PART III

Recursion

Recursive thinking plays a fundamental role in combinatorics, theoretical computer science and programming. The next chapter introduces the recursive approach and the following two chapters discuss important applications.

Definition Recursive approach *A* **recursive approach** *to a problem consists of two parts:*

- *The problem is reduced to one or more problems of the same sort which are simpler in some sense.*
- *There is a collection of simplest problems to which all others are reduced after one or more steps. A solution to these simplest problems is given.*

As you might expect, a *recursive algorithm* is one that refers to itself. This seemingly simple notion is extremely important. Recursive algorithms and their translation into recursive procedures and recursive data structures are fundamental in computer science. For example, here's a recursive algorithm for sorting a list. (Sorting a list means putting the items in increasing order.)

- divide the list roughly in half,
- sort each half, and
- "merge" the two sorted halves.

Proof by induction and recursive algorithms are closely related. We'll begin Chapter 7 by examining inductive proofs and recursive equations. Then we'll look briefly at thinking and programming recursively.

Suppose that we have some items that have a "natural" order; e.g., the natural order for student records might be

- alphabetic by name (last name first),
- first by class and, within a class, alphabetic by name, or
- by grade point average with highest first.

We may allow ties. The problem of sorting is as follows: Given a list of items in no particular order, rearrange it so that it is in its natural order. In the event of a tie, the relative order of the tied items is arbitrary. In Chapter 8, we'll study some of the recursive aspects of software and hardware sorting algorithms. Many of these use the "divide and conquer" technique, which often appears in recursive algorithms. We end the chapter with a discussion of this important technique.

One of the most important conceptual tools in computer science is the idea of a rooted plane tree, which we introduced in Section 5.4. This leads naturally to methods for ranking and unranking various classes of unlabeled RP-trees. Many combinatorial algorithms involve "traversing" RP-trees. Grammars can often be thought of in terms of RP-trees, and generating machine code from a higher level language is related to the traversal of such trees. These topics are discussed in Chapter 9.