This file contains the exercises, hints, and solutions for Chapter 3 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 3.1

1. a. Give an example of an algorithm that should not be considered an application of the brute-force approach.

   b. Give an example of a problem that cannot be solved by a brute-force algorithm.

2. a. What is the efficiency of the brute-force algorithm for computing a^n as a function of n? As a function of the number of bits in the binary representation of n?

   b. If you are to compute $a^n \bmod m$ where $a > 1$ and n is a large positive integer, how would you circumvent the problem of a very large magnitude of a^n?

3. For each of the algorithms in Problems 4, 5, and 6 of Exercises 2.3, tell whether or not the algorithm is based on the brute-force approach.

4. a. Design a brute-force algorithm for computing the value of a polynomial

   $$p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

   at a given point x_0 and determine its worst-case efficiency class.

   b. If the algorithm you designed is in $\Theta(n^2)$, design a linear algorithm for this problem.

   c. Is it possible to design an algorithm with a better than linear efficiency for this problem?

5. Sort the list E, X, A, M, P, L, E in alphabetical order by selection sort.

6. Is selection sort stable? (The definition of a stable sorting algorithm was given in Section 1.3.)

7. Is it possible to implement selection sort for linked lists with the same $\Theta(n^2)$ efficiency as the array version?

8. Sort the list E, X, A, M, P, L, E in alphabetical order by bubble sort.

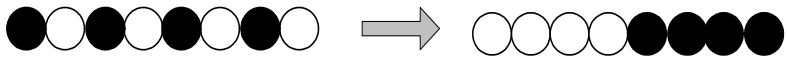
9. a. Prove that if bubble sort makes no exchanges on its pass through a list, the list is sorted and the algorithm can be stopped.

b. Write a pseudocode of the method that incorporates this improvement.

c. Prove that the worst-case efficiency of the improved version is quadratic.

10. Is bubble sort stable?

11. *Alternating disks* You have a row of $2n$ disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those which interchange the positions of two neighboring disks.



Design an algorithm for solving this puzzle and determine the number of moves it makes. [Gar99], p.75

Hints to Exercises 3.1

1. a. Think of algorithms that have impressed you with their efficiency and/or sophistication. Neither characteristic is indicative of a brute-force algorithm.

   b. Surprisingly, it is not a very easy question to answer. Mathematical problems (including those you have studied in your secondary school and college courses) are a good source of such examples.

2. a. The first question was all but answered in the section. Expressing the answer as a function of the number of bits can be done by using the formula relating the two metrics.

   b. How can we compute $(ab) \bmod m$?

3. It helps to have done the exercises in question.

4. a. The most straightforward algorithm, which is based on substituting x_0 into the formula, is quadratic.

   b. Analyzing what unnecessary computations the quadratic algorithm does should lead you to a better (linear) algorithm.

   c. How many coefficients does a polynomial of degree n have? Can one compute its value at an arbitrary point without processing all of them?

5. Just trace the algorithm on the input given. (It was done for another input in the section.)

6. Although the majority of elementary sorting algorithms are stable, do not rush with your answer. A general remark about stability made in Section 1.3, where the notion of stability is introduced, could be helpful, too.

7. Generally speaking, implementing an algorithm for a linked list poses problems if the algorithm requires accessing the list's elements not in a sequential order.

8. Just trace the algorithm on the input given. (See an example in the section.)

9. a. A list is sorted if and only if all its adjacent elements are in a correct order. Why?

   b. Add a boolean flag to register the presence or absence of switches.

   c. Identify worst-case inputs first.

10. Can bubble sort change the order of two equal elements in its input?

11. Thinking about the puzzle as a sorting-like problem may and may not lead you to the most simple and efficient solution.

Solutions to Exercises 3.1

1. a. Euclid's algorithm and the standard algorithm for finding the binary representation of an integer are examples from the algorithms previously mentioned in this book. There are, of course, many more examples in its other chapters.

   b. Solving nonlinear equations or computing definite integrals are examples of problems that cannot be solved exactly (except for special instances) by any algorithm.

2. a. $M(n) = n \approx 2^b$ where $M(n)$ is the number of multiplications made by the brute-force algorithm in computing a^n and b is the number of bits in the n's binary representation. Hence, the efficiency is linear as a function of n and exponential as a function of b.

   b. Perform all the multiplications modulo m, i.e.,

   $$a^i \bmod m = (a^{i-1} \bmod m \cdot a \bmod m) \bmod m \quad \text{for } i = 1, ..., n.$$

3. Problem 4 (computes $\sum_1^n i^2$): yes

   Problem 5 (computes the range of an array's values): yes

   Problem 6 (checks whether a matrix is symmetric): yes

4. a. Here is a pseudocode of the most straightforward version:

   Algorithm *BruteForcePolynomialEvaluation*($P[0..n]$, x)
   //The algorithm computes the value of polynomial P at a given point x
   //by the "highest-to-lowest term" brute-force algorithm
   //Input: Array $P[0..n]$ of the coefficients of a polynomial of degree n,
   // stored from the lowest to the highest and a number x
   //Output: The value of the polynomial at the point x
   $p \leftarrow 0.0$
   for $i \leftarrow n$ **downto** 0 **do**
   $power \leftarrow 1$
   for $j \leftarrow 1$ **to** i **do**
   $power \leftarrow power * x$
   $p \leftarrow p + P[i] * power$
   return p

   We will measure the input's size by the polynomial's degree n. The basic operation of this algorithm is a multiplication of two numbers; the number of multiplications $M(n)$ depends on the polynomial's degree only.

Although it is not difficult to find the total number of multiplications in this algorithm, we can count just the number of multiplications in the algorithm's inner-most loop to find the algorithm's efficiency class:

$$M(n) = \sum_{i=0}^{n} \sum_{j=1}^{i} 1 = \sum_{i=0}^{n} i == \frac{n(n+1)}{2} \in \Theta(n^2).$$

b. The above algorithm is very inefficient: we recompute powers of x again and again as if there were no relationship among them. Thus, the obvious improvement is based on computing consecutive powers more efficiently. If we proceed from the highest term to the lowest, we could compute x^{i-1} by using x^i but this would require a division and hence a special treatment for $x = 0$. Alternatively, we can move from the lowest term to the highest and compute x^i by using x^{i-1}. Since the second alternative uses multiplications instead of divisions and does not require any special treatment for $x = 0$, it is both more efficient and cleaner. It leads to the following algorithm:

Algorithm *BetterBruteForcePolynomialEvaluation*($P[0..n]$, x)
//The algorithm computes the value of polynomial P at a given point x
//by the "lowest-to-highest term" algorithm
//Input: Array $P[0..n]$ of the coefficients of a polynomial of degree n,
// from the lowest to the highest, and a number x
//Output: The value of the polynomial at the point x
$p \leftarrow P[0]$; $power \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
  $power \leftarrow power * x$
  $p \leftarrow p + P[i] * power$
return p

The number of multiplications here is

$$M(n) = \sum_{i=1}^{n} 2 = 2n$$

(while the number of additions is n), i.e., we have a linear algorithm.

Note: Horner's Rule discussed in Section 6.5 needs only n multiplications (and n additions) to solve this problem.

c. No, because any algorithm for evaluating an arbitrary polynomial of degree n at an arbitrary point x must process all its $n + 1$ coefficients. (Note that even when $x = 1$, $p(x) = a_n + a_{n-1} + ... + a_1 + a_0$, which needs at least n additions to be computed correctly for arbitrary $a_n, a_{n-1}, ..., a_0$.)

5.

$$
\begin{array}{|ccccccc}
E & X & \mathbf{A} & M & P & L & E \\
A & X & \mathbf{E} & M & P & L & E \\
A & E & X & M & P & L & \mathbf{E} \\
A & E & E & M & P & \mathbf{L} & X \\
A & E & E & L & P & \mathbf{M} & X \\
A & E & E & L & M & \mathbf{P} & X \\
A & E & E & L & M & P & X
\end{array}
$$

6. Selection sort is not stable: In the process of exchanging elements that are not adjacent to each other, the algorithm can reverse an ordering of equal elements. The list $2'$, $2''$, 1 is such an example.

7. Yes. Both operations—finding the smallest element and swapping it—can be done as efficiently with a linked list as with an array.

8. E, X, A, M, P, L, E

$$
\begin{array}{ccccccccccc}
E & \overset{?}{\leftrightarrow} & X & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E \\
E & & A & & X & \overset{?}{\leftrightarrow} & M & & P & & L & & E \\
E & & A & & M & & X & \overset{?}{\leftrightarrow} & P & & L & & E \\
E & & A & & M & & P & & X & \overset{?}{\leftrightarrow} & L & & E \\
E & & A & & M & & P & & L & & X & \overset{?}{\leftrightarrow} & E \\
E & & A & & M & & P & & L & & E & & |X \\
E & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E & & \\
A & & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & P & \overset{?}{\leftrightarrow} & L & & E & & \\
A & & E & & M & & L & & P & \overset{?}{\leftrightarrow} & E & & \\
A & & E & & M & & L & & E & & |P & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & L & & E & & & & \\
A & & E & & L & & M & \overset{?}{\leftrightarrow} & E & & & & \\
A & & E & & L & & E & & |M & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & \overset{?}{\leftrightarrow} & E & & & & & & \\
A & & E & & E & & |L & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & & & & & &
\end{array}
$$

The algorithm can be stopped here (see the next question).

9. a. Pass i ($0 \le i \le n-2$) of bubble sort can be represented by the following diagram:

$$A_0, ..., A_j \overset{?}{\leftrightarrow} A_{j+1}, ..., A_{n-i-1} \le | \underbrace{A_{n-i} \le ... \le A_{n-1}}_{\text{in their final positions}}$$

If there are no swaps during this pass, then

$$A_0 \le A_1 \le ... \le A_j \le A_{j+1} \le ... \le A_{n-i-1},$$

with the larger (more accurately, not smaller) elements in positions $n-i$ through $n-1$ being sorted during the previous iterations.

b. Here is a pseudocode for the improved version of bubble sort:

Algorithm *BetterBubbleSort*$(A[0..n-1])$
//The algorithm sorts array $A[0..n-1]$ by improved bubble sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in ascending order
count $\leftarrow n-1$ //number of adjacent pairs to be compared
sflag $\leftarrow$ **true** //swap flag
while *sflag* **do**
    sflag $\leftarrow$ **false**
    for $j \leftarrow 0$ **to** *count*-1 **do**
        if $A[j+1] < A[j]$
            swap $A[j]$ and $A[j+1]$
            sflag $\leftarrow$ **true**
    count $\leftarrow$ *count* -1

c. The worst-case inputs will be strictly decreasing arrays. For them, the improved version will make the same comparisons as the original version, which was shown in the text to be quadratic.

10. Bubble sort is stable. It follows from the fact that it swaps adjacent elements only, provided $A[j+1] < A[j]$.

5. Here is a simple and efficient (in fact, optimal) algorithm for this problem: Starting with the first and ending with the last light disk, swap it with each of the i $(1 \le i \le n)$ dark disks to the left of it. The ith iteration of the algorithm can be illustrated by the following diagram, in which 1s and 0s correspond to the dark and light disks, respectively.

$$\underbrace{00..0}_{i-1}\underbrace{11..1}_{i-1}\mathbf{1}0\,1..10 \quad \Rightarrow \quad \underbrace{00..0}_{i}\mathbf{0}\underbrace{11..11}_{i}0..10$$

The total number of swaps made is equal to $\sum_{i=1}^{n} i = n(n+1)/2$.

Exercises 3.2

1. Find the number of comparisons made by the sentinel version of sequential search

   a. in the worst case.

   b. in the average case if the probability of a successful search is p ($0 \leq p \leq 1$).

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 ($0 \leq p \leq 1$) for which this formula yields the largest value of $C_{avg}(n)$ and the smallest value of $C_{avg}(n)$.

3. *Gadgets 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 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) are made by the brute-force string-matching algorithm in searching for each of the following patterns in the binary text of 1000 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 the worst-case input for the brute-force string-matching algorithm. Exactly how many character comparisons are made for such input?

7. Write a visualization program 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, the same cell may be used no more than once. Write a computer program for solving this puzzle.

11. *Battleship game* Write a program for playing Battleship (a classic strategy game) on the computer which is based on a version of brute-force pattern matching. 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. A 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 this player misses. 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.)

Hints to Exercises 3.2

1. Modify the analysis of the algorithm's version in Section 2.1.
2. As a function of p, what kind of function is C_{avg}?
3. Solve a simpler problem with a single gadget first. Then design a better than linear algorithm for the problem with two gadgets.
4. The content of this quote from Mahatma Gandhi is more thought provoking than this drill.
5. For each input, one iteration of the algorithm yields all the information you need to answer the question.
6. It will suffice to limit your search for an example to binary texts and patterns.
7. You may use either bit strings or a natural language text for the visualization program. It would be a good idea to implement, as an option, a search for all occurrences of a given pattern in a given text.
8. The answer, surprisingly, is yes.
9. a. For a given occurrence of A in the text, what are the substrings you need to count?

   b. For a given occurrence of B in the text, what are the substrings you need to count?
10. Test your program thoroughly. Be especially careful about a possibility of words read diagonally with wrapping around the table's border.
11. A (very) brute-force algorithm can simply shoot at adjacent feasible cells starting at, say, one of the corners of the board. Can you suggest a better strategy? (You can investigate relative efficiencies of different strategies by making two programs implementing them play each other.) Is your strategy better than the one that shoots at randomly generated cells of the opponent's board?

Solutions to Exercises 3.2

1. a. $C_{worst}(n) = n + 1$.

   b. $C_{avg}(n) = \frac{(2-p)(n+1)}{2}$. In the manner almost identical to the analysis in Section 2.1, we obtain

$$\begin{aligned} C_{avg}(n) &= [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + ... + i \cdot \frac{p}{n} + ... + n \cdot \frac{p}{n}] + (n+1) \cdot (1-p) \\ &= \frac{p}{n}[1 + 2 + ... + i + ... + n] + (n+1)(1-p) \\ &= \frac{p}{n}\frac{n(n+1)}{2} + (n+1)(1-p) = \frac{(2-p)(n+1)}{2}. \end{aligned}$$

2. The expression

$$\frac{p(n+1)}{2} + n(1-p) = p\frac{n+1}{2} + n - np = n - p(n - \frac{n+1}{2}) = n - \frac{n-1}{2}p$$

   is a linear function of p. Since the p's coefficient is negative for $n > 1$, the function is strictly decreasing on the interval $0 \le p \le 1$ from n to $(n+1)/2$. Hence $p = 0$ and $p = 1$ are its maximum and minimum points, respectively, on this interval. (Of course, this is the answer we should expect: The average number of comparisons should be the largest when the probability of a successful search is 0, and it should be the smallest when the probability of a successful search is 1.)

3. Drop the first gadget from floors $\lceil\sqrt{n}\rceil$, $2\lceil\sqrt{n}\rceil$, and so on until either the floor $i\lceil\sqrt{n}\rceil$ a drop from which makes the gadget malfunction is reached or no such floor in this sequence is encountered before the top of the building is reached. In the former case, the floor to be found is higher than $(i-1)\lceil\sqrt{n}\rceil$ and lower than $i\lceil\sqrt{n}\rceil$. So, drop the second gadget from floors $(i-1)\lceil\sqrt{n}\rceil + 1$, $(i-1)\lceil\sqrt{n}\rceil + 2$, and so on until the first floor a drop from which makes the gadget malfunction is reached. The floor immediately preceeding that floor is the floor in question. If no drop in the first-pass sequence resulted in the gadget's failure, the floor in question is higher than $i\lceil\sqrt{n}\rceil$, the last tried floor of that sequence. Hence, continue the successive examination of floors $i\lceil\sqrt{n}\rceil + 1$, $i\lceil\sqrt{n}\rceil + 2$, and so on until either a failure is registered or the last floor is reached. The number of times the two gadgets are dropped doesn't exceed $\lceil\sqrt{n}\rceil + \lceil\sqrt{n}\rceil$, which puts it in $O(\sqrt{n})$.

4. 43 comparisons.
   The algorithm will make $47 - 6 + 1 = 42$ trials: In the first one, the G of the pattern will be aligned against the first T of the text; in the last one, it will be aligned against the last space. On each but one trial, the algorithm will make one unsuccessful comparison; on one trial—when the G of the pattern is aligned against the G of the text —it will make two

comparisons. Thus, the total number of character comparisons will be $41 \cdot 1 + 1 \cdot 2 = 43$.

5. a. For the pattern `00001`, the algorithm will make four successful and one unsuccessful comparison on each of its trials and then shift the pattern one position to the right:

```
0 0 0 0 0 0                                   0 0 0 0 0
0 0 0 0 1
  0 0 0 0 1
                  etc.
                                              0 0 0 0 1
```

The total number of character comparisons will be $C = 5 \cdot 996 = 4980$.

b. For the pattern `10000`, the algorithm will make one unsuccessful comparison on each of its trials and then shift the pattern one position to the right:

```
0 0 0 0 0 0                                   0 0 0 0 0
1 0 0 0 0
  1 0 0 0 0
                  etc.
                                              1 0 0 0 0
```

The total number of character comparisons will be $C = 1 \cdot 996 = 996$.

c. For the pattern `01010`, the algorithm will make one successful and one unsuccessful comparison on each of its trials and then shift the pattern one position to the right:

```
0 0 0 0 0 0                                   0 0 0 0 0
0 1 0 1 0
  0 1 0 1 0
                  etc.
                                              0 1 0 1 0
```

The total number of character comparisons will be $C = 2 \cdot 996 = 1,992$.

6. The text composed of n zeros and the pattern $\underbrace{0 \ldots 0}_{m-1}1$ is an example of the worst-case input. The algorithm will make $m(n - m + 1)$ character comparisons on such input.

7. n/a

8. Comparing pairs of the pattern and text characters righ-to-left can allow farther pattern shifts after a mismatch. This is the main insight the two string matching algoirthms discussed in Section 7.2 are based on. (As a specific example, consider searching for the pattern 11111 in the text of one thousand zeros.)

9. a. Note that the number of desired substrings that starts with an A at a given position i $(0 \leq i < n-1)$ in the text is equal to the number of B's to the right of that position. This leads to the follwing simple algorithm:

   Initialize the count of the desired substrings to 0. Scan the text left to right doing the following for every character except the last one: If an A is encountered, count the number of all the B's following it and add this number to the count of desired substrings. After the scan ends, return the last value of the count.

   For the worst case of the text composed of n A's, the total number of character comparisons is

$$n + (n-1) + ... + 2 = n(n+1)/2 - 1 \in \Theta(n^2).$$

   b. Note that the number of desired substrings that ends with a B at a given position i $(0 < i \leq n-1)$ in the text is equal to the number of A's to the left of that position. This leads to the follwing algorithm:

   Initialize the count of the desired substrings and the count of A's encountered to 0. Scan the text left to right until the text is exhausted and do the following. If an A is encountered, increment the A's count; if a B is encountered, add the current value of the A's count to the desired substring count. After the text is exhausted, return the last value of the desired substring count.

   Since the algorithm makes a single pass through a given text spending constant time on each of its characters, the algorihtm is linear.

10. n/a

11. n/a

Exercises 3.3

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

2. Let $x_1 < x_2 < ... < x_n$ 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 particular, the ***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 that 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 in a field (Euclidean plane) so that each has a unique nearest neighbor. Each person has a cream pie. At 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 in k-dimensional space in which the Euclidean distance between two points $P' = (x'_1, ..., x'_k)$ and $P'' =$

$(x''_1, ..., x''_k)$ is defined as

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

What is the time-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 given 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:

$$\begin{array}{rl} \text{maximize} & 3x + 5y \\ \text{subject to} & x + y \leq 4 \\ & x + 3y \leq 6 \\ & x \geq 0, \;\; y \geq 0 \end{array}$$

    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.

Hints to Exercises 3.3

1. Sorting n real numbers can be done in $O(n \log n)$ time.

2. a. Solving the problem for $n = 2$ and $n = 3$ should lead you to the critical insight.

   b. Where would you put the post office if it would not have to be at one of the village locations?

3. a. Check requirements (i)–(iii) by using basic properties of absolute values.

   b. For the Manhattan distance, the points in question are defined by equation $|x - 0| + |y - 0| = 1$. You can start by sketching the points in the positive quadrant of the coordinate system (i.e., the points for which $x, y \geq 0$) and then sketch the rest by using the symmetries.

   c. The assertion is false. You can choose, say, $P_1(0, 0)$, $P_2(1, 0)$ and find P_3 to complete a counterexample.

4. True; prove it by mathematical induction.

5. Your answer should be a function of two parameters: n and k. A special case of this problem (for $k = 2$) was solved in the text.

6. Review the examples given in the section.

7. Some of the extreme points of a convex hull are easier to find than others.

8. If there are other points of a given set on the straight line through P_i and P_j, which of all these points need to be preserved for further processing?

9. Your program should work for any set of n distinct points, including sets with many colinear points.

10. a. The set of points satisfying inequality $ax + by \leq c$ is the half-plane of the points on one side of the straight line $ax + by = c$, including all the points on the line itself. Sketch such a half-plane for each of the inequalities and find their intersection.

    b. The extreme points are the vertices of the polygon obtained in part a.

    c. Compute and compare the values of the objective function at the extreme points.

Solutions to Exercises 3.3

1. Sort the numbers in ascending order, compute the differences between adjacent numbers in the sorted list, and find the smallest such difference. If sorting is done in $O(n \log n)$ time, the running time of the entire algorithm will be in
$$O(n \log n) + \Theta(n) + \Theta(n) = O(n \log n).$$

2. a. If we put the post office at location x_i, the average distance between it and all the points $x_1 < x_2 < ... < x_n$ is given by the formula $\frac{1}{n} \sum_{j=1}^{n} |x_j - x_i|$. Since the number of points n stays the same, we can ignore the multiple $\frac{1}{n}$ and minimize $\sum_{j=1}^{n} |x_j - x_i|$. We'll have to consider the cases of even and odd n separately.

Let n be even. Consider first the case of $n = 2$. The sum $|x_1 - x| + |x_2 - x|$ is equal to $x_2 - x_1$, the length of the interval with the endpoints at x_1 and x_2, for any point x of this interval (including the endpoints), and it is larger than $x_2 - x_1$ for any point x outside of this interval. This implies that for any even n, the sum

$$\sum_{j=1}^{n} |x_j - x| = [|x_1 - x| + |x_n - x|] + [|x_2 - x| + |x_{n-1} - x|] + ... + [|x_{n/2} - x| + |x_{n/2+1} - x|]$$

is minimized when x belongs to each of the intervals $[x_1, x_n] \supset [x_2, x_{n-1}] \supset ... \supset [x_{n/2}, x_{n/2+1}]$. If x must be one of the points given, either $x_{n/2}$ or $x_{n/2+1}$ solves the problem.

Let $n > 1$ be odd. Then, the sum $\sum_{j=1}^{n} |x_j - x|$ is minimized when $x = x_{\lceil n/2 \rceil}$, the point for which the number of the given points to the left of it is equal to the number of the given points to the right of it.

Note that the point $x_{\lceil n/2 \rceil}$—the $\lceil n/2 \rceil$th smallest called the *median*—solves the problem for even n's as well. For a sorted list implemented as an array, the median can be found in $\Theta(1)$ time by simply returning the $\lceil n/2 \rceil$th.element of the array. (Section 5.6 provides a more general discussion of algorithms for computing the median.)

b. Assuming that the points $x_1, x_2, ..., x_n$ are given in increasing order, the answer is the point x_i that is the closest to $m = (x_1 + x_n)/2$, the middle point between x_1 and x_n. (The middle point would be the obvious solution if the post-post office didn't have to be at one of the given locations.) Indeed, if we put the post office at any location x_i to the left of m, the longest distance from a village to the post office would be $x_n - x_i$; this distance is minimal for the rightmost among such points. If we put the post office at any location x_i to the right of m, the longest distance from a village to the post office would be $x_i - x_1$; this distance is minimal for the leftmost among such points.

Algorithm *PostOffice1*(P)
//Input: List P of n $(n \geq 2)$ points $x_1, x_2, ..., x_n$ in increasing order

//Output: Point x_i that minimizes $\max\limits_{1\le j\le n} |x_j - x_i|$ among all $x_1, x_2, ..., x_n$
$m \leftarrow (x_1 + x_n)/2$
$i \leftarrow 1$
while $x_i < m$ **do**
$\quad i \leftarrow i + 1$
if $x_i - x_1 < x_n - x_{i-1}$
$\quad$ **return** x_i
else return x_{i-1}

The time efficiency of this algorithm is $O(n)$.

3. a. For $d_M(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$, we have the following:

(i) $d_M(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2| \ge 0$ and $d_M(P_1, P_2) = 0$ if and only if both $x_1 = x_2$ and $y_1 = y_2$, i.e., P_1 and P_2 coincide.

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

(iii) $d_M(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$
$= |(x_1 - x_3) + (x_3 - x_2)| + |(y_1 - y_3) + (y_3 - y_2)|$
$\le |x_1 - x_3| + |x_3 - x_2| + |y_1 - y_3| + |y_3 - y_2| = d(P_1, P_3) + d(P_3, P_2)$.

b. For the Manhattan distance, the points in question are defined by the equation

$$|x - 0| + |y - 0| = 1, \text{ i.e., } |x| + |y| = 1.$$

The graph of this equation is the boundary of the square with its vertices at $(1, 0)$, $(0, 1)$, $(-1, 0)$, and $(-1, -1)$.

For the Euclidean distance, the points in question are defined by the equation

$$\sqrt{(x-0)^2 + (y-0)^2} = 1, \text{ i.e., } x^2 + y^2 = 1.$$

The graph of this equation is the circumference of radius 1 and the center at $(0, 0)$.

c. False. Consider points $P_1(0, 0)$, $P_2(1, 0)$, and, say, $P_3(\frac{1}{2}, \frac{3}{4})$. Then

$$d_E(P_1, P_2) = 1 \text{ and } d_E(P_3, P_1) = d_E(P_3, P_2) = \sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{3}{4}\right)^2} < 1.$$

Therefore, for the Euclidean distance, the two closest points are either P_1 and P_3 or P_2 and P_3. For the Manhattan distance, we have

$$d_M(P_1, P_2) = 1 \text{ and } d_M(P_3, P_1) = d_M(P_3, P_2) = \frac{1}{2} + \frac{3}{4} = \frac{5}{4} > 1.$$

Therefore, for the Manhattan distance, the two closest points are P_1 and P_2.

4. We'll prove by induction that there will always remain at least one person not hit by a pie. The basis step is easy: If $n = 3$, the two persons with the smallest pairwise distance between them throw at each other, while the third person throws at one of them (whoever is closer). Therefore, this third person remains "unharmed".
   For the inductive step, assume that the assertion is true for odd $n \geq 3$, and consider $n + 2$ persons. Again, the two persons with the smallest pairwise distance between them (the closest pair) throw at each other. Consider two possible cases as follows. If the remaining n persons all throw at one another, at least one of them remains "unharmed" by the inductive assumption. If at least one of the remaining n persons throws at one of the closest pair, among the remaining $n - 1$ persons, at most $n - 1$ pies are thrown at one another, and hence at least one person must remain "unharmed" because there is not enough pies to hit everybody in that group. This completes the proof.

5. The number of squarings will be

$$\begin{aligned} C(n,k) &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{s=1}^{k} 1 = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} k = k\sum_{i=1}^{n-1}(n-i) \\ &= k[(n-1)+(n-2)+....+1] = \frac{k(n-1)n}{2} \in \Theta(kn^2). \end{aligned}$$

6. a. The convex hull of a line segment is the line segment itself; its extreme points are the endpoints of the segment.

   b. The convex hull of a square is the square itself; its extreme points are the four vertices of the square.

   c. The convex hull of the boundary of a square is the region comprised of the points within that boundary and on the boundary itself; its extreme points are the four vertices of the square.

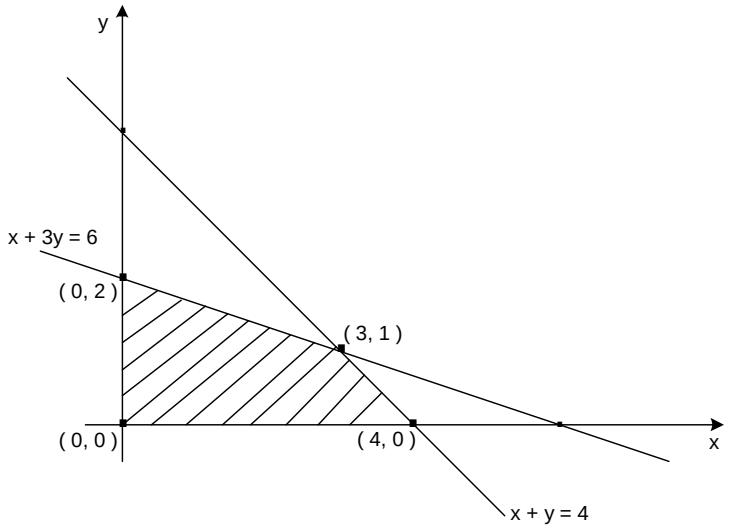
   d. The convex hull of a straight line is the straight line itself. It doesn't have any extreme points.

7. Find the point with the smallest x coordinate; if there are several such points, find the one with the smallest y coordinate among them. Similarly, find the point with the largest x coordinate; if there are several such points, find the one with the largest y coordinate among them.

8. If there are other points of a given set on the straight line through P_i and P_j (while all the other points of the set lie on the same side of the line), a line segment of the convex hull's boundary will have its end points at the two farthest set points on the line. All the other points on the line can be eliminated from further processing.

9. n/a

10. a. Here is a sketch of the feasible region in question:



b. The extreme points are: $(0, 0)$, (4,0), (3,1), and (0,2).

c.

Extreme point	Value of $3x + 5y$
(0,0)	0
(4,0)	12
(3,1)	14
(0,2)	10

So, the optimal solution is (3, 1), with the maximal value of $3x + 5y$ equal to 14. (Note: This instance of the linear programming problem is discussed further in Section 10.1.)

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 one 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 an 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. 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.

   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 ***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 the 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 correspondence between letters and digits is one-to-one, that is 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'e puzzle the way it was expected to be solved when it was first published in 1924.

Hints to Exercises 3.4

1. a. Identify the algorithm's basic operation and count the number of times it will be executed.

   b. For each of the time amounts given, find the largest value of n for which this limit will not be exceeded.

2. How different is the traveling salesman problem from the problem of finding a Hamiltonian circuit?

3. Your algorithm should check the well-known conditions that are both necessary and sufficient for the existence of a Eulerian circuit in a connected graph.

4. Generate the remaining $4! - 6 = 18$ possible assignments, compute their costs, and find the one with the smallest cost.

5. Make the size of your counterexample as small as possible.

6. Rephrase the problem so that the sum of elements in one subset, rather than two, needs to be checked on each try of a possible partition.

7. Follow the definitions of a clique and of an exhaustive search algorithm.

8. Try all possible orderings of the elements given.

9. a. Add all the elements in the magic square in two different ways.

   b. What combinatorial objects do you have to generate here?

10. a. For testing, you may use alphametic collections available on the Internet.

    b. Given the absence of electronic computers in 1924, you must refrain here from using the Internet.

Solutions to Exercises 3.4

1. a. $\Theta(n!)$
   For each tour (a sequence of $n+1$ cities), one needs n additions to compute the tour's length. Hence, the total number of additions $A(n)$ will be n times the total number of tours considered, i.e., $n*\frac{1}{2}(n-1)! = \frac{1}{2}n! \in \Theta(n!)$.

   b. (i) $n_{\max} = 15$; (ii) $n_{\max} = 16$; (iii) $n_{\max} = 18$; (iv) $n_{\max} = 20$.
   Given the answer to part a, we have to find the largest value of n such that
   $$\frac{1}{2}n!10^{-9} \leq t$$
   where t is the time available (in seconds). Thus, for $t = 1\text{hr} = 3.6*10^3\text{sec}$, we get the inequality
   $$n! \leq 2 * 10^9 t = 7.2 * 10^{12}.$$
   The largest value of n for which this inequality holds is 15 (since $15! \approx 1.3 * 10^{12}$ and $16! \approx 2.1 * 10^{13}$).

   For the other given values of t, the answers can be obtained in the same manner.

2. The problem of finding a Hamiltonian circuit is very similar to the traveling salesman problem. Generate permutations of n vertices that start and end with, say, the first vertex, and check whether every pair of successive vertices in a current permutation are connected by an edge. If it's the case, the current permutation represents a Hamiltonian circuit, otherwise, a next permutation needs to be generated.

3. A connected graph has a Eulerian circuit if and only if all its vertices have even degrees. An algorithm should check this condition until either an odd vertex is encountered (then a Eulerian circuit doesn't exist) or all the vertices turn out to be even (then a Eulerian circuit must exist). For a graph (with no loops) represented by its n-by-n adjacency matrix, the degree of a vertex is the number of ones in the vertex's row. Thus, computing its degree will take the $\Theta(n)$ time, checking whether it's even will take $\Theta(1)$ time, and it will be done between 1 and n times. Hence, the algorithm's efficiency will be in $O(n^2)$.

4. The following assignments were generated in the chapter's text:
   $$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

1, 2, 3, 4	cost = 9+4+1+4 = 18
1, 2, 4, 3	cost = 9+4+8+9 = 30
1, 3, 2, 4	cost = 9+3+8+4 = 24
1, 3, 4, 2	cost = 9+3+8+6 = 26
1, 4, 2, 3	cost = 9+7+8+9 = 33
1, 4, 3, 2	cost = 9+7+1+6 = 23

   etc.

The remaining ones are

2, 1, 3, 4	cost = 2+6+1+4 = 13
2, 1, 4, 3	cost = 2+6+8+9 = 25
2, 3, 1, 4	cost = 2+3+5+4 = 14
2, 3, 4, 1	cost = 2+3+8+7 = 20
2, 4, 1, 3	cost = 2+7+5+9 = 23
2, 4, 3, 1	cost = 2+7+1+7 = 17
3, 1, 2, 4	cost = 7+6+8+4 = 25
3, 1, 4, 2	cost = 7+6+8+6 = 27
3, 2, 1, 4	cost = 7+4+5+4 = 20
3, 2, 4, 1	cost = 7+4+8+7 = 26
3, 4, 1, 2	cost = 7+7+5+6 = 25
3, 4, 2, 1	cost = 7+7+8+7 = 29
4, 1, 2, 3	cost = 8+6+8+9 = 31
4, 1, 3, 2	cost = 8+6+1+6 = 21
4, 2, 1, 3	cost = 8+4+5+9 = 26
4, 2, 3, 1	cost = 8+4+1+7 = 20
4, 3, 1, 2	cost = 8+3+5+6 = 22
4, 3, 2, 1	cost = 8+3+8+7 = 26

The optimal solution is: Person 1 to Job 2, Person 2 to Job 1, Person 3 to Job 3, and Person 4 to Job 4, with the total (minimal) cost of the assignment being 13.

5. Here is a very simple example:

$$\begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix}$$

6. Start by computing the sum S of the numbers given. If S is odd, stop because the problem doesn't have a solution. If S is even, generate the subsets until either a subset whose elements' sum is $S/2$ is encountered or no more subsets are left. Note that it will suffice to generate only subsets with no more than $n/2$ elements.

7. Generate a subset of k vertices and check whether every pair of vertices in the subset is connected by an edge. If it's true, stop (the subset is a clique); otherwise, generate the next subset.

8. Generate a permutation of the elements given and check whether they are ordered as required by comparing values of its consecutive elements. If they are, stop; otherwise, generate the next permutation. Since the

number of permutations of n items is equal to $n!$ and checking a permutation requires up to $n-1$ comparisons, the algorithm's efficiency class is in $O(n!(n-1)) = O((n+1)!)$.

9. a. Let s be the sum of the numbers in each row of an n-by-n magic square. Let us add all the numbers in rows 1 through n. We will get the following equality:

$$sn = 1+2+...+n^2, \text{ i.e., } sn = \frac{n^2(n^2+1)}{2}, \text{ which implies that } s = \frac{n(n^2+1)}{2}.$$

b. Number positions in an n-by-n matrix from 1 through n^2. Generate a permutation of the numbers 1 through n^2, put them in the corresponding positions of the matrix, and check the magic-square equality (proved in part (a)) for every row, every column, and each of the two main diagonals of the matrix.

c. n/a

d. n/a

10. a. Since the letter-digit correspondence must be one-to-one and there are only ten distinct decimal digits, the exhaustive search needs to check $P(10,k) = 10!/(10-k)!$ possible substitutions, where k is the number of distinct letters in the input. (The requirement that the first letter of a word cannot represent 0 can be used to reduce this number further.) Thus a program should run in a quite reasonable amount of time on today's computers. Note that rather than checking two cases—with and without a "1-carry"—for each of the decimal positions, the program can check just one equality, which stems from the definition of the decimal number system. For Dudeney's alphametic, for example, this equality is 1000(S+M) + 100(E+O) + 10(N+R) + (D+E) = 10000M + 1000O + 100N + 10E + Y

b. Here is a "computerless" solution to this classic problem. First, notice that M must be 1. (Since both S and M are not larger than 9, their sum, even if increased by 1 because of the carry from the hundred column, must be less than 20.) We will have to rely on some further insigts into specifics of the problem. The leftmost digits of the addends imply one of the two possibilities: either S + M = 10 + O (if there was no carry from the hundred column) or 1 + S + M = 10 + O (if there was such a carry). First, let us pursue the former of the two possibilities. Since M = 1, S $\leq$ 9 and O $\geq$ 0, the equation S + 1 = 10 + O has only one solution: S = 9 and O = 0. This leaves us with

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

Since we deal here with the case of no carry from the hundreds and E and N must be distinct, the only possibility is a carry from the tens: 1 + E = N and either N + R = 10 + E (if there was no carry from the rightmost column) or 1 + N + R = 10 + E (if there was such a carry). The first combination leads to a contradiction: Substituting 1 + E for N into N + R = 10 + E, we obtain R = 9, which is incompatible with the same digit already represented by S. The second combination of 1 + E = N and 1 + N + R = 10 + E implies, after substituting the first of these equations into the second one, R = 8. Note that the only remaining digit values still unassigned are 2, 3, 4, 5, 6, and 7. Finally, for the righmost column, we have the equation D + E = 10 + Y. But 10 + Y $\geq$ 12, because the smallest unassigned digit value is 2 while D + E $\leq$ 12 because the two largest unassigned digit values are 6 and 7 and E < N. Thus, D + E = 10 + Y = 12. Hence Y = 2 and D + E = 12. The only pair of still unassigned digit values that add up to 12, 5 and 7, must be assigned to E and D, respectively, since doing this the other way (E = 7, D = 5) would imply N = E + 1 = 8, which is already represented by R. Thus, we found the following solution to the puzzle:

$$\begin{array}{r} 9\ 5\ 6\ 7 \\ +\ 1\ 0\ 8\ 5 \\ \hline 1\ 0\ 6\ 5\ 2 \end{array}$$

Is this the only solution? To answer this question, we should pursue the carry possibility from the hundred column to the thousand column (see above). Then 1 + S + M = 10 + O or, since M =1, S = 8 + O. But S $\leq$ 9, while 8 + O $\geq$ 10 since O $\geq$ 2. Hence the last equation has no solutions in our domain. This proves that the puzzle has no other solutions.