Bill Manaris : Spring 2016 / CSCI 230 Homework 5

Assigned Date: Thursday, Apr. 14, 2016
Due Date: Wednesday, Apr. 27, 2016
Due Time: Section 2 at 8:25am / Section 1 at 11:15am

Last modified on April 25, 2016, at 08:23 PM (see updates)

Purpose

This assignment focuses on:

- implementing and evaluating three sort algorithms in Java

This is a **solo** assignment. **You may discuss the assignment only with the TA or the instructor.**

Learning Outcomes

- Analyze algorithms using asymptotic analysis.
- Distinguish between various sorting algorithms in terms of how they work and their efficiency in specific problem situations.
- Apply written communication skills to produce a report on a topic of data structures and algorithms.

Assignment

Create a Java program, `compareSorts.java`, which implements insertion sort, mergesort, and quicksort.

Run several comparisons of the above algorithms, as implemented, and collect your findings. While doing that,

- explore the list size below which insertion sort becomes faster than quicksort, and (possibly) mergesort; then

- modify the quicksort, and (possibly) the mergesort algorithms to use insertion sort for list sizes smaller than the threshold list size.
    - **Note:** If mergesort indeed can benefit by using insertion sort for smaller lists, chances are it will probably have a **different** threshold list size compared to quicksort. Keep that in mind.

Write a short report summarizing your findings.

Code

Your program should have the following sorting methods (as provided in class):

1. `public static void insertionSort(int [] a)`

2. `public static void mergesort(int [] a)`, which calls:
    - `private static void mergesort(int [] a, int [] temp, int left, int right)`, which recursively sorts list `a`.

3. `public static void quicksort(int [] a)`, which calls:
    - `private static void quicksort(int [] a, int left, int right)`, which recursively sorts list `a`.
    - Additional helper methods include:
        - `private static int getPivot(int [] a, int left, int right)`, which returns the index of the median of three values (left, center, and middle)

- `private static void swap(int [] a, int i, int j)`, which swaps values at the two indices `i` and `j`.

Input

The program should accept several lines of input, as follows:

The first line contains an `int` (e.g., 1), which specifies which method to run (e.g., the first method above).
- If the input is negative (e.g., -1), your program should create a random list to be sorted (see below). The absolute value of the input still specifies which method to run.

The second line contains an `int` (e.g., 6), which specifies the size of the list to read in.
- If the first input was negative, your program should automatically create a list with this many random integers to be sorted. Each integer should range from 0 to 1000, inclusive.

The third line contains a list of `int`s, whose length was given in the previous line (e.g., the list contains 6 `int`s).
- If the first input was negative, your program should ignore the third line (if any).

You may assume that the input is formatted properly and is valid.

Output

On the first output line, your program should output the original list (all in one line).

On the second line, it should output the sorted list (again, in one line).

The third line should consist of this message "Sorting by *method* took *x* milliseconds...", where *method* is the name of the method (e.g., "mergesort"), and *x* is a double number.

Hint: To time your code you may use the following:

```
long startTime = System.nanoTime();   // start timing

methodToTime();

long endTime = System.nanoTime();    // end timing
double runtimeInMilliseconds = (double)(endTime - startTime) / 1000000.0;
```

I/O Sample 1

For example, given this input:

```
2
6
24 7 54 50 22 20
```

the output of your program should be (of course, the timing may differ):

```
24 7 54 50 22 20
7 20 22 24 50 54
Sorting by mergesort took 0.01 milliseconds...
```

I/O Sample 2

For example, given this input:

```
-2
10
```

the output of your program should be something like this (the actual numbers will most likely be different, as they are randomly generated to be sorted - also, the timing may differ):

```
809 619 419 389 620 197 804 597 722 889
197 389 419 597 619 620 722 804 809 889
Sorting by mergesort took 0.013 milliseconds...
```

Documentation

Follow the Golden Rule of Style: "A program should be as easy for a human being to read and understand as it is for a computer to execute." [1]

In general, you should comment any variable, obscure statement, block of code, method, and class you create.

Your comments should express **why** something is being done, as opposed to **how** – the how is shown by the code.

Also see the course documentation standards in homework #1.

NOTE: You may use Javadoc formatting, if you prefer. If so, do it consistently.

Report

The report should be formatted using the ACM Proceedings format.

Provide a meaningful title, e.g., "Comparison of ... Employing Random Data"

It should consist of the following sections:

- **Introduction** - provide a short introduction of what this report is about.
- **Methodology** - describe how the results were generated, i.e., how many runs did you do, and with what list sizes. Try to be very systematic and thorough (why not? - this step should be pretty easy, as you already have the code to support your study).
- **Results** - provide:
    - a table summarizing test runs (for example, see Fig. 2.2 in textbook).
    - a graph of input size (x-axis) vs. time (y-axis), properly formatted labeled (for this, you may use Excel, Google Drive, etc.). The graph should contain different curves, one per sorting algorithm. For example, see Fig. 2.4 in textbook.
    - a second graph (magnification of the first) which demonstrates the threshold list sizes for quicksort, and possibly mergesort. For example, see Fig. 2.3 in textbook.
    - the graphs should be surrounded by explanatory text describing how they were generated and what they show, and have meaningful captions. Again, for example, see explanatory text provided in textbook for Figs. 2.2, 2.3, and 2.4.
- **Discussion** - provide a summary of what was tried and what was learned. In particular, if you find that mergesort also benefits from a threshold list size, this is the place to (profusely) mention it. This

is your BIG finding (to the best of your knowledge).

- **References** - provide any references used (e.g., textbook, etc.)

Submissions

Submit via OAKS as follows:

1. Create a folder named 'First.Last', where 'First' is your first name, and 'Last' is your last name.
2. Place your Java program inside that folder.
3. Place your report (as a PDF file) inside that folder also. Name it, "Report.pdf".
4. Compress the folder as a .zip file.
5. Upload on OAKS.

Grading

Your grade will be based on how well you followed the above instructions, and the depth/quality of your work.

Reference

1. Cooper, D. and Clancy, M. (1985) "Oh! Pascal", 2nd ed., W.W. Norton & Company, New York, p. 42.