CS 325 HW 1 – 30 points

1) (6 pts) For each of the following pairs of functions, select the best relationship from the options: f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) is Θ(g(n)) and give a brief explanation.

   a. f(n) = $n^{0.25}$; g(n) = $n^{0.5}$
   b. f(n) = log n^2; g(n) = log n
   c. f(n) = nlogn + n; g(n) = $n\sqrt{n}$
   d. f(n) = e^n; g(n) = 2^n
   e. f(n) = 2^n; g(n) = 2^{n+1}
   f. f(n) = n^n; g(n) = n!

2) (4 pts) Use mathematical induction to show that when n is an exact power of 2, the solution to 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) (10 pts) **Merge Sort and Insertion Sort Programs**

   Implement merge sort and insertion sort to sort an array/vector of integers. You may implement the algorithms in C, C++ or Python, name the programs "mergesort" and "insertsort". Your programs should be able to 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.txt" and "insert.txt".

   For the above example the output would be:

   2 5 11 19
   1 1 2 2 3 4 5 6

   Submit a copy of all your code files and a README file that explains how to compile and run your code in a ZIP file to TEACH. We will test execution with an input file named data.txt.

CS 325 HW 1 – 30 points

4) (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 ten different values of n for example: n = 5000, 10000, 15000, 20,000, …, 50,000. You may need to modify the values of n if an algorithm runs too fast or too slow to collect the running time data (do not collect times over a minute). 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 3.

b) Collect running times - Collect your timing data on the engineering server. You will need at least eight 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) Plot data and fit a curve - 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. 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) Combine - 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) Comparison – How does your experimental running times compare to the theoretical running times of the algorithms?