Batch - T5
Assignment No. - 1
Title – Part 1: Sorting Algorithm
Student Name - Sharaneshwar Bharat Punjal
Student PRN - 23520011

1) You are given two sorted array, A and B, where A has a large enough buffer at the end to hold B. Write a method to merge B into A in sorted order.

```
import java.util.Scanner;

public class Q1 {
    public static void merge(int[] A, int[] B, int n, int m) {
        int i = n - 1;
        int j = m - 1;
        int k = n + m - 1;

        while (i >= 0 && j >= 0) {
            if (A[i] > B[j]) {
                A[k] = A[i];
                i--;
            } else {
                A[k] = B[j];
                j--;
            }
            k--;
        }

        while (j >= 0) {
            A[k] = B[j];
            j--;
            k--;
        }
    }

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

        int[] A = new int[n + m];
        int[] B = new int[m];

        for (int i = 0; i < n; i++)
            A[i] = sc.nextInt();

        for (int i = 0; i < m; i++)
```

```
            B[i] = sc.nextInt();

        merge(A, B, n, m);

        for (int i = 0; i < n + m; i++)
            System.out.print(A[i] + " ");

        System.out.println();
        sc.close();
    }
}
```

Test Case 1:

```
5 5
1 3 5 7 9
2 4 6 8 10
```

Output:

```
1 2 3 4 5 6 7 8 9
```

Test Case 2:

```
5 3
1 3 5 7 9
2 4 6
```

Output:

```
1 2 3 4 5 6 7 9
```

Test Case 3:

```
6 5
2 4 6 8 10 12
1 3 5 7 9
```

Output:

```
1 2 3 4 5 6 7 8 9 10 12
```

Test Case 4:

```
3 3
10 20 30
15 25 35
```

Output:

```
10 15 20 25 30 35
```

2) Write a method to sort an array of string so that all the anagrams are next to each other.

```
import java.util.Arrays;
import java.util.Scanner;

public class Q2 {
    public static void sortAnagrams(String[] arr) {
        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {
```

```java
            for (int j = 0; j < n - i - 1; j++) {
                if (canonicalForm(arr[j]).compareTo(canonicalForm(arr[j + 1])) > 0) {
                    String temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    private static String canonicalForm(String s) {
        char[] chars = s.toCharArray();
        for (int i = 0; i < chars.length - 1; i++) {
            for (int j = 0; j < chars.length - i - 1; j++) {
                if (chars[j] > chars[j + 1]) {
                    char temp = chars[j];
                    chars[j] = chars[j + 1];
                    chars[j + 1] = temp;
                }
            }
        }
        return new String(chars);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        String[] arr = new String[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.next();
        sc.close();
        sortAnagrams(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

Test Case 1:

```
6
cat dog tac god act odg
```

Output:

```
[act, cat, tac, dog, god, odg]
```

Test Case 2:

```
4
listen silent enlist inlets
```

Output:

```
[enlist, inlets, listen, silent]
```

Test Case 3:

```
5
abc bca cab xyz zyx
```

Output:

```
[abc, bca, cab, xyz, zyx]
```

Test Case 4:

```
7
bat tab cat act tca rat tar
```

Output:

```
[bat, tab, act, cat, tca, rat, tar]
```

Q) Given a sorted array of *n* integers that has been rotated an unknown number of times, write code to find an element in the array. You may assume that the array was originally sorted in increasing order.

EXAMPLE

Input: find 5 in {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14}

Output: 8 (the index of 5 in the array)

```java
import java.util.Scanner;

public class Q3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        int target = sc.nextInt();
        sc.close();

        System.out.println(findElement(arr, target));
    }

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

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

            if (arr[mid] == target) {
                return mid;
            }

            if (arr[left] <= arr[mid]) {
                if (target >= arr[left] && target < arr[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            } else {
                if (target > arr[mid] && target <= arr[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }

        return -1;
    }
}
```

Test Case 1:

```
12
15 16 19 20 25 1 3 4 5 7 10 14
5
```

Output:

```
8
```

Test Case 2:

```
10
4 5 6 7 8 9 1 2 3
3
```

Output:

```
9
```

Test Case 3:

```
7
10 20 30 40 50 5 7
50
```

Output:

```
4
```

Test Case 4:

```
8
50 60 70 80 90 100 10 20
70
```

Output:

```
2
```

Q) Imagine you have a 20GB file with one string per line. Explain how you would sort the file.

The idea is to use the external sort algorithm. We can't bring all the data into memory, so we need to use a divide and conquer approach. We can divide the file into chunks which are x megabytes each, where x is the amount of memory we have available. Each chunk is sorted separately and then saved back to the file system. Once all the chunks are sorted, we merge the chunks one by one. We can use a min heap to keep track of the next element to write to the file. We read the first element of each chunk into the heap and then write the smallest one to the file.

We then read the next element from the chunk we took the smallest element from and add it to the heap. We continue this process until all elements have been written to the file. This is the external sort algorithm.

Q) Given a sorted array of string which is interspersed with empty string, write a method to find the location of a given string.

EXAMPLE

Input: find “ball” in {“at”, “”, “”, “ball”, “”, “”, “car”, “”, “”, “dad”, “”,””}

Output: 4

```
import java.util.Scanner;

public class Q5 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        String[] arr = new String[n];
        sc.nextLine();
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextLine();
        String target = sc.nextLine();

        sc.close();
        System.out.println(findString(arr, target));
    }

    public static int findString(String[] arr, String target) {
        if (arr == null || target == null || target.isEmpty()) {
            return -1;
        }
        return search(arr, target, 0, arr.length - 1);
    }

    private static int search(String[] arr, String target, int left, int right) {
        if (left > right) {
            return -1;
        }

        int mid = left + (right - left) / 2;

        if (arr[mid].isEmpty()) {
            int leftMid = mid - 1;
            int rightMid = mid + 1;

            while (true) {
                if (leftMid < left && rightMid > right) {
                    return -1;
                } else if (rightMid <= right && !arr[rightMid].isEmpty()) {
                    mid = rightMid;
                    break;
                } else if (leftMid >= left && !arr[leftMid].isEmpty()) {
                    mid = leftMid;
                    break;
                }
                rightMid++;
                leftMid--;
            }
        }

        if (arr[mid].equals(target)) {
            return mid;
        } else if (arr[mid].compareTo(target) < 0) {
            return search(arr, target, mid + 1, right);
        } else {
            return search(arr, target, left, mid - 1);
        }
    }
}
```

Test Case 1:

```
12
at
""
""
ball
""
""
car
""
""
dad
""
""
ball
```

Output:

```
3
```

Test Case 2:

```
8
apple
""
banana
""
""
cat
""
dog
banana
```

Output:

```
2
```

Test Case 3:

```
10
at
""
ball
""
""
""
car
""
""
cat
car
```

Output:

```
6
```

Test Case 4:

```
10
at
""
ball
""
""
""
car
""
""
cat
car
```

Output:

```
4
```

Q) Given an M*N matrix in which each row and each column is sorted in ascending order, write a method to find an element.

```
import java.util.Arrays;

import java.util.Scanner;

public class Q6 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt();
        int n = sc.nextInt();
        int[][] matrix = new int[m][n];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                matrix[i][j] = sc.nextInt();
        int target = sc.nextInt();
        sc.close();

        System.out.println(Arrays.toString(findElement(matrix, target)));
    }

    public static int[] findElement(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return new int[]{-1, -1};
        }

        int row = 0;
        int col = matrix[0].length - 1;

        while (row < matrix.length && col >= 0) {
            if (matrix[row][col] == target) {
                return new int[]{row, col};
            } else if (matrix[row][col] < target) {
                row++;
            } else {
                col--;
            }
        }

        return new int[]{-1, -1};
    }
}
```

Test Case 1:

```
3 4
1 4 7 11
2 5 8 12
3 6 9 16
8
```

Output:

```
[1, 2]
```

Test Case 2:

```
3 4
1 4 7 11
2 5 8 12
3 6 9 16
10
```

Output:

```
[-1, -1]
```

Test Case 3:

```
1 1
5
```

```
5
Output:
[0, 0]

Test Case 4:
0 0
5
Output:
[-1, -1]
```

Q) A circus is designing a tower routine consisting of people standing atop one another's shoulders. For practical and aesthetic reasons, each person must be both shorter and lighter than the person below him or her. Given the heights and weight of each circus, write a method to compute the largest possible number of people in such tower.

EXAMPLE:

Input(ht,wt): (65, 100) (70, 150) (56, 90) (75,190) (60, 95) (68, 110).

Output: The longest tower is length 6 and includes from top to bottom:

(56, 90) (60, 95) (65, 100) (68, 110) (70, 150) (75, 190)

```java
import java.util.Arrays;
import java.util.Scanner;

public class Q7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        int[][] people = new int[n][2];

        for (int i = 0; i < n; i++) {
            people[i][0] = sc.nextInt();
            people[i][1] = sc.nextInt();
        }

        sc.close();

        System.out.println(findLongestTower(people));
    }

    public static int findLongestTower(int[][] people) {
        if (people == null || people.length == 0) {
            return 0;
        }

        Arrays.sort(people, (p1, p2) -> {
            if (p1[0] != p2[0]) {
                return Integer.compare(p1[0], p2[0]);
            } else {
                return Integer.compare(p1[1], p2[1]);
            }
        });

        int[] dp = new int[people.length];
        Arrays.fill(dp, 1);

        for (int i = 1; i < people.length; i++) {
            for (int j = 0; j < i; j++) {
                if (people[i][0] > people[j][0] && people[i][1] > people[j][1]) {
                    dp[i] = Math.max(dp[i], dp[j] + 1);
                }
            }
        }

        return Arrays.stream(dp).max().getAsInt();
    }
}
```

Test Case 1:

```
6
65 100
70 150
56 90
75 190
60 95
68 110
```

Output:

```
6
```

Test Case 2:

```
4
80 200
75 210
85 180
90 170
```

Output:

```
1
```

Test Case 3:

```
5
65 100
70 150
56 105
75 130
60 90
```

Output:

```
3
```

Test Case 4:

```
8
56 90
60 110
65 100
70 150
75 130
80 190
68 120
66 80
```

Output:

```
5
```

Q) Imagine you are reading in stream of integers. Periodically, you wish to be able to look up the rank of number *x* (the number of values less than or equal to *x*). Implement the data structures and algorithms to support these operations. That is, Implement the method *track (int x),* which is called when each number is generated, and the method *getRankOfNumber (int x)*, which return the number of values less than or equal to *x* (not including x itself).

EXAMPLE

Stream (in order of appearance) : 5, 1, 4, 4, 5, 9, 7, 13, 3

getRankOfNumber(1) = 0
getRankOfNumber(3) = 1
getRankOfNumber(4) =3

```java
import java.util.Arrays;
import java.util.Scanner;

public class Q8 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        int stream[] = new int[n];
        for (int i = 0; i < n; i++) {
            stream[i] = sc.nextInt();
        }
        sc.close();

        Arrays.sort(stream);

        System.out.println(getRankOfNumber(stream, 1));
        System.out.println(getRankOfNumber(stream, 3));
        System.out.println(getRankOfNumber(stream, 4));
        System.out.println(getRankOfNumber(stream, 9));
    }

    private static int getRankOfNumber(int stream[], int x) {
        int left = 0;
        int right = stream.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (stream[mid] <= x) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left - 1;
    }
}
```

Test Case 1:

```
9
5 1 4 4 5 9 7 13 3
```

Output:

```
0
1
3
6
```

Test Case 2:

```
7
2 2 2 2 2 2 2
```

Output:

```
6
6
6
6
```

Test Case 3:

```
6
-3 -1 0 2 4 6
```

Output:

```
0
1
2
5
```

Test Case 4:

```
1
10
```

Output:

```
-1
-1
-1
0
```