**Assignment 1 Questions - Arrays | DSA**

**Q1.** Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

**Example:** Input: nums = [2,7,11,15], target = 9 Output0 [0,1]

**------------**

**Solution:** class Solution:

**------------** 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]

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity: O(n2).

For each element, we try to find its complement by looping through the rest of the array which takes O(n) time. Therefore, the time complexity is O(n2).

* Space complexity: O(1).

The space required does not depend on the size of the input array, so only constant space is used.

**===================================x=================================**

**Q2.** Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val.

Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

* Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.
* Return k.

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

**------------**

**Solution:** class Solution:

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

k = 0

for x in nums:

if x != val:

nums[k] = x

k += 1

return k

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n).

Assume the array has a total of n elements, both i and j traverse at most 2n steps.

* Space complexity : O(1).

**===================================x=================================**

**Q3.** 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

**--------------**

**Algorithm:** - set start = 0 and end = N - 1.

**--------------** - loop while (start <= end)

- mid = (start + end)/2

- if target > nums[mid]

- start = mid + 1

- else if target < nums[mid]

- end = mid - 1

- else

- return mid

- return start

**------------**

**Solution:** class Solution:

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

left, right = 0, len(nums)

while left < right:

mid = (left + right) >> 1

if nums[mid] >= target:

right = mid

else:

left = mid + 1

return left

class Solution:

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

return bisect\_left(nums, target)

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(log n).
* Space complexity : O(1).

**===================================x=================================**

**Q4.** 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]

**------------**

**Solution:** class Solution:

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

n = len(digits)

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

digits[i] += 1

digits[i] %= 10

if digits[i] != 0:

return digits

return [1] + digits

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n), where n is the size of the array.
* Space complexity : O(1).

**===================================x=================================**

**Q5.** You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

**Example 1:** Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3 Output: [1,2,2,3,5,6]

**------------**

**Solution:** class Solution:

**------------** def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:

k = m + n - 1

i, j = m - 1, n - 1

while j >= 0:

if i >= 0 and nums1[i] > nums2[j]:

nums1[k] = nums1[i]

i -= 1

else:

nums1[k] = nums2[j]

j -= 1

k -= 1

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : Here we are traversing through the arrays of length m and n. So our time complexity will be **O(m + n).**
* Space complexity : We are not making use of any extra space. So our space complexity will be **O(1).**

**===================================x=================================**

**Q6.** Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

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

**------------**

**Solution:** class Solution:

**------------** 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

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : **O(N\*N),**Because we are traversing the whole array again and again for every integer.
* **Space Complexity:**O(1), Since, we are not using any extra space.

**===================================x=================================**

**Q7.** Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the nonzero elements.

Note that you must do this in-place without making a copy of the array.

**Example 1:** Input: nums = [0,1,0,3,12] Output: [1,3,12,0,0]

**------------**

**Solution:** class Solution:

**------------** def pushZerosToEnd(arr, n):

count = 0

for i in range(n):

         if arr[i] != 0:

             arr[count] = arr[i]

             count+=1

       while count < n:

         arr[count] = 0

         count += 1

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

n = len(arr)

pushZerosToEnd(arr, n)

print(arr)

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n). However, the total number of operations are still sub-optimal. The total operations (array writes) that code does is n (Total number of elements).
* Space Complexity : O(1). Only constant space is used.

**===================================x=================================**

**Q8.** You have a set of integers s, which originally contains all the numbers from 1 to n. Unfortunately, due to some error, one of the numbers in s got duplicated to another number in the set, which results in repetition of one number and loss of another number.

You are given an integer array nums representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return them in the form of an array.

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

**------------**

**Solution:** class Solution:

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

N, dupe = len(nums), 0

seen, sumN = [0] \* (N+1), N \* (N+1) // 2

for num in nums:

sumN -= num

if seen[num]: dupe = num

seen[num] += 1

return [dupe, sumN + dupe]

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n2). We traverse over the nusm array of size n for each of the numbers from 1 to n.
* Space complexity : O(1). Constant extra space is used.