CS 325 Fall 17
HW 1 – 30 points

1) (3 pts) Describe a $\Theta(n \lg n)$ time algorithm that, given a set S of *n* integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x. Explain why the running time is $\Theta(n \lg n)$.

2) (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. $f(n) = n^{0.25}$; $g(n) = n^{0.5}$
   b. $f(n) = \log n^2$; $g(n) = \ln n$
   c. $f(n) = n\log n$; $g(n) = n\sqrt{n}$
   d. $f(n) = 4^n$; $g(n) = 3^n$
   e. $f(n) = 2^n$; $g(n) = 2^{n+1}$
   f. $f(n) = 2^n$; $g(n) = n!$

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 $f_1(n) + f_2(n) = O(g(n))$.

   b. If $f(n) = O(g_1(n))$ and $f(n) = O(g_2(n))$ then $g_1(n) = \Theta(g_2(n))$

4) (10 pts) **Merge Sort and Insertion Sort Programs**
   Implement merge sort and insertion sort to sort an array/vector of integers. You may implement the algorithms in the language of your choice, name one program "mergesort" and the other "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.out" and "insert.out".
   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 Fall 17
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 a file to sort, you will now generate arrays of size n containing random integer values from 0 to 10,000 and then time how long it takes to sort the arrays. We will not be executing the code that generates the running time data so it does not have to be submitted to TEACH or even execute on flip. Include a "text" copy of the modified code in the written HW submitted in Canvas.

b) Use the system clock to record the running times of each algorithm for n = 1000, 2000, 5000, 10,000, …. You may need to modify the values of n if an algorithm runs too fast or too slow to collect the running time data. If you program in C your algorithm will run faster than if you use python. 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.

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: *It was the best of times, it was the worst of times…*

Generate best case and worst case input for both algorithms and repeat the analysis in parts b) to d) above. Discuss your results and submit your code to TEACH.