CS2040 Data Structures and Algorithms: Problem Set 3

Due: Check Coursemology

Harold Soh

This assignment is GRADED. Please take note of the due date and submit your solutions on Coursemology.

Overview In this problem set, you're a detective: use your knowledge of sorting algorithms to determine which algorithm is which.

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.

A Mystery of Sorts

(Total: 30 points)

Dear Mr. Suresort Holmes,

We are writing to you in the hopes of obtaining your assistance.

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**. But one of the impostors is *not a sorting algorithm*: *Prof. NotaSort* maliciously returns unsorted arrays! Prof. NotaSort has proven very difficult to identify: he tricks us by sometimes returning correctly sorted arrays. Especially on easy instances, he doesn't slip up.

Please help us. We need someone of your brilliance to identify who is who and what is what. Attached to this letter, 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 <T extends Comparable<T>> void sort(T[] array)
```

You can test these sorting routines in the normal way: create an array, e.g., `Integer testArray[]`, and create a sorter object, e.g., `ISort sorter = new SorterA()`. Then, to sort, simply call:

`sorter.sort(testArray).`

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 Prof. NotaSort.

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 all sized inputs!)

You'll need two test functions that will help you figure out who is who.

Problem 1.1 (6 points) Implement the function:

```
boolean checkSorted(ISort sorter, int size)
```

which tests whether the sorter works correctly by sorting an input of the specified size (once) and checking if it is sorted.

Problem 1.2 (8 points) Implement code for the following function that tests whether a given sort is stable by testing it on an input of the specified size.

```
boolean isStable(ISort sorter, int size)
```

The function should just check if a result is stable, not necessarily sorted. If you want to check *both* sortedness and stability, you should use both `checkSorted` and `isStable`. To do a full test, you may need to run both functions several times.

Problem 1.3 (6 points) Correctly identify each of the six sorting algorithms by filling in the blanks below with the name of the respective sort. Enter your answer on Coursemology.

A: _____
B: _____
C: _____
D: _____
E: _____
F: _____

Problem 1.4 (10 points) Give an explanation of how you performed the identification. Ideally, your identification should be based on how the algorithms execute on certain inputs. The key is to identify certain unique properties of each algorithm.

We implore you to help us. Please submit your responses on Coursemology.

Eagerly awaiting your reply,

J. LeSort

Inspector James LeSort
Sortland Yard

— END OF PROBLEM SET 3 —