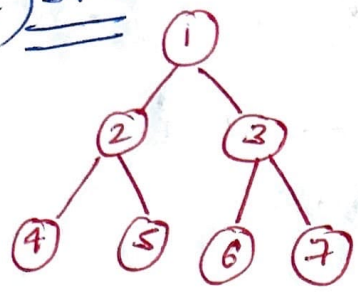
L4: Binary Tree Traversals in BT — BFS / DFS

(I) DFS



→ Inorder Traversal (Left Root Right)

4 2 5 1 6 3 7

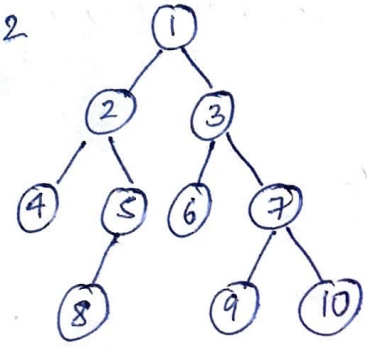
→ Preorder Traversal (Root Left Right)

1 2 4 5 3 6 7

→ Postorder Traversal (Left Right Root)

4 5 2 6 7 3 1

eg 2



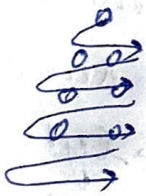
Inorder : 4, 2, 8, 5, 1, 6, 3, 9, 7, 10

Preorder : 1, 2, 4, 5, 8, 3, 6, 7, 9, 10

Postorder : 4, 8, 5, 2, 6, 9, 10, 7, 3, 1

(II) BFS

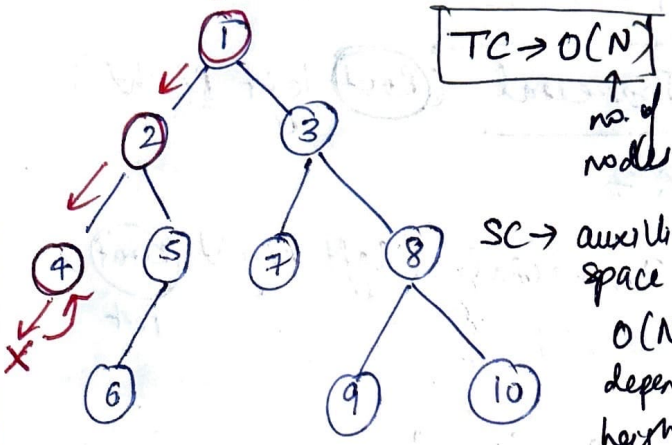
1, 2, 3, 4, 5, 6, 7, 8, 9, 10



goes level-wise

LS : Preorder Traversal Code

RLR
Root Left Right



TC → O(N) ← no. of nodes

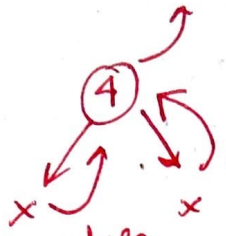
SC → auxiliary space
O(N) depends on height of tree.

```
void preorder(node) {
    if (node == null) {
        return;
    }
    print(node->data);
    preorder(node->left);
    preorder(node->right);
}
```

1 (root)

1, 2, 4,

Dry Run

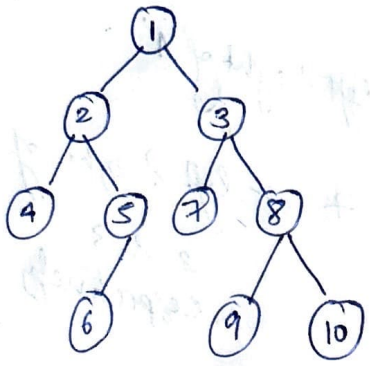


Code :-

```
#include <iostream>
using namespace std;
struct node {
    int data;
    struct node * left;
    struct node * right;
    node (int val) {
        data = val;
        left = right = NULL;
    }
};
void preorder_traverse (struct node * root) {
    if (root == NULL) return;
    cout << root -> data << " ";
    preorder_traverse (root -> left);
    preorder_traverse (root -> right);
}
int main() {
    struct node * root = new node(1);
    root -> left = new node (2);
    root -> right = new node (3);
    root -> left -> left = new node (4);
    root -> left -> right = new node (5);
    root -> left -> right -> left = new
                node (6);
    cout << "Pre order traversal";
    preorder_traverse (root);
    return 0;
}
```

L6: Inorder Traversal Code

Left Root Right



inorder: 4, 2, 6, 5, 1, 7, 3,
9, 8, 10

postorder:- 4, 6, 5, 2, 7,
9, 10, 8, 3, 1

L7:- Postorder Traversal

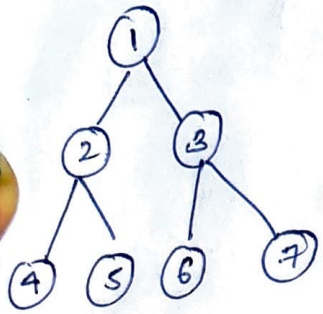
left right root

```
void postorder_traverse (struct
    node *root){
if (root==NULL) return;
postorder_traverse (root->left);
postorder_traverse (root->right);
cout<< root->data << " ";
}
```

CODE

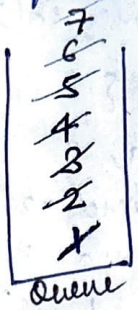
```
#include <iostream>
using namespace std;
struct node {
    int data;
    struct node *left;
    struct node *right;
    node (int val) {
        data = val;
        left = right = NULL;
    }
};

void inorder_traverse ( struct node *root){
    if (root==NULL) return;
    inorder_traverse (root->left);
    cout << root->data<<" ";
    inorder_traverse (root->right);
}

int main(){
    struct node *root = new node (1);
    root->left= new node (2);
    root->right= new node (3);
    root->left->left = new node (4);
    root->left->right = new node (5);
    root->left->right->left = new node (6);
    :
    inorder_traverse (root);
    cout<< endl;
    postorder_traverse (root);
    return 0;
}
```

L 8 - Level Order Traversal of Binary Trees (BFS)



Binary Tree

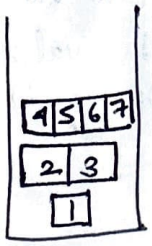
Using queue



1

2 3 ← left & right of 1

4 5 6 7 ← left & right of 2 & 3 respectively



TC = O(N)

SC = O(N)

important

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if (root == nullptr) return ans;
        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();
                if (node->left != NULL) q.push(node->left);
                if (node->right != NULL) q.push(node->right);
                level.push_back(node->val);
            }
            ans.push_back(level);
        }
        return ans;
    }
};
```