1.7 Exercises

Reinforcement

R-1.1 Graph the functions $12n$, $6n\log n$, n^2, n^3, and 2^n using a logarithmic scale for the x- and y-axes; that is, if the function value $f(n)$ is y, plot this as a point with x-coordinate at $\log n$ and y-coordinate at $\log y$.

R-1.2 Algorithm A uses $10n\log n$ operations, while algorithm B uses n^2 operations. Determine the value n_0 such that A is better than B for $n \geq n_0$.

R-1.3 Repeat the previous problem assuming B uses $n\sqrt{n}$ operations.

R-1.4 Show that $\log^3 n$ is $o(n^{1/3})$.

R-1.5 Show that the following two statements are equivalent:

(a) The running time of algorithm A is $O(f(n))$.

(b) In the worst case, the running time of algorithm A is $O(f(n))$.

R-1.6 Order the following list of functions by the big-Oh notation. Group together (for example, by underlining) those functions that are big-Theta of one another.

$6n\log n$	2^{100}	$\log\log n$	$\log^2 n$	$2^{\log n}$
2^{2^n}	$\lceil\sqrt{n}\rceil$	$n^{0.01}$	$1/n$	$4n^{3/2}$
$3n^{0.5}$	$5n$	$\lfloor 2n\log^2 n\rfloor$	2^n	$n\log_4 n$
4^n	n^3	$n^2\log n$	$4^{\log n}$	$\sqrt{\log n}$

Hint: When in doubt about two functions $f(n)$ and $g(n)$, consider $\log f(n)$ and $\log g(n)$ or $2^{f(n)}$ and $2^{g(n)}$.

R-1.7 For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t assuming that the algorithm to solve the problem takes $f(n)$ microseconds. Recall that $\log n$ denotes the logarithm in base 2 of n. Some entries have already been completed to get you started.

	1 Second	1 Hour	1 Month	1 Century
$\log n$	$\approx 10^{300000}$			
$\sqrt{n}$				
n				
$n\log n$				
n^2				
n^3				
2^n				
$n!$		12		

R-1.8 Bill has an algorithm, find2D, to find an element x in an $n \times n$ array A. The algorithm find2D iterates over the rows of A and calls the algorithm arrayFind, of Algorithm 1.12, on each one, until x is found or it has searched all rows of A. What is the worst-case running time of find2D in terms of n? Is this a linear-time algorithm? Why or why not?

R-1.9 Consider the following recurrence equation, defining $T(n)$, as

$$T(n) = \begin{cases} 4 & \text{if } n = 1 \\ T(n-1) + 4 & \text{otherwise.} \end{cases}$$

Show, by induction, that $T(n) = 4n$.

R-1.10 Give a big-Oh characterization, in terms of n, of the running time of the Loop1 method shown in Algorithm 1.22.

R-1.11 Perform a similar analysis for method Loop2 shown in Algorithm 1.22.

R-1.12 Perform a similar analysis for method Loop3 shown in Algorithm 1.22.

R-1.13 Perform a similar analysis for method Loop4 shown in Algorithm 1.22.

R-1.14 Perform a similar analysis for method Loop5 shown in Algorithm 1.22.

```
Algorithm Loop1($n$):
  $s \leftarrow 0$
  for $i \leftarrow 1$ to $n$ do
    $s \leftarrow s + i$

Algorithm Loop2($n$):
  $p \leftarrow 1$
  for $i \leftarrow 1$ to $2n$ do
    $p \leftarrow p \cdot i$

Algorithm Loop3($n$):
  $p \leftarrow 1$
  for $i \leftarrow 1$ to $n^2$ do
    $p \leftarrow p \cdot i$

Algorithm Loop4($n$):
  $s \leftarrow 0$
  for $i \leftarrow 1$ to $2n$ do
    for $j \leftarrow 1$ to $i$ do
      $s \leftarrow s + i$

Algorithm Loop5($n$):
  $s \leftarrow 0$
  for $i \leftarrow 1$ to $n^2$ do
    for $j \leftarrow 1$ to $i$ do
      $s \leftarrow s + i$
```

Algorithm 1.22: A collection of loop methods.

R-1.15 Show that if $f(n)$ is $O(g(n))$ and $d(n)$ is $O(h(n))$, then the summation $f(n)+d(n)$ is $O(g(n)+h(n))$.

R-1.16 Show that $O(\max\{f(n),g(n)\}) = O(f(n)+g(n))$.

R-1.17 Show that $f(n)$ is $O(g(n))$ if and only if $g(n)$ is $\Omega(f(n))$.

R-1.18 Show that if $p(n)$ is a polynomial in n, then $\log p(n)$ is $O(\log n)$.

R-1.19 Show that $(n+1)^5$ is $O(n^5)$.

R-1.20 Show that 2^{n+1} is $O(2^n)$.

R-1.21 Show that n is $o(n\log n)$.

R-1.22 Show that n^2 is $\omega(n)$.

R-1.23 Show that $n^3\log n$ is $\Omega(n^3)$.

R-1.24 Show that $\lceil f(n)\rceil$ is $O(f(n))$ if $f(n)$ is a positive nondecreasing function that is always greater than 1.

R-1.25 Justify the fact that if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then the product $d(n)e(n)$ is $O(f(n)g(n))$.

R-1.26 What is the amortized running time of an operation in a series of n add operations on an initially empty extendable table implemented with an array such that the capacityIncrement parameter is always maintained to be $\lceil\log(m+1)\rceil$, where m is the number of elements of the stack? That is, each time the table is expanded by $\lceil\log(m+1)\rceil$ cells, its capacityIncrement is reset to $\lceil\log(m'+1)\rceil$ cells, where m is the old size of the table and m' is the new size (in terms of actual elements present).

R-1.27 Describe a recursive algorithm for finding both the minimum and the maximum elements in an array A of n elements. Your method should return a pair (a,b), where a is the minimum element and b is the maximum. What is the running time of your method?

R-1.28 Rewrite the proof of Theorem 1.31 under the assumption that the the cost of growing the array from size k to size $2k$ is $3k$ cyber-dollars. How much should each add operation be charged to make the amortization work?

R-1.29 Plot on a semi-log scale, using the ratio test, the comparison of the set of points

$$S = \{(1,1),(2,7),(4,30),(8,125),(16,510),(32,2045),(64,8190)\}$$

against each of the following functions:

a. $f(n) = n$
b. $f(n) = n^2$
c. $f(n) = n^3$.

R-1.30 Plot on a log-log scale the set of points

$$S = \{(1,1),(2,7),(4,30),(8,125),(16,510),(32,2045),(64,8190)\}.$$

Using the power rule, estimate a polynomial function $f(n) = bn^c$ that best fits this data.

Creativity

C-1.1 What is the amortized running time of the operations in a sequence of n operations $P = p_1p_2 \ldots p_n$ if the running time of p_i is $\Theta(i)$ if i is a multiple of 3, and is constant otherwise?

C-1.2 Let $P = p_1p_2 \ldots p_n$ be a sequence of n operations, each either a red or blue operation, with p_1 being a red operation and p_2 being a blue operation. The running time of the blue operations is always constant. The running time of the first red operation is constant, but each red operation p_i after that runs in time that is twice as long as the previous red operation, p_j (with $j < i$). What is the amortized time of the red and blue operations under the following conditions?

a. There are always $\Theta(1)$ blue operations between consecutive red operations.
b. There are always $\Theta(\sqrt{n})$ blue operations between consecutive red operations.
c. The number of blue operations between a red operation p_i and the previous red operation p_j is always twice the number between p_j and its previous red operation.

C-1.3 What is the total running time of counting from 1 to n in binary if the time needed to add 1 to the current number i is proportional to the number of bits in the binary expansion of i that must change in going from i to $i+1$?

C-1.4 Consider the following recurrence equation, defining a function $T(n)$:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{otherwise,} \end{cases}$$

Show, by induction, that $T(n) = n(n+1)/2$.

C-1.5 Consider the following recurrence equation, defining a function $T(n)$:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 2^n & \text{otherwise,} \end{cases}$$

Show, by induction, that $T(n) = 2^{n+1} - 1$.

C-1.6 Consider the following recurrence equation, defining a function $T(n)$:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n-1) & \text{otherwise,} \end{cases}$$

Show, by induction, that $T(n) = 2^n$.

C-1.7 Al and Bill are arguing about the performance of their sorting algorithms. Al claims that his $O(n \log n)$-time algorithm is *always* faster than Bill's $O(n^2)$-time algorithm. To settle the issue, they implement and run the two algorithms on many randomly generated data sets. To Al's dismay, they find that if $n < 100$, the $O(n^2)$-time algorithm actually runs faster, and only when $n \geq 100$ is the $O(n \log n)$-time algorithm better. Explain why this scenario is possible. You may give numerical examples.

C-1.8 Communication security is extremely important in computer networks, and one way many network protocols achieve security is to encrypt messages. Typical *cryptographic* schemes for the secure transmission of messages over such networks are based on the fact that no efficient algorithms are known for factoring large integers. Hence, if we can represent a secret message by a large prime number p, we can transmit over the network the number $r = p \cdot q$, where $q > p$ is another large prime number that acts as the *encryption key*. An eavesdropper who obtains the transmitted number r on the network would have to factor r in order to figure out the secret message p.

Using factoring to figure out a message is very difficult without knowing the encryption key q. To understand why, consider the following naive factoring algorithm:

> For every integer p such that $1 < p < r$, check if p divides r. If so, print "The secret message is p!" and stop; if not, continue.

a. Suppose that the eavesdropper uses the above algorithm and has a computer that can carry out in 1 microsecond (1 millionth of a second) a division between two integers of up to 100 bits each. Give an estimate of the time that it will take in the worst case to decipher the secret message if r has 100 bits.
b. What is the worst-case time complexity of the above algorithm? Since the input to the algorithm is just one large number r, assume that the input size n is the number of bytes needed to store r, that is, $n = (\log_2 r)/8$, and that each division takes time $O(n)$.

C-1.9 Give an example of a positive function $f(n)$ such that $f(n)$ is neither $O(n)$ nor $\Omega(n)$.

C-1.10 Show that $\sum_{i=1}^{n} i^2$ is $O(n^3)$.

C-1.11 Show that $\sum_{i=1}^{n} i/2^i < 2$.

Hint: Try to bound this sum term by term with a geometric progression.

C-1.12 Show that $\log_b f(n)$ is $\Theta(\log f(n))$ if $b > 1$ is a constant.

C-1.13 Describe a method for finding both the minimum and maximum of n numbers using fewer than $3n/2$ comparisons.

Hint: First construct a group of candidate minimums and a group of candidate maximums.

C-1.14 Suppose you are given a set of small boxes, numbered 1 to n, identical in every respect except that each of the first i contain a pearl whereas the remaining $n-i$ are empty. You also have two magic wands that can each test if a box is empty or not in a single touch, except that a wand disappears if you test it on an empty box. Show that, without knowing the value of i, you can use the two wands to determine all the boxes containing pearls using at most $o(n)$ wand touches. Express, as a function of n, the asymptotic number of wand touches needed.

C-1.15 Repeat the previous problem assuming that you now have k magic wands, with $k > 2$ and $k < \log n$. Express, as a function of n and k, the asymptotic number of wand touches needed to identify all the magic boxes containing pearls.

C-1.16 An n-degree ***polynomial*** $p(x)$ is an equation of the form

$$p(x) = \sum_{i=0}^{n} a_i x^i,$$

where x is a real number and each a_i is a constant.

a. Describe a simple $O(n^2)$ time method for computing $p(x)$ for a particular value of x.

b. Consider now a rewriting of $p(x)$ as

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n)\cdots))),$$

which is known as ***Horner's method***. Using the big-Oh notation, characterize the number of multiplications and additions this method of evaluation uses.

C-1.17 Consider the following induction "proof" that all sheep in a flock are the same color:

Base case: One sheep. It is clearly the same color as itself.

Induction step: A flock of n sheep. Take a sheep, a, out of the flock. The remaining $n-1$ are all the same color by induction. Now put sheep a back in the flock, and take out a different sheep, b. By induction, the $n-1$ sheep (now with a in their group) are all the same color. Therefore, a is the same color as all the other sheep; hence, all the sheep in the flock are the same color.

What is wrong with this "proof"?

C-1.18 Consider the following "proof" that the Fibonacci function, $F(n)$, defined as $F(1) = 1, F(2) = 2, F(n) = F(n-1) + F(n-2)$, is $O(n)$:
Base case ($n \leq 2$): $F(1) = 1$, which is $O(1)$, and $F(2) = 2$, which is $O(2)$.
Induction step ($n > 2$): Assume the claim is true for $n' < n$. Consider n. $F(n) = F(n-1) + F(n-2)$. By induction, $F(n-1)$ is $O(n-1)$ and $F(n-2)$ is $O(n-2)$. Then, $F(n)$ is $O((n-1)+(n-2))$, by the identity presented in Exercise R-1.15. Therefore, $F(n)$ is $O(n)$, since $O((n-1)+(n-2))$ is $O(n)$.
What is wrong with this "proof"?

C-1.19 Consider the Fibonacci function, $F(n)$, from the previous exercise. Show by induction that $F(n)$ is $\Omega((3/2)^n)$.

C-1.20 Draw a visual justification of Theorem 1.13 analogous to that of Figure 1.11b for the case when n is odd.

C-1.21 An array A contains $n-1$ unique integers in the range $[0, n-1]$, that is, there is one number from this range that is not in A. Design an $O(n)$-time algorithm for finding that number. You are allowed to use only $O(1)$ additional space besides the array A itself.

C-1.22 Show that the summation $\sum_{i=1}^{n} \lceil \log_2 i \rceil$ is $O(n \log n)$.

C-1.23 Show that the summation $\sum_{i=1}^{n} \lceil \log_2 i \rceil$ is $\Omega(n \log n)$.

C-1.24 Show that the summation $\sum_{i=1}^{n} \lceil \log_2 (n/i) \rceil$ is $O(n)$. You may assume that n is a power of 2.

Hint: Use induction to reduce the problem to that for $n/2$.

C-1.25 An evil king has a cellar containing n bottles of expensive wine, and his guards have just caught a spy trying to poison the king's wine. Fortunately, the guards caught the spy after he succeeded in poisoning only one bottle. Unfortunately, they don't know which one. To make matters worse, the poison the spy used was very deadly; just one drop diluted even a billion to one will still kill someone. Even so, the poison works slowly; it takes a full month for the person to die. Design a scheme that allows the evil king to determine exactly which one of his wine bottles was poisoned in just one month's time while expending at most $O(\log n)$ of his taste testers.

C-1.26 Let S be a set of n lines such that no two are parallel and no three meet in the same point. Show by induction that the lines in S determine $\Theta(n^2)$ intersection points.

C-1.27 Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Assuming A is already in memory, describe a method running in $O(n)$ time (not $O(n^2)$ time) for finding the row of A that contains the most 1's.

C-1.28 Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row i of A, all the 1's come before any 0's in that row. Suppose further that the number of 1's in row i is at least the number in row $i+1$, for $i = 0, 1, \ldots, n-2$. Assuming A is already in memory, describe a method running in $O(n)$ time (not $O(n^2)$ time) for counting the number of 1's in the array A.

C-1.29 Describe, using pseudo-code, a method for multiplying an $n \times m$ matrix A and an $m \times p$ matrix B. Recall that the product $C = AB$ is defined so that $C[i][j] = \sum_{k=1}^{m} A[i][k] \cdot B[k][j]$. What is the running time of your method?

C-1.30 Give a recursive algorithm to compute the product of two positive integers m and n using only addition.

C-1.31 Give complete pseudo-code for a new class, ShrinkingTable, that performs the add method of the extendable table, as well as methods, remove(), which removes the last (actual) element of the table, and shrinkToFit(), which replaces the underlying array with an array whose capacity is exactly equal to the number of elements currently in the table.

C-1.32 Consider an extendable table that supports both add and remove methods, as defined in the previous exercise. Moreover, suppose we grow the underlying array implementing the table by doubling its capacity any time we need to increase the size of this array, and we shrink the underlying array by half any time the number of (actual) elements in the table dips below $N/4$, where N is the current capacity of the array. Show that a sequence of n add and remove methods, starting from an array with capacity $N = 1$, takes $O(n)$ time.

C-1.33 Consider an implementation of the extendable table, but instead of copying the elements of the table into an array of double the size (that is, from N to $2N$) when its capacity is reached, we copy the elements into an array with $\lceil \sqrt{N} \rceil$ additional cells, going from capacity N to $N + \lceil \sqrt{N} \rceil$. Show that performing a sequence of n add operations (that is, insertions at the end) runs in $\Theta(n^{3/2})$ time in this case.

Projects

P-1.1 Program the two algorithms, prefixAverages1 and prefixAverages2 from Section 1.4, and perform a careful experimental analysis of their running times. Plot their running times as a function of their input sizes as scatter plots on both a linear-linear scale and a log-log scale. Choose representative values of the size n, and run at least five tests for each size value n in your tests.

P-1.2 Perform a careful experimental analysis that compares the relative running times of the methods shown in Algorithm 1.22. Use both the ratio test and the power test to estimate the running times of the various methods.

P-1.3 Implement an extendable table using arrays that can increase in size as elements are added. Perform an experimental analysis of each of the running times for performing a sequence of n add methods, assuming the array size is increased from N to the following possible values:

a. $2N$
b. $N + \lceil \sqrt{N} \rceil$
c. $N + \lceil \log N \rceil$
d. $N + 100$.

Chapter Notes

The topics discussed in this chapter come from diverse sources. Amortization has been used to analyze a number of different data structures and algorithms, but it was not a topic of study in its own right until the mid 1980's. For more information about amortization, please see the paper by Tarjan [201] or the book by Tarjan [200].

Our use of the big-Oh notation is consistent with most authors' usage, but we have taken a slightly more conservative approach than some. The big-Oh notation has prompted several discussions in the algorithms and computation theory community over its proper use [37, 92, 120]. Knuth [118, 120], for example, defines it using the notation $f(n) = O(g(n))$, but he refers to this "equality" as being only "one way," even though he mentions that the big-Oh is actually defining a set of functions. We have chosen to take a more standard view of equality and view the big-Oh notation truly as a set, following the suggestions of Brassard [37]. The reader interested in studying average-case analysis is referred to the book chapter by Vitter and Flajolet [207].

We include a number of useful mathematical facts in Appendix A. The reader interested in further study into the analysis of algorithms is referred to the books by Graham, Knuth, and Patashnik [90], and Sedgewick and Flajolet [184]. The reader interested in learning more about the history of mathematics is referred to the book by Boyer and Merzbach [35]. Our version of the famous story about Archimedes is taken from [155]. Finally, for more information about using experimentation to estimate the running time of algorithms, we refer the interested reader to several papers by McGeoch and coauthors [142, 143, 144].