Merge Sort

Name : Ritu Choudhary　　　Roll no : 51　　　Batch : b3

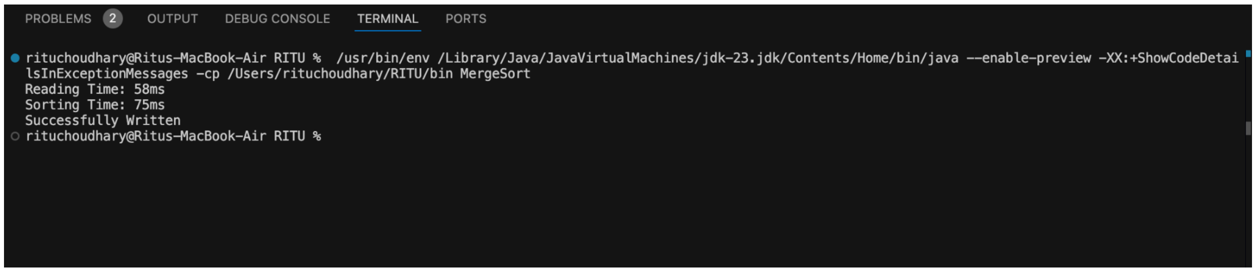
```java
import java.io.*;
import java.util.*;

class MergeSort {
   public static void main(String[] args) {
       String inputPath = "file2.txt";
       String outputPath = "MergeSort.txt";
       ArrayList<Integer> numbers = new ArrayList<>();
       long startTime = System.currentTimeMillis();
       try (BufferedReader br = new BufferedReader(new FileReader(inputPath))) {
           String line;
           while ((line = br.readLine()) != null) {
               String[] values = line.split("\\s+");
               for (String value : values) {
                   numbers.add(Integer.parseInt(value));
               }
           }
       } catch (IOException e) {
           System.out.println("Error reading file: " + e.getMessage());
           return;
       }
       long endTime = System.currentTimeMillis();
       long timeReq = (endTime - startTime);
       System.out.println("Reading Time: " + timeReq + "ms");
       startTime = System.currentTimeMillis();
       mergeSort(numbers, 0, numbers.size() - 1);
       endTime = System.currentTimeMillis();
       timeReq = (endTime - startTime);
       System.out.println("Sorting Time: " + timeReq + "ms");
       try (PrintWriter pw = new PrintWriter(new FileWriter(outputPath))) {
           for (Integer number : numbers) {
               pw.print(number + "\t");
           }
           System.out.println("Successfully Written");
       } catch (IOException e) {
           System.out.println("Error writing file: " + e.getMessage());
       }
   }
```

```java
  public static void mergeSort(ArrayList<Integer> arr, int low, int high) {
      if (low < high) {
          int mid = (low + high) / 2;
          mergeSort(arr, low, mid);
          mergeSort(arr, mid + 1, high);
          merge(arr, low, mid, high);
      }
  }

  public static void merge(List<Integer> arr, int low, int mid, int high) {
      ArrayList<Integer> left = new ArrayList<>(arr.subList(low, mid + 1));
      ArrayList<Integer> right = new ArrayList<>(arr.subList(mid + 1, high + 1));
      int i = 0, j = 0, k = low;
      while (i < left.size() && j < right.size()) {
          if (left.get(i) <= right.get(j))
              arr.set(k++, left.get(i++));
          else
              arr.set(k++, right.get(j++));
      }
      while (i < left.size())
          arr.set(k++, left.get(i++));
      while (j < right.size())
          arr.set(k++, right.get(j++));
  }
}
```

Output :



```
rituchoudhary@Ritus-MacBook-Air RITU %  /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCodeDetai
lsInExceptionMessages -cp /Users/rituchoudhary/RITU/bin MergeSort
Reading Time: 58ms
Sorting Time: 75ms
Successfully Written
rituchoudhary@Ritus-MacBook-Air RITU %
```

Comparing With SelectionSort And InsertionSort

Merge Sort is more efficient for large datasets with a time complexity of O(n log n), while Insertion Sort and Selection Sort are slower with O(n²) complexity.
Insertion Sort performs better on small or nearly sorted datasets, whereas Selection Sort is simpler but typically slower than Insertion Sort.

Merge Sort is stable but uses extra memory (O(n)), while Insertion Sort and Selection Sort are in-place with O(1) extra space, but Insertion Sort is stable and more efficient in practice.