Preface

The most valuable acquisitions in a scientific or technical education are the general-purpose mental tools which remain serviceable for a lifetime.
—George Forsythe, “What to do till the computer scientist comes.” (1968)

Algorithms play the central role both in the science and practice of computing. Recognition of this fact has led to the appearance of a considerable number of textbooks on the subject. By and large, they follow one of two alternatives in presenting algorithms. One classifies algorithms according to a problem type. Such a book would have separate chapters on algorithms for sorting, searching, graphs, and so on. The advantage of this approach is that it allows an immediate comparison of, say, the efficiency of different algorithms for the same problem. The drawback of this approach is that it emphasizes problem types at the expense of algorithm design techniques.

The second alternative organizes the presentation around algorithm design techniques. In this organization, algorithms from different areas of computing are grouped together if they have the same design approach. I share the belief of many (e.g., [BaY95]) that this organization is more appropriate for a basic course on the design and analysis of algorithms. There are three principal reasons for emphasis on algorithm design techniques. First, these techniques provide a student with tools for designing algorithms for new problems. This makes learning algorithm design techniques a very valuable endeavor from a practical standpoint. Second, they seek to classify multitudes of known algorithms according to an underlying design idea. Learning to see such commonality among algorithms from different application areas should be a major goal of computer science education. After all, every science considers classification of its principal subject as a major if not the central point of its discipline. Third, in my opinion, algorithm design techniques have utility as general problem solving strategies, applicable to problems beyond computing.
Unfortunately, the traditional classification of algorithm design techniques has several serious shortcomings, from both theoretical and educational points of view. The most significant of these shortcomings is the failure to classify many important algorithms. This limitation has forced the authors of other textbooks to depart from the design technique organization and to include chapters dealing with specific problem types. Such a switch leads to a loss of course coherence and almost unavoidably creates a confusion in students' minds.

**New taxonomy of algorithm design techniques**

My frustration with the shortcomings of the traditional classification of algorithm design techniques has motivated me to develop a new taxonomy of them [Lev99], which is the basis of this book. Here are the principal advantages of the new taxonomy:

- The new taxonomy is more comprehensive than the traditional one. It includes several strategies—brute-force, decrease-and-conquer, transform-and-conquer, space and time trade-offs, and iterative improvement—that are rarely if ever recognized as important design paradigms.
- The new taxonomy covers naturally many classic algorithms (Euclid’s algorithm, heapsort, search trees, hashing, topological sorting, Gaussian elimination, Horner’s rule—to name a few) that the traditional taxonomy cannot classify. As a result, the new taxonomy makes it possible to present the standard body of classic algorithms in a unified and coherent fashion.
- It naturally accommodates the existence of important varieties of several design techniques. For example, it recognizes three variations of decrease-and-conquer and three variations of transform-and-conquer.
- It is better aligned with analytical methods for the efficiency analysis (see Appendix B).

**Design techniques as general problem solving strategies**

Most applications of the design techniques in the book are to classic problems of computer science. (The only innovation here is an inclusion of some material on numerical algorithms, which are covered within the same general framework.) But these design techniques can be considered general problem solving tools, whose applications are not limited to traditional computing and mathematical problems. Two factors make this point particularly important. First, more and more computing applications go beyond the traditional domain, and there are reasons to believe that this trend will strengthen in the future. Second, developing students’ problem solving skills has come to be recognized as a major goal of college education. Among all the courses in a computer science curriculum, a course on the design and analysis of algorithms is uniquely suitable for this task because it can offer a student specific strategies for solving problems.

I am not proposing that a course on the design and analysis of algorithms should become a course on general problem solving. But I do believe that the
unique opportunity provided by studying the design and analysis of algorithms should not be missed. Toward this goal, the book includes applications to puzzles and puzzle-like games. Although using puzzles in teaching algorithms is certainly not a new idea, the book tries to do this systematically by going well beyond a few standard examples.

**Textbook pedagogy**

My goal was to write a text that would not trivialize the subject but would still be readable by most students on their own. Here are some of the things done toward this objective.

- Sharing the opinion of George Forsythe expressed in the epigraph, I have sought to stress major ideas underlying the design and analysis of algorithms. In choosing specific algorithms to illustrate these ideas, I limited the number of covered algorithms to those that demonstrate an underlying design technique or an analysis method most clearly. Fortunately, most classic algorithms satisfy this criterion.

- In Chapter 2, which is devoted to efficiency analysis, the methods used for analyzing nonrecursive algorithms are separated from those typically used for analyzing recursive algorithms. The chapter also includes sections devoted to empirical analysis and algorithm visualization.

- The narrative is systematically interrupted by questions to the reader. Some of them are asked rhetorically, in anticipation of a concern or doubt, and are answered immediately. The goal of the others is to prevent the reader from drifting through the text without a satisfactory level of comprehension.

- Each chapter ends with a summary recapping the most important concepts and results discussed in the chapter.

- The book contains over 600 exercises. Some of them are drills; others make important points about the material covered in the body of the text or introduce algorithms not covered there at all. A few exercises take advantage of Internet resources. More difficult problems—there are not many of them—are marked by special symbols in the Instructor’s Manual. (Because marking problems as difficult may discourage some students from trying to tackle them, problems are not marked in the book itself.) Puzzles, games, and puzzle-like questions are marked in the exercises with a special icon.

- The book provides hints to all the exercises. Detailed solutions, except for programming projects, are provided in the Instructor’s Manual, available to qualified adopters through Pearson’s Instructor Resource Center. (Please contact your local Pearson sales representative or go to www.pearsonhighered.com/irc to access this material.) Slides in PowerPoint are available to all readers of this book via anonymous ftp at the CS Support site: http://cssupport.pearsoncmg.com/.
Changes for the third edition

There are a few changes in the third edition. The most important is the new order of the chapters on decrease-and-conquer and divide-and-conquer. There are several advantages in introducing decrease-and-conquer before divide-and-conquer:

- Decrease-and-conquer is a simpler strategy than divide-and-conquer.
- Decrease-and-conquer is applicable to more problems than divide-and-conquer.
- The new order makes it possible to discuss insertion sort before mergesort and quicksort.
- The idea of array partitioning is now introduced in conjunction with the selection problem. I took advantage of an opportunity to do this via the one-directional scan employed by Lomuto’s algorithm, leaving the two-directional scan used by Hoare’s partitioning to a later discussion in conjunction with quicksort.
- Binary search is now considered in the section devoted to decrease-by-a-constant-factor algorithms, where it belongs.

The second important change is restructuring of Chapter 8 on dynamic programming. Specifically:

- The introductory section is completely new. It contains three basic examples that provide a much better introduction to this important technique than computing a binomial coefficient, the example used in the first two editions.
- All the exercises for Section 8.1 are new as well; they include well-known applications not available in the previous editions.
- I also changed the order of the other sections in this chapter to get a smoother progression from the simpler applications to the more advanced ones.

The other changes include the following. More applications of the algorithms discussed are included. The section on the graph-traversal algorithms is moved from the decrease-and-conquer chapter to the brute-force and exhaustive-search chapter, where it fits better, in my opinion. The Gray code algorithm is added to the section dealing with algorithms for generating combinatorial objects. The divide-and-conquer algorithm for the closest-pair problem is discussed in more detail. Updates include the section on algorithm visualization, approximation algorithms for the traveling salesman problem, and, of course, the bibliography.

I also added about 70 new problems to the exercises. Some of them are algorithmic puzzles and questions asked during job interviews.

Prerequisites

The book assumes that a reader has gone through an introductory programming course and a standard course on discrete structures. With such a background, he or she should be able to handle the book’s material without undue difficulty.
Still, fundamental data structures, necessary summation formulas, and recurrence relations are reviewed in Section 1.4, Appendix A, and Appendix B, respectively. Calculus is used in only three sections (Section 2.2, 11.4, and 12.4), and to a very limited degree; if students lack calculus as an assured part of their background, the relevant portions of these three sections can be omitted without hindering their understanding of the rest of the material.

Use in the curriculum

The book can serve as a textbook for a basic course on design and analysis of algorithms organized around algorithm design techniques. It might contain slightly more material than can be covered in a typical one-semester course. By and large, portions of Chapters 3 through 12 can be skipped without the danger of making later parts of the book incomprehensible to the reader. Any portion of the book can be assigned for self-study. In particular, Sections 2.6 and 2.7 on empirical analysis and algorithm visualization, respectively, can be assigned in conjunction with projects.

Here is a possible plan for a one-semester course; it assumes a 40-class meeting format.

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1.1–1.3</td>
</tr>
<tr>
<td>2, 3</td>
<td>Analysis framework; $O$, $\Omega$, $\Theta$ notations</td>
<td>2.1, 2.2</td>
</tr>
<tr>
<td>4</td>
<td>Mathematical analysis of nonrecursive algorithms</td>
<td>2.3</td>
</tr>
<tr>
<td>5, 6</td>
<td>Mathematical analysis of recursive algorithms</td>
<td>2.4, 2.5 (+ App. B)</td>
</tr>
<tr>
<td>7</td>
<td>Brute-force algorithms</td>
<td>3.1, 3.2 (+ 3.3)</td>
</tr>
<tr>
<td>8</td>
<td>Exhaustive search</td>
<td>3.4</td>
</tr>
<tr>
<td>9</td>
<td>Depth-first search and breadth-first search</td>
<td>3.5</td>
</tr>
<tr>
<td>10, 11</td>
<td>Decrease-by-one: insertion sort, topological sorting</td>
<td>4.1, 4.2</td>
</tr>
<tr>
<td>12</td>
<td>Binary search and other decrease-by-a-constant-factor algorithms</td>
<td>4.4</td>
</tr>
<tr>
<td>13</td>
<td>Variable-size-decrease algorithms</td>
<td>4.5</td>
</tr>
<tr>
<td>14, 15</td>
<td>Divide-and-conquer: mergesort, quicksort</td>
<td>5.1–5.2</td>
</tr>
<tr>
<td>16</td>
<td>Other divide-and-conquer examples</td>
<td>5.3 or 5.4 or 5.5</td>
</tr>
<tr>
<td>17–19</td>
<td>Instance simplification: presorting, Gaussian elimination, balanced search trees</td>
<td>6.1–6.3</td>
</tr>
<tr>
<td>20</td>
<td>Representation change: heaps and heapsort or Horner's rule and binary exponentiation</td>
<td>6.4 or 6.5</td>
</tr>
<tr>
<td>21</td>
<td>Problem reduction</td>
<td>6.6</td>
</tr>
<tr>
<td>22–24</td>
<td>Space-time trade-offs: string matching, hashing, B-trees</td>
<td>7.2–7.4</td>
</tr>
<tr>
<td>25–27</td>
<td>Dynamic programming algorithms</td>
<td>3 from 8.1–8.4</td>
</tr>
</tbody>
</table>
Acknowledgments

I would like to express my gratitude to the reviewers and many readers who have shared with me their opinions about the first two editions of the book and suggested improvements and corrections. The third edition has certainly benefited from the reviews by Andrew Harrington (Loyola University Chicago), David Levine (Saint Bonaventure University), Stefano Lombardi (UC Riverside), Daniel McKee (Mansfield University), Susan Brilliant (Virginia Commonwealth University), David Akers (University of Puget Sound), and two anonymous reviewers.

My thanks go to all the people at Pearson and their associates who worked on my book. I am especially grateful to my editor, Matt Goldstein; the editorial assistant, Chelsea Bell; the marketing manager, Yez Alayan; and the production supervisor, Kayla Smith-Tarbox. I am also grateful to Richard Camp for copyediting the book, Paul Anagnostopoulos of Windfall Software and Jacqui Scarlott for its project management and typesetting, and MaryEllen Oliver for proofreading the book.

Finally, I am indebted to two members of my family. Living with a spouse writing a book is probably more trying than doing the actual writing. My wife, Maria, lived through several years of this, helping me any way she could. And help she did: over 400 figures in the book and the Instructor’s Manual were created by her. My daughter Miriam has been my English prose guru over many years. She read large portions of the book and was instrumental in finding the chapter epigraphs.

Anany Levitin
anany.levitin@villanova.edu
June 2011