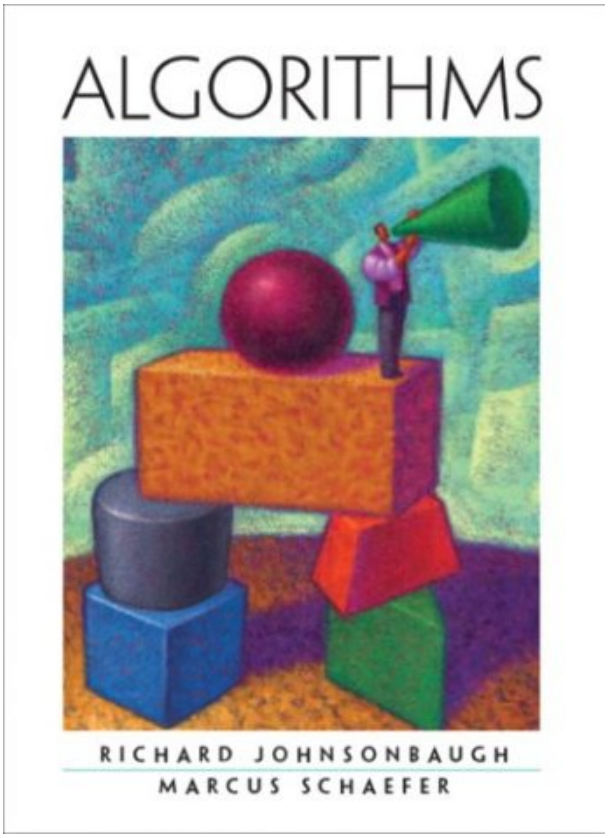
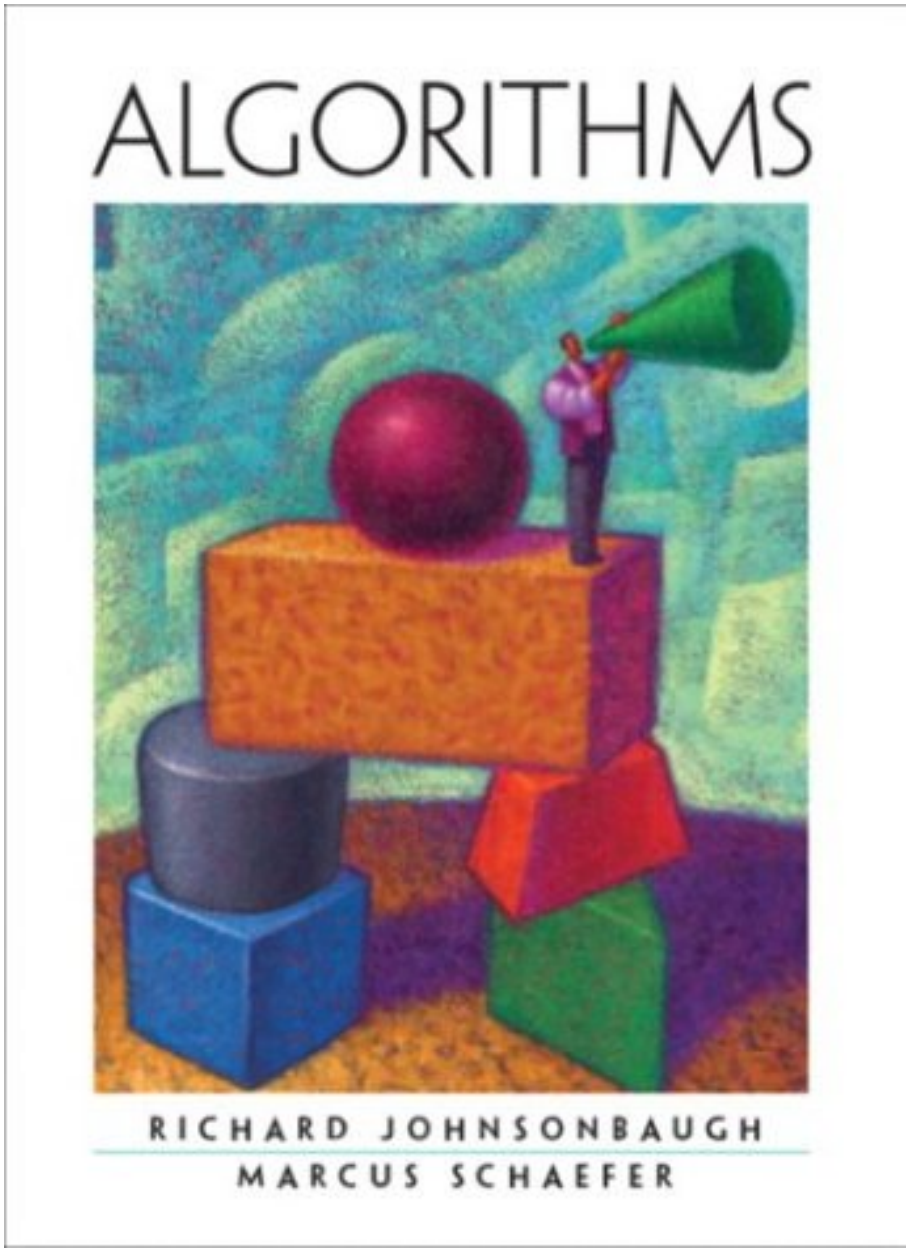
ALGORITHMS BY RICHARD JOHNSONBAUGH, MARCUS SCHAEFER





ALGORITHMS BY RICHARD JOHNSONBAUGH, MARCUS SCHAEFER PDF

You could save the soft documents of this publication **Algorithms By Richard Johnsonbaugh, Marcus Schaefer** It will depend on your spare time and also activities to open and read this book Algorithms By Richard Johnsonbaugh, Marcus Schaefer soft documents. So, you could not hesitate to bring this book Algorithms By Richard Johnsonbaugh, Marcus Schaefer all over you go. Simply include this sot file to your gadget or computer system disk to allow you check out whenever as well as almost everywhere you have time.

From the Back Cover

Algorithms is written for an introductory upper-level undergraduate or graduate course in algorithms. With/their many years of experience in teaching algorithms courses, Richard Johnsonbaugh and Marcus Schaefer include applications of algorithms, examples, end-of-section exercises, end-of-chapter exercises, solutions to selected exercises, and notes to help the reader understand and master algorithms.

Key Features

- Links theory to real-world applications such as data compression, region-finding in digital pictures, cellular phone networks, and the implementation of agrep.
- Includes five chapters that emphasize design techniques: searching (including backtracking), divide and conquer, sorting, selection, the greedy method, and dynamic programming.
- Covers distributed algorithms—a topic recommended by the ACM (2001 report) for an undergraduate curriculum.
- Features a collection of techniques, including approximation, parameterization (a recent area of research), and use of heuristics, to deal with NP-complete problems.
- Contains more than 1450 carefully developed and classroom-tested exercises, from routine to challenging. About one-third of the end-of-section exercises include solutions.
- Provides a robust Companion Website that supplements the text by providing algorithm simulation software, PowerPoint® slides, late breaking news about algorithms, references about the book's topics, computer programs, and more.
- Includes more than 300 worked examples, which provide motivation, clarify concepts, and show how to develop algorithms, demonstrate applications of the theory, and elucidate proofs.

About the Author

Richard Johnsonbaugh is Professor Emeritus of Computer Science at DePaul University. He has degrees in computer science and mathematics from the University of Oregon, Yale University, and the University of Illinois at Chicago. He is the author of numerous articles and books, including Discrete Mathematics, Fifth Edition, and, with co-author Martin Kalin, Object-Oriented Programming in C++, Second Edition,

Applications Programming in C++, and Applications Programming in ANSI C, Third Edition.

Marcus Schaefer is Assistant Professor of Computer Science at DePaul University. He holds degrees in computer science and mathematics from the University of Chicago and the Universitat Karlsruhe. He has authored and co-authored several articles on complexity theory, computability, and graph theory.

Why We Wrote This Book

Intended for an upper-level undergraduate or graduate course in algorithms, this book is based on our combined 25 years of experience in teaching this course. Our major goals in writing this book were to

- Emphasize design techniques.
- Show that algorithms are fun and exciting.
- Include real-world applications.
- Provide numerous worked examples and exercises.

Faced with a new computational problem, a designer will often be able to solve it by using one of the algorithms in this book, perhaps after modifying or adapting it slightly. However, some problems cannot be solved by any of the algorithms in this book. For this reason, we present a repertoire of design techniques that can be used to solve the problem and help the reader to develop intuition about which techniques are likely to succeed. The chapters on NP-completeness and how to deal with it also tell how to recognize problems that are hard to solve and which techniques are available in that case.

Working with algorithms should be fun and exciting. The design of algorithms is a creative task requiring the solution of new problems and old problems in disguise. To be successful, we believe that it is important to enjoy the challenge that a new problem poses. To this end, we have included more examples and exercises of a combinatorial and recreational nature than is typical for a book of this type. All too often the challenge of an unsolved problem is experienced as a threat rather than as an opportunity, and we hope that these examples and exercises help to remove the threat.

Examples of real-word applications of algorithms in this book include data compression in Section 7.5, and the Boyer-Moore-Horspool algorithm in Section 9.4, which is used as part of the implementation of agrep. Most sections of the book introduce a motivating example in the first paragraph. The closest-pair problem (Section 5.3) begins with a pattern recognition example, and Section 8.4, which is concerned with the longest-common-subsequence problem, begins with a discussion of the analysis of proteins.

Algorithm design and analysis are best learned by experience. For this reason, we provide large numbers of worked examples and exercises. Worked examples show how to deal with algorithms, and exercises let the reader practice the techniques. There are over 300 worked examples throughout the book. These examples clarify and show how to develop algorithms, demonstrate applications of the theory, elucidate proofs, and help to motivate the material. The book contains over 1450 exercises, from routine to challenging, which were carefully developed through classroom testing. Close attention was paid to clarity and precision. Because some instant feedback is essential for students, we provide answers to about one-third of the end-of-section exercises (marked with "S" in the exercises) in the back of the book. Solutions to the remaining end-of-section exercises are reserved for instructors (see the Instructor Supplement section that follows).

Prerequisites

The principal computer science prerequisite is a data structures course that covers stacks, queues, linked lists,

trees, and graphs. A course in discrete mathematics that covers logic, asymptotic notation (e.g., "big oh" notation), and recurrence relations and their solution by iteration is the main mathematics prerequisite. We do not use advanced methods such as generating functions. In one or two places, we use some basic concepts from calculus. The mathematics topics and data structures used in this book are summarized in Chapters 2 and 3. Some or all of these chapters can be used for reference or review or incorporated into an algorithms course as needed.

Content

Following the first three chapters (containing an introduction, mathematics topics, and data structures), the book presents five chapters that emphasize design techniques.

Chapter 4 features searching techniques, including novel applications such as region-finding in digital pictures.

The divide-and-conquer technique is introduced in Chapter 5. Among the problems considered are a tiling problem, finding the closest pair of points in the plane, and Strassen's matrix-product algorithm. Chapter 6 deals with sorting and selection. Divide-and-conquer is used to develop many of the algorithms in this chapter.

Chapter 7 shows how to use the greedy method to develop algorithms. After showing how to use the greedy method in a simple setting (coin changing), we present Kruskal's algorithm, Prim's algorithm, Dijkstra's algorithm, Huffman's algorithm, and a solution of the continuous-knapsack problem.

Chapter 8 covers the technique of dynamic programming. As in Chapter 7, we first show how dynamic programming operates in a simple setting (computing Fibonacci numbers). We next revisit the coin-changing problem (from Chapter 7 on the greedy method) and contrast dynamic programming with the greedy method. We then discuss optimal grouping of matrices, the longest-common-subsequence problem, and the algorithms of Floyd and Warshall.

Chapter 9 discusses text-searching techniques, including the Knuth-Morris-Pratt and Boyer-Moore-Horspool algorithms, and algorithms for non-exact searching.

In Chapter 10, we investigate NP-completeness—a theoretical approach to recognizing and understanding the limitations of algorithms. We include many examples from different areas such as cellular phone networks, games, and biological computing to illustrate the ubiquity and universality of NPcompleteness.

It is widely believed that NP-complete problems cannot be solved efficiently by algorithms. Nevertheless, these problems arise in applications and have to be solved in practice. Chapter 11, Coping with NP-Completeness, presents a collection of techniques originating in practice and theory to deal with NP-complete problems. Among the approaches discussed are approximation, parameterization, and use of heuristics.

Chapter 12 presents fundamental algorithms for parallel architectures, including algorithms for the PRAM and sorting networks, and offers an introduction to computation in distributed environments.

Pedagogy

Each section (except Section 12.1, which is an introductory section) concludes with Section Exercises. The book contains over 1100 Section Exercises. Some of these exercises check for basic understanding of the material (e.g., some ask for a trace of an algorithm), while others check for a deeper understanding of the

material (e.g., some investigate alternative algorithms). Exercises that are more challenging than average are indicated with a star, *.

Each chapter ends with a Notes section, which is followed by Chapter Exercises. Notes sections contain suggestions for further reading and pointers to references. Chapter Exercises, some of which have hints, integrate the material of the chapter. The book contains over 350 Chapter Exercises. They are, on the whole, more challenging than the Section Exercises. We have included some very challenging Chapter Exercises marked with two stars. These will probably require instructor guidance, and some are appropriate for a small project.

Lower bounds for problems are integrated into the chapters that discuss those problems rather than being segregated into separate chapters. For example, after presenting several sorting algorithms, we discuss a lower bound for comparison-based sorting (Section 6.3).

We present and discuss many recent results, for example, parameterized complexity (Section 11.4), a recent area of research.

Algorithms are written in pseudocode that is close to the syntax of the familiar C, C++, and Java family of languages. Data types, semicolons, obscure features of the languages, and so on, are not used because we have found that specifying algorithms by writing actual code obscures the algorithm description and makes it difficult for someone not familiar with the language to understand the algorithm. The pseudocode used is completely described

Figures illustrate concepts, show how algorithms work, elucidate proofs, and motivate the material. Several figures illustrate proofs of theorems. The captions of these figures provide additional explanation and insight into the proofs.

Attention has been given to finding the most direct and comprehensible proofs of correctness as examples, see Theorem 7.2.5 from which the correctness of both Kruskal's and Prim's algorithms are derived and the proof of the correctness of Dijkstra's algorithm (Theorem 7.4.5).

We present several examples and arguments to show that our time bounds for algorithms are sharp. See, for example, the subsection in Section 7.3, Lower Bound Time Estimate, which shows that the upper bound for the worst-case time of Prim's algorithm using a binary heap is sharp, and the discussion just before Theorem 7.5.4, which shows that the upper bound for the worst-case time of Huffman's algorithm is sharp.,

ALGORITHMS BY RICHARD JOHNSONBAUGH, MARCUS SCHAEFER PDF

Algorithms By Richard Johnsonbaugh, Marcus Schaefer. Checking out makes you much better. Which says? Several wise words claim that by reading, your life will certainly be better. Do you think it? Yeah, prove it. If you require guide Algorithms By Richard Johnsonbaugh, Marcus Schaefer to check out to prove the wise words, you could see this web page completely. This is the site that will provide all the books that possibly you need. Are guide's compilations that will make you really feel interested to review? One of them here is the Algorithms By Richard Johnsonbaugh, Marcus Schaefer that we will suggest.

Reading routine will consistently lead people not to pleased reading *Algorithms By Richard Johnsonbaugh, Marcus Schaefer*, a publication, 10 e-book, hundreds books, and a lot more. One that will make them feel satisfied is finishing reviewing this publication Algorithms By Richard Johnsonbaugh, Marcus Schaefer as well as getting the notification of guides, after that finding the various other next e-book to read. It proceeds an increasing number of. The time to finish checking out an e-book Algorithms By Richard Johnsonbaugh, Marcus Schaefer will be constantly different relying on spar time to spend; one instance is this Algorithms By Richard Johnsonbaugh, Marcus Schaefer

Now, how do you know where to purchase this publication Algorithms By Richard Johnsonbaugh, Marcus Schaefer Don't bother, now you may not go to guide establishment under the bright sunlight or evening to look guide Algorithms By Richard Johnsonbaugh, Marcus Schaefer We right here constantly help you to locate hundreds sort of publication. One of them is this book entitled Algorithms By Richard Johnsonbaugh, Marcus Schaefer You could visit the link web page given in this collection and afterwards go with downloading. It will certainly not take even more times. Just hook up to your internet access and also you could access guide Algorithms By Richard Johnsonbaugh, Marcus Schaefer online. Obviously, after downloading and install Algorithms By Richard Johnsonbaugh, Marcus Schaefer, you might not publish it.

ALGORITHMS BY RICHARD JOHNSONBAUGH, MARCUS SCHAEFER PDF

Filling the void left by other algorithms books, Algorithms and Data Structures provides an approach that emphasizes design techniques. The volume includes application of algorithms, examples, end-of-section exercises, end-of-chapter exercises, hints and solutions to selected exercises, figures and notes to help the reader master the design and analysis of algorithms. This volume covers data structures, searching techniques, divided-and-conquer sorting and selection, greedy algorithms, dynamic programming, text searching, computational algebra, P and NP and parallel algorithms. For those interested in a better understanding of algorithms.

- Sales Rank: #869648 in Books
- Published on: 2003-08-10
- Original language: English
- Number of items: 1
- Dimensions: 10.10" h x 1.80" w x 8.30" l, 3.30 pounds
- Binding: Hardcover
- 768 pages

From the Back Cover

Algorithms is written for an introductory upper-level undergraduate or graduate course in algorithms. With/their many years of experience in teaching algorithms courses, Richard Johnsonbaugh and Marcus Schaefer include applications of algorithms, examples, end-of-section exercises, end-of-chapter exercises, solutions to selected exercises, and notes to help the reader understand and master algorithms.

Key Features

- Links theory to real-world applications such as data compression, region-finding in digital pictures, cellular phone networks, and the implementation of agrep.
- Includes five chapters that emphasize design techniques: searching (including backtracking), divide and conquer, sorting, selection, the greedy method, and dynamic programming.
- Covers distributed algorithms—a topic recommended by the ACM (2001 report) for an undergraduate curriculum.
- Features a collection of techniques, including approximation, parameterization (a recent area of research), and use of heuristics, to deal with NP-complete problems.
- Contains more than 1450 carefully developed and classroom-tested exercises, from routine to challenging. About one-third of the end-of-section exercises include solutions.
- Provides a robust Companion Website that supplements the text by providing algorithm simulation software, PowerPoint® slides, late breaking news about algorithms, references about the book's topics, computer programs, and more.
- Includes more than 300 worked examples, which provide motivation, clarify concepts, and show how to

develop algorithms, demonstrate applications of the theory, and elucidate proofs.

About the Author

Richard Johnsonbaugh is Professor Emeritus of Computer Science at DePaul University. He has degrees in computer science and mathematics from the University of Oregon, Yale University, and the University of Illinois at Chicago. He is the author of numerous articles and books, including Discrete Mathematics, Fifth Edition, and, with co-author Martin Kalin, Object-Oriented Programming in C++, Second Edition, Applications Programming in C++, and Applications Programming in ANSI C, Third Edition.

Marcus Schaefer is Assistant Professor of Computer Science at DePaul University. He holds degrees in computer science and mathematics from the University of Chicago and the Universitat Karlsruhe. He has authored and co-authored several articles on complexity theory, computability, and graph theory.

Why We Wrote This Book

Intended for an upper-level undergraduate or graduate course in algorithms, this book is based on our combined 25 years of experience in teaching this course. Our major goals in writing this book were to

- Emphasize design techniques.
- Show that algorithms are fun and exciting.
- Include real-world applications.
- Provide numerous worked examples and exercises.

Faced with a new computational problem, a designer will often be able to solve it by using one of the algorithms in this book, perhaps after modifying or adapting it slightly. However, some problems cannot be solved by any of the algorithms in this book. For this reason, we present a repertoire of design techniques that can be used to solve the problem and help the reader to develop intuition about which techniques are likely to succeed. The chapters on NP-completeness and how to deal with it also tell how to recognize problems that are hard to solve and which techniques are available in that case.

Working with algorithms should be fun and exciting. The design of algorithms is a creative task requiring the solution of new problems and old problems in disguise. To be successful, we believe that it is important to enjoy the challenge that a new problem poses. To this end, we have included more examples and exercises of a combinatorial and recreational nature than is typical for a book of this type. All too often the challenge of an unsolved problem is experienced as a threat rather than as an opportunity, and we hope that these examples and exercises help to remove the threat.

Examples of real-word applications of algorithms in this book include data compression in Section 7.5, and the Boyer-Moore-Horspool algorithm in Section 9.4, which is used as part of the implementation of agrep. Most sections of the book introduce a motivating example in the first paragraph. The closest-pair problem (Section 5.3) begins with a pattern recognition example, and Section 8.4, which is concerned with the longest-common-subsequence problem, begins with a discussion of the analysis of proteins.

Algorithm design and analysis are best learned by experience. For this reason, we provide large numbers of worked examples and exercises. Worked examples show how to deal with algorithms, and exercises let the reader practice the techniques. There are over 300 worked examples throughout the book. These examples clarify and show how to develop algorithms, demonstrate applications of the theory, elucidate proofs, and

help to motivate the material. The book contains over 1450 exercises, from routine to challenging, which were carefully developed through classroom testing. Close attention was paid to clarity and precision. Because some instant feedback is essential for students, we provide answers to about one-third of the end-of-section exercises (marked with "S" in the exercises) in the back of the book. Solutions to the remaining end-of-section exercises are reserved for instructors (see the Instructor Supplement section that follows).

Prerequisites

The principal computer science prerequisite is a data structures course that covers stacks, queues, linked lists, trees, and graphs. A course in discrete mathematics that covers logic, asymptotic notation (e.g., "big oh" notation), and recurrence relations and their solution by iteration is the main mathematics prerequisite. We do not use advanced methods such as generating functions. In one or two places, we use some basic concepts from calculus. The mathematics topics and data structures used in this book are summarized in Chapters 2 and 3. Some or all of these chapters can be used for reference or review or incorporated into an algorithms course as needed.

Content

Following the first three chapters (containing an introduction, mathematics topics, and data structures), the book presents five chapters that emphasize design techniques.

Chapter 4 features searching techniques, including novel applications such as region-finding in digital pictures.

The divide-and-conquer technique is introduced in Chapter 5. Among the problems considered are a tiling problem, finding the closest pair of points in the plane, and Strassen's matrix-product algorithm. Chapter 6 deals with sorting and selection. Divide-and-conquer is used to develop many of the algorithms in this chapter.

Chapter 7 shows how to use the greedy method to develop algorithms. After showing how to use the greedy method in a simple setting (coin changing), we present Kruskal's algorithm, Prim's algorithm, Dijkstra's algorithm, Huffman's algorithm, and a solution of the continuous-knapsack problem.

Chapter 8 covers the technique of dynamic programming. As in Chapter 7, we first show how dynamic programming operates in a simple setting (computing Fibonacci numbers). We next revisit the coin-changing problem (from Chapter 7 on the greedy method) and contrast dynamic programming with the greedy method. We then discuss optimal grouping of matrices, the longest-common-subsequence problem, and the algorithms of Floyd and Warshall.

Chapter 9 discusses text-searching techniques, including the Knuth-Morris-Pratt and Boyer-Moore-Horspool algorithms, and algorithms for non-exact searching.

In Chapter 10, we investigate NP-completeness—a theoretical approach to recognizing and understanding the limitations of algorithms. We include many examples from different areas such as cellular phone networks, games, and biological computing to illustrate the ubiquity and universality of NPcompleteness.

It is widely believed that NP-complete problems cannot be solved efficiently by algorithms. Nevertheless, these problems arise in applications and have to be solved in practice. Chapter 11, Coping with NP-Completeness, presents a collection of techniques originating in practice and theory to deal with NP-complete problems. Among the approaches discussed are approximation, parameterization, and use of heuristics.

Chapter 12 presents fundamental algorithms for parallel architectures, including algorithms for the PRAM and sorting networks, and offers an introduction to computation in distributed environments.

Pedagogy

Each section (except Section 12.1, which is an introductory section) concludes with Section Exercises. The book contains over 1100 Section Exercises. Some of these exercises check for basic understanding of the material (e.g., some ask for a trace of an algorithm), while others check for a deeper understanding of the material (e.g., some investigate alternative algorithms). Exercises that are more challenging than average are indicated with a star, *.

Each chapter ends with a Notes section, which is followed by Chapter Exercises. Notes sections contain suggestions for further reading and pointers to references. Chapter Exercises, some of which have hints, integrate the material of the chapter. The book contains over 350 Chapter Exercises. They are, on the whole, more challenging than the Section Exercises. We have included some very challenging Chapter Exercises marked with two stars. These will probably require instructor guidance, and some are appropriate for a small project.

Lower bounds for problems are integrated into the chapters that discuss those problems rather than being segregated into separate chapters. For example, after presenting several sorting algorithms, we discuss a lower bound for comparison-based sorting (Section 6.3).

We present and discuss many recent results, for example, parameterized complexity (Section 11.4), a recent area of research.

Algorithms are written in pseudocode that is close to the syntax of the familiar C, C++, and Java family of languages. Data types, semicolons, obscure features of the languages, and so on, are not used because we have found that specifying algorithms by writing actual code obscures the algorithm description and makes it difficult for someone not familiar with the language to understand the algorithm. The pseudocode used is completely described

Figures illustrate concepts, show how algorithms work, elucidate proofs, and motivate the material. Several figures illustrate proofs of theorems. The captions of these figures provide additional explanation and insight into the proofs.

Attention has been given to finding the most direct and comprehensible proofs of correctness as examples, see Theorem 7.2.5 from which the correctness of both Kruskal's and Prim's algorithms are derived and the proof of the correctness of Dijkstra's algorithm (Theorem 7.4.5).

We present several examples and arguments to show that our time bounds for algorithms are sharp. See, for example, the subsection in Section 7.3, Lower Bound Time Estimate, which shows that the upper bound for the worst-case time of Prim's algorithm using a binary heap is sharp, and the discussion just before Theorem 7.5.4, which shows that the upper bound for the worst-case time of Huffman's algorithm is sharp.,

Most helpful customer reviews

3 of 3 people found the following review helpful.
Difficult Introduction to Algorithms
By Thomas Manning
We are using this book as a text in our Computer Science II course. Unfortunately, this is a somewhat

difficult book to try and learn from. The descriptions are very dense, filled with many proofs, and little in the way of an explanation as to what an algorithm does, or what it's purpose is. In other words, rather than simply stating what an algorithm is used for and the general ideas behind it in a couple clear paragraphs, it spreads this information out between unclear and hard to follow proofs (that skip over important steps), using difficult language, and between multiple "failed attempts" which try to show how to arrive at the optimal way of implementing an algorithm without clearly stating so.

I want to state now that I spend a lot of time reading dense texts from other computer science books, to Microsoft documentation, to even some Wikipedia articles. While I can bear through this book, it is one of the more difficult things I've had to deal with. In my opinion, its worse than the level of perceived greatness from teachers, and practical uselessness to students as is found in many math books that can prove everything and explain almost nothing.

However, I have not had a chance to check out any other computer science texts, so perhaps as the last reviewer stated, this *IS* one of the better ones...

If you're a professor, before choosing this book, or any other really, try to see if it at least has clear explanations in addition to whatever else one might want. Because, as a student who often relies on the textbook for learning things, this one is near impossible.

2 of 2 people found the following review helpful.
Algorithms Richard Johnsonbaugh
By bikejpmz
Good book, explains many algorithms in more plain, easier to understand English than the Introduction to Algorithms by Cormen, Lieserson, Rivest, and Stein. Although it is different in the amount and focus of material. Both books used in conjunction is helpful.

0 of 2 people found the following review helpful.
Five Stars
By Julio
book came in and was in great condition.

See all 6 customer reviews...

ALGORITHMS BY RICHARD JOHNSONBAUGH, MARCUS SCHAEFER PDF

You could save the soft documents of this book **Algorithms By Richard Johnsonbaugh, Marcus Schaefer** It will rely on your downtime and also tasks to open up as well as review this book Algorithms By Richard Johnsonbaugh, Marcus Schaefer soft file. So, you may not hesitate to bring this book Algorithms By Richard Johnsonbaugh, Marcus Schaefer all over you go. Just include this sot file to your kitchen appliance or computer system disk to permit you check out whenever as well as all over you have time.

From the Back Cover

Algorithms is written for an introductory upper-level undergraduate or graduate course in algorithms. With/their many years of experience in teaching algorithms courses, Richard Johnsonbaugh and Marcus Schaefer include applications of algorithms, examples, end-of-section exercises, end-of-chapter exercises, solutions to selected exercises, and notes to help the reader understand and master algorithms.

Key Features

- Links theory to real-world applications such as data compression, region-finding in digital pictures, cellular phone networks, and the implementation of agrep.
- Includes five chapters that emphasize design techniques: searching (including backtracking), divide and conquer, sorting, selection, the greedy method, and dynamic programming.
- Covers distributed algorithms—a topic recommended by the ACM (2001 report) for an undergraduate curriculum.
- Features a collection of techniques, including approximation, parameterization (a recent area of research), and use of heuristics, to deal with NP-complete problems.
- Contains more than 1450 carefully developed and classroom-tested exercises, from routine to challenging. About one-third of the end-of-section exercises include solutions.
- Provides a robust Companion Website that supplements the text by providing algorithm simulation software, PowerPoint® slides, late breaking news about algorithms, references about the book's topics, computer programs, and more.
- Includes more than 300 worked examples, which provide motivation, clarify concepts, and show how to develop algorithms, demonstrate applications of the theory, and elucidate proofs.

About the Author

Richard Johnsonbaugh is Professor Emeritus of Computer Science at DePaul University. He has degrees in computer science and mathematics from the University of Oregon, Yale University, and the University of Illinois at Chicago. He is the author of numerous articles and books, including Discrete Mathematics, Fifth Edition, and, with co-author Martin Kalin, Object-Oriented Programming in C++, Second Edition, Applications Programming in C++, and Applications Programming in ANSI C, Third Edition.

Marcus Schaefer is Assistant Professor of Computer Science at DePaul University. He holds degrees in computer science and mathematics from the University of Chicago and the Universitat Karlsruhe. He has authored and co-authored several articles on complexity theory, computability, and graph theory.

Why We Wrote This Book

Intended for an upper-level undergraduate or graduate course in algorithms, this book is based on our combined 25 years of experience in teaching this course. Our major goals in writing this book were to

- Emphasize design techniques.
- Show that algorithms are fun and exciting.
- Include real-world applications.
- Provide numerous worked examples and exercises.

Faced with a new computational problem, a designer will often be able to solve it by using one of the algorithms in this book, perhaps after modifying or adapting it slightly. However, some problems cannot be solved by any of the algorithms in this book. For this reason, we present a repertoire of design techniques that can be used to solve the problem and help the reader to develop intuition about which techniques are likely to succeed. The chapters on NP-completeness and how to deal with it also tell how to recognize problems that are hard to solve and which techniques are available in that case.

Working with algorithms should be fun and exciting. The design of algorithms is a creative task requiring the solution of new problems and old problems in disguise. To be successful, we believe that it is important to enjoy the challenge that a new problem poses. To this end, we have included more examples and exercises of a combinatorial and recreational nature than is typical for a book of this type. All too often the challenge of an unsolved problem is experienced as a threat rather than as an opportunity, and we hope that these examples and exercises help to remove the threat.

Examples of real-word applications of algorithms in this book include data compression in Section 7.5, and the Boyer-Moore-Horspool algorithm in Section 9.4, which is used as part of the implementation of agrep. Most sections of the book introduce a motivating example in the first paragraph. The closest-pair problem (Section 5.3) begins with a pattern recognition example, and Section 8.4, which is concerned with the longest-common-subsequence problem, begins with a discussion of the analysis of proteins.

Algorithm design and analysis are best learned by experience. For this reason, we provide large numbers of worked examples and exercises. Worked examples show how to deal with algorithms, and exercises let the reader practice the techniques. There are over 300 worked examples throughout the book. These examples clarify and show how to develop algorithms, demonstrate applications of the theory, elucidate proofs, and help to motivate the material. The book contains over 1450 exercises, from routine to challenging, which were carefully developed through classroom testing. Close attention was paid to clarity and precision. Because some instant feedback is essential for students, we provide answers to about one-third of the end-of-section exercises (marked with "S" in the exercises) in the back of the book. Solutions to the remaining end-of-section exercises are reserved for instructors (see the Instructor Supplement section that follows).

Prerequisites

The principal computer science prerequisite is a data structures course that covers stacks, queues, linked lists, trees, and graphs. A course in discrete mathematics that covers logic, asymptotic notation (e.g., "big oh" notation), and recurrence relations and their solution by iteration is the main mathematics prerequisite. We do not use advanced methods such as generating functions. In one or two places, we use some basic concepts from calculus. The mathematics topics and data structures used in this book are summarized in Chapters 2 and 3. Some or all of these chapters can be used for reference or review or incorporated into an algorithms course as needed.

Content

Following the first three chapters (containing an introduction, mathematics topics, and data structures), the book presents five chapters that emphasize design techniques.

Chapter 4 features searching techniques, including novel applications such as region-finding in digital pictures.

The divide-and-conquer technique is introduced in Chapter 5. Among the problems considered are a tiling problem, finding the closest pair of points in the plane, and Strassen's matrix-product algorithm. Chapter 6 deals with sorting and selection. Divide-and-conquer is used to develop many of the algorithms in this chapter.

Chapter 7 shows how to use the greedy method to develop algorithms. After showing how to use the greedy method in a simple setting (coin changing), we present Kruskal's algorithm, Prim's algorithm, Dijkstra's algorithm, Huffman's algorithm, and a solution of the continuous-knapsack problem.

Chapter 8 covers the technique of dynamic programming. As in Chapter 7, we first show how dynamic programming operates in a simple setting (computing Fibonacci numbers). We next revisit the coin-changing problem (from Chapter 7 on the greedy method) and contrast dynamic programming with the greedy method. We then discuss optimal grouping of matrices, the longest-common-subsequence problem, and the algorithms of Floyd and Warshall.

Chapter 9 discusses text-searching techniques, including the Knuth-Morris-Pratt and Boyer-Moore-Horspool algorithms, and algorithms for non-exact searching.

In Chapter 10, we investigate NP-completeness—a theoretical approach to recognizing and understanding the limitations of algorithms. We include many examples from different areas such as cellular phone networks, games, and biological computing to illustrate the ubiquity and universality of NPcompleteness.

It is widely believed that NP-complete problems cannot be solved efficiently by algorithms. Nevertheless, these problems arise in applications and have to be solved in practice. Chapter 11, Coping with NP-Completeness, presents a collection of techniques originating in practice and theory to deal with NP-complete problems. Among the approaches discussed are approximation, parameterization, and use of heuristics.

Chapter 12 presents fundamental algorithms for parallel architectures, including algorithms for the PRAM and sorting networks, and offers an introduction to computation in distributed environments.

Pedagogy

Each section (except Section 12.1, which is an introductory section) concludes with Section Exercises. The book contains over 1100 Section Exercises. Some of these exercises check for basic understanding of the material (e.g., some ask for a trace of an algorithm), while others check for a deeper understanding of the material (e.g., some investigate alternative algorithms). Exercises that are more challenging than average are indicated with a star, *.

Each chapter ends with a Notes section, which is followed by Chapter Exercises. Notes sections contain suggestions for further reading and pointers to references. Chapter Exercises, some of which have hints, integrate the material of the chapter. The book contains over 350 Chapter Exercises. They are, on the whole, more challenging than the Section Exercises. We have included some very challenging Chapter Exercises

marked with two stars. These will probably require instructor guidance, and some are appropriate for a small project.

Lower bounds for problems are integrated into the chapters that discuss those problems rather than being segregated into separate chapters. For example, after presenting several sorting algorithms, we discuss a lower bound for comparison-based sorting (Section 6.3).

We present and discuss many recent results, for example, parameterized complexity (Section 11.4), a recent area of research.

Algorithms are written in pseudocode that is close to the syntax of the familiar C, C++, and Java family of languages. Data types, semicolons, obscure features of the languages, and so on, are not used because we have found that specifying algorithms by writing actual code obscures the algorithm description and makes it difficult for someone not familiar with the language to understand the algorithm. The pseudocode used is completely described

Figures illustrate concepts, show how algorithms work, elucidate proofs, and motivate the material. Several figures illustrate proofs of theorems. The captions of these figures provide additional explanation and insight into the proofs.

Attention has been given to finding the most direct and comprehensible proofs of correctness as examples, see Theorem 7.2.5 from which the correctness of both Kruskal's and Prim's algorithms are derived and the proof of the correctness of Dijkstra's algorithm (Theorem 7.4.5).

We present several examples and arguments to show that our time bounds for algorithms are sharp. See, for example, the subsection in Section 7.3, Lower Bound Time Estimate, which shows that the upper bound for the worst-case time of Prim's algorithm using a binary heap is sharp, and the discussion just before Theorem 7.5.4, which shows that the upper bound for the worst-case time of Huffman's algorithm is sharp.,