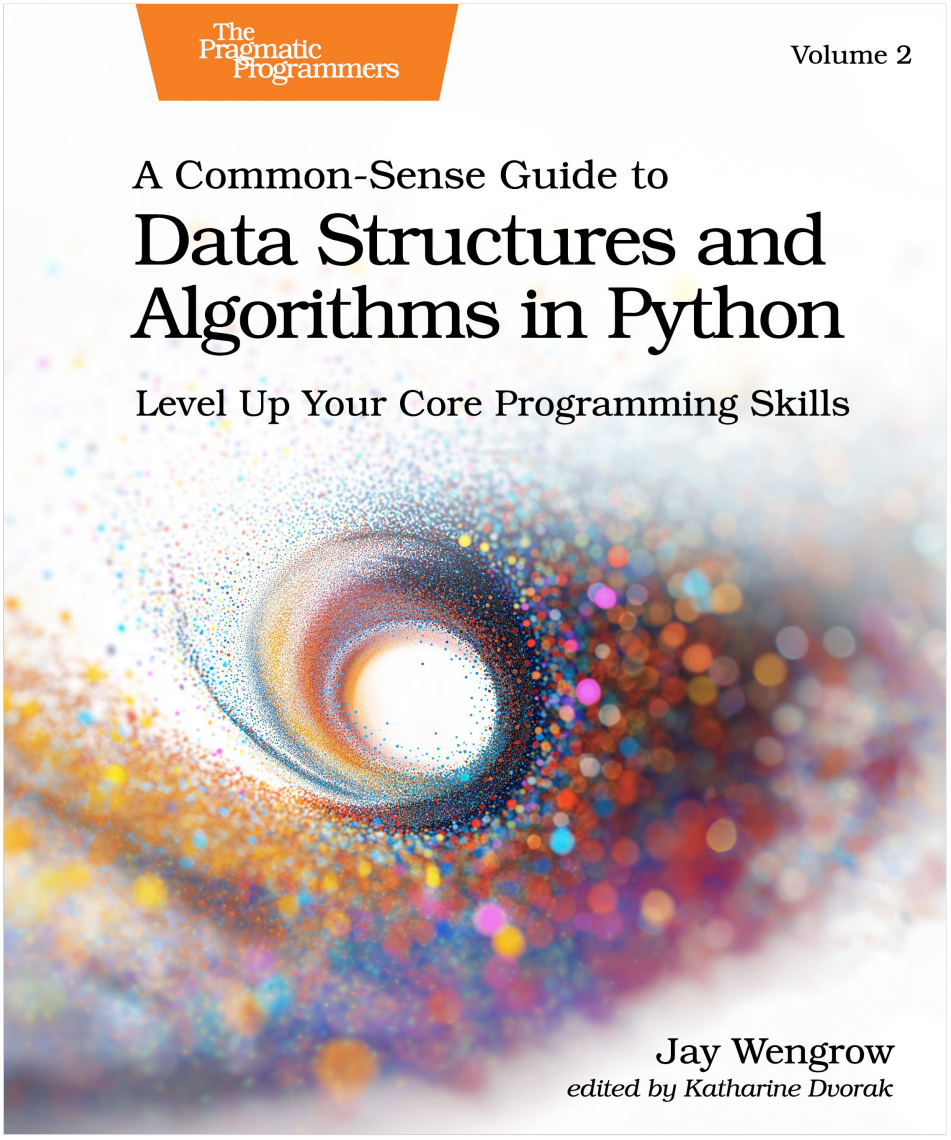

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Preface

The days of slow software are long gone. People expect their technology to be fast, and I mean *really* fast. According to one study, more than half of users of mobile websites will abandon a site if it takes more than three seconds to load. Another study reports that for each second that a web page takes to load, user satisfaction goes down by 16%.

At the same time, people also want their technology to fit on even the smallest of devices. Whether it's a smartphone that fits easily into your pocket or a smartwatch that fits on your wrist, people want to bring their computers wherever they go. And they don't want these devices to be any less capable than a full-fledged desktop computer. However, to get apps to work on such small devices, those apps must consume the least memory possible.

The key to writing software that is both fast and memory-efficient is the mastery of data structures and algorithms. Designing the "right" algorithm usually involves a combination of knowledge, ingenuity, and persistence. This book is where you'll gain the knowledge.

As you may have gathered from the title, this volume is the second in a series. In *A Common-Sense Guide to Data Structures and Algorithms in Python, Volume 1*, I covered the foundational concepts. If you've read that volume or already know the concepts therein, you already know a lot! But now it's time to level up. Not only will you learn a wide variety of new algorithms and data structures, but you'll also become more sophisticated in algorithmic analysis and design.

Now, there are quite a few books that have already been written on these subjects. But if you've read any of them, you may have encountered the same problem I have: they're really hard to understand! It's not just you—I can find myself reading and rereading the same paragraph in such a book many times before I get an inkling of what's going on. Many a developer has given up trying to learn these concepts, feeling incapable of grasping such complex ideas.

But here's the thing. Yes, these concepts are complex. But every complex concept is made up of a combination of *simple* concepts. It's not beyond anyone to master data structures and algorithms. The subject just needs to be *taught* correctly.

And that's the entire point of this book. I've pulled out each thread from the tapestry of each complex concept and laid them all out for you. By presenting each thread individually and in the correct sequence, I'll teach you these ideas so that you'll grasp them easily and clearly. Critically, there are literally hundreds of diagrams, all clear and beautiful, that will help clarify each concept for the visual learner. Finally, you'll have a lot of fun along the way. I write in an informal style, and crack jokes whenever my editor will let me. Sometimes, the jokes are even funny.

Who Is This Book For?

You may be a professional software engineer of any experience level, a budding computer science student, or a code hobbyist. No matter what box you fit into, if you already know the basics of data structures and algorithms and want to level up, this is the book you're looking for.

If you've already read Volume 1 of this book, you're ready to dive into this book. (And welcome back!)

If you *haven't* read Volume 1, I won't take it personally. However, you *do* need to be pretty familiar with certain topics if you want to comprehend this volume. These topics, in alphabetical order, are:

- Arrays
- Big O notation for time and space complexity
- Binary search
- Binary search trees
- Hash tables
- Heaps
- Recursion
- Sorting algorithms (the gist of Selection Sort and Quicksort)

In any case, I make many direct references to Volume 1 throughout this volume so you can always gain more context if you need to.

What's in This Book?

In this volume, I don't simply cover some laundry list of additional data structures and algorithms. Yes, I cover those too, but the main focus of this

book is to help you become more proficient with your analysis and design of algorithms.

Specifically, you'll find that there are three main themes that are threaded throughout this volume:

1. *Going beyond Big O.* While Big O notation is a useful, and even crucial tool, for algorithmic analysis, it has some significant limitations when applied to the real world. I'll show you where Big O notation falls short, and how to use benchmarking and other forms of analysis to ensure that your code will truly be efficient in real life.

2. *Randomization.* There are many components you can integrate into an algorithm you're designing. One of the most useful, but perhaps surprising, of these components is randomness. We begin with the basics of randomization algorithms, and then see how they can make your code more efficient in a wide variety of scenarios.

3. *Hardware.* An all-too-often overlooked factor in algorithm design is how your computer's hardware setup can impact the efficiency of your code. Sure, in an academic vacuum, how much memory your computer has shouldn't affect the Big O classification of an algorithm. But in truth, a computer's hardware can have a *huge* impact on how fast your code will perform when you actually run it.

With regard to specific data structures and algorithms, here's a list of some of them that you'll encounter. I've listed them in the order in which they're presented in the book:

- Mergesort
- Fisher-Yates Shuffle
- Load balancing with the power of two choices
- LRU caches
- Red-black trees
- Randomized treaps
- External-memory algorithms
- B-trees
- Merging K sorted lists
- M/B-Way Mergesort
- Monte Carlo algorithms
- Random sampling
- Fermat's Primality Test
- Randomized hashing
- Hash function families

- Rabin-Karp substring search
- Sliding-window technique
- Boolean arrays
- Bit vectors
- Bit manipulation
- Bloom filters

In addition, each chapter contains exercises that will help you practice everything you've learned. Solutions to each exercise can be found at the back of the book.

A handful of exercises are "special." You'll see that I marked some as being a "Puzzle" or "Exploration" or the like. You aren't expected to know the answer even if you've mastered the chapter. Rather, they ask you to apply ingenuity and see if you can stretch yourself even further. Occasionally, I'll also mark an exercise as a "New Concept" if I'll be teaching a brand new idea in the associated solution.

How to Read This Book

In short, this book was designed to be read in order.

This isn't a book containing a hodgepodge of topics. I've painstakingly built up the concepts so that each chapter builds upon the previous one, and there's a kind of "story line" arc that carries you through the book.

You *might* be able to skip to a chapter of interest, but you run the risk of not catching every bit of nuance, since I'll have assumed that you read the previous chapters. Of course, after you've read the book once, you should be able to reread any chapter you wish for reference.

The one exception is Chapters 7 and 8, which are a bit of a "side quest." You could technically skip those and come back to them later if you'd like.

A Note About the Code

I strived to follow PEP 8 standards (for the most part) and write the code in such a way so that it runs on Python Version 3.

That being said, I want to emphasize that the concepts in this book apply to virtually all coding languages, and I expect that some people not as familiar with Python will be reading this book. Because of this, I've sometimes avoided certain Python idioms where I thought that this would utterly confuse people coming from other languages. It's a tricky balance to keep the code Pythonic

while also being welcoming to non-Python coders, but I hope that I've maintained an equilibrium that satisfies most readers.

Virtually all the code in this book can be downloaded online from the book's web page.[1] This code repository contains automated tests too! I encourage you to check out the tests since they don't appear in the actual book, and can help clarify how to run the main modules.

You'll find some longer code snippets under the headings that read "Code Implementation." I certainly encourage you to study these code samples, but you don't necessarily have to understand every last line to proceed to the next section of the book. If these long pieces of code are bogging you down, just skim (or skip) them for now.

Finally, it's important to note that the code in this book is not "production ready." My greatest focus has been to clarify the concept at hand, and while I did also try to make the code generally complete, I have not accounted for every edge case and optimization. There's certainly room for you to optimize the code further—so feel free to go crazy with that.

Online Resources

You can find more information about the book, download the source code for the code examples, and help improve the book by reporting errata, typos, and content suggestions on the book's web page[2] on pragprog.com.

Additionally, I post updates about my writing at my website.[3] There you can find more information about my books as well as video tutorials made by my colleagues and me in which we use the "common-sense" approach to explaining all sorts of technologies and concepts. In particular, you might enjoy my "Jay Vs. Leetcode"[4] video series where I solve programming puzzles using many of the techniques discussed in this book.

Connecting

I enjoy connecting with my readers and invite you to find me on LinkedIn.[5] I'd gladly accept your connection request—just send a message that you're a reader of this book. I look forward to hearing from you!

1. https://pragprog.com/titles/jwpython2
2. https://pragprog.com/titles/jwpython2
3. https://commonsensedev.com
4. https://www.commonsensedev.com/jay-vs-leetcode
5. https://www.linkedin.com/in/jaywengrow