Assignment 3
Due: Oct 12, 2023 at 11:59PM

1 Assignment 3 Specification

1.1 Goals

1. Practice converting pseudocode to Java
2. Implement and test two sorting algorithms

1.2 Implementation

The skeleton code at `https://classroom.github.com/a/Jz3Li82v` is the start of a program to sort latitude/longitude coordinates based on distance from a starting point. An additional class, **Coord**, is included in the code. It holds information about the coordinate, including the value that you should sort by: the distance field. The distances are already calculated in the **Coord** array by the time a sorting algorithm is called. You can access those distances with the `getDist` method.

In order to sort the input array, change its order in-place. Instead of an explicit return value, the updated order of input array should be the result of the sorting method calls.

There are two provided algorithms:

insertionSort Uses an $O(n^2)$ sorting algorithm to sort the array.

systemSort Calls Java's built-in sorting algorithm, which is a variant on mergesort.

For this assignment, you will implement two sorting algorithms: *quickSort* and *randQuickSort*. You should implement quicksort as it is presented in the lecture slides or book: when partitioning, pick the left or right element in each subarray as the pivot. For randomized quicksort, use the Java **Random** class to generate a valid index in the subarray for each partition call and use that value as the randomly chosen pivot. Look up Java's documentation to figure out how to use **Random** and adapt its behavior to the index ranges you want. You can write additional private methods in **A3.java** to help with your solution. Do not edit anything in the **Coord.java** file.

1.3 Testing

For this assignment, full JUnit tests are provided in the **edu.wit.cs.comp2350.tests** package. You can run these tests to see if your code is performing correctly. Your assignment grade will be based entirely on these tests. In future assignments I will not provide all of the grading tests so you will have to be thoughtful with testing your code. If tests are timing out, that means that you have a lot of extra operations that are slowing down your algorithm – most likely from extra memory allocation. If you get a `StackOverflowError` for any of the tests, that means that you are allocating an excessive amount of memory in each quicksort call.

A **ChartMaker** class is also included, which will create a chart of the runtimes of each algorithm with different input sizes. You can use this chart to verify that the time complexities of the algorithms are what you expect. Your randomized quicksort should be competetive with the system's sorting speed!

2 Grading

Quicksort: 70%

Randomized quicksort: 30%