This file contains the exercises, hints, and solutions for Chapter 1 of the book "Introduction to the Design and Analysis of Algorithms," 2nd edition, by A. Levitin. The problems that might be challenging for at least some students are marked by ▷; those that might be difficult for a majority of students are marked by ►.

Exercises 1.1

1. Do some research on al-Khorezmi (also al-Khwarizmi), the man from whose name the word "algorithm" is derived. In particular, you should learn what the origins of the words "algorithm" and "algebra" have in common.

2. Given that the official purpose of the U.S. patent system is the promotion of the "useful arts," do you think algorithms are patentable in this country? Should they be?

3. a. Write down driving directions for going from your school to your home with the precision required by an algorithm.

   b. Write down a recipe for cooking your favorite dish with the precision required by an algorithm.

4. Design an algorithm for computing $\lfloor \sqrt{n} \rfloor$ for any positive integer n. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.

5. a. Find gcd(31415, 14142) by applying Euclid's algorithm.

   b. Estimate how many times faster it will be to find gcd(31415, 14142) by Euclid's algorithm compared with the algorithm based on checking consecutive integers from $\min\{m, n\}$ down to $\gcd(m, n)$.

6. ▷ Prove the equality $\gcd(m, n) = \gcd(n, m \bmod n)$ for every pair of positive integers m and n.

7. What does Euclid's algorithm do for a pair of numbers in which the first number is smaller than the second one? What is the largest number of times this can happen during the algorithm's execution on such an input?

8. a. What is the smallest number of divisions made by Euclid's algorithm among all inputs $1 \leq m, n \leq 10$?

   b. What is the largest number of divisions made by Euclid's algorithm among all inputs $1 \leq m, n \leq 10$?

9. a. Euclid's algorithm, as presented in Euclid's treatise, uses subtractions rather than integer divisions. Write a pseudocode for this version of Euclid's algorithm.

b.▶ *Euclid's game* (see [Bog]) starts with two unequal positive numbers on the board. Two players move in turn. On each move, a player has to write on the board a positive number equal to the difference of two numbers already on the board; this number must be new, i.e., different from all the numbers already on the board. The player who cannot move loses the game. Should you choose to move first or second in this game?

10. The ***extended Euclid's algorithm*** determines not only the greatest common divisor d of two positive integers m and n but also integers (not necessarily positive) x and y, such that $mx + ny = d$.

    a. Look up a description of the extended Euclid's algorithm (see, e.g., [KnuI], p. 13) and implement it in the language of your choice.

    b. Modify your program for finding integer solutions to the Diophantine equation $ax + by = c$ with any set of integer coefficients a, b, and c.

11. *Locker doors* There are n lockers in a hallway numbered sequentially from 1 to n. Initially, all the locker doors are closed. You make n passes by the lockers, each time starting with locker #1. On the ith pass, $i = 1, 2, ..., n$, you toggle the door of every ith locker: if the door is closed, you open it, if it is open, you close it. For example, after the first pass every door is open; on the second pass you only toggle the even-numbered lockers (#2, #4, ...) so that after the second pass the even doors are closed and the odd ones are opened; the third time through you close the door of locker #3 (opened from the first pass), open the door of locker #6 (closed from the second pass), and so on. After the last pass, which locker doors are open and which are closed? How many of them are open?

Hints to Exercises 1.1

1. It is probably faster to do this by searching the Web, but your library should be able to help too.

2. One can find arguments supporting either view. There is a well established principle pertinent to the matter though: scientific facts or mathematical expressions of them are not patentable. (Why do you think it is the case?) But should this preclude granting patents for all algorithms?

3. You may assume that you are writing your algorithms for a human rather than a machine. Still, make sure that your descriptions do not contain obvious ambiguities. Knuth [KnuI], p.6 provides an interesting comparison between cooking recipes and algorithms.

4. There is a quite straightforward algorithm for this problem based on the definition of $\lfloor \sqrt{n} \rfloor$.

5. a. Just follow Euclid's algorithm as described in the text.

   b. Compare the number of divisions made by the two algorithms.

6. Prove that if d divides both m and n (i.e., $m = sd$ and $n = td$ for some positive integers s and t), then it also divides both n and $r = m \bmod n$ and vice versa. Use the formula $m = qn + r$ $(0 \le r < n)$ and the fact that if d divides two integers u and v, it also divides $u + v$ and $u - v$. (Why?)

7. Perform one iteration of the algorithm for two arbitrarily chosen integers $m < n$.

8. The answer to part (a) can be given immediately; the answer to part (b) can be given by checking the algorithm's performance on all pairs $1 < m < n \le 10$.

9. a. Use the equality

$$\gcd(m, n) = \gcd(m - n, n) \text{ for } m \ge n > 0.$$

   b. The key is to figure out the total number of distinct integers that can be written on the board, starting with an initial pair m, n where $m > n \ge 1$. You should exploit a connection of this question to the question of part (a). Considering small examples, especially those with $n = 1$ and $n = 2$, should help, too.

10. Of course, for some coefficients, the equation will have no solutions.

11. Tracing the algorithm by hand for, say, $n = 10$ and studying its outcome should help answering both questions.

Solutions to Exercises 1.1

1. Al-Khwarizmi (9th century C.E.) was a great Arabic scholar, most famous for his algebra textbook. In fact, the word "algebra" is derived from the Arabic title of this book while the word "algorithm" is derived from a translation of Al-Khwarizmi's last name (see, e.g., [KnuI], pp. 1-2, [Knu96], pp. 88-92, 114).

2. This legal issue has yet to be settled. The current legal state of affairs distinguishes mathematical algorithms, which are not patentable, from other algorithms, which may be patentable if implemented as computer programs (e.g., [Cha00]).

3. n/a

4. A straightforward algorithm that does not rely on the availability of an approximate value of $\sqrt{n}$ can check the squares of consecutive positive integers until the first square exceeding n is encountered. The answer will be the number's immediate predecessor. Note: A much faster algorithm for solving this problem can be obtained by using Newton's method (see Sections 11.4 and 12.4).

5. a. $\gcd(31415, 14142) = \gcd(14142, 3131) = \gcd(3131, 1618) = \gcd(1618, 1513) = \gcd(1513, 105) = \gcd(1513, 105) = \gcd(105, 43) = \gcd(43, 19) = \gcd(19, 5) = \gcd(5, 4) = \gcd(4, 1) = \gcd(1, 0) = 1$.

   b. To answer the question, we need to compare the number of divisions the algorithms make on the input given. The number of divisions made by Euclid's algorithm is 11 (see part a). The number of divisions made by the consecutive integer checking algorithm on each of its 14142 iterations is either 1 and 2; hence the total number of multiplications is between $1 \cdot 14142$ and $2 \cdot 14142$. Therefore, Euclid's algorithm will be between $1 \cdot 14142/11 \approx 1300$ and $2 \cdot 14142/11 \approx 2600$ times faster.

6. Let us first prove that if d divides two integers u and v, it also divides both $u + v$ and $u - v$. By definition of division, there exist integers s and t such that $u = sd$ and $v = td$. Therefore

   $$u \pm v = sd \pm td = (s \pm t)d,$$

   i.e., d divides both $u + v$ and $u - v$.

Also note that if d divides u, it also divides any integer multiple ku of u. Indeed, since d divides u, $u = sd$. Hence

$$ku = k(sd) = (ks)d,$$

i.e., d divides ku.

Now we can prove the assertion in question. For any pair of positive integers m and n, if d divides both m and n, it also divides both n and $r = m \bmod n = m - qn$. Similarly, if d divides both n and $r = m \bmod n = m - qn$, it also divides both $m = r + qn$ and n. Thus, the two pairs (m, n) and (n, r) have the same finite nonempty set of common divisors, including the largest element in the set, i.e., $\gcd(m, n) = \gcd(n, r)$.

7. For any input pair m, n such that $0 \leq m < n$, Euclid's algorithm simply swaps the numbers on the first iteration:

$$\gcd(m, n) = \gcd(n, m)$$

because $m \bmod n = m$ if $m < n$. Such a swap can happen only once since $\gcd(m, n) = \gcd(n, m \bmod n)$ implies that the first number of the new pair (n) will be greater than its second number $(m \bmod n)$ after every iteration of the algorithm.

8. a. For any input pair $m \geq n \geq 1$, in which m is a multiple of n, Euclid's algorithm makes exactly one division; it is the smallest number possible for two positive numbers.

   b. The answer is 5 divisions, which is made by Euclid's algorithm in computing $\gcd(5, 8)$. It is not too time consuming to get this answer by examining the number of divisions made by the algorithm on all input pairs $1 < m < n \leq 10$.
   Note: A pertinent general result (see [KnuII], p. 360) is that for any input pair m, n where $0 \leq n < N$, the number of divisions required by Euclid's algorithm to compute $\gcd(m, n)$ is at most $\lfloor \log_\phi (3 - \phi)N) \rfloor$ where $\phi = (1 + \sqrt{5})/2$.

9. a. Here is a nonrecursive version:
   Algorithm *Euclid2*(m, n)
   //Computes $\gcd(m, n)$ by Euclid's algorithm based on subtractions
   //Input: Two nonnegative integers m and n not both equal to 0
   //Output: The greatest common divisor of m and n
   while $n \neq 0$ **do**
   if $m < n$ *swap*(m, n)
   $m \leftarrow m - n$
   return m

b. It is not too difficult to prove that the integers that can be written on the board are the integers generated by the subtraction version of Euclid's algorithm and only them. Although the order in which they appear on the board may vary, their total number always stays the same: It is equal to $m/\gcd(m,n)$, where m is the maximum of the initial numbers, which includes two integers of the initial pair. Hence, the total number of possible moves is $m/\gcd(m,n)-2$. Consequently, if $m/\gcd(m,n)$ is odd, one should choose to go first; if it is even, one should choose to go second.

10. n/a

11. Since all the doors are initially closed, a door will be open after the last pass if and only if it is toggled an odd number of times. Door i ($1 \le i \le n$) is toggled on pass j ($1 \le j \le n$) if and only if j divides i. Hence, the total number of times door i is toggled is equal to the number of its divisors. Note that if j divides i, i.e. $i = jk$, then k divides i too. Hence all the divisors of i can be paired (e.g., for $i = 12$, such pairs are 1 and 12, 2 and 6, 3 and 4) unless i is a perfect square (e.g., for $i = 16$, 4 does not have another divisor to be matched with). This implies that i has an odd number of divisors if and only if it is a perfect square, i.e., $i = j^2$. Hence doors that are in the positions that are perfect squares and only such doors will be open after the last pass. The total number of such positions not exceeding n is equal to $\lfloor\sqrt{n}\rfloor$: these numbers are the squares of the positive integers between 1 and $\lfloor\sqrt{n}\rfloor$ inclusively.

Exercises 1.2

1. *Old World puzzle* A peasant finds himself on a riverbank with a wolf, a goat, and a head of cabbage. He needs to transport all three to the other side of the river in his boat. However, the boat has room for only the peasant himself and one other item (either the wolf, the goat, or the cabbage). In his absence, the wolf would eat the goat, and the goat would eat the cabbage. Solve this problem for the peasant or prove it has no solution. (Note: The peasant is a vegetarian but does not like cabbage and hence can eat neither the goat nor the cabbage to help him solve the problem. And it goes without saying that the wolf is a protected species.)

2. *New World puzzle* There are four people who want to cross a bridge; they all begin on the same side. You have 17 minutes to get them all across to the other side. It is night, and they have one flashlight. A maximum of two people can cross the bridge at one time. Any party that crosses, either one or two people, must have the flashlight with them. The flashlight must be walked back and forth; it cannot be thrown, for example. Person 1 takes 1 minute to cross the bridge, person 2 takes 2 minutes, person 3 takes 5 minutes, and person 4 takes 10 minutes. A pair must walk together at the rate of the slower person's pace. For example, if person 1 and person 4 walk across first, 10 minutes have elapsed when they get to the other side of the bridge. If person 4 returns the flashlight, a total of 20 minutes have passed and you have failed the mission. (Note: According to a rumor on the Internet, interviewers at a well-known software company located near Seattle have given this problem to interviewees.)

3. Which of the following formulas can be considered an algorithm for computing the area of a triangle whose side lengths are given positive numbers a, b, and c?

   a. $S = \sqrt{p(p-a)(p-b)(p-c)}$, where $p = (a+b+c)/2$

   b. $S = \frac{1}{2}bc \sin A$, where A is the angle between sides b and c

   c. $S = \frac{1}{2}ah_a$, where h_a is the height to base a

4. Write a pseudocode for an algorithm for finding real roots of equation $ax^2 + bx + c = 0$ for arbitrary real coefficients a, b, and c. (You may assume the availability of the square root function $sqrt(x)$.)

5. Describe the standard algorithm for finding the binary representation of a positive decimal integer

   a. in English.

   b. in a pseudocode.

6. Describe the algorithm used by your favorite ATM machine in dispensing cash. (You may give your description in either English or a pseudocode, whichever you find more convenient.)

7. a. Can the problem of computing the number π be solved exactly?

   b. How many instances does this problem have?

   c. Look up an algorithm for this problem on the World Wide Web.

8. Give an example of a problem other than computing the greatest common divisor for which you know more than one algorithm. Which of them is simpler? Which is more efficient?

9. Consider the following algorithm for finding the distance between the two closest elements in an array of numbers.

   Algorithm *MinDistance*$(A[0..n-1])$
   //Input: Array $A[0..n-1]$ of numbers
   //Output: Minimum distance between two of its elements
   $dmin \leftarrow \infty$
   for $i \leftarrow 0$ **to** $n-1$ **do**
      for $j \leftarrow 0$ **to** $n-1$ **do**
         if $i \neq j$ **and** $|A[i] - A[j]| < dmin$
            $dmin \leftarrow |A[i] - A[j]|$
   return $dmin$

   Make as many improvements as you can in this algorithmic solution to the problem. (If you need to, you may change the algorithm altogether; if not, improve the implementation given.)

10. One of the most influential books on problem solving, titled *How To Solve It* [Pol57], was written by the Hungarian-American mathematician George Polya (1887–1985). Polya summarized his ideas in a four-point summary. Find this summary on the Web or, better yet, in his book, and compare it with the plan outlined in Section 1.2. What do they have in common? How are they different?

Hints to Exercises 1.2

1. The peasant would have to make several trips across the river, starting with the only one possible.
2. Unlike the Old World puzzle of Problem 1, the first move solving this puzzle is not obvious.
3. The principal issue here is a possible ambiguity.
4. Your algorithm should work correctly for all possible values of the coefficients, including zeros.
5. You almost certainly learned this algorithm in one of your introductory programming courses. If this assumption is not true, you have a choice between designing such an algorithm on your own or looking it up.
6. You may need to make a field trip to refresh your memory.
7. Question (a) is difficult though the answer to it—discovered in 1760s by the German mathematician Johann Lambert —is well-known. By comparison, question (b) is incomparably simpler.
8. You probably know two or more different algorithms for sorting an array of numbers.
9. You can: decrease the number of times the innermost loop is executed, make that loop run faster (at least for some inputs), or, more significantly, design a faster algorithm from scratch.
10. n/a

Solutions to Exercises 1.2

1. Let `P`, `w`, `g`, and `c` stand for the peasant, wolf, goat, and cabbage head, respectively. The following is one of the two principal sequences that solve the problem:

	`P g`	`g`	`Pwg`	`w`	`Pw c`	`w c`	`Pwgc`
`Pwgc`	`w c`	`Pw c`	`c`	`P gc`	`g`	`P g`	

   Note: This problem is revisited later in the book (see Section 6.6).

2. Let 1, 2, 5, 10 be labels representing the men of the problem, f represent the flashlight's location, and the number in the parenthesis be the total amount of time elapsed. The following sequence of moves solves the problem:

	f,1,2	2	f,2,5,10	5,10	f,1,2,5,10
(0)	(2)	(3)	(13)	(15)	(17)
f,1,2,5,10	5,10	f,1,5,10	1	f,1,2	

3. a. The formula can be considered an algorithm if we assume that we know how to compute the square root of an arbitrary positive number.

   b. The difficulty here lies in computing $\sin A$. Since the formula says nothing about how it has to be computed, it should not be considered an algorithm. This is true even if we assume, as we did for the square root function, that we know how to compute the sine of a given angle. (There are several algorithms for doing this but only approximately, of course.) The problem is that the formula says nothing about how to compute angle A either.

   c. The formula says nothing about how to compute h_a.

4. **Algorithm** *Quadratic*(a, b, c)
   //The algorithm finds real roots of equation $ax^2 + bx + c = 0$
   //Input: Real coefficients a, b, c
   //Output: The real roots of the equation or a message about their absence
   if $a \neq 0$
      $D \leftarrow b * b - 4 * a * c$
      if $D > 0$
         $temp \leftarrow 2 * a$
         $x1 \leftarrow (-b + sqrt(D))/temp$
         $x2 \leftarrow (-b - sqrt(D))/temp$

return $x1, x2$
else if $D = 0$ **return** $-b/(2 * a)$
else return '*no real roots*'
else $//a = 0$
if $b \neq 0$ **return** $-c/b$
else $//a = b = 0$
if $c = 0$ **return** '*all real numbers*'
else return '*no real roots*'

Note: See a more realistic algorithm for this problem in Section 11.4.

5. a. Divide the given number n by 2: the remainder r_n (0 or 1) will be the next (from right to left) digit of the binary representation in question. Replace n by the quotient of the last division and repeat this operation until n becomes 0.

   b. **Algorithm** *Binary*(n)
   //The algorithm implements the standard method for finding
   //the binary expansion of a positive decimal integer
   //Input: A positive decimal integer n
   //Output: The list $b_k\ b_{k-1}...b_1\ b_0$ of n's binary digits
   $k \leftarrow 0$
   while $n \neq 0$
   $b_k \leftarrow n \bmod 2$
   $n \leftarrow \lfloor n/2 \rfloor$
   $k \leftarrow k + 1$

6. n/a

7. a. π, as an irrational number, can be computed only approximately.

   b. It is natural to consider, as an instance of this problem, computing π's value with a given level of accuracy, say, with n correct decimal digits. With this interpretation, the problem has infinitely many instances.

8. n/a

9. The following improved version considers the same pair of elements only once and avoids recomputing the same expression in the innermost loop:

Algorithm *MinDistance2*$(A[0..n-1])$
//Input: An array $A[0..n-1]$ of numbers
//Output: The minimum distance d between two of its elements

```
dmin ← ∞
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        temp ← |A[i] − A[j]|
        if temp < dmin
            dmin ← temp
return dmin
```

A faster algorithm is based on the idea of presorting (see Section 6.1).

10. Polya's general four-point approach is:
    1. Understand the problem
    2. Devise a plan
    3. Implement the plan
    4. Look back/check

Exercises 1.3

1. Consider the algorithm for the sorting problem that sorts an array by counting, for each of its elements, the number of smaller elements and then uses this information to put the element in its appropriate position in the sorted array:

Algorithm *ComparisonCountingSort*($A[0..n-1]$, $S[0..n-1]$)
//Sorts an array by comparison counting
//Input: Array $A[0..n-1]$ of orderable values
//Output: Array $S[0..n-1]$ of A's elements sorted in nondecreasing order
for $i \leftarrow 0$ **to** $n-1$ **do**
    $Count[i] \leftarrow 0$
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] < A[j]$
            $Count[j] \leftarrow Count[j] + 1$
        else $Count[i] \leftarrow Count[i] + 1$
for $i \leftarrow 0$ **to** $n-1$ **do**
    $S[Count[i]] \leftarrow A[i]$

   a. Apply this algorithm to sorting the list 60, 35, 81, 98, 14, 47.

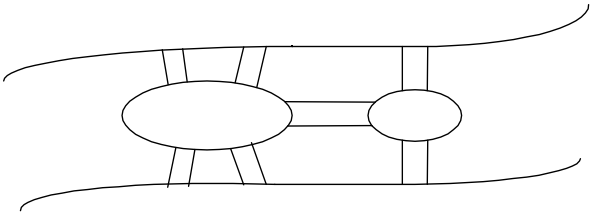
   b. Is this algorithm stable?

   c. Is it in place?

2. Name the algorithms for the searching problem that you already know. Give a good succinct description of each algorithm in English. (If you know no such algorithms, use this opportunity to design one.)

3. Design a simple algorithm for the string-matching problem.

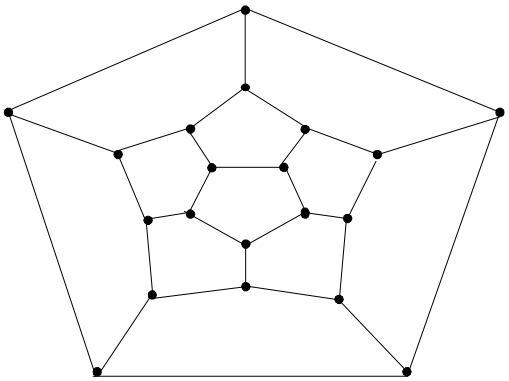
4. *Königsberg bridges* The Königsberg bridge puzzle is universally accepted as the problem that gave birth to graph theory. It was solved by the great Swiss-born mathematician Leonhard Euler (1707–1783). The problem asked whether one could, in a single stroll, cross all seven bridges of the city of Königsberg exactly once and return to a starting point. Following is a sketch of the river with its two islands and seven bridges:



   a. State the problem as a graph problem.

b. Does this problem have a solution? If you believe it does, draw such a stroll; if you believe it does not, explain why and indicate the smallest number of new bridges that would be required to make such a stroll possible.

5. *Icosian Game* A century after Euler's discovery (see Problem 4), another famous puzzle—this one invented by the renown Irish mathematician Sir William Hamilton (1805-1865)—was presented to the world under the name of the Icosian Game. The game was played on a circular wooden board on which the following graph was carved:



Find a ***Hamiltonian circuit***—a path that visits all the graph's vertices exactly once before returning to the starting vertex—for this graph.

6. Consider the following problem: Design an algorithm to determine the best route for a subway passenger to take from one designated station to another in a well-developed subway system similar to those in such cities as Washington, D.C., and London, UK.

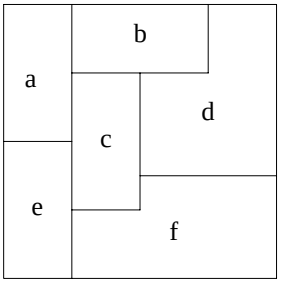
   a. The problem's statement is somewhat vague, which is typical of real-life problems. In particular, what reasonable criterion can be used for defining the "best" route?

   b. How would you model this problem by a graph?

7. a. Rephrase the traveling salesman problem in combinatorial object terms.

   b. Rephrase the graph-coloring problem in combinatorial object terms.

8. Consider the following map:



a. Explain how we can use the graph-coloring problem to color the map so that no two neighboring regions are colored the same.

b. Use your answer to part (a) to color the map with the smallest number of colors.

9. Design an algorithm for the following problem: Given a set of n points in the Cartesian plane, determine whether all of them lie on the same circumference.

10. Write a program that reads as its inputs the (x, y) coordinates of the endpoints of two line segments P_1Q_1 and P_2Q_2 and determines whether the segments have a common point.

Hints to Exercises 1.3

1. Trace the algorithm on the input given. Use the definitions of stability and being in place that were introduced in the section.

2. If you do not recall any searching algorithms, you should design a simple searching algorithm (without succumbing to the temptation to find one in the latter chapters of the book).

3. This algorithm is introduced later in the book but you should have no trouble to design it on your own.

4. If you have not encountered this problem in your previous courses, you may look up the answers on the Web or in a discrete structures textbook. The answers are, in fact, surprisingly simple.

5. No efficient algorithm for solving this problem for an arbitrary graph is known. This particular graph does have Hamiltonian circuits which are not difficult to find. (You need to find just one of them.)

6. a. Put yourself (mentally) in a passenger's place and ask yourself what criterion for the "best" route you would use. Then think of people that may have different needs.

   b. The representation of the problem by a graph is straightforward. Give some thoughts though to stations where trains can be changed.

7. a. What are tours in the traveling salesman problem?

   b. It would be natural to consider vertices colored the same color as elements of the same subset.

8. Create a graph whose vertices represent the map's regions. You will have to decide on the edges on your own.

9. Assume that the circumference in question exists and find its center first. Also, do not forget to give a special answer for $n \leq 2$.

10. Be careful not to miss some special cases of the problem.

Solutions to Exercises 1.3

1. a. Sorting 60, 35, 81, 98, 14, 47 by comparison counting will work as follows:

Array $A[0..5]$		60	35	81	98	14	47
Initially	$Count[]$	0	0	0	0	0	0
After pass $i = 0$	$Count[]$	3	0	1	1	0	0
After pass $i = 1$	$Count[]$		1	2	2	0	1
After pass $i = 2$	$Count[]$			4	3	0	1
After pass $i = 3$	$Count[]$				5	0	1
After pass $i = 4$	$Count[]$					0	2
Final state	$Count[]$	3	1	4	5	0	2
Array $S[0..5]$		14	35	47	60	81	98

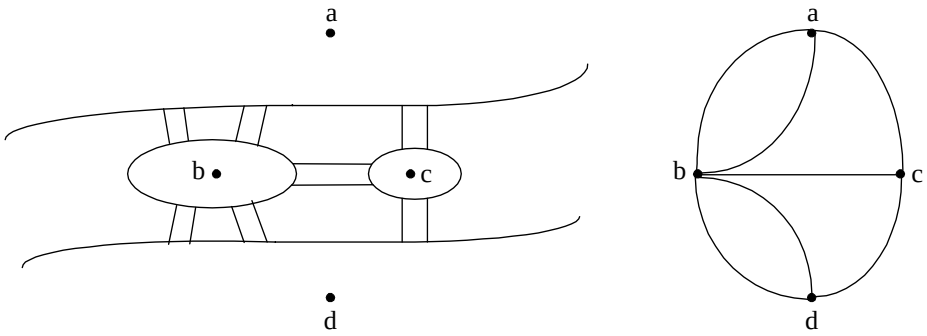
   b. The algorithm is not stable. Consider, as a counterexample, the result of its application to $1'$, $1''$.

   c. The algorithm is not in place because it uses two extra arrays of size n: $Count$ and S.

2. Answers may vary but most students should be familiar with sequential search, binary search, binary tree search and, possibly, hashing from their introductory programming courses.

3. Align the pattern with the beginning of the text. Compare the corresponding characters of the pattern and the text left-to right until either all the pattern characters are matched (then stop—the search is successful) or the algorithm runs out of the text's characters (then stop—the search is unsuccessful) or a mismatching pair of characters is encountered. In the latter case, shift the pattern one position to the right and resume the comparisons.

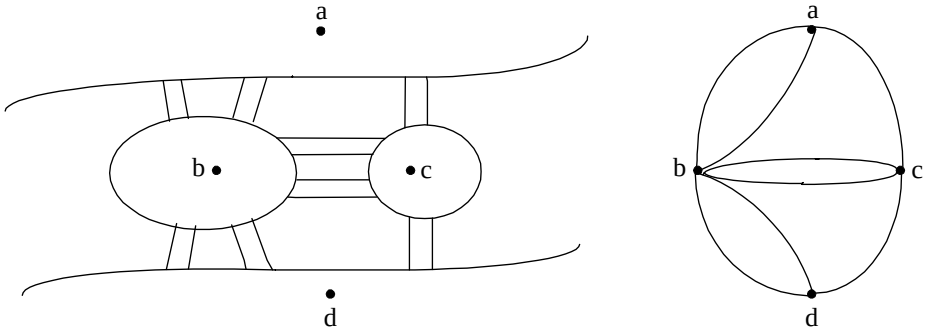
4. a. If we represent each of the river's banks and each of the two islands by vertices and the bridges by edges, we will get the following graph:



(This is, in fact, a multigraph, not a graph, because it has more than one edge between the same pair of vertices. But this doesn't matter for the issue at hand.) The question is whether there exists a path (i.e., a sequence of adjacent vertices) in this multigraph that traverses all the edges exactly once and returns to a starting vertex. Such paths are called *Eulerian circuits*; if a path traverses all the edges exactly once but does not return to its starting vertex, it is called an *Eulerian path*.

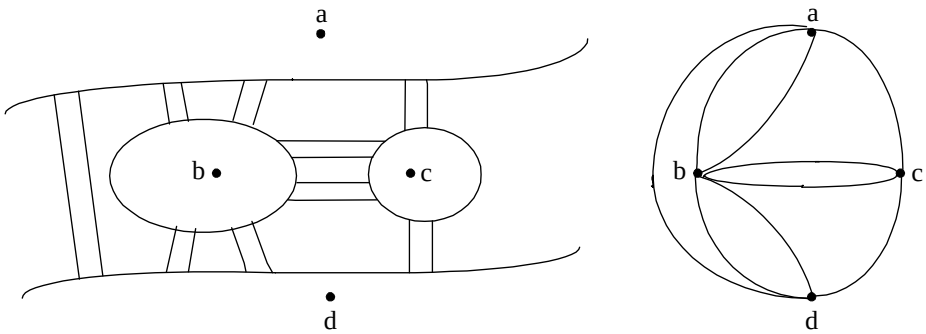
b. Euler proved that an Eulerian circuit exists in a connected (multi)graph if and only if all its vertices have even degrees, where the degree of a vertex is defined as the number of edges for which it is an endpoint. Also, an Eulerian path exists in a connected (multi)graph if and only if it has exactly two vertices of odd degrees; such a path must start at one of those two vertices and end at the other. Hence, for the multigraph of the puzzle, there exists neither an Eulerian circuit nor an Eulerian path because all its four vertices have odd degrees.

If we are to be satisfied with an Eulerian path, two of the multigraph's vertices must be made even. This can be accomplished by adding one new bridge connecting the same places as the existing bridges. For example, a new bridge between the two islands would make possible, among others, the walk $a - b - c - a - b - d - c - b - d$



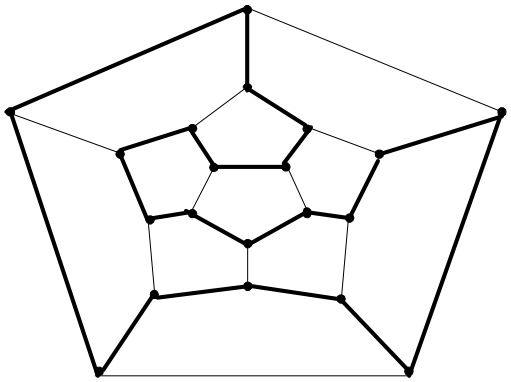
If we want a walk that returns to its starting point, all the vertices in the

corresponding multigraph must be even. Since a new bridge/edge changes the parity of two vertices, at least two new bridges/edges will be needed. For example, here is one such "enhancement":



This would make possible $a-b-c-a-b-d-c-b-d-a$, among several other such walks.

5. A Hamiltonian circuit is marked on the graph below:



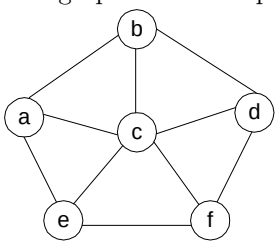
6. a. At least three "reasonable"criteria come to mind: the fastest trip, a trip with the smallest number of train stops, and a trip that requires the smallest number of train changes. Note that the first criterion requires the information about the expected traveling time between stations and the time needed for train changes whereas the other two criteria do not require such information.

   b. A natural approach is to mimic subway plans by representing stations by vertices of a graph, with two vertices connected by an edge if there is a train line between the corresponding stations. If the time spent on changing a train is to be taken into account (e.g., because the station in question is on more than one line), the station should be represented by more then one vertex.

7. a. Find a permutation of n given cities for which the sum of the distances between consecutive cities in the permutation plus the distance between its last and first city is as small as possible.

   b. Partition all the graph's vertices into the smallest number of disjoint subsets so that there is no edge connecting vertices from the same subset.

8. a. Create a graph whose vertices represent the map's regions and the edges connect two vertices if and only if the corresponding regions have a common border (and therefore cannot be colored the same color). Here is the graph for the map given:



   Solving the graph coloring problem for this graph yields the map's coloring with the smallest number of colors possible.

   b. Without loss of generality, we can assign colors 1 and 2 to vertices c and a, respectively. This forces the following color assignment to the remaining vertices: 3 to b, 2 to d, 3 to f, 4 to e. Thus, the smallest number of colors needed for this map is four.

   Note: It's a well-known fact that *any* map can be colored in four colors or less. This problem—known as the Four-Color Problem—has remained unresolved for more than a century until 1976 when it was finally solved by the American mathematicians K. Appel and W. Haken by a combination of mathematical arguments and extensive computer use.

9. If $n = 2$, the answer is always "yes"; so, we may assume that $n \geq 3$. Select three points P_1, P_2, and P_3 from the set given. Write an equation of the perpendicular bisector l_1 of the line segment with the endpoints at P_1 and P_2, which is the locus of points equidistant from P_1 and P_2. Write an equation of the perpendicular bisector l_2 of the line segment with the endpoints at P_2 and P_3, which is the locus of points equidistant from P_2 and P_3. Find the coordinates (x, y) of the intersection point P of the lines l_1 and l_2 by solving the system of two equations in two unknowns x and y. (If the system has no solutions, return "no": such a circumference does not exist.) Compute the distances (or much better yet the distance squares!) from P to each of the points P_i, $i = 3, 4, ..., n$ and check whether all of them are the same: if they are, return "yes," otherwise, return "no".

Exercises 1.4

1. Describe how one can implement each of the following operations on an array so that the time it takes does not depend on the array's size n.

   a. Delete the ith element of an array ($1 \leq i \leq n$).

   b. Delete the ith element of a sorted array (the remaining array has to stay sorted, of course).

2. If you have to solve the searching problem for a list of n numbers, how can you take advantage of the fact that the list is known to be sorted? Give separate answers for

   a. lists represented as arrays.

   b. lists represented as linked lists.

3. a. Show the stack after each operation of the following sequence that starts with the empty stack:

   push(a), push(b), pop, push(c), push(d), pop

   b. Show the queue after each operation of the following sequence that starts with the empty queue:

   enqueue(a), enqueue(b), dequeue, enqueue(c), enqueue(d), dequeue

4. a. Let A be the adjacency matrix of an undirected graph. Explain what property of the matrix indicates that
   i. the graph is complete.
   ii. the graph has a loop, i.e., an edge connecting a vertex to itself.
   iii. the graph has an isolated vertex, i.e., a vertex with no edges incident to it.

   b. Answer the same questions for the adjacency list representation.

5. Give a detailed description of an algorithm for transforming a free tree into a tree rooted at a given vertex of the free tree.

6. Prove the inequalities that bracket the height of a binary tree with n vertices:

   $$\lfloor \log_2 n \rfloor \leq h \leq n - 1.$$

7. Indicate how the ADT priority queue can be implemented as

   a. an (unsorted) array.

b. a sorted array.

c. a binary search tree.

8. How would you implement a dictionary of a reasonably small size n if you knew that all its elements are distinct (e.g., names of 50 states of the United States)? Specify an implementation of each dictionary operation.

9. For each of the following applications, indicate the most appropriate data structure:

   a. answering telephone calls in the order of their known priorities.

   b. sending backlog orders to customers in the order they have been received.

   c. implementing a calculator for computing simple arithmetical expressions.

10. *Anagram checking* Design an algorithm for checking whether two given words are anagrams, i.e., whether one word can be obtained by permuting the letters of the other. (For example, the words *tea* and *eat* are anagrams.)

Hints to Exercises 1.4

1. a. Take advantage of the fact that the array is not sorted.

   b. We used this trick in implementing one of the algorithms in Section 1.1.

2. a. For a sorted array, there is a spectacularly efficient algorithm you almost certainly have heard about.

   b. Unsuccessful searches can be made faster.

3. a. *Push(x)* puts x on the top of the stack, *pop* deletes the item from the top of the stack.

   b. *Enqueue(x)* adds x to the rear of the queue, *dequeue* deletes the item from the front of the queue.

4. Just use the definitions of the graph properties in question and data structures involved.

5. There are two well-known algorithms that can solve this problem. The first uses a stack, the second uses a queue. Although these algorithms are discussed later in the book, do not miss this chance to discover them by yourself!

6. The inequality $h \leq n-1$ follows immediately from the height's definition. The lower bound inequality follows from inequality $2^{h+1} - 1 \geq n$, which can be proved by considering the largest number of vertices a binary tree of height h can have.

7. You need to indicate how each of the three operations of the priority queue will be implemented.

8. Because of insertions and deletions, using an array of the dictionary's elements (sorted or unsorted) is not the best implementation possible.

9. You need to know about the postfix notation in order to answer one of these questions. (If you are not familiar with it, find the information on the Internet.)

10. There are several algorithms for this problem. Keep in mind that the words may contain multiple occurrences of the same letter.

Solutions to Exercises 1.4

1. a. Replace the ith element with the last element and decrease the array size by 1.

   b. Replace the ith element with a special symbol that cannot be a value of the array's element (e.g., 0 for an array of positive numbers) to mark the ith position as empty. (This method is sometimes called the "lazy deletion".)

2. a. Use binary search (see Section 4.3 if you are not familiar with this algorithm).

   b. When searching in a sorted linked list, stop as soon as an element greater than or equal to the search key is encountered.

3. a.

$push(a)$		$push(b)$	b	pop		$push(c)$	c	$push(d)$	d	pop	c
	a		a		a		a		c		a
									a		

   b.

$enqueue(a)$	$enqueue(b)$	$dequeue$	$enqueue(c)$	$enqueue(d)$	$dequeue$
a	ab	b	bc	bcd	cd

4. a. For the adjacency matrix representation:

   i. A graph is complete if and only if all the elements of its adjacency matrix except those on the main diagonal are equal to 1, i.e., $A[i, j] = 1$ for every $1 \le i, j \le n$, $i \ne j$.

   ii. A graph has a loop if and only if its adjacency matrix has an element equal to 1 on its main diagonal, i.e., $A[i, i] = 1$ for some $1 \le i \le n$.

   iii. An (undirected, without loops) graph has an isolated vertex if and only if its adjacency matrix has an all-zero row.

   b. For the adjacency list representation:

   i. A graph is complete if and only if each of its linked lists contains all the other vertices of the graph.

   ii. A graph has a loop if and only if one of its adjacency lists contains the

vertex defining the list.

iii. An (undirected, without loops) graph has an isolated vertex if and only if one of its adjacency lists is empty.

5. The first algorithm works as follows. Mark a vertex to serve as the root of the tree, make it the root of the tree to be constructed, and initialize a stack with this vertex. Repeat the following operation until the stack becomes empty: If there is an unmarked vertex adjacent to the vertex on the top to the stack, mark the former vertex, attach it as a child of the top's vertex in the tree, and push it onto the stack; otherwise, pop the vertex off the top of the stack.

   The second algorithm works as follows. Mark a vertex to serve as the root of the tree, make it the root of the tree to be constructed, and initialize a queue with this vertex. Repeat the following operations until the queue becomes empty: If there are unmarked vertices adjacent to the vertex at the front of the queue, mark all of them, attach them as children to the front vertex in the tree, and add them to the queue; then dequeue the queue.

6. Since the height is defined as the length of the longest simple path from the tree's root to its leaf, such a pass will include no more than n vertices, which is the total number of vertices in the tree. Hence, $h \le n - 1$.

   The binary tree of height h with the largest number of vertices is the full tree that has all its $h + 1$ levels filled with the largest number of vertices possible. The total number of vertices in such a tree is $\sum_{l=0}^{h} 2^l = 2^{h+1} - 1$. Hence, for any binary tree with n vertices and height h

   $$2^{h+1} - 1 \ge n.$$

   This implies that

   $$2^{h+1} \ge n + 1$$

   or, after taking binary logarithms of both hand sides and taking into account that $h + 1$ is an integer,

   $$h + 1 \ge \lceil \log_2(n + 1) \rceil.$$

   Since $\lceil \log_2(n + 1) \rceil = \lfloor \log_2 n \rfloor + 1$ (see Appendix A), we finally obtain

   $$h + 1 \ge \lfloor \log_2 n \rfloor + 1 \text{ or } h \ge \lfloor \log_2 n \rfloor.$$

7. a. Insertion can be implemented by adding the new item after the array's last element. Finding the largest element requires a standard scan

through the array to find its largest element. Deleting the largest element $A[i]$ can be implemented by exchanging it with the last element and decreasing the array's size by 1.

b. We will assume that the array $A[0..n-1]$ representing the priority queue is sorted in ascending order. Inserting a new item of value v can be done by scanning the sorted array, say, left to right until an element $A[j] \geq v$ or the end of the array is reached. (A faster algorithm for finding a place for inserting a new element is binary search discussed in Section 4.3.) In the former case, the new item is inserted before $A[j]$ by first moving $A[n-1], ..., A[j]$ one position to the right; in the latter case, the new item is simply appended after the last element of the array. Finding the largest element is done by simply returning the value of the last element of the sorted array. Deletion of the largest element is done by decreasing the array's size by one.

c. Insertion of a new element is done by using the standard algorithm for inserting a new element in a binary search tree: recursively, the new key is inserted in the left or right subtree depending on whether it is smaller or larger than the root's key. Finding the largest element will require finding the rightmost element in the binary tree by starting at the root and following the chain of the right children until a vertex with no right subtree is reached. The key of that vertex will be the largest element in question. Deleting it can be done by making the right pointer of its parent to point to the left child of the vertex being deleted;. if the rightmost vertex has no left child, this pointer is made "null". Finally, if the rightmost vertex has no parent, i.e., if it happens to be the root of the tree, its left child becomes the new root; if there is no left child, the tree becomes empty.

8. Use a bit vector, i.e., an array on n bits in which the ith bit is 1 if the ith element of the underlying set is currently in the dictionary and 0 otherwise. The search, insertion, and deletion operations will require checking or changing a single bit of this array.

9. Use: (a) a priority queue; (b) a queue; (c) a stack (and *reverse Polish notation*—a clever way of representing arithmetical expressions without parentheses, which is usually studied in a data structures course).

10. The most straightforward solution is to search for each successive letter of the first word in the second one. If the search is successful, delete the first occurrence of the letter in the second word, stop otherwise.

    Another solution is to sort the letters of each word and then compare

them in a simple parallel scan.

We can also generate and compare "letter vectors" of the given words: $V_w[i]$ = the number of occurrences of the alphabet's ith letter in the word w. Such a vector can be generated by initializing all its components to 0 and then scanning the word and incrementing appropriate letter counts in the vector.