This assignment is due by **10pm on Monday October 23** and is worth 16 points.

1 Goals

The goal of this assignment is to study the performance characteristics of several sorting algorithms and to get practice and how to analyze algorithms.

2 Setup and Requirements

This is a partner assignment (unless you were not assigned a partner). You will be working with the same partner as the previous partner assignments. You may certainly discuss the assignment and how to get to your final solutions with your other classmates, but make sure to cite any sources used or classmates consulted!

In this assignment, you'll be doing a mixture of coding, designing timing experiments, and analyzing your results. You'll be thinking through what timing data makes sense to collect, and writing your reasoning in a short report. Your report should be typed, and must be a PDF. The easiest way to produce a PDF is to write your text in a text editor or a word processing program, then export or print it to PDF. On a Mac (such as the Macs in the lab), you can choose "Print" and then in the lower left corner there will be a "PDF" button. Click on that button and choose "Save as PDF". No format other than PDF is acceptable. If you are familiar with LATEX, the PDF created from that is also a good option.

3 Investigation 1: Quicksort and small lists

Typically, quicksort implementations switch to using a different sort for small arrays. This is because the overhead of the recursive calls and the constant time factors in quicksort dwarf the poorer asymptotic performance of the other sorting algorithm when n is small. The sorting algorithm that we'll use for these small arrays is insertion sort.

I've provided you with a version of quicksort code in Quicksort.java. Your job is to investigate how the performance of quicksort varies when this switch to insertion sort occurs. Write code in main function of this class that performs experiments varying when this switch to insertion sort occurs and collecting timing data. I've included an example of how to collect timing data. You may want to use one size of array to compare a broad range of possible switch points, and then use a larger array to gather more fine-grained data differentiating a smaller range of switch points.

The second section of your report for this assignment should report on your investigation of quicksort, relating the size of the data being sorted to quicksort's overall performance. Describe what data you collected and why it was appropriate, and present the data in a table or a graph. Explain any patterns you see in your data and any hypotheses you have for why these patterns occur.

4 ShellSort and Gap Distance

ShellSort performs a sequence of insertion sorts at varying "gaps", starting with something large and eventually ending with a gap of one. Any sequence that contains a gap of one will result in a correct sort, but different gap sequences vary significantly in their performance. A common gap sequence

involves creating a sequence where each element differs from the last by some multiplicative factor m. For example, if m is equal to 2, and you start with a gap of 14, then in the next iteration the gap size with be $14/2 = 7$. Then in the following iteration you should have a gap of $7/2 = 3.5$. However, you need to have integer gap sizes. So, to get the actual gap, you will take the ceiling of this number (that is you will round up). You'll find the ceil function (part of the java.lang.Math package) helpful. So the next gap is actually 4. Lastly, you never run with a gap of 2. If you calculate a gap of 2 , you should instead automatically change this to a gap of 1. So, for our example where we start with a gap of 14, we would have the following sequence of gaps: 14, 7, 4, 1.

I've provided you with a version of ShellSort code in Shellsort.java. Your job is to explore how changing the value of the mulitplicative factor m affects the performance of ShellSort. Specifically, you should try out gaps of 2, 2.25, and additional values of your choice to explore the pattern that arises as the gap changes. I'm asking you to try 2.25 as that's one value that's used in many actual implementations of ShellSort; this is known as Tokuda's sequence for its inventor Naoyuki Tokuda. Since 2.25 is a decimal, make sure you are taking the ceiling to get your gap values. Include your timing experiments in the main method of Shellsort.java.

In your report, describe what you find about how the performance of shellsort changes as the multiplicative factor used for the gap changes. For this section, you may have less clear hypotheses about why you see the performance you do and that's okay; do try to articulate what factors you believe may be leading to differences in timing based on what you know about shellsort and why its performance can be better than insertion sort.

5 Report Guidelines

Your report should describe the timing experiments you performed, the timing data that you collect, and answers to the questions posed in the two individual investigations. You should make sure to explain how the evidence you collected supports your claims, and explain your choices about what evidence to collect for each question. Your report should include tables and/or graphs to support your points. Bring the analytical skills you've learned elsewhere to bear on this assignment. I encourage you to think carefully about how to connect evidence and explanations. Imagine your report is intended to help someone make decisions related to these three investigations (e.g., when to switch to insertion sort for small lists) - what information do they need to know? Why do you think the timing data looks the way it does based on what you know about how the algorithms work? You should try to design your experiments in a way that will be convincing to the reader. How could you compensate for the fact that timing data is "noisy," with the same code sometimes taking a little longer or shorter to run? You and/or your partner are welcome to chat with me about your approach in office hours.

Your report should be a PDF file no longer than 5 pages, including any charts and graphs. Note that no format other than PDF is acceptable.

6 Other Tips and Suggestions

- You may change the starter code I gave you in any way you please, although you should ensure you don't break the correctness of any of the sorts.

- Make sure your timing code times only the thing you actually want to time - i.e., the sorting. It shouldn't include file loading or printing to the console, as both of these can take long and variable amounts of time that may swamp true differences in execution time.
- You can use many different array sizes to explore the questions in this assignment. Make sure the arrays you're choosing are big enough to show real differences. If all of your experiments take only 1 or 2 milliseconds, it will be very hard to understand how your variations are related to differences in timing. Instead, use an array that's big enough to take at least hundreds of milliseconds to sort.
- Think carefully about what evidence to collect, and explain your hypotheses about why you see the data that you do. Any time you make an assertion, think about how to support it with evidence from your timing or from the way the code works.
- If you're working with a partner, you should both contribute to all parts of the assignment - experimenting and writing things up. Your timing and experiment code should be done in paired programming style, and you should collaboratively decide what timing data to collect; either of you should be able to explain the results that you got from your timing experiments. You each may write drafts of parts of the writeup by yourselves, but you should edit each other's drafts and both of you should be aware of everything that's in the writeup; if you aren't able to explain why a particular section of the report includes the content it does and what the content means, then you have violated the collaboration guidelines for this assignment. If you're struggling with how to write a report like this, you might go to the Writing Center, located in the library.

7 Submission and Grading

Create a zip file of the following, to be handed in via Moodle:

- `Quicksort.java`, with main running your timing experiments
- `ShellSort.java`, with main running your timing experiments
- Your report. Remember: Your report should be a PDF file no longer than 5 pages including charts and graphs. You can turn any type of a file on a Mac (like the lab computers) into a PDF by selecting "Print..." and then choosing the "PDF" option in the lower right corner. Just changing the file extension does not change the type of a file.

Only one partner from each pair needs to submit the assignment. Specifically, put these files into a directory named `[your_last_name_your_partner's_last_name]HW9`, zip this directory, upload it to Moodle. For example, if my partner was Schiller my directory would be named `OesperSchillerHW9` and the resulting file would be `OesperSchillerHW9.zip`.

This assignment is worth 16 points. 12 points for the write-up and 4 points for your associated testing code.

Start early, ask lots of questions, and have fun! Layla, the lab assistants, and the prefect are all here to help you succeed - don't hesitate to ask for help if you're struggling![1]

[1] This assignment modified from one by Anna Rafferty. Thanks for sharing!