Programming 1 – Due Monday, January 30, 2012
Getting to Know Python

Collaboration  You should so the following assignment on your own, though it is fine, in this specific case, to get code off the web as long you cite your source.

Code  Write Python (version 2.7.2) code to do the following:

1. Given a parameter n, generate a list of n random numbers in the range [0,1].
2. Given a list of numbers, sort it in ascending order using merge sort and print the total execution time of the sort.
3. Given a list of numbers, sort it using insertion sort and print the execution time of the sort.

To calculate the total processing time of a section of code, you can import the time module and insert start = time.clock() before the code you wish to time and elapsed = (time.clock() - start) after the code you wish to time. Then elapsed will give the total time for the section of code.

Note: For this assignment, you are allowed to find and use code on the Internet for insertion sort and merge sort, as long as you cite your source in a comment in the code.

A Simple Experiment Using this code, run insertion sort on a randomly generated list of length 32. Record the processing time; do this a total of 10 times. Repeat this with randomly generated lists of length 64, 128 and so on through 2^{20}. You can stop timing insertion sort when it takes more than a minute to run.

Repeat this process with merge sort.

Repeat using both insertion sort and merge sort on lists that are already in sorted order prior to running the algorithms.

Analysis  Calculate the average processing time and standard deviation for each list length with each sorting algorithm. Create a plot using your plotting program of choice (eg, Excel) with this data to show list length vs. processing time. Include error bars.

Questions:

1. When did insertion sort perform better? When did merge sort perform better? [You may want to experiment with list sizes that are not powers of 2 to get a better sense of where mergesort starts to outperform insertion sort.]
2. Do the execution times appear to scale as \( n^2 \) and \( n \log(n) \), respectively? If so, what are the constants of proportionality on your system? If not, what sort of scaling did you observe and what factors do you think might be making the experimental observations diverge from the theoretical prediction?

3. Suppose you change the code to mergesort so that it uses insertion sort whenever the array is below a certain length. Can you set the length appropriately to speed up your mergesort code? By how much does it speed things up?

Submit You should submit the following via Angel

1. Your appropriately documented Python code;
2. A table of your average running times for each algorithm with each list type;
3. the plots of average running times, with error bars;
4. your answers to the questions above.