This file contains the exercises, hints, and solutions for Chapter 11 of the book "Introduction to the Design and Analysis of Algorithms," 3rd 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 11.1

1. Prove that any algorithm solving the alternating-disk puzzle (Problem 14 in Exercises 3.1) must make at least $n(n+1)/2$ moves to solve it. Is this lower bound tight?

2. Prove that the classic recursive algorithm for the Tower of Hanoi problem (Section 2.4) makes the minimum number of disk moves needed to solve the problem.

3. Find a trivial lower-bound class for each of the following problems and indicate, if you can, whether this bound is tight.

   a. Finding the largest element in an array

   b. Checking completeness of a graph represented by its adjacency matrix

   c. Generating all the subsets of a n-element set

   d. Determining whether n given real numbers are all distinct

4. Consider the problem of identifying a lighter fake coin among n identical-looking coins with the help of a balance scale. Can we use the same information-theoretic argument as the one in the text for the number of questions in the guessing game to conclude that any algorithm for identifying the fake will need at least $\lceil \log_2 n \rceil$ weighings in the worst case?

5. Prove that any comparison-based algorithm for finding the largest element of an n-element set of numbers must make $n-1$ comparisons in the worst case.

6. Find a tight lower bound for sorting an array by exchanging its adjacent elements.

7. ▷ Give an adversary argument proof that the time efficiency of any algorithm that checks connectivity of a graph with n vertices is in $\Omega(n^2)$, provided the only operation allowed for an algorithm is to inquire about the presence of an edge between two vertices of the graph. Is this lower bound tight?

8. What is the minimum number of comparisons needed for a comparison-based sorting algorithm to merge any two sorted lists of sizes n and $n+1$ elements, respectively? Prove the validity of your answer.

9. Find the product of matrices A and B through a transformation to a product of two symmetric matrices if

$$A = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix}.$$

10. a. Can one use this section's formulas that indicate the complexity equivalence of multiplication and squaring of integers to show the complexity equivalence of multiplication and squaring of square matrices?

    b. Show that multiplication of two matrices of order n can be reduced to squaring a matrix of order $2n$.

11. Find a tight lower bound class for the problem of finding two closest numbers among n real numbers x_1, x_2, ..., x_n.

12. Find a tight lower-bound class for the number placement problem (Problem 9 in Exercises 6.1).

Hints to Exercises 11.1

1. Is it possible to solve the puzzle by making fewer moves than the brute-force algorithm? Why?

2. Since you know that the number of disk moves made by the classic algorithm is $2^n - 1$, you can simply prove (e.g., by mathematical induction) that for any algorithm solving this problem the number of disk moves $M(n)$ made by the algorithm is greater than or equal to $2^n - 1$. Alternatively, you can show that if $M^*(n)$ is the minimum needed number of disk moves then $M^*(n)$ satisfies the recurrence relation

   $$M^*(n) = 2M^*(n-1) + 1 \text{ for } n > 1 \text{ and } M^*(1) = 1,$$

   whose solution is $2^n - 1$.

3. All these questions have straightforward answers. If a trivial lower bound is tight, don't forget to mention a specific algorithm that proves its tightness.

4. Reviewing Section 4.4, where the fake-coin problem was introduced, should help in answering the question.

5. Pay attention to comparison losers.

6. Think inversions.

7. Divide the set of vertices of an input graph into two disjoints subsets U and W having $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ vertices respectively and show that any algorithm will have to check for an edge between every pair of vertices (u, w), where $u \in U$ and $w \in W$, before the graph's connectivity can be established.

8. The question and the answer are quite similar to the case of two n-element sorted lists discussed in the section. So is the proof of the lower bound.

9. Simply follow the transformation formula suggested in the section.

10. a. Check whether the formulas hold for two arbitrary square matrices.

    b. Use a formula similar to the one showing that multiplication of arbitrary square matrices can be reduced to multiplication of symmetric matrices.

11. What problem with a known lower bound is most similar to the one in question? After finding an appropriate reduction, do not forget to indicate an algorithm that makes the lower bound tight.

12. Use the problem reduction method.

Solutions to Exercises 11.1

1. In the initial position of the puzzle, the ith light disk has exactly i dark disks to the left of it ($i = 1, 2, ..., n$). Hence the total number of the dark disks that are to the left of the light disks is initially equal to $\sum_{i=1}^{n} i = n(n+1)/2$. There are no dark disks to the left of a light disk in the final state of the puzzle. Since one move can only swap two neighboring disks—and hence decrease the total number of dark disks to the left of a light disk by one—the puzzle requires at least $n(n+1)/2$ moves to be solved. This bound is tight because the brute-force algorithm (see the solution to Problem 14 in Exercises 3.1) makes exactly this number of moves.

2. Let $M(n)$ be the number of disk moves made by some algorithm solving the Tower of Hanoi problem. We'll prove by induction that

$$M(n) \geq 2^n - 1 \text{ for } n \geq 1.$$

   For the basis case of $n = 1$, $M(1) \geq 2^1 - 1$ holds. Assume now that the inequality holds for $n \geq 1$ disks and consider the case of $n+1$ disks. Before the largest disk can be moved, all n smaller disks must be in a tower on another peg. By the inductive assumption, it will require at least $2^n - 1$ disk moves. Moving the largest disk to the destination peg will take at least 1 move. After the largest disk is moved on the destination peg for the last time, the n smaller disks will have to be moved from its tower formation on top of the largest disk, which will require at least $2^n - 1$ moves by the inductive assumption. Therefore:

$$M(n+1) \geq (2^n - 1) + 1 + (2^n - 1) = 2^{n+1} - 1.$$

   The alternative proof via setting a recurrence for the minimum number of moves $M^*(n)$ follows essentially the same logic as the proof above.

3. a. All n elements of a given array need to be processed to find its largest element (otherwise, if an unprocessed element is larger than all the others, the output cannot be correct) and just one item needs to be produced (if just the value of the largest element or a position of the largest element needs to be returned). Hence the trivial lower bound is linear. It is tight because the standard one-pass algorithm for this problem is in $\Theta(n)$.

   b. Since the existence of an edge between all $n(n-1)/2$ pairs of vertices needs to be verified in the worst case before establishing completeness of a graph with n vertices, the trivial lower bound is quadratic. It is tight because this is the amount of work done by the brute-force algorithm that simply checks all the elements in the upper-triangular part of the matrix until either a zero is encountered or no unchecked elements are left.

c. The size of the problem's output is 2^n. Hence, the lower bound is exponential. The bound is tight because efficient algorithms for subset generation (Section 4.3) spend a constant time on each of them (except, possibly, for the first one).

d. The size of the problem's input is n while the output is just one bit. Hence, the trivial lower bound is linear. It is not tight: according to the known result quoted in the section, the tight lower bound for this problem is $n \log n$.

4. The answer is no. The problem can be solved with fewer weighings by dividing the coins into three rather than two subsets with about the same number of coins each. The information-theoretic argument, if used, has to take into account the fact that one weighing reduces uncertainty more than for the number-guessing problem because it can have three rather than two outcomes.

5. Every comparison of two (distinct) elements produces one "winner" and one "loser". If less than $n-1$ comparisons are made, at least two elements will be with no losses. Hence, it would be impossible to say which of them is the largest element.

6. Recall that an inversion in an array is any pair of its elements that are out of order, i.e., $A[i] > A[j]$ while $i < j$. The maximum number of inversions in an n-element array is attained when its elements are strictly decreasing; this maximum is equal to $\sum_{i=2}^{n}(i-1) = (n-1)n/2$. Since a swap of two adjacent elements can decrease the total number of inversions only by one, the worst-case number of such swaps required for sorting an n-element array will be equal to $(n-1)n/2$ as well. This bound is tight because it is attained by both bubble sort and insertion sort.

7. Consider the following rule for the adversary to follow: Divide the set of vertices of an input graph into two disjoints subsets U and W having $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ vertices respectively (e.g., by putting the first $\lfloor n/2 \rfloor$ vertices into U and the remaining $\lceil n/2 \rceil$ vertices into W). Whenever an algorithm inquires about an edge between two vertices, reply yes if and only if both vertices belong to either U or to W. If the algorithm stops before inquiring about a pair of vertices (u, v), where $u \in U$ and $w \in W$, with the positive answer about the graph's connectivity, present the disconnected graph with all possible edges between vertices in U, all possible edges between vertices in W, and no edges between vertices of U and W, including u and w. If the algorithm stops before inquiring about a pair of vertices (u, w), where $u \in U$ and $w \in W$, with the negative answer

about the graph's connectivity, present the connected graph with all possible edges between vertices in U, all possible edges between vertices in W, and the edge between u and w. Hence, any correct algorithm must inquire about at least $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \in \Omega(n^2)$ possible edges in the worst case.

The quadratic lower bound is tight because a depth-first search traversal solves the problem in $O(n^2)$ time.

Note: In fact, all $n(n-1)/2$ potential edges need to be checked in the worst case (see [Bra96, Sec. 12.3.2]).

8. Any comparison-based algorithm will need at least $2n$ comparisons to merge two arbitrary sorted lists of sizes n and $n+1$, respectively. The proof is obtained via the adversary argument similar to the one given in the section for the case of two n-element lists.

9. For $A = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix}$, the respective transposes are

$$A^T = \begin{bmatrix} 1 & 2 \\ -1 & 3 \end{bmatrix} \text{ and } B^T = \begin{bmatrix} 0 & -1 \\ 1 & 2 \end{bmatrix}.$$

Hence, $X = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$ and $Y = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}$ are, respectively,

$$X = \begin{bmatrix} 0 & 0 & 1 & -1 \\ 0 & 0 & 2 & 3 \\ 1 & 2 & 0 & 0 \\ -1 & 3 & 0 & 0 \end{bmatrix} \text{ and } Y = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ -1 & 2 & 0 & 0 \end{bmatrix}.$$

Thus,

$$XY = \begin{bmatrix} 0 & 0 & 1 & -1 \\ 0 & 0 & 2 & 3 \\ 1 & 2 & 0 & 0 \\ -1 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ -1 & 2 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -3 & 8 & 0 & 0 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 3 & 7 \end{bmatrix},$$

with the product AB produced in the upper left quadrant of XY.

10. a. The formula $XY = \frac{1}{4}[(X+Y)^2 - (X-Y)^2]$ does not hold for arbitrary square matrices because it relies on commutativity of multiplication (i.e., $XY = YX$):

$$\begin{aligned} \frac{1}{4}[(X+Y)^2 - (X-Y)^2] &= \frac{1}{4}[(X+Y)(X+Y) - (X-Y)(X-Y)] \\ &= \frac{1}{4}[2XY + 2YX] = \frac{1}{2}[XY + YX] \neq XY. \end{aligned}$$

b. The problem is solved by the following reduction formula

$$\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}^2 = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix},$$

which allows us to obtain products AB and BA as a result of matrix squaring.

11. The element uniqueness problem can be reduced to the closest numbers problem. (After solving the latter, it suffices to check whether the distance between the closest numbers is zero to solve the former.) This implies that $\Omega(n \log n)$, the lower bound for the element uniqueness problem, is also a lower bound for the closest numbers problem. The presorting-based algorithm (see Example 1 in Section 6.1)—with an $n \log n$ sorting method such as mergesort—makes this lower bound tight for the closest numbers problem as well.

12. Sorting a list of numbers is a special case of the number placement problem. This implies that $\Omega(n \log n)$, the lower bound for sorting, is also a lower bound for the number placement problem. The presorting-based algorithm for the latter (see the solution to Problem 9 in Exercises 6.1)—with an $n \log n$ sorting method such as mergesort or heapsort— makes this lower bound tight for the number placement problem as well.

Exercises 11.2

1. Prove by mathematical induction that

   a. $h \geq \lceil \log_2 l \rceil$ for any binary tree with height h and the number of leaves l.

   b. $h \geq \lceil \log_3 l \rceil$ for any ternary tree with height h and the number of leaves l.

2. Consider the problem of finding the median of a three-element set $\{a, b, c\}$ of orderable items.

   a. What is the information-theoretic lower bound for comparison-based algorithms solving this problem?

   b. Draw a decision tree for an algorithm solving this problem.

   c. If the worst-case number of comparisons in your algorithm is greater than the information-theoretic lower bound, do you think an algorithm matching the lower bound exists? (Either find such an algorithm or prove its impossibility).

3. Draw a decision tree and find the number of key comparisons in the worst and average cases for

   a. the three-element basic bubble sort.

   b. the three-element enhanced bubble sort (which stops if no swaps have been made on its last pass).

4. Design a comparison-based algorithm for sorting a four-element array with the smallest number of element comparisons possible.

5. ▶ Design a comparison-based algorithm for sorting a five-element array with seven comparisons in the worst case.

6. Draw a binary decision tree for searching a four-element ordered list by sequential search.

7. ▷ Compare the two lower bounds for searching a sorted array—$\lceil \log_3(2n+1) \rceil$ and $\lceil \log_2(n+1) \rceil$—to show that

   a. $\lceil \log_3(2n+1) \rceil \leq \lceil \log_2(n+1) \rceil$ for every positive integer n.

   b. $\lceil \log_3(2n+1) \rceil < \lceil \log_2(n+1) \rceil$ for every positive integer $n \geq n_0$.

8. What is the information-theoretic lower bound for finding the maximum of n numbers by comparison-based algorithms? Is this bound tight?

9. A ***tournament tree*** is a complete binary tree reflecting results of a "knockout tournament": its leaves represent n players entering the tournament, and each internal node represents a winner of a match played by the players represented by the node's children. Hence, the winner of the tournament is represented by the root of the tree.

   a. What is the total number of games played in such a tournament?

   b. How many rounds are there in such a tournament?

   c. Design an efficient algorithm to determine the second best player using the information produced by the tournament. How many extra games does your algorithm require?

10. *Advanced fake-coin problem* There are $n \geq 3$ coins identical in appearance; either all are genuine or exactly one of them is fake. It is unknown whether the fake coin is lighter or heavier than the genuine one. You have a balance scale with which you can compare any two sets of coins. That is, by tipping to the left, to the right, or staying even, the balance scale will tell whether the sets weigh the same or which of the sets is heavier than the other, but not by how much. The problem is to find whether all the coins are genuine and, if not, to find the fake coin and establish whether it is lighter or heavier then the genuine ones.

    a. Prove that any algorithm for this problem must make at least $\lceil \log_3(2n+1) \rceil$ weighings in the worst case.

    b. Draw a decision tree for an algorithm that solves the problem for $n = 3$ coins in two weighings.

    c. Prove that there exists no algorithm that solves the problem for $n = 4$ coins in two weighings.

    d. Draw a decision tree of an algorithm that solves the problem for $n = 4$ coins in two weighings by using an extra coin known to be genuine.

    e.▶ Draw a decision tree that solves the classic version of the problem—that for $n = 12$ coins in three weighings (with no extra coins being used).

11. *Jigsaw puzzle* A jigsaw puzzle contains n pieces. A "section" of the puzzle is a set of one or more pieces that have been connected to each other. A "move" consists of connecting two sections. What algorithm will minimize the number of moves required to complete the puzzle?

Hints to Exercises 11.2

1. a. Prove first that $2^h \geq l$ by induction on h.

   b. Prove first that $3^h \geq l$ by induction on h.

2. a. How many outcomes does the problem have?

   b. Of course, there are many ways to solve this simple problem.

   c. Thinking about a, b, and c as points on the real line should help.

3. This is a straightforward question. You may assume that three elements to be sorted are distinct. (If you need help, see decision trees for the three-element selection sort and three-element insertion sort in the section).

4. Compute a nontrivial lower bound for sorting a four-element array and then identify a sorting algorithm whose number of comparisons in the worst case matches the lower bound.

5. This is not an easy task. None of the standard sorting algorithms can do this. Try to design a special algorithm that squeezes as much information as possible from each of its comparisons.

6. This is a very straightforward question. Use the obvious observation that sequential search in a sorted array can be stopped as soon as an element larger than the search key is encountered.

7. a. Start by transforming the logarithms to the same base.

   b. The easiest way is to prove that

$$\lim_{n\to\infty} \frac{\lceil \log_2(n+1) \rceil}{\lceil \log_3(2n+1) \rceil} > 1.$$

   To get rid of the ceiling functions, you can use

$$\frac{f(n)-1}{g(n)+1} < \frac{\lceil f(n) \rceil}{\lceil g(n) \rceil} < \frac{f(n)+1}{g(n)-1}$$

   where $f(n) = \log_2(n+1)$ and $g(n) = \log_3(2n+1)$ and show that

$$\lim_{n\to\infty} \frac{f(n)-1}{g(n)+1} = \lim_{n\to\infty} \frac{f(n)+1}{g(n)-1} > 1.$$

8. The answer to the first question follows directly from inequality (11.1). The answer to the second is no (why?).

9. a. Think losers.

   b. Think the height of the tournament tree or, alternatively, the number of steps needed to reduce an n-element set to a one-element set by halving.

   c. After the winner has been determined, which player can be the second best?

10. a. How many outcomes does this problem have?

    b. Draw a ternary decision tree that solves the problem.

    c. Show that each of the two cases—weighing two coins (one on each cup of the scale) or four coins (two on each cup of the scale)—yields at least one situation with more than three outcomes still possible. The latter cannot be resolved uniquely with a single weighing.[1]

    d. Decide first whether you should start with weighing two coins. Do not forget that you can take advantage of the extra coin known to be genuine.

    e. This is a famous puzzle. The principal insight is that of the solution to part d.

11. If you want to solve the problem in the spirit of the section, represent a process of assembling the puzzle by a binary tree.

[1] This approach of using an information-theoretic reasoning for the problem was suggested by Brassard and Bratley [BB96].

Solutions to Exercises 11.2

1. a. We'll prove by induction on h that that $2^h \geq l$ for any nonempty binary tree with height $h \geq 0$ and the number of leaves l. For the basis case of $h = 0$, we have $2^0 \geq 1$. For the general case, assume that $2^h \geq l$ holds for any binary tree whose height doesn't exceed h. Consider an arbitrary binary tree T of height $h+1$; let T_L and T_R be its left and right subtrees, respectively. (One of T_L and T_R, but not both of them, can be empty.) The heights of T_L and T_R cannot exceed h, and the number of leaves in T is equal to the number of leaves in T_L and T_R. Whether or not T_L is empty, $l(T_L) \leq 2^h$. Indeed, if it's not empty, it is true by the inductive assumption; if it is empty, $l(T_L) = 0$ and therefore smaller than 2^h. By the same reasons, $l(T_R) \leq 2^h$. Hence, we obtain the following inequality for the number of leaves $l(T)$ in T:

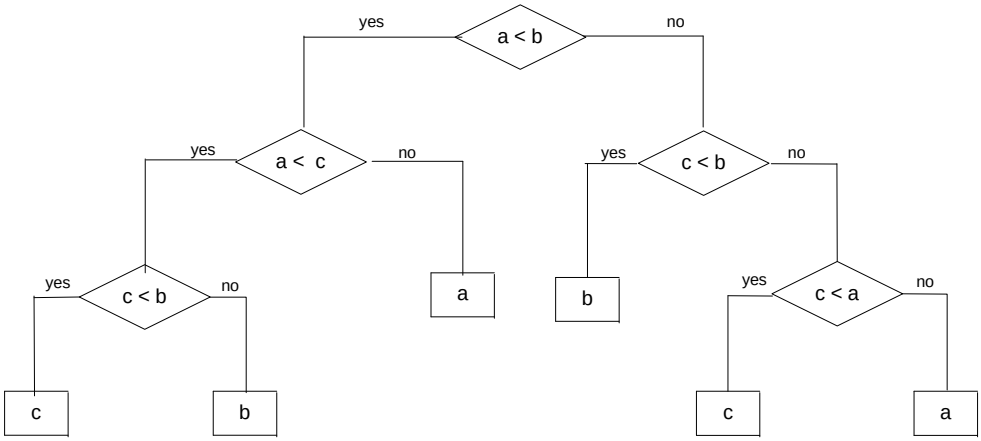
$$l(T) = l(T_L) + l(T_R) \leq 2^h + 2^h = 2^{h+1}.$$

   Taking binary logarithms of both hand sides of the proved inequality $l \leq 2^h$ yields $\log_2 l \leq h$ and, since h is an integer, $h \geq \lceil \log_2 l \rceil$.

   b. The proof is analogous to the one given for part a.

2. a. Since the problem has three possible outcomes, the information-theoretic lower bound is $\lceil \log_2 3 \rceil = 2$.

   b. Here is a decision tree for an algorithm for the problem of finding the median of a three-element set $\{a, b, c\}$:



   Note: The problem can also be solved by sorting a, b, c by one of the sorting algorithms.

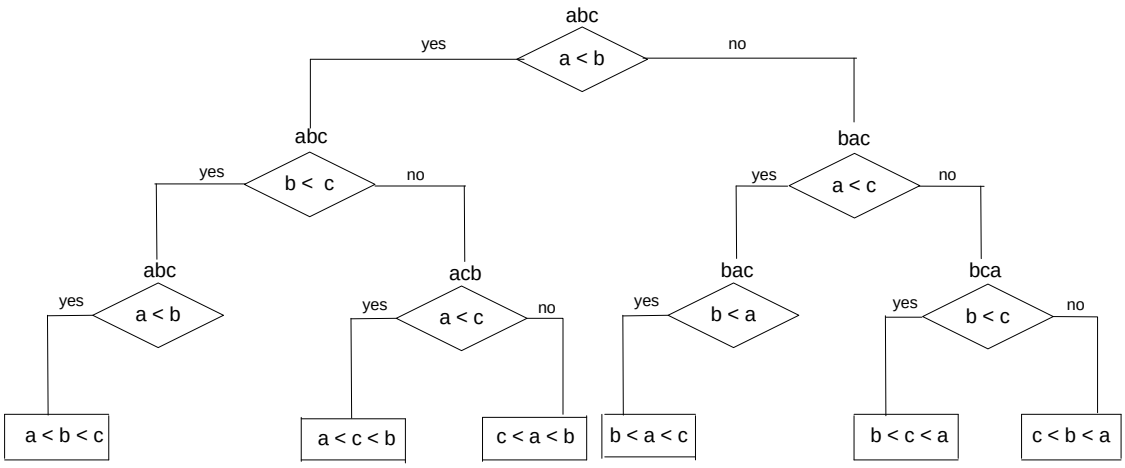
   c. It is impossible to find the median of three arbitrary real numbers in less than three comparisons (with a comparison-based algorithm). The first comparison will establish endpoints of some interval. The second

comparison (unless it's the same as the first one) will compare the third number with one of these endpoints. If that third number is compared to the left end of the interval and it is larger than it, it would be impossible to say without one more comparison whether this number or the right endpoint is the median. Similarly, if the third number is compared with the right end of the interval and it is smaller than it, it would be impossible to say without one more comparison whether this number or the left endpoint is the median. (This proof can be rephrased as an adversary argument.)

Note: As mentioned in Section 11.1, the following lower bound is known for comparison-based algorithms for finding the median of an n-element set (see, e.g., [Baa00, p.240]):

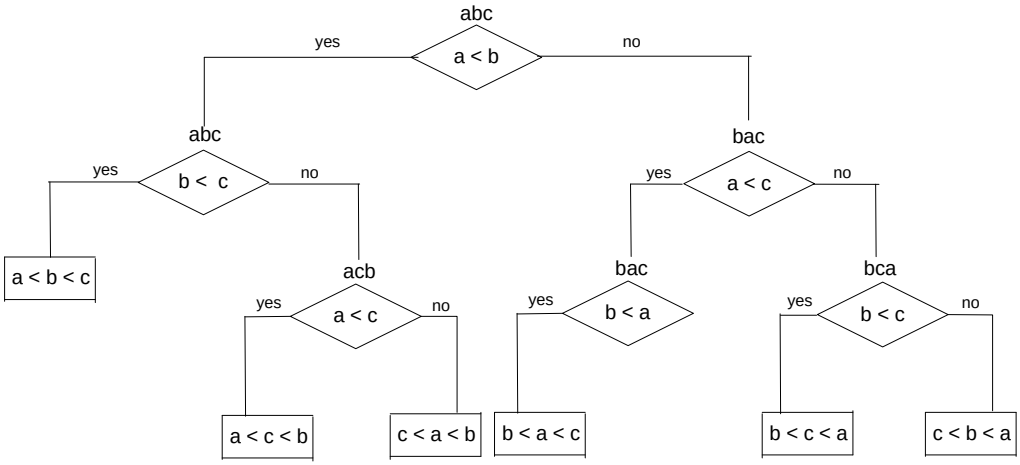
$$C_w(n) \geq \frac{3}{2}(n-1) \quad \text{for odd } n.$$

3. a. Here is a decision tree for sorting an array of three distinct elements a, b, and c by basic bubble sort:



The algorithm makes exactly three comparisons on any of its inputs.

b. Here is a decision tree for sorting an array of three distinct elements by enhanced bubble sort:



The algorithm makes three comparisons in the worst case and $(2+3+3+3+3+3)/6 = 2\frac{5}{6}$ comparisons in the average case.

4. $\lceil \log_2 4! \rceil = 5$. Mergesort (as described in Section 5.1) sorts any four-element array with no more than five comparisons: two comparisons are made to sort the two halves and no more than three comparisons are needed to merge them.

5. The following solution is presented by Knuth [KnuIII, pp. 183–184]. Let a, b, c, d, e be five distinct elements to be sorted. First, we compare a with b and c with d. Without loss of generality we can assume that $a < b$ and $c < d$. This can be depicted by the following digraph of five vertices, in which a path indicates an ordered subsequence:

$$\begin{array}{ccc} b & & d \\ \uparrow & & \uparrow \\ a & & c \end{array} \qquad e$$

Then we compare the larger elements of the two pairs; without loss of generality, we can assume that $b < d$:

$$\begin{array}{ccc} b & \rightarrow & d \\ \uparrow & & \uparrow \\ a & & c \end{array} \qquad e$$

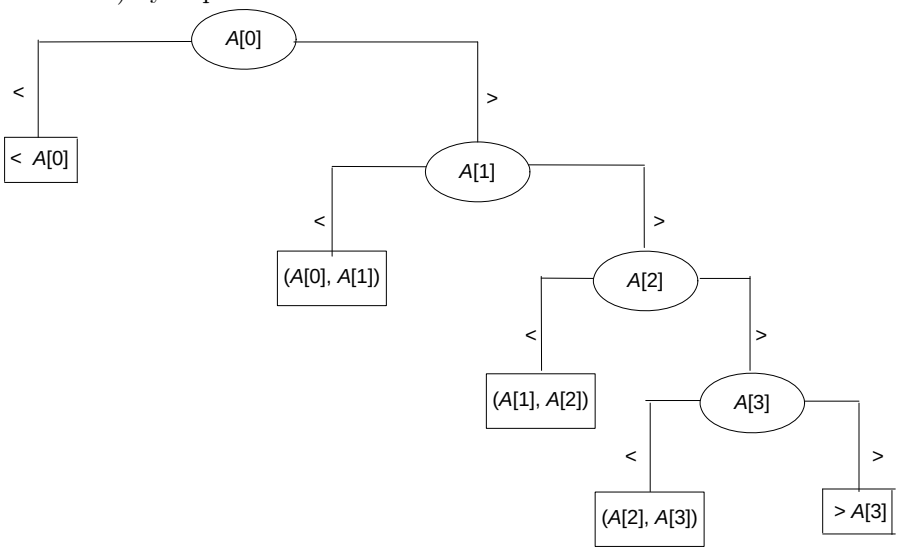
So far, we performed 3 comparisons. Now, we insert e in its appropriate position in the ordered subsequence $a < b < d$; it can be done in 2 comparisons if we start by comparing e with b. This results in one of the four

following situations:

$$\begin{array}{llll} e \to a \to b \to d & a \to e \to b \to d & a \to b \to e \to d & a \to b \to d \to e \\ \uparrow & \uparrow & \uparrow & \uparrow \\ c & c & c & c \end{array}$$

What remains to be done is to insert c in its appropriate position. Taking into account the information that $c < d$, it can be done in 2 comparisons in each of the four cases depicted above. (For example, for the first case, compare c with a and then, depending on the result, with either e or b.)

6. Here is a decision tree for searching a four-element ordered list (indexed from 0 to 3) by sequential search:



7. a. Since $a \le b$ implies $\lceil a \rceil \le \lceil b \rceil$ (because $a \le b \le \lceil b \rceil$, and therefore, since $\lceil b \rceil$ is an integer, $\lceil a \rceil \le \lceil b \rceil$ by the definition of $\lceil a \rceil$), it will suffice to show that

$$\log_3(2n+1) \le \log_2(n+1) \text{ for every } n \ge 1.$$

Using elementary properties of logarithms, we obtain the following chain of equivalent inequalities:

$$\begin{array}{rcl} \dfrac{\log_2(2n+1)}{\log_2 3} & \le & \log_2(n+1) \\ \log_2(2n+1) & \le & \log_2 3 \log_2(n+1) \\ \log_2(2n+1) & \le & \log_2(n+1)^{\log_2 3} \end{array}$$

$$(2n+1) \le (n+1)^{\log_2 3}.$$

One way to prove that the last inequality holds for every $n \geq 1$ is to consider the function

$$f(x) = (x+1)^{\log_2 3} - (2x+1)$$

and verify that $f(1) = 2^{\log_2 3} - 3 = 0$ and $f'(1) = \log_2 3 (1+1)^{\log_2 3 - 1} - 2 = \log_2 3 \cdot 2^{\log_2 (3/2)} - 2 = \log_2 3 \cdot (3/2) - 2 > 0$.

b. The easiest way is to prove a stronger assertion:

$$\lim_{n\to\infty} \frac{\lceil \log_2 (n+1) \rceil}{\lceil \log_3 (2n+1) \rceil} > 1,$$

which implies that

$$\frac{\lceil \log_2 (n+1) \rceil}{\lceil \log_3 (2n+1) \rceil} > 1 \quad \text{for every } n \geq n_0.$$

To get rid of the ceiling functions, we can use

$$\frac{f(n)-1}{g(n)+1} < \frac{\lceil f(n) \rceil}{\lceil g(n) \rceil} < \frac{f(n)+1}{g(n)-1}$$

where $f(n) = \log_2 (n+1)$ and $g(n) = \log_3 (2n+1)$ for $n > 1$ and show that

$$\lim_{n\to\infty} \frac{f(n)-1}{g(n)+1} = \lim_{n\to\infty} \frac{f(n)+1}{g(n)-1} > 1.$$

Indeed, using l'Hôpital's Rule, we obtain the following:

$$\begin{aligned} \lim_{n\to\infty} \frac{f(n)-1}{g(n)+1} &= \lim_{n\to\infty} \frac{\log_2 (n+1) - 1}{\log_3 (2n+1) + 1} \\ &= \lim_{n\to\infty} \frac{\frac{1}{n+1} \cdot \frac{1}{\ln 2}}{\frac{1}{2n+1} \cdot \frac{2}{\ln 3}} = \frac{\ln 3}{\ln 2} = \log_2 3. \end{aligned}$$

Computing the limit of $\frac{f(n)+1}{g(n)-1}$ in the exactly same way yields the same result. Therefore, according to a well-known theorem of calculus,

$$\lim_{n\to\infty} \frac{\lceil f(n) \rceil}{\lceil g(n) \rceil} = \lim_{n\to\infty} \frac{\lceil \log_2 (n+1) \rceil}{\lceil \log_3 (2n+1) \rceil} = \log_2 3 > 1,$$

and hence there exists n_0 such that

$$\frac{\lceil \log_2 (n+1) \rceil}{\lceil \log_3 (2n+1) \rceil} > 1 \text{ for every } n \geq n_0.$$

8. Since any of the n numbers can be the maximum, the number of leaves in a decision tree of any algorithm solving the problem will be at least n. Hence, according to inequality (11.1), the lower bound for the number of comparisons made by any comparison-based algorithm in the worst case is $\lceil \log_2 n \rceil$. This bound is not tight. At least $n-1$ comparisons are needed because at least $n-1$ numbers should "lose" their comparions before the maximum is found.

9. a. Before a winner can be determined, $n-1$ players must lose a game. Since each game results in one loser, $n-1$ games are played in a single-elimination tournament with n players.

   b. The tournament tree has n leaves (the number of players) and $n-1$ internal nodes (the number of games—see part a). Hence the total number of nodes in this binary tree is $2n-1$, and, since it is complete, its height h is equal to

   $$h = \lfloor \log_2(2n-1) \rfloor = \lceil \log_2(2n-1+1) \rceil - 1 = \lceil \log_2 2 + \log_2 n \rceil - 1 = \lceil \log_2 n \rceil.$$

   Alternatively, one can solve the recurrence relation for the number of rounds

   $$R(n) = R(\lceil n/2 \rceil) + 1 \quad \text{for } n > 1, \quad R(1) = 0,$$

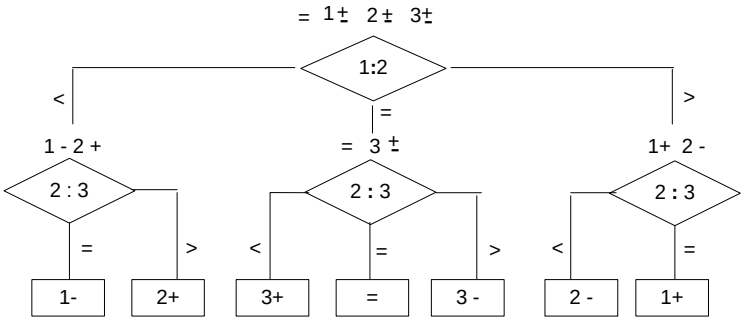
   to obtain $R(n) = \lceil \log_2 n \rceil$.

   c. The second best player can be any player who lost to the winner and nobody else. These players can be made play their own knock-out tournament by following the path from the leaf representing the winner to the root and assuming that the winner lost the first match. This will require no more than $\lceil \log_2 n \rceil - 1 = \lfloor \log_2(n-1) \rfloor$ games.

10. a. Since each of the n coins can be either lighter or heavier than all the others and all the coins can be genuine, the total number of possible outcomes is $2n+1$. Since each weighing can have three results, the smallest number of weighings must be at least $\lceil \log_3(2n+1) \rceil$.

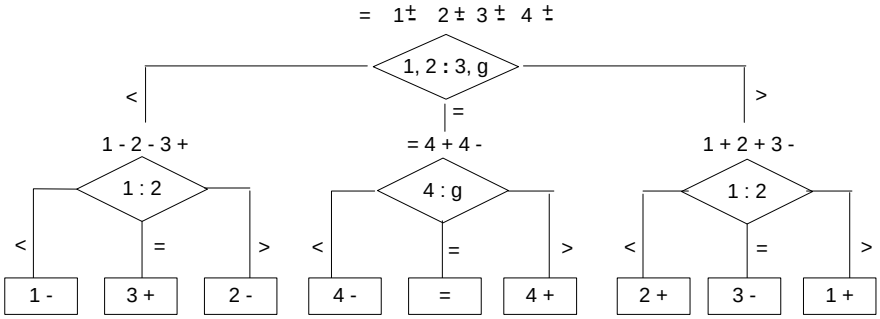
    b. In the decision tree below, the coins are numbered from 1 to 3. Internal nodes indicate weighings, with the coins being weighted listed inside the node. (E.g., the root corresponds to the first weighing in which coin 1 and 2 are put on the left and right cup of the scale, respectively). Edges to a node's children are marked according to the node's weighing outcome: $<$ means that the left cup's weight is smaller than that of the right's cup; $=$ means that the weights are equal; $>$ means that the left cup's weight is larger than that of the right cup. Leaves indicate final outcomes: $=$ means that all the coins are genuine, a particular coin followed by $+$ or $-$

means this coins is heavier or lighter, respectively. A list above an internal node indicates the outcomes that are still possible before the weighing indicated inside the node. For example, before coins 1 and 2 are put on the left and right cups, respectively, in the first weighing of the algorithm, the list = 1± 2± 3± indicates that either all the coins are genuine (=), or coin 1 is either heavier or lighter (1±) or coin 2 is either heavier or lighter (2±) or coin 3 is either heavier or lighter (3±).



c. There are two reasonable possibilities for the first weighing: to weigh two coins (i.e., one on each cup of the scale) and to weigh four coins (i.e., two on each cup). If an algorithm weighs two coins and they weigh the same, five outcomes remain possible for the two other coins. If an algorithm weighs four coins and they don't weigh the same, four outcomes remain possible. Since more than three possible outcomes cannot be resolved by one weighing, no algorithm will be able to solve the puzzle with the total of two weighings.

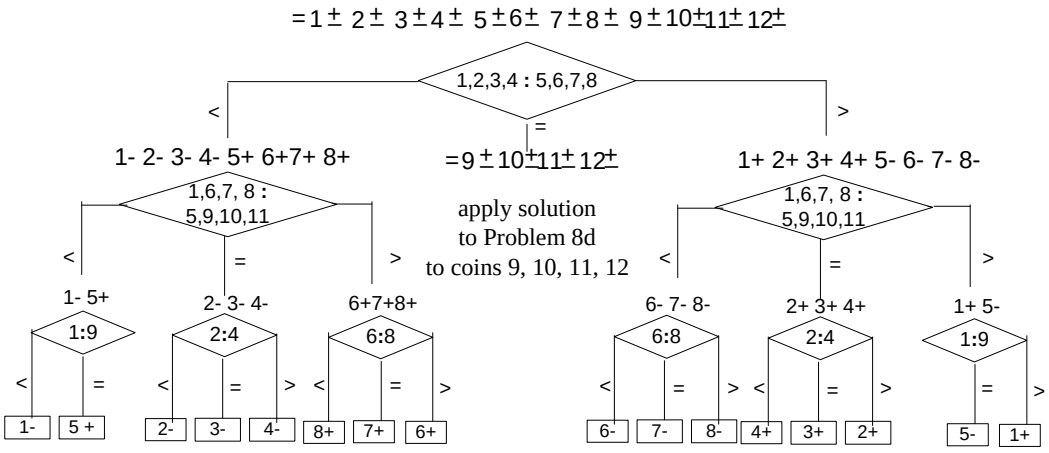
d. Here is a decision three of an algorithm that solves the fake-coin puzzle for $n = 4$ in two weighings by using an extra coin (denoted g) known to be genuine:



(The coins are assumed to be numbered form 1 to 4. All possible outcomes before a weighing are listed above the weighing's diamond box. A

plus or minus next to a coin's number means that the coin is heavier or lighter than the genuine one, respectively; the equality sign in the leaf means that all the coins are genuine.)

e. Here is a decision tree of an algorithm that solves the fake-coin puzzle for 12 coins in three weighings



Note: This problem is discussed on many puzzle-related sites on the Internet (see, e.g., http://www.cut-the-knot.com/blue/weight1.shtml). The attractiveness of the solution given above lies in its symmetry: the second weighings involve the same coins if the first one tips the scale either way, and the subsequent round of weighings involves the same pairs of coins. (In fact, the problem has a completely non-adaptive solution, i.e., a choice of what to put on the balance for the second weighing doesn't depend on the outcome of the first one, and a choice of what to weigh in the third round doesn't depend on what happened on either the first or second weighing.)

11. Any algorithm to assemble the puzzle can be represented by a binary tree whose leaves represents the single pieces and internal nodes represent connecting two sections (its children). Since each internal node in such a tree has two children, the leaves can be interpreted as extended nodes of a tree formed by its internal nodes. According to equality (4.5), the number of internal nodes (moves) is equal to $n - 1$, one less than the number of leaves (single pieces), for any such tree (assembling algorithm).

Note 1: Alternatively, one can reason that the task starts with n one-piece sections and ends with a single section. Since each move connects two sections and hence decreases the total number of sections by 1, the total

number of moves made by any algorithm that doesn't disconnect already connected pieces is equal to $n - 1$.
Note 2: This puzzle was published in *Mathematics Magazine*, vol. 26, p. 169. See also Problem 11 in Exercises 5.3 for a better known version of this puzzle (chocolate bar puzzle).

Exercises 11.3

1. A game of chess can be posed as the following decision problem: given a legal positioning of chess pieces and information about which side is to move, determine whether that side can win. Is this decision problem decidable?

2. A certain problem can be solved by an algorithm whose running time is in $O(n^{\log_2 n})$. Which of the following assertions is true?

   a. The problem is tractable.

   b. The problem is intractable.

   c. Impossible to tell.

3. Give examples of the following graphs or explain why such examples cannot exist.

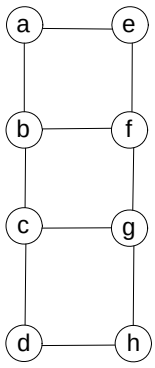
   a. graph with a Hamiltonian circuit but without an Eulerian circuit

   b. graph with an Eulerian circuit but without a Hamiltonian circuit

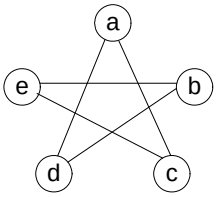
   c. graph with both a Hamiltonian circuit and an Eulerian circuit

   d. graph with a cycle that includes all the vertices but with neither a Hamiltonian circuit nor an Eulerian circuit

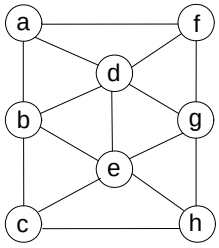
4. For each of the following graphs, find its chromatic number.



(a)



(b)



(c)

5. Design a polynomial-time algorithm for the graph 2-colorability problem: determine whether vertices of a given graph can be colored in two colors so that no two adjacent vertices are colored the same color.

6. Consider the following brute-force algorithm for solving the composite number problem: Check successive integers from 2 to $\lfloor n/2 \rfloor$ as possible divisors of n. If one of them divides n evenly, return yes (i.e., the number is composite), if none of them does, return no. Why does this algorithm not put the problem in class P?

7. State the decision version for each of the following problems and outline a polynomial-time algorithm that verifies whether or not a proposed solution solves the problem. (You may assume that a proposed solution represents a legitimate input to your verification algorithm.)

   a. knapsack problem

   b. bin packing problem

8. ▷ Show that the partition problem is polynomially reducible to the decision version of the knapsack problem.

9. ▷ Show that the following three problems are polynomially reducible to each other.

   (i) Determine, for a given graph $G = \langle V, E \rangle$ and a positive integer $m \leq |V|$, whether G contains a ***clique*** of size m or more. (A clique of size k in a graph is its complete subgraph of k vertices.)

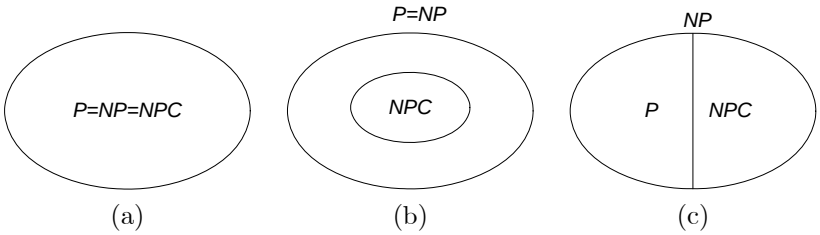
   (ii) Determine, for a given graph $G = \langle V, E \rangle$ and a positive integer $m \leq |V|$, whether there is a ***vertex cover*** of size m or less for G. (A vertex cover of size k for a graph $G = \langle V, E \rangle$ is a subset $V' \subseteq V$ such that $|V'| = k$ and, for each edge $(u, v) \in E$, at least one of u and v belongs to V'.)

   (iii) Determine, for a given graph $G = \langle V, E \rangle$ and a positive integer $m \leq |V|$, whether G contains an ***independent set*** of size m or more. (An independent set of size k for a graph $G = \langle V, E \rangle$ is a subset $V' \subseteq V$ such that $|V'| = k$ and for all $u, v \in V'$, vertices u and v are *not* adjacent in G.)

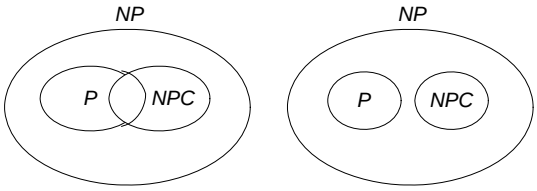
10. Determine whether the following problem is *NP*-complete. Given several sequences of uppercase and lowercase letters, is it possible to select a letter from each sequence without selecting both the upper- and lowercase versions of any letter? For example, if the sequences are Abc, BC, aB, and ac, it is possible to choose A from the first sequence, B from the second and third, and c from the fourth. An example where there is no way to make the required selections is given by the four sequences AB, Ab, aB, and ab. [Kar86]

11. ▷ Which of the following diagrams do not contradict the current state of our knowledge about the complexity classes *P*, *NP*, and *NPC* (*NP*-

complete problems)?



(a) (b) (c)



(d) (e)

12. King Arthur expects 150 knights for an annual dinner at Camelot. Unfortunately, some of the knights quarrel with each other, and Arthur knows who quarrels with whom. Arthur wants to seat his guests around a table so that no two quarreling knights sit next to each other.

    a. Which standard problem can be used to model King Arthur's task?

    b. As a research project, find a proof that Arthur's problem has a solution if each knight does not quarrel with at least 75 other knights.

Hints to Exercises 11.3

1. Check the definition of a decidable decision problem.
2. First, determine whether $n^{\log_2 n}$ is a polynomial function. Then, read carefully the definitions of tractable and intractable problems.
3. All four combinations are possible and none of the examples needs to be large.
4. Simply use the definition of the chromatic number. Solving Problem 5 first might be helpful but not necessary.
5. Use a depth-first search forest (or a breadth-first search forest) of a given graph.
6. What is a proper measure of an input's size for this problem?
7. See the formulation of the decision version of graph coloring and the verification algorithm for the Hamiltonian circuit problem given in the section.
8. You may start by expressing the partition problem as a linear equation with 0–1 variables x_i, $i = 1, ..., n$.
9. If you are not familiar with the notions of a clique, vertex cover, and independent set, it would be a good idea to start by finding a maximum-size clique, a minimum-size vertex cover, and a maximum-size independent set for a few simple graphs such as those in Problem 4. As far as Problem 9 is concerned, try to find a relationship between these three notions. You will find it useful to consider the *compliment* of your graph, which is the graph with the same vertices and the edges between vertices that are *not* adjacent in the graph itself.
10. The same problem in a different wording can be found in the section.
11. Just two of them do not contradict the current state of our knowledge about the complexity classes.
12. The problem you need is mentioned explicitly in the section.

Solutions to Exercises 11.3

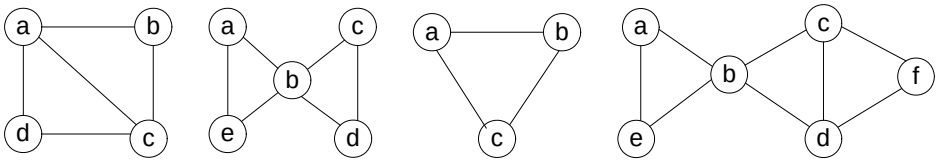
1. Yes, it's decidable. Theoretically, we can simply generate all the games (i.e., all the sequences of all legal moves of both sides) from the position given and check whether one of them is a win for the side that moves next.

2. First, $n^{\log_2 n}$ grows to infinity faster than any polynomial function n^k. This follows immediately from the fact that while the exponent k of n^k is fixed, the exponent $\log_2 n$ of $n^{\log_2 n}$ grows to infinity. (More formally:

$$\lim_{n\to\infty} \frac{n^k}{n^{\log_2 n}} = \lim_{n\to\infty} n^{k-\log_2 n} = 0.)$$

   We cannot say whether or not this algorithm solves the problem in polynomial time because the O-notation doesn't preclude the possibility that this algorithm is, in fact, polynomial-time. Even if this algorithm is not polynomial, there might be another which is. Hence, the correct answer is (c).

3. Here are examples of graphs required:



(a) (b) (c) (d)

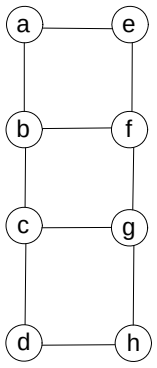
   a. Graph (a) has a Hamiltonian circuit ($a-b-c-d-a$) but no Eulerian circuit because it has vertices of odd degrees (a and c).

   b. Graph (b) has an Eulerian circuit ($a-b-c-d-b-e-a$) but no Hamiltonian circuit. If a Hamiltonian circuit existed, we could consider a as its starting vertex without loss of generality. But then it would have to visit b at least twice: once to reach c and the other to return to a.

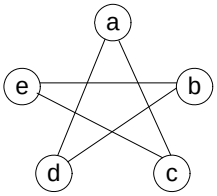
   c. Graph (c) has both a Hamiltonian circuit and an Eulerian circuit ($a-b-c-a$).

   d. Graph (d) has a cycle that includes all the vertices ($a-b-c-f-d-b-e-a$). It has neither a Hamiltonian circuit (by the same reason graph (b) doesn't) nor an Eulerian circuit (because vertices c and d have odd degrees).

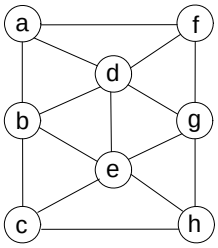
4. The chromatic numbers for the graphs below are 2, 3, and 4, respectively.



(a)



(b)



(c)

a. The chromatic number of graph (a) is 2 because we can assign color 1 to vertices a, f, c, and h and color 2 to vertices e, b, g, and d. (Note: This is an example of a bipartite graph. A *bipartite graph* is a graph whose vertices can be partitioned into two disjoint sets V_1 and V_2 so that every edge of the graph connects a vertex from V_1 and a vertex from V_2 . A graph is bipartite if and only if it doesn't have a cycle of an odd length.)

b. The chromatic number of graph (b) is 3. It needs at least 3 colors because it has an odd-length cycle: $a - c - e - b - d - a$ and 3 colors will suffice if we assign, for example, color 1 to vertices a and b, color 2 to vertices c and d, and color 3 to vertex e.

c. The chromatic number of graph (c) is 4. It needs at least 3 colors because it has cycles of length 3. But 3 colors will not suffice. If we try to use just 3 colors and assign, without loss of generality, colors 1, 2, and 3 to vertices a, f, and d, then we'll have to assign color 2 to vertex b and color 1 to vertex g. But then vertex e, being adjacent to b, d, and g, will require a different color, i.e., color 4. Coloring e with color 4 does yield a legitimate coloring with four colors, with c and h colored with, say, colors 1 and 3, respectively. Hence the chromatic number of this graph is equal to 4.

5. Consider a depth-first search forest obtained by a DFS traversal of a given graph. If the DFS forest has no back edges, the graph's vertices can be colored in two colors by alternating the colors on odd and even levels of the forest. (Hence, an acyclic graph is always 2-colorable.) If the DFS forest has back edges, it is clear that it is 2-colorable if and only if every back edge connects vertices on the levels of different parity: the one on an even level and the other on an odd level. This property can be checked

by a minor modification of depth-first search, which, of course, is a polynomial time algorithm.

Note: A breadth-first search forest can be used in the same manner. Namely, a graph is two colorable if and only if its BFS forest has no cross edges connecting vertices on the same level.

6. The brute-force algorithm is in $O(n)$. However, a proper measure of size for this problem is b, the number of bits in n's binary expansion. Since $b = \lfloor \log_2 n \rfloor + 1$, $O(n) = O(2^b)$.

7. a. Determine whether there is a subset of a given set of n items that fits into the knapsack and has a total value not less than a given positive integer m. To verify a proposed solution to this problem, sum up the weights and (separately) the values of the subset proposed: the weight sum should not exceed the knapsack's capacity, and the value sum should be not less than m. The time efficiency of this algorithm is obviously in $O(n)$.

   b. Determine whether one can place a given set of items into no more than m bins. A proposed solution can be verified by checking that the number of bins in this solution doesn't exceed m and that the sum of the sizes of the items assigned to the same bin doesn't exceed the bin capacity for each of the bins. The time efficiency of this algorithm is obviously in $O(n)$ where n is the number of items in the given instance.

8. The partition problem for n given integers s_i, $i = 1, ..., n$, can be expressed as a selection of 0–1 variables x_i, $i = 1, ..., n$ such that

$$\sum_{i=1}^{n} x_i s_i = \frac{1}{2} \sum_{i=1}^{n} s_i.$$

   This is equivalent to

$$2 \sum_{i=1}^{n} x_i s_i = \sum_{i=1}^{n} s_i \quad \text{or} \quad \sum_{i=1}^{n} x_i 2 s_i = \sum_{i=1}^{n} s_i.$$

   Denoting $w_i = 2s_i$, $W = \sum_{i=1}^{n} s_i$, and selecting $m = W$ as the value's lower bound in the decision knapsack problem, we obtain the following (equivalent) instance of the latter:

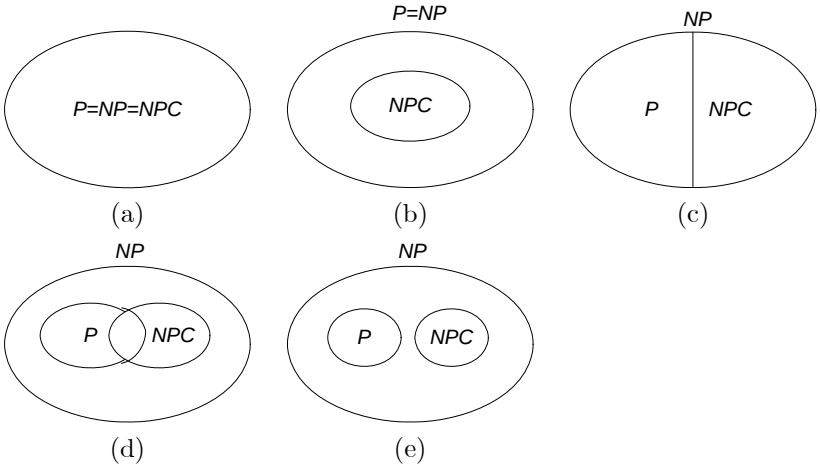
$$\sum_{i=1}^{n} x_i w_i \geq m \ (m = W)$$
$$\sum_{i=1}^{n} x_i w_i \leq W$$
$$x_i \in \{0, 1\} \text{ for } i = 1, ..., n.$$

9. The reductions follow from the following result (see, e.g., [Gar79]), which immediately follows from the definitions of the clique, independent set, vertex cover, and that of the complement graph: For any graph $G = \langle V, E \rangle$ and subset $V' \subseteq V$, the following statements are equivalent:

   i. V' is a vertex cover for G.
   ii. $V - V'$ is an independent set for G.
   iii. $V - V'$ is a clique in the complement of G.

10. Since the problem is a different wording of the CNF-satisfiability problem, it is *NP*-complete.

11. The given diagrams are:



   (a) is impossible because the trivial decision problem whose solution is "yes" for all inputs is clearly not *NP*-complete.

   (b) is possible (depicts the case of $P = NP$)

   (c) is impossible because, as mentioned in Section 11.3, there must exist problems in *NP* that are neither in P nor *NP*-complete, provided $P \neq NP$.

   (d) is impossible because it implies the existence of an *NP*-complete problem that is in P, which requires $P = NP$.

   (e) is possible (depicts the case of $P \neq NP$).

12. a. Create a graph in which vertices represent the knights and an edge connects two vertices if and only if the knights represented by the vertices can sit next to each other. Then a solution to King Arthur's problem exits if and only if the graph has a Hamiltonian circuit.

b. According to Dirac's theorem, a graph with $n \geq 3$ vertices has a Hamiltonian circuit if the degree of each of its vertices is greater than or equal to $n/2$. A more general sufficient condition—the sum of degrees for each pair of nonadjacent vertices is greater than or equal to n—is due to Ore (see, e.g., *Graph Theory and Its Applications* by J. Gross and J. Yellen, CRC Press, 1999).

Exercises 11.4

1. Some textbooks define the number of significant digits in the approximation of number α^* by number α as the largest nonnegative integer k for which
$$\frac{|\alpha - \alpha^*|}{|\alpha^*|} < 5 \cdot 10^{-k}.$$
According to this definition, how many significant digits are there in the approximation of π by
a. 3.1415? b. 3.1417?

2. If $\alpha = 1.5$ is known to approximate some number α^* with the absolute error not exceeding 10^{-2}, find

   a. the range of possible values of α^*.

   b. the range of the relative errors of these approximations.

3. Find the approximate value of $\sqrt{e} = 1.648721...$ obtained by the fifth-degree Taylor's polynomial about 0 and compute the truncation error of this approximation. Does the result agree with the theoretical prediction made in the section?

4. Derive formula (11.7) of the composite trapezoidal rule.

5. Use the composite trapezoidal rule with $n = 4$ to approximate the following definite integrals. Find the truncation error of each approximation and compare it with the one given by formula (11.9).

   a. $\int_0^1 x^2 dx$ b. $\int_1^3 \frac{1}{x} dx$

6. ▷ If $\int_0^1 e^{\sin x} dx$ is to be computed by the composite trapezoidal rule, how large should the number of subintervals be to guarantee a truncation error smaller than 10^{-4}? What about 10^{-6}?

7. Solve the two systems of linear equations and indicate whether they are ill-conditioned.
   a.
   $$\begin{aligned} 2x + 5y &= 7 \\ 2x + 5.000001y &= 7.000001 \end{aligned}$$
   b.
   $$\begin{aligned} 2x + 5y &= 7 \\ 2x + 4.999999y &= 7.000002 \end{aligned}$$

8. Write a computer program for solving equation $ax^2 + bx + c = 0$.

9. a.▶ Prove that for any nonnegative number D, the sequence of Newton's method for computing $\sqrt{D}$ is strictly decreasing and converges to $\sqrt{D}$ for any value of the initial approximation $x_0 > \sqrt{D}$.

b.▶ Prove that if $0.25 \leq D < 1$ and $x_0 = (1 + D)/2$, no more than four iterations of Newton's method are needed to guarantee that

$$|x_n - \sqrt{D}| < 4 \cdot 10^{-15}.$$

10. Apply four iterations of Newton's method to compute $\sqrt{3}$ and estimate the absolute and relative errors of this approximation.

Hints to Exercises 11.4

1. As the given definition of the number of significant digits requires, compute the relative errors of the approximations. One of the answers does not agree with our intuitive idea of this notion.

2. Use the definitions of the absolute and relative errors and properties of the absolute value.

3. Compute the value of $\sum_{i=0}^{5} \frac{0.5^i}{i!}$ and the magnitude of the difference between it and $\sqrt{e} = 1.648721....$

4. Apply the formula for the area of a trapezoid to each of the n approximating trapezoid strips and sum them up.

5. Apply formulas (11.7) and (11.9).

6. Find an upper bound for the second derivative of $e^{\sin x}$ and use formula (11.9) to find a value of n guaranteeing the truncation error smaller than a given error limit ε.

7. A similar problem is discussed in the section.

8. At the very least, your program should implement all the improvements mentioned in the section. Also, your program should handle correctly the case of $a = 0$.

9. a. Prove that every element x_n of the sequence is (i) positive, (ii) greater than $\sqrt{D}$ (by computing $x_{n+1} - \sqrt{D}$), and (iii) decreasing (by computing $x_{n+1} - x_n$). Then take the limit of both sides of equality (11.15) as n goes to infinity.

   b. Use the equality

$$x_{n+1} - \sqrt{D} = \frac{(x_n - \sqrt{D})^2}{2x_n}.$$

10. It is done for $\sqrt{2}$ in the section.

Solutions to Exercises 11.4

1. a. For $\alpha = 3.1415$, the relative error of the approximation is

$$\frac{|\alpha - \alpha^*|}{|\alpha^*|} = \frac{|3.1415 - \pi|}{\pi} \approx 2.9 \cdot 10^{-5}.$$

Since $2.9 \cdot 10^{-5} < 5 \cdot 10^{-5}$ but $2.9 \cdot 10^{-5} > 5 \cdot 10^{-6}$, the number of significant digits is 5.

b. For $\alpha = 3.1417$, the relative error of the approximation is

$$\frac{|\alpha - \alpha^*|}{|\alpha^*|} = \frac{|3.1417 - \pi|}{\pi} \approx 3.4 \cdot 10^{-5}.$$

Since $3.4 \cdot 10^{-5} < 5 \cdot 10^{-5}$ but $3.4 \cdot 10^{-5} > 5 \cdot 10^{-6}$, the number of significant digits is also 5. (Note that the correct value of π is 3.14159265....)

2. Since $|\alpha - \alpha^*| \leq 10^{-2}$ is equivalent to $\alpha - 10^{-2} \leq \alpha^* \leq \alpha + 10^{-2}$,

$$1.49 \leq \alpha^* \leq 1.51.$$

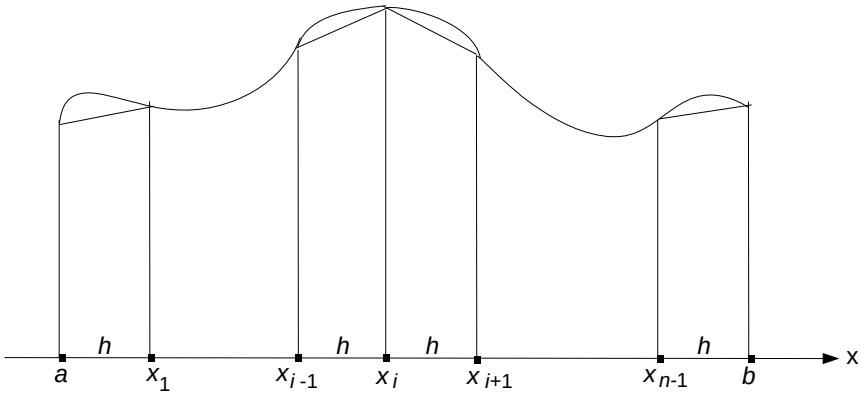
These bounds imply the following for the possible values of the relative error

$$0 \leq \frac{|\alpha - \alpha^*|}{|\alpha^*|} \leq \frac{10^{-2}}{1.49}.$$

3. $\sum_{i=0}^{5} \frac{0.5^i}{i!} = 1.648697916\bar{} \approx 1.6487$. The absolute error of approximating $\sqrt{e} = 1.648721....$ by this sum is 0.00002..., which is smaller than the required 10^{-4} (see the text immediately following formula (11.8)).

4. The composite trapezoidal rule is based on dividing the interval $a \leq x \leq b$ into n equal subintervals of length $h = (b-a)/n$ by the points $x_i = a + ih$ for $i = 0, 1, ..., n$ and approximating the area between the graph of $f(x)$ and the x axis (equal to $\int_a^b f(x)dx$) by the sum of the areas of the trapezoid

strips:



The area of the ith such trapezoid (with the lengths of two parallel sides $f(x_i)$ and $f(x_{i+1})$ and height h) is obtained by the standard formula

$$A_i = \frac{h}{2}[f(x_i) + f(x_{i+1})].$$

The entire area A is approximated by the sum of these areas:

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{i=0}^{n-1} A_i \approx \sum_{i=0}^{n-1} \frac{h}{2}[f(x_i) + f(x_{i+1})] \\ &= \frac{h}{2}[f(x_0) + f(x_1) + f(x_1) + f(x_2) + ... + f(x_{n-1}) + f(x_n)] \\ &= \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)].\end{aligned}$$

5. a. Applying formula (11.7) to function $f(x) = x^2$ on the interval $0 \le x \le 1$ divided into four equal subintervals of length $h = (1-0)/4 = 0.25$ yields

$$\int_0^1 x^2 dx \approx \frac{0.25}{2}[0^2 + 2(0.25^2 + 0.5^2 + 0.75^2) + 1^2] = 0.34375.$$

The exact value is

$$\int_0^1 x^2 dx = \frac{1}{3} = 0.\bar{3}.$$

Hence, the (magnitude of) truncation error of this approximation is $0.34375 - 0.\bar{3} = 0.010416\bar{6}$.
Formula (11.9) yields the following upper bound for the magnitude of the truncation error:

$$\frac{(b-a)h^2}{12} M_2 = \frac{(1-0)0.25^2}{12} 2 = 0.010416\bar{6},$$

which is exactly equal to the actual magnitude of the truncation error.

b. Applying formula (11.7) to function $f(x) = 1/x$ on the interval $1 \le x \le 3$ divided into four equal subintervals of length $h = (3-1)/4 = 0.5$ yields

$$\int_1^3 \frac{1}{x}dx \approx \frac{0.5}{2}[\frac{1}{1} + 2(\frac{1}{1.5} + \frac{1}{2} + \frac{1}{2.5}) + \frac{1}{3}] = 1.11\bar{6}.$$

The exact value is

$$\int_1^3 \frac{1}{x}dx = \ln 3 - \ln 1 = 1.09861....$$

Hence, the (magnitude of) truncation error of this approximation is $1.11\bar{6} - 1.09861... = 0.01805...$

Before we use formula (11.9), let us find

$$M_2 = \max_{1 \le x \le 3} |f''(x)| \text{ where } f(x) = \frac{1}{x}.$$

Since $f(x) = 1/x$, $f'(x) = -1/x^2$ and $f''(x) = 2/x^3$, $M_2 = 2$. Formula (11.9) yields the following upper bound for the magnitude of the truncation error:

$$\frac{(b-a)h^2}{12}M_2 = \frac{(3-1)0.5^2}{12}2 = \frac{1}{12} = 0.08\bar{3},$$

which is (appropriately) larger than the actual truncation error of 0.01805....

6. Since $f(x) = e^{\sin x}$, $f'(x) = e^{\sin x}\cos x$ and

$$\begin{aligned} f''(x) &= [e^{\sin x}\cos x]' = [e^{\sin x}]'\cos x + e^{\sin x}[\cos x]' = e^{\sin x}\cos^2 x - e^{\sin x}\sin x \\ &= e^{\sin x}(\cos^2 x - \sin x) = e^{\sin x}(1 - \sin^2 x - \sin x). \end{aligned}$$

Since $\sin x$ is increasing from 0 to $\sin 1$ and $1 - \sin^2 x - \sin x$ is decreasing from 1 to $1 - \sin^2 1 - \sin 1 \approx -0.55$ on the interval $0 \le x \le 1$,

$$|f''(x)| = |e^{\sin x}(1 - \sin^2 x - \sin x)| = e^{\sin x}|1 - \sin^2 x - \sin x| < e^{\sin 1} < e^1 < 3.$$

Using formula (11.9), we get

$$\frac{b-a}{12}\left(\frac{b-a}{n}\right)^2 M_2 < \frac{1}{12}\frac{1}{n^2}3 < \varepsilon,$$

where ε is a given upper error limit. Solving the last inequality for n yields

$$n^2 > \frac{1}{4\varepsilon} \quad \text{or} \quad n > \frac{1}{2\sqrt{\varepsilon}}.$$

Hence, the answers for $\varepsilon = 10^{-4}$ and $\varepsilon = 10^{-6}$ are $n > 50$ and $n > 500$, respectively.

7. The solution to the first system is $x = 1$, $y = 1$; the solution to the second one is $x = 8.5$, $y = -2$. Both systems are ill-conditioned.

8. In addition to the points made in Section 11.4 about solving quadratic equations, the program should handle correctly the case of $a = 0$ as well: If $b \neq 0$, the equation $bx = c$ has a single root $x = -b/c$. If $b = 0$ and $c \neq 0$, the equation $0x = c$ has no roots. If $b = 0$ and $c = 0$, any number is a root of $0x = 0$.

9. a. Let us prove by induction that $x_n > \sqrt{D}$ for $n = 0, 1, ...$, if $x_0 > \sqrt{D}$. Indeed, $x_0 > \sqrt{D}$ by assumption. Further, assuming that $x_n > \sqrt{D}$, we can prove that this implies that $x_{n+1} > \sqrt{D}$ as follows:

$$x_{n+1} - \sqrt{D} = \frac{1}{2}(x_n + \frac{D}{x_n}) - \sqrt{D} = \frac{x_n^2 - 2x_n\sqrt{D} + D}{2x_n} = \frac{(x_n - \sqrt{D})^2}{2x_n} > 0.$$

Now, consider

$$x_{n+1} - x_n = \frac{1}{2}(x_n + \frac{D}{x_n}) - x_n = \frac{1}{2}(\frac{D}{x_n} - x_n) = \frac{D - x_n^2}{2x_n} < 0,$$

i.e., the sequence $\{x_n\}$ is strictly decreasing. Since it is also bound below (e.g., by $\sqrt{D}$), it must have a limit (to be denoted by l) according to a well-known theorem of calculus. Taking the limit of both sides of

$$x_{n+1} = \frac{1}{2}(x_n + \frac{D}{x_n})$$

as n goes to infinity yields

$$l = \frac{1}{2}(l + \frac{D}{l}),$$

whose nonnegative solution is $l = \sqrt{D}$.

(Note that $x_0 = (1+D)/2 > \sqrt{D}$ for any $D \neq 1$ because $(1+D)/2 - \sqrt{D} = (1-\sqrt{D})^2/2 > 0$. If $D = 1$, $x_0 = (1+D)/2 = 1$ and all the other elements of the approximation sequence are also equal to 1.)

b. We will take advantage of the equality

$$x_{n+1} - \sqrt{D} = \frac{(x_n - \sqrt{D})^2}{2x_n},$$

which was derived in the solution to part (a) above. Since $x_0 = (1 + D)/2 > \sqrt{D}$ for any $D \neq 1$, according to the fact proved by induction in part (a) $x_n > \sqrt{D} \geq 0.5$. Therefore,

$$x_{n+1} - \sqrt{D} = \frac{(x_n - \sqrt{D})^2}{2x_n} \leq (x_n - \sqrt{D})^2.$$

Applying the last inequality n times yields

$$|x_n - \sqrt{D}| \leq (x_{n-1} - \sqrt{D})^2 \leq (x_{n-2} - \sqrt{D})^4 \leq .. \leq (x_0 - \sqrt{D})^{2^n}.$$

For $0.25 \leq D < 1$, $0.5 \leq \sqrt{D} < 1$, and hence

$$x_0 - \sqrt{D} = \frac{1+D}{2} - \sqrt{D} = \frac{(1 - 2\sqrt{D} + D)}{2} = \frac{(1-\sqrt{D})^2}{2} \leq 2^{-3}.$$

By combining the last two inequalities, we obtain

$$|x_n - \sqrt{D}| \leq (x_0 - \sqrt{D})^{2^n} \leq 2^{-3 \cdot 2^n}.$$

In particular, for $n = 4$,

$$|x_4 - \sqrt{D}| \leq 2^{-48} < 4 \cdot 10^{-15}.$$

10. We will roundoff the numbers to 6 decimal places.

$$\begin{aligned} x_0 &= \frac{1}{2}(1+3) = 2.000000 \\ x_1 &= \frac{1}{2}(x_0 + \frac{3}{x_0}) = 1.750000 \\ x_2 &= \frac{1}{2}(x_1 + \frac{3}{x_1}) \doteq 1.732143 \\ x_3 &= \frac{1}{2}(x_2 + \frac{3}{x_2}) \doteq 1.732051 \\ x_4 &= \frac{1}{2}(x_3 + \frac{3}{x_3}) \doteq 1.732051 \end{aligned}$$

Thus, $x_4 \doteq 1.732051$ and, had we continued, all other approximations would've been the same. The exact value of $\sqrt{3}$ is 1.73205080.... Hence, we can bound the absolute error by

$$1.732051 - 1.73205080... < 2 \cdot 10^{-7}$$

and the relative error by

$$\frac{1.732051 - 1.73205080...}{1.73205080...} < \frac{2 \cdot 10^{-7}}{1.73205080} < 1.2 \cdot 10^{-7}.$$