CSE 373 Data Structures: Homework #2

Due: at the BEGINNING of class, 2:30 PM, Wednesday, 4/15/2015

Here are *five* questions on complexity and algorithm analysis. You will need to turn in two files, a PDF with all of your answers, and java code for your tests for problem 5. Begin each problem on a new page. Please write down your FULL name on the top of each page.

For your writeup submission, turn in one PDF file with your answers to the Catalyst Drop Box. `https://catalyst.uw.edu/collectit/assignment/cmbaker/35027/142087` . You may either type up all of the questions and save to PDF or complete the questions on paper and scan them or take a high-quality picture of each page and join them all into one PDF file, or any hybrid of these strategies. Make sure it's legible - if we cannot read it we cannot grade it.

Problem 1.

Prove by 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 step, show that

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

  via a sequence of steps, including one step that uses the inductive hypothesis.
- You already know what $\sum_{i=1}^{n} i$ is for any n (we discussed this in class), and you should use this fact a couple of times in your proof.
- You will also need to do a little bit of factoring and other algebra manipulations.

Problem 2.

Order the following functions from slowest growth rate to fastest growth rate.

- n^2
- $n \log n$
- $\frac{2}{n}$
- $\log^2 n$
- 3^n
- $\sqrt{n}$
- 13
- $n^2 \log n$
- $n^{1.5}$
- $3^{n/2}$
- $\log n$
- $n \log n^2$
- n^3
- $n \log \log n$
- $n \log^2 n$
- n

If any of the functions grow at the same rate, be sure to indicate this.

Problem 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. $T_1(n)$ is $\Theta(f(n))$.

b. $T_1(n) \cdot T_2(n)$ is $\Omega(f(n))$.

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.

Problem 4.

Show that the function $5n^3 + 30n + 303$ is $O(n^3)$. You will need to use the definition of $O(f(n))$ to do this. That is, you will need to find values for c and n_0 such that the definition of big-O holds true as we did with the examples in lecture.

Problem 5.

For each of the following *six* program fragments, do the following:

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 for several (at least four) values of n.

c. Compare your analysis with the actual running time.

For part (b), please turn in a copy of your Java code. 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/cse373/15sp/homework/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 = 0; i < n; i++) {
    for (j = 0; j < i*i; j++) {
        if (j % i == 0) {
            for (k = 0; k < j; k++) {
                sum++;
            }
        }
    }
}
```

Note that there are *three* parts to this question, so be sure to 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.