Lab 10: Let's Sort This Out

Pre-Lab Notes

Lab submissions will **not be accepted late** – they must be submitted by **midnight the day after they're assigned** (to account for time zone issues). Some latitude might be allowed (I won't disqualify a submission that's mere minutes late!) but that's purely discretionary.

Labs are treated differently from assignments – you're encouraged to work with fellow students, ask questions of the TA during your lab section, and generally collaborate to complete your work. You should still submit your own work, **including citing with whom you collaborated in your submission**, but using answers developed by others will not be treated as plagiarism.

Nevertheless, you are still encouraged to make a good faith effort to complete your labwork, in order to exercise your understanding of the topics it covers. Lab submissions are marked purely 1 or 0 by whether the TA believes you at least made an honest attempt, even if you didn't succeed, so the value is in what the attempt teaches you rather than simply having the right answer.

1. Sorting

Sorting algorithms are a key component of Computer Science theory and practice; a surprisingly large chunk of the work computers do boils down to putting reams of data in the right order. As such, knowing the best techniques for doing this efficiently is also a large part of being an effective programmer.

I say techniques, because there's more than one way to sort a set of data. Some only work for certain kinds of input data, or need to be paired with particular data structures, but there's still a lot of situations where more than one sorting algorithm would work. In those cases, it's down to the programer to know how the strengths and weaknesses of different sorting algorithms depending on the data they expect to see.

2. Outline

Sorting efficiency is another case where sometimes seeing the difference for yourself is easier than simply absorbing rules and theory. That's why our goal today will be to implement two different sorting methods and then find two different data sets who each run faster with a different method. Hopefully, this'll help develop your intuitive understanding of what makes these algorithms tick and therefore when to use them.

Remember Insertion-Sort? Yeah, blast from the past (and unit 4 on arrays, in case you're looking for it now). Your first task today is to implement both it and Merge-Sort for sorting basic arrays of ints from smallest to largest. You don't need to do anything fancy, just write a class with two sorting functions, each of which either sorts a given array of ints or returns a sorted array of ints (your choice).

Once you have your two sorting functions, you'll also need a way to keep track of how much work each of them is doing to sort the array. The details of exactly how many primitive operations are involved may vary depending on your implementation, but there's two sources of work tied to the number of inputs that are very particular to these two methods – **inversions** for Insertion-Sort, and **comparisons** for Merge-Sort.

Inversions are how many ranks back Insertion Sort must move the currently-considered integer to find its place in the order – typically, the more out-of-order an array is (the more inverted), the more of these it will take to set it right. Comparisons for Merge-Sort are simply how many times the subsequences S1 and S2 must be compared to each other as they unload their sorted contents into S, before one empties out and the rest of the other can just be appended on to the end. Add up how many of either of these you need to perform for either sorting method and print the results.

Now that you have two functions based on two different sorting algorithms who can report their most input-sensitive sources of work, the last step is to come up with two data sets, each of which gets you a better result (i.e. fewer inversions or comparisons) with a different function. Again, you don't need anything fancy here, try coming up with two arrays of ten integers and thinking about how they could be different from one another to make the work of Insertion-Sort and Merge-Sort easier or harder.

3. Deliverables

All lab submissions will be done through CourSys at https://courses.cs.sfu.ca/. For this lab, that should include:

1. A .zip archive of your code, organized into a single Java project. To ensure compatibility, it's recommended to use IntelliJ while developing your code, then using the export function under File -> Export -> Project To .zip File.

Be sure you tidy up your code before submitting! Check the Java code style guide posted on the course website, and do your best to provide helpful comments.