Lab 7: Implementing Sorts
Monday, 26 March 2018
Due: April 1 11:59pm (no joke!)

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) as well as header and cpp SortedSpeed files to use as a starting point. There is also a spreadsheet that you will use to log run times and calculations 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, 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. Import the given header and driver files into a new project.

2. Implement the sorting functions given in the header file.

3. Using the random number vector generator getNums(list,min,max) from Lab6(see header file), 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). Record the size. Then find and record the time required when the size is doubled. Record the ratio of the two times. [If you record your results in a spreadsheet, the spreadsheet can be used for the calculations of part 4.]

   For some of the sort algorithms the ratio will be near 2.0; for others the ratio will be near 4.0 or 8.0.

4. Using your results from part 3, select three algorithms, one for each of the three ratios. For a ratio:R, the time:T2, required to sort a sequence of size N2 can be estimated using the time:T1, required to sort a sequence of size N1 with the following equation.

*Edited from Dr. Tom Bailey's Sorting Speed Lab

$$T2 = T1 * \left(\frac{N2}{N1}\right)^{lg\,R}$$

For R = 2, lg R = 1; for R = 4, lg R = 2; for R = 8, lg R = 3.

For each of your three combinations, estimate the time required to sort a sequence of 2 million numbers.

Turn In:
Submit your code, spreadsheet, and a screenshot of your experiment output through the Github Classroom repository. In the README please include your name as well as what type of machine you used to run your experiments(see note below).

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.

Criteria:
All algorithms implemented: 5pts
Experiment timing results recorded in spreadsheet: 5pts
Correct timing ratio calculations recorded in spreadsheet: 5pts
Correct 2 Million run time estimates: 5pts
Total: 20pts

*Edited from Dr. Tom Bailey's Sorting Speed Lab