1.6 Exercises

Reinforcement

EXERCISE 1.6.1

(a) 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 $\log y$.

Feedback?

EXERCISE 1.6.2 All solutions visible to students

(a) Show that the `MaxsubSlow` algorithm runs in $\Omega(n^3)$ time.

Solution Visible to students

The outer loop, for index *j*, makes *n* iterations. In *n/2* of those iterations (for $j < n/2$), the next-inner loop, for index *k*, makes at least *n/2* iterations. Finally, for *n/4* of those iterations (for $k > 3n/4$), the inner-most loop, for index *i*, makes at least *n/4* iterations. Thus, the `MaxsubSlow` algorithm uses at least $n(n/2)(n/4) = n^3/8$ steps, which is $\Omega(n^3)$.

Feedback?

EXERCISE 1.6.3

(a) 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 $n \geq n_0$.

Feedback?

EXERCISE 1.6.4

(a) Repeat the previous problem assuming B uses $n\sqrt{n}$ operations.

Feedback?

EXERCISE 1.6.5

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

Feedback?

EXERCISE 1.6.6

Show that the following two statements are equivalent:

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

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

Feedback?

EXERCISE 1.6.7 All solutions visible to students

(a) 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 $2^{g(n)}$.

Solution Visible to students

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

EXERCISE | 1.6.8 | All solutions visible to students

(a) 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		

Solution Visible to students

The numbers in the first row are quite large. The table below calculates it approximately in powers of 10. People might also choose to use powers of 2. Being close to the answer is enough for the big numbers (within a few factors of 10 from the answers shown).

	1 Second	1 Hour	1 Month	1 C
$\log n$	$2^{10^6} \approx 10^{300000}$	$2^{3.6\times10^9} \approx 10^{10^9}$	$2^{2.6\times10^{12}} \approx 10^{0.8\times10^{12}}$	$2^{3.1\times1}$
$\sqrt{n}$	$\approx 10^{12}$	$\approx 1.3 \times 10^{19}$	$\approx 6.8 \times 10^{24}$	≈ 9
n	10^6	3.6×10^9	$\approx 2.6 \times 10^{12}$	$\approx 3.$
$n \log n$	$\approx 10^5$	$\approx 10^9$	$\approx 10^11$	
n^2	1000	6×10^4	$\approx 1.6 \times 10^6$	≈ 5

n^3	100	≈ 1500	≈ 14000	$\approx$
2^n	19	31	41	
$n!$	9	12	15	

Feedback?

EXERCISE 1.6.9 All solutions visible to students

(a) 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 1.3.2, 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?

Solution Visible to students

The worst case running time of `find2D` is $o(n^2)$. This is seen by examining the worst case where the element *x* is the very last item in the *n* × *n* array to be examined. In this case, `find2d` calls the algorithm `arrayFind` n times. `arrayFind` will then have to search all n elements for each call until the final call when *x* is found. Therefore, *n* comparisons are done for each `arrayFind` call. Since `arrayFind` is called *n* times, we have $n * n$ operations, or an $o(n^2)$ running time. This is not a linear time algorithm; it is quadratic. If this were a linear time algorithm, the running time would be proportional to its input size.

Feedback?

EXERCISE 1.6.10

(a) 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 $T(n)= 4n$.

Feedback?

EXERCISE 1.6.11 All solutions visible to students

(a) Give a big-Oh characterization, in terms of n, of the running time of the `Loop1` method shown in Algorithm 1.6.1.

Solution Visible to students

The `Loop1` method runs in $O(n)$ time.

Feedback?

EXERCISE 1.6.12 All solutions visible to students

(a) Perform a similar analysis for method `Loop2` shown in Algorithm 1.6.1.

Solution Visible to students

The `Loop2` method runs in $O(n)$ time.

Feedback?

EXERCISE 1.6.13 All solutions visible to students

(a) Perform a similar analysis for method `Loop3` shown in Algorithm 1.6.1.

Solution Visible to students

The `Loop3` method runs in $O(n^2)$ time.

Feedback?

EXERCISE 1.6.14 All solutions visible to students

(a) Perform a similar analysis for method `Loop4` shown in Algorithm 1.6.1.

Solution Visible to students

The `Loop4` method runs in $O(n^2)$ time.

Feedback?

Figure 1.6.1: A collection of loop methods.

```
Algorithm Loop1(n):
    s ← 0
    for i ← 1 to n do
        s ← s + i

Algorithm Loop2(n):
    p ← 1
    for i ← 1 to 2n do
        p ← p · i

Algorithm Loop3(n):
    p ← 1
    for i ← 1 to n^2 do
        p ← p · i

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

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

Feedback?

EXERCISE | 1.6.15 | All solutions visible to students

(a) Perform a similar analysis for method Loop5 shown in Algorithm 1.6.1.

Solution Visible to students

The Loop5 method runs in $O(n^4)$ time.

Feedback?

EXERCISE | 1.6.16

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

Feedback?

EXERCISE | 1.6.17

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

Feedback?

EXERCISE | 1.6.18

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

Feedback?

EXERCISE | 1.6.19

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

Feedback?

EXERCISE | 1.6.20 | All solutions visible to students

(a) Show that $(n+1)^5$ is $O(n^5)$.

Solution ^ Visible to students

By the definition of big-Oh, we need to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $(n+1)^5 \leq c\,(n^5)$ for every integer $n \geq n_0$. Since

$(n+1)^5 = n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1$, $(n+1)^5 \leq c\left(n^5\right)$ for $c = 8$ and $n \geq n_0 = 2$.

Feedback?

EXERCISE 1.6.21 All solutions visible to students

(a) Show that 2^{n+1} is $O(2^n)$.

Solution Visible to students

By the definition of big-Oh, we need to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $2^{n+1} \leq c\left(2^n\right)$ for$n \geq n_0$. One possible solution is choosing $c = 2$ and $n_0 = 1$, since $2^{n+1} = 2 \cdot 2^n$.

Feedback?

EXERCISE 1.6.22

(a) Show that n is $o(n \ \log \ n)$.

Feedback?

EXERCISE 1.6.23

(a) Show that n^2 is $\omega(n)$.

Feedback?

EXERCISE 1.6.24 All solutions visible to students

(a) Show that $n^3 \ \log \ n$ is $\Omega(n^3)$.

Solution Visible to students

By the definition of big-Omega, we need to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $n^3 \log n \geq cn^3$ for $n \geq n_0$. Choosing $c = 1$ and $n_0 = 2$, shows $n^3 \log n \geq cn^3$ for $n \geq n_0$, since $\log n >\geq 1$ in this range.

Feedback?

EXERCISE 1.6.25

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

Feedback?

EXERCISE 1.6.26

(a) 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 $O(f(n)g(n))$.

Feedback?

EXERCISE 1.6.27

(a) Given the values of the maximum suffix sums, M_t $(t = 1, \cdots, 11)$, for the array $A = [-2, -4, 3, -1, 5, 6, -7, -2, 4, -3, 2]$.

Feedback?

EXERCISE 1.6.28

(a) 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).

Feedback?

EXERCISE 1.6.29

(a) 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?

Feedback?

EXERCISE 1.6.30

(a) Suppose you have an array of n numbers and you select each one independently with probability $1/n^{1/2}$. Use the Chernoff bound to determine an upper bound on the probability that you would have more than $4n^{1/2}$ elements in this random sample.

Feedback?

EXERCISE 1.6.31

(a) Rewrite the proof of Theorem 1.5.2 under the assumption that 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?

Feedback?

EXERCISE 1.6.32

(a) Suppose we have a set of n balls and we choose each one independently with probability $1/n^{1/2}$ to go into a basket. Derive an upper bound on the probability that there are more than $3n^{1/2}$ balls in the basket.

Feedback?

Creativity

EXERCISE 1.6.33

(a) Describe how to modify the description of the `MaxsubFastest` algorithm so that, in addition to the value of the maximum subarray summation, it also outputs the indices j and k that identify the maximum subarray $A[j:k]$.

Feedback?

EXERCISE 1.6.34

(a) Describe how to modify the `MaxsubFastest` algorithm so that it uses just a single loop and, instead of computing $n+1$ different M_t values, it maintains just a single variable M.

Feedback?

EXERCISE 1.6.35

(a) What is the amortized running time of the operations in a sequence of n operations $P = p_1 p_2 \ldots p_n$ if the running time of p_i is $\Theta(i)$ if i is a multiple of 3, and is constant otherwise?

Feedback?

EXERCISE 1.6.36

(a) 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 $i+1$?

Feedback?

EXERCISE 1.6.37

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

Feedback?

EXERCISE 1.6.38

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

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n-1) + 2^n & \text{otherwise,} \end{cases}$$

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

Feedback?

EXERCISE 1.6.39

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

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2T(n-1) & \text{otherwise,} \end{cases}$$

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

Feedback?

EXERCISE | 1.6.40 | All solutions visible to students

(a) 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.

Solution Visible to students

To say that Al's algorithm is "big-oh" of Bill's algorithm implies that Al's algorithm will run faster than Bill's for all input greater than some nonzero positive integer n_0. In this case, $n_0 = 100$.

Feedback?

EXERCISE | 1.6.41 | All solutions visible to students

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

Solution Visible to students

One possible solution is $f(n) = n^2 + (1 + \sin(n))$.

Feedback?

EXERCISE | 1.6.42 | All solutions visible to students

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

Solution Visible to students

$$\sum_{i=1}^{n} i^2 < \int_0^{n+1} x^2 dx < \frac{(n+1)^3}{3} = O\left(n^3\right)$$

Feedback?

EXERCISE | 1.6.43

(a) Show that $\sum_{i=1}^{n} i/2^i < 2$.
Hint: Try to bound this sum term by term with a geometric progression.

Feedback?

EXERCISE 1.6.44

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

Feedback?

EXERCISE 1.6.45

(a) 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.

Feedback?

EXERCISE 1.6.46

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.

Feedback?

EXERCISE | 1.6.47 | All solutions visible to students

(a) 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"?

Solution Visible to students

The induction assumes that the set of $n-1$ sheep without a and the set of $n-1$ sheep without b have sheep in common. Clearly this is not true with the case of 2 sheep. If a base case of 2 sheep could be shown, then the induction would be valid.

Feedback?

EXERCISE | 1.6.48

(a) 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 1.6.1. Therefore, $F(n)$ is $O(n)$, since $O((n-1)+(n-2))$ is $O(n)$.
- What is wrong with this "proof"?

Feedback?

EXERCISE | 1.6.49

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

Feedback?

EXERCISE 1.6.50

(a) Draw a visual justification of Theorem 1.3.2 analogous to that of Figure 1.3.1b for the case when n is odd.

Feedback?

EXERCISE 1.6.51 All solutions visible to students

(a) 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.

Solution Visible to students

First calculate the sum $\sum_{i=1}^{n-1} = \frac{n(n-1)}{2}$. Then calculate the sum of all values in the array A. The missing element is the difference between these two numbers.

Feedback?

EXERCISE 1.6.52 All solutions visible to students

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

Solution Visible to students

$\sum_{i=1} n \log_2 i < \sum_{i=1} n \log_2 n = n \log_2 n$

Feedback?

EXERCISE 1.6.53 All solutions visible to students

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

Solution Visible to students

For convenience assume that n is even. Then

$\sum_{i=1}^{n} \log_2 \ i / geq \sum_{i=\frac{n}{2}+1}^{n} \log_2 \ \frac{n}{2} = \frac{n}{2} \log_2 \ \frac{n}{2}$,

which is $\Omega(n \log n)$.

Feedback?

EXERCISE 1.6.54

(a) 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 $n/2$.

Feedback?

EXERCISE 1.6.55

(a) 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.

Feedback?

EXERCISE 1.6.56 All solutions visible to students

(a) 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.

Solution Visible to students

Start at the upper left of the matrix. Walk across the matrix until a 0 is found. Then walk down the matrix until a 1 is found. This is repeated until the last row or column is encountered. The row with the most 1's is the last row which was walked across.

Clearly this is an $O(n)$-time algorithm since at most $2 \cdot n$ comparisons are made.

Feedback?

EXERCISE | 1.6.57 | All solutions visible to students

(a) 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.

Solution Visible to students

Using the two properties of the array, the method is described as follows.

- Starting from element $A[n-1, 0]$, we scan A moving only to the right and upwards.
- If the number at $A[i, j]$ is 1, then we add the number of 1s in that column $(i+1)$ to the current total of 1s
- Otherwise we move up one position until we reach another 1.

The running time is $O(n)$. In the worst case, you will visit at most $2n - 1$ places in the array.

Feedback?

EXERCISE | 1.6.58

(a) Describe, using pseudocode, 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?

Feedback?

EXERCISE | 1.6.59

(a) Give a recursive algorithm to compute the product of two positive integers m and n using only addition.

Feedback?

EXERCISE 1.6.60

(a) Give complete pseudocode 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.

Feedback?

EXERCISE 1.6.61

(a) 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.

Feedback?

EXERCISE 1.6.62

(a) 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.

Feedback?

Applications

EXERCISE | 1.6.63 All solutions visible to students

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 whether 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 $O(n)$.

Solution Visible to students

Since r is represented with 100 bits, any candidate p that the eavesdropper might use to try to divide r uses also at most 100 bits. Thus, this very naive algorithm requires 2^{100} divisions, which would take about 2^{80} seconds, or at least 2^{55} years. Even if the eavesdropper uses the fact that a candidate p need not ever be more than 50 bits, the problem is still difficult. For in this case, 2^{50} divisions would take about 2^{30} seconds, or about 34 years.

Since each division takes time $O(n)$ and there are 2^{4n} total divisions, the asymptotic running time is $O\left(n \cdot 2^{4n}\right)$.

Feedback?

EXERCISE 1.6.64

(a) Program the three algorithms given in the chapter for the maximum subarray problem, 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. Note that the slope of a line plotted on a log-log scale is based on the exponent of a function, since $\log n^c = c \log n$.

Feedback?

EXERCISE 1.6.65

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) $N + 100$.

Feedback?

EXERCISE 1.6.66

(a) 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.

Feedback?

Note: All the remaining problems are inspired by questions reported to have been asked in job interviews for major software and Internet companies.

EXERCISE 1.6.67

(a) 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 whether 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.

Feedback?

EXERCISE 1.6.68

(a) 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.

Feedback?

EXERCISE 1.6.69 All solutions visible to students

(a) Suppose you are given an integer c and an array, A, indexed from 1 to n, of n integers in the range from 1 to $5n$ (possibly with duplicates). Describe an efficient algorithm for determining if there are two integers, $A[i]$ and $A[j]$, in A that sum to c, that is, such that $c = A[i]+A[j]$, for $1 \leq i < j \leq n$. What is the running time of your algorithm?

Solution Visible to students

We can rewrite the equation as $A[j] = c - A[i]$. Create a Boolean array, B, indexed from 0 to $10n$, all of whose elements are initially **false**. For each element, $A[i]$, if

$c - A[i] > 0$, set $B[c - A[i]]$ to **true**. Then, for each $A[j]$ in A, check if $B[A[j]]$ is **true**. If any such cell of B is **true**, then the answer is "yes." Otherwise, the answer is "no." The running time of this method is $O(n)$.

Feedback?

EXERCISE 1.6.70

(a) Given an array, A, describe an efficient algorithm for reversing A. For example, if $A = [3, 4, 1, 5]$, then its reversal is $A = [5, 1, 4, 3]$. You can only use $O(1)$ memory in addition to that used by A itself. What is the running time of your algorithm?

Feedback?

EXERCISE 1.6.71 All solutions visible to students

(a) Given a string, S, of n digits in the range from 0 to 9, describe an efficient algorithm for converting S into the integer it represents. What is the running time of your algorithm?

Solution Visible to students

Initialize your value $x = 0$. Go through S from beginning to end, and, for each digit d, update $x \leftarrow 10x + d$. The running time is $O(n)$.

Feedback?

EXERCISE 1.6.72

(a) Given an array, A, of n integers, find the longest subarray of A such that all the numbers in that subarray are in sorted order. What is the running time of your method?

Feedback?

EXERCISE 1.6.73 All solutions visible to students

(a) Given an array, A, of n positive integers, each of which appears in A exactly twice, except for one integer, x, describe an $O(n)$-time method for finding x using only a

single variable besides A.

Solution Visible to students

Initialize y to 0 and then XOR all the values in A with y. The result will be x.

Feedback?

EXERCISE 1.6.74 All solutions visible to students

(a) Given an array, A, of $n-2$ unique integers in the range from 1 to n, describe an $O(n)$-time method for finding the two integers in the range from 1 to n that are not in A. You may use only $O(1)$ space in addition to the space used by A.

Solution Visible to students

Compute the sum of all the integers in A and compute the sum of the squares of all the integers in A. Using the identities given in the appendix of this book, we know that, if all the numbers from 1 to n were present, then the first sum would be $n(n+1)/2$ and the second would be $n(n+1)(2n+1)/6$. So if we denote the missing numbers by i and j, and we denote the first sum by a and the second by b, then we know $a = n(n+1)/2 - i - j$ and $b = n(n+1)(2n+1)/6 - i^2 - j^2$. Solving these two equations will give us the values of i and j.

Feedback?

EXERCISE 1.6.75 All solutions visible to students

(a) Suppose you are writing a simulator for a single-elimination sports tournament (like in NCAA Division-1 basketball). There are n teams at the beginning of the tournament and in each round of the tournament teams are paired up and the games for each pair are simulated. Winners progress to the next round and losers are sent home. This continues until a grand champion team is the final winner. Suppose your simulator takes $O(\log\ n)$ time to process each game. How much time does your simulator take in total?

Solution Visible to students

Each time a game is played, one of the teams is sent home. If we start with n times, this means that there are $n-1$ games played in total. Thus, the total time for doing this simulation is $O(n \log n)$.

Feedback?

EXERCISE | 1.6.76

(a) Suppose you are given an array, A, of n positive integers. Describe an $O(n)$ algorithm for removing all the even numbers from A. That is, if A has m odd numbers, then, after you are done, these odd numbers should occupy the first m cells of A in the same relative order they were in originally.

Feedback?

EXERCISE | 1.6.77 | All solutions visible to students

(a) Given an integer $k > 0$ and an array, A, of n bits, describe an efficient algorithm for finding the shortest subarray of A that contains k 1's. What is the running time of your method?

Solution Visible to students

Scan through A using two pointers, i and j, such that $A[i:j]$ always has k 1's and i is as close to j as possible. Each time you increment j, you need to move it to the next 1 and then move i to get as close to j as possible to maintain $A[i:j]$ having k 1's. Since each operation increments either i or j, we can charge $2n$ cyber-dollars to pay for all operations. Thus, the total running time is $O(n)$.

Feedback?

EXERCISE | 1.6.78 | All solutions visible to students

(a) A certain town has exactly n married heterosexual couples. Every wife knows whether every other wife's husband is cheating on his wife or not, but no wife knows if her own husband is cheating or not. In fact, if a wife ever learns that her husband is cheating on her, then she will poison him that very night. So no husband will ever confess that he is cheating. One day, the mayor (who is not married) announces that there is at least one cheating husband in the town. What happens next?

Solution Visible to students

Let's apply induction to this problem. Note that if there is exactly 1 cheating husband, then his wife thinks that there are 0 cheating husbands in the town. So, on the day that the mayor makes his announcement, she learns that her husband must be cheating on her, and she poisons him that very night. By induction, if i nights have passed and no husbands have been poisoned, then every wife who thinks that there are exactly i