CSE332 Week 2 Section Worksheet Solutions

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

a.

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

$g(n)=n^4$

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 $n>=n_0$

Pick $n_0=1$ & c=10; f & cg are equal at n=1, and g rises more quickly than f after that.

b.

f(n)=n+2nlogn

g(n)=nlogn

Solution:

$n_0=2$ & c=10

**Why will $n_0=1$ & c=10 not work? Consider

f(1)=1+2*1*log1=1+2*1*0=1

g(1)=1*log1=1*0=0

so f(1) > 10*g(1)

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

c.

f(n)=1000

$g(n)=3n^3$

Solution:

$n_0=1$ & c=400 works

d.

f(n)=7n

g(n)=n/10

Solution:

$n_0=1$ & c=100 works

2. True or false, & explain

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

Solution:

True: Intuitively, Θ is an equals, and so is symmetric.

More specifically, we know

f is O(g) & f is Ω(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 Ω(f)

so g is Θ(f)

b. 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))

c. f(n) is Ω(g(n)) implies f(n) is O(g(n))

Solution:

False: Counter example: $f(n)=n^2$ & $g(n)=n$; f(n) is Ω(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:

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)=…=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)=3logn+1, so O(logn) 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
$= \sum_{i=0}^{k-1} 3 \cdot 2^i + 2^k \cdot 5$
Because $\sum_{i=0}^{j} m^i = m^{j+1}$, we can replace the summation with
$= 3 \cdot 2^k + 2^k \cdot 5$
And in this case, since we know that the number of iterations that occur is just n, k=n, and so
$= 3 \cdot 2^n + 5 \cdot 2^n$
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. What's the O( ) run-time of this code fragment in terms of n:

```
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).