Preorder Traversal:

```
class Solution {
public:
    typedef TreeNode Node;
    void inorderT(Node*root,vector<int>& ds){
        if(root==NULL){
            return;
        }
        ds.push_back(root->val);
        inorderT(root->left,ds);
        inorderT(root->right,ds);
    }
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int>ds;
        inorderT(root,ds);
        return ds;
    }
};
```

Time:O(no. Of nodes)
Space:O(1)

Inorder Traversal:

```
class Solution {
public:
    typedef TreeNode Node;
    void inorderT(Node*root,vector<int>& ds){
        if(root==NULL){
            return;
        }
         inorderT(root->left,ds);
         ds.push_back(root->val);
           inorderT(root->right,ds);
    }
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int>ds;
        inorderT(root,ds);
        return ds;
    }
};
```

Time:O(no. Of nodes)
Space:O(1)

Postorder Traversal:

```
class Solution {
public:
    typedef TreeNode Node;
    void inorderT(Node*root,vector<int>& ds){
        if(root==NULL){
            return;
        }
        inorderT(root->left,ds);
        inorderT(root->right,ds);
        ds.push_back(root->val);

    }
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int>ds;
        inorderT(root,ds);
        return ds;
    }
};
```

Time:O(no. Of nodes)
Space:O(1)

Validate a BST:

```
class Solution {
public:

    typedef TreeNode Node;
    Node*prev=NULL;
    bool ans=true;


    void isBST(Node*root){
        if(root==NULL){
            return;
        }
        isBST(root->left);
        if(prev!=NULL){
            if(prev->val>=root->val){
                ans=false;
                return;
            }
        }
        prev=root;
        isBST(root->right);
    }
    bool isValidBST(TreeNode* root) {
        isBST(root);
        return ans;
    }
};
```

Time:O(n)
Space:O(1)

Max Depth of a Binary Tree:

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        TreeNode*temp=root;
        if(temp==NULL){
            return 0;
        }
        int ans=1+maxDepth(root->left);
            int answ=1+maxDepth(root->right);
            return max(ans,answ);
    }
};
```

Time:O(n)
Space:Recursion Stack Space

Level Order Traversal:

```
 vector<vector<int>> res;
        if(!root) return res;
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()){
            // storing current q size helps to process nodes of current row only
            int n=q.size();
            vector<int> temp;
            for(int i=0;i<n;i++){
                TreeNode *parent=q.front();
                q.pop();
                //store nodes at current level in temp result
                temp.push_back(parent->val);
                //push next row nodes/child nodes in queue to be processed in next
                // iteration
                if(parent->left) q.push(parent->left);
                if(parent->right) q.push(parent->right);

            }
            //at this point we have nodes of current level stored in temp vector
            res.push_back(temp);

        }

        return res;
```

Time:O(N)
Space:O(Max Nodes at any level in queue)

Zig-Zag Traversal:

```
class Solution {
public:
    typedef TreeNode Node;
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL){
            return ans;
        }
        queue<Node*>q;
        q.push(root);
        bool leftToRight=true;
        while(!q.empty()){
            int n=q.size();
            vector<int>ds;
            for(int i=0;i<n;i++){
                Node*node=q.front();
                q.pop();

                ds.push_back(node->val);

                if(node->left!=NULL){
                    q.push(node->left);
                }
                if(node->right!=NULL){
                    q.push(node->right);
                }
            }
            if(leftToRight==true){
                ans.push_back(ds);
            }
            else{
                reverse(ds.begin(),ds.end());
                ans.push_back(ds);
            }
            leftToRight=!leftToRight;
        }
        return ans;
    }
};
```

Time:O(N)
Space:O(Max Nodes at any level in queue)