Problem Set 3

Collaboration Policy. You are encouraged to work with other students on solving these problems. However, you **must** write up your solution **by yourself**. We may, randomly, ask you questions about your solution, and if you cannot answer them, we will assume you have cheated. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Problem 3. (The CS2040S Detectives)

We have six impostors on ours hands. Each *claims* to be **Mr. QuickSort**, the most popular sorting algorithm around. Only one of these six is telling the truth. Four of the others are just harmless imitators, **Mr. BubbleSort**, **Ms. SelectionSort**, **Mr. InsertionSort**, and **Ms. MergeSort**. Beware, however, one of the impostors is *not a sorting algorithm*: *Dr. Evil* maliciously returns unsorted arrays! And he won't be easy to catch. He will try to trick you by often returning correctly sorted arrays. Especially on easy instances, he's not going to slip up.

Your job is to investigate, and identify who is who and what is what. Attached to this problem set, you will find six sorting implementations: (i) `SorterA.class`, (ii) `SorterB.class`, (iii) `SorterC.class`, (iv) `SorterD.class`, (v) `SorterE.class`, and (vi) `SorterF.class`. These are provided in a single JAR file: `Sorters.jar`. Each of these class files contains a class that implements the `ISort` interface which supports the following method:

```
public void sort(KeyValuePair[] array)
```

You can find the code for the `KeyValuePair` class attached as well: it is a simple container that holds two integers: a key and a value. The sort routines will sort the array of objects by key.

You can test these sorting routines in the normal way: create an array, create a sorter object, and sort. See the example file `SortTestExample.java`.

You can then use the `StopWatch` to measure how fast each of these sorting routines runs. Each sorting algorithm has some inputs on which it is fast, and some inputs on which it is slow. Some sorting algorithms are stable, and others are not. Using these properties, you can figure out who is who, and identify Dr. Evil.

Beware, however, that these characters can be deceptive. While they cannot hide their asymptotic running time, they may well choose to run consistently slower than you expect. (You should not assume that QuickSort is always the fastest, for example. You should only rely on relative speed, not absolute speed.)

IntelliJ tips: The first thing you will need to do is to import the class files into your project in IntelliJ.

- Put the `ISort.java` file in the `src` folder.
- Put the `Sorters.jar` file in the `src` folder.
- Right click on the .jar file and click "Add as Library."

Problem 3.a.   Write a routine `boolean checkSort(ISort sorter, int size)` that runs a test on an array of the specified size to determine if the specified sorting algorithm sorts correctly.

Problem 3.b.   Write a routine `boolean isStable(ISort sorter, int size)` that runs a test on an array of the specified size to determine if the specified sorting algorithm is stable.

Problem 3.c.   Write whatever additional code you need in order to test the sorters to determine which is which. All evidence you give below you rely on properties of the sorting algorithms, along with data from your tests that supports your claim.

Problem 3.d.   What is the true identity of SorterA? Give the evidence that proves your claim.

Problem 3.e.   What is the true identity of SorterB? Give the evidence that proves your claim.

Problem 3.f.   What is the true identity of SorterC? Give the evidence that proves your claim.

Problem 3.g.   What is the true identity of SorterD? Give the evidence that proves your claim.

Problem 3.h.   What is the true identity of SorterE? Give the evidence that proves your claim.

Problem 3.i.   What is the true identity of SorterF? Give the evidence that proves your claim.