DSA – ASSIGNMENT -1

ARRAYS

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:

Approach 1: Brute Force:

1) Use two loops to traverse through each element to find if the sum equals to target.
2) TC: O(n^2) – for each element we try to traverse through the rest of the elements.
3) SC: O(1) – It does not take any additional space, constant space is used.

Approach 2: Hash-table:

1) We traverse through elements to insert into hash table.
2) If the other element that equals to target is found, return them immediately.
3) **TC: O(n)** – traversing through the elements, only once (n). Lookup in table costs O(1).
4) **SC: O(n)** – it depends on the number of elements stored, it stores at most n elements.

Code:

```
class Solution:

    def twoSum(self, nums: List[int], target: int) -> List[int]:
        index_map={}
        for i, n in enumerate(nums):
            diff = target-n
            if diff in index_map:
                return [index_map[diff], i]
            index_map[n]=i
```

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:

1) If nums[i] not equal to value, keep incrementing the value of k.
2) Finally, return k.
3) **TC: O(n)** – as we traverse through the elements only once.
4) **SC: O(1)** – as no extra space is used.

Code:

```
class Solution:

    def removeElement(self, nums: List[int], val: int) -> int:
                k = 0
                for i in range(len(nums)):
                    if nums[i] != val:
                        nums[k] = nums[i]
                        k += 1
                return k
```

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: Binary search method

1) Use two pointers, left -l & right -r. Add them both to find the mid value.
2) If the mid value = target, return it.
3) Else increment on the right of mid value if the target is greater than the mid value or decrement on the left if its less than the target value.
4) **TC: O(logn), SC: O(1)**

Code:

```
class Solution:

   def searchInsert(self, nums: List[int], target: int) -> int:
        l,r = 0, len(nums)-1
        while l <= r :
            mid = (l + r)//2
           if target == nums[mid]:
                  return mid
            if target > nums[mid]:
                  l = mid + 1
             else:
                  r =  mid - 1
         return l
```

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]

Explanation: The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1,2,4].

Solution:

1) Reverse the given array.
2) Use a loop to traverse through the elements, when it encounters 9 set it to 0 where there is a carry of 1.
3) If it encounters other elements, set carry to 0 to exit the loop.
4) When array length is out of bound append the carry value of 1 to the array.
5) Finally, reverse the array.
6) **TC: O(n)** – as we travers only once through the array.
7) **SC: O(1)** – as no extra space is used.

Code:

```
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:
        digits = digits[::-1]
        carry, i = 1, 0

        while carry:
            if  i < len(digits):
                if digits[i]== 9:
                    digits[i]=0
                else:
                    digits[i] += 1
                    carry = 0
            else:
                digits.append(1)
                carry = 0
            i += 1
        return digits[::-1]
```

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:

1) Find max of m and n and populate from last of nums1 array.
2) **TC: O(n), SC: O(1)**

Code:

```
class Solution:
   def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
            """
            Do not return anything, modify nums1 in-place instead.
            """
            last = m+n-1
            while m > 0 and n > 0:
                if nums1[m-1] > nums2[n-1]:
                    nums1[last] = nums1[m-1]
                    m -= 1
```

```
            else:
                nums1[last] = nums2[n-1]
                n -= 1
            last -= 1

        while  n > 0:
            nums1[last] = nums2[n-1]
            last, n = last-1, n-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:

1) Traverse through the array of elements and add them to a set.
2) If they already exist in the set return True.
3) Else return False if all the elements are unique.
4) **TC: O(n). SC: O(1)**

Code:

```
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        dup = set()
        for i in nums:
            if i in dup:
                return True
            dup.add(i)
        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: Quick Sort

1) Use two pointers, left and right.
2) If the element is non-zero add it to the left and if its zero add it to the right.
3) Keep incrementing the left pointers, and traverse through array of elements.
4) **TC: O(n)** – traverse only once.
5) **SC: O(1)** – as no extra space is used.

Code:

```
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        l = 0
        for r in range(len(nums)):
            if nums[r] !=0: # if nums[r]:
                nums[l],nums[r]=nums[r],nums[l]
                l += 1
        return 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:

1) The length of the array is assigned to n.
2) Find the difference between the sum of elements and the sum of unique elements. This gives the repeated elements value assign it to x.
3) Then, find the difference between the sum of elements and the squarte of them, again subtract it with x. This gives the missing number in the array.
4) **TC: O(n), SC:O(1)**

Code:

```
class Solution:
    def repeatedNumber(self, A):
        n = len(A)
        x = sum(A) - sum(set(A))
        k = sum(A) - int(n*(n+1)/2)

        return [x,x-k]

A = [1,2,2,4]
ans = Solution()
res = ans.repeatedNumber(A)
print(res)
```