Preface

This book serves as a textbook for a sophomore-level course in data structures and algorithms. We assume students have had the equivalent of a year of programming, at least some of that in an object-oriented setting, and that they have seen basic data structures. A course in data structures and algorithms systematizes what students already know about programming, endows that body of knowledge with rigor, and widens the variety of data structures and algorithmic techniques that students master. For the sake of rigor, we also assume students know some discrete mathematics, especially sets, formal logic, and proofs, including mathematical induction. I think of this course as the most important one in undergraduate computer science. It is a student's transition from *programming* to *computer science* itself. Everything that follows in the curriculum depends on it.

Many good books on algorithms and their supporting data structures have been written throughout computer science's history, pitched at readers at various levels. In this book's preparation I owe much to other authors' work at organizing this body of knowledge—and ultimately we all are indebted to the great repository of algorithmic wisdom, Don Knuth's *The Art of Computer Programming*.[1] It seems it's the lot of the instructor and textbook author, mother-robin-like, to chew the Knuthian worm and spit it gently in the students' mouths.

[1] Donald E. Knuth. *The Art of Computer Programming*. 2nd ed. Addison Wesley Longman Publishing Co., Inc., 1997.

With this, my contribution to that effort, I hope to push our students further in their internalizing this body of knowledge. This book does not attempt to be an encyclopedia of various data structures and algorithmic techniques, or even a curated tour of selected topics. Instead, I've focused on the themes—

the *loci* or *commonplaces*—that tie the topics together. Nearly everything we do in a course like this can be knit from a few basic principles about how memory is structured and the cost of algorithmic elements. This book directs students' attention to the *story* of how these elements can be adapted and recombined for better efficiency and more powerful problem-solving. They can then retain the principles and arc of the story even if the details fade in time.

PREPARING THIS BOOK required making a variety of decisions. The first was how to present the code. This is a dilemma because we need to show that the concepts are independent of programming languages. Some textbooks do this by using only pseudocode. But presenting algorithms in real, runnable code keeps the textbook honest—with pseudocode there is always the temptation to abstract away inconvenient details. Consequently there is almost no pseudocode in this text. Other textbooks choose one programming language and show language independence by avoiding features that are distinctive to that language. This text's philosophy, however, is that a programming language is a programmer's most important tool for expressing algorithmic ideas. To take full advantage of that expressive power, we don't discard those distinctive features that embody that language's spirit.

To work out that philosophy but avoid bias towards a single language, this text uses four languages for code examples and exercises: **Java**, **Python**, **C**, and **Standard ML** (SML). In some cases, to present a well-rounded perspective we present the same algorithm or data structure in more than one of these languages. At other times the text contains code in one language and leaves implementation in another language as an exercise. But most of the time we simply choose whichever of these four languages expresses the concept at hand most clearly: Java is the preferred medium for data abstraction, and consequently this book uses Java the most. Python's clean syntax makes it appropriate for succinct descriptions of algorithms. C is best when low-level details are relevant. SML allows us to show applicative, stateless programming in its pure form.

The decision to use four languages comes with the risk of scaring readers off. Do students need to know four programming languages in order to read this book? No, they don't. What they need is competence in Java—or a similar language like C# or Objective C from which they can transition to Java—and a *willingness* to learn enough Python as they go along so that they can read the Python code examples. That isn't a tall order, really, since Python reads like pseudocode to those who have experience in another language. The bulk of the code examples and exercises are in Java, as are all of the projects. Students with no background in C can read most of the C code as if it were Java, and they can skip code that uses distinctive C features without missing the flow of any section. The SML code similarly can be skipped, if necessary. Nevertheless, readers will have a richer experience if they have even a smattering of at least one of C or SML.

Another decision that warrants mention here is including a chapter on dynamic programming. The risk in this case is giving readers the impression that this book is designed for an advanced course in algorithms rather than sophomore data structures and algorithms. But I'm convinced dynamic programming is a crucial topic that students need to see early. The chapter on dynamic programming in this book has modest goals for the students. Specifically, after working through the chapter, they should understand what dynamic programming is, and they will have gained experience implementing dynamic programming algorithms when the solution is described for them mathematically. An experience like this makes it more likely they can master the technique in a more advanced course. Greedy algorithms, which are often presented as a companion topic to dynamic programming, aren't given their own chapter but are treated opportunistically throughout the book as problems with greedy solutions come up.

ADAPTING THIS BOOK for a specific course comes down to finding the right pace to cover Chapters 1 and 2, which together constitute a prologue for the rest of the text. We expect that most students have seen, at least in passing, most of the

topics in these chapters, but not systematically or with precision. These chapters review, fill in the gaps, and show how the topics all fit together. A class of students with a strong familiarity with data structures, ADTs, big-oh notation, and loop invariants may be able to breeze through these two chapters in a few weeks. But the chapters themselves contain a thorough introduction to all the foundations of data structures and algorithms for students who need that level of detail. For a class of students with a less strong background, the first two chapters have enough material to fill half a semester or more.

The remaining chapters are independent of each other, for the most part. They can be reordered, although the order in which they appear fits with a natural progression of more efficient implementations of the map abstract data type. Chapter 3, "Case Studies," contains topics that aren't long enough to get their own chapters but are each nice illustrations of the principles laid out in the first two chapters. Instructors can choose which sections in that chapter to cover, though some later topics use results from specific case studies presented in this chapter.

The project sequence in this book follows the principle that students need to implement the data structures and algorithms for themselves in order to master them. Accordingly, the items designated as *projects* follow the pattern of "Implement the data structure or algorithm described in this section." The book is also full of exercises of various shapes and sizes. Some require mathematical reasoning, others are short coding tasks, and others are substantial coding problems that apply the topics of the section. The complete code base, including in-text examples and starter code for the exercises and projects, can be downloaded from the website for this book: `tjvandrunen.github.io/algo-common/`

THIS BOOK EVOLVED from teaching notes I use in my own data structures and algorithms course, CSCI 345 at Wheaton College (IL). I credit two students, Lisa Hemphill and Stirling Joyner, for the origin of the book. I once mentioned during the Spring 2016 offering of CSCI 345 that I had a wish to write

my own textbook for the course. They came up to me after class and encouraged me to do it. Without that affirmation, this might have remained nothing but a nifty, vague idea.

I thank all my students who suffered through drafts of this book during the writing process. In particular, Caleb Veth, Andy Peterson, Andy Holmberg, Sharon Dunbar, Josiah Hsu, Claire Wagner, and Eliénaï Ouoba caught many errors and gave other valuable feedback. Much gratitude is due to Caleb Clark, who served as teaching assistant to that course for several semesters and spent summer 2017 revising the book's code base and supporting the writing process in other ways. I thank all those involved in the nominating and selecting process for the senior teaching award that paid both for Caleb Clark's summer work and for a course release for me in the Spring 2018 term; similarly I thank the Wheaton College sabbatical committee which awarded me a sabbatical for Fall 2019. Thanks also to John Jeffrey of Elmhurst University and David Stucki of Otterbein University who served as guest instructors at Wheaton College during my release time.

This book was typeset in LaTeX using the `tufte-book` document class. I am indebted to various participants in online LaTeX discussion communities for help on typesetting details. The page geometry is based on that used by Ian Oliver. The adornments that mark off exercises and projects come from the `adforn` package by the Arkandis Digital Foundry.

I thank Tom Sumner at Franklin, Beedle, who guided this project from proposal, through writing, reviewing, and revising, to publishing. I thank the reviewers of the chapter drafts:

John Jeffrey	Elmhurst University
Samuel McCauley	Williams College
Vijay Ramachandran	Colgate University
Darren Strash	Hamilton College
John Zelle	Wartburg College

Nearly all of their comments and calls for improvement were spot-on, though I regret that for the sake of keeping this book to a manageable length, some good suggestions for additions couldn't be taken. I also thank Tom MacWright for providing useful comments on a draft of Section 6.6.

I thank my wife Esther and children Annika, Isaac, Silas, and Timothy for constant moral support. In particular, little Tim is a living reminder of how long I have been working on this book, since I first pitched the idea of the book to my publisher a few weeks after Tim was born. Finally, I thank God for this project. Διὰ τῶν οἰκτιρμῶν τοῦ θεοῦ... τὴν λογικὴν λατρείαν ὑμῶν. "In light of the mercies of God... your rational service." Rom 12:1.