CS325 Winter 2013: HW 1

Daniel Reichert
Trevor Bramwell
Lance Stringham

January 17, 2013

0.2 from text book

Show that, if c is a positive real number, then $g(n) = 1 + c + c^2 + \cdots + c^n$ is:

1. $\Theta(1)$ if $c < 1$
2. $\Theta(n)$ if $c = 1$
3. $\Theta(c^n)$ if $c > 1$

If $c < 1$ then

The moral: in big-Θ terms, the sum of a geometric series is simply the first term if the series is strictly decreasing, the last term if the series is strictly increasing, or the number of terms if the series is unchanging.

$$\begin{aligned}
g(n) &= 1 + c + c^2 + \cdots + c^n \\
cg(n) &= c + c^2 + \cdots + c^{n+1} \\
g(n) - cg(n) &= 1 - c^{n+1} \\
g(n)(1 - c) &= 1 - c^{n+1} \\
g(n) &= \frac{1 - c^{n+1}}{1 - c}
\end{aligned}$$

0.3(a) from text book

The Fibonacci numbers $F_0, F_1, F_2, \ldots,$ are defined by the rule

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}.$$

In this problem we will confirm that this sequence grows exponentially fast and obtain some bounds on its growth.

1. Use induction to prove that $F_n \geq 2^{0.5n}$ for $n \geq 6$.

2.3 from text book

Section 2.2 describes a method for solving recurrence relations which is based on analyzing the recursion tree and deriving a formula for the work done at each level. Another (closely related) method is to expand out the recurrence a few times, until a pattern emerges. For instance, lets start with the familiar $T(n) = 2T(n/2) + \mathrm{O}(n)$. Think of $\mathrm{O}(n)$ as being $\leq cn$ for some constant c, so: $T(n) \leq 2T(n/2) + cn$.

By repeatedly applying this rule, we can bound $T(n)$ in terms of $T(n/2)$, then $T(n/4)$, then $T(n/8)$, and so on, at each step getting closer to the value of $T(\cdot)$ we do know, namely $T(1) = \mathrm{O}(1)$.

$$\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&\leq 2[2T(n/4) + cn/2] + cn = 4T(n/4) + 2cn \\
&\leq 4[2T(n/8) + cn/4] + 2cn = 8T(n/8) + 3cn \\
&\leq 8[2T(n/16) + cn/8] + 3cn = 16T(n/16) + 4cn \\
&\ \ \vdots
\end{aligned}$$

A pattern is emerging... the general term is

$$T(n) \leq 2^k T(n/2^k) + kcn.$$

Plugging in $k = \log_2 n$, we get $T(n) \leq nT(1) + cn\log_2 n = \mathrm{O}(n \log n)$.

1. Do the same thing for the recurrence $T(n) = 3T(n/2) + \mathrm{O}(n)$. What is the general kth term in this case? And what value of k should be plugged in to get the answer?

2. Now try the recurrence $T(n) = T(n1) + \mathrm{O}(1)$, a case which is not covered by the master theorem. Can you solve this too?

2.4 from text book

Suppose you are choosing between the following three algorithms:

1. Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.

2. Algorithm B solves problems of size n by recursively solving two subproblems of size $n1$ and then combining the solutions in constant time.

3. Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $\mathrm{O}\left(n^2\right)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

2.17 from text book

Given a sorted array of distinct integers $A[1, \ldots, n]$, you want to find out whether there is an index i for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $\mathrm{O}\left(logn\right)$.