:
21
```

```
```
```
    return 0;
}
```