Solutions for Homework 1

Maxwell Young

August 29, 2003

Exercise 3.1-2

We want to show that $(n+a)^b = \Theta(n^b)$. As done in class, we set up the following:

$$0 \leq c_1 n^b \leq (n+a)^b \leq c_2 n^b$$

and then look at

$$0 \leq c_1 \leq (1+a/n)^b \leq c_2$$

You can solve this on a case by case basis. What you have to worry about (at least initially, it that a can be negative). When you solve for both cases (a positive and a negative), then you can just choose your biggest value for c_2 and your smallest value for c_1. You can only do this if you recognize that we always consider n to be growing (towards infinity), so no matter how largely negative a can become, n will at some point surpass it (at some $n = n_0$). By this reasoning, letting $n_0 = |a| + 1$, c_2=2^b, and $c_1 = 1/n_0$ should work.

Exercise 3.1-5

This question is used to test your understanding of the formal definitions for both O and Ω notations. We want to prove an "if and only if" (abbreviated iff) and so we most prove two implications.

First Implication:

Prove: assuming f(n)=Θ(g(n)), then f(n)=O(g(n)) and f(n)=Ω(g(n)).

Proof: by the definition on page 42 of CLRS, f(n)=Θ(g(n)) means that there exist positive constants c_1 and c_2 and n_0 such that:

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0.$$

Hence, it is certainly true that there exist positive constants c and m_0 such that $0 \leq f(n) \leq cg(n)$. In particular, let c=c_2 and m_0=n_0. Therefore, by definition, f(n)=O(g(n)). Similarly, it is certainly true that there exist positive constants d and k_0 such that $0 \leq cg(n) \leq f(n)$. In particular, let d=c_2 and k_0=n_0. Therefore, by definition, f(n)=Ω(g(n)). We have just proved the first implication.

Second Implication:

Prove: assuming f(n)=O(g(n)) AND f(n)=Ω(g(n)), then f(n)=Θ(g(n)).
Proof: our assumption tells us the following:
1. There exist positive constants c_2 and p_0 such that $0 \leq f(n) \leq c_2 g(n)$ for all $n \geq p_0$.
2. There exist positive constants c_1 and m_0 such that $0 \leq c_1 g(n) \leq f(n)$ for all $n \geq m_0$.
Let n_0=max(p_0,m_0), then it is true that:

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$

which is what we wanted to prove. We have proved both implications, so we are done.

Exercise 3.1-7

Proof: Assume that we have a function f(n) and that f(n)=o(g(n)) and that f(n)=ω(g(n)). Now what does this mean? By the definition in CLRS, f(n)=o(g(n)) implies that for any constant a>0 there exists $n_0 > 0$ such that $0 \leq f(n) < ag(n)$. Also by definition, $f(n) = \omega(g(n))$ means that for any constant b>0 there exists n_0 >0 such that $0 \leq bg(n) < f(n)$.

So, in particular, it must be that $bg(n) < f(n) < ag(n)$ for all constants a,b>0 and for n sufficiently large. If we let a=b then we have a contradiction. Hence,it is impossible for f(n) to be in both o(g(n)) and ω(g(n)).

Exercise 4.1-6

We have $T(n) = 2T(\sqrt{n}) + 1$. We can alter this via the change of variable method; we let $n = 2^m$ and rewrite the recurrence to get $T(2^m) = 2T(2^{m/2})+1$. Now we rename $S(m) = T(2^m)$ in order to get the new recurrence $S(m) = 2S(m/2) + 1$. At this point, we are almost done - we simply have to solve the recurrence and change back to our orignal variable. We can solve the recurrence in a number of ways (recursion tree, unravelling the recurrence, Master Theorem) to see that $S(m) = \Theta(m)$. Then we get our orignal variable n back $T(n) = T(2^m) = S(m) = \Theta(m) = \Theta(lgn)$.

Exercise 4.2-1

See Figure 1. There are $lg(n) + 1$ levels in this recursion tree and from this figure we can see that we need to calculate the summation of $(3/2)^0 n + (3/2)^1 n + (3/2)^2 n + (3/2)^3 n + ...$ so we set up the summation: $\Sigma_{i=0}^{lgn}(3/2)^i n$. We can solve this using the formula for the geometric series (for the case where $|x| \geq 1$) given on page 1060 of CLRS. Using this formula we get: $2((3/2)^{lgn+1} - 1)n$ which simplifies to $3n^{lg3} - 2n$. Hence, we can guess that $T(n) = O(n^{lg3})$ (you can also check this using the Master Theorem; try it, it's good practice).

Now we want to verify this guess by using the subsitution method. To do these requires THREE steps: establishing a base case, citing the Induction Hypothesis (IH), and performing the Induction Step (IS). This question is good because it illustrates how sometimes we must modify our guess in order to perform the induction...here we go.

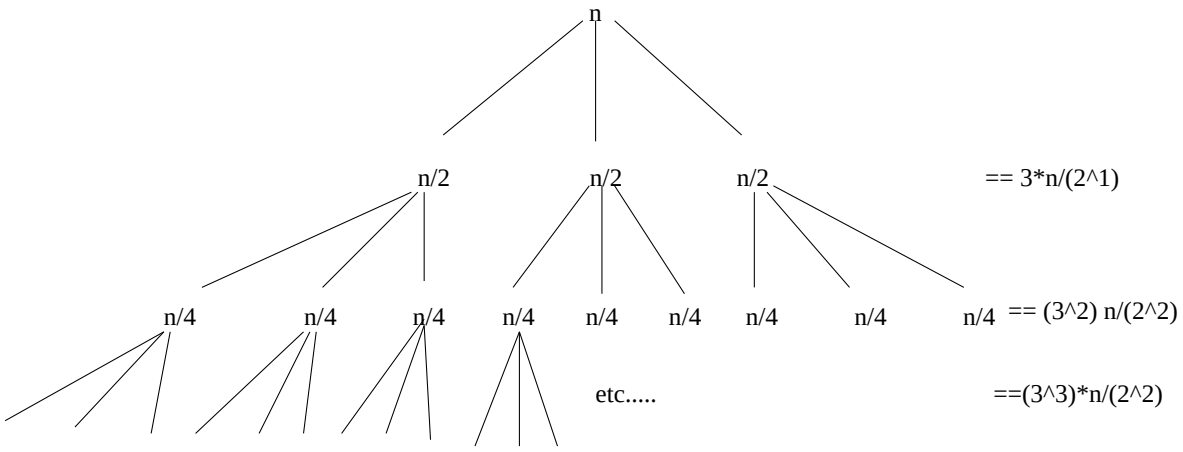

Figure 1: Recursion Tree for 4.2-1

For the base case, let n=2. Then $T(2) = 3T(1) + 2 = 5 \leq c2^{lg3}$ must be true. We can make this true by setting $c \geq 5/3$.

Next, we make the Induction Hypothesis: $T(n/2) \leq c(n/2)^{lg3}$. Then we proceed with the Induction Step: $T(n) = 3T(n/2) + n \rightarrow T(n) \leq 3c((n/2)^{lg3}) + n$ by the IH $\rightarrow T(n) \leq cn^{lg3} + n$. But now we are stuck because in order to prove the exact form required by induction, we need to prove that $cn^{lg3} + n \leq cn^{lg3}$ which is impossible no matter how we choose c. This does not mean that our recursion analysis was off, it simply means that we have to modify our IH, so here we go again...
Induction Hypothesis: assume $T(n/2) \leq c(n/2)^{lg3} - n$
Base Case: for $n = 2$, we have must have $T(2) = 3T(1) + 2 = 5 \leq c2^{lg3} - 2$ which is established if $c \geq 7/3$.
Induction Step: $T(n) = 3T(n/2) + n \rightarrow T(n) \leq 3(c((n/2)^{lg3} - n)) + n$ by the IH $\rightarrow T(n) \leq cn^{lg3} - 2n$. We have proved the correct form (according to our IH) and this is true for all positive c so we are good.

Since both the base case and induction step hold, we have proved that $T(n) = O(n^{lg3} - n)$. This is clearly in $O(n^{lg3})$ so we are done.

NOTE: Most of you who have learned mathematical induction via math classes have probably not experienced the slight twist on the induction done here; ie. usually, you are not required to choose a value for the constnt c, the induction step is much more "cooperative". Here, choosing c is very important since it is often the case that a judicious choice is required in order to prove the result you seek. Also, it is important to note that, when doing induction, you end up with exactly the form you sought to prove (see page 65 in CLRS for another example of this).

Exercise 4.2-3

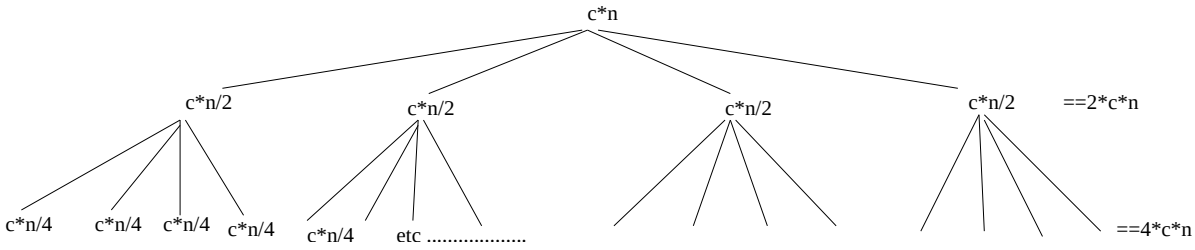

Figure 2: Recursion Tree for 4.2-3

Note: for this question I am assuming $c > 0$.

See Figure 2 There are $lg(n) + 1$ levels in this recursion tree and from the picture figure we can see that we need to calculate the summation of: $\Sigma_{i=0}^{lgn+1} 2^i cn$. Again, we use the closed form for the geometric series to find that the recurrence has a summed complexity of $(2^{lgn+1} - 1)n{=}2cn^2 - cn$. Clearly, $T(n) = O(n^2)$ (why?). We also want to show that $T(n) = \Omega(n^2)$ so that we have a tight asymptotic bound on the solution. We can do this by solving for k in: $0 \leq kn^2 \leq 2cn^2 - cn$. You can rearrange this to get the following: $k \leq c((2n-1)/n)$. Since n is a positive integer, we know that $(2n-1)/n$ is always positive and approaches a value of 2 as n tends to infinity. Hence, to pick a suitable value of k, it suffices to pick k such that $k \leq c$ (note we are placing a bound on k, not on c).

Now we want to use the substitution method to verify our solution. In order to utilize induction as a method of proof, we follow the same steps as outline in the previous question. For the base case, let $n = 2$. Then $T(2) = 4T(1)+2c \leq 4k$ which is true if we choose $k \geq (2 + c)/2$.
Induction Hypothesis: $T(n/2) \leq (n/2)^2$
Induction Step: $T(n) = 4T(n/2) + cn \rightarrow T(n) \leq 4k(n/2)^2 + 2c \rightarrow T(n) \leq kn^2 + 2c$, but then we are stuck again (just like the previous problem). Hence, we need to modify our IH.
IH: $T(n) \leq k(n/2)^2 - c(n/2)$ where k is some postive integer
BC: for $n = 2$, we must have it be true that $T(2) = 4T(1) + 2c \leq 4k - c$ so choose $k \geq (4 + 3c)/4$.
IS: $T(n) = 4T(n/2) + cn \rightarrow T(n) \leq 4(k(n/2)^2 - c(n/2)) + cn \rightarrow T(n) \leq kn^2 - 2cn + cn \rightarrow T(n) \leq kn^2 - cn$ which is the correct form we sought (ie. form of our IH).

Since both the base case and induction step hold, we have proved that $T(n) = O(n^2 - cn)$. This is clearly in $O(n^2)$, so we are HALF done. Now we want to prove that $T(n) = \Omega(n^2)$.

BC: let $n = 2$, then $T(2) = 4 + 2c \geq 2c_2$ if we let $c_1 = 1$.
IH: assume that $T(n/2) \geq c_2(n/2)^2$ where k is some constant yet to be determined.
IS: $T(n) = 4T(n/2) + cn \rightarrow T(n) \geq 4(c_2(n/2)^2) + cn$ by the IH $\rightarrow T(n) =$

$c_2n^2 + cn \rightarrow T(n) \geq c_2n^2$ for c positive.

Since both the base case and induction step hold, we have proved that $T(n) = \Omega(n^2)$. Hence, $T(n) = \Theta(n^2)$.

NOTE: it is often tedious to reguess our IH - this often means we need to redo the base case as well. Often times, for this type of question, it is good to try the IH and IS first, then prove your base case works.

Exercise 4.3-1

A) $T(n) = 4T(n/2) + n$

Clearly, a=4, b=2, and $f(n)$=n. We appeal to the almighty Master Theorem and see that $f(n)$=$O(n^{(log_2)4-\epsilon})$. Remember that $lg4$=2 and so ϵ could be, say, 0.5. By the Master Theorem, this means that this recurrence has complexity $\Theta(n^2)$.

B) $T(n) = 4T(n/2) + n^2$

In this case, a=4, b=2, $f(n)$=n^2. We see that $f(n)$=$\Theta(n^{lg4})$=$\Theta(n^2)$. Hence, by the Master Theorem, $T(n)$=$\Theta(n^2lgn)$.

C) $T(n) = 4T(n/2) + n^3$

a=4, b=2, $f(n)$=n^3. Now, $f(n)$=$\Omega(n^{lg4} + \epsilon)$ since I could choose, for example, ϵ=0.6. BUT WE ARE NOT DONE WITH THE THIRD CASE OF THE MASTER THEOREM!!! We also have to check that there exists a c <1 such that for sufficiently large n we can show $4f(n/2) \leq cf(n)$. If we let $1/2 \leq c < 1$, then we satisfy this second criteria; therefore, by the Master Theorem, $T(n) = \Theta(n^3)$.

Exercise 4-3-2

For $T(n)$ we have that a=7, b = 2, $f(n)$=n^2. We see that $f(n)$=$O(n^{lg7-\epsilon})$ and so the first case of the Master Theorem allows to prove that $T(n) = \Theta(n^{lg7})$.

For $T'(n)$ we have that b=4 and $f(n) = n^2$. We want to use the first case of the Master Theorem in order to prove an asymptotically faster running time of this algorithm (the other two cases are invalid because?). We look at $f(n)$=$O(n^{log_4a})$ and ask "for what value of a will $log_4a \leq log_27$?" To find such a value we do the following: $log_4a \leq log_27 \rightarrow (log_2a/log_24) \leq log_27 \rightarrow log_2a \leq 2log_27$ and so this gives $a \leq 49$. Hence, the largest value for which $T'(n)$ has an asymptotically faster running time is a=48 (we need it to be FASTER).

Challenge Problem 4-2

Come to my office hours if you want to see a more detailed solution to this question.

HINT: Try looking at the last bit of all n numbers. After such a search, can you eliminate half of the numbers you have to search through? If so then you have a recurrence of the form $T(n) = T(n/2) + n$ which has what kind of complexity?

Challenge Problem 4-6

Come to my office hours if you want to see a more detailed solution to this question.

HINT: For A), intuitively, if more than $n/2$ chips are bad then these chips can consistently act like good chips in order to validate the bad chips; this makes pairwise testing useless. For B), if I compare two chips and the result is that one of the chips is bad can I throw them both away and still have the remainder of my chips contain a majority of good chips? If I compare two chips and the result is that both are good or both are bad, can I throw away only one of the chips and still be left with a majority of good chips in remaining pile? Think about these questions to come up with your algorithm.