287. Find the Duplicate Number
We will use similar way to find out the entry point in a cycled linked list. Use slow and fast points to find a meeting point in the cycle. Then let fast equal to 0, move slow and fast by one step until they meet. TC is O(n), SC is O(1).

```
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        slow, fast = nums[0], nums[nums[0]]
        while slow != fast:
            slow = nums[slow]
            fast = nums[nums[fast]]

        fast = 0
        while slow != fast:
            slow = nums[slow]
            fast = nums[fast]
        return fast
```

97. Interleaving String
We will all elements in s1 and s2, we will set memo[i][j] = (memo[i][j - 1] and s2[j - 1] == s3[i + j - 1]) or (memo[i - 1][j] and s1[i - 1] == s3[i + j - 1]) TC is O(m + n)

```
class Solution:
    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
        if len(s1) + len(s2) != len(s3):
            return False
        memo = [[False for j in range(len(s2) + 1)] for i in range(len(s1) + 1)]
        for i in range(len(s1) + 1):
            for j in range(len(s2) + 1):
                if i == 0 and j == 0:
                    memo[i][j] = True
                elif i == 0:
                    memo[i][j] = memo[i][j - 1] and s2[j - 1] == s3[i + j - 1]
                elif j == 0:
                    memo[i][j] = memo[i - 1][j] and s1[i - 1] == s3[i + j - 1]
                else:
                    memo[i][j] = (memo[i][j - 1] and s2[j - 1] == s3[i + j - 1]) or (memo[i - 1][j] and s1[i - 1] == s3[i + j - 1])
        return memo[-1][-1]
```

148. Sort List
We will use merge sort to sort these linked list. We will split linked list into two parts until there is only one or zero node. Then we will merge them recursively. TC is O(nlogn)

```
class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        if not head or not head.next:
            return head

        prev, slow, fast = None, head, head
        while fast and fast.next:
            prev = slow
            slow = slow.next
            fast = fast.next.next

        prev.next = None
        l1 = self.sortList(head)
        l2 = self.sortList(slow)
        return self.merge(l1, l2)

    def merge(self, l1, l2):
        dummy = ListNode(0)
        dummy_mem = dummy
        while l1 and l2:
            if l1.val < l2.val:
                dummy.next = l1
                l1 = l1.next
            else:
                dummy.next = l2
                l2 = l2.next
            dummy = dummy.next
        if l1:
            dummy.next = l1
        if l2:
            dummy.next = l2
        return dummy_mem.next
```

974. Subarray Sums Divisible by K

We will use prefix sum to record all previous sum % K. If sum < 0, we will add by K, for the convenience of adding K. TC is O(N)

```
class Solution:
    def subarraysDivByK(self, A: List[int], K: int) -> int:
        memo = [0] * (K + 1)
        memo[0] = 1
        cur_sum = 0
        count = 0
        for a in A:
```

```
            cur_sum = (cur_sum + a) % K
            if cur_sum < 0:
                cur_sum += K
            count += memo[cur_sum]
            memo[cur_sum] += 1
        return count
```

69. Sqrt(x)

We will use binary search to find number that num ** 2 <= x. TC is O(logn)

```
class Solution:
    def mySqrt(self, x: int) -> int:
        l, r = 0, x
        while l < r:
            mid = (l + r) // 2
            if mid ** 2 == x:
                return mid
            elif mid ** 2 < x:
                l = mid + 1
            else:
                r = mid - 1
        return l - 1 if l ** 2 > x else l
```