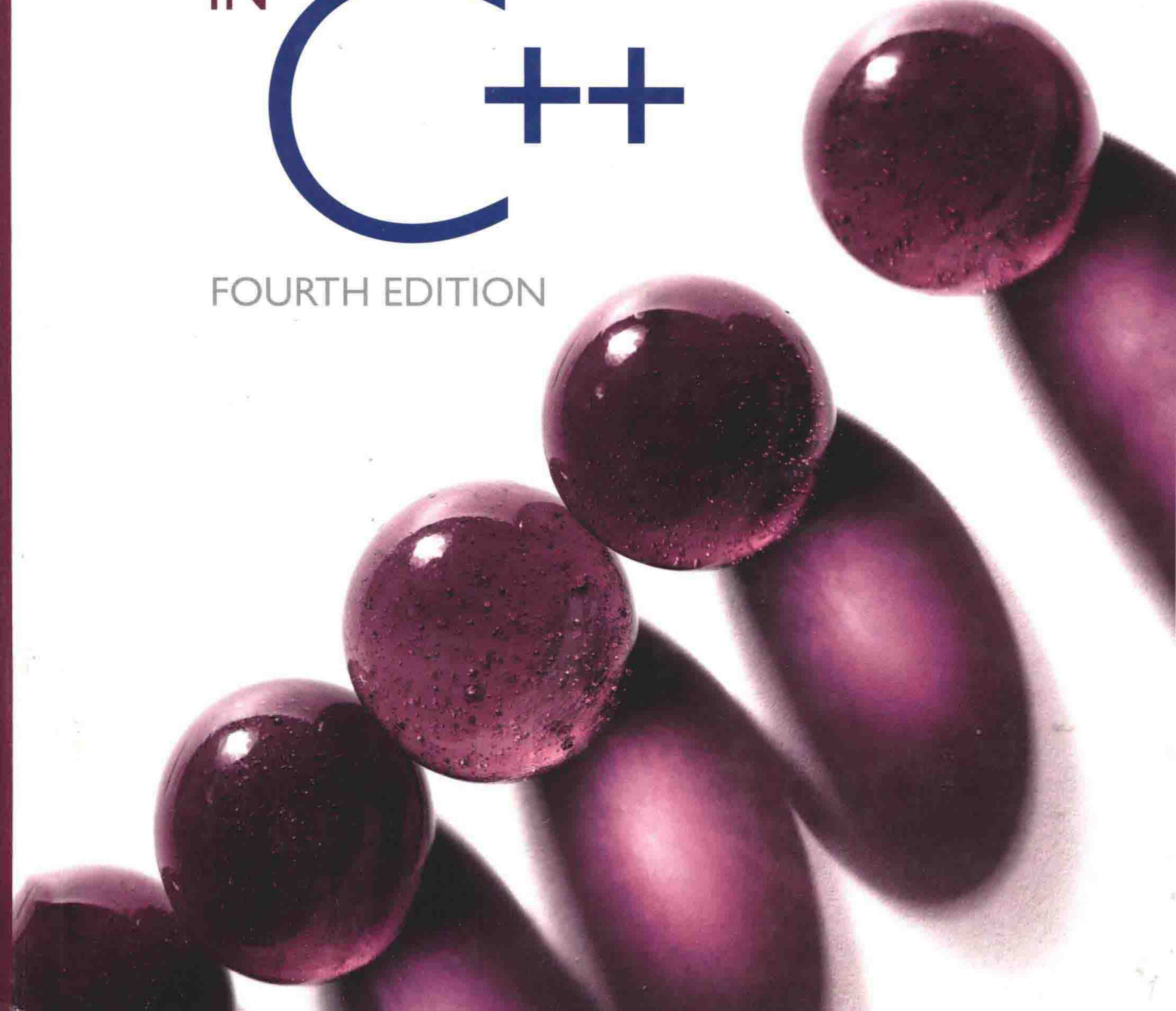
MARK ALLEN WEISS

DATA STRUCTURES AND ALGORITHM ANALYSIS IN C++

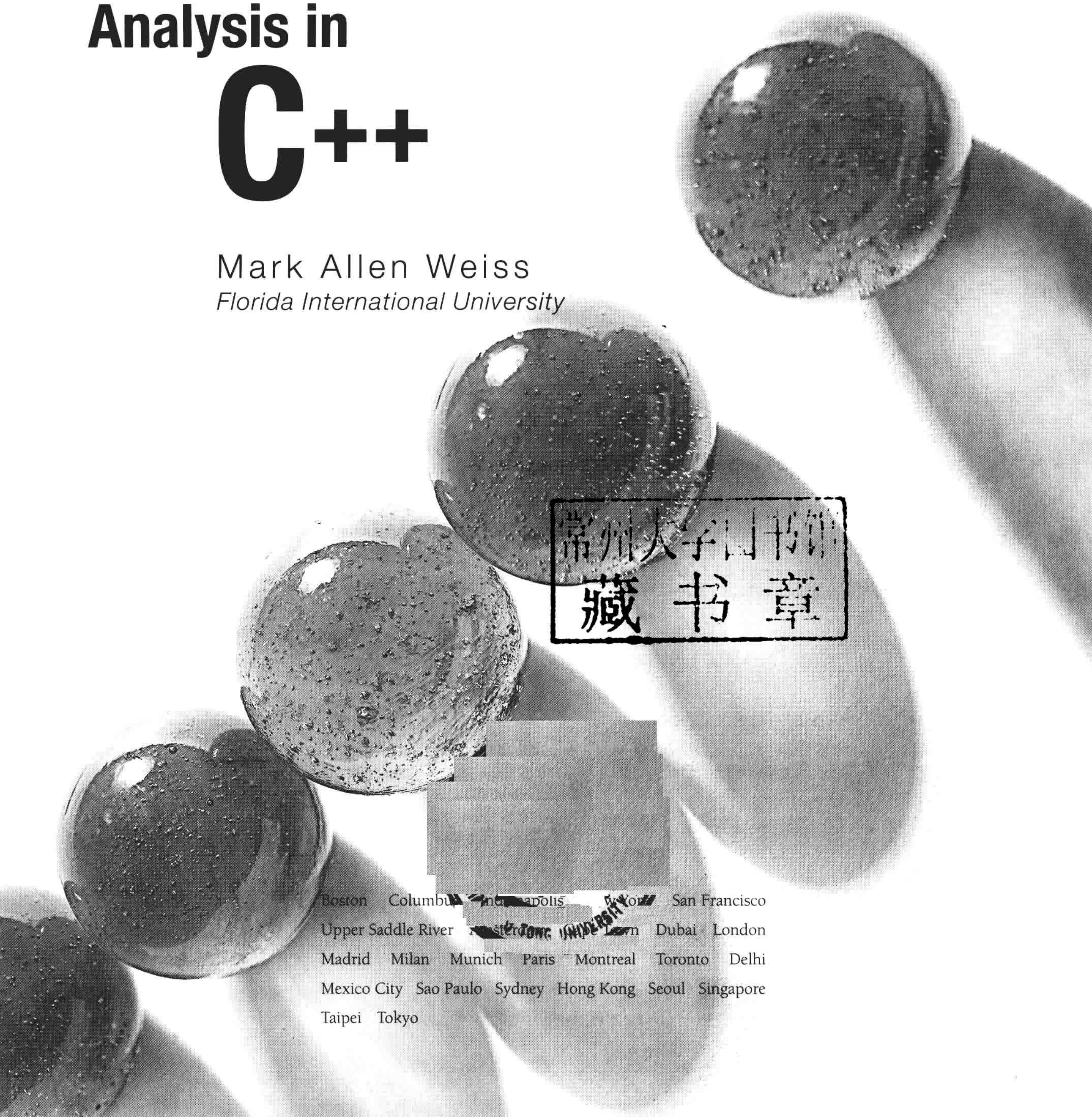
FOURTH EDITION



Fourth Edition

Data Structures and Algorithm Analysis in C++

Mark Allen Weiss
Florida International University



Boston Columbus Indianapolis New York San Francisco
Upper Saddle River Amsterdam Cape Town Dubai London
Madrid Milan Munich Paris Montreal Toronto Delhi
Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore
Taipei Tokyo

Editorial Director, ECS: Marcia Horton
Executive Editor: Tracy Johnson
Editorial Assistant: Jenah Blitz-Stoehr
Director of Marketing: Christy Lesko
Marketing Manager: Yez Alayan
Senior Marketing Coordinator: Kathryn Ferranti
Marketing Assistant: Jon Bryant
Director of Production: Erin Gregg
Senior Managing Editor: Scott Disanno
Senior Production Project Manager: Marilyn Lloyd
Manufacturing Buyer: Linda Sager
Art Director: Jayne Conte
Cover Designer: Bruce Kenselaar
Permissions Supervisor: Michael Joyce
Permissions Administrator: Jenell Forschler
Cover Image: © De-kay | Dreamstime.com
Media Project Manager: Renata Butera
Full-Service Project Management: Integra Software Services Pvt. Ltd.
Composition: Integra Software Services Pvt. Ltd.
Text and Cover Printer/Binder: Courier Westford

Library of Congress Cataloging-in-Publication Data

Weiss, Mark Allen.
Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition.
pages cm
ISBN-13: 978-0-13-284737-7 (alk. paper)
ISBN-10: 0-13-284737-X (alk. paper)
1. C++ (Computer program language) 2. Data structures (Computer science) 3. Computer algorithms. I. Title.
QA76.73.C153W46 2014
005.7′3—dc23
2013011064

10 9 8 7 6 5 4 3 2 1

PEARSON

www.pearsonhighered.com

ISBN-10: 0-13-284737-X
ISBN-13: 978-0-13-284737-7

Fourth Edition

Data Structures and Algorithm Analysis in C++

To my kind, brilliant, and inspiring Sara.

PREFACE

Purpose/Goals

The fourth edition of *Data Structures and Algorithm Analysis in C++* describes *data structures,* methods of organizing large amounts of data, and *algorithm analysis,* the estimation of the running time of algorithms. As computers become faster and faster, the need for programs that can handle large amounts of input becomes more acute. Paradoxically, this requires more careful attention to efficiency, since inefficiencies in programs become most obvious when input sizes are large. By analyzing an algorithm before it is actually coded, students can decide if a particular solution will be feasible. For example, in this text students look at specific problems and see how careful implementations can reduce the time constraint for large amounts of data from centuries to less than a second. Therefore, no algorithm or data structure is presented without an explanation of its running time. In some cases, minute details that affect the running time of the implementation are explored.

Once a solution method is determined, a program must still be written. As computers have become more powerful, the problems they must solve have become larger and more complex, requiring development of more intricate programs. The goal of this text is to teach students good programming and algorithm analysis skills simultaneously so that they can develop such programs with the maximum amount of efficiency.

This book is suitable for either an advanced data structures course or a first-year graduate course in algorithm analysis. Students should have some knowledge of intermediate programming, including such topics as pointers, recursion, and object-based programming, as well as some background in discrete math.

Approach

Although the material in this text is largely language-independent, programming requires the use of a specific language. As the title implies, we have chosen C++ for this book.

C++ has become a leading systems programming language. In addition to fixing many of the syntactic flaws of C, C++ provides direct constructs (the *class* and *template*) to implement generic data structures as abstract data types.

The most difficult part of writing this book was deciding on the amount of C++ to include. Use too many features of C++ and one gets an incomprehensible text; use too few and you have little more than a C text that supports classes.

The approach we take is to present the material in an *object-based approach.* As such, there is almost no use of inheritance in the text. We use class templates to describe generic data structures. We generally avoid esoteric C++ features and use the `vector` and `string` classes that are now part of the C++ standard. Previous editions have implemented class templates by separating the class template interface from its implementation. Although this is arguably the preferred approach, it exposes compiler problems that have made it difficult for readers to actually use the code. As a result, in this edition the online code represents class templates as a single unit, with no separation of interface and implementation. Chapter 1 provides a review of the C++ features that are used throughout the text and describes our approach to class templates. Appendix A describes how the class templates could be rewritten to use separate compilation.

Complete versions of the data structures, in both C++ and Java, are available on the Internet. We use similar coding conventions to make the parallels between the two languages more evident.

Summary of the Most Significant Changes in the Fourth Edition

The fourth edition incorporates numerous bug fixes, and many parts of the book have undergone revision to increase the clarity of presentation. In addition,

- Chapter 4 includes implementation of the AVL tree deletion algorithm—a topic often requested by readers.
- Chapter 5 has been extensively revised and enlarged and now contains material on two newer algorithms: cuckoo hashing and hopscotch hashing. Additionally, a new section on universal hashing has been added. Also new is a brief discussion of the `unordered_set` and `unordered_map` class templates introduced in C++11.
- Chapter 6 is mostly unchanged; however, the implementation of the binary heap makes use of move operations that were introduced in C++11.
- Chapter 7 now contains material on radix sort, and a new section on lower-bound proofs has been added. Sorting code makes use of move operations that were introduced in C++11.
- Chapter 8 uses the new union/find analysis by Seidel and Sharir and shows the $O(\, M\,\alpha(M,N)\,)$ bound instead of the weaker $O(\, M\log^* N\,)$ bound in prior editions.
- Chapter 12 adds material on suffix trees and suffix arrays, including the linear-time suffix array construction algorithm by Karkkainen and Sanders (with implementation). The sections covering deterministic skip lists and AA-trees have been removed.
- Throughout the text, the code has been updated to use C++11. Notably, this means use of the new C++11 features, including the `auto` keyword, the range `for` loop, move construction and assignment, and uniform initialization.

Overview

Chapter 1 contains review material on discrete math and recursion. I believe the only way to be comfortable with recursion is to see good uses over and over. Therefore, recursion is prevalent in this text, with examples in every chapter except Chapter 5. Chapter 1 also includes material that serves as a review of basic C++. Included is a discussion of templates and important constructs in C++ class design.

Chapter 2 deals with algorithm analysis. This chapter explains asymptotic analysis and its major weaknesses. Many examples are provided, including an in-depth explanation of logarithmic running time. Simple recursive programs are analyzed by intuitively converting them into iterative programs. More complicated divide-and-conquer programs are introduced, but some of the analysis (solving recurrence relations) is implicitly delayed until Chapter 7, where it is performed in detail.

Chapter 3 covers lists, stacks, and queues. This chapter includes a discussion of the STL `vector` and `list` classes, including material on iterators, and it provides implementations of a significant subset of the STL `vector` and `list` classes.

Chapter 4 covers trees, with an emphasis on search trees, including external search trees (B-trees). The UNIX file system and expression trees are used as examples. AVL trees and splay trees are introduced. More careful treatment of search tree implementation details is found in Chapter 12. Additional coverage of trees, such as file compression and game trees, is deferred until Chapter 10. Data structures for an external medium are considered as the final topic in several chapters. Included is a discussion of the STL `set` and `map` classes, including a significant example that illustrates the use of three separate maps to efficiently solve a problem.

Chapter 5 discusses hash tables, including the classic algorithms such as separate chaining and linear and quadratic probing, as well as several newer algorithms, namely cuckoo hashing and hopscotch hashing. Universal hashing is also discussed, and extendible hashing is covered at the end of the chapter.

Chapter 6 is about priority queues. Binary heaps are covered, and there is additional material on some of the theoretically interesting implementations of priority queues. The Fibonacci heap is discussed in Chapter 11, and the pairing heap is discussed in Chapter 12.

Chapter 7 covers sorting. It is very specific with respect to coding details and analysis. All the important general-purpose sorting algorithms are covered and compared. Four algorithms are analyzed in detail: insertion sort, Shellsort, heapsort, and quicksort. New to this edition is radix sort and lower bound proofs for selection-related problems. External sorting is covered at the end of the chapter.

Chapter 8 discusses the disjoint set algorithm with proof of the running time. This is a short and specific chapter that can be skipped if Kruskal's algorithm is not discussed.

Chapter 9 covers graph algorithms. Algorithms on graphs are interesting, not only because they frequently occur in practice but also because their running time is so heavily dependent on the proper use of data structures. Virtually all of the standard algorithms are presented along with appropriate data structures, pseudocode, and analysis of running time. To place these problems in a proper context, a short discussion on complexity theory (including *NP*-completeness and undecidability) is provided.

Chapter 10 covers algorithm design by examining common problem-solving techniques. This chapter is heavily fortified with examples. Pseudocode is used in these later chapters so that the student's appreciation of an example algorithm is not obscured by implementation details.

Chapter 11 deals with amortized analysis. Three data structures from Chapters 4 and 6 and the Fibonacci heap, introduced in this chapter, are analyzed.

Chapter 12 covers search tree algorithms, the suffix tree and array, the *k*-d tree, and the pairing heap. This chapter departs from the rest of the text by providing complete and careful implementations for the search trees and pairing heap. The material is structured so that the instructor can integrate sections into discussions from other chapters. For example, the top-down red-black tree in Chapter 12 can be discussed along with AVL trees (in Chapter 4).

Chapters 1 to 9 provide enough material for most one-semester data structures courses. If time permits, then Chapter 10 can be covered. A graduate course on algorithm analysis could cover chapters 7 to 11. The advanced data structures analyzed in Chapter 11 can easily be referred to in the earlier chapters. The discussion of *NP*-completeness in Chapter 9 is far too brief to be used in such a course. You might find it useful to use an additional work on *NP*-completeness to augment this text.

Exercises

Exercises, provided at the end of each chapter, match the order in which material is presented. The last exercises may address the chapter as a whole rather than a specific section. Difficult exercises are marked with an asterisk, and more challenging exercises have two asterisks.

References

References are placed at the end of each chapter. Generally the references either are historical, representing the original source of the material, or they represent extensions and improvements to the results given in the text. Some references represent solutions to exercises.

Supplements

The following supplements are available to all readers at http://cssupport.pearsoncmg.com/

- Source code for example programs
- Errata

In addition, the following material is available only to qualified instructors at Pearson Instructor Resource Center (www.pearsonhighered.com/irc). Visit the IRC or contact your Pearson Education sales representative for access.

- Solutions to selected exercises
- Figures from the book
- Errata

Acknowledgments

Many, many people have helped me in the preparation of books in this series. Some are listed in other versions of the book; thanks to all.

As usual, the writing process was made easier by the professionals at Pearson. I'd like to thank my editor, Tracy Johnson, and production editor, Marilyn Lloyd. My wonderful wife Jill deserves extra special thanks for everything she does.

Finally, I'd like to thank the numerous readers who have sent e-mail messages and pointed out errors or inconsistencies in earlier versions. My website www.cis.fiu.edu/~weiss will also contain updated source code (in C++ and Java), an errata list, and a link to submit bug reports.

M.A.W.
Miami, Florida

CONTENTS