Computing and Algorithms III: Program 2

CS-203 **Fall 2018**

Assignment Weight: 1.0

Functional Requirements

You are to implement three sorting algorithms and perform empirical tests and analysis on various data sizes. You must program in the JAVA language using good style and documentation. Specifically, you will implement the following algorithms (as presented in Levitin) which will sort arrays of integers in increasing order: Insertion Sort, Merge Sort and Quick Sort. So that your functions can be tested for correctness, you must implement each sort such that functions use the following names:

- InsertionSort(int [ ] sort array, int size)
- MergeSort(int [ ] sort array, int size)
- QuickSort(int [ ] sort array, int size)

Implementations should be placed in source files with exactly the following names: InsertionSort.java, MergeSort.java, and QuickSort.java respectively. Note: you are passing an array pointer to each sort function, so you will need to send a COPY of each array to each sort function to ensure that these three different sorts are acting on the same array.

In addition, you will also create a file (name it main.java) that will do some empirical tests on your algorithms. First, you will analyze each algorithm on randomly generated instances of various array sizes. You can choose your own test sample sizes if you wish but be sure to justify your choices in your analysis. A good suggestion would be $n = 1000$, $10,000$, $20,000$, ..., $50,000$. See the man pages on `srand()` and `rand()` or the Random class in Java to populate your arrays with random integers. In addition to random data samples, you may want to see how the algorithms behave on other types of data sets already sorted, completely unsorted, almost sorted, small integers, large variations etc.

To compare and contrast the three different algorithms you will need to keep track of how many *comparisons*, *swaps*, and *CPU time* it takes for each instance to be sorted by each algorithm, plus any other characteristics you can think of. You may want to run each instance of n a number of times and take an average, the choice is yours. However, you must detail and justify the data sets you use. To report the final results, the main program should output a nice looking table. The following is merely a suggestion. As long as it is readable and conveys all the necessary information you may design your own table.

In computing the CPU time, you should use `System.nanoTime()`. This method returns a `long` value representing the current time, relative to some unknown starting value. Successive calls to `System.nanoTime()` allow one to measure the elapsed time between calls, *e.g.*:

n	**1000**	**10000**	...	**50,00**
InsertionSort	comps: 4332.5	comps: xx	...	comps: xx
	swaps: 324.3	swaps: xx	...	swaps: xx
	time: xx	time: xx	...	time: xx
MergeSort	comps: 4332.5	comps: xx	...	comps: xx
	swaps: 324.3	swaps: xx	...	swaps: xx
	time: xx	time: xx	...	time: xx
QuickSort	comps: 4332.5	comps: xx	...	comps: xx
	swaps: 324.3	swaps: xx	...	swaps: xx
	time: xx	time: xx	...	time: xx

```
long startTime = System.nanoTime();
doTheAlgorithm();
long endTime = System.nanoTime();
long elapsedTime = endTime - startTime;
```

Your timing should specifically exclude the time required to initialize the program (*i.e.* reading the input file into your internal data structures) as well as the final output tasks (*i.e.* printing out the final results). That is, your timing should only measure the time taken to find the solution.

Your program should use "good style". See the separate handout on style requirements for CS-203 programs. In particular, note that you should describe the algorithms you implement in sufficient detail to demonstrate your deep understanding of the algorithms in question.

Additional Requirements

Additionally, you should create the following ordinary text files:

- README: Information on how to compile and run your program.
- ANALYSIS: An analysis of your program, including the table as above.

  Your analysis should reflect your understanding and critical analysis of the algorithms as well as demonstrate that you gave some thought to the assignment beyond mindlessly coding the algorithms. Discuss how each algorithm works. For each of the criteria, (comparisons, swaps, CPU time) compare and contrast your empirical results with each algorithms asymptotic characterization, with each other, and with regard to increasing data set size. On what types of data sets would each algorithm perform well (already in order, out of order, etc.)? Which criteria are more relevant and in what context? Are there any other criteria that could be explored? Did the data match the theoretical asymptotic behavior? How and why or why not? Were you able to approximate the multiplicative constant for each algorithm? Discuss any modifications that you tried to the algorithms and justify why you did so. Discuss anything else that you feel is relevant, pitfalls that you faced, etc.

Submitting Your Program

Before 11:59:59 p.m., Thursday, 8 November 2018 (6th Thursday), you must upload a zip archive to the course Blackboard assignment for Programming Assignment 2. This zip archive must contain all source code files for your program, as well as the README and ANALYSIS files.

Explain Your Program

You must explain your program to the instructor before 5:00 p.m., Thursday, 15 November 2018 (7th Thursday), either in the lab or in the office hour. Highly encourage you to explain your program before the submission deadline, which is 11:59:59 p.m., Thursday, 8 November 2018 (6th Thursday). Then, you can modify your program if the instructor finds problems.