CS206 Introduction to Data Structures

Lab 4

How long did that take? (again!!)

UNIX: Killing things

In Unix, to kill a running process you can often type CTRL-C

For instance, if you start a java program from the command line using “java Timer 10000000” then CTRL-C will kill it.

Sometimes is is more complex to stop a running process. For instance, suppose you start a “background” process, using

UNIX> java Timer 10000 &

Then you will immediately get a unix prompt back, but the java program may be running amok. To kill that you need to know the process ID. You can get this using the ps command

UNIX> ps

Then identify the process ID number of your running program in the resulting list Call it PID. Finally

UNIX> kill -9 PID

Killing things might come in handy in this lab.

How long did that take:

Here is timing code similar to that used in class and the last lab. (It differs from the last lab in that doWork is empty and it expects command line input.)

```
public class Timer {
    private static final double NANOS_SEC = 
1000000000.0; // nanosec per sec

    public static void main(String[] args) {
        int len = Integer.parseInt(args[0]);
        long[] data = new long[len];
        for (int i=0; i<len; i++)
```

```
        data[i]=i;
        long startTime = System.nanoTime(); //
yes, timing in nanoseconds
        new Timer().doWork(data);
        long endTime = System.nanoTime();
        // now covert nanoseconds to seconds
        System.out.println("Time: " + (endTime-
startTime)/NANOS_SEC);
    }
    public void doWork(long[] data) {
        double res=0.0;
        System.out.println(res);
    }
}
```

In this lab you will write several versions of the doWork and collect data on each version. Use all of that to fill in the following table

	100	10,000	1,000,000	100,000,0(
O(n)				
O(log n)				
O(n^2)				
O(n^3)				
O(n^n)				

Into each cell of the table, put the time required by an algorithm of complexity given by the row and the size of input given by the column and observed on your machine. Run each test 5 times. (So each cell will have 5 numbers in it). When yo are implementing each algorithm be as simple as you can. However, beware of the Java compiler, it is smart and will skip steps it believes are not relevant. As a result you O(N^2) algorithm could run in O(N) or even O(1) time.

Note that you will likely not be able to get a time for 1000000 and O(n^n). My implementation of n^n could only do about 9000, it was not time but system limitations. Similarly, a time for N^2 at 100,000,000 may be beyond your computer. (Also I started out by writing an O(n^(n^n)) algorithm. This one got boring by about 9.

What to Hand In:

Send email to gtowell206@cs.brynmawr.edu with the following:

The filled out table. (Or as far as you got).