CS350 Homework 2

Russell Miller

January 26, 2011

1. Prove that $\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1}$, where r is a constant.

Assume the given statement, for n.
Rewriting in ... form. $r^0 + r^1 + \ldots + r^n = \frac{r^{n+1}-1}{r-1}$
Going to prove for n+1.
$r^0 + r^1 + \ldots + r^n + r^{n+1} = \frac{r^{n+2}-1}{r-1}$
The given statement can be substituted here, resulting in:
$\frac{r^{n+1}-1}{r-1} + r^{n+1} = \frac{r^{n+2}-1}{r-1}$
Then multiply both sides by $(r-1)$.
$\frac{\cancel{(r-1)}(r^{n+1})}{\cancel{(r-1)}} - \frac{\cancel{(r-1)}}{\cancel{(r-1)}} + (r-1)(r^{n+1}) = \frac{\cancel{(r-1)}(r^{n+2})}{\cancel{(r-1)}} - \frac{\cancel{(r-1)}}{\cancel{(r-1)}}$
Resulting in:
$r^{n+1} - \cancel{1} + (r-1)(r^{n+1}) = r^{n+2} - \cancel{1}$
Which is the same as:
$\cancel{(r^{n+1})} + r^{n+2} - \cancel{(r^{n+1})} = r^{n+2}$
Thus,
$r^{n+2} = r^{n+2}$
This proves the case for n inductively.

2. Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence:

$$T(n) = \begin{cases} 2 & if\ n = 2, \\ 2T(n/2) + n & if\ n = 2^k, for\ k > 1 \end{cases}$$

is $T(n) = n\ lg\ n$.

Suppose $n = 2^p$, where $p > 1$
$T(n) = T(2^p)$
Which by the recurrence,
$T(n) = 2T(2^{p-1}) + 2^p$
And substituting $T(n)$ for itself,
$T(n) = 4T(2^{p-2}) + 2(2^{p-1}) + 2^p$
And this continues until the leading coefficient is 2^p.
The rest of the terms are just 2^p. Note that $2(2^{p-1})$ is just like 2^{p-1+1}.
There are a total of p terms. So this can be rewritten as:
$T(n) = p2^p$
Since $n = 2^p$, $p = lg\ n$.
Thus $T(n) = lg\ n2^{lg\ n}$.
Or just $T(n) = n\ lg\ n$.

3. Write a recurrence for the running time of a recursive insertion sort.

$$T(n) = \begin{cases} \Theta(1) & if\ n \leq 1 \\ T(n-1) + \Theta(n) & if\ n > 1 \end{cases}$$

4. Write pseudocode for binary search of the structure: $A = \langle a_1, a_2, ..., a_n \rangle$. $v = A[i]$ *or* NIL.

BINARYSEARCH(A,lo,hi,value)
1 $x \leftarrow lo + (hi - lo)/2$
2 $if\ lo = hi\ and\ value \neq A[x]$
3     $return\ False$
4 $if\ value = A[x]$
5     $return\ True$
6 $else\ if\ value < A[x]$
7     $return$ BINARYSEARCH(A,lo,x-1,value)
8 $else$
9     $return$ BINARYSEARCH(A,x+1,hi,value)

In step 1, the value x is calculated to be half of the current array values. The recursive calls to BINARYSEARCH use x to only search half of the array. This means that $T(n) = \Theta(1)$ (for each pass through the function doing the comparisons) $+T(n/2)$ (for the recursive calls that are searching only half of the elements). Much like the proof for Question 2, this will result in a logarithmic total, but will not be $T(n) = n\ lg\ n$ because the comparisons are not $\Theta(n)$.
Because of the $\Theta(1)$ comparisons, rather, $T(n) = lg\ n$.

5. Describe a $\Theta(n\ lg\ n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.

SUM(S,start,size,value)
$ifsize - 1 = start$
    $return\ False$
fi
$for\ i \leftarrow start + 1\ to\ n$
    $if\ S[start] + S[i] = value$
        $return\ True$
    fi
$return$ SUM(S,start+1,size,value)

6. Use the master method to show that the solution to the binary-search recurrence $T(n) = T(n/2) + \Theta(1)$ is $T(n) = \Theta(lg\ n)$.

To set up the master method, $a = 1$, $b = 2$, and $f(n) = \Theta(1)$.
$n^{log_b a} = n^{log_2 1} = n^0 = 1$
Since $f(n) = \Theta(1)$, which is $\Theta(n^0) = \Theta(1)$,
$T(n) = \Theta(n^{log_b a} lg\ n) = \Theta(lg\ n)$.

7. Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 lg\ n$? Why or why not? Give an asymptotic upper bound for this recurrence.

No.
Here $a = 4$, $b = 2$, and $f(n) = n^2 lg\ n$.
$n^{log_b a} = n^{log_2 4} = n^2$
$f(n) = n^2 lg\ n$ is asymptotically larger than $n^{log_b a} = n^2$, but it is not *polynomially* larger. The ratio $\frac{f(n)}{n^{log_b a}} = \frac{n^2 lg\ n}{n^2} = lg\ n$ is asymptotically less than n^ϵ for any positive constant ϵ. So the recurrence falls between case 2 and 3.

Guessing that $T(n) = O(n^3)$
Going to prove that $T(n) \leq cn^3$, where $c > 0$.
Assuming this holds for $T(n/2)$, thus $T(n/2) \leq c\frac{n}{2}^3$.
Substituting into the recurrence,
$T(n) \leq 4(c(\frac{n}{2})^3) + n^2 lg\ n$
$\quad \leq (c/2)n^3 + n^2 lg\ n$
$\quad \leq (c/2)n^3$, when $c \geq 2$.
Going to show that this holds for the boundary conditions.
Assuming a boundary condition $T(1) = 1$ for $n = 1$,
$T(n) \leq (c/2)n^3$ yields $T(1) \leq (c/2)(1^3)$, or $T(1) \leq c/2$.
This holds for $c \geq 2$, which satisfies the above hypothesis that $T(n) = O(n^3)$.

8. Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is a constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

a. $T(n) = 2T(n/2) + n^3$

Using the master theorem, $a = 2$, $b = 2$, $f(n) = n^3$.
$n^{log_b a} = n^{log_2 2} = n^1$.
$f(n) = \Omega(n^{log_2 2 + \epsilon})$, where $\epsilon = 2$.
This means case 3 applies.
$af(n/b) \leq cf(n)$
$(1/4)n^3 \leq cn^3$, and let $c = 1/4$.
So $T(n) = \Theta(n^3)$.

b. $T(n) = T(9n/10) + n$

Using the master theorem, $a = 1$, $b = 10/9$, $f(n) = n$.
$n^{log_b a} = n^{log_{10/9} 1} = n^0$.
$f(n) = \Omega(n^{log_{10/9} + \epsilon})$, where $\epsilon = 1$.
This means case 3 applies.
$af(n/b) \leq cf(n)$
$9n/10 \leq cn$, and let $c = 9/10$.
So $T(n) = \Theta(n)$.

c. $T(n) = 16T(n/4) + n^2$

Using the master theorem, $a = 16$, $b = 4$, $f(n) = n^2$.
$n^{log_b a} = n^{log_4 16} = n^2$.
$f(n) = \Theta(n^{log_4 16}) = \Theta(n^2)$
This means case 2 applies.
So $T(n) = \Theta(n^2 lg\ n)$.

d. $T(n) = 7T(n/3) + n^2$

Using the master theorem, $a = 7$, $b = 3$, $f(n) = n^2$.
$n^{log_b a} = n^{log_3 7} = n^{1.77}$.
$f(n) = \Omega(n^{log_3 7+\epsilon})$, where $\epsilon \approx .33$.
This means case 3 applies.
$af(n/b) \leq cf(n)$
$(7/9)n^2 \leq cn^2$, and let $c = 7/9$.
So $T(n) = \Theta(n^2)$.

e. $T(n) = 7T(n/2) + n^2$

Using the master theorem, $a = 7$, $b = 2$, $f(n) = n^2$.
$n^{log_b a} = n^{log_2 7} = n^{2.81}$.
$f(n) = O(n^{log_2 7-\epsilon}$, where $\epsilon \approx .81$.
This means case 1 applies.
So $T(n) = \Theta(n^{log_2 7})$.

f. $T(n) = 2T(n/4) + n^{1/2}$

Using the master theorem, $a = 2$, $b = 4$, $f(n) = n^{1/2}$.
$n^{log_b a} = n^{log_4 2} = n^{1/2}$.
$f(n) = \Theta(n^{log_b a}) = \Theta(n^{1/2})$.
This means case 2 applies.
So $T(n) = \Theta(n^{1/2} lg\ n)$.

g. $T(n) = T(n-1) + n$

This is a sum from 1 to n.
The formula for that kind of sum is $\frac{n^2+n}{2}$.
Thus, $T(n) = \Theta(n^2)$.

h. $T(n) = T(sqrt(n)) + 1$

Let $m = lg\ n$.
$2^m = 2^{lg\ n} = n$
Substitute 2^m for n in $T(n)$:
$T(2^m) = T(2^{m/2}) + 1$
New function:
$S(m) = S(m/2) + 1$
Use the master method, $a = 1$, $b = 2$, $f(m) = 1$.
$m^{log_b a} = m^0 = 1$
$f(m) = \Theta(m^{log_2 1} = \Theta(1)$
This fits case 2, thus $S(m) = \Theta(lg\ m)$.
Substituting $lg\ n$ back in for m,
$T(n) = \Theta(lg(lg\ n))$.

i. $T(n) = 3T(n/2) + nlg\ n$

Using the master method, $a = 3$, $b = 2$, $f(n) = nlg\ n$.
$n^{log_b a} = n^{log_2 3} \approx n^{1.585}$
This fits case 1, thus $f(n) = O(n^{log_2 3-\epsilon}$, where $\epsilon \approx .085$.
Since $n^{log_b a}$ is not polynomially larger than $f(n)$, going to prove the above statement.
$nlg\ n \leq cn^{1.5}$ Divide by n.
$lg\ n \leq c\sqrt{n}$
Since lg grows slower than $\sqrt{}$, the above always holds.
Thus $T(n) = \Theta(n^{log_2 3})$.

j. $T(n) = T(n-1) + 1$

This is a sum of 1, n times.
This is exactly the same as $T(n) = n$, which is $\Theta(n)$.