**Lab 2 Addendum**

You will run five shell scripts to let your program generate some data that you will analyze by creating graphs in Excel.

**Script #1**:

The first script runs both pure-Quicksort (never switching over to Shell or Insertion) in its first pass, and then runs pure-Shellsort in its second pass. This will let you evaluate the relative performance of the two algorithms. You will collect data for array sizes of: 1, 2, 5, 10, 20, 50, 100, 200, and 500 million. On my PC, this took about 3-1/2 hours. The script:

date

for THRESHOLD in 1 700000000

do echo $THRESHOLD

for SIZE in 1000000 2000000 5000000 10000000 20000000 50000000 100000000 200000000 500000000

do echo $SIZE

for TIMES in {1..10}

do ./project2 -n $SIZE -s $THRESHOLD -m n -a S -r -1| grep "Clock"

done

done

done

date

By bookending the script with the “date” command, you will be able to tell when it started and ended, even if you’re not at the PC. Knowing the elapsed time will help you estimate the time required for the other four scripts. The first pass sets the “switch to the alternate algorithm” threshold to 1, meaning that it will NEVER call Shell sort. This will let you measure the “pure” performance of Quicksort. The second pass sets the threshold to 700,000,000 – larger than any size for which you’ll run it, which means that Quicksort will immediately have Shell sort do all of the work.

Take the resulting output and combine it with the data from the next script, to produce ONE graph. The X-axis is to be labeled “Array Size”, the Y-axis is to be labeled “Sort Time”, and the graph is to be labeled “Sort Times by Size and Algorithm”. Use “Shell”, “Quick”, and “Insertion” as the series names, and display the legend

**Script #2 (about 45 minutes on my PC)**

This script runs pure-Insertion sort. Because this is an *O*(*n*2) algorithm, it would be absolutely impractical to run Insertion sort for values of SIZE as large as Quick or Shell can handle. As such, you’ll be testing it with values of 10, 20, 50, 100, 200, and 500 *thousand* (rather than *million* , as Script #1 did). If you are so inclined, it would be instructive for you to also run this one for 1 million, but you’re not required to. If you want to do so, just add 1000000 to the end of the “for SIZE” line below.

For each run, use the ten repetitions, and take the average of the Wall Clock and CPU times (i.e., for ten runs, you’ll be averaging together 20 values), and use those to plot the data point.

The combined graph, showing the data from Scripts 1 and 2 will be an X-Y scatter plot (NOT a line graph), with straight lines (and data points) and no smoothing.

Use an X-axis “Display Units” value of “Thousands”, and force the X-axis’s range to 10,000 – 500,000,000. Plot the graph two different ways – one with a logarithmic X scale, and one with a linear X scale. Feel free to play around with logarithmic displays of the Y-axis, too, but turn it in with at least a linear Y-axis, and both versions of the X-axis.

This is Script #2:

date

for SIZE in 10000 20000 50000 100000 200000 500000

do echo $SIZE

for TIMES in {1..10}

do ./project2 -n $SIZE -s 700000000 -m n -a I -r -1 | grep "Clock"

done

done

date

**Script #3:**

Script #3 will run single-threaded quicksort switching to BOTH Insertion and Shell sort as the alternate with various THRESHOLD values, in an attempt to find the better of the two algorithms, and the optimal THRESHOLD value. This ran about 8.25 hours on my PC:

date

for ALG in I S

do echo $ALG

for THRESHOLD in {3..30}

do echo $THRESHOLD

for TIMES in {1..10}

do ./project2 -n 200000000 -s $THRESHOLD -m n -a $ALG -r -1| grep "Clock"

done

done

done

date

Create an X-Y straight-line (with data points) graph with an X-axis of “THRESHOLD” and a Y-axis of “SORT TIME”. Title the graph “Sort Time for Various Threshold Values for Insertion and Shell”. Let the two data series be called “Insertion” and “Shell”. Note which alternate algorithm and what THRESHOLD value gives the best performance (i.e., the lowest run time) – you’ll need that for Script #4.

**Script #4:**

This script will let you examine the effect of varying the number of PIECES and THREADS. You must specify the algorithm and THRESHOLD values identified from Script #3 for X and Y below in order to complete this script:

date

for PIECES in {2..20}

do echo $PIECES pieces

for THREADS in {2..4}

do echo $THREADS threads

for TIMES in {1..10}

do ./project2 -n 200000000 -m y -p $PIECES -t $THREADS -r -1 -a **X** -s **Y**| grep "Total Run"

done

done

done

date

Plot the average of the total run times (start-to-finish; NOT just the sort times) for each set of ten repetitions. You will have one series for Threads = 2, one for Threads = 3, and one for Threads = 4, and each will run PIECES (along the X-axis) from 2 to 20

**Script #5:**

This script ran for about 20 minutes on my PC. It will let us evaluate the effect of the median-of-three option. It runs the same value of N (200 million) 20 times, both with and without Median-of-Three. You are to average the wall clock and CPU times (each set of 20 runs will produce 40 values, which you are to average together). Produce a two-column graph, whose Y-axis starts at zero, and show the times using and not using median-of-three.

date

for MO3 in Y N

do echo $MO3

for TIMES in {1..20}

./project2 -n 200000000 -a i -m y -s 15 -m3 $MO3 -r -1 | grep "Clock"

done

done

date

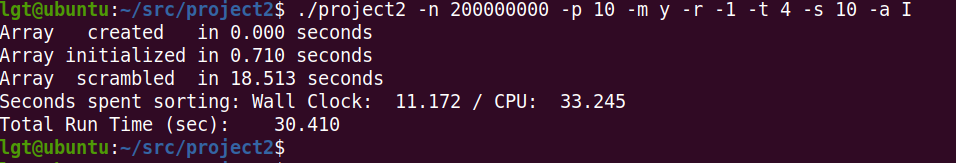
Create an MS Word document, and paste the five plots (two versions of the Script1/Script2 plot, and one each for Scripts 3, 4, and 5) into the document, with a brief paragraph after each, describing what each graph tells you. The paragraph can be just a couple of sentences, but you need to comment on each. If there is something surprising, or otherwise noteworthy, a little exposition is a good idea. This is your opportunity to tell me not only what you *found*, but how you *interpret* it.

Each graph should be the width of the page (feel free to use 1” margins all the way around, or you can use “Narrow” if you prefer). Make each graph tall enough to clearly show what’s going on in the data. Use letter-size, portrait orientation.

Submit the Word document, along with your source file (a SINGLE .c file). You must use Excel and Word.

Name both files <LastName><comma><space><FirstName> (one .docx, and the other .c) to submit them.

Note: You may need to change the string on some of the scripts that grep will use for filtering the output. Here is a screenshot of my code running:



By filtering on “Wall” (or “Clock”), I can get the time spent sorting (for that matter, I could have filtered on “spent” or “sorting”). By filtering on “Total Run”, I can get the gross run time. How you labeled you output will determine what your grep string should look like. Note that most of the scripts are looking for SORT time, but Script #4 is looking for TOTAL run time (from when main starts up until, but not including, when isSorted() is called at the very end – i.e., DON’T count the time isSorted() takes).