**Assignment- 1**

**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]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1][

Solution: **Brute force**:

**Approach 1**: Traverse the array and check the sum of two nums equal to the target.

Def two\_sum (self, nums: list[int], target : int ) -> list[int]:

For i in nums():

For j in nums(i+1,len(nums)):

If num[i]+num[j] == target

Return [i,j]

TC: O(n^2) {Since we are using two loops here}

**Approach 2**: **Hashmap**

**Algorithm:**

1. **Maintain a HashMap/Dictionary with value and index pair.**
2. **Start from index 0 , Subtract the target with value at 0 index, and update in the hashmap.**
3. **Continue till the end and keep checking if the difference (Target - value) is present in the hashmap, if yes then return the indices otherwise update the value in the hashmap and repeat.**

**TC: O(n); SC: O(n)**

**Code:**

**class Solution:**

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

**prevMap = {} //Value:Index**

**for i, n in enumerate((nums)):**

**diff = target – n**

**if diff in prevMap:**

**return [prevMap[diff], i]**

**prevMap[n] = i**

**return**

**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,*\*,*\*]

**Explanation:** Your function should return k = 2, with the first two elements of nums being 2. It does not matter what you leave beyond the returned k (hence they are underscores)

Solution:

Brute force:

Algorithm1:

1. Assign a pointer ‘k’ pointing towards the 1st element of the array.
2. If num[i] != val, then swap the value of I and k, and increment the value of k

Code:

Def removeElement(self, num: list, val:int):

K=0

For I in range(len(num)):

If num[i] != val:

num[k] = num[i]

k = k+1

return k

Algorithm2:

1. Apply while loop and if the num[i] == val then pop the num from array else increment the value of I and return the len(num)

Code:

While (i<len(num)):

If num[i] == val :

num.pop(i)

else:

I += 1

return len(num)

Algorithm 3:

Apply while loop and then if num[i]==val then swap the value at I with the last index value and pop the last element. TC: O(n), SC: O(1)

While (i<len(num)):

If num[i] == val:

num[i], num[-1] = num[-1], num[i]

num.pop()

else:

i+=1

return len(num)

**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

Output: 2

Solution: Since it’s a sorted array and we want the algorithm with O(logn) runtime Complexity then Binary search algorithm is the best fit.

def binarySearch(arr, I,j,target):

While i<=j:

(i) + (j-i) //2 = mid

If mid == target :

return mid

elif mid<target:

return binarySearch(arr,mid+1,j, target)

elif mid>target:

return binarySearch(arr, i, mid-1,target )

else:

return -1

#Drivercode

arr = [1,3,5,6]

i = arr[0]

j = len(arr)-1

target = 5

**Q4.** You are given a large integer represented as an integer array of digits, where each digit [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 res ult should be [1,2,4].

Solution:

Algorithm:

1. Take the digits array last index and add one to it and return the digits array if the number is less than 9.

idx = len(digits) -1

While idx>=0:

if digits[i]<9:

digits[idx-1] + 1

return digits

1. If the last digit is equal to 9 then write 0 on that place and add one to last second digit.

If digit[idx] == 9:

digit[idx]=0

digit[idx-1] = digit[idx-1] + 1

**Code: TC: O(n), SC: O(1)**

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

**Idx = len(digits) – 1**

**While idx>=0:**

**If digits[idx] == 9:**

**Digit[idx] = 0**

**else:**

**digit[idx] += 1**

**return digits**

**idx -=1**

**return [1] + digits**

**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]

**Explanation:** The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Solution:

Approach1 : 2-pointer method with a num1copy array

Algorithm:

1. Initialize new array(num1 copy) containing the first m elements of the num1 array.
2. Initialize the p1 pointer to beginning of the num1copy.
3. Initialize p2 to beginning of the num2 array.
4. If num1copy[p1] exists and is less than or equal to num2[p2] write num1copy[p1] in num1 and increment p1

Else,

Write num2[p2] in num1 and increment p2.

Code: TC: O(m+n) [ since traversing and comparing both the arrays of size m and n]

SC: O(m)

Approach2: Without taking the num1copy or extra space

Algorithm:

1. We can point a pointer on the last index of the num1 array which is already empty.
2. Initialize two pointers one for num1 and the other for num2 at their last indices.
3. Now start comparing the num1 pointer and num2 pointer and whichever is greater place it in the last index value of num1 and decrement the last pointer of num1.
4. If num1 pointer value> num2 pointer value:

Put the num1 pointer value at the last index pointer now

Decrement the num1pointer value with the last pointer value.

CODE: TC: O(m+n) SC : O(1)

Def mergesorted(self, num1 : list, m : int, num2: list, n: int) -> none :

last = m+n-1

While m>0 and n>0: ///Merge in reverse order

If num1(m-1) > num2( n-1):

Num1(last) = num1(m-1)

m -=1

else:

num1(last) = num2(n-1)

n -=1

last -=1

# fill num1 with leftover elements of num2

While n>0:

num1(last) = num2(n-1)

n ,last = n-1, last-1

**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:

Brute force: Traverse the complete array and compare each element with all the elements present in the array. TC: O(n^2){Since for all n elements present in the array we will have to compare with all the elements present in the array}

Approach 2: Sorting the array

We can sort the array and then check the duplicate values by comparing the near by element. This will traverse the array only a single time and without any extra space. But we are doing sorting as well therefore the TC will be: O(nlogn)

Approach 3: Hash Set {It allows us to inser elements in hashset with O(1) complexity Plus it also allows us to check whether the value already exists or not}

**CODE: TC- O(n) ; SC – O(n)**

def checkduplicate(self, nums : list[int]) -> bool:

hashset = set()

for n in nums:

if n in hashset:

return True

else:

hashset.add(n)

return false

**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:

Approach: pointer method.

Algorithm:

1. Initialize a pointer with index = 0, j=0
2. Start traversing the array and check for the element, if element != 0 then swap the element with the pointer. Ex: if nums[i] != 0:
3. Swap nums[i] and nums[j] and increment pointer j otherwise continue

CODE: TC: O(n) SC: O(1)

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

n = len(nums)

j = 0

for i in range(len(nums)):

if nums[i] != 0:

nums[i], nums[j] = nums[j], nums[i] //partitioning the array

j +=1

print(nums)

**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: Hashmap can be used.

Algorithm:

1. Initialize the hashmap. {hashmap = Counter(nums)}
2. Traverse the array and check for the duplicate value with the frequency in the hashmap. If frequency is greater than 1 then it’s a duplicate number.
3. And we will traverse the hashmap again and check the values from 0 to n, the missing one will be the missing number.

CODE:

# Using hashmap

from collections import Counter

nums = [1,2,2,4]

def duplicateremoval(nums : list[int]) -> list[int]:

hashmap = Counter(nums)

missing = None

duplicate = None

for i in range(len(nums)):

if i not in hashmap: # Values in hashmap

missing = i

if hashmap[i]>1: # frequency in hashmap = hashmap[i]

duplicate = i

return [duplicate, missing]

print(duplicateremoval(nums))