```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

/**
 * COMP215-Programming Project 2: Multiple Sort Analysis.
 * ALGOTITHMTESTER holds the main testing methodology, as well as some utility methods.
 * @author Andrew Parsons
 * @version 05 March 2017
 */
class AlgorithmTester {

    private Sorter sorter;
    private MultiFileWriter multiFileWriter = MainApp.multiFileWriter;

    AlgorithmTester(Sorter sorter) {
        this.sorter = sorter;
    }

    void testAlgorithm(Comparable[] dataset, int repetitions) {

        Comparable[] dataToSort = dataset.clone();
        ArrayList<Long> timeList = new ArrayList<>();

        /* --- RANDOM SORT --- */
        for (int r = 0; r < repetitions; r++) {
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " ORIGINAL ", Arrays.toString(dataset));
            System.out.println();
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " R RAW ", Arrays.toString(dataToSort));
            System.out.println();
            if (sorter instanceof MergeSort || sorter instanceof QuickSort)
                dataToSort = sorter.sort(dataToSort, 0, dataToSort.length-1);
            else
                dataToSort = sorter.sort(dataToSort);
            timeList.add(sorter.getElapsedTime());
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " ORIGINAL ", Arrays.toString(dataset));
            System.out.println();
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " R SORTED ",Arrays.toString(dataToSort));
            // reset the dataToSort
            if (r != repetitions-1)
                dataToSort = dataset.clone();
        }
        try {
            multiFileWriter.processTestResult(new TestResult(dataToSort.length,
            calculateMean(timeList)), sorter,"random");
        } catch (Exception e) {
            System.out.println("Problem with the MultiFileWriter!");
            e.printStackTrace();
        }
        System.out.printf("%1$-15s %2$-30s %3$15d %4$15f",sorter.getClass().getCanonicalName(),
        ": completed RANDOM", dataset.length, ((double) calculateMean(timeList) / 1E6));
        System.out.println();

        /* --- ASCENDING SORT --- */
        // reset the ArrayList
        timeList = new ArrayList<>();

        for (int r = 0; r < repetitions; r++) {
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " ORIGINAL ", Arrays.toString(dataset));
```

```
            System.out.println();
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " A RAW ", Arrays.toString(dataToSort));
            System.out.println();
            if (sorter instanceof MergeSort || sorter instanceof QuickSort)
                dataToSort = sorter.sort(dataToSort, 0, dataToSort.length-1);
            else
                dataToSort = sorter.sort(dataToSort);
            timeList.add(sorter.getElapsedTime());
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " ORIGINAL ", Arrays.toString(dataset));
            System.out.println();
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " A SORTED ", Arrays.toString(dataToSort));
        }
        try {
            multiFileWriter.processTestResult(new TestResult(dataToSort.length,
            calculateMean(timeList)), sorter,"ascending");
        } catch (Exception e) {
            System.out.println("Problem with the MultiFileWriter!");
            e.printStackTrace();
        }
        System.out.printf("%1$-15s %2$-30s %3$15d %4$15f",sorter.getClass().getCanonicalName(),
        ": completed ASCENDING", dataset.length, ((double) calculateMean(timeList) / 1E6));
        System.out.println();

        /* --- DESCENDING SORT --- */
        // reset the ArrayList
        timeList = new ArrayList<>();

        // put the dataset in reverse order
        Arrays.sort(dataToSort, Collections.reverseOrder());

        for (int r = 0; r < repetitions; r++) {
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " ORIGINAL ", Arrays.toString(dataset));
            System.out.println();
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " D RAW ", Arrays.toString(dataToSort));
            System.out.println();
            if (sorter instanceof MergeSort || sorter instanceof QuickSort)
                dataToSort = sorter.sort(dataToSort, 0, dataToSort.length-1);
            else
                dataToSort = sorter.sort(dataToSort);
            timeList.add(sorter.getElapsedTime());
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " ORIGINAL ", Arrays.toString(dataset));
            System.out.println();
            //System.out.printf("%1$-15s %2$-10s %3$-30s
            ",sorter.getClass().getCanonicalName(), " D SORTED ", Arrays.toString(dataToSort));

            // reset the dataToSort
            Arrays.sort(dataToSort, Collections.reverseOrder());
        }
        try {
            multiFileWriter.processTestResult(new TestResult(dataToSort.length,
            calculateMean(timeList)), sorter,"descending");
        } catch (Exception e) {
            System.out.println("Problem with the MultiFileWriter!");
            e.printStackTrace();
        }
        System.out.printf("%1$-15s %2$-30s %3$15d %4$15f",sorter.getClass().getCanonicalName(),
        ": completed DESCENDING", dataset.length, ((double) calculateMean(timeList) / 1E6));
        System.out.println();
```

```
    }

    /**
     * Calculates the mean time to sort an array using insertion sort.
     * @param arrayOfTimes, an array of sorting times for an array, using insertion sort.
     * @return long, the mean time listed in the parametrized array.
     */
    private long calculateMean(ArrayList<Long> arrayOfTimes) {

        long sum = 0;
        long size = arrayOfTimes.size();

        if (arrayOfTimes.isEmpty())
            return 0;

        for (long time: arrayOfTimes) {
            sum += time;
        }
        return sum / size;
    }
}
```