HW9: Detecting sorting algorithms

Due: 1:59M, Monday, November 24 through turnin
Total: 20pts

You may work on this assignment with one other student. A team of two members must practice pair programming. Pair programming "is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test." Both members in a team must read the article All I really need to know about pair programming I learned in kindergarten. This article can be downloaded to any computer that is in the Marquette network.

Preparation

Read the sorting algorithms and source codes in sections 10.2-10.3 in the textbook. Pay attention to the following characteristics of each algorithm:

1) The worst, average, and best case input arrays for a given array size.
2) The number of comparisons, data swaps and data copies the algorithm performs for worst, average, and best cases. A data swap requires three assignments, while a data copy requires one assignment.
3) The extra space the algorithm requires. In-place algorithms need constant (O(1)) extra space, while out-of-space algorithms need more. For example, mergesort needs O(n) extra space. Though both mergesort and quicksort performs O(nlgn) comparisons, mergesort runs slower than quicksort for average cases.

Task overview

For this assignment, the instructor has provided a file `Mysteries.jar` that contains five mystery sorting algorithms. These five algorithms are named by sort1, sort2, sort3, sort4, and sort5. Each algorithm accepts an integer array argument and sorts the elements in the array. For example, the following segment will be valid.

```
//class Mysteries is implemented in Mysteries.jar
int numValues = 10;
Mysteries theMystery = new Mysteries();
int[] values = new int[numValues];
theMystery.sort1(values);
theMystery.sort2(values);
theMystery.sort3(values);
theMystery.sort4(values);
theMystery.sort5(values);
```

These algorithms in alphabetic order are:

- bubble sort, which is implemented as bubbleSort on textbook page 684. Note that the bubblesort on page 685 is a little different.
- insertion sort
- merge sort

- quick sort, where the last element is used as pivot
- selection sort

Your job is to figure out which algorithm is implemented by each mystery sort.

Programming (70% of the total points)

The first part of your job is to write a driver program to find out the mapping between the mystery algorithms and the actual ones. This driver program shall be named `MysterySolver.java`.

You will design experiments, generate various arrays and pass them to the mystery sorting algorithms, and time the executions. The following is an example of how to time a code segment.

```
long startTime, endTime;
startTime = System.currentTimeMillis();
theMystery.sort1(values) ;                //time algorithm sort1
endTime = System.currentTimeMillis();
System.out.println("Elapsed time: " + endTime - startTime + " Millisecond" );
```

Download `Mysteries.jar` and place it in the directory of your project. To compile `MysterySolver.java` with `Mysteries.jar`, use command at cmd console:

```
javac –classpath "Mysteries.jar;." MysterySolver.java
```

or add `Mysteries.jar` as a library in eclipse. Your program doesn't require commandline arguments and shall run as:

```
java –classpath "Mysteries.jar;." MysterySolver
```

Write-up (30% of the total points)

You will write a report that includes a description of your experiments, the experimental data and analysis, and conclusion about the map between the mystery sorts and actual algorithms. Submit your report as `MysterySolver.pdf` to turnin.

Submission.

Submit `MysterySolver.java`, and `MysterySolver.pdf` to turnin system. You can have multiple submissions but only the last one will be saved by turnin. If you work in a team of two, always let the same user submit the file. The lower grade from two submissions will be used if each team member submits.