Exercises

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

	$f(n)$	$g(n)$
(a)	$n - 100$	$n - 200$
(b)	$n^{1/2}$	$n^{2/3}$
(c)	$100n + \log n$	$n + (\log n)^2$
(d)	$n \log n$	$10n \log 10n$
(e)	$\log 2n$	$\log 3n$
(f)	$10 \log n$	$\log(n^2)$
(g)	$n^{1.01}$	$n \log^2 n$
(h)	$n^2 / \log n$	$n(\log n)^2$
(i)	$n^{0.1}$	$(\log n)^{10}$
(j)	$(\log n)^{\log n}$	$n / \log n$
(k)	$\sqrt{n}$	$(\log n)^3$
(l)	$n^{1/2}$	$5^{\log_2 n}$
(m)	$n2^n$	3^n
(n)	2^n	2^{n+1}
(o)	$n!$	2^n
(p)	$(\log n)^{\log n}$	$2^{(\log_2 n)^2}$
(q)	$\sum_{i=1}^{n} i^k$	n^{k+1}

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

(a) $\Theta(1)$ if $c < 1$.

(b) $\Theta(n)$ if $c = 1$.

(c) $\Theta(c^n)$ 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.

0.3. The Fibonacci numbers $F_0, F_1, F_2, \ldots$, are defined by the 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 $n \geq 6$.

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

(c) What is the largest c you can find for which $F_n = \Omega(2^{cn})$?

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 2×2 matrix, call it X, to the nth power.

(a) Show that two 2×2 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 $O(\log n)$ 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 $O(\log n)$, as compared to $O(n)$ for `fib2`. *Have we broken another exponential barrier?*

The catch is that our new algorithm involves multiplication, not just addition; and multiplications of large numbers are slower than additions. We have already seen that, when the complexity of arithmetic operations is taken into account, the running time of `fib2` becomes $O(n^2)$.

(c) Show that all intermediate results of `fib3` are $O(n)$ bits long.

(d) Let $M(n)$ be the running time of an algorithm for multiplying n-bit numbers, and assume that $M(n) = O(n^2)$ (the school method for multiplication, recalled in Chapter 1, achieves this). Prove that the running time of `fib3` is $O(M(n) \log n)$.

(e) Can you prove that the running time of `fib3` is $O(M(n))$? (*Hint:* The lengths of the numbers being multiplied get doubled with every squaring.)

In conclusion, whether `fib3` is faster than `fib2` depends on whether we can multiply n-bit integers faster than $O(n^2)$. Do you think this is possible? (The answer is in Chapter 2.)

Finally, there is a formula for the Fibonacci numbers:

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

So, it would appear that we only need to raise a couple of numbers to the nth power in order to compute F_n. The problem is that these numbers are irrational, and computing them to sufficient accuracy is nontrivial. In fact, our matrix method `fib3` can be seen as a roundabout way of raising these irrational numbers to the nth power. If you know your linear algebra, you should see why. (*Hint:* What are the eigenvalues of the matrix X?)