## Lab 7: Implementing Sorts

**Monday, 12 November 2018**

Included in your repo will be the same Timer code you used in the previous lab (remember winTimer is only for Windows machines, unixTimer is only for Mac/Linux machines). There is also a spreadsheet that you will use to log run times for comparison.

A large part of Computer Science is sorting collections of information efficiently. As you saw in the last lab, many problems can be tackled more efficiently when the problem set is sorted for you, and sometimes impossible otherwise. In this lab you will implement several different sorting algorithms and compare the run times of each. These are VERY COMMON QUESTIONS FOR INTERVIEWS! Understanding how these work will impress employers and could help you land that dream job.

Your job for this lab is to implement several different sorting algorithms and compare the run times for each. The sorting algorithms you will implement are **bubble, insertion, selection, merge, and quick** **sort**. There are definitions for these algorithms all over the internet, but make sure your source is reliable and trustworthy (find something other than Stack Overflow, there is also very good pseudo-code explanations at rosettacode.org). You will perform tests on different sizes of collections and compare the run times to the Standard Template Library’s sort function.

Lab Assignment:

1. Create a brand new project, generate two new files using the following format Lab7-<UWyoname>.h and Lab7-<UWyoname>.cpp

2. Define in the header file 5 functions each implementing a different sorting algorithm, using function names that clearly describe which algorithm is being used.

3. Using the random number vector generator you created in the last lab, generate a vector for each sorting algorithm whose size takes the sorting algorithm around 4 seconds to run (use random numbers between 1-500 inclusive). You will now have 6 vectors whose sizes may or may not be the same size as each algorithm has a certain time complexity (don’t forget to test STL::sort).

4. Record the sizes of these vectors in the corresponding fields in the spreadsheet. The spreadsheet will then auto-populate the sizes for the next two vectors you need to create and test for each algorithm.

5. Generate vector of these sizes and test against your algorithms, recording the corresponding times in the spreadsheet.

6. Now analyze the runtimes you recorded and give an estimate of how long you believe it would take each algorithm to sort a vector containing two million elements.

**Turn In:**

Submit your code and spreadsheet through the Github Classroom repository. I should be able to see you generating each vector for each algorithm (you may comment these out so you’re not testing every algorithm every time you run your program).

The spreadsheet should contain your record of **sizes** needed for a 4.0 second base sort time, and the **times** for doubled and quadrupled sizes. It should also contain your **estimates** of the time required to sort two million numbers for the 5 selected combinations.

**Note:**

Remember that these times will vary between machines, processors, operating systems, and the weather(not really). If you are planning on developing on multiple machines that is fine, but run your final tests all on one machine and record those time from that one machine. Recording times generated from multiple machines will skew your results and make it difficult to observe exactly the differences in algorithms.