Koç University
COMP202
Data Structures and Algorithms
Assignment 2

Instructor: Deniz Yuret

Due Date: Mar 26 2024, 23:59

Submission Trough: Blackboard

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. **Coming up with the same algorithm and talking about the ways of implementation leads to very similar code which is treated as plagiarism!** 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.

The files provided to you have comments that will help you with implementation.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 wherever 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 Java's default sorting method. You can take a look if you are interested.

• ***InsertionSort.java***: Implement the insertion sort algorithm in this file. Remember to use the swap and compare methods from the abstract parent class wherever applicable.

• ***HeapSort.java***: Implement the heap-sort algorithm in this file. Remember to use the swap and compare methods from the abstract parent class wherever 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 wherever 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 wherever 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 algorithms and tests whether your implementations sort the data correctly. In addition to integers, doubles and strings are also utilized. The code also checks for the number of swaps and compares for three algorithms. Small implementation details might result in slightly different numbers. If you are getting sorted elements but swaps and compares do not match, make sure to finish the rest of the homework and come back to the issue.

• *DataGenerator.java*: This file has the code to generate data to test your implementations. Take a look at it if you are interested but you do not need to worry about it.

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 released 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, which you should not have modified anyways, will be deleted and replaced with default versions.

We download all the submissions from blackboard together and put them in a folder. We then run multiple scripts to extract, cleanup and grade your codes. Your codes are compiled from the command line, no IDE is used. If you do not follow the instructions given below, then scripts might fail. This will lead you to get a lower grade than what the autograder suggests. More importantly, your code might not compile, which will result in a grade of 0. Make sure to follow these instructions:

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

• Make sure to remove all the unused imports and that you have the original package names. What compiles on your machine may not compile on ours due to simple import errors. Code that does not compile will receive a grade of 0.

• You are going to submit a compressed archive through the blackboard site. The file can have *zip*, *tar*, *rar*, *tar.gz* or *7z* format.

• The compressed file should have all the files that you need to modify. If anything is missing, we automatically copy over the default versions.

• The compressed file should not contain another compressed file.

• 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 is the one you intended to submit.

• Do not submit code that does not terminate or that blows up the memory.