Homework 3: Evaluating Sorting Algorithm Runtime  
Due Friday, February 25th, 10:15 AM

We’ve spent the last few class periods discussing the implementation of a number of elementary sorting algorithms. For this assignment you will empirically test the theoretical runtime predictions of the three sorting algorithms we have studied: Bubble sort, Insertion sort, and Selection sort. In particular, we want to assess the following claims:

- Bubble sort has $O(N^2)$ runtime on random data
- Insertion sort has $O(N^2)$ runtime on random data
- Selection sort has $O(N^2)$ runtime on random data but will be faster than the other two algorithms as it performs fewer swaps
- Insertion sort has $O(N)$ runtime on "almost sorted" data whereas the other two algorithms still have $O(N^2)$ runtime

Part A — Data Collection

In order to verify these claims, you should perform the following experiment comparing the three algorithms. You will also compile your collected data and report on the efficiency of each of the algorithms as compared to its theoretical runtime. To begin, you will need to collect some data in regards to the runtime of the algorithms:

1. Create a program SortTest that has three methods available to sort integer arrays, one using each technique.

2. Define three methods (one for each sorting algorithm) as compactly and efficiently as possible (e.g., do not include extraneous output statements). Make sure that the context for each method is as similar as possible to the other methods so that any variation in run time is due only to the algorithm being used, not a lack of symmetry in variable usage, method usage, etc. If you find that you are having trouble uniforming your methods, you should feel free to refer to your in class notes and to the algorithms’ descriptions in the textbook.

3. Collect data about the average runtime of each algorithm on random data by creating an array of 100 random integers and then measuring the time it takes each of your algorithms to sort that same random array (consider whether you need to do any work to make sure that later methods are not simply sorting a now-sorted array that is produced by the first method being run on the random array. Use the debugger to make entirely sure this is not happening).

4. Repeat this routine, in total creating 10 different arrays of 100 random integers and sorting them with each of the three separate algorithms. Report the average runtime required by each algorithm to sort 100 random integers. The System.nanoTime() method will help you measure runtime, but be sure that you are placing your time sampling calls in the appropriate place to allow you to average the runtime for each individual sort across the entire set of sorts being performed. Make note of the data collected. You will need to analyze it later and include it in your final report.

5. Using the same technique as above, report the average runtime required by each algorithm to sort 1000 random integers, then 5000 random integers, then 10000 random integers, then 50000 random integers, again making note of the growth in runtime.

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/us/.
6. Finally, produce 10 "almost sorted" integer arrays of each of size 100, 1000, 5000, 10000, and 50000 and collect runtime data for each algorithm on these arrays. We will consider an "almost sorted" array to be one where 6% of the data is out of sorted order. This means you should create an array of sorted data and then perform random swaps (swaps where the starting and ending location are randomly chosen) until 6% of the array has taken part in a swap. So for an array of size 100 you should do 3 swaps, not worrying about whether the randomness leads to the same item being swapped more than once.

Part B — Formal Algorithmic Analysis

Given the data you collected in Part A of this assignment, you should submit a well-written, well-organized laboratory report that satisfies each of the following criteria for each hypothesis:

1. Summarize the hypothesis and how it is being tested.

2. Present your data from any relevant test in both a table format and using suitable graphs. If you choose to include tables or graphs generated by Microsoft Excel, you should remember to copy and paste them into your final written report. Remember to represent data appropriately (e.g., label axes on graphs, include units in table headings, etc.).

3. Argue whether your data supports the hypothesis being analyzed. Specify how your data supports your analysis and what the data illustrates.

4. Conclude each section by discussing what issues arise when comparing the theoretically predicted runtime with empirically measured runtime. For what reasons might your empirical data not necessarily match the theoretically predicted runtime?

A well-written lab report should fill around 5-6 pages, although there is no precise page limit required for this assignment. In addition to submitting a written lab report, you must also provide a copy SortTest.java. Remember that the data collected and your lab report will be worth half of your grade on this assignment; be sure you complete your implementation in enough time to give your write-up sufficient attention.