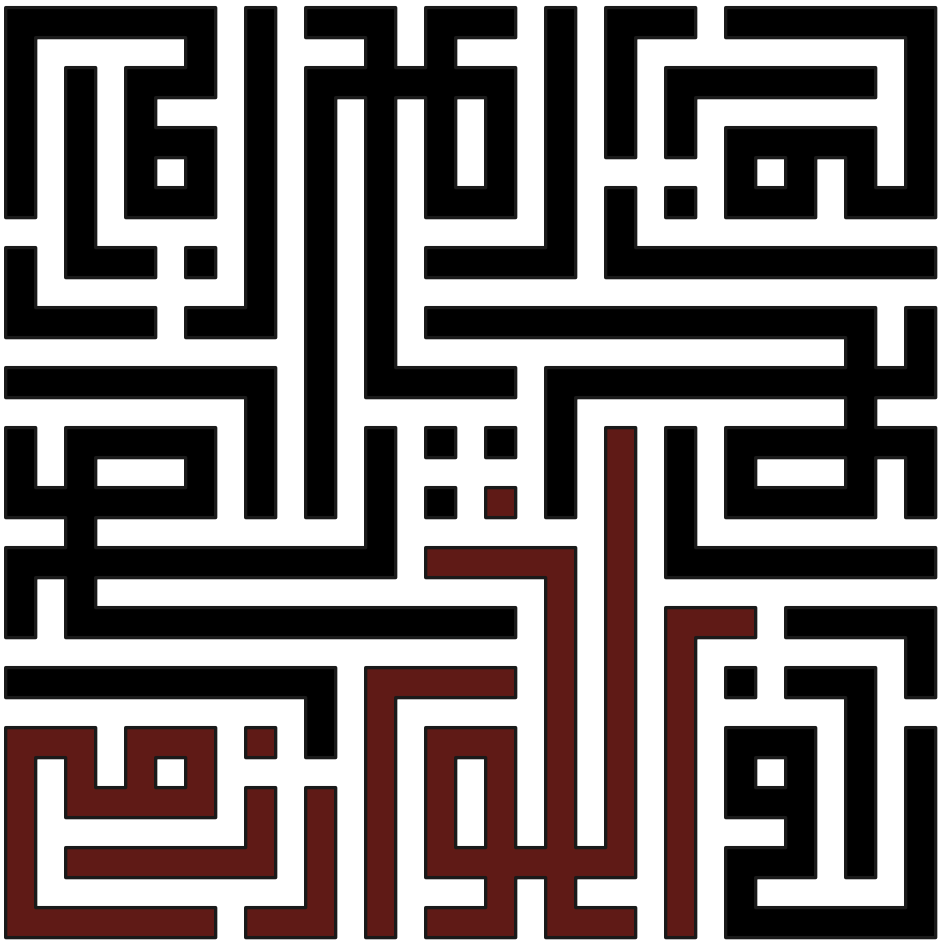
# *Algorithms*

*Jeff Erickson*



December 25, 2013



http://www.cs.illinois.edu/~jeffe/teaching/algorithms/

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

— Henry Home, Lord Kames (1696–1782), 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)

*The very form of the scientific article as it has evolved over the last three centuries normatively requires authors to acknowledge on whose shoulders they stand, whether these be the shoulders of giants or, as is often the case, those of men and women of science of approximately average dimensions for the species* scientificus*.*

— Robert K. Merton, "The Matthew Effect in Science, II" (1988)

## About These Notes

These are lecture notes that I wrote for algorithms classes at the University of Illinois at Urbana-Champaign, which I have taught on average once a year since January 1999. The most recent revision of these notes (or nearly so) is available online at http://www.cs.illinois.edu/~jeffe/teaching/algorithms/, along with a near-complete archive of all my past homeworks and exams. Whenever I teach an algorithms class, I revise, update, and sometimes cull these notes as the course progresses, so you may find more recent versions on the web page of whatever course I am currently teaching.

With few exceptions, each of these "lecture notes" contains far too much material to cover in one lecture. In a typical 75-minute class period, I cover about 4 or 5 pages of material—a bit more if I'm teaching graduate students than undergraduates. Moreover, I can only cover at most two-thirds of these notes in *any* capacity in a single 15-week semester. Your mileage may vary! (Arguably, that means that as I continue to add material, the label "lecture notes" becomes less and less accurate.) The undergraduate and graduate classes I teach cover different but overlapping subsets of this material. The notes are numbered in roughly the order I use them in undergraduate classes, with more advanced material (indicated by *stars) inserted near the more basic material it builds on.

## About The Exercises

Each note ends with several exercises, most of which have been used at least once in a homework assignment, discussion section, or exam. *Stars indicate more challenging problems; many of these starred problems appeared on qualifying exams for the algorithms PhD students at UIUC. A small number of *really* hard problems are marked with a ★larger star; one or two *open* problems are indicated by ★enormous stars. A large fraction of these exercises were contributed by some amazing teaching assistants:

Aditya Ramani, Akash Gautam, Alina Ene, Amir Nayyeri, Asha Seetharam, Ashish Vulimiri, Ben Moseley, Brian Ensink, Chris Neihengen, Dan Bullok, Dan Cranston, Daniel Khashabi, David Morrison, Johnathon Fischer, Ekta Manaktala, Erin Wolf Chambers, Hsien-Chih Chang, Igor Gammer, Gio Kao, Kent Quanrud, Kevin Milans, Kevin Small, Kyle Fox, Kyle Jao, Lan Chen, Michael Bond, Mitch Harris, Nick Hurlburt, Nirman Kumar, Nitish Korula, Rachit Agarwal, Reza Zamani-Nasab, Rishi Talreja, Rob McCann, Shripad Thite, Subhro Roy, and Yasu Furakawa.

***Please do not ask me for solutions to the exercises.*** If you are a student, seeing the solution will rob you of the experience of solving the problem yourself, which is the only way to learn the material. If you are an instructor, you shouldn't assign problems that you can't solve yourself! (Because I don't always follow my own advice, I sometimes assign buggy problems, but I've tried to keep these out of the lecture notes themselves.)

## Acknowledgments

All of this material draws heavily on the creativity, wisdom, and experience of thousands of algorithms students, teachers, and researchers. In particular, I am immensely grateful to the more than 2000 Illinois students who have used these notes as a primary reference, offered useful (if sometimes painful) criticism, and suffered through some truly awful first drafts. I'm also grateful for the contributions and feedback from the teaching assistants listed above. Finally, thanks to many colleagues at Illinois and elsewhere who have used these notes in their classes and have sent helpful feedback and bug reports.

Naturally, these notes owe a great deal to the people who taught me this algorithms stuff in the first place: Bob Bixby and Michael Pearlman at Rice; David Eppstein, Dan Hirshberg, and George Lueker at UC Irvine; and Abhiram Ranade, Dick Karp, Manuel Blum, Mike Luby, and Raimund Seidel at UC Berkeley. I've also been helped tremendously by many discussions with faculty colleagues at Illinois—Cinda Heeren, Edgar Ramos, Herbert Edelsbrunner, Jason Zych, Lenny Pitt, Madhu Parasarathy, Mahesh Viswanathan, Margaret Fleck, Shang-Hua Teng, Steve LaValle, and especially Chandra Chekuri, Ed Reingold, and Sariel Har-Peled. I stole the first iteration of the overall course structure, and the idea to write up my own lecture notes, from Herbert Edelsbrunner.

Finally, "Johnny's" multi-colored crayon homework was found under the TA office door among the other Fall 2000 Homework 1 submissions. The square Kufi rendition of the name "al-Khwārizmī" on the cover is my own.

## Prerequisites

For the most part, these notes assume the reader has mastered the material covered in the first two years of a strong undergraduate computer science curriculum, and that they have the intellectual maturity to recognize and repair any remaining gaps in their mastery. (Mastery is not the same thing as "exposure" or "a good grade", which is why I start every semester with Homework Zero.) In particular, for most students, these notes are *not* suitable for a *first* course in data structures and algorithms. Specific prerequisites include the following:

- 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.
- Proof techniques: direct, indirect, contradiction, exhaustive case analysis, and induction (especially "strong" and "structural" induction). Lecture 0 requires induction, and whenever Lecture $n-1$ requires induction, so does Lecture $n$.
- Elementary discrete probability: uniform vs non-uniform distributions, expectation, conditional probability, linearity of expectation, independence.
- Iterative programming concepts: variables, conditionals, loops, 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 indirection and recursion.

- Fundamental abstract data types: scalars, sequences, vectors, sets, stacks, queues, priority queues, dictionaries.
- Fundamental data structures: arrays, linked lists (single and double, linear and circular), binary search trees, at least one *balanced* binary search tree (AVL trees, red-black trees, treaps, skip lists, splay trees, etc.), binary heaps, hash tables, and most importantly, ***the difference between this list and the previous list***.
- Fundamental algorithmic problems: sorting, searching, enumeration.
- Fundamental algorithms: elementary arithmetic, sequential search, binary search, comparison-based sorting (selection, insertion, merge-, heap-, quick-), radix sort, pre-/post-/inorder tree traversal, breadth- and depth-first search (at least in trees), and most importantly, ***the difference between this list and the previous list***.
- Basic algorithm analysis: Asymptotic notation ($o$, $O$, $\Theta$, $\Omega$, $\omega$), 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.

Two notes on prerequisite material appear as an appendix to the main lecture notes: one on proofs by induction, and one on solving recurrences. The main lecture notes also *briefly* cover some prerequisite material, but more as a reminder than a good introduction. For a more thorough overview, I strongly recommend the following online resources:

- Margaret M. Fleck. *Building Blocks for Theoretical Computer Science*, unpublished textbook, most recently revised January 2013.
- Eric Lehman, F. Thomson Leighton, and Albert R. Meyer. *Mathematics for Computer Science*, unpublished lecture notes, most recently revised January 2013.
- Pat Morin. *Open Data Structures*, most recently revised July 2013 (edition 0.1F). A permanently free open-source textbook, which Pat maintains and regularly updates.

## Additional References

I strongly encourage my students (and other readers) *not* to restrict themselves to a single textual reference. Authors and readers bring their own perspectives to the material; no instructor 'clicks' with every student, or even every very strong student. Finding the author that most effectively gets their intuition into *your* head take some effort, but that effort pays off handsomely in the long run. The following references have been particularly valuable to me as sources of inspiration, intuition, examples, and problems.

- 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. (The second edition was my recommended textbook until 2005; I also used the first edition as a teaching assistant at Berkeley.)

- Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani. *Algorithms*. McGraw-Hill, 2006. (This is the current recommended textbook for my undergraduate algorithms class.)
- 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. (This is the current recommended textbook for my graduate algorithms class.)
- 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, but 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.)
- Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- Ian Parberry. *Problems on Algorithms*. Prentice-Hall, 1995 (out of print). You can download this book from http://www.eng.unt.edu/ian/books/free/license.html after promising to make a small charitable donation. Please honor your promise.
- Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003. Just in case you thought Knuth was the only author who could stun oxen.
- Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley, 2011. (This book and its prequels have by far the best algorithm *illustrations* I've seen anywhere, in part both the examples and the printed pseudocode are compiled from the same source.)
- Robert Endre Tarjan. *Data Structures and Network Algorithms*. SIAM, 1983.
- Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 2001.
- 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.

### Caveat Lector!

Despite several rounds of revision, these notes still contain mnay mistakes, errors, bugs, gaffes, omissions, snafus, kludges, typos, mathos, grammaros, thinkos, brain farts, nonsense, garbage, cruft, junk, and outright lies, ***all of which are entirely Steve Skiena's fault.*** I revise and update these notes every time I teach the course, so please let me know if you find a bug. (Steve is unlikely to care.) I regularly award extra credit to students who post explanations and/or corrections of errors in the lecture notes. If I'm not teaching your class, encourage your instructor to set up a similar extra-credit scheme, and forward the bug reports to ~~Steve~~ me!

Of course, any 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)