1. Exercise 5.5.

5. *Hidden surface removal* is a problem in computer graphics that scarcely needs an introduction: when Woody is standing in front of Buzz, you should be able to see Woody but not Buzz; when Buzz is standing in front of Woody, . . . well, you get the idea.

The magic of hidden surface removal is that you can often compute things faster than your intuition suggests. Here’s a clean geometric example to illustrate a basic speed-up that can be achieved. You are given \( n \) nonvertical lines in the plane, labeled \( L_1, \ldots, L_n \), with the \( i^{th} \) line specified by the equation \( y = a_i x + b_i \). We will make the assumption that no three of the lines all meet at a single point. We say line \( L_i \) is *uppermost* at a given \( x \)-coordinate \( x_0 \) if its \( y \)-coordinate at \( x_0 \) is greater than the \( y \)-coordinates of all the other lines at \( x_0 \): \( a_i x_0 + b_i > a_j x_0 + b_j \) for all \( j \neq i \). We say line \( L_i \) is *visible* if there is some \( x \)-coordinate at which it is uppermost—intuitively, some portion of it can be seen if you look down from \( y = \infty \).

Give an algorithm that takes \( n \) lines as input and in \( O(n \log n) \) time returns all of the ones that are visible. Figure 5.10 gives an example.

---

**Figure 5.10** An instance of hidden surface removal with five lines (labeled 1-5 in the figure). All the lines except for 2 are visible.
2. **Exercise 3-10.**

10. A number of art museums around the country have been featuring work by an artist named Mark Lombardi (1951–2000), consisting of a set of intricately rendered graphs. Building on a great deal of research, these graphs encode the relationships among people involved in major political scandals over the past several decades: the nodes correspond to participants, and each edge indicates some type of relationship between a pair of participants. And so, if you peer closely enough at the drawings, you can trace out ominous-looking paths from a high-ranking U.S. government official, to a former business partner, to a bank in Switzerland, to a shadowy arms dealer.

Such pictures form striking examples of social networks, which, as we discussed in Section 3.1, have nodes representing people and organizations, and edges representing relationships of various kinds. And the short paths that abound in these networks have attracted considerable attention recently, as people ponder what they mean. In the case of Mark Lombardi’s graphs, they hint at the short set of steps that can carry you from the reputable to the disreputable.

Of course, a single, spurious short path between nodes $v$ and $w$ in such a network may be more coincidental than anything else; a large number of short paths between $v$ and $w$ can be much more convincing. So in addition to the problem of computing a single shortest $v$-$w$ path in a graph $G$, social networks researchers have looked at the problem of determining the number of shortest $v$-$w$ paths.

This turns out to be a problem that can be solved efficiently. Suppose we are given an undirected graph $G = (V, E)$, and we identify two nodes $v$ and $w$ in $G$. Give an algorithm that computes the number of shortest $v$-$w$ paths in $G$. (The algorithm should not list all the paths; just the number suffices.) The running time of your algorithm should be $O(m + n)$ for a graph with $n$ nodes and $m$ edges.
3. **CLRS 4.5-1** Solve the following recurrences

- a) \( T(n) = 2T(n/4) + 1 \)
- b) \( T(n) = 2T(n/4) + \sqrt{n} \)
- c) \( T(n) = 2T(n/4) + n \)
- d) \( T(n) = 2T(n/4) + n^2 \)

4. **Prove by induction that**

\[ T(n) = 2T(n/8) + T(n/2) + n^2. \]

is solved as \( T(n) \in O(n^2) \).

5. **Matrix product**

Pan (1980) showed how to multiply two 70-by-70 matrices with only 143640 scalar multiplications. This resulted in a \( \Theta(n^{\log_{70}143640}) \) divide-and-conquer algorithm for multiplying n-by-n matrices. How few scalar multiplications would be necessary to multiply two 70-by-70 matrices in order to obtain a divide-and-conquer algorithm that is asymptotically faster than the method of [François Le Gall-2014]. Exhibit how you got this value.

6. **Tree properties**

Prove any two among \( A \land B \Rightarrow C \), \( B \land C \Rightarrow A \), \( C \land A \Rightarrow B \)

- A. \( G \) is connected,
- B. \( G \) does not contain a cycle,
- C. \( G \) has \( n-1 \) edges.
7. Exercise 3-11.

11. You’re helping some security analysts monitor a collection of networked computers, tracking the spread of an online virus. There are \( n \) computers in the system, labeled \( C_1, C_2, \ldots, C_n \), and as input you’re given a collection of trace data indicating the times at which pairs of computers communicated. Thus the data is a sequence of ordered triples \((C_i, C_j, t_k)\), such a triple indicates that \( C_i \) and \( C_j \) exchanged bits at time \( t_k \). There are \( m \) triples total.

We’ll assume that the triples are presented to you in sorted order of time. For purposes of simplicity, we’ll assume that each pair of computers communicates at most once during the interval you’re observing.

The security analysts you’re working with would like to be able to answer questions of the following form: If the virus was inserted into computer \( C_a \) at time \( x \), could it possibly have infected computer \( C_b \) by time \( y \)? The mechanics of infection are simple: if an infected computer \( C_i \) communicates with an uninfected computer \( C_j \) at time \( t_k \) (in other words, if one of the triples \((C_i, C_j, t_k)\) or \((C_j, C_i, t_k)\) appears in the trace data), then computer \( C_j \) becomes infected as well, starting at time \( t_k \). Infection can thus spread from one machine to another across a sequence of communications, provided that no step in this sequence involves a move backward in time. Thus, for example, if \( C_i \) is infected by time \( t_k \), and the trace data contains triples \((C_i, C_j, t_k)\) and \((C_j, C_q, t_r)\), where \( t_k \leq t_r \), then \( C_q \) will become infected via \( C_j \). (Note that it is okay for \( t_k \) to be equal to \( t_r \); this would mean that \( C_j \) had open connections to both \( C_i \) and \( C_q \) at the same time, and so a virus could move from \( C_i \) to \( C_q \).)

For example, suppose \( n = 4 \), the trace data consists of the triples

\[
(C_1, C_2, 4), \ (C_2, C_4, 8), \ (C_3, C_4, 8), \ (C_1, C_4, 12),
\]

and the virus was inserted into computer \( C_1 \) at time 2. Then \( C_3 \) would be infected at time 8 by a sequence of three steps: first \( C_2 \) becomes infected at time 4, then \( C_4 \) gets the virus from \( C_2 \) at time 8, and then \( C_3 \) gets the virus from \( C_4 \) at time 8. On the other hand, if the trace data were

\[
(C_2, C_3, 9), \ (C_1, C_4, 12), \ (C_1, C_2, 14),
\]

and again the virus was inserted into computer \( C_1 \) at time 2, then \( C_3 \) would not become infected during the period of observation: although \( C_2 \) becomes infected at time 14, we see that \( C_3 \) only communicates with \( C_2 \) before \( C_2 \) was infected. There is no sequence of communications moving forward in time by which the virus could get from \( C_1 \) to \( C_3 \) in this second example.

Design an algorithm that answers questions of this type: given a collection of trace data, the algorithm should decide whether a virus introduced at computer \( C_a \) at time \( x \) could have infected computer \( C_b \) by time \( y \). The algorithm should run in time \( O(m + n) \).