How to Think about Algorithms

Second Edition

Understand algorithms and their design with this revised student-friendly textbook. Unlike other algorithms books, this one is approachable, the methods it explains are straightforward, and the insights it provides are numerous and valuable. Without grinding through lots of formal proof, students will benefit from step-by-step methods for developing algorithms, expert guidance on common pitfalls, and an appreciation of the bigger picture. Revised and updated, this second edition includes a new chapter on machine learning algorithms, and concise key concept summaries at the end of each part for quick reference. Also new to this edition are more than 150 new exercises: selected solutions are included to let students check their progress, while a full solutions manual is available online for instructors. No other text explains complex topics such as loop invariants as clearly, helping students to think abstractly and preparing them for creating their own innovative ways to solve problems.

Jeff Edmonds is Professor in the Department of Electrical Engineering and Computer Science at York University, Canada.

"Jeff Edmonds' *How to Think about Algorithms* offers a fresh perspective, placing methodical but intuitive design principles (pre- and post-conditions, invariants, 'transparent' correctness) as the bedrock on which to build and practice algorithmic thinking. The book reads like an epic guided meditation on the vast universe of algorithms, directing the reader's attention to the core of each insight, while stimulating the mind through well-paced examples, playful but concise analogies, and thought-provoking exercises."

Nathan Chenette, *Rose-Hulman Institute of Technology*

"With a good book like this in your hands, learning about algorithms and getting programs to work well will be fun and empowering. Anybody who wants to be a good programmer will get a great deal from this surprisingly readable book. Its approach makes it perfect for reading on your own if you want to enjoy learning about algorithms without being distracted by heavy maths. It has lots of exercises that are worth doing. Most importantly, *How to Think about Algorithms* does just that: it shows you how to think about algorithms and become a better programmer. Knowing how to think about algorithms gives you the insights and skills to make computers do anything more reliably and faster. The book is also ideal for any taught university course, because it is self-contained and systematically sets out the essential material, but most importantly because it empowers students to think for themselves."

Harold Thimbleby, *Swansea University*

How to Think about Algorithms

Second Edition

JEFF EDMONDS
York University, Toronto

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

Shaftesbury Road, Cambridge CB2 8EA, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre, New Delhi – 110025, India

103 Penang Road, #05–06/07, Visioncrest Commercial, Singapore 238467

Cambridge University Press is part of Cambridge University Press & Assessment,
a department of the University of Cambridge.

We share the University's mission to contribute to society through the pursuit of
education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/highereducation/isbn/9781009302142

DOI: 10.1017/9781009302180

First edition published 2008
Second edition published 2024

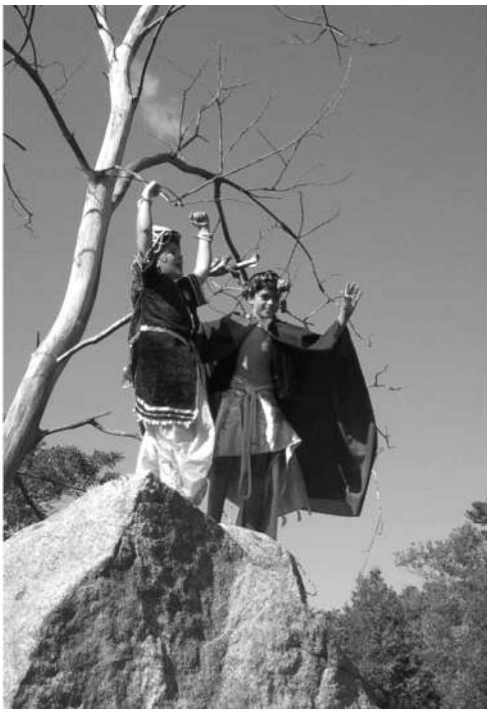
Printed in the United Kingdom by TJ Books Limited, Padstow, Cornwall, 2024

A catalogue record for this publication is available from the British Library

A Cataloging-in-Publication data record for this book is available from the Library of Congress

ISBN 978-1-009-30214-2 Hardback
ISBN 978-1-009-30213-5 Paperback

Additional resources for this publication at www.cambridge.org/Edmonds2e



Dedicated to my siblings, Jennifer, Martin, Alex, and Laura, and to my children, Josh and Micah.

May the love and the mathematics continue to flow between the generations.



Problem Solving
Out of the Box Leaping
Deep Thinking
Creative Abstracting
Logical Deducing
with Friends Working
Fun Having
Fumbling and Bumbling
Bravely Persevering
Joyfully Succeeding


Contents

Preface *page* xiii

Introduction 1

Part I Iterative Algorithms and Loop Invariants

1 Iterative Algorithms: Measures of Progress and Loop Invariants 5
1.1 A Paradigm Shift: A Sequence of Actions vs. a Sequence of Assertions 5
1.2 The Steps to Develop an Iterative Algorithm 9
1.3 More about the Steps 13
1.4 Different Types of Iterative Algorithms 21
1.5 Code from Loop Invariants 28
1.6 Typical Errors 31
1.7 Exercises 32

2 Examples Using More-of-the-Input Loop Invariants 33
2.1 Coloring the Plane 33
2.2 Deterministic Finite Automaton 35
2.3 More of the Input vs. More of the Output 42

3 Abstract Data Types 47
3.1 Specifications and Hints at Implementations 47
3.2 Link List Implementation 55
3.3 Merging with a Queue 61
3.4 Parsing with a Stack 62

4 Narrowing the Search Space: Binary Search 64
4.1 Binary Search Trees 64
4.2 Magic Sevens 66
4.3 VLSI Chip Testing 68
4.4 Exercises 72

5 Iterative Sorting Algorithms 74
5.1 Bucket Sort by Hand 74

5.2 Counting Sort (a Stable Sort) 75
5.3 Radix Sort 78

6 More Iterative Algorithms 80
6.1 Euclid's GCD Algorithm 80
6.2 Multiplying 84

7 The Loop Invariant for Lower Bounds 88

8 Key Concepts Summary: Loop Invariants and Iterative Algorithms 97
8.1 Loop Invariants and Iterative Algorithms 97
8.2 System Invariants 99

9 Additional Exercises: Part I 102

10 Partial Solutions to Additional Exercises: Part I 124

Part II Recursion

11 Abstractions, Techniques, and Theory 133
11.1 Thinking about Recursion 133
11.2 Looking Forward vs. Backward 134
11.3 With a Little Help from Your Friends 135
11.4 The Towers of Hanoi 138
11.5 Checklist for Recursive Algorithms 139
11.6 The Stack Frame 144
11.7 Proving Correctness with Strong Induction 146

12 Some Simple Examples of Recursive Algorithms 149
12.1 Sorting and Selecting Algorithms 149
12.2 Operations on Integers 157
12.3 Ackermann's Function 162
12.4 Fast Fourier Transformations 163
12.5 Exercise 168

13 Recursion on Trees 169
13.1 Tree Traversals 174
13.2 Simple Examples 177
13.3 Heap Sort and Priority Queues 180
13.4 Representing Expressions with Trees 187

14 Recursive Images 192
14.1 Drawing a Recursive Image from a Fixed Recursive and a Base Case Image 192
14.2 Randomly Generating a Maze 195

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Frontmatter
More Information

15 Parsing with Context-Free Grammars 198

16 Key Concepts Summary: Recursion 208

17 Additional Exercises: Part II 211

18 Partial Solutions to Additional Exercises: Part II 230

Part III Optimization Problems

19 Definition of Optimization Problems 241

20 Graph Search Algorithms 243
20.1 A Generic Search Algorithm 243
20.2 Breadth-First Search for Shortest Paths 248
20.3 Dijkstra's Shortest-Weighted-Path Algorithm 253
20.4 Depth-First Search 259
20.5 Recursive Depth-First Search 263
20.6 Linear Ordering of a Partial Order 264
20.7 Exercise 267

21 Network Flows and Linear Programming 268
21.1 A Hill-Climbing Algorithm with a Small Local Maximum 270
21.2 The Primal–Dual Hill-Climbing Method 276
21.3 The Steepest-Ascent Hill-Climbing Algorithm 284
21.4 Linear Programming 288
21.5 Exercises 293

22 Greedy Algorithms 294
22.1 Abstractions, Techniques, and Theory 294
22.2 Examples of Greedy Algorithms 307
22.3 Exercises 320

23 Recursive Backtracking 321
23.1 Recursive Backtracking Algorithms 321
23.2 The Steps in Developing a Recursive Backtracking 325
23.3 Pruning Branches 329
23.4 Satisfiability 331
23.5 Exercises 334

24 Dynamic Programming Algorithms 336
24.1 Start by Developing a Recursive Backtracking Algorithm 336
24.2 The Steps in Developing a Dynamic Programming Algorithm 340
24.3 Subtle Points 346

24.4 The Longest-Common-Subsequence Problem 364
24.5 Dynamic Programs as More-of-the-Input Iterative Loop Invariant Algorithms 368
24.6 A Greedy Dynamic Program: The Weighted Job/Event Scheduling Problem 371

25 Designing Dynamic Programming Algorithms via Reductions 375

26 The Game of Life 380
26.1 Graph ***G*** from Computation 380
26.2 The Graph of Life 382
26.3 Examples of the Graph of Life 385

27 Solution Is a Tree 390
27.1 The Solution Viewed as a Tree: Chains of Matrix Multiplications 390
27.2 Generalizing the Problem Solved: Best AVL Tree 395
27.3 All Pairs Using Matrix Multiplication 397
27.4 Parsing with Context-Free Grammars 398

28 Reductions and NP-Completeness 402
28.1 Satisfiability Is at Least as Hard as Any Optimization Problem 404
28.2 Steps to Prove NP-Completeness 407
28.3 Example: 3-Coloring Is NP-Complete 415
28.4 An Algorithm for Bipartite Matching Using the Network Flow Algorithm 419

29 Randomized Algorithms 423
29.1 Using Randomness to Hide the Worst Cases 423
29.2 Solutions of Optimization Problems with a Random Structure 427

30 Machine Learning 431

31 Key Concepts Summary: Greedy Algorithms and Dynamic Programming 439
31.1 Greedy Algorithms 439
31.2 Dynamic Programming 444

32 Additional Exercises: Part III 454
32.1 Graph Algorithms 454
32.2 Greedy Algorithms 457
32.3 Dynamic Programming 465
32.4 Reductions and NP-Completeness 476

33 Partial Solutions to Additional Exercises: Part III 482
33.1 Graph Algorithms 482

33.2 Greedy Algorithms 482
33.3 Dynamic Programming 485
33.4 Reductions and NP-Completeness 492

Part IV Additional Topics

34 Existential and Universal Quantifiers 499

35 Time Complexity 508
35.1 The Time (and Space) Complexity of an Algorithm 508
35.2 The Time Complexity of a Computational Problem 513

36 Logarithms and Exponentials 515

37 Asymptotic Growth 518
37.1 Steps to Classify a Function 519
37.2 More about Asymptotic Notation 525

38 Adding-Made-Easy Approximations 529
38.1 The Technique 530
38.2 Some Proofs for the Adding-Made-Easy Technique 534

39 Recurrence Relations 540
39.1 The Technique 540
39.2 Some Proofs 543

40 A Formal Proof of Correctness 549

41 Additional Exercises: Part IV 551
41.1 Existential and Universal Quantifiers 551
41.2 Time Complexity 553
41.3 Asymptotic Growth 554
41.4 Adding Made-Easy Approximations 554

42 Partial Solutions to Additional Exercises: Part IV 556
42.1 Existential and Universal Quantifiers 556
42.2 Time Complexity 560

Exercise Solutions 561

Conclusion 588

Index 589

Preface

This book is designed to be used in a twelve-week, third-year algorithms course. The goal is to teach students to think abstractly about algorithms and about the key algorithmic techniques used to develop them.

Meta-Algorithms: Students must learn so many algorithms that they are sometimes overwhelmed. In order to facilitate their understanding, most textbooks cover the standard themes of iterative algorithms, recursion, greedy algorithms, and dynamic programming. Generally, however, when it comes to presenting the algorithms themselves and their proofs of correctness, the concepts are hidden within optimized code and slick proofs. One goal of this book is to present a uniform and clean way of thinking about algorithms. We do this by focusing on the structure and proof of correctness of *iterative* and *recursive* meta-algorithms, and within these the *greedy* and *dynamic programming* meta-algorithms. By learning these and their proofs of correctness, most actual algorithms can be easily understood. The challenge is that thinking about meta-algorithms requires a great deal of abstract thinking.

Abstract Thinking: Students are very good at learning how to apply a concrete code to a concrete input instance. They tend, however, to find it difficult to think abstractly about the algorithms. I maintain that the more abstractions a person has from which to view the problem, the deeper their understanding of it will be, the more tools they will have at their disposal, and the better prepared they will be to design their own innovative ways to solve new problems. Hence, I present a number of different notations, analogies, and paradigms within which to develop and to think about algorithms.

Levels: The psychological profiling of a successful person is mostly the ability to shift levels of abstraction.
To understand the detailed workings.
To understand the big picture.
To understand complex things in simple ways.

Way of Thinking: People who develop algorithms have various ways of thinking and intuition that tend not to get taught. The assumption, I suppose, is that these cannot be taught but must be figured out on one's own. This text attempts to teach students to think like a designer of algorithms.

Not a Reference Book: My intention is not to teach a specific selection of algorithms for specific purposes. Hence, the book is not organized according to the application of the algorithms, but according to the techniques and abstractions used to develop them.

Developing Algorithms: The goal is not to present completed algorithms in a nice clean package, but to go slowly through every step of the development. Many false starts have been added. The hope is that this will help students learn to develop algorithms on their own. The difference is a bit like the difference between studying carpentry by looking at houses and by looking at hammers.

Proof of Correctness: Our philosophy is not to follow an algorithm with a formal proof that it is correct. Instead, this text is about learning how to think about, develop, and describe algorithms in such way that their correctness is transparent.

Big Picture vs. Small Steps: For each topic, I attempt both to give the big picture and to break it down into easily understood steps.

Level of Presentation: This material is difficult. There is no getting around that. I have tried to figure out where confusion may arise and to cover these points in more detail. I try to balance the succinct clarity that comes with mathematical formalism against the personified analogies and metaphors that help to provide both intuition and humor.

Point Form: The text is organized into blocks, each containing a title and a single thought. Hopefully, this will make the text easier to lecture and study from.

Prerequisites: The text assumes that the students have completed a first-year programming course and have a general mathematical maturity. The Part IV, Additional Topics, covers much of the mathematics that will be needed.

Homework Questions: A few additional questions are included. I am hoping to develop many more, along with their solutions. Contributions are welcome.

Read Ahead: The student is expected to read the material *before* attending lectures or classes. This will facilitate productive discussion during class.

Explaining: To be able to prove yourself on a test or on the job, you need to be able to explain the material well. In addition, explaining it to someone else is the best way to learn it yourself. Hence, I highly recommend spending a lot of time explaining the material over and over again out loud to yourself, to each other, and to your stuffed bear.

Dreaming: I would like to emphasis the importance of thinking, even daydreaming, about the material. This can be done while going through your day – while swimming, showering, cooking, or lying in bed. Ask questions. Why is it done this way and not that way? Invent other algorithms for solving a problem. Then look for input instances for which your algorithm gives the wrong answer. Mathematics is not all linear thinking. If the essence of the material, what the questions are really asking, is allowed to seep down into your subconscious then with time little thoughts will begin to percolate up. Pursue these ideas. Sometimes even flashes of inspiration appear.

Acknowledgments

I would like to thank Andy Mirzaian, Franck van Breugel, James Elder, Suprakash Datta, Eric Ruppert, Russell Impagliazzo, Toniann Pitassi, and Kirk Pruhs, with whom I co-taught and co-researched algorithms for many years. I would like to thank Jennifer Wolfe, Lauren Cowles, Julie Lancashire, Anna Scriven, Rachel Norridge, and Beth Morel for their fantastic editing jobs. All of these people were a tremendous support for this work.