**Homework 1**

**Problem 1** **(4 points)**

Order the following functions by growth rate in increasing order:

$N$, $\sqrt{N}$, $N^{1.5}$, $N^2$, $N \cdot logN$, $N \cdot log(logN)$, $N \cdot log^2 N$, $N \cdot log(N^2)$, $\frac{2}{N}$, $2^N$, $2^{N/2}$, $37$, $N^2 \cdot \log N$, $N^3$. Indicate which functions grow at the same rate.

*Note: all logs are base 2, unless otherwise specified.*

**Problem 2** **(3 points)**

Which function grows faster: $N \cdot logN$ or $N^{(1+\frac{\varepsilon}{\sqrt{logN}})}$, $\varepsilon > 0$ ($\varepsilon$ is a constant value greater than zero)? Show your work in details.

**Problem 3** **(3 points)**

For each of the following six program fragments, give an analysis of the running time (providing Big-Oh analysis of each fragment is enough).

(a)     sum = 0;

        for( i = 0; i < n; ++i )

           ++sum;

(b)     sum = 0;

      for( i = 0; i < n; ++i )

        for( j = 0; j < n; ++j )

          ++sum;

(c)     sum = 0;

      for( i = 0; i < n; ++i )

        for( j = 0; j < n * n; ++j )

          ++sum;

(d)      sum = 0;

        for( i = 0; i < n; ++i )

            for( j = 0; j < i; ++j )

                ++sum;

(e)      sum = 0;

        for( i = 0; i < n; ++i )

            for( j = 0; j < i * i; ++j )

                for( k = 0; k < j; ++k )

                    ++sum;


(f)      sum = 0;

        for( i = 1; i < n; ++i )

            for( j = 1; j < i * i; ++j )

              if( j % i == 0 )

                    for( k = 0; k < j; ++k )

                        ++sum;


## Problem 4    (10 points)

**Merge Sort and Insertion Sort Programs**

Implement merge sort and insertion sort to sort an array/vector of integers in **ascending order**. 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. The input file may have several lines as shown in the following example.

Example for data.txt:

```
4 19 2 5 11
8 1 2 3 4 5 6 1 2
5 0 -5 3 7 -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
-5 -2 0 3 7
```

*We will test execution with an input file named data.txt. The input values will be always valid.*

**Problem 5     (10 points)**

**Merge Sort vs Insertion Sort Running time analysis**

(a) 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 -10,000 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.

(b) Collect your timing 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.  Create a table of running times for each algorithm.

(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, 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) 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) 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.