CSE 373 Data Structures 13au, Homework 2

Due at the BEGINNING of class, Friday, 10/11/2013

Here are *five* questions on complexity and algorithm analysis. You only need to turn in written (or printed) solutions, although you will need to write and run some code for one of the problems.

Problems

1. Prove (using induction) that for all N greater than or equal to 1:

$$\sum_{i=1}^{N} i^3 = \left(\sum_{i=1}^{N} i\right)^2$$

   Hints:

   - Start with N=1 as the base case. In the inductive case, start with $\sum_{i=1}^{N+1} i^3$ and show via a sequence of steps, including one step that uses the induction hypothesis, that it is equal to $\left(\sum_{i=1}^{N+1} i\right)^2$.
   - You already know what the sum of $\sum_{i=1}^{N} i$ is for any N (we discussed it in class), and you should use this fact a couple of times in your inductive case.
   - You will also need to do a little bit of factoring and other algebra manipulations.

2. Order the following functions from slowest growth rate to fastest growth rate.
   N^2, $N \log N$, $2/N$, $\log^2 N$, 2^N, $\sqrt{N}$, 56, $N^2 \log N$, $N^{1.5}$,
   $2^{N/2}$, $\log N$, $N \log (N^2)$, N^5, $N \log \log N$, $N \log^2 N$, N.
   If any of the functions grow at the same rate, be sure to indicate this.

3. Suppose $T_1(N)$ *is* $O(f(N))$ and $T_2(N)$ *is* $O(f(N))$. Which of the following are always true (for all T_1, f, *and* T_2)?

   a. $T_1(N) / T_2(N)$ *is* $O(1)$
   b. $T_1(N) - T_2(N)$ *is* $\Theta(f(N))$ *(notice here we are using "big-Theta")*
   c. $T_1(N) + T_2(N)$ *is* $O(f(N))$
   d. $T_1(N)$ *is* $O(T_2(N))$

   You do not need to prove an item is true (just saying true is enough for full credit), but if an item is false need to give a counterexample to demonstrate it is false. To give a counterexample, give values for $T_1(N)$, $T_2(N)$, and $f(N)$ for which the statement is false (for example, you could write, "The statement is false if $T_1(N)=100N$, $T_2(N)=2N^2$ and $f(N)=N^3$"). Hints: Think about the definitions of big-O and big-Θ.

4. For each of the following **seven** program fragments:
   a. Give an asymptotic analysis of the running time using big-O (or big-Θ, which would technically be more precise)
   b. Implement the code in Java, and give the actual running time for several (at least 4) values of N.
   c. Compare your analysis with the actual running times.

   For part (b), please turn in a printout of your Java code, (no electronic submission required). Hints: you will want to use assorted (at least 4) large values of *n* to get meaningful experimental results. You may find the library function `System.nanoTime()` to be useful in timing code fragments. A link to some Java code showing an example of timing can be found at http://courses.cs.washington.edu/courses/cse373/13au/Timing.java

1)
```
sum = 0;
for (i = 0; i < n; i++) {
    sum++;
}
```

2)
```
sum = 0;
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        sum++;
    }
}
```

3)
```
sum = 0;
for (i = 0; i < n; i++) {
    for (j = 0; j < i; j++) {
        sum++;
    }
}
```

4)
```
sum = 0;
for (i = 0; i < n; i++) {
    for (j = 0; j < n * n; j++) {
        sum++;
    }
}
```

5)
```
sum = 0;
for (i = 0; i < n; i++) {
    for (j = 0; j < i; j++) {
        sum++;
    }
    for (k = 0; k < 8000; k++) {
        sum++;
    }
}
```

6)
```
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++;
            }
        }
    }
}
```

7)
```
sum = 0;
for (i = 0; i < n; i++) {
    for (j = 0; j < i * i; j++) {
            sum++;
    }
}
```

Note that there are THREE parts to this question, do all three. a) calculate big-*O*, b) run the code for *several values of N* (4 or more) and time it, c) discuss what you see. For part (c), be sure to say something about what you saw in your run-times, are they what you expected based on your big-*O* calculations? If not, any ideas why not? Graphing the values you got from part (b) might be useful for your discussion. Remember that when giving the big-*O* running time for a piece of code we always prefer the tightest bound we can get.

It is entirely possible that your run-times will not be exactly what you might predict because Java compilers and modern computers are sophisticated and do many things more than just "naively run your code." That is okay (though do make sure your code is implemented correctly). You will hopefully still at least see some relative trends for different values of N, but in any case report what you observe and your best possible explanations for what you are seeing.

5. Show that the function $6n^3 + 30n + 503$ is $O(n^3)$. You will need to use the definition of $O(f(n))$ to do this. In other words, find values for c and n_0 such that the definition of big-*O* holds true as we did with the examples in lecture.