**Assignment 3 Questions - Arrays | DSA**

**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

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

**Solution :** class Solution(object):

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

nums.sort()

n = len(nums)

closest = nums[0] + nums[1] + nums[n - 1]

for i in range(0, n - 2):

j = i + 1

k = n - 1

while j < k:

current\_sum = nums[i] + nums[j] + nums[k]

if current\_sum <= target:

j += 1

else:

k -= 1

if abs(closest - target) > abs(current\_sum - target):

closest = current\_sum

return closest

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

**Complexity Analysis:**

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

* Time complexity : O(n2). We are scanning the entire array keeping one element fixed. We are doing this for every element in the array. Thus, we are scanning each element of array n number of times. And we are doing this for n times, hence the worst case time complexity will be O(n2 + n \* log n) which comes down to O(n2).
* Space Complexity : O(1). We are not using any data structure for the intermediate computations, hence the space complexity is O(1).

====================================x===================================

**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]]

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

**Solution :** class Solution(object):

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

quadruplets = list()

if nums is None or len(nums) < 4:

return quadruplets

nums.sort()

n = len(nums)

for i in range(0, n - 3):

if i > 0 and nums[i] == nums[i - 1]:

continue

for j in range(i + 1, n - 2):

if j != i + 1 and nums[j] == nums[j - 1]:

continue

k = j + 1

l = n - 1

while k < l:

current\_sum = nums[i] + nums[j] + nums[k] + nums[l]

if current\_sum < target:

k += 1

elif current\_sum > target:

l -= 1

else:

quadruplets.append([nums[i], nums[j], nums[k], nums[l]])

k += 1

l -= 1

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

k += 1

while k < l and nums[l] == nums[l + 1]:

l -= 1

return quadruplets

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

**Complexity Analysis:**

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

* Time Complexity : O(n3). We are scanning the entire array keeping one element fixed and then doing it for another element fixed. We are doing this for every element in the array. Thus, we are scanning each element of array n number of times. And we are doing this for n times, hence the worst case time complexity will be O(n3 + n \* log n) which comes down to O(n3).
* Space Complexity : O(1). We are not using any data structure for the intermediate computations, hence the space complexity is *O(1)*.

===================================x=====================================

**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 as 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]

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

**Solution :** class Solution(object):

**-------------** def reverse(nums, i, j):

while i < j:

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

i += 1

j -= 1

def nextPermutation(nums: List[int]):

n = len(nums)

index = -1

for i in range(n - 1, 0, -1):

if nums[i] > nums[i - 1]:

index = i - 1

break

if index == -1:

reverse(nums, 0, n - 1)return

j = n - 1

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

if nums[i] > nums[index]:

j = i

break

nums[index], nums[j] = nums[j], nums[index]

reverse(nums, index + 1, n - 1)

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

**Complexity Analysis:**

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

### Time Complexity : O(n). We are iterating the array two times. In the worst case, the time complexity will be O(2n) which is equivalent to O(n).

### Space Complexity : O(1). We are not using any data structure for intermediate computations. Hence, the space complexity will be O(1).

===================================x=====================================

### 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 :**

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

- 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(object):

**-------------** def find\_index(arr, n, K):

start = 0

end = n - 1

while start<= end:

mid =(start + end)//2

if arr[mid] == K:

return mid

elif arr[mid] < K:

start = mid + 1

else:

end = mid-1

return end + 1

arr = [1, 3, 5, 6]

n = len(arr)

K = 2

print(find\_index(arr, n, K))

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

**Complexity Analysis:**

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

* Time Complexity : O(log N)
* Space Complexity : O(1)

===================================x=====================================

**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]

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

**Solution :** class Solution(object):

**-------------** 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 number of digits.
* Space Complexity : O(1)

===================================x=====================================

**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.

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

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

**Solution :** class Solution(object):

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

singleNum = 0

for i in range(len(nums)):

singleNum ^= nums[i]

return singleNum

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

**Complexity Analysis:**

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

* Time Complexity : O(n)
* Space Complexity : O(1)

===================================x=====================================

**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]]

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

**Solution :** class Solution(object):

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

ranges = []

i = 0

while i < len(nums):

start = nums[i]

while i + 1 < len(nums) and nums[i] + 1 == nums[i + 1]:

i += 1

if start != nums[i]:

ranges.append(str(start) + "->" + str(nums[i]))

else:

ranges.append(str(nums[i]))

i += 1

return ranges

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

**Complexity Analysis:**

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

Here n is the number of elements in nums.

* Time Complexity : O(n). We iterate over each nums element once, either including it in the current range or creating a new range from it, which takes O(n) time for n elements.
* Space Complexity : O(1). Except for a few integer variables like i and start that use constant space, we do not consume any space (if we ignore the space consumed by the input and output).

===================================x=====================================

**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 :** class Solution():

**-------------** def canAttend(intervals):

if intervals is None or len(intervals) == 0:

return True

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

if intervals[i][0] < intervals[i - 1][1]:

return False

return True

t = [[0, 3], [5, 10], [15, 20]]

print(canAttend(t))

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

**Complexity Analysis:**

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

* Time complexity : O(nlog n) since sort the list will take O(nlogn) and iteration will take O(n) where n is the size of input.
* Space Complexity : *O*(*N*) because we construct the min-heap and that can contain N elements in the worst case as described above in the time complexity section. Hence, the space complexity is *O*(*N*).

===================================x=====================================