CSU22012 Data Structure and Algorithms II: Assignment 1

In this assignment you will implement a number of common sort algorithms and compare their performance for different input files. You will also write JUnit tests to test your code.

Total points for this assignment: 200 (100 automatic, 100 marked by demonstrators)

The following will be marked:

1. Correctness of your results (eg items are sorted), JUnit tests, and test code coverage – automatic mark through web-cat, 100 points
2. Running time analysis – marked by demonstrators, 100 points

Submission and automatic marking is through https://webcat.scss.tcd.ie/cs2012/WebObjects/Web-CAT.woa. **Submission of final version both through Web-CAT and Blackboard.**

Deadline: Friday February 25th 5pm.

You have an option to penalty-free submit assignment until Sunday evening (midnight), with two conditions: (1) your first attempt must be submitted before the official deadline (or normal late penalty will apply), (2) there will be no blackboard/email/demonstrator/TA/lecturer support for assignment (or web-cat or logins or any other issues) past the official deadline.

Assignments submitted later than Sunday will be deducted 40 points (20% of the overall mark) per day.

Please submit only SortComparison.java and SortComparisonTests.java. Do not submit input files.

Assignment specification

1. Implementation

Download SortComparison.java file.

Write a java class SortComparison in SortComparison.java file (please do not use custom packages as web-cat will give an error) which should implement the following methods

- static double[] insertionSort (double a[]);
- static double[] selectionSort (double a[]);
- static double[] quickSort (double a[]);
- static double[] mergeSortRecursive (double a[]);
- static double[] mergeSortIterative (double a[]);

In each of the methods parameter a[] is an unsorted array of doubles. Each method should sort the elements in ascending order and return a sorted array. Each method should implement a different sorting algorithm, specified in method name. Note that for some of the algorithms you will need to add additional methods apart from the ones listed above. It is up to you whether to implement basic versions of the algorithms, or add additional improvements as discussed in lectures.

2. Testing

Download SortComparisonTest.java file.

Write a java class SortComparisonTest in SortComparisonTest.java file, which should implement JUnit tests for SortComparison.

Your goal is to write enough tests so that:

- Each method in SortComparison.java is tested at least once,
- Each decision (that is, every branch of if-then-else, for, and other kinds of choices) in SortComparison.java is tested at least once,
- Each line of code in SortComparison.java is executed at least once from the tests.

The submission server will analyse your tests and code to determine if the above criteria have been satisfied.

3. Algorithm performance comparison

Create a main method in SortComparisonTest that runs all the experiments on SortComparison described below and prints the time in milliseconds that each method execution took. This method will not run on the submission server, but you should run it locally on your computers. You need to record the results in a comment at the top of your SortComparisonTest file.

Your experiments in this section should be run from within the provided main method. Do not run these experiments from within a jUnit test.

The following input files are available for download from Blackboard:

- numbers1000.txt – contains 1000 random decimal numbers, one per line
- numbers1000Duplicates.txt – contains 1000 random decimal numbers, one per line, but those 1000 consist of only up to 100 unique ones
- numbersNearlyOrdered1000.txt – contains 1000 decimal numbers, one per line, where most of the numbers are in correct ascending order, with approx. ~6% of the numbers out of place
- numbersReverse1000.txt – contains 1000 decimal numbers, one per line, sorted in reverse (ie descending) order
- numbersSorted1000.txt – contains 1000 decimal numbers, one per line, sorted in ascending order
- numbers10000.txt – contains 10000 decimal numbers, one per line, in random order

In the comment at the top of your SortComparisonTest record the time it took to execute each of 5 methods implementing 5 different sorting algorithms, for each of 6 different input files, containing different size and type of input. Run each experiment 3 times and record the average running time. Your results should be displayed in a format that enables easy comparison per algorithm and per data type, for example, as in the table below.

	Insert	Selection	Quick	Merge Rec	Merge It
1000 random					
1000 few unique					
1000 nearly ordered					

1000 reverse order					
1000 sorted					
10000 random					

Also, under the table, please answer the following questions:

a. Which of these sorting algorithms does the order of input have an impact on? Why?
b. Which algorithm has the biggest difference between the best and worst performance, based on the type of input, for the input of size 1000? Why?
c. Which algorithm has the best/worst scalability, i.e., the difference in performance time based on the input size? Please consider only input files with random order for this answer.
d. Did you observe any difference between iterative and recursive implementations of merge sort?
e. Which algorithm is the fastest for each of the 7 input files?

For fun:

This part will not be marked, but if you are curious you could also try the following:

1. Add counters to sort methods that count the number of value comparisons, and value swaps for each of the sort algorithms, to compare the numbers of each per algorithm, to see where the performances gains come from

2. Implement a multi-threaded version of merge sort and compare its performance to single-threaded one.

Appendix: reminder of general assignment instructions from Semester 1

Please see a walkthrough on how to submit an assignment on Web-CAT.

When you upload code for an assignment to the submission server, it compiles it and runs your JUnit tests, giving you back an automatic score. This score is part of your marks for this assignment; the other part is given manually by the teaching staff. You can improve and reupload your code to improve your score. The only limit is the submission deadline.

For security reasons the submission server is only accessible from the campus network. If you want to access it from home you need to connect to the campus network via a Virtual Private Network (VPN) with your SCSS account. Instructions.

Students are allowed to discuss assignments but not to share code! Sharing code will result in reduced marks for all students involved and the consequences described in the College rules. If you discuss an assignment with fellow students then you must write the names of the students in your submission. All students must complete the College's online seminar about plagiarism before submitting any assignment.