Algorithms
By
Sanjoy Dasgupta Christos Papadimitriou Umesh Vazirani
Chapter 0: Prologue

Yukteshwar Baranwal

July 8, 2018

Problem 0.1: In each of the following situations, indicate whether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$).

Understanding:

1. $f = O(g)$ means that as n (the problem size tends to infinity) f is bounded above by g.
2. $f = \Omega(g)$ means that as n (the problem size tends to infinity) f is bounded below by g.
3. $f = \Theta(g)$ means that as n (the problem size tends to infinity) f is bounded above and below by g.

(a) $f(n) = n - 100$, $g(n) = n - 200 \Rightarrow f = \Theta(g)$.

(b) $f(n) = n^{1/2}$, $g(n) = n^{2/3} \Rightarrow f = O(g)$.

(c) $f(n) = 100n + log(n)$, $g(n) = n + (log(n))^2 \Rightarrow f = \Theta(g)$.
Since, $(log(n))^2 = O(n)$

(d) $f(n) = nlog(n)$, $g(n) = 10nlog(10n) \Rightarrow f = \Theta(g)$.

(e) $f(n) = log(2n)$, $g(n) = log(3n) \Rightarrow f = \Theta(g)$.

(f) $f(n) = 10log(n)$, $g(n) = log(n^2) \Rightarrow f = \Theta(g)$.

(g) $f(n) = n^{1.01}$, $g(n) = n(log(n))^2 \Rightarrow f = \Omega(g)$.

(h) $f(n) = n^2/log(n)$, $g(n) = n(log(n))^2 \Rightarrow f = \Omega(g)$.

(i) $f(n) = n^{0.1}$, $g(n) = (log(n))^{10} \Rightarrow f = \Omega(g)$.

(j) $f(n) = (log(n))^{log(n)}$, $g(n) = n/(log(n) \Rightarrow f = \Omega(g)$.

(k) $f(n) = \sqrt{n}$, $g(n) = (log(n))^3 \Rightarrow f = \Omega(g)$.

(l) $f(n) = n^{1/2}$, $g(n) = 5^{log_2(n)} \Rightarrow f = O(g)$.
Since, $5^{log_2(n)} = n^{log_2(5)} = n^{2.32} > n^{1/2}$

(m) $f(n) = n2^n$, $g(n) = 3^n \Rightarrow f = \Theta(g)$.
Since,

$$\begin{aligned}
log(f(n)) &= n + log(n) \\
log(g(n)) &= nlog(3) \\
\Rightarrow log(f(n)) &= \Theta(log(g(n))) \\
\Rightarrow f(n) &= \Theta(g(n))
\end{aligned}$$

(n) $f(n) = 2^n$, $g(n) = 2^{n+1} \Rightarrow f = \Theta(g)$.
Since, $2^{n+1} = 2 \times 2^n$.

(o) $f(n) = n!$, $g(n) = 2^n \Rightarrow f = \Omega(g)$.
Since, $n^n > n! > (n/2)^{n/2}$

(p) $f(n) = (log(n))^{log(n)}$, $g(n) = 2^{(log_2 n)^2} \Rightarrow f = O(g)$. Since,

$$\begin{aligned}
f(n) &= (log(n))^{log(n)} \\
&= (2^{log(log(n))})^{log(n)} \\
&= (2^{log(n)})^{log(log(n))} \\
&= n^{log(log(n))} \\
g(n) &= 2^{log(n)log(n)} \\
&= n^{log(n)}
\end{aligned}$$

(q) $f(n) = \sum_{i=1}^{n} i^k$, $g(n) = n^{k+1} \Rightarrow f = O(g)$.
Since, $f(n) = \sum_{i=1}^{n} i^k < n \times n^k$

Problem 0.2: Show that, if c is a positive real number, then $g(n) = 1 + c + c^2 + ... + c^n$ is:

(a) $\Theta(1)$ if $c < 1$
If $c < 1$, then $g(n)$ can be expressed as $\frac{1-c^{n+1}}{1-c}$. For higher values of n, it reaches to limiting value i.e., $\frac{1}{1-c}$. I can always get a constant number greater than this and lower bound is 1 thus it is $\Theta(1)$.

(b) $\Theta(n)$ if $c = 1$
If $c < 1$, then $g(n) = n = \Theta(n)$

(c) $\Theta(c^n)$ if $c > 1$
If $c > 1$, then $g(n)$ can be expressed as $\frac{c^{n+1}-1}{c-1}$. It is bounded from both sides by c^n functions and thus $\Theta(c^n)$.

Problem 0.3: The Fibonacci numbers F_0, F_1, F_2, ..., are defined by 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.

(a) Use induction to prove that $F_n \geq 2^{0.5n}$ for $n \geq 6$.
Base case: $F_6 = F_5 + F_4 = 5 + 3 = 8 \geq 2^{0.5\times6}$.

Similarily, $F_7 = F_6 + F_5 = 8 + 5 = 13 \geq 2^{0.5\times7}$.

Inductive hypothesis: For an index $k > 6$, if $F_k \geq 2^{0.5k}$, then $F_{k+1} \geq 2^{0.5(k+1)}$.

Inductive step: $F_{k+1} = F_k+F_{k-1} \geq 2^{0.5k}+2^{0.5(k-1)} = 2^{0.5k}\left(\frac{\sqrt{2}+1}{\sqrt{2}}\right) = 2^{0.5k}\sqrt{2}\left(\frac{\sqrt{2}+1}{2}\right) = 2^{0.5(k+1)}\left(\frac{\sqrt{2}+1}{2}\right) \geq 2^{0.5(k+1)}$

(b) Find a constant $c < 1$ such that $F_n \leq 2^{cn}$ for all $n \geq 0$. Show that your answer is correct.

$$\begin{aligned}
&F_n = F_{n-1} + F_{n-2} \leq 2^{cn} \\
&\Rightarrow 2^{c(n-1)} + 2^{c(n-2)} \leq 2^{cn} \\
&\Rightarrow 2^{-c} + 2^{-2c} \leq 1 \\
&\Rightarrow x^2 + x - 1 \leq 0
\end{aligned}$$

where $x = 2^{-c} > 0$. Solving for x.

$$\begin{aligned}
&0 \leq x \leq \frac{\sqrt{5}-1}{2} \\
&\Rightarrow 0 \leq 2^{-c} \leq \frac{\sqrt{5}-1}{2} \\
&\Rightarrow -\infty \leq -c \leq log(\sqrt{5}-1) - 1 \\
&\Rightarrow c \geq log(\sqrt{5}-1) - 1
\end{aligned}$$

Since, $c < 1$, hence $log(\sqrt{5}-1) - 1 \leq c < 1$.

(c) What is the largest c you can find for which $F_n = \Omega(2^{cn})$?
The minimum value of c in last problem would satisfy $F_n = \Omega(2^{cn})$. Hence, max value of c is $log(\sqrt{5}-1) - 1$.

Problem 0.4: Is there a faster way to compute the nth Fibonacci number than by *fib2*? Refer question in book.

(a) Show that two 2×2 matrices can be multiplied using 4 additions and 8 multiplications.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & ce+dg \\ af+bh & cf+dh \end{bmatrix}$$

Hence, there are 4 additions and 8 multiplications.

(b) Refer problem in book.
Computing X^n is multiplication by squaring. Recursively, if you have X^n, this is $(X^{n/2})^2$ if N is even or $X \times X^{n-1}$ if n is odd. Then apply this recursively. This alternates between halving and reducing the problem size by one at each iteration. So it will take $log(n)$ times.

(c) Refer problem in book.
Intermediate results are multiplication and addition of two numbers. We start with just one bit. When we multiply two 1 bit numbers and add them, we get at most 2 bits numbers. The general idea is by adding two numbers that have n-1 bits, we get at most a n bit number.

(d) Refer problem in book.
There are $log(n)$ iterations and each iteration has 8 multiplications with running time $M(n)$. Hence, the running time of algorithm is $O(M(n)log(n))$.

(e) Refer problem in book.
We can prove this using induction. Our base cases are F_0 and F_1, which clearly require $O(M(n))$ time. Now, assume that the claim holds for $1 \leq n < k$. If k is odd, then $F_k = F \times F_{\lfloor k/2 \rfloor} \times F_{\lfloor k/2 \rfloor}$. By the inductive hypothesis, determining $F_{\lfloor k/2 \rfloor}$ requires $O(M(k/2)) = O(M(k))$ time. Multiplying these arrays requires $O(M(k))$ time as well, so F_k can be found in $O(M(k))$ time. Similarly, F_k can be computed in $O(M(k))$ time if k is even. Therefore, the claim holds by induction.