CS 325 Spring 2020

HW 1 – 30 points

1) (1 pt) Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in 64nlgn steps. For what values of n does insertion sort run faster than merge sort?

Note: lg n is log "base 2" of n or $\log_2 n$. For most calculators you would use the change of base theorem to numerically calculate lgn.

2) (5 pts) For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and explain.

   a. $f(n) = n^{0.25}$; $g(n) = n^{0.5}$

   b. $f(n) = n$; $g(n) = \log^2 n$

   c. $f(n) = \log n$; $g(n) = \ln n$

   d. $f(n) = 1000n^2$; $g(n) = 0.0002n^2 – 1000n$

   e. $f(n) = n\log n$; $g(n) = n\sqrt{n}$

   f. $f(n) = e^n$; $g(n) = 3^n$

   g. $f(n) = 2^n$; $g(n) = 2^{n+1}$

   h. $f(n) = 2^n$; $g(n) = 2^{2n}$

   i. $f(n) = 2^n$; $g(n) = n!$

   j. $f(n) = \lg n$; $g(n) = \sqrt{n}$

3) (4 pts) Let f_1 and f_2 be asymptotically positive non-decreasing functions. Prove or disprove each of the following conjectures. To disprove give a counter example.

   a. If $f_1(n) = \Theta(g(n))$ and $f_2(n) = \Theta(g(n))$ then $f_1(n) = \Theta(f_2(n))$.

   b. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then $f_1(n) + f_2(n) = \Theta(g_1(n) + g_2(n))$

4) (10 pts) **Merge Sort and Insertion Sort Programs**

Implement merge sort and insertion sort to sort an array/vector of integers. Name your programs "mergesort" and "insertsort". Both programs should read inputs from a file called "data.txt" where the first value of each line is the number of integers that need to be sorted, followed by the integers.

Example values for data.txt:

4 19 2 5 11

8 1 2 3 4 5 6 1 2

The output will be written to files called "merge.out" and "insert.out".

For the above example the output would be:

2 5 11 19

1 1 2 2 3 4 5 6

To receive full credit all code must be commented.

Submit a copy of your insertsort and mergesort in a ZIP file to TEACH. We will test execution with an input file named data.txt.

5) (10 pts) **Merge Sort vs Insertion Sort Running time analysis**

a) Modify code- Now that you have verified that your code runs correctly using the data.txt input file, you can modify the code to collect running time data. Instead of reading arrays from the file data.txt and sorting, you will now generate arrays of size n containing random integer values from 0 to 10,000 to sort. Use the system clock to record the running times of each algorithm for n = 5000, 10000, 15000, 20,000, …. You may need to modify the values of n if an algorithm runs too fast or too slow to collect the running time data. Output the array size n and time to the terminal. Name these new programs insertTime and mergeTime.

Submit a copy of the timing programs to TEACH in the Zip file from problem 5, also include a "text" copy of the modified timing code in the written HW submitted in Canvas.

b) Collect running times - Collect your timing data on the engineering server. You will need at least seven values of t (time) greater than 0. If there is variability in the times between runs of the same algorithm you may want to take the average time of several runs for each value of n. Create a table of running times for each algorithm.

c) <u>Plot data and fit a curve</u> - For each algorithm plot the running time data you collected on an individual graph with n on the x-axis and time on the y-axis. You may use Excel, Matlab, or any other software. What type of curve best fits each data set? Give the equation of the curves that best "fits" the data and draw that curves on the graphs.

d) <u>Combine</u> - Plot the data from both algorithms together on a combined graph. If the scales are different you may want to use a log-log plot.

e) <u>Comparison</u> - Compare your experimental running times to the theoretical running times of the algorithms? Remember, the experimental running times were the "average case" since the input arrays contained random integers.

Submit your code to TEACH in a zip file with the code from Problems 4 & 5.