Incipit prologus in libro alghoarismi de practica arismetrice.

— Ioannis Hispalensis [John of Seville?],
Liber algorismi de pratica arismetrice (c.1135)

Shall I tell you, my friend, how you will come to understand it?
Go and write a book upon it.

— Henry Home, Lord Kames (1696–1782),
in a letter to to Sir Gilbert Elliot

The individual is always mistaken. He designed many things, and drew in other persons as coadjutors, quarrelled with some or all, blundered much, and something is done; all are a little advanced, but the individual is always mistaken. It turns out somewhat new and very unlike what he promised himself.

— Ralph Waldo Emerson, "Experience", *Essays, Second Series* (1844)

What I have outlined above is the content of a book the realization of whose basic plan and the incorporation of whose details would perhaps be impossible; what I have written is a second or third draft of a preliminary version of this book

— Michael Spivak, preface of the first edition of
Differential Geometry, Volume I (1970)

Preface

About This Book

This textbook grew out of a collection of lecture notes that I wrote for various algorithms classes at the University of Illinois at Urbana-Champaign, which I have been teaching about once a year since January 1999. Spurred by changes of our undergraduate theory curriculum, I undertook a major revision of my notes in 2016; this book consists of a subset of my revised notes on the most fundamental course material, mostly reflecting the algorithmic content of our new required junior-level theory course.

Prerequisites

The algorithms classes I teach at Illinois have two significant prerequisites: a course on discrete mathematics and a course on fundamental data structures. Consequently, this textbook is probably not suitable for most students as a *first*

course in data structures and algorithms. In particular, I assume at least passing familiarity with the following specific topics:

- **Discrete mathematics:** High-school algebra, logarithm identities, naive set theory, Boolean algebra, first-order predicate logic, sets, functions, equivalences, partial orders, modular arithmetic, recursive definitions, trees (as abstract objects, not data structures), graphs (vertices and edges, not function plots).
- **Proof techniques:** direct, indirect, contradiction, exhaustive case analysis, and induction (especially "strong" and "structural" induction). Chapter o uses induction, and whenever Chapter $n-1$ uses induction, so does Chapter n.
- **Iterative programming concepts:** variables, conditionals, loops, records, indirection (addresses/pointers/references), subroutines, recursion. I do not assume fluency in any particular programming language, but I do assume experience with at least one language that supports both indirection and recursion.
- **Fundamental abstract data types:** scalars, sequences, vectors, sets, stacks, queues, maps/dictionaries, ordered maps/dictionaries, priority queues.
- **Fundamental data structures:** arrays, linked lists (single and double, linear and circular), binary search trees, at least one form of *balanced* binary search tree (such as AVL trees, red-black trees, treaps, skip lists, or splay trees), hash tables, binary heaps, and most importantly, the difference between this list and the previous list.
- **Fundamental computational problems:** elementary arithmetic, sorting, searching, enumeration, tree traversal (preorder, inorder, postorder, level-order, and so on).
- **Fundamental algorithms:** elementary algorism, sequential search, binary search, sorting (selection, insertion, merge, heap, quick, radix, and so on), breadth- and depth-first search in (at least binary) trees, and most importantly, the difference between this list and the previous list.
- **Elementary algorithm analysis:** Asymptotic notation (o, O, Θ, Ω, ω), translating loops into sums and recursive calls into recurrences, evaluating simple sums and recurrences.
- **Mathematical maturity:** facility with abstraction, formal (especially recursive) definitions, and (especially inductive) proofs; writing and following mathematical arguments; recognizing and avoiding syntactic, semantic, and/or logical nonsense.

The book *briefly* covers some of this prerequisite material when it arises in context, but more as a reminder than a good introduction. For a more thorough overview, I strongly recommend the following freely available references:

- Margaret M. Fleck. *Building Blocks for Theoretical Computer Science*, unpublished textbook, most recently revised January 2013. Available from http://mfleck.cs.illinois.edu/building-blocks/.
- Eric Lehman, F. Thomson Leighton, and Albert R. Meyer. *Mathematics for Computer Science*, unpublished lecture notes, most recent (public) revision June 2018. Available from https://courses.csail.mit.edu/6.042/spring18/. (I strongly recommend searching for the most recent revision.)
- Pat Morin. *Open Data Structures*, most recently revised January 2016 (edition 0.1Gβ). A free open-content textbook, which Pat maintains and regularly updates. Available from http://opendatastructures.org/.

Additional References

Please do not restrict yourself to this or any other single reference. Authors and readers bring their own perspectives to any intellectual material; no instructor "clicks" with every student, or even with every very strong student. Finding the author that most effectively gets *their* intuition into *your* head takes some effort, but that effort pays off handsomely in the long run.

The following references have been particularly valuable sources of intuition, examples, exercises, and inspiration; this is not meant to be a complete list.

- Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. (I used this textbook as an undergraduate at Rice and again as a masters student at UC Irvine.)
- Thomas Cormen, Charles Leiserson, Ron Rivest, and Cliff Stein. *Introduction to Algorithms*, third edition. MIT Press/McGraw-Hill, 2009. (I used the first edition as a teaching assistant at Berkeley.)
- Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani. *Algorithms*. McGraw-Hill, 2006. (Probably the closest in content to this book, but considerably less verbose.)
- Jeff Edmonds. *How to Think about Algorithms*. Cambridge University Press, 2008.
- Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- Michael T. Goodrich and Roberto Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons, 2002.
- Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2005. Borrow it from the library if you can.
- Donald Knuth. *The Art of Computer Programming*, volumes 1–4A. Addison-Wesley, 1997 and 2011. (My parents gave me the first three volumes for Christmas when I was 14. Alas, I didn't actually read them until *much* later.)

- Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989. (I used this textbook as a teaching assistant at Berkeley.)
- Ian Parberry. *Problems on Algorithms*. Prentice-Hall, 1995 (out of print). Downloadable from https://larc.unt.edu/ian/books/free/license.html after you agree to make a small charitable donation. Please honor your agreement.
- Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley, 2011.
- Robert Endre Tarjan. *Data Structures and Network Algorithms*. SIAM, 1983.
- Class notes from my own algorithms classes at Berkeley, especially those taught by Dick Karp and Raimund Seidel.
- Lecture notes, slides, homeworks, exams, video lectures, research papers, blog posts, and full-fledged MOOCs made freely available on the web by innumerable colleagues around the world.

About the Exercises

Each chapter ends with several exercises, most of which I have used at least once in a homework assignment, discussion/lab section, or exam. The exercises are *not* ordered by increasing difficulty, but (generally) clustered by common techniques or themes. Some problems are annotated with symbols as follows:

- ♥Red hearts indicate particularly challenging problems; many of these have appeared on qualifying exams for PhD students at Illinois. A small number of *really* hard problems are marked with ♥larger hearts, and a few *open* problems are marked with ♥enormous hearts.
- ♦Blue diamonds indicate problems that require familiarity with material from later chapters, but thematically belong where they are. Problems that require familiarity with *earlier* material are not marked, however; the book, like life, is cumulative.
- ♣Green clubs indicate problems that require familiarity with material outside the scope of this book, such as finite-state machines, linear algebra, probability, or planar graphs. These are rare.
- ♠Black spades indicate problems that require a significant amount of grunt work and/or coding. These are rare.
- ★Orange stars indicate that you are eating Lucky Charms that were manufactured before 1998. Ew.

These exercises are designed as opportunities to practice, not as targets for their own sake; the goal of these problems not to solve these particular problems, but to practice exercising a particular skill, or solving a *type* of problem. Partly for this reason, I don't provide solutions to the exercises; the solutions are not the point. In particular, there is no "instructor's manual"; if you can't solve a

problem yourself, you probably shouldn't assign it to your students. That said, you can probably find solutions to whatever homework problems I've assigned *this* semester on the web page of whatever course I'm teaching. And nothing is stopping *you* from writing an instructor's manual!

Steal This Book!

Acknowledgments

This textbook draws heavily on the contributions of countless algorithms students, teachers, and researchers. In particular, I am immensely grateful to more than three thousand Illinois students who have used my lecture notes as a primary reference, offered useful (if sometimes painful) criticism, and suffered through some truly awful early drafts. Thanks also to many colleagues and students around the world who have used these notes in their own classes and have sent helpful feedback and bug reports.

I am particularly grateful for the feedback and contributions (especially exercises) from my amazing teaching assistants:

> Aditya Ramani, Akash Gautam, Alex Steiger, Alina Ene, Amir Nayyeri, Asha Seetharam, Ashish Vulimiri, Ben Moseley, Brad Sturt, Brian Ensink, Chao Xu, Charlie Carlson, Chris Neihengen, Connor Clark, Dan Bullok, Dan Cranston, Daniel Khashabi, David Morrison, Ekta Manaktala, Erin Wolf Chambers, Gail Steitz, Gio Kao, Grant Czajkowski, Hsien-Chih Chang, Igor Gammer, Jacob Laurel, John Lee, Johnathon Fischer, Junqing Deng, Kent Quanrud, Kevin Milans, Kevin Small, Konstantinos Koiliaris, Kyle Fox, Kyle Jao, Lan Chen, Mark Idleman, Michael Bond, Mitch Harris, Naveen Arivazhagen, Nick Bachmair, Nick Hurlburt, Nirman Kumar, Nitish Korula, Patrick Lin, Phillip Shih, Rachit Agarwal, Reza Zamani-Nasab, Rishi Talreja, Rob McCann, Sahand Mozaffari, Shalan Naqvi, Srihita Vatsavaya, Shripad Thite, Spencer Gordon, Subhro Roy, Tana Wattanawaroon, Umang Mathur, Vipul Goyal, Yasu Furakawa, and Yipu Wang.

I've also been helped tremendously by many discussions with faculty colleagues at Illinois: Alexandra Kolla, Cinda Heeren, Edgar Ramos, Herbert Edelsbrunner, Jason Zych, Kim Whittlesey, Lenny Pitt, Madhu Parasarathy, Mahesh Viswanathan, Margaret Fleck, Shang-Hua Teng, Steve LaValle, and especially Chandra Chekuri, Ed Reingold, and Sariel Har-Peled.

Of course this book owes a great debt to the people who taught me this algorithms stuff in the first place: Bob Bixby and Michael Pearlman at Rice; David Eppstein, Dan Hirschberg, and George Lueker at Irvine; and Abhiram Ranade, Dick Karp, Manuel Blum, Mike Luby, and Raimund Seidel at Berkeley.

I stole the first iteration of the overall course structure, and the idea to write up my own lecture notes in the first place, from Herbert Edelsbrunner; the idea of turning a subset of my notes into a book from Steve LaValle; and several components of the book design from Robert Ghrist.

Caveat Lector!

Of course, none of those people should be blamed for any flaws in the resulting book. Despite many rounds of revision and editing, this book contains many

mistakes, bugs, gaffes, omissions, snafus, kludges, typos, mathos, grammaros, thinkos, brain farts, poor design decisions, historical inaccuracies, anachronisms, inconsistencies, exaggerations, dithering, blather, distortions, oversimplification, nonsense, garbage, cruft, junk, and outright lies, **all of which are entirely Steve Skiena's fault.**

I maintain an issue tracker at https://github.com/jeffgerickson/algorithms, where readers like you can submit bug reports, feature requests, and general feedback on the book. Please let me know if you find an error of any kind, whether mathematical, grammatical, historical, typographical, cultural, or otherwise, whether in the main text, in the exercises, or in my other course materials. (Steve is unlikely to care.) Of course, all other feedback is also welcome!

Enjoy!

— Jeff

It is traditional for the author to magnanimously accept the blame for whatever deficiencies remain. I don't. Any errors, deficiencies, or problems in this book are somebody else's fault, but I would appreciate knowing about them so as to determine who is to blame.

— Steven S. Skiena, *The Algorithm Design Manual* (1997)

No doubt this statement will be followed by an annotated list of all textbooks, and why each one is crap.

— Adam Contini, MetaFilter, January 4, 2010