111. Minimum Depth of Binary Tree
Return longest length if node is leaf or has one branch, or return the minimum one. TC is O(n)

```
class Solution:
    def minDepth(self, root: TreeNode) -> int:
        if not root:
          return 0
        left = self.minDepth(root.left)
        right = self.minDepth(root.right)
        return left + right + 1 if left * right == 0 else min(left, right) + 1
```

110. Balanced Binary Tree
We will go through our tree and return False if any of its subtree is not balanced.

```
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 == -1 or right == -1 or abs(right - left) > 1:
          return -1
        return max(left, right) + 1
      return False if traverse(root) == -1 else True
```

100. Same Tree
We will check each node of p and q. Return false if any node is not the same. TC is O(n)

```
class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
      if p and q:
        return p.val == q.val and self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)
      return p == None and q == None
```

101. Symmetric Tree
We will solve it using recursion. TC is O(n)

```
class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        def traverse(left_node, right_node):
          if left_node == None or right_node == None:
            return left_node == right_node
          return left_node.val == right_node.val and traverse(left_node.left, right_node.right) and
traverse(left_node.right, right_node.left)
        return True if not root else traverse(root.left, root.right)
```

104. Maximum Depth of Binary Tree
We will use recursion to solve this question. TC is O(n)

```
class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if not root:
          return 0
        return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1
```