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).

CODE

```
import java.util.*;


public class thrSum {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);


      System.out.print("Enter the number of elements in the array: ");
      int n = scanner.nextInt();


      int[] nums = new int[n];
      System.out.println("Enter the elements of the array:");
      for (int i = 0; i < n; i++) {
         nums[i] = scanner.nextInt();
      }


      System.out.print("Enter the target sum: ");
      int target = scanner.nextInt();


      int ClSum = thrSum(nums, target);
      System.out.println("The sum closest to the target is: " + ClSum);
   }


   public static int thrSum(int[] nums, int target) {
      Arrays.sort(nums);
      int ClSum = nums[0] + nums[1] + nums[2];
      for (int i = 0; i < nums.length - 2; i++) {
```

```
        int left = i + 1;
        int right = nums.length - 1;

        while (left < right) {
          int sum = nums[i] + nums[left] + nums[right];

          if (sum == target) {
            return sum;
          }

          if (Math.abs(sum - target) < Math.abs(ClSum - target)) {
            ClSum = sum;
          }

          if (sum < target) {
            left++;
          } else {
            right--;
          }
        }
      }

      return ClSum;
    }
}
```

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]]

CODE

```
import java.util.*;

public class FourSum {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        System.out.print("Enter the target sum: ");
        int target = scanner.nextInt();

        List<List<Integer>> qdrupList = fourSum(nums, target);
        System.out.println("Unique qdrupList with the target sum:");
        for (List<Integer> quadruplet : qdrupList) {
            System.out.println(quadruplet);
        }
    }

    public static List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> qdrupList = new ArrayList<>();
        if (nums == null || nums.length < 4) {
            return qdrupList;
        }
```

```
Arrays.sort(nums);
int n = nums.length;
for (int i = 0; i < n - 3; i++) {
   if (i > 0 && nums[i] == nums[i - 1]) {
      continue;
   }

   for (int j = i + 1; j < n - 2; j++) {
      if (j > i + 1 && nums[j] == nums[j - 1]) {
         continue;
      }

      int left = j + 1;
      int right = n - 1;

      while (left < right) {
         int sum = nums[i] + nums[j] + nums[left] + nums[right];

         if (sum == target) {
            qdrupList.add(Arrays.asList(nums[i], nums[j], nums[left], nums[right]));

            while (left < right && nums[left] == nums[left + 1]) {
               left++;
            }
            while (left < right && nums[right] == nums[right - 1]) {
               right--;
            }

            left++;
            right--;
         } else if (sum < target) {
```

```
                left++;
            } else {
                right--;
            }
        }
    }
}

    return qdrupList;
  }
}
```

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]

CODE

```
import java.util.*;


public class NextPermutation {
    public static void main(String[] args) {
```

```
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the number of elements in the array: ");
    int n = scanner.nextInt();

    int[] nums = new int[n];
    System.out.println("Enter the elements of the array:");
    for (int i = 0; i < n; i++) {
        nums[i] = scanner.nextInt();
    }

    nextPermutation(nums);

    System.out.println("The next permutation is:");
    for (int num : nums) {
        System.out.print(num + " ");
    }
}

public static void nextPermutation(int[] nums) {
    int n = nums.length;
    int i = n - 2;


    while (i >= 0 && nums[i] >= nums[i + 1]) {
        i--;
    }

    if (i >= 0) {
        int j = n - 1;
```

```
            while (j >= 0 && nums[j] <= nums[i]) {
                j--;
            }


            swap(nums, i, j);
        }

         reverse(nums, i + 1);
    }

    private static void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

    private static void reverse(int[] nums, int start) {
        int i = start;
        int j = nums.length - 1;

        while (i < j) {
            swap(nums, i, j);
            i++;
            j--;
        }
    }
}
```

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

CODE : import java.util.*;

```
public class SearchInsertPosition {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of elements in the sorted array: ");
      int n = scanner.nextInt();

      int[] nums = new int[n];
      System.out.println("Enter the elements of the sorted array:");
      for (int i = 0; i < n; i++) {
         nums[i] = scanner.nextInt();
      }

      System.out.print("Enter the target value: ");
      int target = scanner.nextInt();

      int insertPosition = searchInsert(nums, target);
      System.out.println("The index to insert the target value is: " + insertPosition);
   }

   public static int searchInsert(int[] nums, int target) {
      int left = 0;
      int right = nums.length - 1;

      while (left <= right) {
         int mid = left + (right - left) / 2;
```

```
        if (nums[mid] == target) {
            return mid;
        } else if (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].

CODE :

```
import java.util.*;

public class PlusOne {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of digits in the large integer: ");
      int n = scanner.nextInt();

      int[] digits = new int[n];
      System.out.println("Enter the digits of the large integer (from left to right):");
```

```
        for (int i = 0; i < n; i++) {
            digits[i] = scanner.nextInt();
        }

        int[] result = plusOne(digits);
        System.out.println("The incremented large integer is:");
        for (int digit : result) {
            System.out.print(digit + " ");
        }
    }

    public static int[] plusOne(int[] digits) {
        int n = digits.length;

        for (int i = n - 1; i >= 0; i--) {
            if (digits[i] < 9) {
                digits[i]++;
                return digits;
            }

            digits[i] = 0;
        }


        int[] result = new int[n + 1];
        result[0] = 1;
        return result;
    }
}
```

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

CODE : import java.util.*;

```
public class SingleNumber {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        int singleNumber = findSingleNumber(nums);
        System.out.println("The single number is: " + singleNumber);
    }

    public static int findSingleNumber(int[] nums) {
        int result = 0;

        for (int num : 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]

CODE:

```
import java.util.ArrayList;
import java.util.List;

public class MissingRanges {
    public static List<List<Integer>> findMissingRanges(int[] nums, int lower, int upper) {
        List<List<Integer>> result = new ArrayList<>();

        if (nums[0] > lower) {
            result.add(createRange(lower, nums[0] - 1));
        }

        for (int i = 0; i < nums.length - 1; i++) {
            if (nums[i + 1] - nums[i] > 1) {
                result.add(createRange(nums[i] + 1, nums[i + 1] - 1));
            }
        }

        if (nums[nums.length - 1] < upper) {
            result.add(createRange(nums[nums.length - 1] + 1, upper));
        }
```

```
        return result;
    }

    private static List<Integer> createRange(int start, int end) {
        List<Integer> range = new ArrayList<>();
        range.add(start);
        range.add(end);
        return range;
    }

    public static void main(String[] args) {
        int[] nums = {0, 1, 3, 50, 75};
        int lower = 0;
        int upper = 99;

        List<List<Integer>> missingRanges = findMissingRanges(nums, lower, upper);

        System.out.println("The shortest sorted list of ranges is:");
        for (List<Integer> range : missingRanges) {
            System.out.println(range);
        }
    }
}
```