CPS 430/590.06 **Prof. John Reif**
Design and Analysis of Algorithms **Fall 2013**
Homework 1 **Matt Dickenson**

Question 1

From largest to smallest, the asymptotic size of the given functions are (with lg taken to represent log_2 and lg^{1000} taken to mean performing the lg operation one-thousand times):

$$lg(n^{1000}) \tag{1}$$
$$(lgn)^n \tag{2}$$
$$n^{lgn} \tag{3}$$
$$n^{lglgn} \tag{4}$$
$$(lgn)^{(lgn)} \tag{5}$$
$$n^2 \tag{6}$$
$$nlgn \tag{7}$$
$$n \tag{8}$$
$$lgn \tag{9}$$
$$n^{(1/lgn)} \tag{10}$$
$$(1+0.001)^n \tag{11}$$
$$lg_{1000}n \tag{12}$$
$$lg^{1000}n \tag{13}$$
$$1 \tag{14}$$

Question 2

The answers to this question rely on the discussion of the master method in chapter 4 of the CLRS textbook.

2.1 $T(n) = 2T(n/3) + 1$, so $a = 2, b = 3, f(n) = 1)$. Using the master method, this is case 1: $f(n) = O(n^{\log_3 2-\epsilon})$, so $\mathbf{T(n) = \Theta(n^{\log_3 2})}$.

2.2 $T(n) = 5T(n/4) + n$, so $a = 5, b = 4, f(n) = n$. Again this is case 1 of the master method: $f(n) = O(n^{\log_4 5-\epsilon})$, so $\mathbf{T(n) = \Theta(n^{\log_5 4})}$.

2.3 $T(n) = 8T(n/2) + n^3$, so $a = 8, b = 2, f(n) = n^3$. This is case 2 of the master method: $f(n) = \Theta(n^{\log_2 8})$, so $\mathbf{T(n) = \Theta(n^3 lgn)}$

2.4 $T(n) = T(n^{1/2}) + 1$. This recurrence relation cannot be directly translated into the form of the master method, so we use a change of variables $m = lgn$. $S(m) = S(m/2) + 1$, so $a_m = 1, b_m = 2, f(m) = 1$. This is case 3 of the master method, because $S(m) =$

$\Theta(m^{\log_2 1}) = \Theta(lgm)$. Changing back to our original variables, $T(n) = T(2^m) = S(m) = \Theta(lgm) = \mathbf{\Theta(lglgn)}$

Question 3

We wish to show that $T(n) = 2T(n/2) + O(n \log n)$ is $O(n \log^2 n$For this question we cannot use the Master Theorem, since $f(n) = O(n \log n)$ falls into one of that method's two gaps. However, what follows still uses the notation of the master method for coefficients: $a = 2, b = 2, f(n) = O(n \log n)$.

Using a recursion tree, we can write out the terms of the recursion as it proceeds toward the base case. At the first step we have the "glue cost" $O(n \log n)$. At the second step (the first recursion) we have two branches ($a = 2$) each of size $n/2$ ($b = 2$). In general, at the $(j-1)^{th}$ step our operation is of size $2^j(\frac{n}{2^j})$. At what level L will we reach the base case $n_j = 1$? We compute as follows:

$$
\begin{aligned}
\frac{n}{L} &= 1 \\
n &= b^L \\
\log_b n &= L \\
lgn &= L
\end{aligned}
$$

and see that the recursion stops at the $lg(n)^{th}$ step. This gives us the full cost of our recursion operations: $O(nlgn) + \sum_{j=0}^{lgn-1} 2^j \frac{n}{2^j}$. The solution to this is given by

$$
\begin{aligned}
T(n) &= O(nlgn) + \sum_{j=0}^{lgn-1} a^j \frac{n}{b^j} \\
&= O(nlg((\sum_{j=0}^{lgn-1} 2^j \frac{n}{2^j}))) \\
&= O(nlg(lg(n))) \\
&= O(n \log^2 n)
\end{aligned}
$$

Question 4

Suppose we label the three pegs P_1, P_2, and P_3. Without loss of generality, we can assume that initially all n pegs are on P_1 and our goal is to move them to P_3 under the constraints posed in the question. For convenience we also label the $n \geq 1$ discs $d_1, \ldots, d_n$ where d_1 is the smallest and d_n is the largest disc. Let $T(n)$ denote the number of steps required for n discs.

Our algorithm must first handle the base case, which is trivial: when $n = 1$, move d_n directly from P_1 to P_3. For any input n, this is the final step.

In general, we can solve the the Towers of Hanoi problem in three steps with recursion:

1. Move all discs $d_1, \ldots, d_{n-1}$ from P_1 to P_2

2. Move disc d_n from P_1 to P_3

3. Move discs $d_1, \ldots, d_{n-1}$ from P_2 to P_3.

Notice that by relabeling the discs that step 1 is equivalent to solving the problem at level $T(n-1)$ if our original goal was to move from P_1 to P_2. Similarly, step 3 is equivalent to $T(n-1)$ if the original goal was to move from P_2 to P_3. Step 2 will always take exactly 1 operation. This gives us the recurrence relation $\mathbf{T(n) = 2T(n-1) + 1}$.

We can get the total number of moves as a function of n using this fact. To help understand the recurrence relation it is helpful to examine a few cases with small n. We can easily see that in the base case $T(n=1) = 1$. Thus, $T(n=2) = 2(1) + 1 = 3$. Further,

$$\begin{aligned} T(n=3) &= 2(3) + 1 \\ &= 2(2(1)) + 2(1) + 1 \\ &= 2^2 + 2^1 + 2^0 \\ &= 7. \end{aligned}$$

More generally, $T(n)$ can be viewed as a summation series: $T(n) = \sum_{i=0}^{n-1} 2^i = 2^n - 1$. To see this, we use the recurrence relation $T(n) = 2T(n-1) + 1$ and see that at each step we add 2^{n-1} steps (this is also apparent from the result in Method 1 above). In the base case we already have a minimum of $2^1 - 1 = 1$ steps. For $n = 2$ we add $2^{2-1} = 2$ steps for a total of $2 + 1 = 3$ steps. In general, at the n^{th} step we are adding 2^{n-1} steps to a total of $2^{n-1} - 1$ steps from the $(n-1)^{th}$ iteration. Because $2^{n-1} + 2^{n-1} = 2 \times 2^{n-1} = 2^n$, we can simplify the series into the closed form equation $2^n - 1$ steps (see also: Equation A.6 in CLRS).

That is, in general for input of size n the Towers of Hanoi algorithm above requires $\mathbf{2^n - 1}$ **operations**.