Computer Science 501
Data Structures & Algorithms
The College of Saint Rose
Fall 2013

Programming Project 2: Analyzing Sorting Algorithms

Due: 6:00 PM, Wednesday, November 6, 2013

For your second programming project, you will perform an empirical analysis of several sorting algorithms that you can compare with the theoretical expectations, and to write up a formal reports of your results.

You may work individually or in groups of 2 or 3 on this project.

Software Development

Develop one or more Java programs that will help us to perform an empirical comparison of the following sorting algorithms:

- bubble sort
- selection sort
- insertion sort
- merge sort
- quicksort using first element as pivot
- quicksort using random pivot
- quicksort using median of three to determine pivot

Your program should operate on arrays of `int` values. It should have options to set the array size, the number of trials (to improve timing accuracy), the ability to generate initial data that is sorted, nearly sorted, completely random, and reverse sorted. Design your program to make it easy to implement a variety of sorting algorithms. Include the ability to count basic operations (number of comparisons and/or number of swaps) as well as to generate timings. You will need the ability to generate and report tabular data to show how your sorting algorithms perform on different sizes and distributions of data.

Timings and Analysis

Use your program(s) to generate basic operation counts and timing data for each sorting algorithm on an appropriate range of data sizes and distributions. Compare your results with your expectations based on our efficiency analysis of these algorithms. If you find discrepencies, how might

they be explained? Please include as many of the details of your test environment as you can: the type of processor or processors in the computer including clock speed, cache sizes, memory sizes, the operating system and version running on the computer, and the Java version you are using.

Tips, Tricks, Precautions, and Suggestions

- Use command line parameters rather than prompts, as this makes it much easier when running many (likely hundreds or thousands) of trials to generate timing results. `args[]` has what you need! If you don't know how to run with command-line parameters inside your IDE, run your Java program at the command line. That's what you'll want to do when generating timing results anyway.
- Have one big program rather than lots of little ones. This will help you avoid repeated code as you implement each of the sorting algorithms within the same framework.
- Be careful that you don't reuse an array of values for multiple runs, since all but the first could end up having already-sorted data as input.
- A simple tabular format of output will help you manage the creation of tables and/or graphs. Something like

  ```
  10000 bubble random .034693
  ```

  might indicate for an input size of 10,000, using a bubble sort on random input took .034693 seconds.
- A script or scripts that run all of the necessary instances of your program will likely be useful. Unix shells like `tcsh` and `bash` have powerful looping constructs. Combine that with the command-line parameters and some handy output redirection, and you have a straightforward way to manage your runs and generate your data. If you're typing in lots of problem sizes, etc, to launch runs, and copying and pasting numbers around, you're making this a lot harder than it needs to be.
- The runs should vary the input array size for each combination of sorting algorithm and input data type. To get meaningful results, you want a pretty wide range of sizes. You might start with an array of size 1000 (or better yet 1024) and double the size until you have an input size of 1,000,000 (or better yet 1,048,576). Take the average or best times (and justify your choice) for some number of runs, probably a few dozen to a few hundred. Then, plot your results. Make sure your graphs have a meaningful title, legend, and axis labels. Then see if the numbers fit the expected behavior (*e.g.*, n^2 or $n \log n$).
- Include your graphs and your analysis of them in your writeup. The raw numbers are useful, but should be submitted separately as part of a big table or spreadsheet, or possibly as an appendix. There are likely to be too many numbers for anyone to want to look at them all.

Submission

Before 6:00 PM, Wednesday, November 6, 2013, submit your Java program(s) and writeup for grading. Do this through Submission Box at `http://sb.teresco.org` under assignment "Sorting". Since SubmissionBox can only accept one file per assignment, please package up your files in a "zip" or similar archival format first.

Grading

This assignment is graded out of 50 points, broken down as follows:

Grading Breakdown	
General empirical analysis framework	5 points
Java code for specific sorting algorithms	15 points
Java code style, documentation, and formatting	5 points
Theoretical expectations	5 points
Presentation and analysis of timing results	20 points