**Tree Assignment 1 done by N S K K K Naga Jayanth**

https://leetcode.com/problems/binary-tree-inorder-traversal/

```
class Solution {
    List<Integer> list = new ArrayList<>();
    public List<Integer> inorderTraversal(TreeNode root) {
        if (root!=null) {
            inorderTraversal(root.left);
            list.add(root.val);
            inorderTraversal(root.right);
        }
        return list;
    }
}
```

https://leetcode.com/problems/binary-tree-level-order-traversal/

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> ans = new ArrayList<>();
        Queue<TreeNode> q = new LinkedList<>();
        if(root == null) return ans;
        q.add(root);
        while( !q.isEmpty() ){
            int qSize = q.size();
            List<Integer> ls = new ArrayList<>();
            for(int i = 0; i < qSize; i++){
                TreeNode n = q.poll();
                ls.add(n.val);
                if(n.left!=null)q.add(n.left);
                if(n.right!=null)q.add(n.right);
            }
            ans.add(ls);
        }
        return ans;
    }
}
```

https://leetcode.com/problems/binary-tree-preorder-traversal/

```
class Solution {
    List<Integer>a=new ArrayList<>();
    public List<Integer> preorderTraversal(TreeNode root) {
        traverse(root);
        return a;
    }

    public void traverse(TreeNode root){
        if(root==null){
            return;
        }
        a.add(root.val);
        traverse(root.left);
        traverse(root.right);
    }
}
```

https://leetcode.com/problems/binary-tree-postorder-traversal/

```
class Solution {
    ArrayList<Integer> list;
    public List<Integer> postorderTraversal(TreeNode root) {
        list = new ArrayList<>();
        postorder(root);
        return list;
    }
    public void postorder(TreeNode root){
        if(root==null) return;
        postorder(root.left);
        postorder(root.right);
        list.add(root.val);
    }
}
```

https://leetcode.com/problems/maximum-depth-of-binary-tree/

```
class Solution {
    public int maxDepth(TreeNode root) {
        if(root == null){
            return 0;
        }
        int lh = maxDepth(root.left);
        int rh = maxDepth(root.right);
        return  Math.max(lh,rh)+1;
    }
}
```

https://leetcode.com/problems/symmetric-tree/

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        return helper(root.left, root.right);
    }

    private boolean helper(TreeNode p, TreeNode q) {
        if(p == null || q == null) return p == q;

        return p.val == q.val && helper(p.left, q.right) && helper(p.right, q.left);
    }
}
```

https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree/

```
class Solution {
    public int maxLevelSum(TreeNode root) {
        if(root == null) return 1;
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        int level = 0, ans = 0, sum = Integer.MIN_VALUE;
        while(!q.isEmpty()){
            int size = q.size();
            int max = 0;
            for(int i = 0; i < size; i++){
                TreeNode curr = q.poll();
                max = max + curr.val;
                if(curr.left != null){
                    q.add(curr.left);
                }
                if(curr.right != null){
                    q.add(curr.right);
                }
            }
            level++;
            if(max > sum){
                ans = level;
                sum = max;
            }
        }
        return ans;
    }
}
```

https://leetcode.com/problems/sum-root-to-leaf-numbers/

```
class Solution {
    public int sumNumbers(TreeNode root) {
        // takes root and initial sum (which is 0 at the start)
        return inorder(root, 0);
    }
    private int inorder(TreeNode root, int num) {

        if (root.left == null && root.right == null) return num * 10 + root.val;

        num = num * 10 + root.val;
        int left = 0;
        int right = 0;
        if (root.left != null) {
            left += inorder(root.left, num);
        }
        if (root.right != null) {
            right += inorder(root.right, num);
        }
        return left + right;
    }
}
```

https://www.interviewbit.com/problems/vertical-order-traversal-of-binary-tree/

```
public class Solution {
    public ArrayList<ArrayList<Integer>> verticalOrderTraversal(TreeNode A) {
        ArrayList<ArrayList<Integer>> result = new ArrayList<>();
        if (A == null) {
            return result;
        }

        TreeMap<Integer, ArrayList<Integer>> map = new TreeMap<>();
        Queue<TreeNode> queue = new LinkedList<>();
        Queue<Integer> hdQueue = new LinkedList<>();

        queue.offer(A);
        hdQueue.offer(0);
        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            int hd = hdQueue.poll();

            // Update TreeMap with horizontal distance as key
            map.putIfAbsent(hd, new ArrayList<>());
            map.get(hd).add(node.val);

            // Enqueue left child with horizontal distance - 1
            if (node.left != null) {
                queue.offer(node.left);
                hdQueue.offer(hd - 1);
            }

            // Enqueue right child with horizontal distance + 1
            if (node.right != null) {
                queue.offer(node.right);
                hdQueue.offer(hd + 1);
            }
        }

        // Populate the result list from TreeMap values
        for (ArrayList<Integer> list : map.values()) {
            result.add(list);
        }

        return result;
    }
}
```

https://leetcode.com/problems/binary-tree-right-side-view/description/

```
class Solution {
   int maxLevel = 0;
   List<Integer> list = new ArrayList();
   public List<Integer> rightSideView(TreeNode root) {
      if(root == null) return list;
      rightView(root,1);
      return list;
   }
   void rightView(TreeNode root,int level){
      if(root == null) return;
      if(maxLevel < level){
         list.add(root.val);
         maxLevel = level;
      }
      rightView(root.right,level+1);
      rightView(root.left,level+1);
   }
}
```

https://practice.geeksforgeeks.org/problems/left-view-of-binary-tree/1?

```
/* A Binary Tree node
class Node
{
   int data;
   Node left, right;

   Node(int item)
   {
      data = item;
      left = right = null;
   }
}*/
```

```
class Tree
{
    //Function to return list containing elements of left view of binary tree.
    ArrayList<Integer> leftView(Node root)
    {
        ArrayList<Integer> result = new ArrayList<>();
        if (root == null) {
            return result;
        }

        Queue<Node> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                Node node = queue.poll();
                // For the leftmost node of each level, add it to the result list
                if (i == 0) {
                    result.add(node.data);
                }
                // Enqueue left child
                if (node.left != null) {
                    queue.offer(node.left);
                }
                // Enqueue right child
                if (node.right != null) {
                    queue.offer(node.right);
                }
            }
        }

        return result;
    }
}
```