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).

Soln:

```
def find_close_triplet(arr, n, x, count, sum, ind, ans, minm):
        if ind == n:
        if count == 3:
            if abs(x - sum) < minm[0]:
                minm[0] = abs(x - sum)
                ans[0] = sum
        return

    # Pick this number
    find_close_triplet(arr, n, x, count + 1, sum + arr[ind], ind + 1, ans, minm)

    # Don't pick this number
    find_close_triplet(arr, n, x, count, sum, ind + 1, ans, minm)

# Driver's code
if __name__ == "__main__":
  #Input array
  arr = [-1, 2, 1, -4]
  x = 1
  n = len(arr)
  minm = [float('inf')]
  ans = [0]

  # Function Call
  find_close_triplet(arr, n, x, 0, 0, 0, ans, minm)
  print(ans[0])
```

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]]

Soln:

```
def nextPermutation(nums):
    # Find the first pair of adjacent numbers in descending order
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1

    # If such pair exists, find the smallest number greater than the one at index i
    if i >= 0:
        j = len(nums) - 1
        while nums[j] <= nums[i]:
            j -= 1
        nums[i], nums[j] = nums[j], nums[i]

    # Reverse the subarray from i+1 to the end
    start = i + 1
    end = len(nums) - 1
    while start < end:
        nums[start], nums[end] = nums[end], nums[start]
        start += 1
        end -= 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

Soln:

```
def searchInsert(nums, target):
    left = 0
    right = len(nums) - 1

    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return left
```

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].

Soln:

```
def plusOne(digits):

    n = len(digits)

    carry = 1


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

        digits[i] += carry

        carry = digits[i] // 10

        digits[i] %= 10


    if carry:

        digits.insert(0, carry)


    return digits
```

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

Soln:

```
def singleNumber(nums):
    result = 0
    for num in nums:
        result ^= num
    return result
```

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]

Soln:

```
def findMissingRanges(nums, lower, upper):
    result = []
    start = lower

    for num in nums:
        if num > start:
            result.append(getRange(start, num - 1))
        start = num + 1

    if start <= upper:
        result.append(getRange(start, upper))

    return result
```

```
def getRange(start, end):
    if start == end:
        return str(start)
    else:
        return str(start) + "->" + str(end)
```

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

Soln:

```
def canAttendMeetings(intervals):
    # Sort the intervals by the start time
    intervals.sort(key=lambda x: x[0])

    # Check for overlapping meetings
    for i in range(1, len(intervals)):
        if intervals[i][0] < intervals[i-1][1]:
            return False

    return True
```