Preface

This fifth edition of *Foundations of Algorithms* retains the features that made the previous editions successful. As in those editions, I still use pseudocode and not actual C++ code. The presentation of complex algorithms using all the details of any programming language would only cloud the students' understanding of the algorithms. Furthermore, the pseudocode should be understandable to someone versed in any high-level language, which means it should avoid details specific to any one language as much as possible. Significant deviations from C++ are discussed on pages 5–7 of the text. This text is about designing algorithms, complexity analysis of algorithms, and computational complexity (analysis of problems). It does not cover other types of analyses, such as analysis of correctness. My motivation for writing this book was my inability to find a text that rigorously discusses complexity analysis of algorithms, yet is accessible to computer science students at mainstream universities such as Northeastern Illinois University. The majority of Northeastern's students have not studied calculus, which means that they are not comfortable with abstract mathematics and mathematical notation. The existing texts that I know of use notation that is fine for a mathematically sophisticated student, but is a bit terse for Northeastern's student body.

To make this text more accessible, I do the following:

- assume that the student's mathematics background includes only college algebra and discrete structures;
- use more English description than is ordinarily used to explain mathematical concepts;
- give more detail in formal proofs than is usually done;
- provide many examples.

This text is targeted to a one-semester upper-level undergraduate or graduate course in the design and analysis of algorithms. It is intended to provide students with a basic of understanding of how to write and analyze

algorithms and to impart to them the skills needed to write algorithms using the standard algorithm design strategies. Previously, these included divide-and-conquer, dynamic programming, the greedy approach, backtracking, and branch-and-bound. However, in recent years the use of genetic algorithms has become increasingly important to the computer scientist. Yet a student would only be introduced to such algorithms if the student took a course related to artificial intelligence. There is nothing inherent in genetic algorithms that relegates them to the domain of artificial intelligence. So, to better provide a repertoire of current useful techniques, I have added a chapter on genetic algorithms and genetic programming in this edition.

Because the vast majority of complexity analysis requires only a knowledge of finite mathematics, in most of the discussions I am able to assume only a background in college algebra and discrete structures. That is, for the most part, I do not find it necessary to rely on any concepts learned only in a calculus course. Often students without a calculus background are not yet comfortable with mathematical notation. Therefore, wherever possible, I introduce mathematical concepts (such as "big O") using more English description and less notation than is ordinarily used. It is no mean task finding the right mix of these two; a certain amount of notation is necessary to make a presentation lucid, whereas too much vexes many students. Judging from students' responses, I have found a good mix.

This is not to say that I cheat on mathematical rigor. I provide formal proofs for all results. However, I give more detail in the presentation of these proofs than is usually done, and I provide a great number of examples. By seeing concrete cases, students can often more easily grasp a theoretical concept. Therefore, if students who do not have strong mathematical backgrounds are willing to put forth sufficient effort, they should be able to follow the mathematical arguments and thereby gain a deeper grasp of the subject matter. Furthermore, I do include material that requires knowledge of calculus (such as the use of limits to determine order and proofs of some theorems). However, students do not need to master this material to understand the rest of the text. Material that requires calculus is marked with a  symbol in the table of contents and in the margin of the text; material that is inherently more difficult than most of the text but that requires no extra mathematical background is marked with a  symbol.

Prerequisites

As mentioned previously, I assume that the student's background in mathematics includes only college algebra and finite mathematics. The actual mathematics that is required is reviewed in Appendix A. For computer science background, I assume that the student has taken a data structures course. Therefore, material that typically appears in a data structures text is not presented here.

Chapter Contents

For the most part, I have organized this text by technique used to solve problems, rather than by application area. I feel that this organization makes the field of algorithm design and analysis appear more coherent. Furthermore, students can more readily establish a repertoire of techniques that they can investigate as possible ways to solve a new problem. The chapter contents are as follows:

- **Chapter 1** is an introduction to the design and analysis of algorithms. It includes both an intuitive and formal introduction to the concept of order.
- **Chapter 2** covers the divide-and-conquer approach to designing algorithms.
- **Chapter 3** presents the dynamic programming design method. I discuss when dynamic programming should be used instead of divide-and-conquer.
- **Chapter 4** discusses the greedy approach and ends with a comparison of the dynamic programming and greedy approaches to solving optimization problems.
- **Chapters 5** and **6** cover backtracking and branch-and-bound algorithms respectively.
- In **Chapter 7** I switch from analyzing algorithms to computational complexity, which is the analysis of problems. I introduce computation complexity by analyzing the Sorting Problem. I chose that problem because of its importance, because there are such a large variety of sorting algorithms, and, most significantly, because there are sorting algorithms that perform about as well as the lower bound for the Sorting Problem (as far as algorithms that sort only by comparisons of keys). After comparing sorting algorithms, I analyze the problem of sorting by comparisons of keys. The chapter ends with Radix Sort, which is a sorting algorithm that does not sort by comparison keys.
- In **Chapter 8** I further illustrate computational complexity by analyzing the Searching Problem. I analyze both the problem of searching for a key in a list and the Selection Problem, which is the problem of finding the kth-smallest key in a list.
- **Chapter 9** is devoted to intractability and the theory of *NP*. To keep this text accessible yet rigorous, I give a more complete discussion of this material than is usually given in an algorithms text. I start out by explicitly drawing the distinction between problems for which polynomial-time algorithms have been found, problems that have been proven to be intractable, and problems that have not been proven to be intractable but for which polynomial-time algorithms have never been found. I then discuss the sets *P* and *NP*, *NP*-complete problems, and *NP*-equivalent problems. I have found that students are often left confused if they do not explicitly see the relationships among these sets. I end the chapter with a discussion of approximation algorithms.

- **Chapter 10** covers genetic algorithms and genetic programming. I provide both the theory and practical applications such as a financial trading algorithm.
- **Chapter 11** covers number-theoretic algorithms, including Euclid's algorithm, and the new polynomial-time algorithm for determining whether a number is prime.
- **Chapter 12** covers an introduction to parallel algorithms, including parallel architectures and the PRAM model.
- **Appendix A** reviews the mathematics that is necessary for understanding the text.
- **Appendix B** covers techniques for solving recurrences. The results in Appendix B are used in our analyses of divide-and-conquer algorithms in Chapter 2.
- **Appendix C** presents a disjoint set data structure that is needed to implement two algorithms in Chapter 4.

Pedagogy

To motivate the student, I begin each chapter with a story that relates to the material in the chapter. In addition, I use many examples and end the chapters with ample exercises, which are grouped by section. Following the section exercises are supplementary exercises that are often more challenging.

To show that there is more than one way to attack a problem, I solve some problems using more than one technique. For example, I solve the Traveling Salesperson Problem using dynamic programming, branch-and-bound, and an approximation algorithm. I solve the 0-1 Knapsack Problem using dynamic programming, backtracking, and branch-and-bound. To further integrate the material, I present a theme that spans several chapters, concerning a salesperson named Nancy who is looking for an optimal tour for her sales route.

Course Outlines

As mentioned previously this text is intended for an upper-level undergraduate or graduate course in algorithms.

In a one semester course I recommend covering the following material in this order:

Chapter 1: All
Appendix B: Sections B.1, B.3
Chapter 2: Sections 2.1–2.5, 2.8
Chapter 3: Sections 3.1–3.4, 3.6
Chapter 4: Sections 4.1, 4.2, 4.4
Chapter 5: Sections 5.1, 5.2, 5.4, 5.6, 5.7
Chapter 6: Sections 6.1, 6.2

Chapter 7: Sections 7.1–7.5, 7.7, 7.8.1, 7.8.2, 7.9
Chapter 8: Sections 8.1.1, 8.5.1, 8.5.2
Chapter 9: Sections 9.1–9.4
Chapter 10: Sections 10.1–10.3.2

Chapters 2-6 contain several sections, each solving a problem using the design method presented in the chapter. I cover the ones of most interest to us, but you are free to choose any of the sections.

You may not be able to cover any of Chapters 11 and 12. However, the material in Chapter 12 is quite accessible once students have studies the first ten chapters. Students with a solid mathematics background, such as that obtained by studying calculus, should be able to read Chapter 11 on their own.

Instructor Resources

An Instructor's Manual, PowerPoint presentations, and complete solutions manual are available for qualified instructors. Jones & Bartlett Learning reserves the right to evaluate all requests.

Acknowledgments

I would like to thank all those individuals who read the original manuscript and provided many useful suggestions. In particular, I thank my colleagues William Bultman, Jack Hade, Mary and Jim Kenevan, Stuart Kurtz, Don La Budde, and Miguel Vian, all of whom quite readily and thoroughly reviewed whatever was asked of them. I further thank the academic and professional peer reviewers, who made this a far better text through their insightful critiques. Many of them certainly did a much more thorough job than we would have expected. They include David D. Berry, Xavier University; David W. Boyd, Valdosta State University; Vladimir Drobot, San Jose State University; Dan Hirschberg, University of California at Irvine; Xia Jiang, Northeastern Illinois University; Raghu Karinthi, West Virginia University; Peter Kimmel, Northeastern Illinois University; C. Donald La Budde, Northeastern Illinois University; Y. Daniel Liang, Indiana Purdue University at Fort Wayne; David Magagnosc, Drexel University; Robert J. McGlinn, Southern Illinois University at Carbondale; Laurie C. Murphy, University of Mississippi; Paul D. Phillips, Mount Mercy College; H. Norton Riley, California State Polytechnic University, Pomona; Majid Sarrafzadeh, Northwestern University; Cliff Shaffer, Virginia Polytechnical Institute and State University; Nancy Van Cleave, Texas Tech University; and William L. Ziegler, State University of New York, Binghamton. Finally, I would like to thank Taylor and Francis, in particular Randi Cohen, for allowing me to include material from my 2012 text *Contemporary Artificial Intelligence* in this text's new Chapter 10, titled Genetic Algorithms and Genetic Programming.

Errors

There are sure to be some errors in an endeavor of this magnitude. If you find any errors or have any suggestions for improvements, I would certainly like to hear from you. Please send your comments to Rich Neapolitan. Email: RE-Neapolitan@neiu.edu. Thanks.

R. N.