CSE332 Week 2 Section Worksheet Solutions

1. Prove f(n) is O(g(n)) where

a.

$f(n)=7n$

$g(n)=n/10$

Solution:

According to the definition of O( ), we need to find positive real #'s n_0 & c so that

$f(n)<=c*g(n)$ for all $n>=n_0$

So, set one of them, solve the equation. $n_0=1$ & c greater than or equal to 70 works.

b.

$f(n)=1000$

$g(n)=3n^3$

Solution:

According to the definition of O( ), we need to find positive real #'s n_0 & c so that

$f(n)<=c*g(n)$ for all $n>=n_0$

Easiest way to do this would be to set $n_0=1$ and solve the equation. $n_0=1$ and any c from 334 and up works.

c.

$f(n)=7n^2+3n$

$g(n)=n^4$

Solution:

According to the definition of O( ), we need to find positive real #'s n_0 & c so that

$f(n)<=c*g(n)$ for all $n>=n_0$

Easiest way to do this would be to set $n_0=1$ and solve the equation. We then get c=10, and g rises more quickly than f after that. There are many more other such solutions, just make sure you plug them back in to check that they work.

These, you could solve in a number of ways. You could also graph them and observe their behavior to find an appropriate value.

d.

$f(n)=n+2n\log n$

$g(n)=n\log n$

Solution:

$n_0=2$ & $c=3$

The values we choose do depend on the base of the log; here we'll assume base 2

To keep the math simple, we choose n_0 of 2. Solving the equation gets us c=3.

We could also use log base 10, and we'd get c = 3, and $n_0 = 10$. Or $n_0 = 2$, c=10.

2. True or false, & explain

a. f(n) is Θ(g(n)) implies f(n) is O(g(n))

Solution:

True: Based on the definition of Θ, f(n) is O(g(n))

b. $f(n)$ is $\Theta(g(n))$ implies $g(n)$ is $\Theta(f(n))$
Solution:

True: Intuitively, Θ is an equals, and so is symmetric.
More specifically, we know

f is $O(g)$ & f is $\Omega(g)$

so

There exist positive # c, c', n_0 & n_0' such that

$f(n)<=cg(n)$ for all $n>=n_0$

and

$f(n)>=c'g(n)$ for all $n>=n_0'$

so

$g(n)<=f(n)/c'$ for all $n>=n_0'$

and

$g(n)>=f(n)/c$ for all $n>=n_0$

so g is $O(f)$ and g is $\Omega(f)$
so g is $\Theta(f)$

c. $f(n)$ is $\Omega(g(n))$ implies $f(n)$ is $O(g(n))$
Solution:

False: Counter example: $f(n)=n^2$ & $g(n)=n$; $f(n)$ is $\Omega(g(n))$, but $f(n)$ is NOT $O(g(n))$

3. Find functions $f(n)$ and $g(n)$ such that $f(n)$ is $O(g(n))$ and the constant c for the definition of O( ) must be >1. That is, find f & g such that c must be greater than 1, as there is no sufficient n_0 when c=1.
Solution: Basically, you need to think up two functions where one is always greater than the other and never crosses, but if you multiply one of them by something, there is a crossing point where they reverse, and it will shoot up past the other function.

Consider

$f(n)=n+1$
$g(n)=n$

we know $f(n)$ is $O(g(n))$; both run in linear time
Yet $f(n)>g(n)$ for all values of n; no n_0 we pick will help with this if we set c=1.
Instead, we need to pick c to be something else; say, 2.

$n+1 <= 2n$ for $n>=1$

4. Write the O( ) run-time of the functions with the following recurrence relations
a. $T(n)=3+T(n-1)$, where $T(0)=1$
Solution:

$T(n)=3+3+T(n-2)=3+3+3+T(n-3)=\ldots=3k+T(0)=3k+1$, where $k=n$,
so $O(n)$ time.

b. $T(n)=3+T(n/2)$, where $T(1)=1$
Solution:

$T(n)=3+3+T(n/4)=3+3+3+T(n/8)=\ldots=3k+T(n/2^k)$
we want $n/2^k=1$ (since we know what $T(1)$ is), so $k=\log_2 n$
so $T(n)=3\log n+1$, so $O(\log n)$ time.

c. T(n)=3+T(n-1)+T(n-1) , where T(0)=1
Solution:

We can re-write T(n) as T(n) = 3+2 T(n-1)
Then to expand T(n)
T(n)
= 3 + 2 (3 + 2 T(n-2))
= 3 + 2(3 + 2 (3 + 2 T (n-3)))
= 3 + 2 (3 + 2 (3 + 2 (3 + 2 T (n-4))))
$= 3 \cdot 2^0 + 3 \cdot 2^1 + 3 \cdot 2^2 + \cdots + 3 \cdot 2^{k-1} + 2^k T(0)$ where k is the number of iterations
$= \sum_{i=0}^{k-1} 3 \cdot 2^i + 2^k \cdot 1$

Because $\sum_{i=0}^{j} m^i = m^{j+1}$-1, we can replace the summation with
$= 3 \cdot (2^k - 1) + 2^k \cdot 1$
And in this case, since we know that the number of iterations that occur is just n, k=n, and so
$= 4 \cdot 2^n - 3$
and we see that have T(n) = $8 \cdot 2^n$, and thus T(n) is in O(2^n).

Basically, since we can tell the # of calls to T( ) is doubling every time we expand it further, it runs in O(2^n) time.

5. Prove by induction that the $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

First, check the base case. Set n=1, and show that the right-hand side of the equation above is equal to 0^2 + 1^2.

Second, do the induction step.

$1 + 2^2 + 3^2 + \ldots + n^2 + (n+1)^2$

$= \frac{n(n+1)(2n+1)}{6} + (n+1)^2$

$= \frac{n(n+1)(2n+1) + 6(n+1)^2}{6} = \frac{(n+1)(n(2n+1) + 6(n+1))}{6}$

$= \frac{(n+1)(2n^2 + n + 6n + 6)}{6} = \frac{(n+1)(2n^2 + 7n + 6)}{6}$

$= \frac{(n+1)(n+2)(2n+3)}{6} = \frac{(n+1)(n+2)(2(n+1)+1)}{6}$

The final expression, on the right, is the same as if we had substituted (n+1) for (n) in the original equation, and hence we have proven the equation true for the inductive case.

(equation images in the solution to this problem above, courtesy of http://pirate.shu.edu/~wachsmut/ira/infinity/answers/sm_sq_cb.html)

6. What's the O( ) run-time of this code fragment in terms of n:

a)

```
int x=0;
for(int i=n;i>=0;i--)
        if((i%3)==0) break;
        else x+=i;
```

Solution:

At a glance we see a loop and it looks like it should be O(n); it looks like we go through the loop n times.

However, that 'break' makes things a bit weirder. Consider how the loop will work for any real data; we start at some n, count backwards **until** the value is a multiple of 3, at which point we break.

So the loop's code will run at most 3 times (not a function of n); so the whole thing is O(1).

**Recall that '%' is the remainder operator; i%3 divides i by 3 and returns the remainder (which will be 0, 1 or 2).

b) O(n^3)

Outer loop is n. Inner loop is $\frac{n^2}{3}$ times. Hence, the whole thing runs in $\frac{n^3}{3}$ time. Dropping the 1/3 constant, we get O(n^3)

c) This one is trickier. Outer loop runs in n, but inner loop runs in i*i time. Which means the first time the inner loop runs, i is only 0, so the inner loop runs 0 times. Next, i is 1, so inner loop runs 1 time. Next i=2, inner loop hence runs i^2 times, which is 4. Next time, i=3, inner loop goes 9 times. And so forth. So the number of executions ends up being 0 + 1 + 4 + 9 + … + n^2 times. We can use the formula we just found in problem 5 here, to represent this summation, $\frac{n(n+1)(2n+1)}{6}$. And so, this expression is O(n^3).