277. Find the Celebrity
We will iterate through all nodes and use know to check whether the candidate knows other nodes, if it does, we will assign that node as our candidate, Then in the second iteration, we will check it whether it's real. TC is O(n)

```
class Solution(object):
    def findCelebrity(self, n):
        """
        :type n: int
        :rtype: int
        """
        candidate = 0
        for i in range(1, n):
            if knows(candidate, i):
                candidate = i

        for i in range(n):
            if i == candidate:
                continue

            if knows(candidate, i) or not knows(i, candidate):
                return -1

        return candidate
```

111. Minimum Depth of Binary Tree
We will use recursion to find out min depth. TC is O(n)

```
class Solution(object):
    def minDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        def traverse(node):
            if not node:
                return 0
            left = traverse(node.left)
            right = traverse(node.right)
            if left == 0 and right == 0:
                return 1
            else:
                if left == 0:
                    return 1 + right
```

```
            elif right == 0:
                return 1 + left
            return 1 + min(left, right)
        return traverse(root)
```

572. Subtree of Another Tree
We will traverse all s's nodes and check whther t and s are the same tree. TC is O(n * m)

```
class Solution(object):
    def isSubtree(self, s, t):
        """
        :type s: TreeNode
        :type t: TreeNode
        :rtype: bool
        """
        if not s and not t:
            return True

        if s and t:
            if s.val == t.val:
                if self.isSame(s, t):
                    return True
            if s.left:
                if self.isSubtree(s.left, t):
                    return True
            if s.right:
                if self.isSubtree(s.right, t):
                    return True
        return False


    def isSame(self, s, t):
        if s and t:
            if s.val == t.val:
                return self.isSame(s.left, t.left) and self.isSame(s.right, t.right)
            else:
                return False
        else:
            return s is t
```

965. Univalued Binary Tree
We will traverse all nodes preorder, once we found there is any different value, we will return False. Or in the end, we will return True. TC is O(n)

```
class Solution(object):
```

```
    def isUnivalTree(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        def traverse(node, k):
            if not node:
                return True
            if node.val != k:
                return False
            if not traverse(node.left, k):
                return False
            if not traverse(node.right, k):
                return False
            return True
        return traverse(root, root.val)
```

102. Binary Tree Level Order Traversal

We will use level traverse to iterate through all levels and append each level of values to res. In the end, we will return res.TC is O(n)

```
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        cur = [root]
        res = []
        while cur:
            next_cur = []
            vals = []
            for node in cur:
                vals.append(node.val)
                if node.left:
                    next_cur.append(node.left)
                if node.right:
                    next_cur.append(node.right)
            res.append(vals)
            cur = next_cur
        return res
```