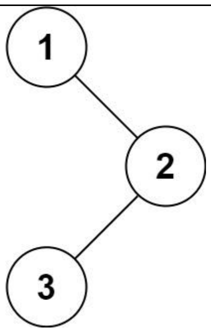
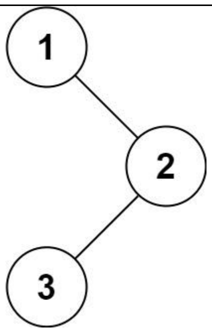
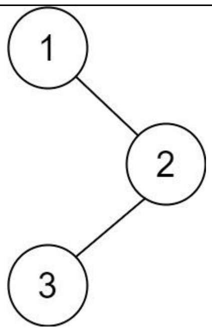
Binary Tree Preorder/Inorder/Postorder Traversal

Given the `root` of a binary tree, return *the preorder / inorder / postorder traversal of its nodes' values*.

Preorder	Inorder	Postorder
		
Example 1: **Input:** root = [1,null,2,3] **Output:** [1,2,3]	**Example 1:** **Input:** root = [1,null,2,3] **Output:** [1,3,2]	**Example 1:** **Input:** root = [1,null,2,3] **Output:** [3,2,1]
Example 2: **Input:** root = [] **Output:** []	**Example 2:** **Input:** root = [] **Output:** []	**Example 2:** **Input:** root = [] **Output:** []
Example 3: **Input:** root = [1] **Output:** [1]	**Example 3:** **Input:** root = [1] **Output:** [1]	**Example 3:** **Input:** root = [1] **Output:** [1]

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */



public class Solution {
    public IList<int> OrderTraversal(TreeNode root) {
        List<int> r = new List<int>();
       Traverse(root,r);
        return r;
    }

    PreOrder
    void Traverse(TreeNode root, IList<int> retVal)
    {
        if(root == null)
        {
            return;
        }

        retVal.Add(root.val);
        Traverse(root.left,retVal);
        Traverse(root.right,retVal);

    }


    InOrder
    void Traverse(TreeNode root, IList<int> retVal)
    {
        if(root == null)
        {
            return;
        }

        Traverse(root.left, retVal);
        retVal.Add(root.val);
        Traverse(root.right, retVal);
    }


    Post Order
    void Traverse(TreeNode root, IList<int> retVal)
    {
        if(root == null)
        {
            return;
        }

        Traverse(root.left, retVal);
        Traverse(root.right, retVal);
        retVal.Add(root.val);
    }
}
```