Winsoft – Java (Assesment)

Q.1. Merge two arrays by satisfying given constraints
Given two sorted arrays X[] and Y[] of size m and n each where m >= n and X[] has exactly n vacant cells,
merge elements of Y[] in their correct position in array X[], i.e., merge (X, Y) by keeping the sorted order.

For example,

Input: X[] = { 0, 2, 0, 3, 0, 5, 6, 0, 0 }
Y[] = { 1, 8, 9, 10, 15 } The vacant cells in X[] is represented by 0
Output: X[] = { 1, 2, 3, 5, 6, 8, 9, 10, 15 }

➔

```
public class MergeArrays {
   public static void main(String[] args) {
      int[] X = { 0, 2, 0, 3, 0, 5, 6, 0, 0 };
      int[] Y = { 1, 8, 9, 10, 15 };

      mergeArrays(X, Y);

      System.out.print("Merged array X[]: ");
      for (int num : X) {
         System.out.print(num + " ");
      }
   }

   public static void mergeArrays(int[] X, int[] Y) {
      int m = X.length;
      int n = Y.length;

      int k = m - 1;
```

```
    for (int i = m - 1; i >= 0; i--) {
      if (X[i] != 0) {
        X[k] = X[i];
        k--;
      }
    }

    int i = k + 1;
    int j = 0;
    int l = 0;

      while (i < m && j < n) {
      if (X[i] < Y[j])
        X[l++] = X[i++];
      else
        X[l++] = Y[j++];
    }

    while (j < n)
      X[l++] = Y[j++];
  }
}
```

Q.2. Find maximum sum path involving elements of given arrays
Given two sorted arrays of integers, find a maximum sum path involving elements of both arrays whose sum is maximum.
We can start from either array, but we can switch between arrays only through its common elements.

For example,

Input: X = { 3, 6, 7, 8, 10, 12, 15, 18, 100 }
Y = { 1, 2, 3, 5, 7, 9, 10, 11, 15, 16, 18, 25, 50 }

```
The maximum sum path is: 1 –> 2 –> 3 –> 6 –> 7 –> 9 –> 10 –>
12 –> 15 –> 16 –> 18 –> 100
The maximum sum is 199
```

➔

```
public class MaximumSumPath {
   public static void main(String[] args) {
      int[] X = { 3, 6, 7, 8, 10, 12, 15, 18, 100 };
      int[] Y = { 1, 2, 3, 5, 7, 9, 10, 11, 15, 16, 18, 25, 50 };

      int maxSum = findMaximumSumPath(X, Y);
      System.out.println("The maximum sum is: " + maxSum);
   }

   public static int findMaximumSumPath(int[] X, int[] Y) {
      int m = X.length;
      int n = Y.length;
      int i = 0, j = 0;
      int sumX = 0, sumY = 0;
      int result = 0;

      while (i < m && j < n) {
         if (X[i] < Y[j])
            sumX += X[i++];
         else if (X[i] > Y[j])
            sumY += Y[j++];
         else {
            result += Math.max(sumX, sumY);
            result += X[i];
            sumX = 0;
            sumY = 0;
```

```
            i++;
            j++;
          }
        }

        while (i < m)
          sumX += X[i++];
        while (j < n)
          sumY += Y[j++];

        result += Math.max(sumX, sumY);

        return result;
    }
}
```

Q.3. Write a Java Program to count the number of words in a string using HashMap.

➔

```
import java.util.HashMap;
import java.util.Map;

public class WordCount {
    public static void main(String[] args) {
        String str = "My name is Siddesh. Siddesh likes playing Volleyball";

        Map<String, Integer> wordCountMap = new HashMap<>();

        String[] words = str.split("\\s+");
```

```
        for (String word : words) {

            word = word.replaceAll("[^a-zA-Z]", "");

            word = word.toLowerCase();

            if (word.length() > 0) {
                if (wordCountMap.containsKey(word)) {
                    wordCountMap.put(word, wordCountMap.get(word) + 1);
                } else {
                    wordCountMap.put(word, 1);
                }
            }
        }

        for (Map.Entry<String, Integer> entry : wordCountMap.entrySet()) {
            System.out.println("Word: " + entry.getKey() + ", Count: " + entry.getValue());
        }
    }
}
```

Q.4. Write a Java Program to find the duplicate characters in a string.

➔

```
        import java.util.HashSet;
import java.util.Set;

public class DuplicateCharacters {
    public static void main(String[] args) {
        String str = "hello world";
        Set<Character> uniqueChars = new HashSet<>();
```

```
        Set<Character> duplicateChars = new HashSet<>();



        for (char c : str.toCharArray()) {

            if (!uniqueChars.add(c)) {

                duplicateChars.add(c);
            }
        }

        if (!duplicateChars.isEmpty()) {
            System.out.println("Duplicate characters: " + duplicateChars);
        } else {
            System.out.println("No duplicate characters found.");
        }
    }
}
```