**Array Assignment : 3**

**Question 1** Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to the target. Return the sum of the three integers.

You may assume that each input would have exactly one solution.

**Example 1:** Input: nums = [-1,2,1,-4], target = 1 Output: 2

**Explanation:** The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

class Solution:

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

        nums.sort()

        res = sum(nums[:3])

        #sum of 1st three elements

        for i in range(0, len(nums)-2):

# -2 becoz when we will left with only 3 elements we will need atleast 2 values for our l and r pointer.

            l,r = i+1, len(nums)-1

            while l<r:

                totalsum = nums[i]+nums[l]+nums[r]

                if nums[l]+nums[r] < target-nums[i]:

                    res = totalsum

                    l +=1

                elif nums[l]+nums[r] > target - nums[i]:

                    res= totalsum

                    r -=1

                else:

                    res = nums[i] + nums[l] + nums[r]

                    return res

        return res

💡 **Question 2** 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] == target

You may return the answer in any order.

**Example 1:** Input: nums = [1,0,-1,0,-2,2], target = 0 Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

class Solution:

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

        res, quad = [], []

        nums.sort()

        def kSum(k, start, target):

            if k!=2:

                for i in range(start, len(nums)-k+1): # since we need to keep k values left for our total sum.

                    if i>start and nums[i] == nums[i-1]:

                        continue

                    quad.append(nums[i])

                    kSum(k-1, i+1, target - nums[i] )

                    quad.pop()

                return

            l, r = start, len(nums)-1

            while l<r:

                if nums[l] + nums[r] > target:

                    r-=1

                elif nums[l] + nums[r] < target:

                    l +=1

                else:

                    res.append([quad, nums[l], nums[r]])

                    l+=1

                    while nums[l]==nums[l-1] and l<r:

                        l+=1

        kSum(4,0,target)

        return res

#we will use the recursion here instead of using 2 for loops, Suppose if k sum is given then we will have to use k for loops which can become really complex and the base case for the recursion will be 2sum II or 2 pointer i.e when k =2

**Question 3** 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 in the lowest possible order (i.e., sorted in ascending order).

● For example, the next permutation of arr = [1,2,3] is [1,3,2]. ● Similarly, the next permutation of arr = [2,3,1] is [3,1,2]. ● While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement.

Given an array of integers nums, find the next permutation of nums. The replacement must be in place and use only constant extra memory.

**Example 1:** Input: nums = [1,2,3] Output: [1,3,2]

**CODE:**

**def** nextPermutation(nums):

n **=** len(nums)

i **=** n **-** 2

*# Find the first decreasing element from the right*

**while** i **>=** 0 **and** nums[i] **>=** nums[i**+**1]:

i **-=** 1

**if** i **>=** 0:

*# Find the smallest element greater than nums[i]*

j **=** n **-** 1

**while** j **>** i **and** nums[j] **<=** nums[i]:

j **-=** 1

*# Swap nums[i] and nums[j]*

nums[i], nums[j] **=** nums[j], nums[i]

*# Reverse the subarray nums[i+1:]*

left **=** i **+** 1

right **=** n **-** 1

**while** left **<** right:

nums[left], nums[right] **=** nums[right], nums[left]

left **+=** 1

right **-=** 1

**return** nums

**Question 4** 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.

**Example 1:** Input: nums = [1,3,5,6], target = 5 Output: 2

Algorithm: Binary search

1. Since the array is sorted we can use the binary search algorithm here.
2. According to binary search first we will find the midpoint of the array and check where the target value lies; either the right side of the midpoint or left side.
3. If target is greater than the midpoint then we will change the value of left pointer to mid+1 or else the right pointer to mid-1.
4. And repeat.
5. At the end we return the left pointer, just in case the target is missing, here we are returning left and not right or mid because if you take one value inside the array for ex: [2] and the target is 3 which is greater than the mid we will move left pointer to mid+1 and so it will be at place where 3 would have been in the array. And if 1 is target then the target is less than mid value therefore R will shift towards -1 which is not a valid index therefore we will return the 0th index where left lies.

CODE:

class Solution:

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

        l = 0

        r = len(nums)-1

        while l<=r:

            mid = (l+r) // 2

            if target == nums[mid]:

                return mid

            if nums[mid] > target:

                r = mid-1

            else:

                l = mid+1

        return l

**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.

**Example 1:** Input: digits = [1,2,3] Output: [1,2,4]

**Explanation:** The array represents the integer 123. Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1,2,4].

Algorithm:

1. Take the length of the array as the idx +1
2. Now use the loop and check if the last value is equal to 9 if yes then adding 1 to it will return the value zero at the end and a carry will be forwarded.
3. In Another case, if values are less than 9 then add one on the last index value.
4. Now, in case if all the values are 9 then we will have to return the list by adding one to it. i.e. [1]+digits.

CODE:

class Solution:

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

        idx = len(digits)-1

        while idx>=0 :

            if digits[idx] == 9:

                digits[idx]=0

            else:

                digits[idx] += 1

                return digits

            idx-=1

        return ([1] + digits)

            # 1 + digit in case if all are 9 then we have to add a 1 in beginning of the list.

**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.

Algorithm: 1. To find the value of the single number without using the hashmap i.e extra space we can use the Bit-Manipulation technique XOR here.

2. XOR returns 0 for both same values and it returns 1 if they are different.

3. Therefore we will initialize a variable here result =0

4. Then in a loop we will write result = n ^ result {^ sign of XOR}

CODE:

class Solution:

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

        res = 0

# n ^ 0 == n therefore we can get distinct number itself i.e 1 ^ 0 = 1, 0^0 = 0

        for n in nums:

            res = n^res

        return res

**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.

**Example 1:** Input: nums = [0,1,3,50,75], lower = 0, upper = 99 Output: [[2,2],[4,49],[51,74],[76,99]]

**Explanation:** The ranges are: [2,2] [4,49] [51,74] [76,99]

Brute Force: Since we know that the upper and lower ranges are inclusive so we will check the difference of arr[0]-lower\_range\_number; if the difference is not equal to zero, we will take [lower\_range\_number, arr[0]-1] in our output. Similarly, we can continue this procedure.

Approach:

1. Add upper range value inside the num list (nums = nums + [upper+1]), upper+1 becoz it’s inclusive.
2. Now take the previous as lower-1 (same reason)

Apply for loop on the nums and check if previous and current element has the difference or 2 or not, if it’s not then there is no missing range and if the difference is 2 it means one number is missing; Now in this case we will appent the previous+1 in our output

# num-previous (1-0)==1 therefore no difference and if num-previous ==2 i.e 3-1==2 there is difference.

1. Now if num> previous+2 then we will have to write the complete the range in it i.e [previous+1 -> num-1 ] inside the output.

nums = [0,1,3,50,75]

lower = 0

upper = 99

#Output: [[2,2],[4,49],[51,74],[76,99]]

previous = lower-1

nums = nums + [uppper+1]

output = []

for i in nums(0,len(nums)-1):

if num == (previous+2):

output.append(f"{previous+1}")

elif num > (previous+2) :

output.append(f"{previous+1}->{num-1}")

previous = num

return output

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

**Example 1:** Input: intervals = [[0,30], [5,10], [15,20]] Output: false

Solution:

Algorithm:

1. Sort the complete array
2. Now simply apply the inner loop and check if the end[i] is greater than the start[i+1] if yes then return false otherwise return true.

class interval(object):

def \_\_init\_\_(self, start, end):

self.start = start

self.end = end

class solution:

def canattendmeetings(self, interval):

interval.sort(key = lambda i : i.start)

for i in range(1, len(interval)):

i1 = interval[i-1]

i2 = interval[i]

if i1.end > i2.start:

return False

return True