Koç University
COMP202
Data Structures and Algorithms
Assignment 5

Instructor: Barış Akgün
Responsible TA: Ali Tugrul Balci
Due Date: May 20 2018, 23:59
Submission Trough: Blackboard
Relevant Programming Session: TBD

This programming assignment will test your knowledge and your implementation abilities of what you have learned in the Sorting parts of the class.

This homework must be completed individually. Discussion about algorithms and data structures are allowed but group work is not. Any academic dishonesty, which includes taking someone else's code or having another person doing the assignment for you will not be tolerated. **By submitting your assignment, you agree to abide by the Koç University codes of conduct.**

Description

This assignment requires you to implement various sorting algorithms. These algorithms include insertion sort, heap-sort, quick-sort, merge-sort and counting sort. In addition, you are given the chance to submit another sorting algorithm to enter a sorting competition for bonus points.

The files provided to you have comments that will help you with implementation. In addition, keep the slides handy as they include the pseudo-codes for the algorithms. The file descriptions are below. Bold ones are the ones you need to modify and they are placed under the *code* folder, with the rest under *given*.

- *AbstractArraySort.java*: This file describes the abstract sorting class which the other sorting algorithms extend. It defines the `sort` method which is the primary function you should overwrite. It also has the `swap` and `compare` methods that you need to call in the concrete classes whereever applicable. Hint; not all algorithms utilize swap. It also has other utility methods which might be of use in debugging.
- *JavaArraySort.java*: A wrapper class for the Java's default sorting method. You can take a look if interested.
- ***InsertionSort.java***: Implement the insertion sort algorithm in this file. Remember to use the `swap` and `compare` methods from the abstract parent class whereever applicable.
- ***HeapSort.java***: Implement the heap-sort algorithm in this file. Remember to use the `swap` and `compare` methods from the abstract parent class whereever applicable. Make sure to fill out the `heapify` method which will also be graded.
- ***QuickSort.java***: Implement the quick-sort algorithm in this file. Remember to use the `swap` and `compare` methods from the abstract parent class whereever applicable. Make sure to fill out the `partition` method which will also be graded. You have two options for this, either version which returns an `indexPair` object to return two indices or comment this out and uncomment the version which returns an integer.
- ***MergeSort.java***: Implement the merge-sort algorithm in this file. Remember to use the `swap` and `compare` methods from the abstract parent class whereever applicable. Make sure to fill out the `merge` method which will also be graded.

- ***CountingSort.java***: Implement the counting sort algorithm in this file. Note that you need to do this only for integers. You do not have to use any function from its parent in this class.
- *SortDebug.java*: This file has a smaller main function which you can use to debug individual sorting algorithms. We suggest that you test your implementations here first.
- *SortGrade.java*: This file grades your sorting implementations with different data distributions and data sizes. It checks aforementioned methods for certain algorithm and tests whether your implementations sort the data correctly. In addition to integers, doubles and strings are also utilizied. The code also checks for number of swaps and compares for three algorithms. Small implementation details might result in slightly different numbers and we might update the autograder to reflect this so do not spend too much time on this and come back to it after you are getting an 88.
- *DataGenerator.java*: This file has the code to generate data to test your implementations. Take a look at it if interested but you do not need to worry about it.

Bonus: Sorting Spree!

This homework includes a bonus component where your sorting algorithm will compete with the rest of the class to receive a bonus up to 30 points. Your bonus grade will depend on how well you do in the benchmarking as compared to other submissions. The exact grading scheme will be announced later. The files you need to look at for this part are below. You are free to submit any one of the implementations you have already done but we suggest to try at least one of the hybrid algorithms we have seen.

- ***ContestEntrySort.java***: This file should have your competition entry sorting algorithm. You do not need to use anything from the parent class apart from overriding the `sort` method. You are not allowed to use an existing sorting library such as Java's own method.
- *SortBenchmark.java*: This file benchmarks the given algorithms. We are going to use something similar but will also add doubles and strings. Your score will depend on a weighted average of your sorting times, smaller the better. If you are interested in the bonus, go over this file.
- *benchmarkLog.java*: This file stores the benchmark performances and calculates statistics.

Grading

Your assignment will be graded through an autograder. Make sure to implement the code as instructed, use the same variable and method names. A version of the autograder is eleased to you. Our version will more or less be similar, potentially including more tests.

Run the main program in the *SortGrade.java* to get the autograder output and your grade.

In case the autograder fails or gives you 0 when you think you should get more credit, do not panic. Let us know. We can go over everything even after your submission. Make sure that your code compiles!

Submission

You are going to submit a compressed archive through the blackboard site. The file should extract to a folder with your student ID without the leading zeros. This folder should only contain files that were in **boldface** in the previous section. Other files will be deleted and/or overwritten.

Important: Make sure to download your submission to make sure it is not corrupted and it has your latest code. You are only going to be graded by your blackboard submission.

Submission Instructions

- You are going to submit a compressed archive through the blackboard site. The file can have *zip*, *tar*, *rar*, *tar.gz* or *7z* format.
- This compressed file should extract to a folder with your student identification number with the two leading zeros removed which should have 5 digits. Multiple folders (apart from operating system ones such as MACOSX or DS Store) greatly slows us down and as such will result in penalties

- Code that does not compile will be penalized and may receive no credits.
- Do not trust the way that your operating system extracts your file. They will mostly put the contents inside a folder with the same name as the compressed file. We are going to call a program (based on your file extension) from the command line. The safest way is to put everyting inside a folder with your ID, then compress the folder and give it your ID as its name.
- One advice is after creating the compressed file, move it to your desktop and extract it. Then check if all the above criteria is met.
- Once you are sure about your assignment and the compressed file, submit it through Blackboard.
- After you submit your code, download it and check if it the one you intended to submit.
- **DO NOT SUBMIT CODE THAT DOES NOT TERMINATE OR THAT BLOWS UP THE MEMORY.**

Let us know if you need any help with setting up your compressed file. This is very important. We will put all of your compressed files into a folder and run multiple scripts to extract, cleanup, grade and do plagiarism checking. If you do not follow the above instructions, then scripts might fail. This will lead you to get a 0. Such structured submissions greatly help manual grading as well.