Q1. 3 sum closest

class Solution:

    def threeSumClosest(self, nums: List[int], target: int) -> int:

        nums.sort()

        n = len(nums)

        diff = float('inf')

        res = 0

        for i in range(n - 2):

            left = i + 1

            right = n - 1

            while left < right:

                s = nums[i] + nums[left] + nums[right]

                if abs(target - s) < diff:

                    diff = abs(target - s)

                    res = s

                if s < target:

                    left += 1

                else:

                    right -= 1

        return res

**Q2.** Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that: ● 0 <= a, b, c, d < n ● a, b, c, and d are distinct. ● nums[a] + nums[b] + nums[c] + nums[d] == targetclass Solution:

def fourSum(self, nums: List[int], target: int) -> List[List[int]]:

ans = set()

nums.sort()

for i in range(len(nums)):

for j in range(i+1,len(nums)):

k,l = j+1,len(nums)-1

while k<l:

s = nums[i]+nums[j]+nums[k]+nums[l]

if s == target:

ans.add((nums[i],nums[j],nums[k],nums[l]))

l-=1

k+=1

elif s > target:

l-=1

else:

k+=1

return ans

**Q3.**A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container.

If such an arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

class Solution:

def nextPermutation(self, nums: List[int]) -> None:

for curr in reversed(range(0, len(nums) - 1)):

if nums[curr] >= nums[curr + 1]:

continue

next = curr + 1

while next < len(nums) and nums[next] > nums[curr]:

next += 1

nums[curr], nums[next - 1] = nums[next - 1], nums[curr]

nums[curr+1:] = reversed(nums[curr+1:])

return nums

return nums.sort()

**Q4.**Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You must write an algorithm with O(log n) runtime complexity.

class Solution:

def searchInsert(self, nums: List[int], target: int) -> int:

left,right=0,len(nums)-1

while left<=right:

mid=(left+right)//2

if nums[mid]==target:

return mid

elif nums[mid]<target:

left=mid+1

else:

right=mid-1

return left

**Q5. Question 5** You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's. Increment the large integer by one and return the resulting array of digits.

class Solution:

def plusOne(self, digits: List[int]) -> List[int]:

d=int("".join(map(str, digits)))

d+=1

return list(str(d))

**Q6. Question 6** Given a non-empty array of integers nums, every element appears twice except for one. Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space. **def** Single( arr, n):

    result **=** arr[0]

**for** i **in** range(1,n):

        result **=** result ^ arr[i]

**return** result

Q7. **Question 7** You are given an inclusive range [lower, upper] and a sorted unique integer array nums, where all elements are within the inclusive range. A number x is considered missing if x is in the range [lower, upper] and x is not in nums. Return the shortest sorted list of ranges that exactly covers all the missing numbers. That is, no element of nums is included in any of the ranges, and each missing number is covered by one of the ranges.  
  
**Asnwer:**

class Solution:

def findMissingRanges(self, nums: List[int], lower: int, upper: int) -> List[str]:

def getRange(lo: int, hi: int) -> str:

if lo == hi:

return str(lo)

return str(lo) + '->' + str(hi)

if not nums:

return [getRange(lower, upper)]

ans = []

if nums[0] > lower:

ans.append(getRange(lower, nums[0] - 1))

for prev, curr in zip(nums, nums[1:]):

if curr > prev + 1:

ans.append(getRange(prev + 1, curr - 1))

if nums[-1] < upper:

ans.append(getRange(nums[-1] + 1, upper))

return ans

Q8. **Question 8** Given an array of meeting time intervals where intervals[i] = [starti, endi], determine if a person could attend all meetings.

def canAttendMeetings(intervals):

intervals.sort()

for i in range(len(intervals)-1):

if intervals[i][1] > intervals[i+1][0]:

return False

return True