50. Pow(x, n)
We will use multiply result when n > 1, each time we will divide n by 2. We also will use a carry to multiply number when n is odd. When n is negative, we should transform it to -n and in the end use 1/result as our final result. TC is Logn
```
class Solution:
    def myPow(self, x: float, n: int) -> float:
        result = x
        carry = 1
        if n == 0:
            return 1
        if n > 0:
            mark = True
        else:
            n = -n
            mark = False
        while n > 1:
            if n % 2 == 1:
                carry *= result
            n //= 2
            if n > 0:
                result *= result

        result *= carry
        return result if mark else 1 / result
```

443. String Compression
We will use two index to traverse the whole string. We will use left index to modify previous string and right index to count how many similar characters there are to encode. TC is O(n), SC is O(1)
```
class Solution:
    def compress(self, chars: List[str]) -> int:
        left, right = 0, 0
        length = len(chars)
        while right < length:
            count = 0
            prev = None
            while right < length and (not prev or prev == chars[right]):
                count += 1
                if not prev:
                    prev = chars[right]
                    right += 1
                    continue
                else:
```

```
                right += 1
            chars[left] = prev
            left += 1
            if count > 1:
                for i in str(count):
                    chars[left] = i
                    left += 1
        return left
```

547. Friend Circles
We will use union find to calculate the number of friend circles. We will set each elements' parent as themselves, friend circles as rows=len(M). Then we will iterate each element on the top of the matrix. Once we find there is a 1 and they are not in the same tree, we will connect these two trees and reduce n by 1. TC is O(n logn)

```
class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:
        rows = len(M)
        count = rows
        self.parent = {}

        for i in range(rows):
            self.parent[i] = i

        for i in range(rows):
            for j in range(i + 1, rows):
                if M[i][j] == 1:
                    parent_i = self.findParent(i)
                    parent_j = self.findParent(j)
                    if parent_i != parent_j:
                        self.parent[parent_j] = parent_i
                        count -= 1
        return count

    def findParent(self, i):
        while i != self.parent[i]:
            i = self.parent[i]
        return i
```

199. Binary Tree Right Side View
We will traverse by level and append last node's element to our result. TC is O(n)

```
class Solution:
    def rightSideView(self, root: TreeNode) -> List[int]:
        if not root:
            return []

        cur, result = [root], []

        while cur:
            next_ite = []
            result.append(cur[-1].val)
            for node in cur:
                if node.left:
                    next_ite.append(node.left)
                if node.right:
                    next_ite.append(node.right)
            cur = next_ite
        return result
```

445. Add Two Numbers II
We will use two variables to get numbers from two linked lists. And sum them up. Then create a new linked list to store the sum. TC is O(n)

```
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        num1, num2 = 0, 0
        dummy = ListNode(0)
        dummy_mem = dummy

        while l1:
            num1 = num1 * 10 + l1.val
            l1 = l1.next

        while l2:
            num2 = num2 * 10 + l2.val
            l2 = l2.next

        num = str(num1 + num2)

        for i in num:
            dummy.next = ListNode(int(i))
            dummy = dummy.next
        return dummy_mem.next
```