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
PRGM:

```
class SumClosest{
   public static int threeSumClosest(int[] nums, int target) {
     int result=nums[0]+nums[1]+nums[nums.length-1];
      Arrays.sort(nums);
      for (int i=0;i<nums.length-2;i++) {
         int start=i+1,end=nums.length-1;
         while(start<end) {
            int sum=nums[i]+nums[start]+nums[end];
            if(sum>target) end--;
            else start++;
            if (Math.abs(sum-target)<Math.abs(result-target))
            result=sum;
         }
      }
      return result;
   }
   public static void main(String[] args) {
      int nums[] = {-1,2,1,-4};
      int x = 1;
      System.out.println(threeSumClosest(nums,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]]
PRGM:

```
class FourSum {
   private static List<List<Integer>> fourSum(int[] nums, int target) {
      List<List<Integer>> quadruplets = new ArrayList<>();

      if (nums == null || nums.length < 4) {
         return quadruplets;
```

```
        }

        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 k = j + 1;
                int l = n - 1;

                while (k < l) {
                    int currentSum = nums[i] + nums[j] + nums[k] + nums[l];
                    if (currentSum < target) {
                        k++;
                    } else if (currentSum > target) {
                        l--;
                    } else {
                        quadruplets.add(Arrays.asList(nums[i], nums[j], nums[k],num

[l]));

                        k++;
                        l--;

                        while (k < l && nums[k] == nums[k - 1]) {
                            k++;
                        }
                        while (k < l && nums[l] == nums[l + 1]) {
                            l--;
                        }
                    }
                }
            }
        }
        return quadruplets;
    }
    public static void main(String[] args) {
        System.out.println(fourSum(new int[]{1, 0, -1, 0, -2, 2}, 0));
    }
}
```

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].
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]
Prgm:

```
class NextPermutation {
    public static void nextPermutation(int[] nums) {
        if(nums==null||nums.length<=1)
        return;

        int i = nums.length-2;
        while(i>=0 && nums[i]>=nums[i+1])
        i--;

        if(i>=0)
        {
            int j = nums.length-1;
            while(nums[j]<=nums[i])
            j--;
            swap(nums,i,j);
        }
        reverse(nums,i+1,nums.length-1);
    }

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

    static void reverse(int[] nums, int i,int j)
    {
        while(i<j)
        swap(nums,i++,j--);
    }

    public static void main(String[] args) {
        int nums[] = {1,2,3};
        nextPermutation(nums);
        System.out.println(Arrays.toString(nums));
    }
}
```

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
Prgm:

```
class search_insert
{
   static int searchInsert(int[] a , int target)
   {
      int l = 0 , r = a.length - 1 , mid , ans = -1;
      while(l <= r)
      {
         mid = l + (r - l) / 2;
         if(a[mid] == target)
            return mid;
         if(a[mid] < target)
         {
            l = mid + 1;
            ans = mid + 1;
         }
         else
         {
            ans = mid;
            r = mid - 1;
         }
      }
      return ans;
   }
   static int search_Insert(int[] a , int target)
   {
      return search_Insert(a , target);
   }
   public static void main(String args[])
   {
      int a[] = {1 , 3 , 5 , 6};
      int target = 5;
      System.out.println(searchInsert(a , target));
   }
}
```

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]
Prgm:

```
class PlusOne {
```

```
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;
    }

    digits = new int [n+1];
    digits[0] = 1;
    return digits;
}
  public static void main(String[] args) {
        int [] arr = {1,2,3};
        int[]ans=plusOne(arr);
        System.out.println(Arrays.toString(ans));
  }
}
```

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
Prgm:

```
class SingleNum {
    public static int singleNumber(int[] nums) {
        int count = 0;
        for (int i = 0; i < nums.length; i++) {
            count = 0;
            for (int j = 0; j < nums.length; j++) {
                if (nums[i] == nums[j]) {
                    count++;
                }
            }
            if (count == 1) {
                return nums[i];
            }
        }
        return 0;
    }
    public static void main(String[] args) {
        int nums[] = {2,2,1};
        System.out.println("Element occuring once is: " + singleNumber(nums));
    }
}
```

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]]
Prgm:

```
class Missing_Ranges {
    public static List<String> findMissingRanges(int[] nums, int lower, int upper) {
      int n = nums.length;
      List<String> ans = new ArrayList<>();
      if (n == 0) {
         ans.add(f(lower, upper));
         return ans;
      }
      if (nums[0] > lower) {
         ans.add(f(lower, nums[0] - 1));
      }
      for (int i = 1; i < n; ++i) {
         int a = nums[i - 1], b = nums[i];
         if (b - a > 1) {
            ans.add(f(a + 1, b - 1));
         }
      }
      if (nums[n - 1] < upper) {
         ans.add(f(nums[n - 1] + 1, upper));
      }
      return ans;
   }

   private static String f(int a, int b) {
      return a == b ? a + "" : a + "->" + b;
   }
   public static void main(String[] args) {
      int nums[] = {0,1,3,50,75};
      int lower = 0 ;
      int upper = 99 ;

      System.out.println(findMissingRanges(nums,lower,upper));
   }
}
```

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
Prgm:

```
class Meeting {
   public static boolean canAttendMeetings(int[][] intervals) {
      int n = intervals.length;
      int[] startTime = new int[n];
      int[] endTime = new int[n];
      int count = 0;
      for (int i = 0; i < n; i++) {
         startTime[count] = intervals[i][0];
         endTime[count++] = intervals[i][1];
      }
      Arrays.sort(startTime);
      Arrays.sort(endTime);
       for(int i = 1; i < n; i++){
          if(startTime[i] < endTime[i - 1])
          return false;
       }
       return true;
   }
   public static void main(String[] args) {
      int[][] intervals = { { 0, 30 }, { 5, 10 }, { 15, 20 } };
      System.out.println(canAttendMeetings(intervals));
   }
}
```