Joanna Klukowska
joannakl@cs.nyu.edu

Programming Project 3: Benchmarking Sort Algorithms

Due date: March 17, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission.

You are responsible for every line in your program: you need to know what it does and why.

If you are considering use of any Java API classes that are not discussed in CSCI101 curriculum and have not been covered yet in CSCI102, ask your instructor first. The objective of these projects is to practice the material that has been discussed in class and the solutions are not always the most efficient ones.

In this assignment you will evaluate relative performance of different sorting algorithms. We discussed them all in class and several different versions of code are in the textbook, OpenDSA website and other resources that you are using. Your job is to provide their implementation so that they can be incorporated in the provided benchmark program for performance evaluation.

The program that you write has to be command line based (no graphical interface) and it should not be interactive (do not ask user for anything).

Objectives

The goal of this project is for you to master (or at least get practice on) the following tasks:

- developing and writing recursive algorithms,
- developing and writing sort methods,
- working with existing code,
- implementing generic methods,
- analyzing data produced by the programs,
- validating program correctness based on data results,
- improving code efficiency based on data results.

Problem Description

We spent some time discussing performance of the algorithms, but in practice what one often wants to test is how a given implementation performs on a given machine. You will write code for several implementations of the sort algorithms and evaluate their performance.

Implementation Details

You are given a set of three source code files.

`SortTester` This class is the actual benchmark program that runs all the test. You should not be modifying anything in the class. (You may while you are developing and testing your code, but the submitted version should be unchanged). This is the program that displays the data that you need to collect and analyze.

Joanna Klukowska
joannakl@cs.nyu.edu

`Sorter` This is an interface that you have to implement for your implementations of merge sort, quick sort and selection sort. For each of the three algorithms you have to provide a class that implements the Sorter interface. In order to work with `SortTester`, your classes need to be called `MergeSort`, `QuickSort`, and `SelectionSort`.

`InsertionSort` This is a class implementing the `Sorter` interface. It implements the insertion sort algorithm. You do not need to do anything with this class. It is provided as a guide for how to implement your other classes.

Sorting Algorithms

You should implement three sorting algorithms: selection sort, merge sort and quick sort. You must implement them using generic methods. They must be implemented as classes that implement the Sorter interface mentioned above.

Report

You need to write a short report (1-2 pages of narrative + pages containing graphs and tables) based on the run of the program that produces the timing results for all implemented algorithm. Combine the numerical results into a table and produce graphs that demonstrate how the time changes as a function of number of elements. The template at https://docs.google.com/spreadsheets/d/14nEYcG5bto6C5eIXt6j96ekyJAR62Px1nxXTtwldi34/edit?usp=sharing should be used to generate the tables and graphs (just fill in your timing results and the graphs will get created automatically).

Your report should include the tables and graphs from the above spreadsheet (you can just copy and paste, or export it as PDF and attach to your document). It should also discuss what you observed and if the results were as expected or not.

You should then try to "improve" your implementation of merge sort and/or quick sort. In your report you should specify what changes you made and how they affected the performance results (you may regenerate the graphs if you wish). Changes may sometimes decrease the performance - that's ok, as long as they are reasonable. You should try to explain why a particular change affected the time performance in a particular way.

Report requirements:

- it must contain your name
- it must be submitted as a single PDF document (if you do not know how to create one, ask ahead of time)
- it must contain the data (graphs and tables) for your initial implementation (before "improvements" are introduced)
- it must contain discussion of the initial results
- it must contain discussion of the "improvements" to merge sort and/or quick sort (the data for this part is optional)

You should look at your data carefully and decide what it says about correctness of your implementation. For example, if the selection sort is the fastest sort method, there is a problem with your implementation of the other two methods!

Working on This Assignment

You should start right away! The code itself is not hard to develop, but you may want to spend some time optimizing and fine-tuning you implementations. You also need to complete your code in time to prepare the report and add improvements.

Grading

40 points - completeness and correctness of the report

15 points - code documentation (using Javadoc and inline comments)

45 points - development and correctness of sorting methods (the three classes that you need to implement)

Joanna Klukowska
joannakl@cs.nyu.edu

How and What to Submit

Your should submit all source code files (the ones with .java extensions only) in a single **zip** file to NYU Classes. Make sure that you submit the files that are posted with this assignment as well as the files that you create. Make sure that the `RUN_ALL` in `SortTester` is set to false. You should have at least 5 different files (classes/interface for `SortTester`, `Sorter`, `MergeSort`, `QuickSort`, `SelectionSort`).

The zip file should also contain your report in PDF format.

Please, do not double zip the files (i.e. do not create a zip file with the source code and then zip it again with the PDF).

Once you submit the program make sure that you receive a confirmation email from NYU Classes. Read through it to double check that correct files were uploaded. Keep this email as your proof of submission until you receive your grade for the assignment.