VIJAY M

22IT125 DAY 5

CODING PRACTICES AND PROBLEM

1.Bubble Sort

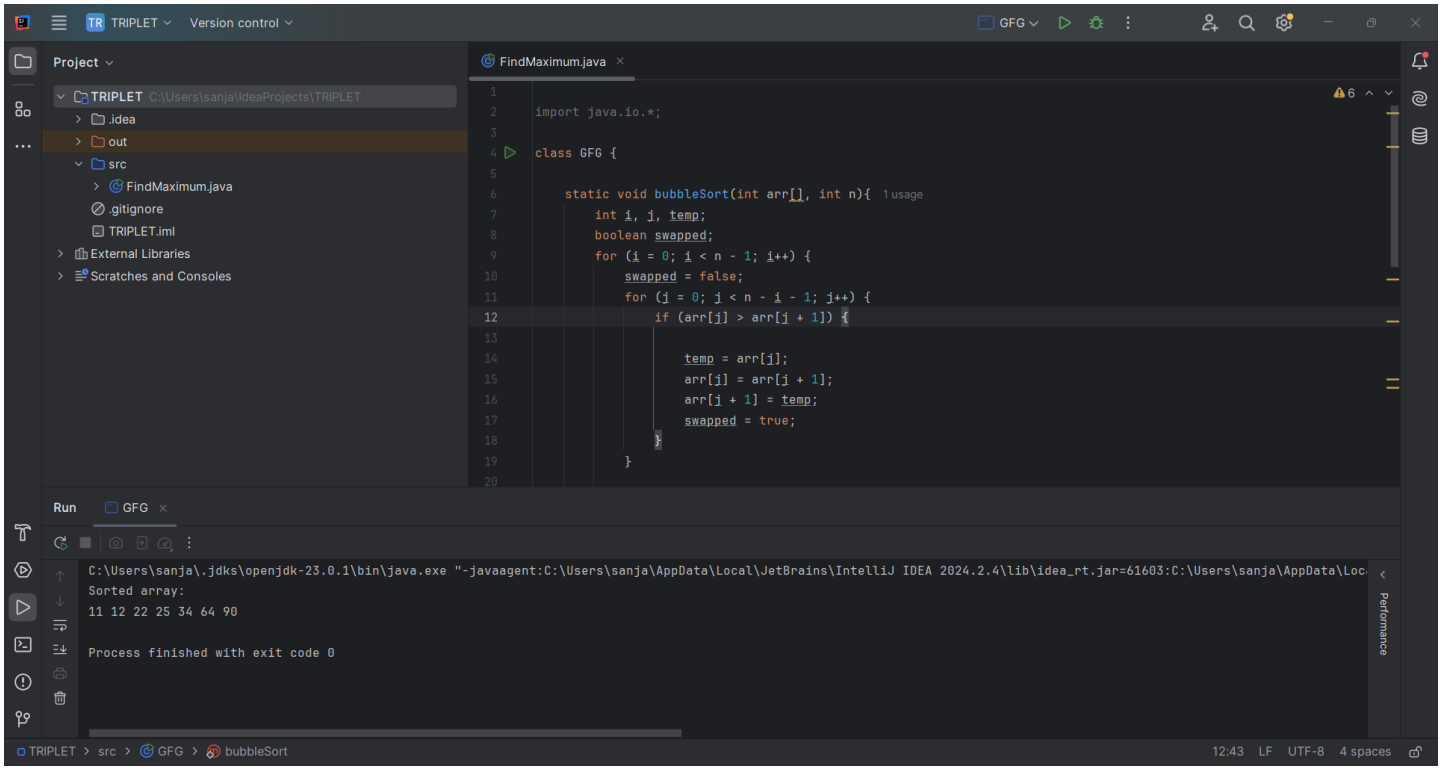
Time Complexity : O(n2)

Solution：

```
import java.io.*;

class GFG {

   static void bubbleSort(int arr[], int n){
       int i, j, temp;
       boolean swapped;
       for (i = 0; i < n - 1; i++) {
           swapped = false;
           for (j = 0; j < n - i - 1; j++) {
               if (arr[j] > arr[j + 1]) {

                   temp = arr[j];
                   arr[j] = arr[j + 1];
                   arr[j + 1] = temp;
                   swapped = true;
               }
           }



           if (swapped == false)
               break;
       }
   }
```

```
  static void printArray(int arr[], int size){
      int i;
      for (i = 0; i < size; i++)
          System.out.print(arr[i] + " ");
      System.out.println();
  }

  public static void main(String args[]){
      int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
      int n = arr.length;
      bubbleSort(arr, n);
      System.out.println("Sorted array: ");
      printArray(arr, n);
  }
}
```

Output:



2.Quick Sort

Time Complexity:

Best case : ($\Omega(n \log n)$)

Average Case : ($\theta(n \log n)$)

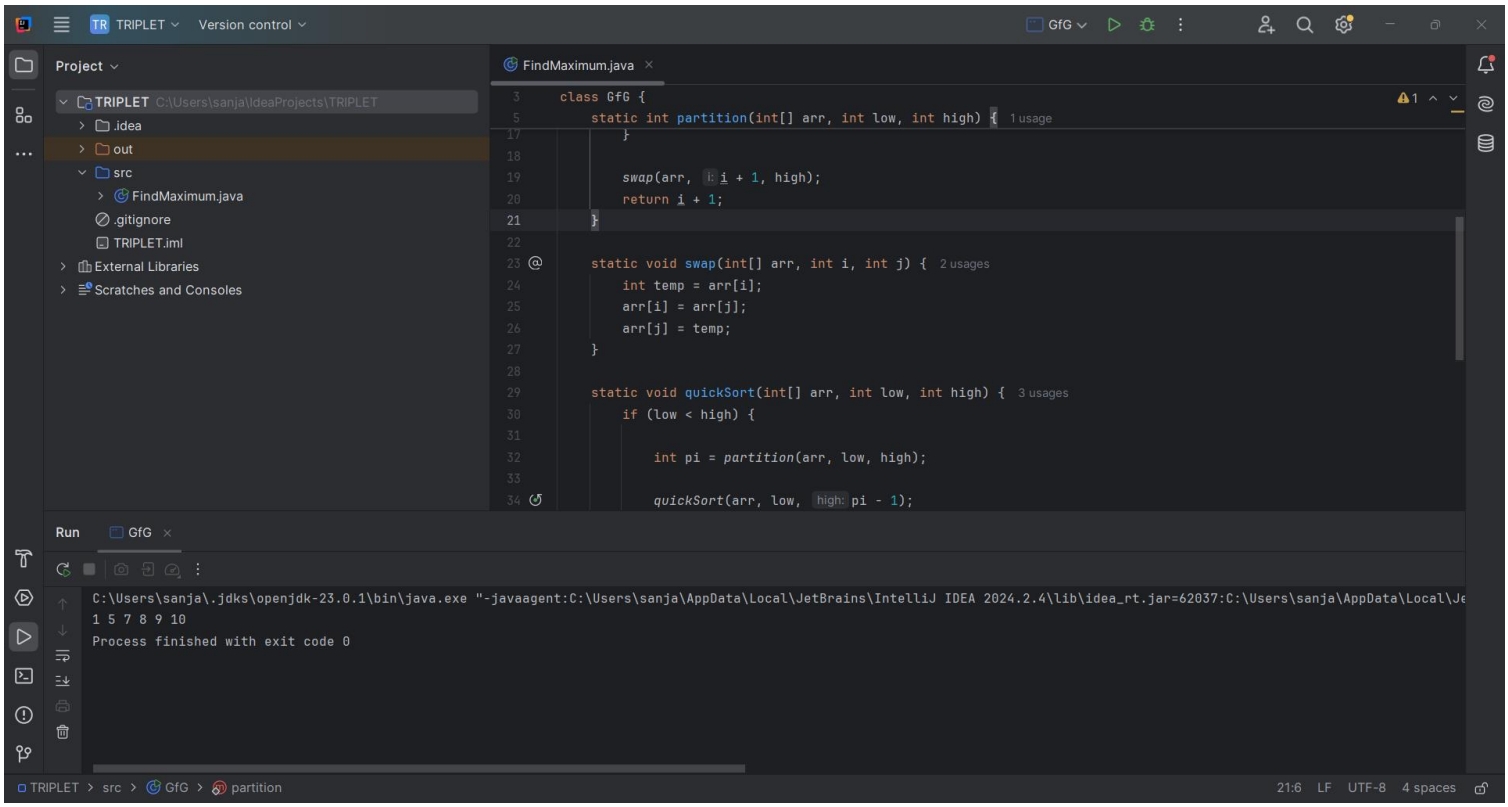
Worst Case : ($O(n^2)$)

Solution:

```
import java.util.Arrays;
class GfG {
    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
        }

        swap(arr, i + 1, high);
        return i + 1;
    }

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

    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {

            int pi = partition(arr, low, high);
```

```
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

public static void main(String[] args) {
    int[] arr = {10, 7, 8, 9, 1, 5};
    int n = arr.length;

    quickSort(arr, 0, n - 1);

    for (int val : arr) {
        System.out.print(val + " ");
    }
}
}
```

Output:



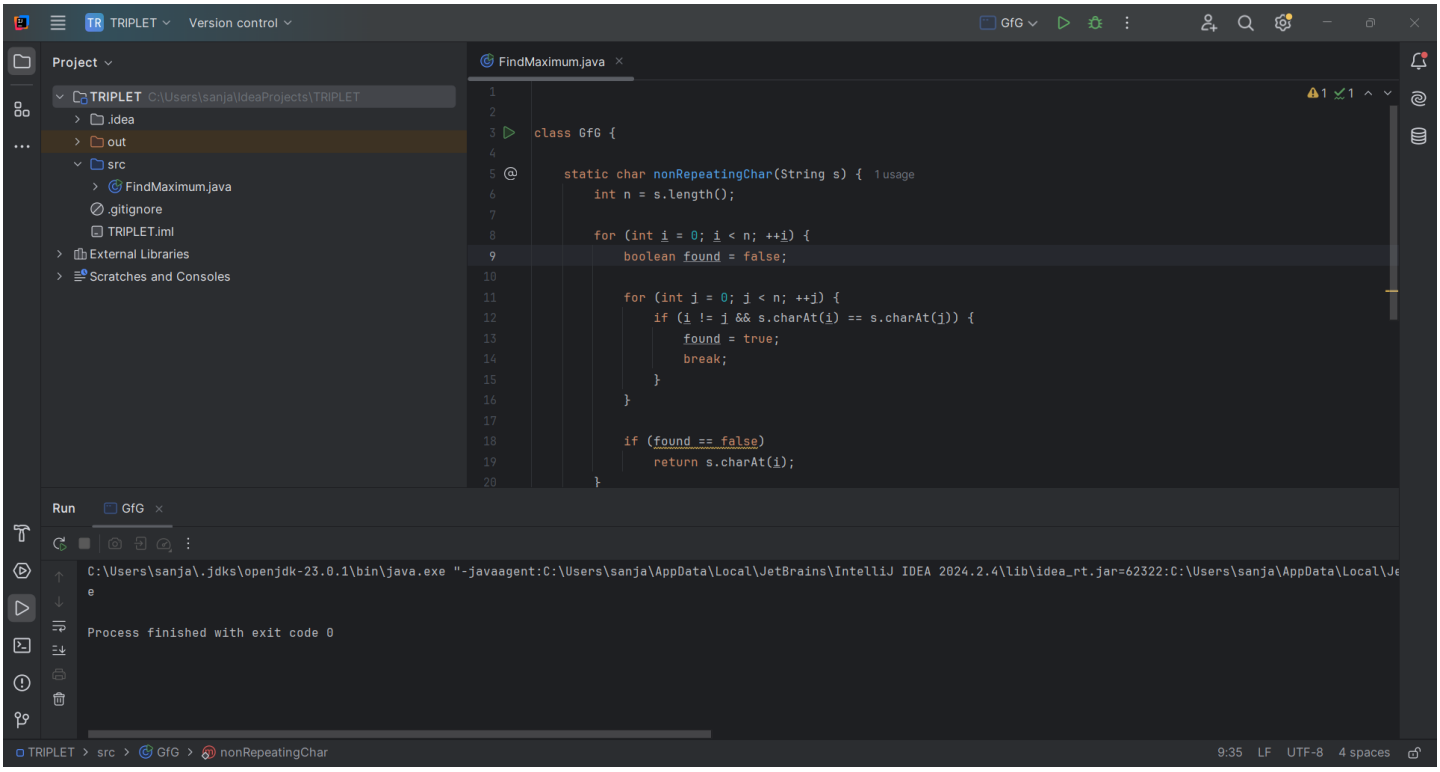
3.Non repeating Character

Time Complexity : O(n^2)

Solution:

```
class GfG {

    static char nonRepeatingChar(String s) {
        int n = s.length();

        for (int i = 0; i < n; ++i) {
            boolean found = false;

            for (int j = 0; j < n; ++j) {
                if (i != j && s.charAt(i) == s.charAt(j)) {
                    found = true;
                    break;
                }
            }

            if (found == false)
                return s.charAt(i);
        }

        return '$';
    }

    public static void main(String[] args) {
        String s = "racecar";

        System.out.println(nonRepeatingChar(s));
    }
}
```

Output:



4.Edit Distance

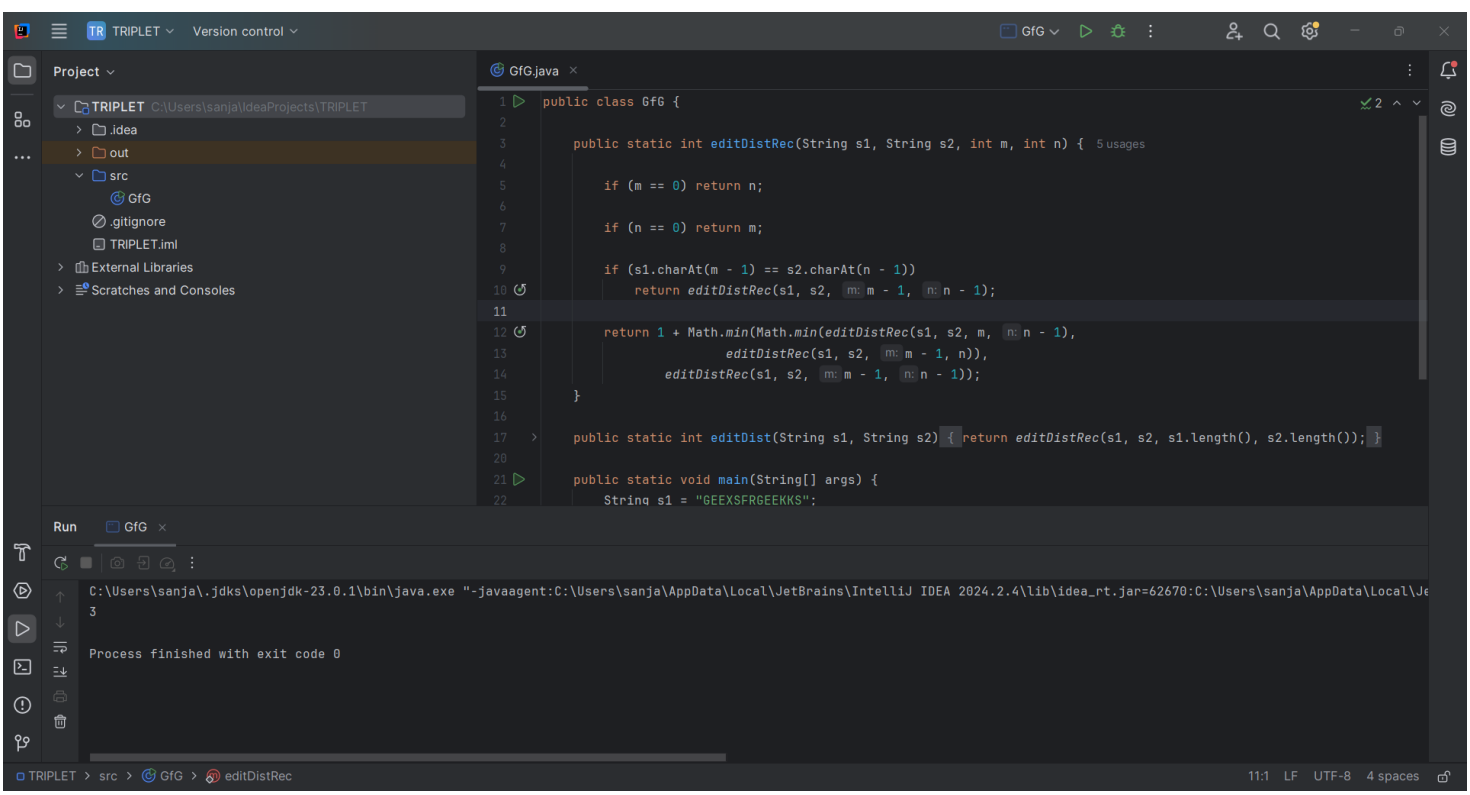
Time Complexity : O(m x n)

Solution:

```
public class GfG {

  public static int editDistRec(String s1, String s2, int m, int n) {

      if (m == 0) return n;

      if (n == 0) return m;

      if (s1.charAt(m - 1) == s2.charAt(n - 1))
          return editDistRec(s1, s2, m - 1, n - 1);

      return 1 + Math.min(Math.min(editDistRec(s1, s2, m, n - 1),
```

```
                editDistRec(s1, s2, m - 1, n)),
            editDistRec(s1, s2, m - 1, n - 1));
    }

    public static int editDist(String s1, String s2) {
        return editDistRec(s1, s2, s1.length(), s2.length());
    }

    public static void main(String[] args) {
        String s1 = "GEEXSFRGEEKKS";
        String s2 = "GEEKSFORGEEKS";
        System.out.println(editDist(s1, s2));
    }
}
```

Output:



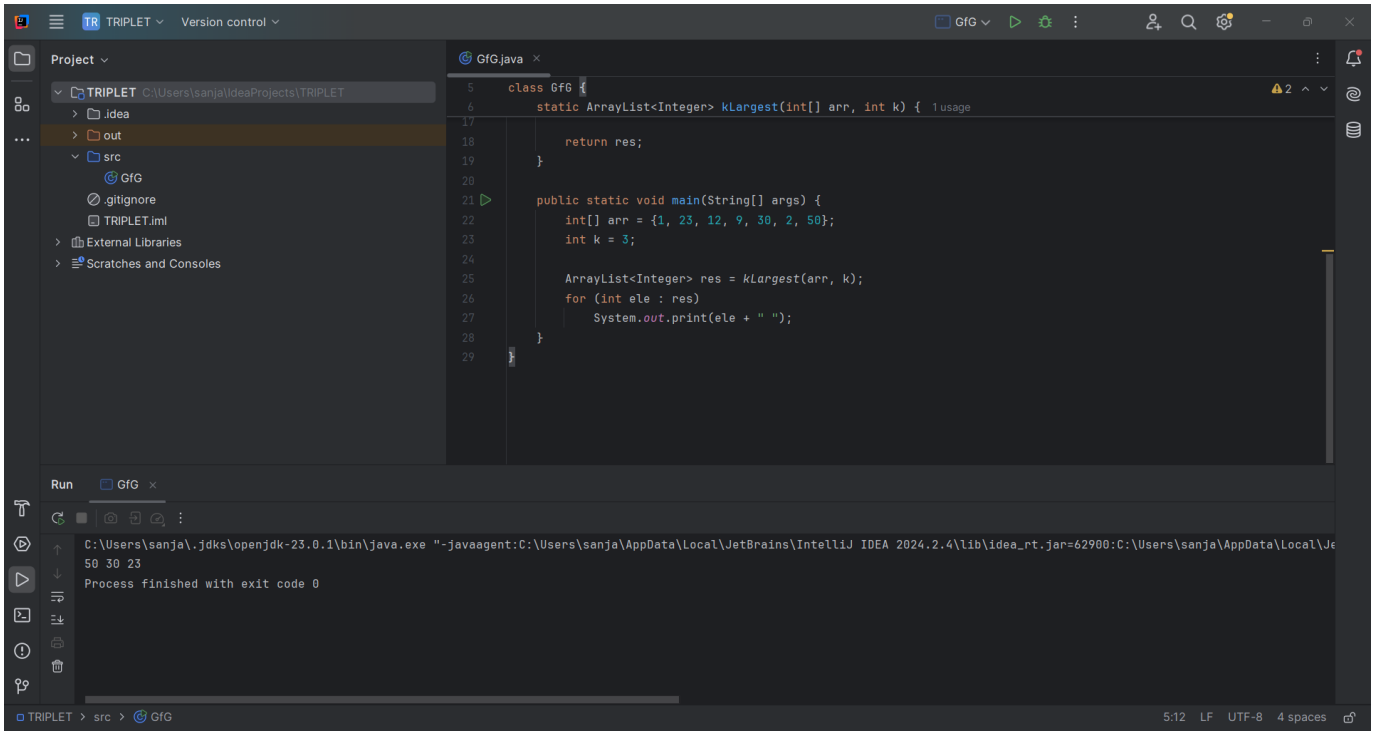
5.K Largest elements

Time Complexity : O(n * log(n))

Solution:

```
import java.util.*;

class GfG {
    static ArrayList<Integer> kLargest(int[] arr, int k) {
        int n = arr.length;

        Integer[] arrInteger =
                Arrays.stream(arr).boxed().toArray(Integer[]::new);

        Arrays.sort(arrInteger, Collections.reverseOrder());

        ArrayList<Integer> res = new ArrayList<>();
        for (int i = 0; i < k; i++)
            res.add(arrInteger[i]);

        return res;
    }

    public static void main(String[] args) {
        int[] arr = {1, 23, 12, 9, 30, 2, 50};
        int k = 3;

        ArrayList<Integer> res = kLargest(arr, k);
        for (int ele : res)
            System.out.print(ele + " ");
    }
}
```

Output:



6.Form The Largest Number

Time Complexity : O(n*log(n))

Solution:

```
import java.util.*;

class GfG {
   static boolean myCompare(String s1, String s2) {
      return (s1 + s2).compareTo(s2 + s1) > 0;
   }

   static String findLargest(int[] arr) {

      ArrayList<String> numbers = new ArrayList<>();
      for (int ele : arr) {
         numbers.add(Integer.toString(ele));
      }
```

```
    Collections.sort(numbers, (s1, s2) -> myCompare(s1, s2) ? -1 : 1);

    if (numbers.get(0).equals("0")) {
        return "0";
    }

    StringBuilder res = new StringBuilder();
    for (String num : numbers) {
        res.append(num);
    }

    return res.toString();
  }

  public static void main(String[] args) {
    int[] arr = { 3, 30, 34, 5, 9 };
    System.out.println(findLargest(arr));
  }
}
```

Output: