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.

```
def threeSumClosest(nums, target):
    nums.sort()
    closest_sum = float('inf')
    n = len(nums)
    for i in range(n - 2):
        left = i + 1
        right = n - 1
        while left < right:
            current_sum = nums[i] + nums[left] + nums[right]
            if abs(current_sum - target) < abs(closest_sum - target):
                closest_sum = current_sum
            if current_sum < target:
                left += 1
            elif current_sum > target:
                right -= 1
            else:
                return target
    return closest_sum
```

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.

```
def fourSum(nums, target):
    nums.sort()
    result = []
    n = len(nums)
    for i in range(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
            left = j + 1
            right = n - 1
            while left < right:
                current_sum = nums[i] + nums[j] + nums[left] + nums[right]
                if current_sum == target:
                    result.append([nums[i], nums[j], nums[left], nums[right]])
                    while left < right and nums[left] == nums[left + 1]:
                        left += 1
                    while left < right and nums[right] == nums[right - 1]:
                        right -= 1
                    left += 1
                    right -= 1
                elif current_sum < target:
                    left += 1
                else:
                    right -= 1
    return result
```

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).

```
def nextPermutation(nums):
    n = len(nums)
    i = n - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1


    if i >= 0:
        j = n - 1
        while j > i and nums[j] <= nums[i]:
            j -= 1
        nums[i], nums[j] = nums[j], nums[i]
    left = i + 1

    right = n - 1

    while left < right:

        nums[left], nums[right] = nums[right], nums[left]

        left += 1

        right -= 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.

```
def searchInsert(nums, target):

    left = 0

    right = len(nums) - 1

    while left <= right:

        mid = left + (right - left) // 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.

```
def plusOne(digits):

    carry = 1

    for i in range(len(digits) - 1, -1, -1):

        digit_sum = digits[i] + carry

        digits[i] = digit_sum % 10

        carry = digit_sum // 10

        if carry == 0:

            break

    if carry == 1:

        digits.insert(0, 1)

    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.

```
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.

```
def findMissingRanges(nums, lower, upper):

    result = []

    start = lower

    for num in nums:

        if num == start:

            start += 1

        elif num > start:

            if start == num - 1:

                result.append(str(start))

            else:

                result.append(str(start) + "->" + str(num - 1))

            start = num + 1



    if start <= upper:

        if start == upper:

            result.append(str(start))

        else:

            result.append(str(start) + "->" + str(upper))

    return result
```

Question 8

Given an array of meeting time intervals where intervals[i] = [starti, endi], determine if a person could attend all meetings.

```
def canAttendMeetings(intervals):

    intervals.sort(key=lambda x: x[0])  # Sort intervals based on start time

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

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

            return False

    return True
```