Algorithms

THIRD EDITION

IN C++

Parts 1–4

FUNDAMENTALS
DATA STRUCTURES
SORTING
SEARCHING

ROBERT SEDGEWICK
with C++ consulting by Christopher J. Van Wyk

Algorithms

THIRD EDITION

in C++

PARTS 1–4

FUNDAMENTALS
DATA STRUCTURES
SORTING
SEARCHING

Robert Sedgewick
Princeton University

ADDISON–WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Publishing Partner: **Peter S. Gordon**
Production Editor: **Amy Willcutt**

Library of Congress Catalog Card Number: 98-71799
ISBN 0-201-35088-2

Reproduced by Addison-Wesley from camera-ready copy supplied by the author.

Text printed on recycled and acid-free paper.
ISBN 0201350882
9 1011121314 CRS 05 04 03 02
9th Printing November 2002

Preface

THIS BOOK IS intended to survey the most important computer algorithms in use today, and to teach fundamental techniques to the growing number of people in need of knowing them. It can be used as a textbook for a second, third, or fourth course in computer science, after students have acquired basic programming skills and familiarity with computer systems, but before they have taken specialized courses in advanced areas of computer science or computer applications. The book also may be useful for self-study or as a reference for people engaged in the development of computer systems or applications programs, since it contains implementations of useful algorithms and detailed information on these algorithms' performance characteristics. The broad perspective taken makes the book an appropriate introduction to the field.

I have completely rewritten the text for this new edition, and I have added more than a thousand new exercises, more than a hundred new figures, and dozens of new programs. I have also added detailed commentary on all the figures and programs. This new material provides both coverage of new topics and fuller explanations of many of the classic algorithms. A new emphasis on abstract data types throughout the book makes the programs more broadly useful and relevant in modern programming environments. People who have read old editions of the book will find a wealth of new information throughout; all readers will find a wealth of pedagogical material that provides effective access to essential concepts.

Due to the large amount of new material, we have split the new edition into two volumes (each about the size of the old edition) of which this is the first. This volume covers fundamental concepts, data structures, sorting algorithms, and searching algorithms; the second volume covers advanced algorithms and applications, building on the basic abstractions and methods developed here. Nearly all the material on fundamentals and data structures in this edition is new.

This book is not just for programmers and computer-science students. Nearly everyone who uses a computer wants it to run faster or to solve larger problems. The algorithms in this book represent a body of knowledge developed over the last 50 years that has become indispensible in the efficient use of the computer, for a broad variety of applications. From N-body simulation problems in physics to genetic-sequencing problems in molecular biology, the basic methods described here have become essential in scientific research; and from database systems to Internet search engines, they have become essential parts of modern software systems. As the scope of computer applications becomes more widespread, so grows the impact of many of the basic methods covered here. The goal of this book is to serve as a resource for students and professionals interested in knowing and making intelligent use of these fundamental algorithms as basic tools for whatever computer application they might undertake.

Scope

The book contains 16 chapters grouped into four major parts: fundamentals, data structures, sorting, and searching. The descriptions here are intended to give readers an understanding of the basic properties of as broad a range of fundamental algorithms as possible. The algorithms described here have found widespread use for years, and represent an essential body of knowledge for both the practicing programmer and the computer-science student. Ingenious methods ranging from binomial queues to patricia tries are described, all related to basic paradigms at the heart of computer science. The second volume consists of four additional parts that cover strings, geometry, graphs, and advanced topics. My primary goal in developing these books has been to bring together the fundamental methods from these diverse areas, to provide access to the best methods known for solving problems by computer.

You will most appreciate the material in this book if you have had one or two previous courses in computer science or have had equivalent programming experience: one course in programming in a high-level language such as C++, Java, or C, and perhaps another course that teaches fundamental concepts of programming systems. This book is thus intended for anyone conversant with a modern programming

language and with the basic features of modern computer systems. References that might help to fill in gaps in your background are suggested in the text.

Most of the mathematical material supporting the analytic results is self-contained (or is labeled as beyond the scope of this book), so little specific preparation in mathematics is required for the bulk of the book, although mathematical maturity is definitely helpful.

Use in the Curriculum

There is a great deal of flexibility in how the material here can be taught, depending on the taste of the instructor and the preparation of the students. There is sufficient coverage of basic material for the book to be used to teach data structures to beginners, and there is sufficient detail and coverage of advanced material for the book to be used to teach the design and analysis of algorithms to upper-level students. Some instructors may wish to emphasize implementations and practical concerns; others may wish to emphasize analysis and theoretical concepts.

I am developing a variety of course materials for use with this book, including slide masters for use in lectures, programming assignments, homework assignments and sample exams, and interactive exercises for students. These materials will be accessible via the book's home page at `http://www.awl.com/cseng/titles/0-201-35088-2`.

An elementary course on data structures and algorithms might emphasize the basic data structures in Part 2 and their use in the implementations in Parts 3 and 4. A course on design and analysis of algorithms might emphasize the fundamental material in Part 1 and Chapter 5, then study the ways in which the algorithms in Parts 3 and 4 achieve good asymptotic performance. A course on software engineering might omit the mathematical and advanced algorithmic material, and emphasize how to integrate the implementations given here into large programs or systems. A course on algorithms might take a survey approach and introduce concepts from all these areas.

Earlier editions of this book have been used in recent years at scores of colleges and universities around the world as a text for the second or third course in computer science and as supplemental reading for other courses. At Princeton, our experience has been that the

breadth of coverage of material in this book provides our majors with an introduction to computer science that can be expanded upon in later courses on analysis of algorithms, systems programming and theoretical computer science, while providing the growing group of students from other disciplines with a large set of techniques that these people can immediately put to good use.

The exercises—most of which are new to this edition—fall into several types. Some are intended to test understanding of material in the text, and simply ask readers to work through an example or to apply concepts described in the text. Others involve implementing and putting together the algorithms, or running empirical studies to compare variants of the algorithms and to learn their properties. Still others are a repository for important information at a level of detail that is not appropriate for the text. Reading and thinking about the exercises will pay dividends for every reader.

Algorithms of Practical Use

Anyone wanting to use a computer more effectively can use this book for reference or for self-study. People with programming experience can find information on specific topics throughout the book. To a large extent, you can read the individual chapters in the book independently of the others, although, in some cases, algorithms in one chapter make use of methods from a previous chapter.

The orientation of the book is to study algorithms likely to be of practical use. The book provides information about the tools of the trade to the point that readers can confidently implement, debug, and put to work algorithms to solve a problem or to provide functionality in an application. Full implementations of the methods discussed are included, as are descriptions of the operations of these programs on a consistent set of examples.

Because we work with real code, rather than write pseudo-code, you can put the programs to practical use quickly. Program listings are available from the book's home page. You can use these working programs in many ways to help you study algorithms. Read them to check your understanding of the details of an algorithm, or to see one way to handle initializations, boundary conditions, and other awkward situations that often pose programming challenges. Run

them to see the algorithms in action, to study performance empirically and check your results against the tables in the book, or to try your own modifications.

When appropriate, empirical and analytic results are presented to illustrate why certain algorithms are preferred. When interesting, the relationship of the practical algorithms being discussed to purely theoretical results is described. Although not emphasized, connections to the analysis of algorithms and theoretical computer science are developed in context. Specific information on performance characteristics of algorithms and implementations is synthesized, encapsulated, and discussed throughout the book.

Programming Language

The programming language used for all of the implementations is C++. The programs use a wide range of standard C++ idioms, and the text includes concise descriptions of each construct.

Chris Van Wyk and I developed a style of C++ programming based on classes, templates, and overloaded operators that we feel is an effective way to present the algorithms and data structures as real programs. We have striven for elegant, compact, efficient, and portable implementations. The style is consistent whenever possible, so that programs that are similar look similar.

For many of the algorithms in this book, the similarities hold regardless of the language: Quicksort is quicksort (to pick one prominent example), whether expressed in Ada, Algol-60, Basic, C, C++, Fortran, Java, Mesa, Modula-3, Pascal, PostScript, Smalltalk, or countless other programming languages and environments where it has proved to be an effective sorting method. On the one hand, our code is informed by experience with implementing algorithms in these and numerous other languages (a C version of this book is also available, and a Java version will appear soon); on the other hand, some of the properties of some of these languages are informed by their designers' experience with some of the algorithms and data structures that we consider in this book.

Chapter 1 constitutes a detailed example of this approach to developing efficient C++ implementations of our algorithms, and Chapter 2 describes our approach to analyzing them. Chapters 3 and 4 are de-

voted to describing and justifying the basic mechanisms that we use for data type and ADT implementations. These four chapters set the stage for the rest of the book.

Acknowledgments

Many people gave me helpful feedback on earlier versions of this book. In particular, hundreds of students at Princeton and Brown have suffered through preliminary drafts over the years. Special thanks are due to Trina Avery and Tom Freeman for their help in producing the first edition; to Janet Incerpi for her creativity and ingenuity in persuading our early and primitive digital computerized typesetting hardware and software to produce the first edition; to Marc Brown for his part in the algorithm visualization research that was the genesis of so many of the figures in the book; and to Dave Hanson and Andrew Appel for their willingness to answer all of my questions about programming languages. I would also like to thank the many readers who have provided me with comments about various editions, including Guy Almes, Jon Bentley, Marc Brown, Jay Gischer, Allan Heydon, Kennedy Lemke, Udi Manber, Dana Richards, John Reif, M. Rosenfeld, Stephen Seidman, Michael Quinn, and William Ward.

To produce this new edition, I have had the pleasure of working with Peter Gordon, Debbie Lafferty, and Helen Goldstein at Addison-Wesley, who have patiently shepherded this project as it has evolved. It has also been my pleasure to work with several other members of the professional staff at Addison-Wesley. The nature of this project made the book a somewhat unusual challenge for many of them, and I much appreciate their forbearance.

I have gained three new mentors in writing this book, and particularly want to express my appreciation to them. First, Steve Summit carefully checked early versions of the manuscript on a technical level, and provided me with literally thousands of detailed comments, particularly on the programs. Steve clearly understood my goal of providing elegant, efficient, and effective implementations, and his comments not only helped me to provide a measure of consistency across the implementations, but also helped me to improve many of them substantially. Second, Lyn Dupre also provided me with thousands of detailed comments on the manuscript, which were invaluable in helping me not only

to correct and avoid grammatical errors, but also—more important—to find a consistent and coherent writing style that helps bind together the daunting mass of technical material here. Third, Chris Van Wyk implemented and debugged all my algorithms in C++, answered numerous questions about C++, helped to develop an appropriate C++ programming style, and carefully read the manuscript twice. Chris also patiently stood by as I took apart many of his C++ programs and then, as I learned more and more about C++ from him, had to put them back together much as he had written them. I am extremely grateful for the opportunity to learn from Steve, Lyn, and Chris—their input was vital in the development of this book.

Much of what I have written here I have learned from the teaching and writings of Don Knuth, my advisor at Stanford. Although Don had no direct influence on this work, his presence may be felt in the book, for it was he who put the study of algorithms on the scientific footing that makes a work such as this possible. My friend and colleague Philippe Flajolet, who has been a major force in the development of the analysis of algorithms as a mature research area, has had a similar influence on this work.

I am deeply thankful for the support of Princeton University, Brown University, and the Institut National de Recherche en Informatique et Automatique (INRIA), where I did most of the work on the book; and of the Institute for Defense Analyses and the Xerox Palo Alto Research Center, where I did some work on the book while visiting. Many parts of the book are dependent on research that has been generously supported by the National Science Foundation and the Office of Naval Research. Finally, I thank Bill Bowen, Aaron Lemonick, and Neil Rudenstine for their support in building an academic environment at Princeton in which I was able to prepare this book, despite my numerous other responsibilities.

Robert Sedgewick
Marly-le-Roi, France, 1983
Princeton, New Jersey, 1990, 1992
Jamestown, Rhode Island, 1997
Princeton, New Jersey, 1998

C++ Consultant's Preface

Algorithms are what first drew me to computer science. Studying algorithms requires thinking in several ways: creatively, to discover an idea that will solve a problem; logically, to analyze its correctness; mathematically, to analyze its performance; and, painstakingly, to express the idea as a detailed sequence of steps so it can become software. Since my greatest satisfaction in studying algorithms comes from realizing them as working computer programs, I jumped at the chance to work with Bob Sedgewick on an algorithms book based on C++ programs.

Bob and I have written the sample programs using appropriate features of C++ in straightforward ways. We use classes to separate the specification of an abstract data type from the details of its implementation. We use templates and overloaded operators so that our programs can be used without change for many different types of data.

We have, however, passed up the chance to display many other C++ techniques. For example, we usually omit at least some of the constructors needed to make every class be "first class" (but see Chapter 4 to learn how you can do this); most constructors use assignment instead of initialization to store values in data members; we use C-style character strings instead of relying on strings in the C++ library; we do not use the most "lightweight" possible wrapper classes; and we use simple, instead of "smart," pointers.

We made most of these choices to keep the focus on algorithms, rather than distracting you with details of techniques specific to C++. Once you understand how the programs in this book work, you will be well equipped to learn about any or all of these C++ techniques, to appreciate the efficiency tradeoffs they involve, and to admire the ingenuity of the designers of the Standard Template Library.

Whether this is your first encounter with the study of algorithms or you are renewing an old acquaintance, may you enjoy it at least as much as I have enjoyed working with Bob Sedgewick on the programs.

Thanks: to Jon Bentley, Brian Kernighan, and Tom Szymanski, from whom I learned much of what I know about programming; to Debbie Lafferty, who suggested I consider this project; and to Bell Labs, Drew University, and Princeton University, for institutional support.

Christopher Van Wyk
Chatham, New Jersey, 1998

To Adam, Andrew, Brett, Robbie,
and especially Linda

Notes on Exercises

Classifying exercises is an activity fraught with peril, because readers of a book such as this come to the material with various levels of knowledge and experience. Nonetheless, guidance is appropriate, so many of the exercises carry one of four annotations, to help you decide how to approach them.

Exercises that *test your understanding* of the material are marked with an open triangle, as follows:

> ▷ **9.57** Give the binomial queue that results when the keys E A S Y Q U E S T I O N are inserted into an initially empty binomial queue.

Most often, such exercises relate directly to examples in the text. They should present no special difficulty, but working them might teach you a fact or concept that may have eluded you when you read the text.

Exercises that *add new and thought-provoking* information to the material are marked with an open circle, as follows:

> ○ **14.20** Write a program that inserts N random integers into a table of size $N/100$ using separate chaining, then finds the length of the shortest and longest lists, for $N = 10^3$, 10^4, 10^5, and 10^6.

Such exercises encourage you to think about an important concept that is related to the material in the text, or to answer a question that may have occurred to you when you read the text. You may find it worthwhile to read these exercises, even if you do not have the time to work them through.

Exercises that are intended to *challenge you* are marked with a black dot, as follows:

> • **8.46** Suppose that mergesort is implemented to split the file at a *random* position, rather than exactly in the middle. How many comparisons are used by such a method to sort N elements, on the average?

Such exercises may require a substantial amount of time to complete, depending upon your experience. Generally, the most productive approach is to work on them in a few different sittings.

A few exercises that are *extremely difficult* (by comparison with most others) are marked with two black dots, as follows:

> •• **15.29** Prove that the height of a trie built from N random bitstrings is about $2 \lg N$.

These exercises are similar to questions that might be addressed in the research literature, but the material in the book may prepare you to enjoy trying to solve them (and perhaps succeeding).

The annotations are intended to be neutral with respect to your programming and mathematical ability. Those exercises that require expertise in programming or in mathematical analysis are self-evident. All readers are encouraged to test their understanding of the algorithms by implementing them. Still, an exercise such as this one is straightforward for a practicing programmer or a student in a programming course, but may require substantial work for someone who has not recently programmed:

> **1.23** Modify Program 1.4 to generate random pairs of integers between 0 and $N-1$ instead of reading them from standard input, and to loop until $N-1$ *union* operations have been performed. Run your program for $N = 10^3$, 10^4, 10^5, and $10^6$and print out the total number of edges generated for each value of N.

In a similar vein, all readers are encouraged to strive to appreciate the analytic underpinnings of our knowledge about properties of algorithms. Still, an exercise such as this one is straightforward for a scientist or a student in a discrete mathematics course, but may require substantial work for someone who has not recently done mathematical analysis:

> **1.13** Compute the *average* distance from a node to the root in a worst-case tree of 2^n nodes built by the weighted quick-union algorithm.

There are far too many exercises for you to read and assimilate them all; my hope is that there are enough exercises here to stimulate you to strive to come to a broader understanding on the topics that interest you than you can glean by simply reading the text.

Contents

Fundamentals

Data Structures

Sorting