Homework 1, CS561, Fall 2014

Aaron Gonzales

September 3, 2014

1 exercise 3.1-5 CLRS - Prove theorem 3.1

2 Let $f(n) and g(n)$ be two functions that take on nonnegative values and assume $f(n) = O(g(n))$. Prove that $g(n) = \Omega(f(n))$.

By definition, $\Omega(g(n)) = \{f(n)$: there exist positive constants c, n_0 such that $0 \leq cg(n) \leq f(n) \forall n \geq n_0\}$.
By definition, $O(g(n)) = \{f(n)$: there exist positive constants c, n_0 such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0\}$.
if $f(n)$ is bounded above by $g(n)$, $g(n)$ must be at least equal to or lower than $f(n)$ for all $n \geq n_0$.

3 Problem 3-2 on Relative asymptotic growths

A	B	O	o	Ω	ω	Θ
lg^k	n^ϵ					
n^k	c^n					
$\sqrt{n}$	n^{sinn}					
2^n	$2^{n/2}$					
n^{lgc}	c^{lgn}					
$lg(n!)$	$lg(n^n)$					

4 Assume you have functions f and g, such that $f(n)$ is $O(g(n))$. for each of the following statements, decide wether you think it is true or false and give either a proof or a counterexample.

(a) $log_2 f(n) = O(log_2(g(n))$

counterexample: f(2) g(1).

(b) $2^{f(n)} = O(2^{g(n)})$

False. if $f(n) = 2n$ and

(c) $f(n)^2 = O(g(n)^2)$

True.
O is a loose upper bound. $0 \leq f(n) \leq g(n)$.

5 Problem 7-3: Alternative Quicksort analysis

(a) Argue that, given an array of size n, the probability that any particular element is chosen as the pivot is $1/n$. Use this to define indicator random variables $X_i = I\{i$ th smallest element is chosen as the pivot$\}$. What is $E[X_i]$?

$Pr(X = X_i) = 1/n$
Defining indicator variable X_i $\begin{bmatrix} 1{:}if pivot \\ 0{:}not pivot \end{bmatrix}$

$$\sum_1^n E(X_i) = X_i Pr(x = X_i) + X_2 Pr(x = X_2) + \cdots + X_n Pr(x = X_n)$$

(b)

Let $T(n)$ be a random variable denoting the running time of quicksort on an array of size n. Argue that

$$E[T(n)] = E\left[\sum_{q=1}^{n} X_q(T(q-1)) + T(n-q) + \Theta(n))\right]$$

The basic quicksort recurrence is taken from equation 7.1:

$$T(n) = T(Q) + (T(n-q-1) + \Theta(n)$$

if we define

$$Q' = q + 1$$

we can rewrite the recurrence as

$$T(n) = T(Q+1) + (T(n-Q') + \Theta(n)$$

and with our indicator variables...

$$T(n) = \sum_{Q'=1}^{n} \left(T(Q+1) + (T(n-Q') + \Theta(n)\right)$$

$$E\left[T(n)\right] = E\left[\sum_{Q'=1}^{n} X_q\left(T(Q+1) + (T(n-Q') + \Theta(n)\right)\right]$$

(c)

Show that we can rewrite equation 7.5 as:

$$E[T(n)] = \frac{2}{n}\sum_{q=2}^{n-1} E[T(q)] + \Theta(n)$$

.

$$E[T(n)] = E\left[\sum_{q=1}^{n} X_q(T(q-1)) + T(n-q) + \Theta(n))\right]$$

by LOE, we get

$$E[T(n)] = E\left[X_q \sum_{q=1}^{n} (T(q-1)) + T(n-q) + \Theta(n))\right]$$

(d)

show that

$$\sum_{k=2}^{n-1} k\,lg\,k \leq \frac{1}{2}n^2\,lg\,n - \frac{1}{8}n^2$$

.

(e)

Using the bound from equation 7.7, show that the recurrance in equation 7.6 has the solution $E[T(n)] = \Theta(nlgn)$.

6 Ladders

Imagine you are doing a stress test on a particular model of smart phones. you have a ladder with n rungs. You want ot determine the highest rung from which you cand rop a phone wihtout it breaking and you want to do it iwth the smallest number of phone drops.

(a) Imagine that you have exactly 2 phones. Devise an algorithm that can determine the highest safe rung using $o(n)$ drops. (little o).

Let it be stated that a ladder must have at least one one rung. If it has one rung, we only need one phone to test and see if that rung is safe (if it breaks, it's unsafe, if not, it's safe).

If a ladder has $n > 1$ rungs, we start by dropping the phone from the middle ladder rung ($n/2$). if the phone breaks, we search the bottom half of the ladder iteratively (first rung, second rung, etc.). If it doesn't break, we iteratively search the upper portion of the ladder.

We only test at most $\frac{n}{2}$ rungs, which $\frac{n}{2} < n \forall n \geq 2$. By definition, $o(g(n)) = \{f(n) : \text{for any positive constant } c > 0,$ there exisists a constant $n_0 > 0$ such that $0 \leq f(n) \leq cg(n) \, \forall \, n \geq n_0\}$. as such, the algorithm is $o(n)$.

(b) Now suppose you have k phones. Devise an algorithm that can determine the highest safe rung with the smallest number of drops. If $f_k(n)$ is the number of drops that your algorithm needs, what is $f_k(n)$ asymptotically? Hint: you shoudl ensure that $f_{k+1}(n) = o(f_k(n))$ for any k.

This is an example of binary search. If we assume that the rungs in the ladder are in a sorted order (and how could they not be?) the algorithm could be stated as follows:

As before, start with the middle rung of the ladder ($\frac{n}{2}$). If the phone dropped breaks, do the same procedure on the bottom half of the ladder. If it doesn't break, do the same procedure on the upper half of the ladder. each recursive step reduces the size of our search space by $\frac{1}{2}$. If for some reason we get to $k = 1$, then iteratively search the current subarray and return the rung prior to the rung on which the phone broke.

we can state this recurrence relation as:

$$T(n) = T(n/2) + C$$

And using the master theorem, where
$Case2 => T(n) = O(n^{\lceil logba \rceil} * (logn)^(k+1))$

7 The game of Match.

(a) If we shuffle the deck and turn over three cards, what is the probability that they form a match? Hint: given the first two cards, what is the probability that the third forms a match?

We know that given any two cards, only one more card can make a match. If we draw two cards, only one more card in the deck can make a match for the prior two cards, so we get out of our deck of $n = 27$ cards, $\frac{1}{25}$.

8 Drunken Debutants

9 Pairs on a circle