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:

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

Exercises

2.1. Use the divide-and-conquer integer multiplication algorithm to multiply the two binary integers 10011011 and 10111010.

2.2. Show that for any positive integers n and any base b, there must some power of b lying in the range $[n, bn]$.

2.3. Section 2.2 describes a method for solving recurrence relations which is based on analyzing the recursion tree and deriving a formula for the work done at each level. Another (closely related) method is to expand out the recurrence a few times, until a pattern emerges. For instance, let's start with the familiar $T(n) = 2T(n/2) + O(n)$. Think of $O(n)$ as being $\leq cn$ for some constant c, so: $T(n) \leq 2T(n/2) + cn$. By repeatedly applying this rule, we can bound $T(n)$ in terms of $T(n/2)$, then $T(n/4)$, then $T(n/8)$, and so on, at each step getting closer to the value of $T(\cdot)$ we do know, namely $T(1) = O(1)$.

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &\leq 2[2T(n/4) + cn/2] + cn \;=\; 4T(n/4) + 2cn \\ &\leq 4[2T(n/8) + cn/4] + 2cn \;=\; 8T(n/8) + 3cn \\ &\leq 8[2T(n/16) + cn/8] + 3cn \;=\; 16T(n/16) + 4cn \\ &\vdots \end{aligned}$$

A pattern is emerging... the general term is

$$T(n) \leq 2^k T(n/2^k) + kcn.$$

Plugging in $k = \log_2 n$, we get $T(n) \leq nT(1) + cn \log_2 n = O(n \log n)$.

(a) Do the same thing for the recurrence $T(n) = 3T(n/2) + O(n)$. What is the general kth term in this case? And what value of k should be plugged in to get the answer?

(b) Now try the recurrence $T(n) = T(n-1) + O(1)$, a case which is not covered by the master theorem. Can you solve this too?

2.4. Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

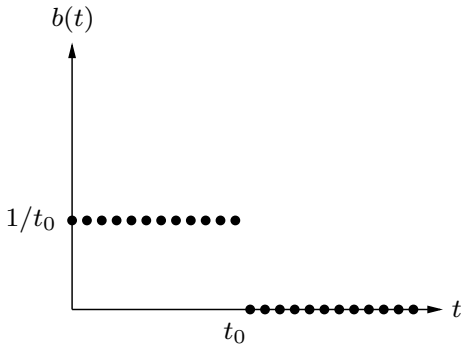
2.5. Solve the following recurrence relations and give a Θ bound for each of them.

(a) $T(n) = 2T(n/3) + 1$

(b) $T(n) = 5T(n/4) + n$

(c) $T(n) = 7T(n/7) + n$
(d) $T(n) = 9T(n/3) + n^2$
(e) $T(n) = 8T(n/2) + n^3$
(f) $T(n) = 49T(n/25) + n^{3/2} \log n$
(g) $T(n) = T(n-1) + 2$
(h) $T(n) = T(n-1) + n^c$, where $c \geq 1$ is a constant
(i) $T(n) = T(n-1) + c^n$, where $c > 1$ is some constant
(j) $T(n) = 2T(n-1) + 1$
(k) $T(n) = T(\sqrt{n}) + 1$

2.6. A linear, time-invariant system has the following impulse response:



(a) Describe in words the effect of this system.
(b) What is the corresponding polynomial?

2.7. What is the sum of the nth roots of unity? What is their product if n is odd? If n is even?

2.8. Practice with the fast Fourier transform.

(a) What is the FFT of $(1, 0, 0, 0)$? What is the appropriate value of ω in this case? And of which sequence is $(1, 0, 0, 0)$ the FFT?
(b) Repeat for $(1, 0, 1, -1)$.

2.9. Practice with polynomial multiplication by FFT.

(a) Suppose that you want to multiply the two polynomials $x + 1$ and $x^2 + 1$ using the FFT. Choose an appropriate power of two, find the FFT of the two sequences, multiply the results componentwise, and compute the inverse FFT to get the final result.
(b) Repeat for the pair of polynomials $1 + x + 2x^2$ and $2 + 3x$.

2.10. Find the unique polynomial of degree 4 that takes on values $p(1) = 2$, $p(2) = 1$, $p(3) = 0$, $p(4) = 4$, and $p(5) = 0$. Write your answer in the coefficient representation.

2.11. In justifying our matrix multiplication algorithm (Section 2.5), we claimed the following block-wise property: if X and Y are $n \times n$ matrices, and

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

where A, B, C, D, E, F, G, and H are $n/2 \times n/2$ submatrices, then the product XY can be expressed in terms of these blocks:

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Prove this property.

2.12. How many lines, as a function of n (in $\Theta(\cdot)$ form), does the following program print? Write a recurrence and solve it. You may assume n is a power of 2.

```
function f(n)
  if n > 1:
    print_line(''still going'')
    f(n/2)
    f(n/2)
```

2.13. A binary tree is *full* if all of its vertices have either zero or two children. Let B_n denote the number of full binary trees with n vertices.

(a) By drawing out all full binary trees with 3, 5, or 7 vertices, determine the exact values of B_3, B_5, and B_7. Why have we left out even numbers of vertices, like B_4?

(b) For general n, derive a recurrence relation for B_n.

(c) Show by induction that B_n is $\Omega(2^n)$.

2.14. You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time $O(n \log n)$.

2.15. In our median-finding algorithm (Section 2.4), a basic primitive is the `split` operation, which takes as input an array S and a value v and then divides S into three sets: the elements less than v, the elements equal to v, and the elements greater than v. Show how to implement this `split` operation *in place*, that is, without allocating new memory.

2.16. You are given an infinite array $A[\cdot]$ in which the first n cells contain integers in sorted order and the rest of the cells are filled with ∞. You are *not* given the value of n. Describe an algorithm that takes an integer x as input and finds a position in the array containing x, if such a position exists, in $O(\log n)$ time. (If you are disturbed by the fact that the array A has infinite length, assume instead that it is of length n, but that you don't know this length, and that the implementation of the array data type in your programming language returns the error message ∞ whenever elements $A[i]$ with $i > n$ are accessed.)

2.17. Given a sorted array of distinct integers $A[1, \ldots, n]$, you want to find out whether there is an index i for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$.

2.18. Consider the task of searching a sorted array $A[1 \ldots n]$ for a given element x: a task we usually perform by binary search in time $O(\log n)$. Show that any algorithm that accesses the array only via comparisons (that is, by asking questions of the form "is $A[i] \leq z$?"), must take $\Omega(\log n)$ steps.

2.19. *A k-way merge operation.* Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

(a) Here's one strategy: Using the `merge` procedure from Section 2.3, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n?

(b) Give a more efficient solution to this problem, using divide-and-conquer.

2.20. Show that any array of integers $x[1 \ldots n]$ can be sorted in $O(n + M)$ time, where

$$M = \max_i x_i - \min_i x_i.$$

For small M, this is linear time: why doesn't the $\Omega(n \log n)$ lower bound apply in this case?

2.21. *Mean and median.* One of the most basic tasks in statistics is to summarize a set of observations $\{x_1, x_2, \ldots, x_n\} \subseteq \mathbb{R}$ by a single number. Two popular choices for this summary statistic are:

- The median, which we'll call μ_1
- The mean, which we'll call μ_2

(a) Show that the median is the value of μ that minimizes the function

$$\sum_i |x_i - \mu|.$$

You can assume for simplicity that n is odd. (*Hint:* Show that for any $\mu \neq \mu_1$, the function decreases if you move μ either slightly to the left or slightly to the right.)

(b) Show that the mean is the value of μ that minimizes the function

$$\sum_i (x_i - \mu)^2.$$

One way to do this is by calculus. Another method is to prove that for any $\mu \in \mathbb{R}$,

$$\sum_i (x_i - \mu)^2 = \sum_i (x_i - \mu_2)^2 + n(\mu - \mu_2)^2.$$

Notice how the function for μ_2 penalizes points that are far from μ much more heavily than the function for μ_1. Thus μ_2 tries much harder to be close to *all* the observations. This might sound like a good thing at some level, but it is statistically undesirable because just a few outliers can severely throw off the estimate of μ_2. It is therefore sometimes said that μ_1 is a more robust estimator than μ_2. Worse than either of them, however, is μ_∞, the value of μ that minimizes the function

$$\max_i |x_i - \mu|.$$

(c) Show that μ_∞ can be computed in $O(n)$ time (assuming the numbers x_i are small enough that basic arithmetic operations on them take unit time).

2.22. You are given two sorted lists of size m and n. Give an $O(\log m + \log n)$ time algorithm for computing the kth smallest element in the union of the two lists.

2.23. An array $A[1 \ldots n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However you *can* answer questions of the form: "is $A[i] = A[j]$?" in constant time.

2.26. Professor F. Lake tells his class that it is asymptotically faster to square an n-bit integer than to multiply two n-bit integers. Should they believe him?

2.27. The *square* of a matrix A is its product with itself, AA.

(a) Show that five multiplications are sufficient to compute the square of a 2×2 matrix.

(b) What is wrong with the following algorithm for computing the square of an $n \times n$ matrix?

> "Use a divide-and-conquer approach as in Strassen's algorithm, except that instead of getting 7 subproblems of size $n/2$, we now get 5 subproblems of size $n/2$ thanks to part (a). Using the same analysis as in Strassen's algorithm, we can conclude that the algorithm runs in time $O(n^{\log_2 5})$."

(c) In fact, squaring matrices is no easier than matrix multiplication. In this part, you will show that if $n \times n$ matrices can be squared in time $S(n) = O(n^c)$, then any two $n \times n$ matrices can be multiplied in time $O(n^c)$.

i. Given two $n \times n$ matrices A and B, show that the matrix $AB + BA$ can be computed in time $3S(n) + O(n^2)$.

ii. Given two $n \times n$ matrices X and Y, define the $2n \times 2n$ matrices A and B as follows:

$$A = \begin{bmatrix} X & 0 \\ 0 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix}.$$

What is $AB + BA$, in terms of X and Y?

iii. Using (i) and (ii), argue that the product XY can be computed in time $3S(2n) + O(n^2)$. Conclude that matrix multiplication takes time $O(n^c)$.

2.28. The *Hadamard matrices* $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix $\begin{bmatrix} 1 \end{bmatrix}$
- For $k > 0$, H_k is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

Show that if v is a column vector of length $n = 2^k$, then the matrix-vector product $H_k v$ can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time.

2.29. Suppose we want to evaluate the polynomial $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ at point x.

(a) Show that the following simple routine, known as *Horner's rule*, does the job and leaves the answer in z.

```
z = a_n
for i = n − 1 downto 0:
    z = zx + a_i
```

(b) How many additions and multiplications does this routine use, as a function of n? Can you find a polynomial for which an alternative method is substantially better?

2.30. This problem illustrates how to do the Fourier Transform (FT) in modular arithmetic, for example, modulo 7.