CS1C Winter 2024 Assignment 7
Possible Approach to Creating the Data Table

The main objective of this assignment is to determine the recursion limit for each array size that produces the fastest sort time. The main challenge in this assignment is doing the data analysis, not necessarily the Java programming. There is some programming involved, but everyone in the class is more than capable of doing it. I'll go over, however, how I think it makes sense to structure the java part of the assignment.

To link your project to the provided repository, follow these steps:

1. Start by cloning the basically empty repository from GitHub. This will set up the project to be integrated with GitHub. **Don't** add your README.txt file until after the project and repository are linked. Then you can add your README.txt file and add it to git.
2. For each folder or file that you want to put under Git version control, right click on the folder and from the dropdown menu select Git and then +Add. You would want to do this for the resources folder, the src folder, and the docs folder (for Javadocs). Don't provide a .gitignore file since one is already in the repository. You will need to use File/Project Structure to tell the project which folder (src) has your source files.
3. From the git menu, select Commit, add your commit message and then select Commit and Push from the Commit button in the lower right hand corner of the Commit pop up window.

First, you'll need to copy FHsort.java from your course_examples to your src folder. I created a new Java class file as outlined in the instructions and just put all of my code in main(). You could also create a few methods to do the various pieces of work. Give some thought to how and when you will capture data. I used a .csv file since I like to use Excel and it can easily open and read a .csv file. I would suggest that a good format for the .csv file (or a .txt file) is to have the recursion limits across the first row (except for the first cell) and your array sizes down the first column (except for the first cell). In this way you will end up with a table where you have a cell for each array size and recursion limit (with the upper left corner cell empty). You will have to populate the file one row at a time, so you need to create the first row with the recursion limit values before you start testing your arrays.

Create an array of integers that represents the various array sizes you will test and how many you will have. You need at least 20 arrays starting with a size of 20,000 integers up to 10,000,000 or more. There are several ways to do this. You could just hard code the sizes into an array, that way you get to decide what sizes to test, and it's easy to change. A word of warning, larger arrays take much longer to run, so don't use a bunch of really large arrays. You could also auto-generate them starting with 20,000 and then increasing the size by some factor. Really your choice, but I think it's a good idea to create an array of your test array sizes ahead of time. Start out with just a couple of small array sizes to make sure everything is working.

The approach is to set up a series of nested loops. The outside loop will cycle through each array size, the middle loop will cycle through the recursion limits, 2 up to 300, (start with just a few to test your

code) and the inside loop will run quickSort(). Run quickSort() however many times you want, but at least 3 for each recursion limit. **Use a fresh new copy of your unsorted array before each quickSort().**

For each array size, be sure and create a new array of that size and populate it with random integers to fill the array. Don't create all your arrays first and populate them with random integers. This will take up a lot of needed memory. Next start the middle loop which will cycle through the recursion limits. Call FHsort.setRecusionLimit(). Then start the inside loop. You should create a new working array at the beginning of each inside loop iteration so that you start with an unsorted array for each run of quicksort. You could simply clone the original array. Start the clock, run quicksort, then capture the total running time. Decide how you want to store the time, and in what time interval, I used milliseconds, and after you finish the inside loop however many times you decided (but at least 3), calculate the time you will use for this array size and for this recursion limit, I just took the average of my 3 times.

Along the way you will be building your data file and will need to add the array size and then the running time for each recursion limit. When you are finished, or as you complete each array size, write or update the file to your resources folder. From there you can easily access it with a number of programs that can accept a .csv or .txt file, show the data in a grid form and create graphs.

Be prepared, the test could take hours to run. Try not to have anything else running. I print the current array size to the console so I can keep track of progress. The rest is up to you to look at your data and see what you think. Review the Assignment instructions to understand what you need to look at.