**Q1. Ans.**

Algorithm: We will do a brute force approach where we will traverse each element X in a loop and compare if there is another element which is equal to [target – X].

Python3

class Answer1:

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

for i in range(len(nums)):

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

if nums[j] == target - nums[i]:

return [i, j]

Time complexity: O(n)

Space complexity:O(1)

**Q2. Ans.**

Algorithm:

Step 1: We will do a brute force approach where we will take a variable count as j, it will count the number of variables except val.

Step 2: Traverse the array from left to right.

Step 3: if the current element of the array is not equal to **val** , then we will place the element on the jth position.

class Answer2:

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

j = 0

# Loop through all the elements of the array

for i in range(len(nums)):

if nums[i] != val:

nums[j] = nums[i]

j += 1

return j

Time complexity: O(n)

Space complexity: O(1)

**Q3 Ans. :** Not getting.

**Q4 Ans.**

Brute Force Approach:

1. Need to find out in what condition we will require digits.length + 1.

Here, by checking different values, we can see that if all digits are 9, in that case only we require digits.length + 1

If any digit value, other then 9 then we don’t need to take an array with size digits.length + 1.

Here, we are moving from Right To Left, as if we find last element’s value 9, then it will be 0 as 9+1 = 10.

2. Now for any other values we dont need the array size as digits.length + 1.

def plusOne(self, digits):

"""

:type digits: List[int]

:rtype: List[int]

"""

digit\_length = len(digits)

i = digit\_length - 1

while digits[i] == 9 and i >= 0:

i -= 1

if i == -1:

results = [0]\*(digit\_length + 1)

results[0] = 1

return results

results = [0]\*(digit\_length)

results[i] = digits[i] + 1

for j in range(i-1, -1, -1):

results[j] = digits[j]

return results

**Q5 Ans.**

Algo.

Step 1: We will initialize 3 variables i,j,k where

i=m-1

j=n-1

k=m+n-1

Step 2: Now just have a traversal until i greater than or equal to 0 and j greater than or equal to 0 and do a check for the following :

if nums1[i] is greater than nums2[j], store nums1[i] to nums1[k] and decrease i and k.

-> else store nums2[j] to nums1[k] and decrease j and k.

Step 3: There can be cases where there are some left out elements in nums2. So just make sure we cover all elements with an extra while loop until j ≥ 0 and store those elements to nums[k] as those will be the largest numbers that are remaining with us to be merged.

def answer1(self, nums1: List[int], nums2: List[int], m: int,n: int):

int i=m-1

int j=n-1

int k=m+n-1

while i>=0 && j>=0:

if nums1[i] > nums2[j]:

nums1[k--] = nums2[i--]

else:

nums1[k--] = nums2[j--]

while j>=0:

nums1[k--] = nums2[j--]

Time complexity : O(M+N)

Space complexity: O(2)

**Q6 Ans.**

Algo.

Brute Force method:

We will iterate over all pairs of numbers then compare for equality

if we will get both element as equal then return true

else return false

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

for i in range(len(nums)):

for j in range(0, i):

if nums[i] == nums[j]:

return True

return False

Time complexity: O(N)

Space complexity: O(1)

**Q7 Ans.**

We will use 0 as a pivot element and whenever we see a non-zero element, we will swap it with the pivot element.

So, all non-zero element will come at the beginning.

A = [0,1,0,3,12]

n = len(A)

j = 0

for i in range(n):

if A[i] != 0:

A[j], A[i] = A[i], A[j]

j += 1

print(A)

Time complexity is O(n) as only one traversal.

Space complexity is O(1) as no extra space was taken.

**Q8 Ans**.

Sort array and use "xor" to find the miss number and the duplicate number.

For the duplicated number, arr[i] == arr[i + 1]

For the miss number, num = 1 ^ 2 ^ ... ^ n ^ arr[0] ^ arr[1] ^ ... ^ arr[len - 1] except for the duplicate number, len is the length of array

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

result = [0, 0]

numsLen = len(nums)

nums = sorted(nums)

num = 0

for i in range(numsLen - 1):

num ^= (i + 1)

if nums[i] == nums[i + 1]:

result[0] = nums[i]

else:

num ^= nums[i]

result[1] = num ^ numsLen ^ nums[numsLen - 1]

return result

Time complexity:O(NLogN)

Space complexity:O(1)