INFS-519 - Assignment 1 – A Comparison of Sorts

Assignment Given: 9/14/2017
Assignment Due: 9/28/2017, 7:30 pm

In this assignment you are asked to write three sort routines in Java that sort various size numeric arrays and write a brief report (1-2 pages, no more please) analyzing and comparing the results. The routines implement two exchange sorts: Bubble Sort and Insertion Sort, and one divide-and-conquer sort. For the divide-and-conquer sort you may choose between Mergesort and Quicksort. The routines will be used to sort single column numeric input arrays of size 100, 1000, 10000, and 100000. The sort order for those input arrays will be random. In addition, an array of size 1000 will be provided that is "almost" in sort order (< 20 items out of place). That array is also to be sorted and included in the comparison report.

In coding the sort routines:

1. Write the code in Java
2. Use command line Java or your chosen IDE to write and debug the code
3. Write the Bubble sort from scratch – a specification for that sort is provided below
4. Feel free to use the code in Weiss as a template for the Insertion and Merge or Quick sorts – if you use it rather than writing your own, be sure to include commentary in the code as to its source (e.g., "Adapted from Weiss, Fig. 8.2")
5. Feel free to use the array structure to hold the input data and perform the sort rather than the generic class code and methods found in Weiss (but let me know if you prefer to use them instead of the array)
6. I suggest testing and debugging the code using a small array to start – 10 to 20 items
7. Write one class that invokes each of the three sort routines for a given input array (each routine can be its own class). One invocation of the program for an array executes the three sorts in succession
8. The five input files will be provided for use as input arrays. Your program may process them in sequence, or you may run your program individually for each of the input files. In the latter case, five separate output files will have to be consolidated for your submission.
9. For each sort, provide the following displayed messages to the execution output stream (15 sets of messages):
   - Start of {Bubble/Insertion/Merge/Quick Sort} for {100/1000/10000/100000/1000-sorted} items – for whichever sort is being done
   - Provide a programmed timing measure for each sort – how long it took
   - For Bubble and Insertion, the number of inversions (exchanges) required to perform the sort
   - End of {Bubble/Insertion/Merge/Quick Sort}

Measuring the Timing

Timing is to be measured in seconds, milliseconds, microseconds, or nanoseconds – pick one that gives an appropriate scale and be consistent within your report. Use the "poor man's timer", running it before and after each sort and compute the time as a difference between the start and end:

```
long start, end, duration;
start = System.currentTimeMillis();
//do something slow here
end = System.currentTimeMillis();
duration = end - start;

long start, end, duration;
start = System.nanoTime();
//do something slow here
end = System.nanoTime();
duration = end - start;
```

Alternately, you may use a timer provided by your IDE (if using an IDE).

Analysis Report

After successfully running the sort code and collecting the output timings, provide a report of no more than two pages (shorter is better) that compares the three sort algorithms run against the five input files. The report has three tables, one for each of the sort methods. They should be similar to the one in Weiss Fig. 5.13, and have the following structure:

<title:Sort method>

N	Run Time (T(n))	# inversions (BS, IS)	T(n)/N	T(n)/N**2	T(n)/n log(n)
100					
1,000-S					
1,000					
10,000					
100,000					

1,000-S = nearly sorted 1,000 element array case

The report text should comment on the timing results in terms of a method-to-method comparison and assess, based on the values in the table, which Big-O complexity is appropriate for each.

What and where to submit:

1. All items are to be submitted through Blackboard
2. The Java code used to perform the sorts, with a comment block that includes your name, date, "INFS-519 Assignment 1", appropriate code attributions, and a short description of the code's input, processing, and outputs. As noted above, the code should be a single stream with potentially multiple classes. The code will be assessed by visually inspecting the source and executed using the Java command line for at least one of the arrays. All code needed to execute from the command line must be included.
3. A pdf file that includes a log or screen shot of the output, including the 15 message sets described above.
4. A .doc or pdf for the report that includes the above information.

How the assignment will be assessed

The Java code will be visually inspected and executed from command line using one of the input arrays. This is not a speed test, but programs that don't finish for the 1,000 array case will be considered as not correct. The report will be reviewed for inclusion of the above items. The tables must be complete and have credible numbers in them. The text should be corroborated by what's in the tables.

Item	Assessment Description	Max Value
Java Code	All code submitted	20
	Code is readable, commented	5
	Code successfully executes against one input array with correct output	20
Execution Output	All expected output submitted and clearly marked (15 sets)	15
Report	Tables and report text submitted and complete	20
	Table data is reasonable for the sort method and input	10
	Report Text supported by table data and is readable	10
	Report size is over 2 pages long (TL/DR)	(minus)-10
Total		**100**

Extra Credit: For the class being held the night the assignment is due (9/28), I would like a few students to volunteer to present their code and results to the class. This would be a ~10 minute presentation that includes (1) a code walk-through, (2) reviewing the result tables, (3) discussing results, and (4) Q&A from the class. Contact me separately if interested. Extra credit would be +5% added to your cumulative assignment score (i.e. if the collective grade for all your assignments is 85% for the semester, this increases it to 90%). Presenting your work in front of peers is good experience (professionally and academically). I would be willing to devote up to 60 minutes at that class for these presentations.

Bubble Sort (aka Rock Sort) "near code"

```
//Bubble Sort

while (!doneFlag) {
      doneFlag = TRUE;
      for (i = 0; i < dataArray.length – 1; i++)  {
            if dataArray[i] > dataArray[i + 1];  {
                  tmp = dataArray[i];
                  dataArray[i] = dataArray[i + 1];
                  dataArray[i + 1] = tmp;
                     doneFlag = FALSE;
            }  // end if
       }   // end for
  }  //  end while
```