590. N-ary Tree Postorder Traversal
We could use dfs to traverse all nodes. TC is O(n)
```
class Solution:
    def postorder(self, root: 'Node') -> List[int]:
        res = []
        def traverse(node, res):
            for i in node.children:
                traverse(i, res)
            res.append(node.val)

        if not root:
            return []
        traverse(root, res)
        return res
```

100. Same Tree
We will use dfs to traverse all nodes and return False directly if any two nodes don’t match. TC is O(n)
```
class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        if p and q:
            if p.val == q.val:
                if not self.isSameTree(p.left, q.left):
                    return False
                if not self.isSameTree(p.right, q.right):
                    return False
                return True
            return False
        elif p or q:
            return False
        else:
            return True
```

101. Symmetric Tree
We will use recursion to traverse all nodes symmetrically. Return False if any two nodes are not eqaul. TC is O(n). SC is O(logn)
```
class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        if not root:
            return True
        return self.isMirror(root.left, root.right)

    def isMirror(self, left, right):
        if not left and not right:
            return True
        if not left or not right:
            return False

        if left.val == right.val:
            return self.isMirror(left.left, right.right) and self.isMirror(left.right, right.left)
        else:
            return False
```

104. Maximum Depth of Binary Tree
We will use recursion to get maximum depth of left and right branches. TC is O(n)

```
class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if not root:
            return 0

        left = self.maxDepth(root.left)
        right = self.maxDepth(root.right)
        return max(left, right) + 1
```

110. Balanced Binary Tree
We will use recursion to check evey node’s left branch tree level and right branch tree level to check whether it’s a balanced binary tree. TC is O(n)

```
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        def traverse(node):
            if not node:
                return 0
            left = traverse(node.left)
            right = traverse(node.right)
            if left is False or right is False:
                return False
            if abs(left - right) > 1:
                return False
            return max(left, right) + 1

        if traverse(root) is False:
            return False
        return True
```