Sorting Detective

The primary objective of this assignment is for you to apply your theoretical knowledge of sorting efficiency to solve a problem of lost code. More specifically, you will be given a program that is designed to measure execution time, # comparisons, and # movements for the four sorts we've learned (selection, insertion, bubble, and merge). Unfortunately, the actual code was lost and all that is left is the executable, so no one knows anymore which sort is which!

As you know from class, the characteristics of the input data set can affect the expected performance of many of our sorting algorithms (e.g. an already sorted list vs. a mostly sorted list vs. a random list vs. a reversed list). Before you begin the lab, you should review the expected performance of the algorithms on various data sets. Note that "movements" is not quite the same things as "swaps". Each time something is copied, that is considered 1 movement. So an entire swap actually takes 3 movements even though it's just one line of code, as to swap items a & b you must copy item a to a temp variable, then copy item b over item a, then copy the temp variable over item b.

We haven't talked about merge sort yet, but you should be able to figure out which it is as simply the one that doesn't correspond to what you expect from any of the 3 sorts you do know.

Deliverables: A short report (likely less than 1 page, 2 at the most if you choose to include graphs) describing which sort (insertion, selection, bubble, or merge) corresponds to the buttons alpha, beta, delta, and gamma buttons, and your data/evidence/graphs/reasoning for your responses.

Instructions

1. Navigate to the Code folder in terminal and type the following command to run the application. Note you need to TYPE this, it will likely not work if you copy and paste from a pdf…

   ```
   java –classpath ".:jpt.jar" SortDetective
   ```

   If for some reason that doesn't work, recompile the code with following command

   ```
   javac –classpath ".:jpt.jar" SortDetective.java
   ```

   Then use the first command to try running again.

2. Play with the application for a bit. Notice the information you are given when you run a sort. Notice also, that if you create a small list, then that list is shown to you in the console window. In the unlikely event that a sort fails (oops!), a message will appear there as well.

3. Devise a set of tests which will enable you to identify the sorts by trying them on different lists of numbers.

4. Describe the results of your experiment in a short report. State which button corresponds to each of our 4 sorts and then explain/show the rationalization process that justifies it.

Note: The report needn't detail every experiment you ran, but it should give sufficient information to justify your conclusions. Statements such as "because it's fast/slow" are not sufficient information. The one exception is Merge Sort, which we haven't talked about yet, so your justification will simply be that it's the one left over when you've matched the other 3 sorts to buttons.

This lab is a modified version of the sorting detective assignment from Nifty Assignments (http://nifty.stanford.edu/), written by David B. Levine