QUESTION-1

```
import java.util.*;

public class IntervalOverlap {

    public static boolean checkOverlap(long[] start, long[] end) {
        int n = start.length;
        Arrays.sort(start);
        Arrays.sort(end);
        for (int i = 0; i < n - 1; i++) {
            if (start[i + 1] < end[i]) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int T = scanner.nextInt();

        for (int t = 0; t < T; t++) {
            int N = scanner.nextInt();
            long[] start = new long[N];
            long[] end = new long[N];
            for (int i = 0; i < N; i++) {
                start[i] = scanner.nextLong();
            }
            for (int i = 0; i < N; i++) {
                end[i] = scanner.nextLong();
            }
            boolean result = checkOverlap(start, end);
            System.out.println(result);
        }
        scanner.close();
    }
}
```

QUESTION-2

```
import java.util.*;
```

```
public class MatrixSearch {

    public static int[] searchMatrix(int[][] matrix, int target) {
        int[] result = new int[]{-1, -1};
        int rows = matrix.length;
        int cols = matrix[0].length;

        int row = 0;
        int col = cols - 1;
        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                result[0] = row;
                result[1] = col;
                return result;
            } else if (matrix[row][col] < target) {
                row++;
            } else {
                col--;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int T = scanner.nextInt();

        for (int t = 0; t < T; t++) {
            int N = scanner.nextInt();
            int X = scanner.nextInt();
            int[][] matrix = new int[N][N];
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                    matrix[i][j] = scanner.nextInt();
                }
            }
            int[] result = searchMatrix(matrix, X);
            System.out.println(result[0] + " " + result[1]);
        }
        scanner.close();
    }
}
```

QUESTION-3

```
import java.util.Scanner;
 class Solution {

    public static int minimumOperations(String a, String b) {

        if (a.length() != b.length()) {
            return -1;
        }
        int n = a.length();
        int count = 0;

        char c1, c2, c3, c4;

        for (int i = 0; i < n / 2; i++) {

            c1 = a.charAt(i);
            c2 = a.charAt(n - i - 1);
            c3 = b.charAt(i);
            c4 = b.charAt(n - i - 1);
            if ((c1 == c2 && c3 == c4) || (c1 == c3 && c2 == c4) || (c1 == c4 && c2 == c3)) {
                continue;
            }

            else if (c1 == c3 || c1 == c4 || c2 == c3 || c2 == c4 || c3 == c4) {
                count++;
            }

            else {
                count += 2;
            }
        }

        if (n % 2 == 1 && a.charAt(n / 2) != b.charAt(n / 2)) {
            count++;
        }

        return count;
    }
        public static void main(String args[]){
                Scanner sc=new Scanner(System.in);
                int p=sc.nextInt();
                for(int i=1;i<=p;i++){
```

```
                String a=sc.next();
                String b=sc.next();
                System.out.println(minimumOperations(a,b));
            }
        }

}
```

QUESTION-4

```
import java.util.*;

public class ReplaceIsland {

    public static void replaceIsland(char[][] grid) {
        if (grid == null || grid.length == 0) return;

        int rows = grid.length;
        int cols = grid[0].length;

        for (int i = 0; i < rows; i++) {
            if (grid[i][0] == 'O') dfs(grid, i, 0);
            if (grid[i][cols - 1] == 'O') dfs(grid, i, cols - 1);
        }

        for (int j = 0; j < cols; j++) {
            if (grid[0][j] == 'O') dfs(grid, 0, j);
            if (grid[rows - 1][j] == 'O') dfs(grid, rows - 1, j);
        }

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == 'O') grid[i][j] = 'X';
                else if (grid[i][j] == '*') grid[i][j] = 'O';
            }
        }
    }

    private static void dfs(char[][] grid, int i, int j) {
        if (i < 0 || i >= grid.length || j < 0 || j >= grid[0].length || grid[i][j] != 'O') return;

        grid[i][j] = '*'; // Mark 'O' connected to boundary as '*'
        dfs(grid, i + 1, j);
        dfs(grid, i - 1, j);
```

```
        dfs(grid, i, j + 1);
        dfs(grid, i, j - 1);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int rows = scanner.nextInt();
        int cols = scanner.nextInt();

        char[][] grid = new char[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                grid[i][j] = scanner.next().charAt(0);
            }
        }

        replaceIsland(grid);

        System.out.println("Updated grid:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print(grid[i][j] + " ");
            }
            System.out.println();
        }

        scanner.close();
    }
}
```

QUESTION-5

```
import java.util.Scanner;
public class PalindromePartitioning {
    public static int minCut(String s) {
        int n = s.length();
        boolean[][] palindrome = new boolean[n][n];
        int[] dp = new int[n];

        for (int i = 0; i < n; i++) {
            dp[i] = i;
            for (int j = 0; j <= i; j++) {
                if (s.charAt(i) == s.charAt(j) && (i - j <= 1 || palindrome[j + 1][i - 1])) {
                    palindrome[j][i] = true;
```

```
                dp[i] = (j == 0) ? 0 : Math.min(dp[i], dp[j - 1] + 1);
            }
        }
    }

    return dp[n - 1];
  }

  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    String str = s.next();
    System.out.println(minCut(str));
  }
}
```

QUESTION-6

```
import java.util.Scanner;
public class EditDistance {
  public static int minDistance(String word1, String word2) {
    int m = word1.length();
    int n = word2.length();

    int[][] dp = new int[m + 1][n + 1];
    for (int i = 0; i <= m; i++) {
      dp[i][0] = i;
    }
    for (int j = 0; j <= n; j++) {
      dp[0][j] = j;
    }

    for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= n; j++) {
        if (word1.charAt(i - 1) == word2.charAt(j - 1)) {
          dp[i][j] = dp[i - 1][j - 1];
        } else {
          dp[i][j] = 1 + Math.min(dp[i - 1][j - 1], Math.min(dp[i - 1][j], dp[i][j - 1]));
        }
      }
    }

    return dp[m][n];
  }
```

```
    public static void main(String[] args) {
        Scanner sc  = new Scanner(System.in);
        String s = sc.next();
        String t = sc.next();
        System.out.println(minDistance(s, t));
    }
}
```

QUESTION-7

```
import java.util.*;

public class TopKFrequentWords {
    public static List<String> topKFrequent(String[] words, int k) {
        Map<String, Integer> wordCount = new HashMap<>();
        for (String word : words) {
            wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }
        PriorityQueue<Map.Entry<String, Integer>> pq = new PriorityQueue<>(
            (a, b) -> a.getValue() == b.getValue() ? b.getKey().compareTo(a.getKey()) : a.getValue()
- b.getValue()
        );
        for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {
            pq.offer(entry);
            if (pq.size() > k) {
                pq.poll();
            }
        }


        List<String> result = new ArrayList<>();
        while (!pq.isEmpty()) {
            result.add(0, pq.poll().getKey());
        }

        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int k = scanner.nextInt();
        scanner.nextLine();
```

```
        String[] words = scanner.nextLine().split(" ");
        List<String> result = topKFrequent(words, k);
        for (String word : result) {
            System.out.print(word + " ");
        }
        scanner.close();
    }
}
```

QUESTION-8

```
import java.util.Collections;
import java.util.PriorityQueue;
import java.util.Scanner;

public class RunningMedian {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        for (int i = 0; i < n; i++) {
            int num = scanner.nextInt();
            addNumber(num, maxHeap, minHeap);
            rebalance(maxHeap, minHeap);
            System.out.print(getMedian(maxHeap, minHeap) + " ");
        }
        scanner.close();
    }

    private static void addNumber(int num, PriorityQueue<Integer> maxHeap,
PriorityQueue<Integer> minHeap) {
        if (maxHeap.isEmpty() || num <= maxHeap.peek()) {
            maxHeap.offer(num);
        } else {
            minHeap.offer(num);
        }
    }

    private static void rebalance(PriorityQueue<Integer> maxHeap, PriorityQueue<Integer>
minHeap) {
        while (maxHeap.size() > minHeap.size() + 1) {
            minHeap.offer(maxHeap.poll());
```

```
        }
        while (minHeap.size() > maxHeap.size()) {
            maxHeap.offer(minHeap.poll());
        }
    }

    private static int getMedian(PriorityQueue<Integer> maxHeap, PriorityQueue<Integer>
minHeap) {
        if (maxHeap.size() == minHeap.size()) {
            return (maxHeap.peek() + minHeap.peek()) / 2;
        } else {
            return maxHeap.peek();
        }
    }
}
```