11. Container With Most Water
We will move from both sides and always record the maximum area of the current situation, and then always move the lower side. TC is O(n), SC is O(1)

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        left, right = 0, len(height) - 1
        result = 0
        while left < right:
            result = max(result, (right - left) * min(height[left], height[right]))
            if height[left] < height[right]:
              left += 1
            else:
              right -= 1
        return result
```

872. Leaf-Similar Trees
We will use inorder traverse to go through all nodes and append leaf node's value to array. In the end, we will return res1 == res2. TC is O(n)

```
class Solution(object):
    def leafSimilar(self, root1, root2):
        """
        :type root1: TreeNode
        :type root2: TreeNode
        :rtype: bool
        """
        res1, res2 = [], []
        def traverse(node, res):
          if not node:
            return
          traverse(node.left, res)
          if not node.left and not node.right:
            res.append(node.val)
          traverse(node.right, res)
        traverse(root1, res1)
        traverse(root2, res2)
        return res1 == res2
```

987. Vertical Order Traversal of a Binary Tree
We will collect our nodes by columns and sort by (level, num), TC is O(nlogn)

```
from collections import defaultdict
class Solution(object):
    def verticalTraversal(self, root):
        """
```

```
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        memo = defaultdict(list)
        res = []
        def traverse(node, count, level):
          if not node:
            return
          memo[count].append((level, node.val))
          traverse(node.left, count - 1, level + 1)
          traverse(node.right, count + 1, level + 1)
        traverse(root, 0, 0)
        for i in sorted(memo.keys()):
          res.append(list(map(lambda a: a[1], sorted(memo[i]))))
        return res
```

107. Binary Tree Level Order Traversal II

```
class Solution:
    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:
        result = []
        cur_level = [root]
        next_level = []
        if not root:
          return result
        while cur_level:
          result.append([node.val for node in cur_level])
          for node in cur_level:
            if node.left:
              next_level.append(node.left)
            if node.right:
              next_level.append(node.right)
          cur_level = next_level
          next_level = []
        return result[::-1]
```

429. N-ary Tree Level Order Traversal

```
class Solution:
    def levelOrder(self, root: 'Node') -> List[List[int]]:
        if not root:
          return []
        cur = [root]
        res = []
        while cur:
          next_ite = []
          temp = []
```

```
  for i in cur:
    temp.append(i.val)
    next_ite.extend(i.children)
  res.append(temp)
  cur = next_ite
return res
```