

SECOND EDITION

ESSENTIAL
Algorithms

A PRACTICAL APPROACH
TO COMPUTER ALGORITHMS
USING PYTHON AND C#

Rod Stephens

WILEY

Essential Algorithms



Essential Algorithms

A Practical Approach to Computer
Algorithms Using Python® and C#

Rod Stephens

WILEY

Essential Algorithms: A Practical Approach to Computer Algorithms Using Python® and C#

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2019 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-119-57599-3
ISBN: 978-1-119-57596-2 (ebk)
ISBN: 978-1-119-57598-6 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that Internet websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2019933736

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Python is a registered trademark of Python Software Foundation. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

For Maki



About the Author

Rod Stephens started out as a mathematician, but while studying at MIT, he discovered how much fun algorithms are. He took every algorithms course MIT offered, and he has been writing complex algorithms ever since.

During his career, Rod has worked on an eclectic assortment of applications in fields such as telephone switching, billing, repair dispatching, tax processing, wastewater treatment, concert ticket sales, cartography, and training for professional football players.

Rod was a Microsoft Visual Basic Most Valuable Professional (MVP) for 15 years and has taught introductory programming courses. He has written more than 30 books that have been translated into languages from all over the world. He has also written more than 250 magazine articles covering C#, Visual Basic, Visual Basic for Applications, Delphi, and Java.

Rod's popular C# Helper website (<http://www.csharpHelper.com>) receives millions of hits per year and contains tips, tricks, and example programs for C# programmers. His VB Helper website (<http://www.vb-helper.com>) contains similar material for Visual Basic programmers.

You can contact Rod at: RodStephens@csharpHelper.com.



About the Technical Editor

John Mueller is a freelance author and technical editor. He has writing in his blood, having produced 112 books and more than 600 articles to date. The topics range from networking to artificial intelligence and from database management to heads-down programming. Some of his current books include discussions of data science, machine learning, and algorithms. His technical editing skills have helped more than 70 authors refine the content of their manuscripts. John has provided technical editing services to numerous magazines, performed various types of consulting, and he writes certification exams as well.

Be sure to read John's blog at: <http://blog.johnmuellerbooks.com/>. You can reach John on the Internet at John@JohnMuellerBooks.com. John also has a website at <http://www.johnmuellerbooks.com/>. Be sure to follow John on Amazon at <https://www.amazon.com/John-Mueller/>.



Senior Acquisitions Editor

Kenyon Brown

Editorial Manager

Pete Gaughan

Associate Publisher

Jim Minatel

Production Manager

Kathleen Wisor

Project Editor

Gary Schwartz

Production Editor

Athiyappan Lalith Kumar

Technical Editor

John Muller

Copy Editor

Kim Wimpsett

Proofreader

Nancy Bell

Indexer

Potomac Indexing, LLC

Cover Designer

Wiley



Acknowledgments

Thanks to Ken Brown, Devon Lewis, Gary Schwartz, Pete Gaughan, Jim Minatel, Athiyappan Lalitkumar, and everyone else at Wiley that helped make this book possible.

Thanks to longtime friend John Mueller, who provided his technical expertise to help make the information in this book as accurate as possible. (Any remaining mistakes are mine, not his.)

Thanks also to Sunil Kumar for his generous feedback on the first edition.



Contents at a glance

Introduction	xxix
Chapter 1 Algorithm Basics	1
Chapter 2 Numerical Algorithms	23
Chapter 3 Linked Lists	71
Chapter 4 Arrays	103
Chapter 5 Stacks and Queues	135
Chapter 6 Sorting	167
Chapter 7 Searching	201
Chapter 8 Hash Tables	209
Chapter 9 Recursion	227
Chapter 10 Trees	285
Chapter 11 Balanced Trees	349
Chapter 12 Decision Trees	367
Chapter 13 Basic Network Algorithms	403
Chapter 14 More Network Algorithms	451
Chapter 15 String Algorithms	493
Chapter 16 Cryptography	519
Chapter 17 Complexity Theory	543

Chapter 18	Distributed Algorithms	561
Chapter 19	Interview Puzzles	595
Appendix A	Summary of Algorithmic Concepts	607
Appendix B	Solutions to Exercises	623
Glossary		711
Index		739



Contents

Introduction	xxix
Chapter 1 Algorithm Basics	1
Approach	2
Algorithms and Data Structures	2
Pseudocode	3
Algorithm Features	6
Big O Notation	7
Rule 1	8
Rule 2	8
Rule 3	9
Rule 4	9
Rule 5	10
Common Run Time Functions	11
1	11
Log N	11
Sqrt N	14
N	14
N log N	15
N ²	15
2 ^N	15
N!	16
Visualizing Functions	16
Practical Considerations	18
Summary	19
Exercises	20

Chapter 2	Numerical Algorithms	23
	Randomizing Data	23
	Generating Random Values	23
	Generating Values	24
	Ensuring Fairness	26
	Getting Fairness from Biased Sources	28
	Randomizing Arrays	29
	Generating Nonuniform Distributions	30
	Making Random Walks	31
	Making Self-Avoiding Walks	33
	Making Complete Self-Avoiding Walks	34
	Finding Greatest Common Divisors	36
	Calculating Greatest Common Divisors	36
	Extending Greatest Common Divisors	38
	Performing Exponentiation	40
	Working with Prime Numbers	42
	Finding Prime Factors	42
	Finding Primes	44
	Testing for Primality	45
	Performing Numerical Integration	47
	The Rectangle Rule	48
	The Trapezoid Rule	49
	Adaptive Quadrature	50
	Monte Carlo Integration	54
	Finding Zeros	55
	Gaussian Elimination	57
	Forward Elimination	58
	Back Substitution	60
	The Algorithm	61
	Least Squares Fits	62
	Linear Least Squares	62
	Polynomial Least Squares	64
	Summary	67
	Exercises	68
Chapter 3	Linked Lists	71
	Basic Concepts	71
	Singly Linked Lists	72
	Iterating Over the List	73
	Finding Cells	73
	Using Sentinels	74
	Adding Cells at the Beginning	75
	Adding Cells at the End	76
	Inserting Cells After Other Cells	77
	Deleting Cells	78
	Doubly Linked Lists	79
	Sorted Linked Lists	81

Self-Organizing Linked Lists	82
Move To Front (MTF)	83
Swap	83
Count	84
Hybrid Methods	84
Pseudocode	85
Linked-List Algorithms	86
Copying Lists	86
Sorting with Insertionsort	87
Sorting with Selectionsort	88
Multithreaded Linked Lists	90
Linked Lists with Loops	91
Marking Cells	92
Using Hash Tables	93
List Retracing	94
List Reversal	95
Tortoise and Hare	98
Loops in Doubly Linked Lists	100
Summary	100
Exercises	101
Chapter 4	
Arrays	103
Basic Concepts	103
One-Dimensional Arrays	106
Finding Items	106
Finding Minimum, Maximum, and Average	107
Finding Median	108
Finding Mode	109
Inserting Items	112
Removing Items	113
Nonzero Lower Bounds	114
Two Dimensions	114
Higher Dimensions	115
Triangular Arrays	118
Sparse Arrays	121
Find a Row or Column	123
Get a Value	124
Set a Value	125
Delete a Value	127
Matrices	129
Summary	131
Exercises	132
Chapter 5	
Stacks and Queues	135
Stacks	135
Linked-List Stacks	136
Array Stacks	138
Double Stacks	139
Stack Algorithms	141

Reversing an Array	141
Train Sorting	142
Tower of Hanoi	143
Stack Insertionsort	145
Stack Selectionsort	146
Queues	147
Linked-List Queues	148
Array Queues	148
Specialized Queues	151
Priority Queues	151
Dequeues	152
Binomial Heaps	152
Binomial Trees	152
Binomial Heaps	154
Merging Trees	155
Merging Heaps	156
Merging Tree Lists	156
Merging Trees	158
Enqueue	161
Dequeue	162
Runtime	163
Summary	163
Exercises	164
Chapter 6	
Sorting	167
$O(N^2)$ Algorithms	168
Insertionsort in Arrays	168
Selectionsort in Arrays	170
Bubblesort	171
$O(N \log N)$ Algorithms	174
Heapsort	175
Storing Complete Binary Trees	175
Defining Heaps	176
Implementing Heapsort	180
Quicksort	181
Analyzing Quicksort's Run Time	182
Picking a Dividing Item	184
Implementing Quicksort with Stacks	185
Implementing Quicksort in Place	185
Using Quicksort	188
Mergesort	189
Sub $O(N \log N)$ Algorithms	192
Countingsort	192
Pigeonhole Sort	193
Bucketssort	195
Summary	197
Exercises	198

Chapter 7	Searching	201
	Linear Search	202
	Binary Search	203
	Interpolation Search	204
	Majority Voting	205
	Summary	207
	Exercises	208
Chapter 8	Hash Tables	209
	Hash Table Fundamentals	210
	Chaining	211
	Open Addressing	213
	Removing Items	214
	Linear Probing	215
	Quadratic Probing	217
	Pseudorandom Probing	219
	Double Hashing	219
	Ordered Hashing	219
	Summary	222
	Exercises	222
Chapter 9	Recursion	227
	Basic Algorithms	228
	Factorial	228
	Fibonacci Numbers	230
	Rod-Cutting	232
	Brute Force	233
	Recursion	233
	Tower of Hanoi	235
	Graphical Algorithms	238
	Koch Curves	239
	Hilbert Curve	241
	Sierpiński Curve	243
	Gaskets	246
	The Skyline Problem	247
	Lists	248
	Divide and Conquer	249
	Backtracking Algorithms	252
	Eight Queens Problem	254
	Knight's Tour	257
	Selections and Permutations	260
	Selections with Loops	261
	Selections with Duplicates	262
	Selections Without Duplicates	264
	Permutations with Duplicates	265
	Permutations Without Duplicates	266
	Round-Robin Scheduling	267
	Odd Number of Teams	268

Even Number of Teams	270
Implementation	271
Recursion Removal	273
Tail Recursion Removal	274
Dynamic Programming	275
Bottom-Up Programming	277
General Recursion Removal	277
Summary	280
Exercises	281
Chapter 10 Trees	285
Tree Terminology	285
Binary Tree Properties	289
Tree Representations	292
Building Trees in General	292
Building Complete Trees	295
Tree Traversal	296
Preorder Traversal	297
Inorder Traversal	299
Postorder Traversal	300
Breadth-First Traversal	301
Traversal Uses	302
Traversal Run Times	303
Sorted Trees	303
Adding Nodes	303
Finding Nodes	306
Deleting Nodes	306
Lowest Common Ancestors	309
Sorted Trees	309
Parent Pointers	310
Parents and Depths	311
General Trees	312
Euler Tours	314
All Pairs	316
Threaded Trees	317
Building Threaded Trees	318
Using Threaded Trees	320
Specialized Tree Algorithms	322
The Animal Game	322
Expression Evaluation	324
Interval Trees	326
Building the Tree	328
Intersecting with Points	329
Intersecting with Intervals	330
Quadtrees	332
Adding Items	335
Finding Items	336

Tries	337
Adding Items	339
Finding Items	341
Summary	342
Exercises	342
Chapter 11 Balanced Trees	349
AVL Trees	350
Adding Values	350
Deleting Values	353
2-3 Trees	354
Adding Values	355
Deleting Values	356
B-Trees	359
Adding Values	360
Deleting Values	361
Balanced Tree Variations	362
Top-down B-trees	363
B+trees	363
Summary	365
Exercises	365
Chapter 12 Decision Trees	367
Searching Game Trees	368
Minimax	369
Initial Moves and Responses	373
Game Tree Heuristics	374
Searching General Decision Trees	375
Optimization Problems	376
Exhaustive Search	377
Branch and Bound	379
Decision Tree Heuristics	381
Random Search	381
Improving Paths	382
Simulated Annealing	384
Hill Climbing	385
Sorted Hill Climbing	386
Other Decision Tree Problems	387
Generalized Partition Problem	387
Subset Sum	388
Bin Packing	388
Cutting Stock	389
Knapsack	390
Traveling Salesman Problem	391
Satisfiability	391
Swarm Intelligence	392
Ant Colony Optimization	393

General Optimization	393
Traveling Salesman	393
Bees Algorithm	394
Swarm Simulation	394
Boids	395
Pseudoclassical Mechanics	396
Goals and Obstacles	397
Summary	397
Exercises	398
Chapter 13 Basic Network Algorithms	403
Network Terminology	403
Network Representations	407
Traversals	409
Depth-First Traversal	410
Breadth-First Traversal	412
Connectivity Testing	413
Spanning Trees	416
Minimal Spanning Trees	417
Euclidean Minimum Spanning Trees	418
Building Mazes	419
Strongly Connected Components	420
Kosaraju's Algorithm	421
Algorithm Discussion	422
Finding Paths	425
Finding Any Path	425
Label-Setting Shortest Paths	426
Label-Correcting Shortest Paths	430
All-Pairs Shortest Paths	431
Transitivity	436
Transitive Closure	437
Transitive Reduction	438
Acyclic Networks	439
General Networks	440
Shortest Path Modifications	441
Shape Points	441
Early Stopping	442
Bidirectional Search	442
Best-First Search	442
Turn Penalties and Prohibitions	443
Geometric Calculations	443
Expanded Node Networks	444
Interchange Networks	445
Summary	447
Exercises	447

Chapter 14	More Network Algorithms	451
	Topological Sorting	451
	Cycle Detection	455
	Map Coloring	456
	Two-Coloring	456
	Three-Coloring	458
	Four-Coloring	459
	Five-Coloring	459
	Other Map-Coloring Algorithms	462
	Maximal Flow	464
	Work Assignment	467
	Minimal Flow Cut	468
	Network Cloning	470
	Dictionaries	471
	Clone References	472
	Cliques	473
	Brute Force	474
	Bron–Kerbosch	475
	Sets R , P , and X	475
	Recursive Calls	476
	Pseudocode	476
	Example	477
	Variations	480
	Finding Triangles	480
	Brute Force	481
	Checking Local Links	481
	Chiba and Nishizeki	482
	Community Detection	483
	Maximal Cliques	483
	Girvan–Newman	483
	Clique Percolation	485
	Eulerian Paths and Cycles	485
	Brute Force	486
	Fleury’s Algorithm	486
	Hierholzer’s Algorithm	487
	Summary	488
	Exercises	489
Chapter 15	String Algorithms	493
	Matching Parentheses	494
	Evaluating Arithmetic Expressions	495
	Building Parse Trees	496
	Pattern Matching	497
	DFAs	497
	Building DFAs for Regular Expressions	500
	NFAs	502

String Searching	504
Calculating Edit Distance	508
Phonetic Algorithms	511
Soundex	511
Metaphone	513
Summary	514
Exercises	515
Chapter 16 Cryptography	519
Terminology	520
Transposition Ciphers	521
Row/Column Transposition	521
Column Transposition	523
Route Ciphers	525
Substitution Ciphers	526
Caesar Substitution	526
Vigenère Cipher	527
Simple Substitution	529
One-Time Pads	530
Block Ciphers	531
Substitution-Permutation Networks	531
Feistel Ciphers	533
Public-Key Encryption and RSA	534
Euler's Totient Function	535
Multiplicative Inverses	536
An RSA Example	536
Practical Considerations	537
Other Uses for Cryptography	538
Summary	539
Exercises	540
Chapter 17 Complexity Theory	543
Notation	544
Complexity Classes	545
Reductions	548
3SAT	549
Bipartite Matching	550
NP-Hardness	550
Detection, Reporting, and Optimization Problems	551
Detection \leq_p Reporting	552
Reporting \leq_p Optimization	552
Reporting \leq_p Detection	552
Optimization \leq_p Reporting	553
Approximate Optimization	553
NP-Complete Problems	554
Summary	557
Exercises	558

Chapter 18	Distributed Algorithms	561
	Types of Parallelism	562
	Systolic Arrays	562
	Distributed Computing	565
	Multi-CPU Processing	567
	Race Conditions	567
	Deadlock	571
	Quantum Computing	572
	Distributed Algorithms	573
	Debugging Distributed Algorithms	573
	Embarrassingly Parallel Algorithms	574
	Mergesort	576
	Dining Philosophers	577
	Randomization	578
	Resource Hierarchy	578
	Waiter	579
	Chandy/Misra	579
	The Two Generals Problem	580
	Byzantine Generals	581
	Consensus	584
	Leader Election	587
	Snapshot	588
	Clock Synchronization	589
	Summary	591
	Exercises	591
Chapter 19	Interview Puzzles	595
	Asking Interview Puzzle Questions	597
	Answering Interview Puzzle Questions	598
	Summary	602
	Exercises	604
Appendix A	Summary of Algorithmic Concepts	607
	Chapter 1: Algorithm Basics	607
	Chapter 2: Numeric Algorithms	608
	Chapter 3: Linked Lists	609
	Chapter 4: Arrays	610
	Chapter 5: Stacks and Queues	610
	Chapter 6: Sorting	610
	Chapter 7: Searching	611
	Chapter 8: Hash Tables	612
	Chapter 9: Recursion	612
	Chapter 10: Trees	614
	Chapter 11: Balanced Trees	615
	Chapter 12: Decision Trees	615
	Chapter 13: Basic Network Algorithms	616
	Chapter 14: More Network Algorithms	617
	Chapter 15: String Algorithms	618

Chapter 16: Cryptography	618
Chapter 17: Complexity Theory	619
Chapter 18: Distributed Algorithms	620
Chapter 19: Interview Puzzles	621
Appendix B Solutions to Exercises	623
Chapter 1: Algorithm Basics	623
Chapter 2: Numerical Algorithms	626
Chapter 3: Linked Lists	633
Chapter 4: Arrays	638
Chapter 5: Stacks and Queues	648
Chapter 6: Sorting	650
Chapter 7: Searching	653
Chapter 8: Hash Tables	655
Chapter 9: Recursion	658
Chapter 10: Trees	663
Chapter 11: Balanced Trees	670
Chapter 12: Decision Trees	675
Chapter 13: Basic Network Algorithms	678
Chapter 14: More Network Algorithms	681
Chapter 15: String Algorithms	686
Chapter 16: Encryption	689
Chapter 17: Complexity Theory	692
Chapter 18: Distributed Algorithms	697
Chapter 19: Interview Puzzles	701
Glossary	711
Index	739



Introduction

Algorithms are the recipes that make efficient programming possible. They explain how to sort records, search for items, calculate numeric values such as prime factors, find the shortest path between two points in a street network, and determine the maximum flow of information possible through a communications network. The difference between using a good algorithm and a bad one can mean the difference between solving a problem in seconds, hours, or never.

Studying algorithms lets you build a useful toolkit of methods for solving specific problems. It lets you understand which algorithms are most effective under different circumstances so that you can pick the one best suited for a particular program. An algorithm that provides excellent performance with one set of data may perform terribly with other data, so it is important that you know how to pick the algorithm that is the best match for your scenario.

Even more important, by studying algorithms, you can learn general problem-solving techniques that you can apply to other problems—even if none of the algorithms you already know is a perfect fit for your current situation. These techniques let you look at new problems in different ways so that you can create and analyze your own algorithms to solve your problems and meet unanticipated needs.

In addition to helping you solve problems while on the job, these techniques may even help you land the job where you can use them! Many large technology companies, such as Microsoft, Google, Yahoo!, IBM, and others, want their programmers to understand algorithms and the related problem-solving techniques. Some of these companies are notorious for making job applicants work through algorithmic programming and logic puzzles during interviews.

The better interviewers don't necessarily expect you to solve every puzzle. In fact, they will probably learn more about you when you don't solve a puzzle. Rather than wanting to know the answer, the best interviewers want to see how you approach an unfamiliar problem. They want to see whether you throw up your hands and say the problem is unreasonable in a job interview. Or perhaps you analyze the problem and come up with a promising line of reasoning for using algorithmic approaches to attack the problem. "Gosh, I don't know. Maybe I'd search the Internet," would be a bad answer. "It seems like a recursive divide-and-conquer approach might work" would be a much better answer.

This book is an easy-to-read introduction to computer algorithms. It describes a number of important classical algorithms and tells when each is appropriate. It explains how to analyze algorithms to understand their behavior. Most importantly, it teaches techniques that you can use to create new algorithms on your own.

Here are some of the useful algorithms that this book describes:

- Numerical algorithms, such as randomization, factoring, working with prime numbers, and numeric integration
- Methods for manipulating common data structures, such as arrays, linked lists, trees, and networks
- Using more-advanced data structures, such as heaps, trees, balanced trees, and B-trees
- Sorting and searching
- Network algorithms, such as shortest path, spanning tree, topological sorting, and flow calculations

Here are some of the general problem-solving techniques this book explains:

- Brute-force or exhaustive search
- Divide and conquer
- Backtracking
- Recursion
- Branch and bound
- Greedy algorithms and hill climbing
- Least-cost algorithms
- Constricting bounds
- Heuristics

To help you master the algorithms, this book provides exercises that you can use to explore ways that you can modify the algorithms to apply them to new situations. This also helps solidify the main techniques demonstrated by the algorithms.

Finally, this book includes some tips for approaching algorithmic questions that you might encounter in a job interview. Algorithmic techniques let you solve many interview puzzles. Even if you can't use algorithmic techniques to solve every puzzle, you will at least demonstrate that you are familiar with approaches that you can use to solve other problems.

Why You Should Study Algorithms

There are several reasons why you should study algorithms. First, they provide useful tools that you can use to solve particular problems such as sorting or finding shortest paths. Even if your programming language includes tools to perform tasks that are handled by an algorithm, it's useful to learn how those tools work. For example, understanding how array and list sorting algorithms work may help you decide which of those data structures would work best in your programs.

Algorithms also teach you methods that you may be able to apply to other problems that have a similar structure. They give you a collection of techniques that you can apply to other problems. Techniques such as recursion, divide and conquer, Monte Carlo simulation, linked data structures, network traversal, and others apply to a wide variety of problems.

Perhaps most importantly, algorithms are like a workout for your brain. Just as weight training can help a football or baseball player build muscle, studying algorithms can build your problem-solving abilities. A professional athlete probably won't need to bench press weights during a game. Similarly, you probably won't need to implement a simple sorting algorithm in your project. In both cases, however, practice can help improve your game, whether it's baseball or programming.

Finally, algorithms can be interesting, satisfying, and sometimes surprising. It never ceases to amaze me when I dump a pile of data into a program and a realistic three-dimensional rendering pops out. Even after decades of study, I still feel the thrill of victory when a particularly complicated algorithm produces the correct result. When all of the pieces fit together perfectly to solve an especially challenging problem, it feels like something at least is right in the world.

Algorithm Selection

Each of the algorithms in this book was included for one or more of the following reasons:

- The algorithm is useful, and a seasoned programmer should be expected to understand how it works and how to use it correctly in programs.

- The algorithm demonstrates important algorithmic programming techniques that you can apply to other problems.
- The algorithm is commonly studied by computer science students, so the algorithm or the techniques it uses could appear in a technical interview.

After reading this book and working through the exercises, you will have a good foundation in algorithms and techniques that you can use to solve your own programming problems.

Who This Book Is For

This book is intended primarily for three kinds of readers: professional programmers, programmers preparing for job interviews, and programming students.

Professional programmers will find the algorithms and techniques described in this book useful for solving problems they face on the job. Even when you encounter a problem that isn't directly addressed by an algorithm in this book, reading about these algorithms will give you new perspectives from which to view problems so that you can find new solutions.

Programmers preparing for job interviews can use this book to hone their algorithmic skills. Your interviews may not include any of the problems described in this book, but they may contain questions that are similar enough so that you can use the techniques you learned in this book to solve them. Even if you can't solve a problem, if you recognize a structure similar to those used in one of the algorithms, you can suggest similar strategies and perhaps get partial credit.

For all the reasons explained in the earlier section "Why You Should Study Algorithms," all programming students should study algorithms. Many of the approaches described in this book are simple, elegant, and powerful, but they're not all obvious, so you won't necessarily stumble across them on your own. Techniques such as recursion, divide and conquer, branch and bound, and using well-known data structures are essential to anyone who has an interest in programming.

NOTE Personally, I think algorithms are just plain fun! They're my equivalent of crossword puzzles or Sudoku. I love the feeling of successfully assembling a complicated algorithm and watching it work.

They also make great conversation starters at parties. "What do you think about label setting versus label-correcting, shortest path algorithms?"

Getting the Most Out of This Book

You can learn some new algorithms and techniques just by reading this book, but to really master the methods demonstrated by the algorithms, you need to work with them. You need to implement them in some programming language. You also need to experiment, modify the algorithms, and try new variations on old problems. The book's exercises and interview questions can give you ideas for new ways to use the techniques demonstrated by the algorithms.

To get the greatest benefit from the book, I highly recommend that you implement as many of the algorithms as possible in your favorite programming language or even in more than one language to see how different languages affect implementation issues. You should study the exercises and at least write down outlines for solving them. Ideally, you should implement them, too. Often there's a reason why an exercise is included, and you may not discover it until you take a hard look at the problem. The exercises may lead you down paths that are very interesting but that are too long to squeeze into the book.

Finally, look over some of the other interview questions available on the Internet and figure out how you would approach them. In many interviews, you won't be required to implement a solution, but you should be able to sketch out solutions. And if you have time to implement solutions, you will learn even more.

Understanding algorithms is a hands-on activity. Don't be afraid to put down the book, break out a compiler, and write some actual code!

This Book's Websites

Actually, this book has two websites: Wiley's version and my version. Both sites contain the book's source code.

The Wiley web page for this book is www.wiley.com/go/essentialalgorithms. You also can go to www.wiley.com and search for the book by title or ISBN. Once you've found the book, click the Downloads tab to obtain all of the source code for the book. Once you download the code, just decompress it with your favorite compression tool.

NOTE At the Wiley website, you may find it easiest to search by ISBN. This book's ISBN is 978-1-119-57599-3.

The C# programs are named with a Pascal case naming convention. For example, the program that displays graphical solutions to the Tower of Hanoi puzzle for Exercise 4 in Chapter 9 is named `GraphicalTowerOfHanoi`. The corresponding Python programs are named with underscore casing as in `graphical_tower_of_hanoi.py`.

To find my web page for this book, go to <http://www.CSharpHelper.com/algorithms2e.html>.

How This Book Is Structured

This section describes the book's contents in detail.

Chapter 1, "Algorithm Basics," explains concepts you must understand to analyze algorithms. It discusses the difference between algorithms and data structures, introduces Big O notation, and describes times when practical considerations are more important than theoretical runtime calculations.

Chapter 2, "Numerical Algorithms," explains several algorithms that work with numbers. These algorithms randomize numbers and arrays, calculate greatest common divisors and least common multiples, perform fast exponentiation, and determine whether a number is prime. Some of the algorithms also introduce the important techniques of adaptive quadrature and Monte Carlo simulation.

Chapter 3, "Linked Lists," explains linked-list data structures. These flexible structures can be used to store lists that may grow, shrink, and change in structure over time. The basic concepts are also important for building other linked data structures, such as trees and networks.

Chapter 4, "Arrays," explains specialized array algorithms and data structures, such as triangular and sparse arrays, which can save a program time and memory.

Chapter 5, "Stacks and Queues," explains algorithms and data structures that let a program store and retrieve items in first-in, first-out (FIFO) or last-in, first-out (LIFO) order. These data structures are useful in other algorithms and can be used to model real-world scenarios such as checkout lines at a store.

Chapter 6, "Sorting," explains sorting algorithms that demonstrate a wide variety of useful algorithmic techniques. Different sorting algorithms work best for different kinds of data and have different theoretical run times, so it's good to understand an assortment of these algorithms. These are also some of the few algorithms for which exact theoretical performance bounds are known, so they are particularly interesting to study.

- Chapter 7, “Searching,”** explains algorithms that a program can use to search sorted lists. These algorithms demonstrate important techniques such as binary subdivision and interpolation.
- Chapter 8, “Hash Tables,”** explains hash tables—data structures that use extra memory to allow a program to locate specific items very quickly. They powerfully demonstrate the space-time trade-off that is so important in many programs.
- Chapter 9, “Recursion,”** explains recursive algorithms—those that call themselves. Some problems are naturally recursive, so these techniques make solving them easier. Unfortunately, recursion can sometimes lead to problems, so this chapter also describes how to remove recursion from an algorithm when necessary.
- Chapter 10, “Trees,”** explains highly recursive tree data structures, which are useful for storing, manipulating, and studying hierarchical data. Trees also have applications in unexpected places, such as evaluating arithmetic expressions.
- Chapter 11, “Balanced Trees,”** explains trees that remain balanced as they grow over time. In general, tree structures can grow very tall and thin, and that can ruin the performance of tree algorithms. Balanced trees solve this problem by ensuring that a tree doesn’t grow too tall and skinny.
- Chapter 12, “Decision Trees,”** explains algorithms that attempt to solve problems that can be modeled as a series of decisions. These algorithms are often used on very hard problems, so they often find only approximate solutions rather than the best solution possible. However, they are very flexible and can be applied to a wide range of problems.
- Chapter 13, “Basic Network Algorithms,”** explains fundamental network algorithms such as visiting all the nodes in a network, detecting cycles, creating spanning trees, and finding paths through a network.
- Chapter 14, “More Network Algorithms,”** explains more network algorithms, such as topological sorting to arrange dependent tasks, graph coloring, network cloning, and assigning work to employees.
- Chapter 15, “String Algorithms,”** explains algorithms that manipulate strings. Some of these algorithms, such as searching for substrings, are built into tools that most programming languages can use without customized programming. Others, such as parenthesis matching and finding string differences, require some extra work and demonstrate useful techniques.
- Chapter 16, “Cryptography,”** explains how to encrypt and decrypt information. It covers the basics of encryption and describes several interesting encryption techniques, such as Vigenère ciphers, block ciphers, and public key