Problem Set 1
Erdun E
September 10, 2024

CS 5800: Algorithm
Problem Set 1

Exercises

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

	f(n)	g(n)	Answer	Reason
(a)	n-100	n-200	f(n) = Θ(g(n))	Both are Θ(n)
(b)	$n^{1/2}$	$n^{2/3}$	f(n) = O(g(n))	1/2 < 2/3
(c)	100n + logn	n+$(logn)^2$	f(n) = Θ(g(n))	Both are Θ(n)
(d)	nlogn	10n logn 10n	f(n) = Θ(g(n))	Both are Θ(nlogn)
(e)	log2n	log 3n	f(n) = Θ(g(n))	Both are Θ(logn)
(f)	10logn	log(n^2)	f(n) = Θ(g(n))	Both are Θ(logn)
(g)	$n^{1.01}$	nlog^2n	f(n) = Ω(g(n))	$n^{1.01}$ growth faster than nlog^2n
(h)	$n^2/logn$	n$(logn)^2$	f(n) = Ω(g(n))	$n^2/logn$ is a square growth function
(i)	$n^{0.1}$	$(logn)^{10}$	f(n) = Ω(g(n))	Any polynomial = Ω() any log
(j)	$(logn)^{logn}$	n/logn	f(n) = Ω(g(n))	$2^{log_2 n} = n$ so $(logn)^{logn+1}$ = Ω(n)
(k)	$\sqrt{n}$	$(logn)^3$	f(n) = Ω(g(n))	Any polynomial = Ω() any log
(l)	$n^{1/2}$	$5^{log_2 n}$	f(n) = O(g(n))	$5^{log_2 n} = n^{log_2 5}$ then 1/2 < $log_2 5$
(m)	$n2^n$	3^n	f(n) = O(g(n))	2^n < 3^n
(n)	2^n	2^{n+1}	f(n) = Θ(g(n))	Both are Θ(n)
(o)	n!	2^n	f(n) = Ω(g(n))	n! > 2^n since n = 4
(p)	$(logn)^{logn}$	$2^{(log_2 n)^2}$	f(n) = O(g(n))	$2^{(log_2 n)^2}$ = n^2, f(logn) < polynomial
(q)	$\sum_{i=1}^{n} i^k$	n^{k+1}	f(n) = Θ(g(n))	$\sum_{i=1}^{n} i^k = \frac{n^{k+1}}{k+1}$, Both are Θ(n^{k+1})

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

(a) Θ(1)if c < 1.
(b) Θ(n)if c = 1.
(c) Θ(c^n)if c > 1.

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.

Answer:

Because c is a positive real number, the g(n)=1+c+c^2 +···+c^n is a sum of geometric series.

Based on the formula of the sum of geometric series we got: $S_n = \frac{a(r^{n+1}-1)}{r-1}$ and given r = c, a = 1

(a) Take the limit $\lim_{n\to\infty} \frac{g(n)}{1} = \lim_{n\to\infty} \frac{\sum_{i=0}^{n} c^i}{1} = \lim_{n\to\infty} \sum_{i=0}^{n} c^i = \frac{1}{1-c}$,

if c < 1, the limit is constant then g(n) = Θ(1)if c < 1

(b) if c = 1, g(n)=1+c+c^2 +···+c^n = $\sum_{i=0}^{n} c^i$ = n + 1 = Θ(n)

(c) For c > 1, the last term of the series can be used to find the sign of Θ domination for the entire series since this term is the largest. so g(n) $= \Theta(c^n)$

0.4. Is there a faster way to compute the nth Fibonacci number than by fib2 (page 13)? One idea involves matrices.
We start by writing the equations $F_1 = F_1$ and $F_2 = F_0 + F_1$ in matrix notation:

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Similarly,

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

and in general

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

So, in order to compute F_n, it suffices to raise this 2 $\times$ 2 matrix, call it *X*, to the *n*th power.
(a) Show that two 2 $\times$ 2 matrices can be multiplied using 4 additions and 8 multiplications.
But how many matrix multiplications does it take to compute X^n?
(b) Show that $O(\text{logn})$ matrix multiplications suffice for computing X^n. (*Hint:*Think about computing X^8.)
Thus the number of arithmetic operations needed by our matrix-based algorithm, call it fib3, is just $O(\text{log n})$, as compared to $O(\text{n})$ for fib2. *Have we broken another exponential barrier?*
The catch is that our new algorithm involves multiplication, not just addition; and multiplica- tions of large numbers are slower than additions. We have already seen that, when the complex- ity of arithmetic operations is taken into account, the running time of fib2 becomes O(n^2).

Answer:

(a) Set up two 2X2 matrices A and B is:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

then A X B = C is given by:

$$C = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

So, requires 4 additions and 8 multiplications

(b) For how many matrix multiplication does it take to compute X^n, would be multiply X by itself n - 1 times, meanwhile X is a 2X2 matrix, this will takes n - 1 times multiplications.

This means to compute X^n, requires 4 X (n - 1) additions, and 8 X (n - 1) multiplications.

Using exponentiation by squaring can reduce the count of matrix multiplications, this method takes $O(logn)$ matrix multiplications:

$$X^n = 1,\ if\ n = 0 \tag{1}$$
$$X^n = X,\ if\ n = 1 \tag{2}$$
$$X^n = X^{2/n} \cdot X^{2/n},\ if\ n\ is\ even \tag{3}$$
$$X^n = X \cdot X^{2/n} \cdot X^{2/n},\ if\ n\ is\ odd \tag{4}$$