Computer Science 445
Analysis of Algorithms
Big project 1: Comparison of sorting algorithms
Due: Wednesday, Oct 11

In this project you will implement several sorting algorithms, measure their running times, and compare them with their theoretical complexity and with each other.

Copy and untar `/homeemp/tvandrun/pub/445/sortProj.tar`. The package `csci445.bigSortProj.driver` contains a driver class and an interface for the sorter classes which are in `csci445.bigSortProj.sorter`. The sorter classes are stubs which you must fill in:

`Bubble`	`Shell` (using insertion sort)
`Insertion`	`Bucket` (using insertion sort)
`Selection`	`Radix` (using selection sort)
`Merge`	`Stooge` (see CLRS exercise 7-3)
`Quick`	

(Shell sort isn't described in our textbook, and we looked at it only briefly in class; if you want more information on it, see
http://linux.wku.edu/ lamonml/algor/sort/shell.html,
http://en.wikipedia.org/wiki/Shell_sort,
http://www.nist.gov/dads/HTML/shellsort.html, or Don Knuth, *The Art of Computer Programming*, vol 3 pg 84).

Using the driver, measure how each sorting algorithm's running time (measured in milliseconds by the driver) and number of comparisons grow with the input size. Then, choose several input sizes and compare the running time and number of comparisons across sorting algorithms.

In addition do **one** of the following (altering the driver as necessary):

1. Add sorters for non-*in situ* versions of insertion and selection sorts, measuring them as before and comparing them with other sorting routines, especially their *in situ*.

2. Add sorters for list (instead of array) versions of bubble, insertion, selection, and merge, measuring them as before and comparing them with other sorting routines, especially their array versions.

3. Vary the auxiliary sorters for radix, bucket, and shell—use bubble, insertion, selection, and merge on each. Measure them as before and compare them with other sorting routines, especially how they compare as the auxiliary sorter varies.

4. Pick two of the sorting routines and change the driver to feed special case arrays into the sorters: larger or smaller ranges of numbers, partially sorted arrays, back sorted arrays, arrays with many repeated numbers, and other variations you can think of. Compare running times and comparisons as before, making an effort to identify cases where certain sorters obtain an advantage. This option requires less programming than the others but more creativity and insight.

Finally, write a report on your experiments and findings. Document your procedures carefully. Use MS Excel or another graphing tool to present data. Draw conclusions about what sorting algorithms are better and under what circumstances, as far as you have experimented. Compare your results with theoretical expectations and make conjectures about when experimental results and theoretical expectations differ. If you have chosen the third option, be sure to use those results when explaining differences.