ASSIGNMENT -3

ARRAYS

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

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

SOLUTION:

Algorithm:

1) Sort the array.
2) Now for each element i, do the following steps
3) Set two pointers left — j = i + 1 and right — k = nums.length - 1.
4) Check if nums[i] + nums[j] + nums[k] <= target, it means we are too left in the array, and we need to move right i.e., we can check for greater number than the current one.
5) If the sum nums[i] + nums[j] + nums[k] > target, it means we are too right in the array, and we need to move left i.e., we can check for smaller number than the current one.
6) Compare the minimum difference between the current sum and the previous sum. The sum which gives minimum difference is the answer.
7) TC: **O(n2),** SC: **O(1)**

Code:

```
class Solution:
    def threeSumClosest(self, nums: List[int], target: int) -> int:
         # Sort the given array
            nums.sort()
            # Length of the array
            n = len(nums)
            # Closest value
            closest = nums[0] + nums[1] + nums[n - 1]
            # Loop for each element of the array
            for i in range(0, n - 2):
                # Left and right pointers
                j = i + 1
                k = n - 1
                # Loop for all other pairs
                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
```

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:

Algorithm:

1. Sort the array in time.
2. Now for each element i and j, do the following steps —
3. Set two pointers left — k = j + 1 and right — l = n - 1.
4. Check if nums[i] + nums[j] + nums[k] + nums[l] == target and add it to the result, if true/
5. If nums[i] + nums[j] + nums[k] + nums[l] < target, then we are too left, and we need to move right so increment left pointer i.e. k++.
6. If nums[i] + nums[j] + nums[k] + nums[l] > target, then we are too right, and we need to decrement the right pointer i.e., l--.
7. TC: **O(n3)**, SC: **O(1)**

Code:

```
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        nums.sort()
        results = []
        self.helper(nums, target, 4, [], results)
        return results

    def helper(self, nums, target, N, res, results):

        if len(nums) < N or N < 2: #1
            return
        if N == 2: #2
```

```
                output_2sum = self.twoSum(nums, target)
                if output_2sum != []:
                    for idx in output_2sum:
                        results.append(res + idx)

            else:
                for i in range(len(nums) -N +1): #3
                    if nums[i]*N > target or nums[-1]*N < target: #4
                        break
                    if i == 0 or i > 0 and nums[i-1] != nums[i]: #5
                        self.helper(nums[i+1:], target-nums[i], N-1, res +
[nums[i]], results)


        def twoSum(self, nums: List[int], target: int) -> List[int]:
            res = []
            left = 0
            right = len(nums) - 1
            while left < right:
                temp_sum = nums[left] + nums[right]

                if temp_sum == target:
                    res.append([nums[left], nums[right]])
                    right -= 1
                    left += 1
                    while left < right and nums[left] == nums[left - 1]:
                        left += 1
                    while right > left and nums[right] == nums[right + 1]:
                        right -= 1

                elif temp_sum < target:
                    left +=1
                else:
                    right -= 1

            return res
```

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:

Algorithm:

1) Assign the last element of nums to variable end.
2) Use a loop where if start is less than the length of the elements, swap elements.
3) Now assign the 2nd last element to a variable i, and while its greater than) and i+1 is less than or equal to i, decrement i.
4) Again swap the elements and return the list.

Code:

```
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        def reverse(nums, start):
            end = len(nums) - 1
            while start < end:
                nums[start], nums[end] = nums[end], nums[start]
                start += 1
                end -= 1
        i = len(nums) - 2
        while i >= 0 and nums[i + 1] <= nums[i]:
            i -= 1
        if i >= 0:
            j = len(nums) - 1
            while nums[j] <= nums[i]:
                j -= 1
            nums[i], nums[j] = nums[j], nums[i]
        reverse(nums, i + 1)
```

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

SOLUTION:

Algorithm: 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
```

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]

Explanation: The array represents the integer 123.

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

SOLUTION:

Algorithm:

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]
```

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:

Algorithm:

1) The code utilizes a dictionary to solve the problem.
2) It iterates through the elements in the nums list and updates the dictionary accordingly.
3) If an element is not present as a key in the dictionary, it is added with a value of True.
4) If an element already exists as a key, its value is updated to False.
5) Finally, it iterates over the dictionary and returns the key with a value of True, which represents the single number.
6) TC: **O(n)**, SC: **O(n)**

Code:

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        dict = {}  # Create an empty dictionary

        # Iterate through each element in nums
        for i in nums:
            if i not in dict:  # If the element is not in the dictionary
                dict[i] = True  # Add it to the dictionary with a value of
True
            else:
                dict[i] = False  # If the element already exists in the
dictionary, set its value to False

        # Iterate through the key-value pairs in the dictionary
        for key, val in dict.items():
            if val == True:  # If the value is True (indicating a single
occurrence)
                return key  # Return the corresponding key as the single
number
```

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]]

Explanation: The ranges are: [2,2] [4,49] [51,74] [76,99]

SOLUTION:

Algorithm:

1) Declare two objects lo and hi in the function getRange.
2) If lo and hi are equal return lo else return hi.
3) If the value is not in nums return its lower and upper range.
4) Create a empty list ans.
5) Now, if the first value of nums is greater than the lower value, return the current and the previous element.
6) If curr is greater than the next element, append the curr-1 and the xurr next element to the list ans.

7) Else if the last element of nums less than the upper value, append the last and the upper value to the list.
8) TC: O(n), SC:O(n) – a new list.

Code:

```
class Solution:
  def findMissingRanges(self, nums: list[int], lower: int, upper: int) ->
list[str]:
    def getRange(lo: int, hi: int) -> str:
      if lo == hi:
        return str(lo)
      return str(lo) + '->' + str(hi)

    if not nums:
      return [getRange(lower, upper)]

    ans = []

    if nums[0] > lower:
      ans.append(getRange(lower, nums[0] - 1))

    for prev, curr in zip(nums, nums[1:]):
      if curr > prev + 1:
        ans.append(getRange(prev + 1, curr - 1))

    if nums[-1] < upper:
      ans.append(getRange(nums[-1] + 1, upper))

    return ans
nums = [0,1,3,50,75]
lower = 0
upper = 99
ans = Solution()
res = ans.findMissingRanges(nums,lower,upper)
print(res)
```

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:

Algorithm:

1) Sort the given list of lists.
2) Use a loop to iterate for the length of the list, and return False if the element in the list of list of the previous index [0][1] is greater than the current [1][0] element.

3) Else return True.
4) TC: **O(n)**, SC: **O(1)**

Code:

```
class Solutions:
  def canAttendMeetings(self, intervals: list[list[int]]) -> bool:
          new_intervals = sorted(intervals, key=lambda x: x[0])
          for i in range(1,len(new_intervals)):
              if new_intervals[i-1][1] > new_intervals[i][0]:return False
          return True

intervals = [[0,30],[5,10],[15,20]]
ans = Solutions()
res = ans.canAttendMeetings(intervals)
print(res)
```