Teaching Suggestions

In this part of the *Instructor's Resource Guide*, I suggest how the text can be used effectively to teach an introductory course in discrete mathematics. These views are based on my personal teaching experience as well as on the experiences of some of the many instructors who have used the text in previous editions.

In the following material I provide an overview of each chapter of the text. Along with a description of the contents of that chapter, I describe its importance in an introductory course. After this overview, I give detailed information about each section of the chapter. First, I state the section goals and identify the prerequisites for that section. Then, I give my suggestions on how to teach from the section. In particular, I identify troublesome concepts and suggest how to handle them. I point out particularly useful examples and important concepts. Finally, I describe exercises that I feel are noteworthy, especially those that tie together diverse concepts or introduce new ideas. I hope this information makes teaching from the text easier and more rewarding for you.

CHAPTER 1
The Foundations: Logic and Proofs

Overview: Chapter 1 begins with an introduction to propositional and predicate logic. It then continues with presentations of the rules of inference for both propositional and predicate logic. After developing logic and rules of inference, the chapter introduces an arsenal of proof methods. The chapter concludes with a discussion of proof strategy, as well as the process of formulating conjectures and then using proof methods and strategies to settle them. The material on logic and proof in this chapter provides the foundations needed throughout higher mathematics and computer science. Without a firm foundation in logic, students have a great deal of difficulty with this course and subsequent courses. If you are lucky enough to have students with strong backgrounds, you might be able to cover quickly, or even skip, some of the contents of this chapter. But be sure to cover what your students need from this chapter, or the rest of the course could be tough sledding.

The first five sections deal with logic; propositional logic is covered in Sections 1.1–1.3, and predicate logic is covered in Sections 1.4–1.5. Studying logic is the best way to start a course in discrete mathematics (unless, of course, your students already know this material) because students must be able to think logically and carry out precise reasoning. Section 1.6 introduces rules of inference, and Section 1.7 introduces basic proof methods. Section 1.8 introduces additional methods of proof and addresses key aspects of strategies for developing proofs. Take note that the proof methods discussed in Sections 1.7 and 1.8 are used throughout the text, particularly in the coverage of sets and functions in Chapter 2, and in the coverage of algorithms and number theory in Chapters 3 and 4. Chapter 5 introduces another key proof method, mathematical induction, together with its variants.

SECTION 1.1 Propositional Logic

Goals: To introduce the basic terminology of propositional logic, including logical connectives, to show how to construct truth tables, to illustrate the importance of logic with applications, and to motivate the study of logic through logic puzzles and system specifications.

Prerequisites: None.

Advice: The material on logical connectives is straightforward. Most difficulties with this material involve confusion between common English usage and precise mathematical definitions. In particular, students have trouble with inclusive versus exclusive *or*; make sure the distinction is clear.

Stress the definition of a conditional statement, especially when the premise is false. That is, emphasize that $p \to q$ is false only when p is true and q is false. This material is used extensively in Section 1.6 when rules of inference are covered, in Section 1.7 when methods of proving conditional statements are discussed, and in Sections 5.1–5.3 when mathematical induction and its variants are covered. Go over the different ways conditional statements are expressed; these are listed after Definition 5. Define the converse, contrapositive, and inverse of a conditional statement (see Example 9); these terms are often confused with each other. Be sure to discuss the notion of equivalence of compound propositions. Explain that a conditional statement and its contrapositive are logically equivalent, whereas a conditional statement and its converse or inverse are not logically equivalent. Introduce biconditionals and how they are expressed. Mention that biconditionals are often implicitly stated. Briefly introduce truth tables; they are used extensively in Section 1.3. Also, quickly mention the precedence of logical operators.

Exercises: Exercises 45–47 introduce fuzzy logic, which is used in expert systems and artificial intelligence. Exercises 48–50 cover some logical paradoxes.

SECTION 1.2 Applications of Propositional Logic

Goals: To introduce some important applications of propositional logic, including many important applications in computer science. Also, to work with logic puzzles, which provide an entertaining way to learn and enjoy propositional logic.

Prerequisites: Section 1.1.

Advice: Cover the material on translating English sentences into logical statements; students often need help with this important task. The subsection on system specifications is of particular appeal to students in computer science and engineering; it shows that logic is of immediate practical importance. Computer science students are usually familiar with logical operators from their use in programming, so make the connection between the material in this section and logical operators used in programming languages. (In fact almost everyone has been forced to understand logical operations from doing Boolean searches on the Web—see Example 6.) You may want to spend time covering the subsection on logical puzzles; many people find these puzzles fascinating—in particular, see Example 7, which introduces one of Raymond Smullyan's knights and knaves puzzles. A brief introduction to logic circuits is included here for instructors who want to make the connection between formal logic and logic circuits. (A thorough treatment of logic circuits is found in Chapter 12.)

Exercises: Exercises 7–12 are devoted to system specifications. Exercises 15–18 and 32–38 are logical puzzles that can challenge students. Exercises 19–23 are puzzles involving Smullyan's knights and knaves, and Exercises 24–31 are puzzles involving knights, knaves, and spies, also introduced by Smullyan.

SECTION 1.3 **Propositional Equivalences**

Goals: To show how propositional equivalences are established and to introduce the most important such equivalences.

Prerequisites: Section 1.1.

Advice: Introduce the notion of propositional equivalences by establishing De Morgan's laws—see Example 2. Table 6 presents basic propositional equivalences. We will see similar tables for set identities in Section 2.2 and for Boolean algebra in Section 12.1. Mention that these properties hold in a wide variety of settings that all fit into one abstract form. Many students have the tendency to just memorize the properties, so stress that it is more important to understand their meaning and why they are true. Explain the different ways that such propositional equivalences can be established, including by the use of truth tables, by showing that the propositions are true (or false) for precisely the same sets of values, and by using previously proved equivalences, including those in Tables 6–8. (Note: We discuss rules of inference formally in Section 1.6 and introduce proofs in Section 1.7. Some practice with straightforward proofs here will help motivate the in-depth coverage that will follow.) The concept of propositional satisfiability is introduced in this section. The section concludes with a discussion of how Sudoku puzzles can be modeled as satisfiability problems.

Exercises: The exercise set introduces some new topics, including duality, disjunctive normal form, and functional completeness. Students can learn about duality by doing Exercises 34–39. Exercises 40–42 develop the concept of disjunctive normal form. Exercises 43–45 introduce the concept of functional completeness, and Exercises 46–54 introduce the operators *NAND* ($|$) and *NOR* ($\downarrow$) and show that the sets $\{\,|\,\}$ and $\{\downarrow\}$ are both functionally complete. Exercises 60–63 deal with satisfiability and Exercises 64–66 involve Sudoku.

SECTION 1.4 Predicates and Quantifiers

Goals: To introduce predicate logic, especially existential and universal quantification. Moreover, to explain how to translate between English sentences (or mathematical statements) and logical expressions.

Prerequisites: Sections 1.1 and 1.3.

Advice: This section is important because students often have trouble proving statements that involve quantification, including the inductive step in mathematical induction. Make sure they have a clear idea what the truth values of existential and universal quantifications mean. Tell students that a quantification is not well-defined unless the domain is specified and that changing the domain can change the truth value of the quantification. Mention that a statement of the form $\forall x\, P(x)$ can be shown to be false with a counterexample. Explain how to negate existential and universal quantifications (see Table 2). We will need this in Sections 1.7 and 1.8 when we discuss how to prove theorems that involve quantification (in particular, with existence proofs and counterexamples). Discuss the different ways to express universal and existential quantifications in English.

Devote special attention to the subsection on translating English sentences into logical statements; this is a particularly difficult task for many students. Be sure to stress that there is more than one way to translate a particular English sentence into a logical statement; see Examples 23 and 24. Example 25 illustrates how to use quantifiers in system specifications. Examples 26 and 27, taken from Lewis Carroll, illustrate the subtleties of translating English sentences into correct statements involving predicate and propositional logic. The subsection on logic programming shows that the material in this section is important in computer programming and AI.

Exercises: Exercises 23–28 provide a wide variety of examples of how quantifiers are used when English sentences are translated into logical statements. Exercises 32–34 deal with negations involving quantified statements. Exercises 38–42 cover the use of quantifiers to express system specifications. Exercises 46–49 introduce some useful logical equivalences called null quantifications. Exercises 52–54 deal with the uniqueness quantifier. Exercises 55–58 deal with Prolog. Exercises 59–62 are questions based on work by Lewis Carroll.

SECTION 1.5 Nested Quantifiers

Goals: This section explains how to work with nested quantifiers and makes clear that the order of quantification matters. This section helps students gain maturity working with complicated logical expressions involving multiple quantifiers.

Prerequisites: Sections 1.1, 1.3, and 1.4.

Advice: Describe how nested quantifiers work (see Table 1); you may find the analogy to nested loops useful. Use Examples 1–5 to illustrate the meaning of statements involving nested quantifiers. Cover Example 4 to illustrate that the order of quantification is important when several different quantifications occur in the same statement. Cover the process of translating mathematical statements into logical expressions involving nested quantifiers—see Examples 6–8. In particular, students who have studied the definition of limit should see Example 8. Then discuss the translation between complicated statements in English and logical expressions involving nested quantifiers—see Examples 9–13. Cover Examples 14–16 to illustrate how to negate logical expression involving quantifiers. In particular, Example 16 shows how to use quantifiers and predicates to express that a limit does not exist.

Exercises: Exercises 14–16 involve translating English sentences into logical expressions involving nested quantifiers. Exercises 17–18 involve translating system specifications into logical expressions involving nested quantifiers. Exercises 19–23 involve translating mathematical statements into expressions involving nested quantifiers. Negations of statements involving nested quantifiers are the subject of Exercises 36–38. Prenex normal form is introduced in Exercises 50–51.

SECTION 1.6 Rules of Inference

Goals: To introduce the notion of a valid argument and rules of inference for propositional logic. To explain how to use rules of inference to build correct arguments in propositional calculus. Moreover, to introduce rules of inference for predicate logic and how to use these rules of inference to build correct arguments in predicate logic. To show how rules of inference for propositional calculus and predicate calculus can be combined. Finally, to learn how to distinguish between correct and incorrect arguments.

Prerequisites: Sections 1.1, 1.3, 1.4, and 1.5.

Advice: Explain what it means for an argument form to be valid in propositional logic. Be sure to tell students that if the hypotheses in a valid argument form are not true, the conclusion of the argument may not be true. Introduce the basic rules of inference for propositional calculus—see Table 1. You may want to cover Examples 6 and 7, which show how to use rules of inference to construct valid arguments in propositional logic. Describe the rules of inference for quantified statements—see Table 2. Explain how rules of inference for propositional logic and predicate logic can be combined.

Mention begging the question; students will think this is not likely to occur, but you can show them examples from their own arguments.

Exercises: Exercises 23 and 24 ask students to find an error in an incorrect argument in predicate calculus. Exercise 26 asks for a justification of a rule of inference in predicate calculus called the rule of universal transitivity. Exercises 27–29 ask students to use rules of inference in predicate calculus to construct valid arguments.

SECTION 1.7 Introduction to Proofs

Goals: To introduce the notion of proof and basic methods of proof, including direct proof, proof by contraposition, and proof by contradiction. Furthermore, to learn how to distinguish between correct and incorrect arguments, and to understand and construct basic types of proofs.

Prerequisites: Sections 1.1, 1.3–1.6.

Advice: Begin with definitions of important terms, including *theorem*, *proof*, *corollary*, *lemma*, and *conjecture*. A key goal is for students to understand what constitutes a valid proof; they need to be able to understand existing proofs and create their own. Let them know that axioms and previously proven results can be used and that arguments must follow correct rules of inference for propositions and for predicates. (You may want to review the axioms for the real numbers in Appendix 1.) Students need to understand the difference between a formal proof and an informal one that could be expanded into a formal proof if necessary.

Spend substantial time showing how to prove conditional statements using direct proofs and proofs by contraposition; this will pay off when you discuss mathematical induction and whenever you prove theorems that are universal quantifications of conditional statements. Introduce some aspects of proof strategy that tell you when to try a direct proof and when to use an indirect proof (proof by contraposition or contradiction). Illustrate this by covering Examples 7 and 8. Be sure to spend adequate time discussing proof by contradiction. Example 9 illustrates a proof by contradiction and foreshadows the pigeonhole principle discussed in depth in Section 6.2. Explain what it means to show that statements are equivalent. Example 13 illustrates how the equivalence of three statements can be established.

Exercises: Assign some of Exercises 1–4, 6, 7, 9, 10, 15, 17, and 18 to give students practice with direct proofs, proof by contraposition, and proof by contradiction. In these exercises the method of proof is specified. Also assign some of Exercises 5, 8, 13, 14, and 16 to give students practice determining which method of proof to use. You may want to assign Exercises 11 and 12, which ask students to either prove or disprove a statement. Exercises 22–24 require proof by contradiction; these are really just examples of the pigeonhole principle. Exercises 26–28 ask students to show that two statements are equivalent. Exercises 30–33 and 41–42 ask for proofs that three or four statements are equivalent.

SECTION 1.8 Proof Methods and Strategy

Goals: To learn important methods of proofs including proof by cases and existence proofs, supplementing the basic methods introduced in Section 1.7. To introduce key strategies for proving theorems, to understand the roles of conjectures and counterexamples, and to learn about some important open problems.

Prerequisites: Sections 1.1, 1.3, and 1.4–1.7.

Advice: Introduce proof by exhaustion and proof by cases. You may want to cover Example 4, which shows how a proof by cases is used to prove that the absolute value of the product of two numbers is the product of their absolute values. Mention the notion of *without loss of generality* and how it can be used to simplify proofs that might need to consider separate cases. Cover some of the common errors that arise in incorrect proofs by cases.

Introduce existence proofs, and discuss the difference between constructive and nonconstructive existence proofs. Examples 11 and 12 provide good examples of nonconstructive existence proofs. Explain what is needed in a uniqueness proof, and use Example 13 to illustrate how a uniqueness proof proceeds.

The material presented here provides students with a window into what mathematics is really about. Explain some of the strategies used to find proofs of theorems. Explain that the proof methods studied in Section 1.7 and the first part of this section provide the tool kit, but the art of finding proofs is something altogether different. You can illustrate this by covering Examples 14 and 15. Also, be sure to cover Example 16, which illustrates how leveraging an existing proof can provide a good starting point for constructing a new proof.

Use the material on tilings to discuss the role of conjectures and how to use the proof methods developed in the text to settle them. This material requires no extra machinery, so it is quite accessible. Formulating conjectures about tilings of checkerboards, and parts of checkerboards, is easy, but settling these conjectures ranges from straightforward to extremely tricky.

Devote some time to discussing the role of open problems. Students will find the story behind Fermat's Last Theorem compelling. Learning about easily understood conjectures that remain unsolved, such as the $3x + 1$ conjecture, also motivates many students.

Exercises: To give student practice with proof by cases, assign Exercises 3 and 4. Exercises 5 and 6 involve the notion of *without loss of generality*. Exercises 8–12 ask for existence proofs, together with an explanation why the proof is constructive or nonconstructive. Exercise 26 is an excellent example of working backwards. Exercise 33 provides an opportunity for students to adapt an existing proof. The famous three jug problem is the subject of Exercise 38, which asks students to prove or disprove that you can solve this problem. Exercises 41–50 asks students questions about tilings of checkerboards. Exercises 41–44 ask students to prove or disprove a statement about tilings; this provides practice analyzing whether a conjecture is true and which proof method to use to prove the conjecture or to show that it is false. Exercise 48 is a challenging exercise about tilings that has a particularly elegant solution.

CHAPTER 2
Basic Structures: Sets, Functions, Sequences, Sums, and Matrices

Overview: Chapter 2 presents an introduction to basic discrete structures, namely sets, functions, sequences, summations, and matrices. Some of this material may be review for your students, but there are some important topics covered not already familiar to your students. You should quickly cover, or not cover at all, material that your students already know. However, be sure to cover topics that your students may not already know, such as set identities, countability, the floor and ceiling functions, and summation formulae.

SECTION 2.1 Sets

Goals: To introduce the basic terminology of set theory.

Prerequisites: Chapter 1.

Advice: Make sure students understand that when specifying the elements of sets the number of times an element is listed and the order in which the elements are listed do not matter. These facts are illustrated by Example 6. Students have trouble distinguishing between the sets $\emptyset$ and

{∅}, so explain that the empty set is the set with no elements and that it is a subset of every set. I like to start with the empty set and take the power set and then the power set again to force students to see the difference between the empty set, the set containing the empty set, and other confusing sets. You may want to present the proof of Theorem 1, which shows that every nonempty set has at least two subsets, the set itself and the empty set, especially because this is an excellent illustration of proof methods covered in Section 1.7.

Exercises: Russell's paradox is described in Exercise 46. This is a difficult exercise for students, but it is important since it shows that a consistent set of axioms is needed for set theory. Exercise 45 shows how to define ordered pairs in terms of sets.

SECTION 2.2 Set Operations

Goals: To show how set identities are established and to introduce the most important such identities.

Prerequisites: Chapter 1 and Section 2.1.

Advice: The relationship between set identities and logical equivalences becomes clear when set operations are expressed using set builder notation and logical operators. Show students several different ways to prove a set identity, namely by showing that each side is a subset of the other, by a membership table, by the use of logical equivalences, or by using set identities that have already been established. Explain that the set identities in Table 1 are analogous to the propositional equivalences in Section 1.3 and to Boolean identities that will be given in Chapter 12. We touch briefly on how to count elements in the union of two sets, foreshadowing the treatment of inclusion–exclusion in Chapter 8.

Exercises: The notion of the symmetric difference of two sets is introduced in the exercise set and studied in Exercises 32–43. Fuzzy sets, used in expert systems and artificial intelligence, are the subject of Exercises 63–65. You can ask your creative students to make the connection between fuzzy logic, introduced in Section 1.1, and fuzzy sets. Multisets are dealt with in Exercises 61–62; multisets are sometimes used when combinations with repetition allowed are studied. The successor of a set is defined in the preamble to Exercise 59.

SECTION 2.3 Functions

Goals: To introduce the concept of a function, the notion of one-to-one functions, onto functions, and the floor and ceiling functions.

Prerequisites: Chapter 1 and Sections 2.1 and 2.2.

Advice: We define functions as assignments and their graphs as the sets of ordered pairs determined by these assignments. As such, the graph of a function is a type of relation, a topic we cover in Chapter 9. Although functions are discussed in a general setting, most of the examples deal with functions from one discrete set to another, as is appropriate for a course in discrete mathematics. Sometimes students have trouble with the definitions of one-to-one and onto functions. Use Figure 5 to help make these concepts clear. Show students how to express the definitions of one-to-one and onto in terms of quantifiers. Make sure your students have a clear understanding of the floor and ceiling functions; there is often confusion about their values at negative real numbers. Examples 27 and 28 show how these functions are applied in basic problems in data communications. Table 1 displays useful properties of the floor and ceiling functions. Make sure students are familiar with these properties. Proving results about the floor and ceiling functions provides more practice with methods of proof. The section concludes with the definition of the factorial function; be sure to cover this if your students are not already familiar with factorials. Finally, the notion of a partial function, important in the study of Turing machines is introduced.

Exercises: Exercises 46–57 give students the opportunity to work with the properties of the floor and ceiling functions, and Exercises 58–61 involve application of these functions to simple calculations in data communications. Exercise 72 asks students to show that the notions of one-to-one and onto are equivalent when the domain and codomain are finite sets of the same size. Exercise 79 establishes some important facts about the cardinality of finite sets and Exercise 80 establishes an important result about infinite sets.

SECTION 2.4 Sequences and Summations

Goals: To introduce terminology used for sequences and summations. To introduce recurrence relations and some methods for solving them. To work with summations and establish several important summation formulae.

Prerequisites: Chapter 1 and Section 2.3.

Advice: The first part of this section deals with sequences. Recurrence relations are introduced and the method of iteration for solving them is discussed. Example 11 illustrates how recurrence relations are used to solve a problem involving compound interest. The topic of integer sequences is covered, which requires more critical and creative thinking than the other material. Examples 12–15 involve conjecturing a formula or rule for generating the terms of a sequence when only the first few terms are known. Encourage students to try the *On-Line Encyclopedia of Integer Sequences*, mentioned in this section.

The second part of the section introduces summation notation. Make sure students can work with the different forms of this notation and with shifting indices in summations. In particular, this will be helpful later when we prove summation formulae using mathematical induction. Students should also understand that sequences and strings are just special types of functions.

Exercises: Exercises 9–10 ask students to conjecture the formula or rule for generating the terms of a sequence from the first few terms; these exercises are more challenging than Exercises 5–6, which ask students to list the terms of sequences defined in different ways. Exercises 7–8 are interesting since they point out that there are many different naturally arising sequences that have the same initial terms. Exercises 18–24 deal with solving problems using recurrence relations. Telescoping sums are defined in Exercise 35 and are used to find the sums of the first n positive integers and the squares of these integers in Exercises 37 and 38, respectively. Product notation is introduced in the exercise set. Assign Exercise 43 if you wish to cover this.

SECTION 2.5 Cardinality of Sets

Goals: To master the concept of the cardinality of sets. In particular, to understand the difference between countable sets and uncountable sets.

Prerequisites: Chapter 1 and Sections 2.1–2.3.

Advice: In general, the material in this section is more difficult than earlier material. Use the idea of Hilbert's Grand Hotel to explain the concept of countable sets. Students find this idea quite helpful when learning about countable sets. Showing that the set of positive rational numbers is countable is covered in Example 4. The proof that the set of real numbers is uncountable, using the Cantor diagonalization method, is elegant and quite subtle; it is given in Example 5. Motivate this by using a numerical example for the construction of a real number that was not listed. Mention the notion of computable and uncomputable functions and explain why uncomputable functions exists. Finally, better students may be fascinated by the continuum hypothesis, briefly described at the end of the section.

Exercises: Exercises 5–10 ask questions about finding rooms for newly arriving guests at Hilbert's Grand Hotel. Exercises 15 and 16 ask students to show that a set containing an uncountable set is also uncountable and a subset of a countable set is countable, respectively. Exercises 26 and 32 provide alternative methods of proving that the set of positive rational numbers is countable. Exercise 39 asks for a proof that there are functions that are not computable.

SECTION 2.6 Matrices

Goals: To introduce basic properties of matrices and matrix arithmetic, including Boolean operations on zero–one matrices.

Prerequisites: Chapter 1 and Sections 2.3 and 2.4.

Advice: This section presents a brief review of the material on matrices needed in later sections of the text. Students should understand how matrix multiplication is defined and know that it is not commutative. The material on zero–one matrices and Boolean operations on them will be new to most students. This material is used only in Section 9.4, when the transitive closure of relations is discussed, and it may be omitted if you do not intend to cover that section.

Exercises: Make sure students know what a diagonal matrix is (see Exercise 14). You may want to assign Exercises 18–21, which deal with the notion of the inverse of a matrix.

CHAPTER 3 Algorithms

Overview: Sections 3.1 introduces the concept of an algorithm. The purpose of this material is to ensure that students understand what an algorithm is and the different ways algorithms are expressed. The section illustrates the concept of an algorithm by covering searching and sorting algorithms. The notion of a greedy algorithm is also introduced. Section 3.2 introduces asymptotic notations used to describe the growth of functions, including big-O, big-Omega, and big-Theta notations. Section 3.3 describes how to express the complexity of an algorithm. This is important since later chapters discuss a variety of algorithms and their complexity. The notion of an algorithm paradigm is also discussed in Section 3.3.

SECTION 3.1 Algorithms

Goals: To introduce the concept and basic properties of an algorithm.

Prerequisites: Chapters 1 and 2.

Advice: The algorithm for finding the largest element in a finite sequence of integers provides a good example of an algorithm since it is simple and it solves a useful problem. Students should understand the steps used in actually solving a problem. First we find an algorithm, which is expressed initially in English and then in pseudocode. Next, we study the complexity of the algorithm. Then we construct a computer program to implement it. Finally we verify the correctness of the program. We concentrate on the mathematical portions of the study of algorithms in the text, namely, how to solve problems using algorithms (in this section), how to study their complexity (in Section 3.3), and how to prove them correct (in Sections 5.4 and 5.5).

Introduce the problems of searching and sorting and present the linear and binary searches and one or both of the bubble sort and the insertion sort. (We will study the complexity of these algorithms later on.) This may be a good time to introduce the notion of a greedy algorithm. The change-making algorithm presented here provides an easy introduction to this topic and the question of whether a particular greedy algorithm produces optimal solutions. Example 7 introduces the problem of the most possible talks that can be scheduled in a lecture hall given their start and end times. This is a good example for examining different criteria to be used at each step.

The subsection on the famous halting problem, which computer science majors should see again in a Theory of Computation course, is optional. It is a beautiful example of proof by contradiction, but the argument is subtle (and gives students difficulties), partly because of its self-referential nature.

Exercises: A variant of the binary search algorithm is introduced in Exercise 26. This version of the algorithm stops if the middle term at any stage equals the desired integer. The ternary search algorithm is introduced in Exercise 27; this exercise gives students the opportunity to develop a search algorithm on their own, generalizing the binary search algorithm. The selection sort is introduced and studied in Exercises 41–42 and the binary insertion sort in Exercises 47–49. Exercises 56 and 58 asks for counterexamples that show that a particular criteria for each step of a greedy algorithm does not always lead to an optimal solution. The notion of a stable assignment is introduced in the preamble to Exercise 60, and the deferred acceptance algorithm is introduced in the preamble of Exercise 61.

SECTION 3.2 The Growth of Functions

Goals: To introduce big-O and related big-Omega and big-Theta notation, and to show how to estimate the size of functions using this notation.

Prerequisites: Chapters 1 and 2.

Advice: Students have trouble with big-O notation. Often they cannot decide how to choose the witnesses C and k in the definition. Show them how different pairs of constants can be used as witnesses. Give several different examples to illustrate the concept. Show how the definition of this notation involves the use of existential and universal quantifiers. Cover Examples 5 and 6, which give estimates for the sum of the n smallest positive integers and for $n!$, respectively. Go over the useful big-O estimates for logarithms, powers, and exponential functions; these provide useful guides for comparing the growth of common functions. Cover big-Omega and big-Theta notation and discuss the connections between them and big-O notation. Indicate the importance of big-O in estimating the complexity of algorithms. We will study this formally in Section 3.3.

Exercises: Exercises 21 and 22 ask that a list of functions be ordered so that each is big-O of the next function on the list. Another type of asymptotic notation is introduced in the exercise set—little-o notation, which depends on the notion of a limit. If your students have a satisfactory background working with limits you may want to assign some of Exercises 61–69, which deal with this concept. We will use the result in Exercise 72 when we use trees to study the complexity of sorting algorithms in Section 11.2.

SECTION 3.3 Complexity of Algorithms

Goals: To introduce computational complexity analysis.

Prerequisites: Chapters 1 and 2 and Sections 3.1–3.2.

Advice: This section deals with complexity of algorithms. This is an important mathematical part of computer science. We define different types of complexity but concentrate on time complexity. Explain the distinction between worst-case and average-case complexity. Tell students the merits, as well as the drawbacks, of using big-O estimates. Explain how the witnesses C and k in a big-O estimate have practical implications. Because average case complexity depends on notions of probability, a topic not formally studied until Chapter 7, tell students in an informal way how average-case analysis depends on the distribution of input values. The complexity of matrix multiplication is studied and the problem of determining the best order for matrix-chain multiplication is introduced—we return to this problem in the exercises of Section 9.1. The notion of algorithmic paradigms is introduced in this section and brute-force algorithms are discussed. (The algorithmic paradigm of greed was introduced in Section 3.1; among the other algorithmic paradigms covered in the book are divide-and-conquer and dynamic programming in Chapter 8 and backtracking in Chapter 11.) I suggest giving students an informal introduction to tractable, intractable, solvable, unsolvable, NP, and NP-complete problems. A more formal treatment of these topics can found at the conclusion of the last section of the last chapter of the text.

Exercises: Exercises 1–4 ask students to give big-O estimates when various segments of algorithms, expressed as blocks of pseudocode, are carried out. Exercise 12 develops a big-Theta estimates for the number of steps used by an algorithm; the big-Omega part of this is harder than the big-O part. You may want to assign Exercise 14, which deals with Horner's method for evaluating polynomials. Have students compare the complexity of this algorithm with the conventional method described in Exercise 13.

CHAPTER 4
Number Theory and Cryptography

Overview: Section 4.1 introduces some basic notions of number theory, including divisibility of integers and congruences. Section 4.2 introduces base b representations of integers (including binary, octal, and hexadecimal) and presents algorithms for integer arithmetic. Primes are discussed in Section 4.3, including conjectures about primes. Section 4.3 also introduces greatest common divisors and the Euclidean algorithm. The fundamental theorem of arithmetic is also introduced in Section 4.3. In Section 4.4 we see how to solve linear congruences. We also see how to solve systems of linear congruences using the Chinese remainder theorem. Section 4.5 presents several important applications of congruences, namely pseudorandom number generation, hashing, and check digits. Finally, Section 4.6 provides an introduction to the basic ideas of cryptography. In this section, both classical and modern cipher systems are studied. Public-key cryptography and two important cryptographic protocols—key exchange and signed messages—are studied.

SECTION 4.1 Divisibility and Modular Arithmetic

Goals: To introduce some fundamental concepts from number theory, including the division algorithm, congruences, and the rules of modular arithmetic.

Prerequisites: Chapters 1 and 2.

Advice: Be sure you mention that what is called the division algorithm is not really an algorithm, because this is quite confusing. (We will present an algorithm that finds the quotient and remainder in Section 4.2.)

Explain the difference between congruence notation and the **mod** function. Cover the basic properties of congruences; we will need this material in Chapter 9 when we discuss congruence modulo m as an equivalence relation. Be sure to mention that working with congruences is similar to working with equalities, but that division of both sides of a congruence by the same integer may not produce a valid congruence. If you plan on covering recursive algorithms in Section 5.4, be sure to cover Corollary 2, which is used to develop an efficient recursive algorithm for modular exponentiation.

You may want to cover the notion of arithmetic on $\mathbf{Z}_m$. This material will be useful to students who study abstract algebra in the future.

Exercises: Exercises 15–16 establish the relationship between the congruence notation and the **mod** function. Exercises 34–36 ask that certain results pertaining to congruences be established. These exercises give students some practice working with the notion of a congruence. Exercises 42–44 asks for proofs of properties of addition and multiplication in $\mathbf{Z}_m$.

SECTION 4.2 Integer Representations and Algorithms

Goals: To study representations of integers in different bases, including binary, octal, and hexadecimal representations, and to introduce algorithms involving integers based on these representations.

Prerequisites: Chapters 1–3 and Section 4.1.

Advice: If your students do not have practice using different bases for representing integers, spend some time on the discussion of such representations in this section. Show students how to convert from one base to another (see Algorithm 1). The algorithms for addition, subtraction, and multiplication of integers were the first procedures to be called algorithms. Students need to study this type of algorithm in order to understand how computers perform arithmetic. (Note: We will introduce a more efficient algorithm for multiplication in Section 8.3.) Performing modular exponentiation is important in cryptography; it is presented as Algorithm 5.

Exercises: The exercise set introduces other ways to represent integers, including those important in computer arithmetic. In particular, balanced ternary expansions are described in Exercise 30, one's complement representations are defined in the preamble to Exercise 34, two's complement representations are defined in the preamble to Exercise 40, binary coded decimal expansions are discussed in Exercise 47, and Cantor expansions are introduced in the preamble to Exercise 48. The simple conversions between binary, octal, and hexadecimal notations are the subject of Exercises 3–19. The complexity of modular exponentiation is the subject of Exercise 58.

SECTION 4.3 Primes and Greatest Common Divisors

Goals: To introduce some fundamental concepts from number theory, including primality, prime factorization, and greatest common divisors. To introduce some important conjectures about primes.

Prerequisites: Chapters 1–2 and Sections 3.1 and 4.1.

Advice: Students often do not see that when factoring an integer it is necessary to do trial divisions only by integers less than or equal to the square root of the integer being factored. This is emphasized in Examples 3 and 4. Show how to use the sieve of Eratosthenes to find all primes less than a positive integer Be sure to prove that there are infinitely many primes (Theorem 3); this is one of most elegant and famous proofs in mathematics. Briefly address the subject of primes in arithmetic progressions, addressing Dirichlet's theorem and the recent result of Green and Tao about arithmetic progressions of prime numbers. This latter result illustrates that new results are still being discovered. Discussing the search for new Mersenne primes (which can be monitored on the Web) also illustrates that number theory is an active field. Briefly discuss conjectures about primes and some of the famous open questions about them, such as Goldbach's conjecture. New discoveries about prime numbers often find their way into the popular press and are the focus of many websites.

Introduce the greatest common divisor and least common multiple of two integers. Make it clear that using prime factorizations to find greatest common divisors is easy once these factorizations are known, but that factoring integers is extremely time consuming. Next, introduce the Euclidean algorithm. Besides being one of the oldest algorithms invented, it is an excellent illustration of the concept of an algorithm. We will study the complexity of the Euclidean algorithm in Section 5.3. (We defer the complexity analysis to that section because we will need an estimate for the size of Fibonacci numbers, which we will establish in that section.)

After showing that the greatest common divisor of two positive integers can be expressed as a linear combination of these integers, you can show that the prime factorization of an integer is unique (up to the order of the factors); the fact that every positive integer has a prime factorization is proved in Section 5.2.

Exercises: Exercise 11 asks for a proof that $\log_2 3$ is irrational; it is a simple, but challenging, exercise that follows from the fundamental theorem of arithmetic. The Euler phi-function is introduced in the preamble to Exercise 21. The extended Euclidean algorithm is introduced in the preamble to Exercise 41. Exercises 54 and 55 asks students to adapt the proof in the text that there are infinitely many primes to prove that there are infinitely many primes of the forms $3k+2$ and $4k+3$, respectively. Exercises 56 and 57 challenge students to present two different ways to show that the set of positive rational numbers is countable using material from this section.

SECTION 4.4 Solving Congruences

Goals: To learn how to solve linear congruences and simultaneous systems of linear congruences. To introduce Fermat's little theorem, pseudoprimes, primitive roots, and discrete logarithms.

Prerequisites: Chapters 1 and 2 and Sections 4.1 and 4.3.

Advice: Introduce the concept of a linear congruence. Explain what an inverse modulo m is and how to use inverses to solve linear congruences. Describe how to use the Euclidean algorithm to find modular inverses. Then introduce systems of linear congruences using Sun-Tsu's puzzle as motivation. Introduce the Chinese remainder theorem and explain the proof of the part of the theorem that asserts existence of a simultaneous solution. You may want to also show how to solve systems of linear equations by back substitution, besides using the construction in the proof of the theorem to find solutions. You may also want to illustrate how arithmetic with large integers can be carried out using the Chinese remainder theorem.

Present Fermat's little theorem and illustrate its use in computations. Introduce the notion of a pseudoprime and discuss the importance of pseudoprimes for finding large primes. You may want to discuss Carmichael numbers too. Introduce the notions of primitive roots and primitive roots. Discuss discrete logarithms, especially if you plan to cover cryptographic protocols in Section 4.6.

Exercises: Exercise 19 outlines a proof of Fermat's little theorem. Exercise 30 establishes the uniqueness part of the Chinese remainder theorem. Exercise 37 uses Fermat's little theorem to show that 341 is a pseudoprime to the base 2. The concept of a strong pseudoprime is introduced in the preamble to Exercise 44. Quadratic residues are introduced in the preamble to Exercise 58 and addressed in Exercises 58–64.

SECTION 4.5 Applications of Congruences

Goals: To introduce three important applications of congruences, which show the usefulness of number theory and also are important in their own right.

Prerequisites: Chapters 1 and 2 and Sections 4.1, 4.3, and 4.4.

Advice: Explain what a hashing function is and explain how to $h(k) = k \textbf{ mod } m$ for hashing. Describe what a collision is and describe how to use a linear probing function to resolve collisions.

Explain what pseudorandom numbers are and the difference between them and truly random numbers. Introduce the linear congruential method for generating pseudorandom numbers and the pure multiplicative generator. Explain the issues involved with the use of these pseudorandom number generators.

Explain what a check digit is for a string of digits. Introduce parity check bits. Continue by discussing check digits for universal product codes and ISBNs. Discuss the types of errors that can arise in string of digits and explain how the check digit for ISBN-10s can detect single errors and transposition errors.

Exercises: Double hashing, a method for resolving collisions, is introduced in the preamble to Exercise 4. The middle-square method for generating pseudorandom numbers is introduced in the preamble to Exercise 9 and power generator is introduced in the preamble to Exercise 11. How the check digit for United States Postal Service money orders is found is explained in the preamble to Exercise 18; Exercises 18–23 involve the use of check digits for these money orders. The check digit for ISSNs is introduced in the preamble to Exercise 32 and is the subject of Exercises 32–35.

SECTION 4.6 Cryptography

Goals: To introduce the basic notions of cryptography and cryptographic protocols. To explain both classical and modern encryption methods, including the RSA public-key cryptography. To introduce two important cryptographic protocols, key exchange and digital signatures.

Prerequisites: Chapters 1 and 2 and Sections 4.1–4.4.

Advice: Introduce the Caesar cipher, shift ciphers and affine ciphers and explain the terminology of cryptography as you introduce these ciphers. Introduce the notion of cryptanalysis and explain how frequencies of letters can help with the cryptanalysis of message enciphered using shift and affine ciphers. Explain the difference between monoalphabetic ciphers and block ciphers and introduce transposition ciphers as an example of a block cipher. Introduce the notion of a cryptosystem and how the shift cipher serves as an example of a cryptosystem.

Introduce the notion of public key cryptography and how it differs from private key cryptography. Explain how the RSA cryptosystem works by explaining how to find keys, encrypt, and decrypt using this cryptosystem. (You may want to explain that the RSA cryptosystem was first invented in secret work in the U.K. by Clifford Cocks.) Explain why RSA works as a public key cryptosystem.

Introduce the notion of a cryptographic protocol. Explain the reason why exchanging secret keys is important and discuss how the Diffie-Hellman key agreement protocol works. Describe why digital signatures are important and explain how to construct digital signatures using the RSA cryptosystem.

Exercises: The Vigenère cipher is introduced in the preamble to Exercise 18 and is the subject of Exercises 18–22. Exercise 33 describes a basic protocol for key exchange using private key cryptography using a trusted third party.

CHAPTER 5
Induction and Recursion

Overview: This chapter is devoted to two interrelated core concepts, induction and recursion. In Section 5.1 we show how to prove a variety of theorems using mathematical induction. In Section 5.2 we study strong induction (sometimes called the second principle of mathematical induction), and we show how to use the well-ordering property of the set of positive integers in proofs. Section 5.3 discusses recursive definitions of functions, sequences, and sets, as well as introducing the notion of structural induction, which is used to prove results about recursively defined sets. In Section 5.4 we deal with recursive algorithms and illustrate how to prove that they are correct. Finally, Section 5.5 ties together some important concepts, namely computer programs and proofs, with a discussion of program correctness.

SECTION 5.1 Mathematical Induction

Goals: To explain how to construct proofs of a variety of theorems using mathematical induction.

Prerequisites: Chapters 1 and 2 and Sections 4.1–4.3.

Advice: Mathematical induction may be the most important topic in the text. Most students easily grasp the notion of climbing an infinite ladder as a way to understand mathematical induction. You may also want to mention the idea of knocking down an infinite sequence of dominoes as another ways to think about mathematical induction. Carefully explain the steps that make up a proof by mathematical induction. Explain why the basis step can begin at any integer. It helps students when you structure proofs by mathematical induction. First, write out the proposition $P(n)$ that is to be proved. Next, establish the basis step $P(n_0)$, making sure to clearly specify the integer n_0 where the induction begins. Then, complete the inductive step, beginning by explicitly stating the goal, that is, to prove that the universal quantification $\forall k(P(k) \to P(k+1))$ is true. (It helps to write down both $P(k)$ and $P(k+1)$ before writing this quantification out.) Sometimes students think proofs by induction are circular reasoning. Make sure they know why this is not the case. Indicate that the basis step often turns out to be a vacuous or trivial proof. You may want to present the template for proofs by mathematical induction given at the end of the section.

Tell students that mathematical induction cannot be used to find formulae or to formulate theorems, but can be used to prove that a formula or theorem is correct. This can be emphasized by following the introductory remarks in the text and in Example 2. This example first shows how to conjecture a formula for the sum of the first n odd positive integers from evidence provided by the smallest cases. Then prove that the guess is correct using mathematical induction.

Students should see inductive proofs of many different types of theorems. Examples 1–4 show how mathematical induction can be used to prove summation formulae. Show them how to use mathematical induction to prove inequalities; see Example 5–7. Surprisingly, students often have trouble taking the inequality in the inductive hypothesis and establishing the corresponding inequality for the next larger integer. Illustrate how to prove a result about divisibility (see Examples 8 and 9), how to prove results about sets (Example 10 uses mathematical induction to show that a set with n elements has 2^n subsets), and how to prove a set identity (see Example 11). Example 12 illustrates how mathematical induction can be used to prove an interesting result about an algorithm, namely that the greedy algorithm for scheduling talks given in Section 3.1 always produces an optimal schedule. Example 13 is an example of how mathematical induction can be used to prove a surprising result in an unusual setting. Example 14 is a particularly appealing use of mathematical induction; it shows how to use mathematical induction to prove that certain checkerboards can be tiled using right triominoes.

Explain why mathematical induction is a valid proof technique by going back to the axioms for the set of positive integers (see Appendix 1). You will probably want to cover the subsection on errors in purported proofs by mathematical induction. I strongly suggest you cover Example 15, which presents one such faulty proof.

Exercises: Give students practice proving a variety of results using mathematical induction. Exercises 3–17 ask for proofs of summation formulae; Exercises 18–29 ask for proofs of inequalities; Exercises 31–37 ask for proofs of divisibility results; Exercises 38–46 ask for proofs of results about sets, with Exercises 38–44 asking for proofs of set identities and Exercises 45–46 asking for proofs that sets have a certain number of subsets of a specified size. Exercises 49–51 ask students to find the error in an incorrect proof by mathematical induction. I suggest assigning one or more of Exercises 52–55, 60–65, 68, and 72–73, which show how mathematical induction can be used to prove a wide variety of results. Exercises 69–71 introduce the *gossip problem*; mathematical induction can be used to establish a key result about this problem (see Exercise 70). You might want to assign Exercise 74, which introduces the important idea of inductive loading. Exercise 83 is the justification for allowing an inductive proof to start at any integer; you may want to assign this to give students practice showing that different forms of mathematical induction are equivalent.

SECTION 5.2 Strong Induction and Well-Ordering

Goals: To explain how to construct proofs of a variety of theorems using strong induction and the well-ordering property.

Prerequisites: Chapters 1 and 2 and Sections 4.1, 4.3, and 5.1.

Advice: Show students how to prove theorems using strong induction. Cover the paradigm example of strong induction, which proves that every positive integer is the product of primes; this is Example 2. (Because students find the basis step difficult to understand if the basis step is the case for the integer 1, the example uses as its basis step the case for the integer 2. You might want to explain why the basis step can be taken to be the case for the integer 1, where the product of primes is the empty product.) Example 3 is a good illustration of how strong induction can be used to prove a result about a version of the game of nim. Example 4 shows how a result can be proved in two ways, one using the principle of mathematical induction and the other using strong induction. I recommend you introduce some notions of computational geometry and show how strong induction is used in the proof of Theorem 1, which shows that every simple polygon can be triangulated. (You may want to discuss the surprisingly difficult to prove lemma needed in the proof of Theorem 1, namely that every simple polygon has an interior diagonal.)

Present the well-ordering property and show how it can used directly to prove results such as that in Example 6.

Exercises: Assign a range of exercises that use strong induction to prove a variety of results. Exercises 1 and 2 are simple applications of strong induction; Exercises 3–8 ask for strong induction proofs that show that certain denominations can be used to produce all sufficiently large denominations; Exercise 9 asks for a strong induction proof that the square root of 2 is irrational; Exercise 10, about breaking a chocolate bar into pieces, is a particularly appealing application of strong induction. Exercises 14–16 ask for strong induction proofs for results about winning moves in games; Exercises 17–20 ask for strong induction proofs of results in computational geometry. Exercises 22–23 show how inductive loading can be used with strong induction, again to prove some results in computational geometry. Exercise 24 asks students to use strong induction to show that the deferred acceptance algorithm, introduced in the exercises of Section 3.1, is optimal for suitors. Exercises 29, 30, and 32 ask students to find errors in an incorrect proofs using strong induction.

Exercise 36 uses the well-ordering property to show that the greatest common divisor of two positive integers can be written as a linear combination of these integers. Exercise 39 presents a challenging paradox involving the well-ordering principle.

SECTION 5.3 Recursive Definitions and Structural Induction

Goals: To show how functions, sequences, and sets can be defined recursively and to show how to use various forms of induction, including structural induction, to prove properties of such entities.

Prerequisites: Chapters 1 and 2, and Sections 4.1,4.3,5.1, and 5.2.

Advice: Make it clear why recursively defined functions and sequences are well-defined as a consequence of mathematical induction. Use Example 4 to prove Lamé's Theorem, which establishes the complexity of the Euclidean algorithm. This theorem was an early result in computational complexity, pre-dating by a century the modern interest in this subject.

Describe how sets and structures can be recursively defined and illustrate this with Example 5. Explain how sets of strings are defined recursively and cover Example 7, which shows how the length of a string can be recursively defined. Cover Example 8 or Example 9, which illustrate how well-formed formulae are defined. You can further illustrate the importance of recursive definitions of sets by explaining how the set of rooted trees and the sets of extended binary and full binary trees are defined (Definitions 3–5).

Introduce the technique of structural induction and provide some examples of how it is used to prove results about recursively defined sets. Useful illustrations of proofs using structural induction are provided by Example 11, which proves a result about well-formed formulae, Example 12, which proves that the length of the concatenation of two strings is the sum of the lengths of the two strings, and Theorem 2, which proves a result about full binary trees.

This section concludes with coverage of a generalized form of mathematical induction and the illustration of how generalized induction can be used to prove a result about ordered pairs of nonnegative integers (Example 13).

Exercises: Assign some exercises on the Fibonacci numbers (see Exercises 12–19). For some challenging exercises have students work with the number of partitions of an integer (see Exercise 47) or the Ackermann function (see Exercises 48–55). Exercises 23, 24, 25, 28, 29, and 31 ask for recursive definitions of sets and strings, and Exercise 40 asks for a recursive definition of the set of bit strings with more zeros than ones. For exercises involving structural induction see Exercises 32, 33, 36, 43, and 44. Exercises 45–46 involve generalized induction.

SECTION 5.4 Recursive Algorithms

Goals: To introduce the concept of a recursive algorithm, to construct recursive versions of some algorithms, and to illustrate how to prove that a recursive algorithm is correct.

Prerequisites: Chapters 1–3 and Sections 4.1, 4.3, and 5.1–5.3.

Advice: The concept of a recursive algorithm is extremely important but difficult for students to master. Your students may have studied this topic in computer science courses, so here we concentrate on how recursion relates to some of the algorithms we have studied and the complexity of using recursion versus iteration. Cover some of Examples 1–6, which present a variety of recursive algorithms: for computing factorials, powers of real numbers, modular powers, and greatest common divisors, and for linear and binary searching. Be sure to cover Example 4, which provides an efficient recursive algorithm for modular exponentiation; students will appreciate seeing a trace of how this algorithm computes a particular value.

Proving that recursive algorithms are correct is an important application of strong induction. To introduce this application, go over Example 7, which illustrates the proof that a simple recursive algorithm is correct, and Example 8, which proves that the efficient recursive algorithm for modular exponentiation presented in Example 4 is correct.

Spend some time explaining the difference between an iterative algorithm and a recursive algorithm. Then compare and contrast the iterative and recursive approaches for computing Fibonacci numbers; this shows that recursion can be considerably less efficient than iteration. Recursive algorithms can be simple to specify once the concept is understood, but can be computational quagmires.

Be sure to introduce the merge sort algorithm and if you have time, explain how to estimate its computational complexity (Theorem 1).

Exercises: Exercises 1–6 ask for traces of recursive algorithms with specific input values; you may want to assign these if your students have trouble grasping how recursive algorithms work. Exercises 7–15, 17, 23, and 24 ask students to devise their own recursive algorithms. Be sure to assign some of Exercises 18–22, which ask students to prove that recursive algorithms are correct. Exercises 29–31 give students the chance to work through the recursion versus iteration question for a sequence similar to the Fibonacci numbers. Exercises 37–38 ask for recursive algorithms to find the reversal and ith power of a string, and Exercises 39–40 ask students to prove that these algorithms are correct. The quick sort is introduced and studied in Exercises 50–55.

SECTION 5.5 Program Correctness

Goals: To introduce the concept of program correctness and to demonstrate how to prove that programs are correct.

Prerequisites: Chapters 1 and 2 and Sections 3.1 and 5.1–5.4.

Advice: Drawing a connection between algorithms and proofs gives students who are interested in computer science but scornful of the value of proofs an appreciation of why mathematical reasoning is important. In this section we present one scheme for proving that programs are correct. This scheme is based on the concept of initial and final assertions. Some basic rules of inference for showing that programs are correct are presented. The examples are simple, but they illustrate the major ideas of the subject. We introduce the concept of a loop invariant, and in Example 4 we use mathematical induction to show how to verify a loop invariant. In Example 5 we show how to verify the correctness of a program using a combination of the various rules of inference.

Exercises: Exercises 1–4 ask for proofs that some simple programs are correct. Exercise 5 asks students to devise a rule of inference for conditional statements with one or more **else** . . . **if** clauses; Exercise 6 asks that this rule of inference be used to verify a program. Exercise 7 asks students to use a loop invariant to verify the correctness of a program.

CHAPTER 6 Counting

Overview: The goal of Chapter 6 is to present a rich set of basic counting techniques. We begin with the product and sum rules, as well as a brief mention of inclusion–exclusion; most counting methods are based on these fundamental principles. We stress throughout the chapter that finding an appropriate technique and model is the substantial part of the solution of a counting problem; applying the appropriate formula once this has been done is the easy part. We introduce the pigeonhole principle and show how it can be used to prove a variety of results. Permutations and combinations are studied formally in Section 6.3, with Section 6.4 devoted specifically to the binomial coefficients. We generalize to counting problems with repetitions allowed and to counting the ways to distribute objects into boxes in Section 6.5. Finally, in the concluding section of the chapter, we discuss algorithms for generating some of the combinatorial objects we have studied.

SECTION 6.1 The Basics of Counting

Goals: To introduce basic counting rules and to show how they are used to solve a variety of counting problems.

Prerequisites: Chapters 1 and 2.

Advice: This section contains a discussion of the product rule and the sum rule. Discuss Example 6 to illustrate the use of the product rule; this example counts the number of functions from a set with m elements to a set with n elements. Similarly, Example 7 uses the product rule to count the number of one-to-one functions from a set with m elements to a set with n elements, and Example 8 uses the product rule to count telephone numbers. Example 10 counts the number of subset of a finite set. Example 11 introduces the use of combinatorics to study DNA and RNA.

The sum and product rules provide the foundation for a large number of sophisticated enumeration methods. Show how counting problems (such as enumerating valid passwords on a computer system, discussed in Example 16, or counting Internet addresses, discussed in Example 17) can be solved using a combination of the two rules. Briefly introduce the subtraction principle for counting (which is the inclusion–exclusion principle for two sets); this will be studied in more depth in Chapter 8 and the division principle for counting. Explain how tree diagrams can be used in counting by considering the number of outcomes of a playoff series, as is done in Example 22 (it is amusing when the World Series, basketball, or Stanley Cup playoffs are taking place when you cover this).

Exercises: Exercise 41 asks for the number of bit strings of length n that are palindromes. Exercises 42–43 involve DNA or RNA sequences. Exercises 44–45 involve the division principle of counting. You may want to assign some of Exercises 48–53 which require the use of inclusion–exclusion. Exercises 54–61 require the use of both the sum rule and the product rule; the last three of these are typical of the type of practical counting problem that arises in computer science or engineering. Tree diagrams can be used to solve Exercises 64–69. Exercise 74 asks for the number of different strings of data that can be transmitted using an IP datagram.

SECTION 6.2 The Pigeonhole Principle

Goals: To introduce the pigeonhole principle and show how to use it in enumeration and in proofs.

Prerequisites: Chapters 1 and 2 and Sections 4.1, 4.3, and 6.1.

Advice: Students have trouble drawing valid conclusions from the pigeonhole principle. You can clarify this using Figure 1, which illustrates what you can and cannot conclude from the pigeonhole principle, namely that if there are more pigeons than pigeonholes, some pigeonhole contains more than one pigeon, but some pigeonholes may contain no pigeons, and others may contain many pigeons. Example 4 provides an interesting application of the pigeonhole principle.

Describe the generalized pigeonhole principle. Make sure students understand how to solve problems such as that posed in Example 8. Often they will give answers that are too small because they do not understand the use of the ceiling function in the generalized pigeonhole principle.

Students have trouble with subtle types of arguments using the pigeonhole principle. Anticipate confusion if you cover any of Examples 10, 11, or 12. I suggest covering Example 13, which solves the puzzle about three mutual friends or enemies in a group of six people. This application of the pigeonhole principle is the simplest result of Ramsey theory.

Exercises: Assign Exercise 8, which relates the pigeonhole principle to the fact that a function cannot be one-to-one if its domain has more elements than its codomain. Exercises 24–30 develop further results from Ramsey theory.

SECTION 6.3 Permutations and Combinations

Goals: To introduce permutations and combinations, to solve counting problems using them, and to show how theorems are proved by combinatorial arguments.

Prerequisites: Section 6.1 and its prerequisites.

Advice: Students need to understand clearly that a combination involves an unordered selection of objects from a set with no repetition allowed, while a permutation involves an ordered selection of objects from a set with no repetition allowed. (We will consider the cases when repetition is allowed in Section 6.5.)

This section also introduces the idea of a combinatorial proof and describes the two major types of combinatorial proof, double counting proofs and bijective proofs. I suggest you present both of these types of proof of Corollary 2.

Exercises: The questions asked in Exercises 8–39 require students to use permutations and combinations to solve counting problems. You may want to assign Exercises 40–42, which deals with the number of circular arrangements of objects. Exercises 43–45 require careful reasoning to count the number of ways races can end when ties are allowed.

SECTION 6.4 Binomial Coefficients

Goals: To introduce the binomial theorem and to show how combinatorial identities can be proved by combinatorial arguments.

Prerequisites: Sections 6.1 and 6.3 and their prerequisites.

Advice: Note that we use $C(n, r)$ and $\binom{n}{r}$ interchangeably to denote the number of r-combinations of a set with n elements. Present the combinatorial proofs of Pascal's identity and the binomial theorem. Because students are uncomfortable with proofs of this ilk, explain carefully how counting arguments really can be used to prove theorems.

Exercises: Exercise 14–17 ask students to prove inequalities involving binomial coefficients. (The result of Exercise 17 is used later in the text.) Combinatorial proofs are required in Exercises 21–22, 27–30, and 38. Exercise 32 asks for a proof by mathematical induction of the Binomial Theorem. Compared to the combinatorial proof, this alternative approach is substantially more complicated. Exercise 39 asks students to conjecture a formula or rule that generates the terms of a sequence from its initial terms; the intended answers here involve binomial coefficients.

SECTION 6.5 Generalized Permutations and Combinations

Goals: To solve counting problems involving permutations and combinations with repetition allowed and permutations where objects may be indistinguishable. To solve counting problems using the notion of distributing objects into boxes, where both the objects and the boxes may be distinguishable or indistinguishable.

Prerequisites: Sections 6.1, 6.3, and 6.4 and their prerequisites.

Advice: This section begins by covering permutations and combinations with repetition allowed. To motivate the discussion, it helps to discuss a variety of counting problems, determining whether you are enumerating combinations or permutations, with or without repetition allowed. Stress that determining the appropriate model is more important than memorizing the formulae. Students often have trouble with combinations with repetition allowed. Give lots of simple examples to motivate this and show the connection with determining nonnegative integer solutions to equations of the form $\sum_{j=1}^{r} x_j = n$. The "stars and bars" proof of the formula for combinations with repetition allowed is subtle and requires careful explanation.

Work through Example 7, which asks for the number of different strings that can be formed by reordering the characters in *SUCCESS*. Make sure that students see that the position of the individual $S's$ doesn't matter and only where there are $S's$ does. (One way to show this is to label the individual letters with subscripts—S_1, S_2, and S_3—and show that different permutations of the letters lead to the same string when subscripts are removed.) Once this point is clear, you will find that the proof of Theorem 3 is easy for students to understand.

Discuss how different counting problems can be solved by counting the number of ways to distribute n objects into k boxes, where both the objects and the boxes may be distinguishable or indistinguishable. Explain how to approach each of the four different possibilities; point out that counting the number of ways to distribute n objects, either distinguishable or indistinguishable, into k distinguishable boxes is equivalent to one of the counting problems studied earlier in the section. Point out that it is much harder to count the number of ways n distinguishable or indistinguishable objects can be distributed into k indistinguishable boxes.

Exercises: Exercises 1–13 involve counting problems where repetition is allowed. Exercises 30–37 involve counting permutations with indistinguishable objects. Assign some of Exercises 15–16, which ask for the number of solutions in integers to equations of the form $\sum_{j=1}^{r} x_j = n$ where some constraints are present. You may want to assign some of Exercises 50–59, which deal with counting the number of ways to distribute distinguishable or indistinguishable objects into distinguishable or indistinguishable boxes. You may also want to assign Exercise 63, which asks for a proof of the Multinomial Theorem.

SECTION 6.6 Generating Permutations and Combinations

Goals: To introduce algorithms for generating permutations and combinations.

Prerequisites: Section 6.3 and its prerequisites.

Advice: There are many occasions when we need to generate the combinations or permutations of a set. Explain that there are many possible procedures for generating permutations and combinations, so that the ones in the text are not the only such algorithms. It helps to work through some small examples to see how an algorithm generates permutations or combinations.

Exercises: Exercises 14–17 develop another algorithm, based on Cantor expansions, for generating permutations. This set of exercises is challenging, but quite interesting.

CHAPTER 7 Discrete Probability

Overview: The goal of Chapter 7 is to develop basic concepts of discrete probability, including conditional probability and expected values, and we show how to use these concepts to study the average case complexity of an algorithm.

SECTION 7.1 An Introduction to Discrete Probability

Goals: To introduce discrete probability theory.

Prerequisites: Chapters 1 and 2 and Sections 3.3, 5.1–5.2, and 6.3–6.4.

Advice: Students in a first course in discrete mathematics should be exposed to the concept of probability. There are many reasons why probability is important for all students, but it is particularly important for computer science students, who need to understand the concept of the average-case complexity of an algorithm, a topic covered in Section 7.4.

Probability can be illustrated in an appealing way by discussing how it applies to lotteries and to card games. Students enjoy learning how to compute the odds of winning the "pick six" lottery by picking six correct numbers out of the first n positive integers where n is between 40 and 50. If you calculate probabilities from poker, make sure that all your students know the contents of a standard deck of cards, what the 13 kinds of cards are, and what the four suits are. Finding the probability of a full house involves some careful counting; it is done in Example 6. Subtle aspects of probabilistic reasoning are illustrated by the notorious Monty Hall Three Door Puzzle (Example 10). Exploring correct and incorrect lines of reasoning involving this puzzle can be quite instructive.

Exercises: Exercises 8–20 ask for the probability of various kinds of poker hands. Exercise 35 asks for an analysis of different bets in roulette. Exercise 38 introduces the concept of independent events. Exercise 41 poses the famous problem that lead to the development of probability theory in the 17th century.

SECTION 7.2 Probability Theory

Goals: To introduce important concepts from discrete probability, including conditional probability, independence, random variables, and the binomial distribution. To introduce the notions of probabilistic algorithms and the probabilistic method.

Prerequisites: Section 7.1 and its prerequisites.

Advice: Describe the rules that probabilities of finitely many outcomes must obey. Then show how these rules are met by probabilities determined using Laplace's definition. Examples 3 and 4 determine the conditional probabilities of events, and Examples 5, 6, and 7 deal with independence of events.

Be sure to cover the famous birthday problem (Example 13) and its extension to computing the probability of collisions in hashing functions (Example 14). You may want to introduce probabilistic algorithms; Example 16, which covers probabilistic primality testing, is important for generating large primes for cryptographic applications.

You may want to introduce the probabilistic method, which shows how probability theory can be used in nonconstructive existence proofs. Theorem 4 illustrates how this method can be used to establish a lower bound on some Ramsey numbers.

Exercises: Exercises 18–22 involve variations of the birthday problem. Exercises 36–37 establish a formula for the probability of the union of pairwise disjoint events. Exercises 39 provides another example of how the probabilistic method is used. Exercise 40 describes a probabilistic algorithm for determining whether a permutation of the integers 1 through n has already been sorted.

SECTION 7.3 Bayes' Theorem

Goals: To introduce Bayes' Theorem and to demonstrate how it can be applied to solve problems involving conditional probabilities. Also, to illustrate how Bayes' Theorem can be used to construct spam filters.

Prerequisites: Sections 7.1 and 7.2 and their prerequisites.

Advice: Use Example 1 to motivate the key concept behind Bayes' Theorem that makes it so useful, namely that extra information allows you to derive a more realistic estimate of the probability that a particular event occurs. Then state Bayes' Theorem. I recommend that you prove this important theorem in your course; its proof is straightforward and relatively short and uses basic notions from probability theory. After covering Bayes' Theorem, present Example 2, which shows how it can be used to realistically interpret the results of a diagnostic medical test. Make sure to mention explicitly the notion of a false positive, as well as the notion of a false negative.

If you cover this section, be sure to cover the application of Bayes' Theorem to spam filters. Explain how to develop the formulae here that estimate the probability that a message is spam based on the appearance of certain words in the message. Be sure to describe the assumptions needed to derive these formulae.

Exercises: Exercises 5–10 require the use of Bayes' Theorem to find probabilities of true positives, false positives, true negatives, and false negatives of different types of drug tests and diagnostic tests. Exercise 15 shows how Bayes' Theorem can be used to solve the Monty Hall puzzle. Exercises 18–23 involve the use of Bayes' Theorem in spam filters.

SECTION 7.4 Expected Value and Variance

Goals: To introduce the concept of the expected value of a random variable. To show how the linearity of expectations can be used to solve a variety of problems. To show how expected values and their properties can be used to find the average case complexity of algorithms. To introduce the notion of the variance of a random variable and to introduce Chebyshev's inequality.

Prerequisites: Sections 7.1 and 7.2 and their prerequisites.

Advice: Describe how expected values are found; be sure to cover Theorem 1 and illustrate how it is used to compute expected values with Example 3 and in the proof of Theorem 2, which gives the expected number of success in n Bernoulli trials. Make sure to cover the linearity of expectations and illustrate the usefulness of this property by covering Examples 6 and 7, which find the expected number of hats returned correctly in the famous hatcheck problem and the expected number of inversions in a permutation.

Explain how the properties of expected values are used in average-case complexity analysis. Cover Example 8, which finds the average-case complexity of linear search, or Example 9, which finds the average-case complexity of the insertion sort.

Introduce the geometric distribution, which provides an example of an infinite sample space, and find the expectation of random variables that follow a geometric distribution (Theorem 4). Cover independence of random variables and Theorem 5, which states that the expected value of the product of independent random variables is the product of their expected values. Introduce the notion of the variance of a random variable and cover Bienaymé's formula that shows that the variance of the sum of two independent random variables is the sum of their variances. Introduce Chebyshev's inequality and explain how it is used.

Instructors should note that probability generating functions and how they are used for calculating expected values and variances are introduced in the exercise set of Section 8.4.

Exercises: Markov's inequality is introduced in Exercise 37. The average-case complexity of a variant of bubble sort and the quick sort are studied in Exercises 41 and 42, respectively. The variance in the number of fixed elements in a random permutation is studied in Exercise 43. Covariance is introduced in Exercises 44–46.

CHAPTER 8
Advanced Counting Techniques

Overview: We introduce several important counting techniques in this chapter. In particular, we show how to use recurrence relations to solve counting problems, and we show how to solve a variety of counting problems using the principle of inclusion–exclusion.

The first three sections of the chapter are devoted to recurrence relations. Section 8.1 shows how recurrence relations can be used to solve counting problems and shows how to solve some recurrence relations using iteration. Section 8.2 develops the methodology for solving linear recurrence relations with constant coefficients. Section 8.3 is devoted to divide-and-conquer algorithms and the recurrence relations used to study their complexity. Another advanced technique, the use of generating functions to solve counting problems, is introduced in Section 8.4. The final two sections of the chapter cover the principle of inclusion–exclusion and its many applications. Specifically, Section 8.5 introduces this principle and presents its proof, and Section 8.6 shows how to apply this principle by showing how it can be used to count such things as the number of primes not exceeding a positive integer, the number of onto functions from one finite set to another, and the number of derangements of a set.

SECTION 8.1 Recurrence Relations

Goals: To show how counting problems can be modeled using recurrence relations. To illustrate how recurrence relations can be used in dynamic programming algorithms.

Prerequisites: Chapters 1–3 and Sections 5.1–5.3, 6.1, 6.3, and 6.4.

Advice: Explain that a recurrence relation is a type of recursive definition, so it requires initial conditions as well as the rule for obtaining subsequent terms of the sequence. Students should be given a clear idea of what it means for a sequence to solve a particular recurrence relation. In Example 2 we determine the number of moves required to solve the famous Tower of Hanoi puzzle. This example introduces the use of iteration to solve recurrence relations. Students will find it interesting that the generalization of the Tower of Hanoi problem to four pegs (known as the Reve's puzzle) still contains open questions. Cover Example 3, which shows how recurrence relations can be used to find bit strings of a particular length that do have two consecutive 0's. If time permits, cover Example 5, which introduces the Catalan numbers in the enumeration of ways to insert parentheses into the product of numbers to determine the order of multiplications.

You may want to introduce dynamic programming, an important algorithmic paradigm that uses recurrence relations to construct solutions using overlapping subproblems. A dynamic programming algorithm is developed for constructing a schedule for talks so that the total audience of these talks is the largest possible.

Exercises: Exercises 2–5 give students the opportunity to set up some relatively simple recurrence relations. I suggest assigning some of Exercises 7–10 and 24–25. These provide some good examples of constructing recurrence relations for the number of bit strings with certain properties. Exercises 33–37 provide a set of challenging exercises concerning the Josephus problem, and Exercises 38–45 involve the generalization of the Tower of Hanoi puzzle to four pegs (the Reve's puzzle). Exercises 46–52 introduce the concept of a backward difference and relate recurrence relations to difference equations. You may want to assign some of these exercises. Exercises 56 and 57 develop dynamic programming algorithms. In particular, Exercise 57 constructs an algorithm for solving the matrix chain multiplication problem introduced in Section 3.3.

SECTION 8.2 Solving Linear Recurrence Relations

Goals: To solve linear recurrence relations with constant coefficients.

Prerequisites: Section 8.1 and its prerequisites.

Advice: We state and prove methods for solving linear recurrence relations with constant coefficients. We start with the simplest case (homogeneous without repeated roots of the characteristic equation) and advance to the general case (Theorems 4 and 6). Show students what it means for a recurrence relation to be linear, to be homogeneous, and to have constant coefficients, using examples where such properties occur and other examples where they do not. Make sure the concept of the degree of such a recurrence relation is clear (e.g., $a_n = a_{n-1} + a_{n-8}$

has degree 8). You may need to review briefly how to factor polynomials. Students who have studied differential equations will recognize the analogy between the methods presented here and those used to solve linear differential equations. If time permits, cover the case of repeated roots for homogeneous recurrence relations. Explain how nonhomogeneous linear recurrence relations are solved, including the case where the right-hand side is a product of a power of a constant and a polynomial.

It might be advisable to discuss the use of computer algebra systems (such as *Maple* or *Mathematica*), which can solve recurrence relations as easily as a pocket calculator can perform long division. In fact these systems have a variety of powerful features important to discrete mathematics.

You may want to cover Example 4, which provides an explicit formula for the Fibonacci numbers, Example 5, which illustrates what needs to be done when the characteristic equation has repeated roots, and Example 6, which shows how to solve a linear homogeneous recurrence relation of degree greater than two.

Exercises: Exercises 38–39 involve solving linear homogeneous recurrence relations with complex roots. We have avoided this situation in most other places because many students will not be comfortable working with complex numbers. Exercises 48–50 introduce the case of nonconstant coefficients.

SECTION 8.3 Divide-and-Conquer Algorithms and Recurrence Relations

Goals: To study the complexity of divide-and-conquer algorithms with functions that satisfy a special kind of recurrence relation.

Prerequisites: Chapters 1 and 2, Chapter 3 (especially Section 3.2), and Sections 5.1–5.3, 6.1, and 8.1.

Advice: Go over different divide-and-conquer algorithms, including binary searching, finding the maximum and minimum of a set of integers, the merge sort, and fast multiplication of integers. Such algorithms are described in Examples 1–4. Student often have trouble with divide-and-conquer recurrence relations, which are the recurrence relations arising in the complexity analysis of divide-and-conquer algorithms. State and show how to use Theorem 1, and perhaps Theorem 2 (the master theorem), which give big-O estimates for the values of functions that satisfy divide-and-conquer recurrence relations of the form $f(n) = af(n/b) + g(n)$ with $g(n) = c$ or $g(n) = cn^d$. Mention that the proof of Theorem 1 produces a formula for $f(n)$ when n is a power of b. Show how to apply the master theorem to give big-O estimates for the complexity of various algorithms (see Examples 9–11). If time permits, describe the closest-pairs algorithm described in Example 12 and explain how to estimate its complexity using the master theorem.

Exercises: Exercises 14, 17, 18, 23, and 28 give students the opportunity to analyze divide-and-conquer algorithms not discussed in the text.

SECTION 8.4 Generating Functions

Goals: To introduce the notion of a generating function, to show how generating functions can be used to model and solve counting problems, and to show how generating functions can be used to solve recurrence relations.

Prerequisites: Section 8.2 and its prerequisites and familiarity with infinite series.

Advice: Students find generating functions more difficult to understand and appreciate than the other counting topics covered in the text. This is a pity, because generating functions provide very powerful tools for enumeration. Stress how to model counting problems using generating functions and thereby obtain a mechanical means of solving them. Students will find Table 1 invaluable in doing the exercises in this section; in fact they will certainly want to learn some of these generating functions well enough not to need to refer to the table or work them out each time.

It might be advisable to discuss the use of computer algebra systems (such as *Maple* or *Mathematica*), which are powerful tools when working with generating functions. The *Student's Solutions Guide* and the *Instructor's Resource Guide* give some details of the use of *Maple* in the solutions of relevant exercises in this chapter. (They also give details on using *Maple* to solve recurrence relations.)

We also illustrate how generating functions can be used to solve recurrence relations—see Examples 16–17. A brief optional final subsection shows that generating functions can be used to prove identities.

Exercises: Exercises 13–29 get at the heart of the issue—using generating functions to model and solve counting problems. Exercises 32–39 ask students to solve recurrence relations using generating functions, including producing an explicit formula for the Fibonacci numbers; the algebra can get a little overwhelming at times, and you might wish to encourage students to use a computer algebra system. Exercises 42–43 ask for proofs of combinatorial identities using generating functions. Exercise 44 outlines a way to derive an explicit formula for the sum of the first n squares. Exponential generating functions are introduced in the preamble to Exercise 45, and probability generating functions are introduced in the preamble to Exercise 57.

SECTION 8.5 Inclusion–Exclusion

Goals: To introduce the principle of inclusion–exclusion and show how it is used to solve some simple counting problems.

Prerequisites: Chapters 1 and 2, Sections 4.1, 5.1–5.3, 6.1, and 6.3–6.5.

Advice: Motivate the proof of the principle of inclusion–exclusion using the case of three sets. Show that elements in one, two, or all three of the sets are counted just once. Cover the proof of inclusion–exclusion carefully. Show that each element in the union of the sets is counted exactly once. This principle can also be proved by mathematical induction (see Exercise 22). You may wish to cover Example 2, which deals with divisibility of integers. This example foreshadows the treatment of the sieve of Eratosthenes in Section 8.6.

Exercises: Exercise 12 is a good thought question; it asks students to determine the number of positive integers not exceeding 1000 that are either squares or cubes. There are $2^n - 1$ terms in the expansion given by the principle of inclusion–exclusion; Exercises 18, 19, and 21 give students the opportunity to make sure that they understand this general theorem.

SECTION 8.6 Applications of Inclusion–Exclusion

Goals: To use inclusion–exclusion to solve complicated counting problems, such as determining the number of onto functions, the number of primes less than a specified integer, and the number of derangements of a finite set of objects.

Prerequisites: Section 8.5 and its prerequisites.

Advice: Make sure that the alternative formulation of inclusion–exclusion in terms of properties is clear. Illustrate the use of this formulation with Example 1, which counts the number of solutions in nonnegative integers of a linear equation with constraints. I suggest that you cover the application to the sieve of Eratosthenes, counting primes less than 100 using inclusion–exclusion (be sure to adjust the number of integers not divisible by 2, 3, 5, or 7 to exclude the integer 1, but to include these four primes). Example 2 illustrates how to count functions from a finite set onto another finite set. Example 3 shows how certain enumeration problems can be solved by counting onto functions. If you have time, go over the hatcheck problem; this can be very entertaining to students. For students who know about infinite series it is worthwhile explaining how the probability that no one receives the correct hat tends extremely rapidly to $1/e$ as $n \to \infty$.

Exercises: I suggest that you assign Exercises 6 and 7; these require students to do a little thinking besides just setting up inclusion–exclusion formulae. Exercises 9–11 are enumeration problems solved by counting onto functions. Exercise 15 is a good thought question; it asks students to count permutations with various properties, including derangements. Exercise 23 asks for a formula for the number of integers not exceeding n that are relatively prime to n, i.e., $\phi(n)$. This is an excellent application of inclusion–exclusion. (My enthusiasm for number theory seems to be showing here.)

CHAPTER 9
Relations

Overview: We study an important discrete structure in this chapter, namely the relation. We first define binary relations and relations on a set. We show how to use zero–one matrices and directed graphs to represent relations on a set. We study the different properties of binary relations and discuss in detail two important classes of binary relations: equivalence relations and partial orderings. We show how to find closures of binary relations with respect to various properties, devoting considerable attention to algorithms for constructing the transitive closure of a relation. We briefly study n-ary relations and describe the applications of these relations to models for data bases. The material in this chapter is straightforward with the exception of the material on transitive closures, some properties of equivalence relations, and some results concerning partial orderings.

SECTION 9.1 Relations and Their Properties

Goals: To introduce the concept of a relation and basic properties of relations, including the reflexive, symmetric, antisymmetric, and transitive properties.

Prerequisites: Chapters 1 and 2, Sections 4.1, 4.3, 5.1–5.3, 6.1, and 6.3–6.4.

Advice: Explain that the graph of a function is a relation, whereas relations can express more general correspondences between elements of two sets. Students have trouble verifying properties of relations when the conditions to check are vacuous, such as showing that the relation $\{(0,1),(0,2)\}$ on the set $\{0,1,2\}$ is transitive because there are no pairs of the form (a,b) and (b,c) in this relation. There is often confusion concerning the relationship between the terms *symmetric* and *antisymmetric*; make sure students know that there are relations that are both symmetric and antisymmetric and relations that are neither symmetric nor antisymmetric.

You may want to cover Examples 6 and 16, which determine the number of relations and reflexive relations, respectively, on a set with n elements.

Exercises: Assign Exercise 10, which asks for examples of relations that are both symmetric and antisymmetric and others that are neither symmetric nor antisymmetric. Be sure to assign some exercises that involve the inverse relation R^{-1} of a relation R; this is defined in the preamble to Exercise 26 and arises in Exercises 26–29. Counting the relations with a particular property is a way to review counting techniques and reinforce students' understanding of the property (students will find such exercises difficult). Example 16 computes the number of reflexive relations on a set with n elements, and Exercise 47 asks for a similar computation involving other properties. You may want to assign some of Exercises 11–15; these deal with irreflexive relations, defined in the preamble to Exercise 11. Exercise 49 asks students to find the error in an incorrect proof involving properties of relations.

SECTION 9.2 n-ary Relations and Their Applications

Goals: To introduce the concept of n-ary relations and show how these relations are used to represent data bases.

Prerequisites: Chapter 1 and Sections 2.1–2.3, 3.1, 4.1, 5.1–5.2, 6.1, and 9.1.

Advice: Students like seeing an application of relations, an abstract subject, to computer science, so this material goes over quite well. I suggest describing how operations on n-ary relations relate to operations of practical interest on large data bases, such as student records or employee data bases. The material in this section is straightforward. However, make sure students understand that different records can collapse to the same record when projections are applied and that the join of two relations can contain more than one record, or no records, for each record in each of the relations being joined. Stress that a domain of an n-ary relation is a key if and only if its value is never the same in two different records *that ever appear* in the data base. Give examples of the selection operator and explain how logical connectives can be used to define the condition that n-tuples must satisfy to be selected. You may want to briefly discuss the database query language SQL and how it is used to carry out the operations discussed in this section.

Exercises: Exercises 10–13 involve the selection operator. Properties of the selection operator are covered in Exercises 20–24; properties of the projection operator are covered in Exercises 25–27. Exercise 19 illustrates the various things that can happen when a join of two relations is formed. Exercises 28–29 deal with SQL.

SECTION 9.3 Representing Relations

Goals: To show how relations can be represented using zero–one matrices and directed graphs.

Prerequisites: Sections 2.6 and 9.1 and their prerequisites.

Advice: This section discusses two ways to represent relations on a set: zero–one matrices and directed graphs. Some of the material in this section and later sections depends on arithmetic with zero–one matrices, which was covered in Section 2.6. Relate properties of relations with the corresponding properties of the zero–one matrices that represent them.

We introduce directed graphs in this section, and use them throughout the remainder of the chapter to represent relations. Again, you should relate properties of relations to the corresponding properties of the directed graphs that represent them. Giving the directed graph interpretation of the transitive property is quite helpful, because students can readily see whether there is always an edge between two vertices that are connected by a path of length 2. (Don't forget about missing loops!) Directed graphs will be studied in greater depth in Chapter 10.

Exercises: Exercises 31–32 ask students to use the directed graph of several relations to determine whether these relations are reflexive, symmetric, antisymmetric, and/or transitive.

SECTION 9.4 Closures of Relations

Goals: To introduce the concept of the closure of a relation with respect to a property, and to develop algorithms for constructing transitive closures.

Prerequisites: Section 9.3 and its prerequisites.

Advice: This is the most difficult section of the chapter, because constructing transitive closures can be complicated. There is no particular difficulty constructing the reflexive and symmetric closures of a relation. Students are tempted to construct the transitive closure of a relation by just adding the pairs that are missing, i.e., pairs not in the relation of the form (a, c) where (a, b) and (b, c) are in the relation. Show them that this is not sufficient with an example (e.g., a path of length 3). They also sometimes forget to add (a, a) and (b, b) when (a, b) and (b, a) are in the relation.

The concept of an interior vertex of a path and how this concept is used in Warshall's algorithm is confusing. Make sure to give at least one example of this before embarking on the development of Warshall's algorithm, as is done in the text. Be sure to give a careful exposition of Lemma 2.

Exercises: You may want to assign Exercises 15 and 35, which show that there are some properties for which the closure of a relation does not exist.

SECTION 9.5 Equivalence Relations

Goals: To study equivalence relations and their equivalence classes.

Prerequisites: Section 9.3 and its prerequisites.

Advice: This section introduces the concept of an equivalence relation. I suggest you discuss Examples 3–5, which present three different equivalence relations. Students often have trouble with the notion of an equivalence class. Make sure you cover Example 9, which finds the equivalence classes of the congruence modulo m equivalence relation. Emphasize that two different elements can be representatives of the same equivalence class. It helps to show that $[1]_4 = [5]_4 = [-3]_4$, and so on. Example 11 shows how the possible identifiers in the C programming language correspond to equivalence classes of the equivalence relation where two strings are equivalent if they agree in their 31 initial characters (or are equal). Spend sufficient time showing that the equivalence classes of an equivalence relation form a partition. Students find this confusing. Also be careful when showing that a partition of a set determines an equivalence relation. Students find this very subtle.

Exercises: Exercise 9 presents an important way to establish that a relation is an equivalence relation. It can be used in many of the subsequent exercises, including Exercises 11–16. Exercises 15 and 16 establish some equivalence relations on ordered pairs of positive integers. These equivalence relations play important roles in the construction of the set of integers and the set of rational numbers from the set of positive integers. Refinements of partitions are the subject of Exercises 49–50.

SECTION 9.6 Partial Orderings

Goals: To study partial orderings and their properties and applications. You may want to cover topological sorting and scheduling. You may also want to discuss lattices and their application to information flow.

Prerequisites: Section 9.3 and its prerequisites.

Advice: This section introduces the concept of a partial ordering. I find that students find it difficult to think of partial orderings (other than the usual less than or equal to relation), such as divisibility of positive integers and set inclusion, as ways to order elements of a set. Also, make it clear that the notation for a partial order (the curly less than or equal to sign, $\preceq$) is general, and refers to more than just the less than or equal to relation ($\leq$). Show how to extend lexicographic ordering to strings. This concept is important in computer science. Hasse diagrams can be confusing. Much of the information they contain must be inferred. Carefully go through the process of constructing a Hasse diagram at least once, such as is done in Examples 12 and 13.

A lot of terminology about posets is presented here. Make sure students understand the distinction between minimal and least elements and maximal and greatest elements. The concepts of greatest lower bounds and least upper bounds also require careful explaining. Once these topics are covered, you can define a lattice. Cover Example 25, which describes the lattice model of information flow; this concrete example helps motivate the definition of a lattice and shows how lattices can be used in applications. The application of topological sorting to project scheduling is something that most students find interesting.

Exercises: Exercises 12 and 13 introduce the concept of the dual of a poset. I suggest assigning Exercise 40, which asks students to show that there is at most one greatest and one least element of a poset. Exercises 64 and 65 ask for scheduling the tasks required in building a house and carrying out a software project; this can be done using topological sorting.

CHAPTER 10 Graphs

Overview: This chapter provides an introduction to graph theory and its applications. Sections l0.1–10.4 cover the basics of the subject, including graph terminology, how graphs are represented, isomorphism, and connectivity. The remaining sections cover particular topics that are treated independently. These topics are Euler and Hamilton paths and circuits, shortest-path problems in weighted graphs, planarity, and coloring. Cover any or all these as time permits.

The graph theoretic terminology used in the text is based on one of the most commonly used sets of such terminology. It is consistent and mostly self-explanatory. However, there are many other ways to define the terms used in graph theory. Pay careful attention to the precise definitions given here and warn your students that there is no standard terminology in graph theory.

SECTION 10.1 Graphs and Graph Models

Goals: To introduce the notion of a graph and to show how to build graph models and to demonstrate the wide applicability of graph models.

Prerequisites: Chapters 1 and 2 and Section 9.3.

Advice: This section introduces the concept a graph. You can quickly cover the basic terminology of graph theory here because students rarely have trouble with it. You may wish to use the modeling of a computer network as motivation.

A variety of graph models are introduced. These demonstrate the wide range of subjects to which graph theory can be applied. Stress that when building a graph model, decisions need to be made whether the edges should be directed or undirected, whether loops are needed, and whether multiple edges are required. Cover some or all of Examples 1–14, which involve applications to a diverse range of disciplines. These applications involve social, communication, information, transportation, and biological networks, as well as software design and tournaments. The social networks discussed include friendship graphs, influence graphs, and collaboration graphs (such as the Hollywood graph and academic collaboration graphs). Biological networks include niche overlap graphs and protein interaction graphs. Students in your class may have interests in a variety of other disciplines, so stress that graph theory has applications to almost every (and maybe every) discipline. References listed in the Suggested Readings section of the text will help you direct students to applications in their particular areas of interest.

Exercises: Intersection graphs are introduced in Exercise 13. Exercises 11 and 12 draw the connection between undirected graphs and relations with certain sets of properties.

SECTION 10.2 Graph Terminology and Special Types of Graphs

Goals: To introduce some of the basic terminology of graph theory and some basic results about graphs. To describe some important families of graphs and to introduce the notion of a bipartite graph.

Prerequisites: Chapters 1 and 2 and Sections 3.1, 6.1–6.3, 9.3, 9.5, and 10.1.

Advice: The material in this section presents no particular difficulties. When showing that the sum of the degrees of the vertices of an undirected graph is even, make the analogy between people shaking hands and vertices being adjacent (that is why this result is called the "handshaking" theorem). Make sure to cover the families of graphs introduced in this section: complete graphs, complete bipartite graphs, cycles, wheels, and cubes. They are used extensively as examples in subsequent sections. Students sometimes have difficulty with the notion of a bipartite graph. When explaining this you might want to use the language of partitions of sets. Theorem 4 shows that a graph is bipartite if and only if it is 2-colorable. This material foreshadows the coverage of graph colorings in Section 10.8.

You may want to cover Example 14, which illustrates how bipartite graphs are used to model matchings. Matchings in bipartite graphs are also covered, including Hall's marriage theorem which establishes necessary and sufficient conditions for complete matchings in a bipartite graph. You might also want to cover Examples 16 and 17, which show how graphs are used to model local area networks and interconnection networks for parallel computers, respectively.

Exercises: The notion of the degree sequence of a graph is introduced in the preamble to Exercise 36, and the notion of a graphic sequence is introduced in the preamble to Exercise 42. You may want to assign Exercises 42 and 43, which ask whether certain sequences are graphic, because these exercises involve several concepts covered in the section. You may want to assign Exercises 53–54, which introduce and use the concept of a regular graph (defined in the preamble to Exercise 53).

SECTION 10.3 Representing Graphs and Graph Isomorphism

Goals: To show how to represent graphs and to study isomorphism of graphs.

Prerequisites: Sections 10.1 and 10.2 and their prerequisites and Section 2.6.

Advice: We describe several ways to represent graphs: adjacency lists, adjacency matrices, and incidence matrices. This material is straightforward, although it does require familiarity with matrices. You may want to discuss the circumstances under which each of these different ways to represent graphs is preferable.

The last part of this section is devoted to isomorphism of graphs. Stress how invariants of graphs can be used to show that two graphs are not isomorphic, but cannot show that two graphs are isomorphic. It is useful to give students a "shopping list" of invariants to examine: number of vertices, number of edges, degree sequence, and so on. In the text we discuss isomorphism only for simple graphs. We give a nontrivial example of determining whether two graphs are isomorphic in Example 11.

Exercises: Exercises 34–44 ask students to determine whether pairs of graphs are isomorphic. Be sure to assign a variety of these exercises. You should assign Exercise 60, which asks students to define isomorphism for directed graphs, and then have them do Exercises 61–64, which ask whether pairs of directed graphs are isomorphic.

SECTION 10.4 Connectivity

Goals: To introduce the notions of paths and circuits in graphs and to define connectivity of graphs.

Prerequisites: Section 10.3 and its prerequisites.

Advice: The material in this section presents no particular difficulties. Make sure that the definitions of paths, circuits, and simple paths and circuits are clear. (Unfortunately the terminology for the concepts discussed in this section varies in discussions by different authors.) Discuss the meaning of paths in acquaintanceship graphs, and collaboration graphs: see Examples 2 and 3 (which defines Erdős numbers and Bacon numbers).

You may want to go into greater depth into how connected a graph is. In particular, define the notions of cut vertices and cut edges. Continue by defining the vertex connectivity of a graph and the notion of k-connectivity. Also, define the edge connectivity of a graph and discuss the inequality between the vertex connectivity, edge connectivity, and minimum degree of a vertex of a graph. If you wish, describe the applications of vertex and edge connectivity to communications and highway networks.

Explain the difference between strongly connected and weakly connected directed graphs, because the difference between these concepts is sometimes confusing. Introduce the notions of connected components in undirected graphs and strongly connected components in directed graphs; illustrate these notions with the connected components of the telephone call graph (Example 6) and the strongly connected components of the Web graph (Example 12), respectively. You may want to show how paths are used to help determine whether two graphs are isomorphic. This is illustrated in Examples 13 and 14. Go over the proof of Theorem 2, which works for all types of graphs.

Exercises: You may want to assign Exercises 40–41, which introduce the concept of a vertex basis. Students need to understand the concepts of vertex connectivity and edge connectivity to solve Exercise 54, which asks for a graph with vertex connectivity 1, edge connectivity 2, and in which the minimum degree of vertex is 3. Assign Exercise 59 if you plan to discuss trees, because the result in this exercise is used in Chapter 11. Exercises 64–66 illustrate how finding a path in a graph model can solve a puzzle.

SECTION 10.5 Euler and Hamilton Paths

Goals: To develop necessary and sufficient conditions for the existence of Euler circuits and paths, to give algorithms for constructing them, and to study Hamilton paths and circuits.

Prerequisites: Section 10.4 and its prerequisites and Chapter 3.

Advice: We introduce the famous Königsberg bridges problem here. It is something every mathematics student should see. Algorithm 1 presents one procedure for constructing Euler circuits based on the idea used in the proof that every simple graph with all vertices of even degree has such a circuit. Tell students that the situation for Hamilton paths and circuits is different from that for Euler paths and circuits: There are no known simple necessary and sufficient conditions for their existence. However, you may want to state some sufficient conditions for the existence of Hamilton circuits, such as Theorem 3 (Dirac's Theorem) and Theorem 4 (Ore's Theorem). You may wish to introduce the application of Hamilton circuits to Gray codes.

Exercises: Exercises 16–17 ask for necessary and sufficient conditions for the existence of Euler paths and circuits in directed graphs. Exercises 50–53 develop Fleury's algorithm for constructing Euler circuits. Exercises 56–64 involve the application of Hamilton paths and circuits to the knight's tour problem. The proof of Ore's Theorem is outlined in Exercise 65.

SECTION 10.5 Shortest-Path Problems

Goals: To present an algorithm for finding a shortest path in a weighted graph, and to discuss the traveling salesman problem.

Prerequisites: Sections 10.1–10.4 and their prerequisites.

Advice: We define weighted graphs in this section and describe one problem involving such graphs, namely the determination of shortest paths. We present Dijkstra's algorithm for determining shortest paths. Students have a tendency to solve shortest-path problems by inspection. Make sure they work through all steps of the algorithm.

The traveling salesman problem is a famous example of a problem for which no known algorithm can produce the optimum solution efficiently. It is worth covering if time permits.

Exercises: It may help to assign Exercise 4, which is difficult to do by inspection. I recommend that you have students work through some or all of Exercises 21–23, which deal with Floyd's algorithm for finding a shortest path between all pairs of vertices in a graph. Assigning exercises on the traveling salesman problem (such as Exercise 27) will convince students that the lack of an efficient algorithm for this problem is a serious issue.

SECTION 10.7 Planar Graphs

Goals: To introduce the concept of planarity of graphs and to develop tools to decide whether a graph is planar.

Prerequisites: Section 10.3 and its prerequisites.

Advice: The three house–three utility problem provides a good introduction to the concept of planarity; almost everyone finds this puzzle captivating. Sometimes students are confused that a graph can be planar even though the standard way of drawing it has crossings, such as K_4 or $K_{2,3}$. Cover Example 3, which gives an *ad hoc* proof that $K_{3,3}$ is not planar. Students often make the mistake of using Corollary 1, which states that a connected planar simple graph with e

edges and $v \geq 3$ vertices satisfies $e \leq 3v-6$, to show that graphs are planar. Cover Example 5, which shows that this cannot be done.

If you cover Kuratowski's Theorem, explain how to use it to show that graphs are nonplanar, such as is done in Example 8. Make sure that students understand how difficult it is to use this theorem to show that a graph is planar, however. Mention that planarity of graphs is an important topic for circuit design; this will perk the interest of electrical engineering students.

Exercises: Exercise 11 asks for a proof that K_5 is nonplanar, which can be constructed in a similar way to the proof given in the text for the nonplanarity of $K_{3,3}$. Exercise 16 asks for a proof that a connected bipartite planar simple graph satisfies $e \leq 2v-4$, where e is the number of edges and $v \geq 3$ is the number of vertices. The concepts of the crossing number and the thickness of a graph are introduced in the preambles to Exercises 26 and 30, respectively. These are important concepts for applications of graph theory to VLSI. Exercises 36–37 ask about drawing graphs on a torus without edges crossing.

SECTION 10.8 Graph Coloring

Goals: To introduce the concept of the coloring of a graph and give applications of graph colorings.

Prerequisites: Section 10.3 and its prerequisites.

Advice: Make sure students understand the connection between coloring maps and coloring graphs. Because students sometimes try to apply the Four Color Theorem to nonplanar graphs, show them that the chromatic number of a graph can be arbitrarily large by using K_n as an example. Also, show that bipartite graphs are the graphs with a chromatic number of at most 2. Sometimes students do not immediately see that the chromatic number of $K_{m,n}$ is 2. I suggest covering the application of graph coloring to scheduling final exams; because you may cover this section near the end of your term, finals may be near.

Exercises: You may want to have students study the coloring algorithm described in the preamble to Exercise 29; emphasize that this algorithm may use too many colors, and tell students that no one has found an efficient algorithm that colors graphs using the fewest possible colors. Exercises 40 and 41 ask for proofs of the Six Color Theorem and the Five Color Theorem, respectively. These proofs are considerable easier than the proof of the Four Color Theorem!

CHAPTER 11 Trees

Overview: Trees have a tremendous variety of applications. They are used extensively throughout computer science to build data structures, to perform encoding, to study formal languages, and in searching and sorting. This chapter introduces the terminology used to describe trees, their basic properties, and some of their important applications.

In Section 11.1 we cover basic terminology and establish relationships between the number of vertices of different types and the number of edges in trees. In Section 11.2 we introduce applications of trees to decision problems, to coding, to binary searching, and to studying games. We introduce tree traversal algorithms in Section 11.3; these are used extensively when compilers evaluate expressions. We discuss various ways to construct spanning trees, such as breadth-first searching and backtracking, and give applications of backtracking for solving a variety of problems in Section 11.4. Finally, in Section 11.5 we give algorithms for constructing minimum spanning trees. I recommend covering Section 11.1 in all courses. Cover the other sections that interest you as time permits.

SECTION 11.1 Introduction to Trees

Goals: To introduce the concept of a tree, to present basic terminology for trees, and to develop relationships among the number of vertices of different kinds and the number of edges in trees.

Prerequisites: Chapters 1–3 and Sections 4.1, 4.3, 5.1–5.4, 6.1, and 10.1–10.4.

Advice: A large amount of terminology is presented in this section, but almost all the terms are self-explanatory. We begin by defining unrooted trees; then we define rooted trees and show how an unrooted tree can be rooted by choosing any vertex as the root. You should show how additional structure can be placed on a rooted tree by ordering the children of each internal vertex. This is a good place to tell students that in many applications of binary trees each vertex except the root is specified to be either a right child or a left child (we will need this when discussing binary search trees). Go over one or two uses of trees in modeling, such as those discussed in Examples 5, 6, 7, and 8. These involve chemistry, business, and computer science.

The relationships between the numbers of vertices, internal vertices, and leaves of a full m-ary tree are straightforward. Explain that the equalities $n = mi + 1$ and $n = l + i$ can be used to solve for each of n, l, and i in terms of the other two. Students sometimes find the application of these equations difficult. It helps to go carefully through Example 9, which describes one such application to chain letters, emphasizing how the model is set up. Be sure to cover Theorem 5 and Corollary 1; they will be needed in our study of the complexity of sorting algorithms.

Exercises: I recommend Exercises 11–13, which ask how many nonisomorphic rooted trees there are with particular numbers of vertices. This is a question whose solution involves synthesizing different concepts, such as the definition of a rooted tree, the concept of isomorphism, and the usefulness of invariants. Exercise 48, which establishes a big-Omega estimate for the average depth of a leaf in a binary tree with n vertices, is needed later in the chapter when trees are used to study the complexity of sorting.

SECTION 11.2 Applications of Trees

Goals: To introduce several applications of trees, including binary search trees, decision trees, prefix codes, and game trees.

Prerequisites: Section 11.1 and its prerequisites.

Advice: We introduce four applications of trees in this section. We begin with binary search trees. Explain the algorithm for locating and adding items to a binary search tree with some examples, such as that given in the text. Go over the complexity analysis for this algorithm; mention that there are procedures for balancing binary search trees to make this algorithm as efficient as possible.

Decision trees are introduced here. Example 3, which shows how to use a decision tree to find a counterfeit coin, gives a good preview of how such trees are used. We show how to use decision trees to study the complexity of sorting algorithms; we provide big-Omega estimates for the worst-case and average-case complexity of sorting algorithms based on binary comparisons.

Next, we introduce the use of binary trees to represent prefix codes. Because students sometimes have trouble with the concept of a prefix code, go over the definition and examples carefully. We introduce Huffman coding for producing optimal prefix codes, a key technique of data compression. Example 5 illustrates how to use this algorithm.

Finally, we show how trees can be used to study games. We introduce the minmax strategy and illustrate how nim and tic-tac-toe are analyzed using game trees.

Exercises: Exercises 6–10 deal with finding a counterfeit coin among a set of coins, where it may not be known whether the counterfeit coin is lighter or heavier than the genuine coins. The tournament sort, which uses ordered binary trees, is studied in Exercises 13–18. Exercise 26 illustrates that there can be more than one Huffman code for the same set of symbols and frequencies, depending on how ties are broken. Exercise 27 asks for the construction of a Huffman code for the letters of English using their frequencies in typical English text. Exercise 31 develops a connection between the Fibonacci numbers and Huffman coding. Games trees are used to study nim and tic-tac-toe in Exercises 33–43.

SECTION 11.3 Tree Traversal

Goals: To introduce tree traversal algorithms and prefix and postfix notation.

Prerequisites: Chapters 1–3, 5.1–5.4, and 11.1–11.2.

Advice: Tree traversal is used extensively in computer science. In this section we present different traversal algorithms. First, I recommend performing these traversal algorithms directly from the recursive definitions, as illustrated in Examples 2, 3, and 4. Then explain how these traversals can be quickly performed by drawing a curve around the tree and listing vertices by how many times they are passed (see Figure 9). Show how expression trees are constructed. Define the infix, prefix, and postfix form of an expression, obtained by traversing this expression tree.

Exercises: Exercises 26–27 ask for proofs that expressions in postfix and prefix notation are well-defined. Well-formed formulae in prefix notation are introduced in the preamble to Exercise 30 and are studied in Exercises 30–31.

SECTION 11.4 Spanning Trees

Goals: To introduce the concept of a spanning tree, to give algorithms for constructing such trees, and to show how to solve problems using backtracking.

Prerequisites: Chapters 1–3, 5.1–5.4, and 11.1–11.2.

Advice: Explain that a simple graph may have many different spanning trees, as is demonstrated in Example 1. Show how to produce spanning trees by removing edges that form circuits. Then show students how to use breadth-first and depth-first searches to produce spanning trees. Define the notion of tree edges and back edges in spanning trees constructed using depth-first search; these concepts are illustrated in Example 4. Give an example to show how backtracking, an important algorithmic paradigm, can be used to solve a problem such as coloring a graph, placing n queens on a chessboard so that no queen can attack another, and finding elements of a subset having a specified sum, covered in Examples 6, 7, and 8, respectively. Introduce the notion of depth-first search in directed graphs and introduce the application of depth-first and breadth-first searching by Web spiders (Example 10).

Exercises: Exercise 30 gives another application of backtracking—the solution of mazes. Spanning forests are introduced in Exercises 31–33. You may want to cover rooted spanning trees; they are introduced in Exercises 57–59. The classification of the edges of a directed graph relative to a spanning tree constructed via depth-first search is introduced in Exercise 51.

SECTION 11.5 Minimum Spanning Trees

Goals: To study minimum spanning trees and produce algorithms for generating them.

Prerequisites: Section 11.4 and its prerequisites.

Advice: The construction of minimum spanning trees is required for many applications, so algorithms for their construction have been studied extensively. We present the two best known such algorithms: Prim's algorithm and Kruskal's algorithm. It is useful to go over both of them to illustrate that the same problem can be solved in different ways. Proving that Prim's algorithm produces a spanning tree is rather subtle, so explain the proof slowly.

Exercises: Exercises 11–15 are concerned with maximum spanning trees. Working through variants of Kruskal's algorithm and Prim's algorithm for maximum spanning trees helps students understand these algorithms. You may be interested in having students work through the development of Sollin's algorithm given in Exercises 24–31; this is an example of an algorithm designed for parallel processing. The reverse-delete algorithm for constructing minimum spanning trees (invented by Kruskal) is introduced in the preamble to Exercise 34 and is the subject of Exercises 34 and 35.

CHAPTER 12 Boolean Algebra

Overview: This chapter presents a brief introduction to Boolean algebra. We do not study Boolean algebra in a general setting, but rather we study the set $\{0,1\}^n$ and Boolean functions on this set. In Section 12.1 we prove some fundamental identities satisfied by Boolean functions. We show in Section 12.2 how to represent Boolean functions using sum-of-products expansions. We show that certain sets of operators are functionally complete, that is, can be used to represent all Boolean expressions. Next, in Section 12.3 we show how to build circuits to perform some simple tasks using the rules of Boolean algebra. Finally, in Section 12.4 we show how to minimize sum-of-products expansions using K-maps and the Quine–McCluskey method.

SECTION 12.1 Boolean Functions

Goals: To introduce Boolean functions and important identities involving these functions.

Prerequisites: Chapters 1 and 2 and Sections 5.1–5.4 and 6.1.

Advice: The distinction between Boolean expressions and functions is subtle. It is hard for students to see that there are 2^{2^n} Boolean functions of degree n. Example 7 shows that this is true. When discussing the identities of Boolean algebra, explain that these rules are analogous to those for propositional equivalences and to those for set identities. If you are so inclined, cover the subsection on the abstract definition of a Boolean algebra. You may also want to mention how lattices are used to define Boolean algebras.

Exercises: Exercise 11 asks for a proof of one of the absorption laws. Exercise 12 will be used to design a circuit in Example 2 of Section 12.3. The Boolean operator *XOR* is introduced in the preamble to Exercise 24 and studied in Exercises 24–27.

SECTION 12.2 Representing Boolean Functions

Goals: To represent Boolean functions with sum-of-products expansions and to find functionally complete sets of operators.

Prerequisites: Section 12.1 and its prerequisites.

Advice: This section shows how to represent Boolean functions using sum-of-products expansions. This presents students with no particular difficulties. We also discuss functional completeness in this section. We use De Morgan's laws to show that $\{+, \overline{\ }\}$ and $\{\cdot, \overline{\ }\}$ are functionally complete. We then show that there is a functionally complete set of just one operator, namely $\{\,|\,\}$.

Exercises: Exercises 7–11 develop another way to represent Boolean expressions, namely by product-of-sums expansions. Exercises 15–16 show that $\{\downarrow\}$ is functionally complete.

SECTION 12.3 Logic Gates

Goals: To introduce logic gates and to build circuits using these gates.

Prerequisites: Section 12.2 and its prerequisites.

Advice: Point out that there are different ways to draw the same circuit, as Figure 3 demonstrates. Go over one or both of Examples 2 and 3, which show how logic gates can be used to build voting circuits and light switch circuits. Be sure to go over half adders and full adders; this shows how logic gates can be used to do computer arithmetic.

Exercises: You may want to assign Exercises 10–11, which ask for circuits to perform subtraction. I suggest assigning some of Exercises 15–18, which deal with circuits built from NAND or NOR gates. Multiplexers are defined in the preamble to Exercise 19. You may want to assign Exercise 19 so that students have some exposure to this important element in circuit design.

SECTION 12.4 Minimization of Circuits

Goals: To simplify sum-of-products expansions using K-maps and the Quine–McCluskey method.

Prerequisites: Section 12.3 and its prerequisites.

Advice: This section presents procedures for producing the simplest possible Boolean sums of products to represent Boolean expressions. We first present K-maps for sum-of-products expressions in two, three, and four variables. Make sure students understand all the ways to combine terms by combining adjacent cells in K-maps. This is illustrated in Figures 6 and 9. Emphasize that the largest combinations of squares should be used first. Do Example 8, which illustrates *don't care* conditions, where certain cells in a K-map can be arbitrarily included or excluded. Finally, go over the Quine–McCluskey method. Emphasize that it is an algorithm that can be mechanized, as opposed to the technique of K-maps, which depends on visual inspection, and that more than one final answer is possible, depending on how the covering of terms is produced. This covering is the difficult part of the procedure; it can be done using backtracking.

Exercises: You may wish to assign Exercise 20, which involves *don't care* conditions. The simplification of product-of-sums expansions using K-maps is the subject of Exercise 26.

CHAPTER 13
Modeling Computation

Overview: This chapter is an introduction to the theory of computation. Phrase-structure grammars and finite-state automata are introduced and the connection between them demonstrated. There are several reasons for covering the topics of this chapter. First, they illustrate how discrete mathematics and discrete structures are used in computer science. Second, they introduce students to a fascinating area that is worthy of extensive further study. And third, they form an attractive set of topics, leading to one central result: Kleene's Theorem. This chapter concludes with a section on Turing machines, which provide a model for essentially all computing machines. This chapter, including this last section, provides a gateway to further studies in the theory of computation.

SECTION 13.1 Languages and Grammars

Goals: To introduce phrase-structure grammars and the Chomsky classification of these grammars. To introduce Backus–Naur form.

Prerequisites: Chapters 1 and 2 and Sections 4.1–4.2, 5.1–5.3, and 11.1.

Advice: We introduce formal languages and grammars through the use of an example that involves a subset of English. This helps motivate the more abstract definitions that follow. It is worthwhile to explain the difference between a natural language, like English, and a formal language, and to mention that this material is used in linguistics and in computer science (especially in the area of compilers).

Students have trouble distinguishing between different types of phrase-structure grammars. Be sure to cover Examples 8, 9, and 10 in the text; they provide useful illustrations of different types of grammars, but more importantly, they will be used later in the chapter.

Backus–Naur form is important in specifying languages. Illustrate its use with Example 13, which shows how it was used to define an identifier in ALGOL 60. Example 15, which shows how to use Backus–Naur form to describe the signed integers, is also worth covering.

Exercises: Exercise 19 gives students the opportunity to distinguish between different types of grammars. Exercises 28–33 involve Backus–Naur form, and extended Backus–Naur form is defined and used in Exercises 34–36.

SECTION 13.2 Finite-State Machines with Output

Goals: To introduce the concept of a finite-state machine with output and to model different types of machines using them.

Prerequisites: Chapters 1 and 2 and Sections 5.1–5.3 and 10.1–10.3.

Advice: We discuss finite-state machines with output in this section. The example of a vending machine described in this section helps students understand the different elements that make up a finite-state machine. The finite-state machines we discuss in the text are known as Mealy machines; they have an output associated with each transition. Moore machines, a variant of finite-state machines with output, have an output associated with each state; these are covered in the exercise set.

Make sure that students understand how a Mealy machine produces an output string from an input string. Examples 5, 6, and 7 provide other good illustrations of Mealy machines.

Example 7 is also an example of a language recognizer, a topic we will develop further for finite-state machines without output in the next two sections of the chapter.

Exercises: Exercises 20–25 deal with Moore machines. Good discussion questions you can raise for students are: When is it easier to use a Moore machine and when is it easier to use a Mealy machine? What would it mean for two finite-state machines, which may be a Mealy machine and a Moore machine, to be equivalent? How can equivalent Mealy and Moore machines be constructed?

SECTION 13.3 Finite-State Machines with No Output

Goals: To show how to perform operations with sets of strings, to introduce deterministic and nondeterministic finite-state automata, and to show how they recognize sets of strings.

Prerequisites: Sections 13.1 and 13.2 and their prerequisites.

Advice: Make sure that students understand how to form concatenations of sets of strings, as is done in Example 1, and Kleene closures of sets of strings, as is done in Example 3. Students confuse the empty string λ, the set $\{\lambda\}$ containing the empty string, the empty set $\emptyset$, and the set $\{\emptyset\}$ containing the empty set, so make sure the distinction between these objects is clear.

Next we discuss finite-state machines with no output. We begin with deterministic finite-state automata, where the transition from each state on each input is well-defined. Later in the section we introduce nondeterministic finite-state automata, where there can be any number, including zero, of possible transitions from a state on an input. Make sure that students have a clear understanding of what it means for a deterministic automaton to recognize a string and what the language recognized by a machine is. Example 5 illustrates this. You may want to cover Examples 6 and 7, which show how to design finite-state automata that recognize a specified set of strings. Students have trouble understanding what it means for a nondeterministic automaton to recognize a string, so you may want to devote extra time to Example 11.

Stress the constructive aspect of Theorem 1, which shows how to find a deterministic finite automaton that recognizes the same language as a given nondeterministic finite automaton. Example 12 illustrates this procedure.

Exercises: Assign some of Exercises 1–8 so that students get some practice working with concatenations and Kleene closures of sets of strings. Exercise 39 provides a useful tool for constructing a finite-state automaton recognizing the complement of the language recognized by a given finite-state automaton. Minimization of finite-state automata is developed in Exercises 58–62.

SECTION 13.4 Language Recognition

Goals: To define regular sets, to show that they are the sets recognized by finite-state automata, and to show that they are the languages generated by regular grammars.

Prerequisites: Section 13.3 and its prerequisites.

Advice: The difference between regular expressions and the regular sets that they represent is subtle; I suggest going over Example 1 carefully. Example 2 illustrates how regular expressions can be found that represent a given set of strings. Kleene's Theorem is the central result of the chapter. It states that regular sets, that is, those sets represented by regular expressions, are precisely those sets recognized by finite-state automata. The proof of Kleene's Theorem is complicated, but it is modular, so that you can, if so desired, cover only some of the parts in detail or split the proof into manageable pieces. Example 3 illustrates the constructive nature of this proof, but obviously produces an overly complicated automaton, as is shown in Figure 3.

We also show that regular sets are precisely those sets that are generated by regular grammars. This tells us that a language is generated by a regular grammar if and only if it is recognized by a finite-state automaton. Example 6 shows that nonregular sets exist; it illustrates the idea behind the pumping lemma (see Exercise 22).

This section concludes with brief hints at more powerful types of machines, such as push-down automata and Turing machines (covered in Section 13.5). Students who have developed an interest in the theory of computation will look forward to learning more about these machines in their subsequent studies.

Exercises: Exercise 22 formally introduces the pumping lemma, generalizing the idea used in Example 5. Exercises 27–31 develop the ideas needed to show that the set of palindromes is not regular.

SECTION 13.5 Turing Machines

Goals: To define Turing machines, to explain how Turing machines can be used to recognize languages, to describe how Turing machines can be used to compute number-theoretic functions, to introduce the Church–Turing thesis, and finally to show how Turing machines are used in the study of the solvability and complexity of problems.

Prerequisites: Section 13.4 and its prerequisites.

Advice: This section presents the definition of a Turing machine. The definition given here is perhaps the simplest and the least technical of many equivalent definitions of Turing machines. The important point to make is that the capabilities of Turing machines are essentially independent of the way they are defined. When Turing machines are studied in greater depth, different variations of Turing machines are employed to carry out particular tasks and these variations are shown to have equivalent capabilities.

Next, this section covers the use of Turing machines to recognize strings. There are also various ways to define what it means for a Turing machine to recognize a string. Again, we introduce perhaps the simplest, and least technical, way. However, it is important to stress that this is one of many possible ways to do this.

Then the section shows how to use Turing machines to compute number-theoretic functions. The example included is extremely simple; building Turing machines to compute more complicated number-theoretic functions can be complex and technical.

The section includes a discussion of the Church–Turing thesis, which plays a key role in the theory of computation. Essentially it states that Turing machines can simulate any computations that a computer can perform. Finally, the section includes a discussion of Turing machines and computational complexity, computability, and decidability, providing a lead-in to more advanced courses in theoretical computer science. And that is a good place to end the text.

Exercises: The busy beaver problem is described in the preamble to Exercise 31. Starred Exercise 31, which involves the busy beaver problem for $n = 2$, should prove interesting for students who want a computational challenge.

Appendixes

APPENDIX 1 Axioms for the Real Numbers and the Positive Integers

Goals: To introduce the basic axioms for the set of real numbers and for the set for positive integers and to show how to use these axioms to prove simple facts. Providing these axioms and some of their simple consequences makes explicit the assumptions allowed in proofs throughout the text.

Prerequisites: None.

Advice: Making explicit the axioms for real numbers and for the positive integers helps clarify what can be assumed in the proofs in the text. You might want to explain that the axioms presented here are often used as a basis for constructing the set of real numbers. Go over the proofs of some of the theorems to show how these axioms can be used to prove basic results about the real numbers. You may want to tell your students that either the well-ordering property or the principle of mathematical induction can be taken as an axiom, with the other proved as a theorem.

Exercises: Assign Exercises 1 and 2, which ask for proofs that the multiplicative identity element is unique and that the inverse of every nonzero real number is unique. Exercise 15 asks for a proof that the sense of an inequality is reversed when both sides are multiplied by a negative number.

APPENDIX 2 Exponential and Logarithmic Functions

Goals: To review basic properties of logarithmic and exponential functions.

Prerequisites: None.

Advice: Most students will have some facility working with logarithmic and exponential functions, but others will need a brief review. Make sure that students are aware of the result given in Theorem 3, which shows how to convert bases in logarithms. Note that in the text "$\log x$" always refers to the base 2 logarithm of x.

Exercises: Exercise 4 should be assigned because it contains a result used in big-O estimates of the complexity of divide-and-conquer algorithms.

APPENDIX 3 Pseudocode

Goals: To introduce the rules for the pseudocode used in the text.

Prerequisites: None.

Advice: This appendix is designed as a reference for the pseudocode used in the text. Point out that pseudocode is designed to be flexible and serves as a compromise between English and a programming language.

Exercises: Exercise 3 illustrates that we could make do with a smaller set of commands, because **for** loops can be simulated by **while** loops.