CS2040S: Data Structures and Algorithms

Problem Set 3

Due: Friday, February 4, 11:59pm

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). You can do so in Coursemology by adding a Comment. Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Problem 1. (The Sorting Detectives)

We have six impostors on our hands. Each *claims* to be **Mr. QuickSort**, the most popular sorting algorithm around. However, only one of these six is telling the truth. Four of the other five 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. Attached to this problem set, you will find six sorter 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, while others are not. Using these properties, you can figure out the real identities of the sorters, and also 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. (For example, you should not assume that QuickSort is always the fastest.) Evidence based on comparison of runtime across different sorters will not be accepted. Only evidence based on *asymptotic running time* are valid.

IntelliJ tips: Refer to `setup.mp4` on Coursemology for instructions on setting up PS3 in IntelliJ.

Note: You are only allowed to modify `SortingTester.java` in your submission. Any modifications made to the other files provided (i.e., `ISort.java`, `KeyValuePair.java`, and `StopWatch.java`) will not be accepted.

Problem 1.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 1.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 1.c. Write whatever additional code you need in order to test the sorters to determine which is which. All evidence you give below must rely on properties of the sorting algorithms, along with data from your tests that supports your claim.

Problem 1.d. What is the true identity of `SorterA`? Give the evidence that proves your claim.

Mergesort - performance does not significantly worsen with size and stable

Problem 1.e. What is the true identity of `SorterB`? Give the evidence that proves your claim.

Dr Evil. Returns false for checksort sometimes

Problem 1.f. What is the true identity of `SorterC`? Give the evidence that proves your claim.

Bubble sort - stable, performance worsens significantly as size increases

Problem 1.g. What is the true identity of `SorterD`? Give the evidence that proves your claim.

Selection sort - unstable, performance does not sigfnificantly worsen with size, sorted and reverse sorted about the same time

Problem 1.h. What is the true identity of `SorterE`? Give the evidence that proves your claim.

Quicksort - performance worsens when list is reverse sorted instead of sorted

Problem 1.i. What is the true identity of `SorterF`? Give the evidence that proves your claim.

insertion sort - unsorted array of size 100000 vs sorted, ratio of time is 10000 (about n), stable