CS 325 Spring 2018
HW 1 – 30 points

1) (3 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) = $\sqrt{n}$
   b. f(n) = log n^2; g(n) = ln n
   c. f(n) = nlog n; g(n) = $n\sqrt{n}$
   d. f(n) = 2^n; g(n) = 3^n
   e. f(n) = 2^n; g(n) = 2^{n+2}
   f. f(n) = 4^n; g(n) = n!

2) (3 pts) Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2, & if\ n = 2 \\ 2T\left(\frac{n}{2}\right) + n, & if\ n = 2^k, \text{for } k > 1 \end{cases}$$

   is $T(n) = n \lg n$.

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

   b. If $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$ then $f_1(n) + f_2(n) = O(g(n))$

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

   Implement merge sort and insertion sort to sort an array/vector of integers. Implement your algorithms in C++ and name your programs "mergesort.cpp" and "insertsort.cpp". Your programs should compile with the commands g++ mergesort.cpp and g++ insertsort.cpp. 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.cpp and mergesort.cpp in a ZIP file to TEACH. We will test execution with an input file named data.txt.

CS 325 Spring 2018
HW 1 – 30 points

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

The goal of this problem is to compare the experimental running times of the two sorting algorithms.

a) Now that you have proven 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=5,000, 10,000, 15,000, …70,000. containing random integer values from 0 to 10,000 to sort. Output the array size n and time in seconds to the terminal using printf or cout. Name these new programs insertTime.cpp and mergeTime.cpp.

Submit a copy of the timing programs to TEACH in the Zip file from problem 4, also include a "text" copy of the modified timing code in the written HW submitted in Canvas.

b) 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. 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. Collect your timing data on the engineering server flip. Create a table of running times.

c) 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, R or any other software. Also plot the data from both algorithms together on a combined graph. Which graphs represent the data best?

d) What type of curve best fits each data set? Again you can use Excel, Matlab, any software or a graphing calculator to calculate a regression equation. Give the equation of the curve that best "fits" the data and draw that curve on the graphs of created in part c).

e) How do your experimental running times compare to the theoretical running times of the algorithms? Remember, the experimental running times were "average case" since the input arrays contained random integers.

EXTRA CREDIT: A Tale of Two Algorithms: *It was the best of times, it was the worst of times…*

Generate best case and worst case inputs for the two algorithms and repeat the analysis in parts b) to e) above. To receive credit you must discuss how you generated your inputs and your results.

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