

DATA STRUCTURES
USING

JAVATM

DUNCAN A. BUELL

University of South Carolina



JONES & BARTLETT
LEARNING

World Headquarters
Jones & Bartlett Learning
5 Wall Street
Burlington, MA 01803
978-443-5000
info@jblearning.com
www.jblearning.com

Jones & Bartlett Learning books and products are available through most bookstores and online booksellers. To contact Jones & Bartlett Learning directly, call 800-832-0034, fax 978-443-8000, or visit our website, www.jblearning.com.

Substantial discounts on bulk quantities of Jones & Bartlett Learning publications are available to corporations, professional associations, and other qualified organizations. For details and specific discount information, contact the special sales department at Jones & Bartlett Learning via the above contact information or send an email to specialsales@jblearning.com.

Copyright © 2013 by Jones & Bartlett Learning, LLC, an Ascend Learning Company

All rights reserved. No part of the material protected by this copyright may be reproduced or utilized in any form, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the copyright owner.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Production Credits

Publisher: Cathleen Sether
Senior Acquisitions Editor: Timothy Anderson
Managing Editor: Amy Bloom
Director of Production: Amy Rose
Marketing Manager: Lindsay White
V.P., Manufacturing and Inventory Control: Therese Connell
Composition: Laserwords
Cover Design: Scott Moden
Cover Image: © Mike Norton/Shutterstock, Inc.
Printing and Binding: Malloy, Inc.
Cover Printing: Malloy, Inc.

Library of Congress Cataloging-in-Publication Data

Buell, Duncan A.

Data structures using Java / Duncan A. Buell. – 1st ed.

p. cm.

Includes index.

ISBN-13: 978-1-4496-2807-9 (pbk.)

ISBN-10: 1-4496-2807-9 (pbk.)

1. Data structures (Computer science) 2. Java (Computer program language) I. Title.

QA76.9.D35B84 2013

005.2'762–dc23

2011033497

6048

Printed in the United States of America

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

For Mary Ann

Contents

Preface xiii

Chapter 1 Introduction 1

- 1.1 The Course of This Course 1
 - 1.1.1 Layers of Software 2
 - 1.1.2 Algorithms and Efficiency 2
 - 1.1.3 Sorting 3
 - 1.1.4 Maintaining Sorted Order 4
 - 1.1.5 Indexing, Search, and Retrieval 4
 - 1.1.6 Dynamic Versus Static Behavior 5
 - 1.1.7 Memory and Bandwidth 6
 - 1.1.8 Heuristics 7
 - 1.1.9 Templates, Generics, and Abstract Classes 8
- 1.2 Examples 9
 - 1.2.1 Card Catalogs 9
 - 1.2.2 Student Records 10
 - 1.2.3 Retail Transactions 10
 - 1.2.4 Packet Traffic 10
 - 1.2.5 Process Queues 11
 - 1.2.6 Google 11
 - 1.2.7 Game Trees 12
 - 1.2.8 Phylogenetics 12
- 1.3 Summary 13

Chapter 2 A Review of Java 15

- 2.1 The Basic Data Payload Class 15
- 2.2 The Main Program 20
- 2.3 Comments about Style and Documentation 23
 - 2.3.1 Javadoc 26
- 2.4 UML 28
- 2.5 Summary 29
- 2.6 Exercises 29

Chapter 3	Flat Files	31
3.1	A Basic Indexed File	32
3.1.1	A Basic Program	32
3.1.2	The Record Code	35
3.1.3	A Driver for the Application	37
3.1.4	The Original Flatfile Code	38
3.2	An Analysis of the Code	44
3.2.1	Sorted Data?	45
3.2.2	A Simple but Inefficient Sorting Algorithm	47
3.2.3	Searching Through a List	49
3.3	A Foreshadowing of Things to Come	49
3.3.1	Effective Sorting Techniques	51
3.3.2	Effective Search Techniques and Data Structures for Maintaining Priority	52
3.3.3	Static Versus Dynamic Storage Structures	53
3.3.4	Generic and Abstract Classes	53
3.3.5	Arrays Versus ArrayLists	54
3.4	Binary Search: Our First Mature Algorithm	54
3.5	Summary	58
3.6	Exercises	59
Chapter 4	Arrays and Linked Lists	61
4.1	Arrays and ArrayLists	62
4.2	Linked Lists	66
4.2.1	Linked Lists and Pointers	66
4.2.2	A Linked List Class	68
4.2.3	Exception Handling	73
4.2.4	Linked Lists Using Arrays	76
4.2.5	Arrays, ArrayLists, and the Free List	77
4.3	Insertionsort	78
4.4	Summary	81
4.5	Exercises	81
Chapter 5	Generics, Collections, and Testing	85
5.1	Using Generics for the Data Payload	85
5.2	Objects and Equality	93
5.3	The Java LinkedList	94
5.4	Iterators	96
5.4.1	The Justification for Iterators	96
5.5	Implementing Our Very Own Collection	99

5.6	Testing with JUnit	105
5.7	Reflection	110
5.8	Summary	112
5.9	Exercises	112
Chapter 6	Estimating Asymptotic Efficiency	115
6.1	Analysis of Algorithms	116
6.1.1	The Definition of a “Logarithm”	117
6.1.2	A Few Comments about the Real World	117
6.2	Some Rigor	118
6.2.1	An Apology for Some Audacity	120
6.2.2	Basic Orders of Magnitude, and Some Intuition	120
6.2.3	Constants Don’t Matter	122
6.2.4	Constant Startup Costs Don’t Matter	124
6.2.5	Big Oh Is Transitive	125
6.2.6	All Logarithms Are the Same	126
6.2.7	All Polynomials of the Same Degree Are the Same	126
6.2.8	All Reasonable Cost-of-Work Functions Go Off to Infinity	129
6.3	Some More Rigor	129
6.3.1	Little oh	129
6.3.2	Big Theta, and Some More Simplifying Theorems	132
6.3.3	Ignoring Fencepost Issues	134
6.3.4	Basic Orders of Magnitude Revisited	135
6.4	More Intuition—Common Asymptotics	137
*6.5	Exponential Orders of Magnitude	141
6.5.1	Exponential Orders of Magnitude	141
*6.6	Big Omega	143
6.6.1	Traditional Big Omega (Hardy, about 1915)	143
6.6.2	Knuth’s Big Omega (Knuth, about 1975)	144
6.7	What Do We Count?	145
6.8	Binary Divide and Conquer Is Optimal	145
6.9	Summary	148
6.10	Exercises	149

Chapter 7	Stacks and Queues	151
7.1	Stacks	152
7.1.1	Exception Handling	154
7.1.2	Stacks Using Nodes	156
7.1.3	Stacks Using Arrays	157
7.1.4	Reverse Polish Arithmetic	158
7.2	HTML/XML and Stacks	160
7.3	Queues	165
7.3.1	Queues Using Nodes	165
7.3.2	Queues Using Arrays	167
7.4	The JCF Classes	170
7.4.1	The JCF Stack	170
7.4.2	The JCF Queue	171
7.4.3	Stacks, Queues, and Deques	173
7.5	Priority Queues	174
7.5.1	The Heap	174
7.5.2	Creating a Heap	178
7.5.3	Maintaining the Heap	179
7.6	Summary	182
7.7	Exercises	183
Chapter 8	Recursion	185
8.1	Factorials and Fibonacci Numbers	186
8.2	The Collatz Problem	187
8.3	Greatest Common Divisor	188
8.4	The Ackermann Function	189
8.5	Binary Divide-and-Conquer Algorithms	190
8.6	Permutations and Combinations	191
8.7	Gaming and Searching Applications	194
8.8	Implementation Issues	196
8.9	Summary	199
8.10	Exercises	199
Chapter 9	A First Look at Graphs	203
9.1	Formal Notions of a Graph	204
9.2	Examples of Graphs	206
9.3	Transitive Closure	210
9.3.1	Facebook Graphs	210
9.3.2	Theory	210
9.3.3	Applications in Computing	212
9.4	Local–Global Questions	214

9.5	Greedy Algorithms	215
9.6	Summary	217
9.7	Exercises	217
Chapter 10	Trees	221
10.1	Trees	222
10.2	Spanning Trees	225
10.3	Data Structures for Trees	226
10.4	Binary Trees	228
10.4.1	Implementing Binary Trees with Nodes	229
10.4.2	Implementing Binary Trees with Arrays	231
10.5	The Heap	233
10.6	Traversing a Tree	234
10.6.1	Breadth-First and Depth-First Traversal	241
10.6.2	Traversal on Parallel Computers	242
10.7	Summary	244
10.8	Exercises	244
Chapter 11	Sorting	247
11.1	Worst-Case, Best-Case, and Average-Case	248
11.2	Bubblesort	249
11.3	Insertionsort	250
11.4	Improving Bubblesort	251
11.5	Heapsort	252
11.6	Worst-Case, Best-Case, and Average-Case (Again)	253
11.7	Mergesort	255
11.7.1	Analysis of Mergesort	259
11.7.2	Disk-Based Sorting	259
11.8	Quicksort	260
11.8.1	The Quicksort Algorithm	261
11.8.2	Average-Case Running Time for Quicksort	265
11.9	Sorting Without Comparisons	266
11.10	Experimental Results	267
11.11	Auxiliary Space and Implementation Issues	269
11.11.1	Space Considerations	270
11.11.2	Memory	270
11.11.3	Multilevel Sorts	271
11.11.4	Stability	271

11.12	The Asymptotics of Sorting	272
11.12.1	Lower Bounds on Sorting	272
11.12.2	A Proof of Lower Bounds	274
11.12.3	Lower Bounds for the Average Case	275
*11.13	Sorting in Parallel	276
11.13.1	Analysis	280
11.14	Summary	282
11.15	Exercises	283
Chapter 12	Searching	287
12.1	Associative Matching	288
12.2	Indexing for Search	289
12.3	Binary Search Trees	289
12.3.1	Inorder Traversal and Binary Search Trees	290
12.3.2	Building a Binary Search Tree	292
12.4	Sophisticated Search Trees	293
12.5	Hashing	295
12.6	Random Numbers and Hash Functions: A Brief Digression	297
12.6.1	Random Numbers	297
12.6.2	Cryptographic Hash Functions	299
12.6.3	Random Numbers in Java	299
12.7	The Java Collections Framework	300
12.8	The Java <code>HashSet</code>	303
12.8.1	A Spell-Checking Program	304
12.9	The Java <code>TreeMap</code>	305
12.9.1	A Digression on Documentation	306
12.10	The Java <code>TreeSet</code> , <code>HashMap</code> , and <code>LinkedList</code>	307
12.11	Summary	308
12.12	Exercises	308
Chapter 13	Graphs	311
13.1	Graphs and Applications	312
13.2	Adjacency Matrices and Linked Lists	313
13.2.1	Adjacency Matrices	314
13.2.2	Sparse Graphs and Sparse Matrices	316
13.2.3	Linked Lists for Sparse Graphs	316
13.2.4	A Middle Ground, Somewhat Helped by Java	320
13.3	Breadth-First and Depth-First Search	320

13.4	Graph Problems	328
13.4.1	<i>NP</i> -Completeness	329
13.4.2	Eulerian Cycles	331
13.4.3	Connected Components	332
13.4.4	Spanning Trees	332
13.4.5	Shortest Path	333
13.4.6	Hamiltonian Cycles	333
13.4.7	Traveling Salesman Problem	333
13.4.8	Cliques	334
13.5	Summary	334
13.6	Exercises	335
Appendix A	The Author's Idiosyncrasies of Coding Style	337
A.1	Utilities	337
A.2	Log Files	338
A.3	Labeling Trace Output	338
A.4	Naming Conventions	338
A.5	Use of <code>this</code> in Java	339
A.6	Labeling Closing Braces	339
A.7	Keep It Simple	339
A.8	The Infamous Double Equals	340
Appendix B	File Utilities	341
B.1	Header, Boilerplate, and a Log File	341
B.2	Setting Up an Output <code>PrintWriter</code> File	343
B.3	Setting Up an Input <code>Scanner</code> File	345
Appendix C	Glossary	347
C.1	The Nature of Jargon	347
C.2	Terms Used in This Book	347
Index		369

Preface

This text is intended for a second-semester course in computer science, the course that is often referred to as “CS2” in the ABET/ACM/IEEE standard curriculum. At my own university, this course has the title “Introduction to Algorithmic Design II,” and I suspect the titles are similar at many other institutions. What this course is *not intended* to be is “Introduction to Programming II,” although there will be a lot of programming discussed in the book. The conceptual material here is on the design and use of algorithms and data structures. The programming vehicle happens to be Java, but it could in fact be almost any language, and this course has no doubt been taught at some college or university in the past in FORTRAN (and probably Fortran), PL/I, Pascal, Basic, C, and C++.

What is important is that *some* programming language be used so that a rigorous use of the concepts is made and that students become facile with the basic ways in which information is manipulated to create an efficient data structure. What is hard to convey to the general public, and sometimes to students, is that the goal is to develop rigorous thinking, using a computer and a programming language so as to enforce the rigor through practice. Although familiarity with the specifics of how to be rigorous in a particular language will come from that practice, this is a secondary goal.

There are two basic methodological premises behind the treatment of the material in this book. The first is that there is no substitute for the school of hard knocks when it comes to understanding why we have some of the more sophisticated features in a modern programming language. A student who writes a program with a global variable that is updated by three different functions, and then changes the updating in two but not all three functions, with a program crash as the inevitable outcome, will learn through that experience why we want to encapsulate data into objects and limit access on what amounts to a “need to know” basis. A student who has never had that experience is much less likely to appreciate fully the concept of objects. The pedagogical problem is that it is difficult, in a fifteen-week semester, in a first-year class, to create assignments that are sufficiently difficult as to create the controlled failure mode that will lead to learning. Much of what we have to say

is tantamount to telling students that they ought to eat their vegetables because years later they will appreciate having done so.

In response to this first problem, I have tried to limit the discussion of metaphysical, moral, and theological aspects of Java (or any language) to those concepts that can be firmly grounded in practice in the second semester of a computer science curriculum. Yes, the compiler writers have done some very cool things. No, I don't see that this is necessary in a second-semester course.

The second premise of the book is that necessity is the mother of invention and that sophisticated algorithms and data structures have their natural place in the world. If one is never going to do anything more complicated than a program to balance one's personal checking account, it's not clear that anything more sophisticated than bubblesort or linear search will ever be needed. It is when the naive methods fail us that we implement more clever approaches, and I have tried to emphasize this point. I have always been of the opinion that a program that executes without crashing, even if it doesn't do exactly what is desired, is much more valuable than a program that has all the sophisticated logic programmed in ... except for a few bugs that crash the program every time. This opinion is what is usually used in the "bottleneckology" of high performance computing—focus on the most time-consuming part of the code and make that part disappear. Since something else will now become the most time-consuming, this is an iterative process until the law of diminishing returns sets in. Similarly, a program that does what is needed, albeit perhaps only on data sets of test size, can be improved piecemeal, method by method, class by class, until it can handle the real job.

For the convenience of the instructor, I have tried to map the contents of this book against the learning objectives of the curricular guidelines from ACM and IEEE in *Computer Science Curriculum 2008*. We have assumed that this is a second-semester text, and thus that the PF/FundamentalConstructs and PF/DataStructures (objectives 1, 3, 4, 5, and 7) areas have already been covered. We also assume some familiarity with DS/FunctionsRelationsAndSets. For the material in Chapter 6 and the discussion of algorithm analysis in general we assume a familiarity with differential calculus. Although we do need differentiation of logarithms and exponentials on a few occasions, we do not need any trigonometry. The crucial part of calculus that is used repeatedly at least in Chapter 6 is L'Hôpital's rule. The numbers in parentheses after the topic area in the following table are our estimates of the coverage of this book compared to the number of hours estimated in *Computer Science Curriculum 2008* to be necessary for minimal coverage of the topic.

Topic Area	Objectives	Our Chapters
DS/GraphsAndTrees (4)	1–4	9, 10, 12, 13
PF/AlgorithmicProblemSolving (6)	7–11	throughout
PF/DataStructures (7)	3–9	throughout
PF/Recursion (4)	1–6	Chapter 8
PF/ObjectOriented (7)	1–4	throughout
AL/AlgorithmsAndComplexity (2)	1–3	Chapter 6
AL/AlgorithmicStrategies (4)	1–4	3, 4, 6, 10, 11, 12
AL/FundamentalAlgorithms (6)	1, 4–7	3, 4, 6, 8, 10, 11, 12
PL/ObjectOrientedProgramming (5)	1–3, 4, 6–7	throughout

On the flip side, most of the Programming Fundamentals (PF) topics are covered throughout this book. For the topics that are covered more in selected chapters but are not pervasive throughout, we also index the other way around.

Chapters	Topic Area
Chapter 3	AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 4	AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 5	
Chapter 6	AL/AlgorithmsAndComplexity AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 8	PF/Recursion AL/FundamentalAlgorithms
Chapter 9	DS/GraphsAndTrees
Chapter 10	DS/GraphsAndTrees AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 11	AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 12	DS/GraphsAndTrees AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 13	DS/GraphsAndTrees

I have generally found that I can teach straight through in chapter order, although the vagaries of semester scheduling sometimes make it necessary to move

the position of Chapter 6 around. Chapters 7 and 8, except for Section 7.5 on priority queues, do not depend on the chapter on asymptotics, and I sometimes move that material ahead of Chapter 6 in order to do the first midterm exam only on completed chapters. A decision like this is usually based on the calendar, and I have found our fall and spring semesters to be slightly different.

Additional resources for students and instructors are provided at go.jblearning.com/Buell. Students and instructors may download source code related to the text. In addition, PowerPoint Lecture Outlines, Solutions to end-of-chapter Exercises, and a Test Bank are available for free instructor download.

For the mistakes and failings of this book I will take full responsibility, but for whatever is good about this book I must thank in part all people who have helped me become a better programmer and a better teacher. The students of the last three years who have looked at versions of this text and made comments have all contributed. My colleagues at the Institute for Defense Analyses have played a part in this text, as have the reviewers and the students who have commented, caught my errors, and helped this be better than it might have been. Thank you to Dr. Alice Armstrong, DSc, Shippensburg University; John Boyland, PhD, University of Wisconsin–Milwaukee; and H el ene Martin, Garfield High School for reviewing the manuscript. I owe a special debt to the copyeditors and proofreaders at Jones & Bartlett Learning. Their work has greatly improved the consistency of the text. We all might like to think we write well; the editors are those who make it appear that we actually do. I would also like to thank the editors at Jones & Bartlett Learning; Tim Anderson, Senior Acquisitions Editor; Amy Rose, Director of Production; and Amy Bloom, Managing Editor. Finally, I thank my wife Mary Ann for being patient with me for my many hours spent in the office staring at the screen.

Duncan Buell
Columbia, South Carolina