

CMPS 102 Solutions to Homework 2

Lindsay Brown, lbrown@soe.ucsc.edu

October 6, 2005

Problem 1. 3-2 p.58 Relative asymptotic growths

Indicate, for each pair of expressions (A, B) in the table below, whether A is O , o , Ω , ω , or Θ of B . Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants.

A	B	O	o	Ω	ω	Θ
$\lg^k n$	n^ϵ	yes	yes	no	no	no
n^k	c^n	yes	yes	no	no	no
\sqrt{n}	$n^{\sin n}$	no	no	no	no	no
2^n	$2^{n/2}$	no	no	yes	yes	no
$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes
$\lg(n!)$	$\lg(n^n)$	yes	no	yes	no	yes

L'Hopital's Rule: In the case where $f(n)$ and $g(n)$ are differentiable functions and $\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} g(n) = \infty$ the limit of $\frac{f(n)}{g(n)}$ can be computed as the limit of their derivatives:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}.$$

a.

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\lg^k n}{n^\epsilon} &= \frac{1}{\ln^k 2} \lim_{n \rightarrow \infty} \frac{\ln^k n}{n^\epsilon} \text{ (by changing the base of the logarithm)} \\ &= \frac{1}{\ln^k 2} \lim_{n \rightarrow \infty} \frac{k(\ln^{k-1} n)(1/n)}{\epsilon n^{\epsilon-1}} \text{ (by L'Hopital's Rule)} \\ &= \frac{1}{\ln^k 2} \lim_{n \rightarrow \infty} \frac{k \ln^{k-1} n}{\epsilon n^\epsilon} \text{ (by L'Hopital's Rule)} \end{aligned}$$

Applying L'Hopital's a total of k times:

$$\lim_{n \rightarrow \infty} \frac{\lg^k n}{n^\epsilon} = \frac{k!}{\epsilon^k \ln^k 2} \lim_{n \rightarrow \infty} \frac{1}{n^\epsilon} = 0.$$

b. $A(n) = n^k$ is a polynomial in n and $B(n) = c^n$ is exponential in n . By k -fold application of L'Hospital:

$$\lim_{n \rightarrow \infty} \frac{n^k}{c^n} = \lim_{n \rightarrow \infty} \frac{k!}{(\ln^k c)c^n} = 0.$$

c. $\sin n$ is a periodic function and takes values in the range $[-1,1]$. When $\sin n = 1$ we have

$$\lim_{n \rightarrow \infty} \frac{n}{n^{1/2}} = \infty.$$

However when $\sin n = -1$, we have

$$\lim_{n \rightarrow \infty} \frac{n^{-1}}{n^{1/2}} = 0.$$

d. $\lim_{n \rightarrow \infty} \frac{2^n}{2^{n/2}} = \lim_{n \rightarrow \infty} 2^{n/2} = \infty$.

e. $n^{\lg c} = c^{\lg n}$. To see this, take \lg of both sides:

$$(\lg c)(\lg n) = (\lg n)(\lg c).$$

f. $\lg n^n = n \lg n$

$\lg n! = \sum_{i=1}^n \lg i \leq \sum_{i=1}^n \lg n = n \lg n$ and thus $\lg n! = O(n \lg n)$.

$\lg n! = \sum_{i=1}^n \lg i \geq \sum_{i=n/2+1}^n \lg i \geq \sum_{i=n/2+1}^n \lg n/2 = n/2 \lg n/2 = (n/2 - 1) \lg n/2$ and thus $\lg n! = \Omega(n \lg n)$.

Finally $\lg n! = o(n \lg n)$ and $\lg n! = \omega(n \lg n)$ do not hold since by the above, the limit $\lim_{n \rightarrow \infty} \frac{\lg n!}{\lg n^n}$ lies in the interval $[1/2, 1]$.

Problem 2. 4.1-5 p.67 Proof by induction

Show that the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(n \lg n)$. Assume that $T(n) = 1$, for all $0 < n < 35$.

We need to show that for some $n_0 > 0$, and some fixed constant $c > 0$,

$$T(n) \leq cn \lg n$$

for all $n \geq n_0$. Let $n_0 = 35$.

If $T(n) \leq c(n-34) \lg n - n$ then $T(n) \leq cn \lg n$ by the transitivity of inequalities, because $c(n-34) \lg n - n \leq cn \lg n$.

Base case: Let $n = 35$.

$$T(35) = 2T(34) + 35 = 2 + 35 = 37$$

and $c(n-34) \lg n - n = c \lg 35 - 35 \geq 37$ when $c \geq 72/\lg 35$.

Inductive step: Suppose $T(k) \leq c(k - 34) \lg k - k$ for all $n_0 \leq k < n$.

$$\begin{aligned}
 T(n) &= 2T(\lfloor n/2 \rfloor + 17) + n \text{ given} \\
 &\leq 2[c(\lfloor n/2 \rfloor + 17 - 34) \lg(\lfloor n/2 \rfloor + 17) - n] + n \text{ applying the ind. hyp.} \\
 &= 2c(\lfloor n/2 \rfloor - 17) \lg(\lfloor n/2 \rfloor + 17) - 2n + n \\
 &= c(n - 34) \lg(\lfloor n/2 \rfloor + 17) - n \\
 &< c(n - 34) \lg n - n(\lfloor n/2 \rfloor + 17 < n)
 \end{aligned}$$

as desired. Therefore, $f(n) \in O(n \lg n)$.

Problem 3. 4.2-2 p.72 Recursion tree

Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + cn$, where c is a constant, is $\Omega(n \lg n)$ by appealing to a recursion tree.

Hopefully you drew a recursion tree and can visualize the following: The root node of the tree has cost cn . The root has two children, one has cost $c\frac{n}{3}$ and the other $c\frac{2n}{3}$. The node with cost $c\frac{n}{3}$ has two children, one has cost $c\frac{n}{9}$ and the other $c\frac{2n}{9}$. Similarly, the children of $c\frac{2n}{3}$ have costs $c\frac{2n}{9}$ and $c\frac{4n}{9}$. This pattern continues down the tree to the leaves.

We sum the cost of each row of the recursion tree. The 0^{th} row, the root, has cost cn . The first row has cost $c\frac{n}{3} + c\frac{2n}{3} = cn$. The third row has cost $c\frac{n}{9} + c\frac{2n}{9} + c\frac{2n}{9} + c\frac{4n}{9} = cn$. We notice that the pattern is that each row has cost cn .

We now need to determine the depth of the tree. Because this tree is unbalanced, the depth varies. The shortest path from root to leaf is $n \rightarrow \frac{n}{3} \rightarrow \frac{n}{9} \rightarrow \dots \rightarrow 1$. Since $(1/3)^k n = 1$ the length of this path is $k = \log_3 n$. Since this is the shortest path, it will give us a lower bound on the solution to the recurrence.

$$T(n) \geq \sum_{i=0}^{\log_3 n} cn = cn(\log_3 n + 1)$$

Therefore, $T(n) \in \Omega(n \lg n)$.

Problem 4. 4-4 a. and c. p.86 Applying the master theorem

The master theorem is stated in the text (Cormen) on page 73. The assumptions are $T(n) = aT(n/b) + f(n)$ where a and b are constants such that $a \geq 1$ and $b > 1$, and $f(n)$ is a function.

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

$a = 3$, $b = 2$, and $f(n) = n \lg n$. We have $n < n^{\log_2 3} < n^2$.

We want to show that case 1 applies. We need to find $\epsilon > 0$ so that $f(n) \in O(n^{\log_2 3 - \epsilon})$. Given $\delta > 0$, let $\epsilon = \log_2 3 - 1 - \delta$. Then

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n \lg n}{n^{\log_2 3 - \epsilon}} &= \lim_{n \rightarrow \infty} \frac{n \lg n}{n^{1+\delta}} \\ &= \lim_{n \rightarrow \infty} \frac{\lg n}{n^\delta} \end{aligned}$$

Since both the numerator and denominator are unbounded, we need to use L'Hopital's rule.

$$\lim_{n \rightarrow \infty} \frac{\lg n}{n^\delta} = \lim_{n \rightarrow \infty} \frac{n^{-1}}{\delta n^{\delta-1}} = \lim_{n \rightarrow \infty} \frac{1}{\delta n^\delta} = 0.$$

We have shown that $f(n) \in O(n^{\log_2 3 - \epsilon})$, so case 1 of the master theorem applies and $T(n) \in \Theta(n^{\log_2 3})$.

c. $T(n) = 4T(n/2) + n^2\sqrt{n}$
 $a = 4, b = 2$, and $f(n) = n^2\sqrt{n}$. We have $n^{\log_2 4} = n^2$ so we want to show that case 3 of the master theorem applies. We need to find ϵ so that $f(n) \in \Omega(n^{2+\epsilon})$. Since $f(n) = n^{2.5}$ any ϵ such that $0 < \epsilon < 1/2$ will work. There is another condition to case 3 of the master theorem that we must show is true, that is $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n . We have $4f(n/2) = 4(n/2)^2\sqrt{n/2} = \frac{n^2\sqrt{n}}{\sqrt{2}}$. When $1/\sqrt{2} < c < 1$, this condition is satisfied. So case 3 of the master theorem applies, and $T(n) \in \Theta(n^2\sqrt{n})$.

Problem 5. Finding the fake coin

An obvious solution would seem to be to divide the coins in half all the time. But note that there are three outcomes to a weighing - one side heavier, one side lighter, both sides equal. We can split the coins into 3 groups and we will know that if the first 2 are of equal weight then the fake coin must be in the third one. So, for every weighing we split the coins into 3 approximately equal groups. The first two should be exactly equal size and they are placed on the scale. The ways the coins can be split are listed below - assuming after every weighing the largest group is selected.

$$\begin{aligned} 70 &= 23 + 23 + 24 \\ 24 &= 8 + 8 + 7 \\ 8 &= 3 + 3 + 2 \\ 3 &= 1 + 1 + 1 \end{aligned}$$

With this method the fake coin is located in just 4 weighings, compared to 6 weighings for the binary method. (Kuzmin, 2003)

Extra credit: Finding an integer in a sorted array of unknown size

A binary search would work well, except that we don't know the upper index bound. But the idea of repeatedly halving a range to find an element can be turned around and used for finding an upper bound by repeatedly doubling the

current candidate. Thus we will start with a right array index of 1, then check 2, 4, 8 and so on. We stop once the array cell at that location is bigger than K - since the array is sorted K will be somewhere to the left. Once an upper bound is found, we can just run binary search as usual. To find the upper index bound we need $\lceil \lg n \rceil + 1$ comparisons in the worst case (we start at 1, plus an extra comparison for overshooting n). Note that in the worst case we will overshoot n by no more than a factor of two. Thus for binary search we will need $\lfloor \lg 2n \rfloor + 1 = \lfloor \lg n \rfloor + 3$ comparisons in the worst case. The total number of comparisons is about $2 \lg n$ which is definitely in $O(\lg n)$. (Kuzmin, 2003)