b. Answer the same question for decreasing arrays.

7. Solve the average-case recurrence for quicksort.

8. Design an algorithm to rearrange elements of a given array of n real numbers so that all its negative elements precede all its positive elements. Your algorithm should be both time- and space-efficient.

9. The ***Dutch flag problem*** is to rearrange any array of characters R, W, and B (red, white, and blue are the colors of the Dutch national flag) so that all the R's come first, the W's come next, and the B's come last. Design a linear in-place algorithm for this problem.

10. Implement quicksort in the language of your choice. Run your program on a sample of inputs to verify the theoretical assertions about the algorithm's efficiency.

11. *Nuts and bolts* You are given a collection of n bolts of different widths and n corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut. Design an algorithm for this problem with average-case efficiency in $\Theta(n \log n)$. [Raw91], p. 293

4.3 Binary Search

Binary search is a remarkably efficient algorithm for searching in a sorted array. It works by comparing a search key K with the array's middle element $A[m]$. If they match, the algorithm stops; otherwise, the same operation is repeated recursively for the first half of the array if $K < A[m]$, and for the second half if $K > A[m]$:

$$\begin{array}{c} K \\ \updownarrow \\ \underbrace{A[0] \dots A[m-1]}_{\substack{\text{search here if} \\ K<A[m]}} \; A[m] \; \underbrace{A[m+1] \dots A[n-1]}_{\substack{\text{search here if} \\ K>A[m]}} . \end{array}$$

As an example, let us apply binary search to searching for $K = 70$ in the array

3	14	27	31	39	42	55	70	74	81	85	93	98

Given the importance of quicksort, there have been persistent efforts over the years to refine the basic algorithm. Among several improvements discovered by researchers are: better pivot selection methods (such as the ***median-of-three partitioning*** that uses as a pivot the median of the leftmost, rightmost, and the middle element of the array); switching to a simpler sort on smaller subfiles; and recursion elimination (so-called nonrecursive quicksort). According to R. Sedgewick [Sed98], the world's leading expert on quicksort, these improvements in combination can cut the running time of the algorithm by 20%–25%.

We should also point out that the idea of partitioning can be useful in applications other than sorting. In particular, it underlines a fast algorithm for the important ***selection problem*** discussed in Section 5.6.

Exercises 4.2

1. Apply quicksort to sort the list

$$E,\ X,\ A,\ M,\ P,\ L,\ E$$

in alphabetical order. Draw the tree of the recursive calls made.

2. For the partitioning procedure outlined in Section 4.2:

a. Prove that if the scanning indices stop while pointing to the same element, i.e., $i = j$, the value they are pointing to must be equal to p.

b. Prove that when the scanning indices stop, j cannot point to an element more than one position to the left of the one pointed to by i.

c. Why is it worth stopping the scans after encountering an element equal to the pivot?

3. Is quicksort a stable sorting algorithm?

4. Give an example of an array of n elements for which the sentinel mentioned in the text is actually needed. What should be its value? Also explain why a single sentinel suffices for any input.

5. For the version of quicksort given in the text:

a. Are arrays made up of all equal elements the worst-case input, the best-case input, or neither?

b. Are strictly decreasing arrays the worst-case input, the best-case input, or neither?

6. a. For quicksort with the median-of-three pivot selection, are increasing arrays the worst-case input, the best-case input, or neither?

this was bubblesort and, by an amazing stroke of luck, my second thought was Quicksort." It is hard to disagree with his overall assessment: "I have been very lucky. What a wonderful way to start a career in Computing, by discovering a new sorting algorithm!" [Hoa96]

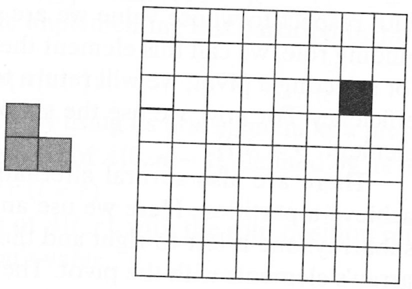
b. Set up a recurrence relation for the number of key comparisons made by mergesort on best-case inputs and solve it for $n = 2^k$.

c. Set up a recurrence relation for the number of key moves made by the version of mergesort given in Section 4.1. Does taking the number of key moves into account change the algorithm's efficiency class?

9. Let $A[0..n-1]$ be an array of n distinct real numbers. A pair $(A[i], A[j])$ is said to be an ***inversion*** if these numbers are out of order, i.e., $i < j$ but $A[i] > A[j]$. Design an $O(n \log n)$ algorithm for counting the number of inversions.

10. One can implement mergesort without a recursion by starting with merging adjacent elements of a given array, then merging sorted pairs, and so on. Implement this bottom-up version of mergesort in the language of your choice.

11. *Tromino puzzle* A tromino is an L-shaped tile formed by 1-by-1 adjacent squares. The problem is to cover any 2^n-by-2^n chessboard with one missing square (anywhere on the board) with trominos. Trominos should cover all the squares except the missing one with no overlaps.



Design a divide-and-conquer algorithm for this problem.

4.2 Quicksort

Quicksort is another important sorting algorithm that is based on the divide-and-conquer approach. Unlike mergesort, which divides its input's elements according to their position in the array, quicksort divides them according to their value. Specifically, it rearranges elements of a given array $A[0..n-1]$ to achieve its ***partition***, a situation where all the elements before some position s are smaller than or equal to $A[s]$ and all the elements after position s are greater than or equal to $A[s]$:

$$\underbrace{A[0] \dots A[s-1]}_{\text{all are } \le A[s]} \quad A[s] \quad \underbrace{A[s+1] \dots A[n-1]}_{\text{all are } \ge A[s]}$$

of extra storage the algorithm requires. Though merging can be done in place, the resulting algorithm is quite complicated and, since it has a significantly larger multiplicative constant, the in-place mergesort is of theoretical interest only.

Exercises 4.1

1. **a.** Write a pseudocode for a divide-and-conquer algorithm for finding a position of the largest element in an array of n numbers.
   b. What will be your algorithm's output for arrays with several elements of the largest value?
   c. Set up and solve a recurrence relation for the number of key comparisons made by your algorithm.
   d. How does this algorithm compare with the brute-force algorithm for this problem?
2. **a.** Write a pseudocode for a divide-and-conquer algorithm for finding values of both the largest and smallest elements in an array of n numbers.
   b. Set up and solve (for $n = 2^k$) a recurrence relation for the number of key comparisons made by your algorithm.
   c. How does this algorithm compare with the brute-force algorithm for this problem?
3. **a.** Write a pseudocode for a divide-and-conquer algorithm for the exponentiation problem of computing a^n where $a > 0$ and n is a positive integer.
   b. Set up and solve a recurrence relation for the number of multiplications made by this algorithm.
   c. How does this algorithm compare with the brute-force algorithm for this problem?
4. We mentioned in Chapter 2, that logarithm bases are irrelevant in most contexts arising in the analysis of an algorithm's efficiency class. Is this true for both assertions of the Master Theorem that include logarithms?
5. Find the order of growth for solutions of the following recurrences.
   a. $T(n) = 4T(n/2) + n, \ T(1) = 1$
   b. $T(n) = 4T(n/2) + n^2, \ T(1) = 1$
   c. $T(n) = 4T(n/2) + n^3, \ T(1) = 1$
6. Apply mergesort to sort the list E, X, A, M, P, L, E in alphabetical order.
7. Is mergesort a stable sorting algorithm?
8. **a.** Solve the recurrence relation for the number of key comparisons made by mergesort in the worst case. (You may assume that $n = 2^k$.)

c. Go to the Internet or your library and find a better algorithm for generating magic squares.

d. Implement the two algorithms—the exhaustive search and the one you have found—and run an experiment to determine the largest value of n for which each of the algorithms is able to find a magic square of order n in less than one minute of your computer's time.

10. *Famous alphametic* A puzzle in which the digits in a correct mathematical expression, such as a sum, are replaced by letters is called a ***cryptarithm***; if, in addition, the puzzle's words make sense, it is said to be an ***alphametic***. The most well-known alphametic was published by renowned British puzzlist H. E. Dudeney (1857–1930):

$$\begin{array}{r} \text{S E N D} \\ +\ \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

Two conditions are assumed: first, the correspndence between letters and decimal digits is one-to-one, i.e., each letter represents one digit only and different letters represent different digits; second, the digit zero does not appear as the left-most digit in any of the numbers. To solve an alphametic means to find which digit each letter represents. Note that a solution's uniqueness cannot be assumed and has to be verified by the solver.

a. Write a program for solving cryptarithms by exhaustive search. Assume that a given cryptarithm is a sum of two words.

b. Solve Dudeney's puzzle the way it was expected to be solved when it was first published in 1924.

SUMMARY

- *Brute force* is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved.
- The principal strengths of the brute-force approach are wide applicability and simplicity; its principal weakness is the subpar efficiency of most brute-force algorithms.
- A first application of the brute-force approach often results in an algorithm that can be improved with a modest amount of effort.
- The following noted algorithms can be considered as examples of the brute-force approach:

Exercises 3.4

1. **a.** Assuming that each tour can be generated in constant time, what will be the efficiency class of the exhaustive-search algorithm outlined in the text for the traveling salesman problem?

   b. If this algorithm is programmed on a computer that makes 1 billion additions per second, estimate the maximum number of cities for which the problem can be solved in

   i. one hour.
   ii. 24-hours.
   iii. one year.
   iv. one century.

2. Outline an exhaustive-search algorithm for the Hamiltonian circuit problem.

3. Outline an algorithm to determine whether a connected graph represented by its adjacency matrix has a Eulerian circuit. What is the efficiency class of your algorithm?

4. Complete the application of exhaustive search to the instance of the assignment problem started in the text.

5. Give an example of the assignment problem whose optimal solution does not include the smallest element of its cost matrix.

6. Consider the ***partition problem***: given n positive integers, partition them into two disjoint subsets with the same sum of their elements. (Of course, the problem does not always have a solution.) Design an exhaustive-search algorithm for this problem. Try to minimize the number of subsets the algorithm needs to generate.

7. Consider the ***clique problem***: given a graph G and a positive integer k, determine whether the graph contains a ***clique*** of size k, i.e., a complete subgraph of k vertices. Design an exhaustive-search algorithm for this problem.

8. Explain how exhaustive search can be applied to the sorting problem and determine the efficiency class of such an algorithm.

9. *Magic squares* A magic square of order n is an arrangement of the numbers from 1 to n^2 in an n-by-n matrix, with each number occurring exactly once, so that each row, each column, and each main diagonal has the same sum.

   a. Prove that if a magic square of order n exists, the sum in question must be equal to $n(n^2 + 1)/2$.

   b. Design an exhaustive-search algorithm for generating all magic squares of order n.

a signal, everybody hurles his or her pie at the nearest neighbor. Assuming that n is odd and that nobody can miss his or her target, true or false: There always remains at least one person not hit by a pie [Car79]?

5. The closest-pair problem can be posed on the k-dimensional space in which the Euclidean distance between two points $P' = (x'_1, \ldots, x'_k)$ and $P'' = (x''_1, \ldots, x''_k)$ is defined as

$$d(P', P'') = \sqrt{\sum_{s=1}^{k} (x'_s - x''_s)^2}.$$

What will be the efficiency class of the brute-force algorithm for the k-dimensional closest-pair problem?

6. Find the convex hulls of the following sets and identify their extreme points (if they have any).

a. a line segment

b. a square

c. the boundary of a square

d. a straight line

7. Design a linear-time algorithm to determine two extreme points of the convex hull of a set of $n > 1$ points in the plane.

8. What modification needs to be made in the brute-force algorithm for the convex-hull problem to handle more than two points on the same straight line?

9. Write a program implementing the brute-force algorithm for the convex-hull problem.

10. Consider the following small instance of the linear programming problem:

$$\begin{aligned} \text{maximize} \quad & 3x + 5y \\ \text{subject to} \quad & x + y \le 4 \\ & x + 3y \le 6 \\ & x \ge 0,\ y \ge 0. \end{aligned}$$

a. Sketch, in the Cartesian plane, the problem's ***feasible region***, defined as the set of points satisfying all the problem's constraints.

b. Identify the region's extreme points.

c. Solve the optimization problem given by using the following theorem: a linear programming problem with a nonempty bounded feasible region always has a solution, which can be found at one of the extreme points of its feasible region.

Second, such a line divides the plane into two half-planes: for all the points in one of them, $ax + by > c$, while for all the points in the other, $ax + by < c$. (For the points on the line itself, of course, $ax + by = c$.) Thus, to check whether certain points lie on the same side of the line, we can simply check whether the expression $ax + by - c$ has the same sign at each of these points. We leave the implementation details as an exercise.

What is the time efficiency of this algorithm? It is in $O(n^3)$: for each of $n(n-1)/2$ pairs of distinct points, we may need to find the sign of $ax + by - c$ for each of the other $n - 2$ points. There are much more efficient algorithms for this important problem, and we discuss one of them later in the book.

Exercises 3.3

1. Can you design a faster algorithm than the one based on the brute-force strategy to solve the closest-pair problem for n points $x_1, \ldots, x_n$ on the real line?

2. Let $x_1 < x_2 < \cdots < x_n$ be real numbers representing coordinates of n villages located along a straight road. A post office needs to be built in one of these villages.

a. Design an efficient algorithm to find the post office location minimizing the average distance between the villages and the post office.

b. Design an efficient algorithm to find the post office location minimizing the maximum distance from a village to the post office.

3. a. There are several alternative ways to define a distance between two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ in the Cartesian plane. In particular, the so-called ***Manhattan distance*** is defined as

$$d_M(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|.$$

Prove that d_M satisfies the following axioms which every distance function must satisfy:

i. $d_M(P_1, P_2) \geq 0$ for any two points P_1 and P_2, and $d_M(P_1, P_2) = 0$ if and only if $P_1 = P_2$;

ii. $d_M(P_1, P_2) = d_M(P_2, P_1)$;

iii. $d_M(P_1, P_2) \leq d_M(P_1, P_3) + d_M(P_3, P_2)$ for any P_1, P_2, and P_3.

b. Sketch all the points in the x, y coordinate plane whose Manhattan distance to the origin (0,0) is equal to 1. Do the same for the Euclidean distance.

c. True or false: A solution to the closest-pair problem does not depend on which of the two metrics—d_E (Euclidean) or d_M (Manhattan)—is used?

4. *Odd pie fight* There are $n \geq 3$ people positioned on a field (Euclidean plane) so that each has a unique nearest neighbor. Each person has a cream pie. At the same cell may be used no more than once. Write a computer program for solving this puzzle.

11. *Battleship game* Write a program based on a version of brute-force pattern matching for playing Battleship (a classic strategy game) on the computer. The rules of the game are as follows. There are two opponents in the game (in this case, a human player and the computer). The game is played on two identical boards (10-by-10 tables of squares) on which each opponent places his or her ships, not seen by the opponent. Each player has five ships, each of which occupies a certain number of squares on the board: a destroyer (2 squares), a submarine (3 squares), a cruiser (3 squares), a battleship (4 squares), and an aircraft carrier (5 squares). Each ship is placed either horizontally or vertically, with no two ships touching each other. The game is played by the opponents taking turns "shooting" at each other's ships. The result of every shot is displayed as either a hit or a miss. In case of a hit, the player gets to go again and keeps playing until missing. The goal is to sink all the opponent's ships before the opponent succeeds in doing it first. (To sink a ship, all squares occupied by the ship must be hit.)

3.3 Closest-Pair and Convex-Hull Problems by Brute Force

In this section, we consider a straightforward approach to two well-known problems dealing with a finite set of points in the plane. These problems, aside from their theoretical interest, arise in two important applied areas: computational geometry and operations research.

Closest-Pair Problem

The closest-pair problem calls for finding two closest points in a set of n points. For simplicity, we consider the two-dimensional case, although the problem can be posed for points in higher-dimensional spaces as well. We assume that the points in question are specified in a standard fashion by their (x, y) Cartesian coordinates and that the distance between two points $P_i = (x_i, y_i)$ and $P_j = (x_j, y_j)$ is the standard Euclidean distance

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

The brute-force approach to solving this problem leads to the following obvious algorithm: compute the distance between each pair of distinct points and find a pair with the smallest distance. Of course, we do not want to compute the distance between the same pair of points twice. To avoid doing so, we consider only the pairs of points (P_i, P_j) for which $i < j$.

2. As shown in Section 2.1, the average number of key comparisons made by sequential search (without a sentinel, under standard assumptions about its inputs) is given by the formula

$$C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p),$$

where p is the probability of a successful search. Determine, for a fixed n, the values of p $(0 \le p \le 1)$ for which this formula yields the largest value of $C_{avg}(n)$ and the smallest value of $C_{avg}(n)$.

3. *Gadget testing* A firm wants to determine the highest floor of its n-story headquarters from which a gadget can fall with no impact on the gadget's functionality. The firm has two identical gadgets to experiment with. Design an algorithm in the best efficiency class you can to solve this problem.

4. Determine the number of character comparisons that will be made by the brute-force algorithm in searching for the pattern GANDHI in the text

   THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED

   (Assume that the length of the text—it is 47 characters long—is known before the search starts.)

5. How many comparisons (both successful and unsuccessful) will be made by the brute-force algorithm in searching for each of the following patterns in the binary text of one thousand zeros?

   a. 00001 **b.** 10000 **c.** 01010

6. Give an example of a text of length n and a pattern of length m that constitutes a worst-case input for the brute-force string-matching algorithm. Exactly how many character comparisons will be made for such input?

7. Write a visualization progtam for the brute-force string-matching algorithm.

8. In solving the string-matching problem, would there be any advantage in comparing pattern and text characters right-to-left instead of left-to-right?

9. Consider the problem of counting, in a given text, the number of substrings that start with an A and end with a B. (For example, there are four such substrings in CABAAXBYA.)

   a. Design a brute-force algorithm for this problem and determine its efficiency class.

   b. Design a more efficient algorithm for this problem [Gin04].

10. *Word find* A popular diversion in the United States, "word find," asks the player to find each of a given set of words in a square table filled with single letters. A word can read horizontally (left or right), vertically (up or down), or along a 45 degree diagonal (in any of the four directions), formed by consecutively adjacent cells of the table; it may wrap around the table's boundaries, but it must read in the same direction with no zigzagging. The same cell of the table may be used in different words, but, in a given word,