Algorithms

ABSOLUTE BEGINNER'S GUIDE

No experience necessary!

Kirupa Chinnathambi

FREE SAMPLE CHAPTER
Absolute Beginner’s Guide to Algorithms

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com

Library of Congress Control Number: 2023947403

Copyright © 2024 Pearson Education, Inc.

Hoboken, NJ

Cover image: Rozdesign/Shutterstock

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions.

ISBN-10: 0-13-822229-0

$PrintCode
Pearson’s Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world’s leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

• Everyone has an equitable and lifelong opportunity to succeed through learning.
• Our educational products and services are inclusive and represent the rich diversity of learners.
• Our educational content accurately reflects the histories and experiences of the learners we serve.
• Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview).

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

• Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.

Register your copy of *Absolute Beginner’s Guide to Algorithms* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN *(9780138222291)* and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.
Figure Credits

FM – Chapter 25: Cute Little Chicks, Rozdesign/Shutterstock

Chapter 1: Portrait of adorable raccoon isolated on white background, sonsedskaya/123RF

Chapter 2: Happy smiley emoji Vector, MaxxGraphic/Shutterstock

FIG04-13, FIG23-01: Cassius Marcellus Coolidge

FIG05-01, FIG09-23, FIG14-04, FIG16-18, FIG19-26, FIG19-27: Microsoft

FIG07-01: Used with permission from Randall Munroe (xcdc) granted by The Gernert Company

Chapter 7, 10, FIG15-01: chick hitching out of an egg, Shahena z/Shutterstock

FIG10-01: Atlassian

FIG10-29: Notion Labs, Inc.

FIG12-24: Apple, Inc.

FIG12-25: The New York Times Company

FIG12-01, FIG12-22, FIG12-23: Google, Inc.

FIG25-01: Winsor McCay
Contents at a Glance

Part I  Data Structures
1  Introduction to Data Structures ...........................................................1
2  Big-O Notation and Complexity Analysis ............................................7
3  Arrays ........................................................................................................17
4  Linked Lists ................................................................................................35
5  Stacks ........................................................................................................53
6  Queues .......................................................................................................61
7  Trees ...........................................................................................................69
8  Binary Trees ...............................................................................................79
9  Binary Search Trees ...............................................................................91
10 Heaps .........................................................................................................113
11 Hashtable (aka Hashmap or Dictionary) ................................................137
12 Trie (aka Prefix Tree) ..............................................................................155
13 Graphs ......................................................................................................183

Part II  Algorithms
14 Introduction to Recursion ........................................................................199
15 Fibonacci and Going Beyond Recursion ..............................................207
16 Towers of Hanoi .......................................................................................221
17 Search Algorithms and Linear Search ..................................................235
18 Faster Searching with Binary Search ......................................................243
19 Binary Tree Traversal .............................................................................259
20 Depth-First Search (DFS) and Breadth-First Search (BFS) .................281
21 Quicksort ..................................................................................................309
22 Bubblesort ...............................................................................................325
23 Insertion Sort ..........................................................................................335
24 Selection Sort ..........................................................................................355
25 Mergesort ..................................................................................................371
26 Conclusion ...............................................................................................383

Index ...........................................................................................................387
# Table of Contents

1 Data Structures

1 Introduction to Data Structures ................................................................. 1
   Right Tool for the Right Job ................................................................. 2
   Back to Data Structures ........................................................................ 5
   Conclusion .............................................................................................. 6
   Some Additional Resources ..................................................................... 6

2 Big-O Notation and Complexity Analysis ................................................. 7
   It's Example Time .................................................................................. 8
   It's Big-O Notation Time! ....................................................................... 11
   Conclusion ............................................................................................. 15
   Some Additional Resources ................................................................... 15

3 Arrays ............................................................................................................. 17
   What Is an Array? .................................................................................. 18
   Adding an Item ....................................................................................... 19
   Deleting an Item .................................................................................... 21
   Searching for an Item ............................................................................. 22
   Accessing an Item ................................................................................ 24
   Array Implementation / Use Cases ....................................................... 24
   Arrays and Memory ............................................................................... 26
   Performance Considerations .................................................................. 30
   Access .................................................................................................... 31
   Insertion .................................................................................................. 31
   Deletion .................................................................................................. 32
   Searching ................................................................................................ 32
   Conclusion ............................................................................................. 32
   Some Additional Resources ................................................................ 33

4 Linked Lists .................................................................................................. 35
   Meet the Linked List ................................................................................ 36
   Finding a Value ....................................................................................... 36
   Adding Nodes ........................................................................................ 38
   Deleting a Node ..................................................................................... 39
<table>
<thead>
<tr>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked List: Time and Space Complexity</td>
<td>40</td>
</tr>
<tr>
<td>Deeper Look at the Running Time</td>
<td>41</td>
</tr>
<tr>
<td>Space Complexity</td>
<td>41</td>
</tr>
<tr>
<td>Linked List Variations</td>
<td>41</td>
</tr>
<tr>
<td>Singly Linked List</td>
<td>42</td>
</tr>
<tr>
<td>Doubly Linked List</td>
<td>42</td>
</tr>
<tr>
<td>Circular Linked List</td>
<td>42</td>
</tr>
<tr>
<td>Skip List</td>
<td>44</td>
</tr>
<tr>
<td>Implementation</td>
<td>44</td>
</tr>
<tr>
<td>Conclusion</td>
<td>52</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>52</td>
</tr>
<tr>
<td><strong>5 Stacks</strong></td>
<td>53</td>
</tr>
<tr>
<td>Meet the Stack</td>
<td>54</td>
</tr>
<tr>
<td>A JavaScript Implementation</td>
<td>56</td>
</tr>
<tr>
<td>Stacks: Time and Space Complexity</td>
<td>58</td>
</tr>
<tr>
<td>Runtime Performance</td>
<td>59</td>
</tr>
<tr>
<td>Memory Performance</td>
<td>59</td>
</tr>
<tr>
<td>Conclusion</td>
<td>59</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>60</td>
</tr>
<tr>
<td><strong>6 Queues</strong></td>
<td>61</td>
</tr>
<tr>
<td>Meet the Queue</td>
<td>62</td>
</tr>
<tr>
<td>A JavaScript Implementation</td>
<td>64</td>
</tr>
<tr>
<td>Queues: Time and Space Complexity</td>
<td>66</td>
</tr>
<tr>
<td>Runtime Performance</td>
<td>66</td>
</tr>
<tr>
<td>Memory Performance</td>
<td>67</td>
</tr>
<tr>
<td>Conclusion</td>
<td>67</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>68</td>
</tr>
<tr>
<td><strong>7 Trees</strong></td>
<td>69</td>
</tr>
<tr>
<td>Trees 101</td>
<td>70</td>
</tr>
<tr>
<td>Height and Depth</td>
<td>75</td>
</tr>
<tr>
<td>Conclusion</td>
<td>77</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>77</td>
</tr>
</tbody>
</table>
8 Binary Trees ................................................................. 79
  Meet the Binary Tree .................................................. 80
  Rules Explained .......................................................... 80
  Binary Tree Variants .................................................. 82
  What about Adding, Removing, and Finding Nodes? ....... 86
A Simple Binary Tree Implementation ......................... 86
Conclusion ........................................................................ 89
Some Additional Resources ........................................... 89

9 Binary Search Trees ..................................................... 91
  It’s Just a Data Structure .............................................. 93
    Adding Nodes .......................................................... 93
    Removing Nodes ...................................................... 97
  Implementing a Binary Search Tree ............................. 103
  Performance and Memory Characteristics .................. 110
Conclusion ....................................................................... 112
Some Additional Resources .......................................... 112

10 Heaps ................................................................. 113
  Meet the Heap .......................................................... 114
    Common Heap Operations ...................................... 116
  Heap Implementation ............................................... 126
    Heaps as Arrays ...................................................... 127
    The Code ................................................................. 128
  Performance Characteristics ...................................... 132
    Removing the Root Node ......................................... 132
    Inserting an Item .................................................... 133
    Performance Summary ........................................... 134
Conclusion ....................................................................... 134
Some Additional Resources .......................................... 135

11 Hashtable (aka Hashmap or Dictionary) ......................... 137
  A Very Efficient Robot ................................................ 138
  From Robots to Hashing Functions .............................. 142
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Hashing Functions to Hashtables</td>
<td>145</td>
</tr>
<tr>
<td>Adding Items to Our Hashtable</td>
<td>146</td>
</tr>
<tr>
<td>Reading Items from Our Hashtable</td>
<td>147</td>
</tr>
<tr>
<td>JavaScript Implementation/Usage</td>
<td>148</td>
</tr>
<tr>
<td>Dealing with Collisions</td>
<td>150</td>
</tr>
<tr>
<td>Performance and Memory</td>
<td>151</td>
</tr>
<tr>
<td>Conclusion</td>
<td>153</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>154</td>
</tr>
<tr>
<td><strong>12 Trie (aka Prefix Tree)</strong></td>
<td>155</td>
</tr>
<tr>
<td>What Is a Trie?</td>
<td>156</td>
</tr>
<tr>
<td>Inserting Words</td>
<td>157</td>
</tr>
<tr>
<td>Finding Items</td>
<td>162</td>
</tr>
<tr>
<td>Deleting Items</td>
<td>165</td>
</tr>
<tr>
<td>Diving Deeper into Tries</td>
<td>167</td>
</tr>
<tr>
<td>Many More Examples Abound!</td>
<td>172</td>
</tr>
<tr>
<td>Implementation Time</td>
<td>173</td>
</tr>
<tr>
<td>Performance</td>
<td>179</td>
</tr>
<tr>
<td>Conclusion</td>
<td>181</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>181</td>
</tr>
<tr>
<td><strong>13 Graphs</strong></td>
<td>183</td>
</tr>
<tr>
<td>What Is a Graph?</td>
<td>184</td>
</tr>
<tr>
<td>Graph Implementation</td>
<td>190</td>
</tr>
<tr>
<td>Representing Nodes</td>
<td>190</td>
</tr>
<tr>
<td>The Code</td>
<td>192</td>
</tr>
<tr>
<td>Conclusion</td>
<td>196</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>197</td>
</tr>
</tbody>
</table>

## II Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>14 Introduction to Recursion</strong></td>
<td>199</td>
</tr>
<tr>
<td>Our Giant Cookie Problem</td>
<td>200</td>
</tr>
<tr>
<td>Recursion in Programming</td>
<td>202</td>
</tr>
</tbody>
</table>
15 Fibonacci and Going Beyond Recursion ............................................. 207
  Recursively Solving the Fibonacci Sequence ....................................... 209
  Recursion with Memoization ................................................................. 213
  Taking an Iteration-Based Approach .................................................... 215
  Going Deeper on the Speed ................................................................. 217
  Conclusion ............................................................................................ 218
  Some Additional Resources ................................................................... 219

16 Towers of Hanoi ..................................................................................... 221
  How Towers of Hanoi Is Played ............................................................ 222
  The Single Disk Case ............................................................................. 223
  It’s Two Disk Time .................................................................................. 224
  Three Disks ............................................................................................ 225
  The Algorithm ......................................................................................... 228
  The Code Solution .................................................................................. 229
  Check Out the Recursiveness! ................................................................. 231
  It’s Math Time ......................................................................................... 232
  Conclusion ............................................................................................ 234
  Some Additional Resources ................................................................... 234

17 Search Algorithms and Linear Search .................................................... 235
  Linear Search ......................................................................................... 236
    Linear Search at Work ........................................................................... 236
    JavaScript Implementation ...................................................................... 238
    Runtime Characteristics .......................................................................... 239
  Conclusion ............................................................................................ 241
  Some Additional Resources ................................................................... 241
18 Faster Searching with Binary Search ......................................................... 243
   Binary Search in Action ................................................................. 244
   Sorted Items Only, Please ............................................................. 244
   Dealing with the Middle Element ................................................... 245
   Dividing FTW! .............................................................................. 247
   The JavaScript Implementation ....................................................... 250
      Iterative Approach .................................................................... 250
      Recursive Approach ................................................................... 251
   Example of the Code at Work ....................................................... 252
   Runtime Performance ..................................................................... 254
   Conclusion .................................................................................... 257
   Some Additional Resources ........................................................... 257

19 Binary Tree Traversal ................................................................. 259
   Breadth-First Traversal .................................................................. 260
   Depth-First Traversal .................................................................... 265
   Implementing Our Traversal Approaches ........................................ 270
      Node Exploration in the Breadth-First Approach ......................... 270
      Node Exploration in the Depth-First Approach ............................. 272
      Looking at the Code .................................................................. 273
   Performance of Our Traversal Approaches .................................... 278
   Conclusion .................................................................................... 279
   Some Additional Resources ........................................................... 279

20 Depth-First Search (DFS) and Breadth-First Search (BFS) ................. 281
   A Tale of Two Exploration Approaches .......................................... 282
      Depth-First Search Overview ..................................................... 283
      Breadth-First Search Overview .................................................. 284
      Yes, They Are Different! ............................................................. 284
   It’s Example Time .......................................................................... 285
      Exploring with DFS .................................................................. 285
      Exploring with BFS .................................................................. 293
   When to Use DFS? When to Use BFS? ........................................... 298
   A JavaScript Implementation ....................................................... 300
      Using the Code ......................................................................... 305
      Implementation Detail ............................................................... 306
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Details</td>
<td>307</td>
</tr>
<tr>
<td>Conclusion</td>
<td>308</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>308</td>
</tr>
<tr>
<td><strong>21 Quicksort</strong></td>
<td>309</td>
</tr>
<tr>
<td>A Look at How Quicksort Works</td>
<td>310</td>
</tr>
<tr>
<td>A Simple Look</td>
<td>310</td>
</tr>
<tr>
<td>Another Simple Look</td>
<td>314</td>
</tr>
<tr>
<td>It's Implementation Time</td>
<td>319</td>
</tr>
<tr>
<td>Performance Characteristics</td>
<td>322</td>
</tr>
<tr>
<td>Time Complexity</td>
<td>323</td>
</tr>
<tr>
<td>Space Complexity</td>
<td>323</td>
</tr>
<tr>
<td>Stability</td>
<td>323</td>
</tr>
<tr>
<td>Conclusion</td>
<td>323</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>324</td>
</tr>
<tr>
<td><strong>22 Bubblesort</strong></td>
<td>325</td>
</tr>
<tr>
<td>How Bubblesort Works</td>
<td>326</td>
</tr>
<tr>
<td>Walkthrough</td>
<td>329</td>
</tr>
<tr>
<td>The Code</td>
<td>333</td>
</tr>
<tr>
<td>Conclusion</td>
<td>333</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>334</td>
</tr>
<tr>
<td><strong>23 Insertion Sort</strong></td>
<td>335</td>
</tr>
<tr>
<td>How Insertion Sort Works</td>
<td>336</td>
</tr>
<tr>
<td>One More Example</td>
<td>347</td>
</tr>
<tr>
<td>Algorithm Overview and Implementation</td>
<td>349</td>
</tr>
<tr>
<td>Performance Analysis</td>
<td>351</td>
</tr>
<tr>
<td>Conclusion</td>
<td>353</td>
</tr>
<tr>
<td>Some Additional Resources</td>
<td>353</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>Selection Sort</td>
<td>355</td>
</tr>
<tr>
<td></td>
<td>Selection Sort Walkthrough</td>
<td>356</td>
</tr>
<tr>
<td></td>
<td>Algorithm Deep Dive</td>
<td>364</td>
</tr>
<tr>
<td></td>
<td>The JavaScript Implementation</td>
<td>366</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
<td>369</td>
</tr>
<tr>
<td></td>
<td>Some Additional Resources</td>
<td>369</td>
</tr>
<tr>
<td>25</td>
<td>Mergesort</td>
<td>371</td>
</tr>
<tr>
<td></td>
<td>How Mergesort Works</td>
<td>372</td>
</tr>
<tr>
<td></td>
<td>Mergesort: The Algorithm Details</td>
<td>379</td>
</tr>
<tr>
<td></td>
<td>Looking at the Code</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
<td>381</td>
</tr>
<tr>
<td></td>
<td>Some Additional Resources</td>
<td>382</td>
</tr>
<tr>
<td>26</td>
<td>Conclusion</td>
<td>383</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>387</td>
</tr>
</tbody>
</table>
Acknowledgments

As I found out, getting a book like this out the door is no small feat. It involves a bunch of people in front of (and behind) the camera who work tirelessly to turn my ramblings into the beautiful pages that you are about to see. To everyone at Pearson who made this possible, thank you!

With that said, there are a few people I’d like to explicitly call out. First, I’d like to thank Kim Spenceley for making this book possible, Chris Zahn for meticulously ensuring everything is human-readable, Carol Lallier for her excellent copyediting, and Loretta Yates for helping make the connections that made all of this happen years ago. The technical content of this book has been reviewed in great detail by my long-time collaborators Cheng Lou and Ashwin Raghav.

Lastly, I’d like to thank my parents for having always encouraged me to pursue creative hobbies like painting, writing, playing video games, and writing code. I wouldn’t be half the rugged indoorsman I am today without their support 😊

Dedication

To my wife, Meena!
(For her support and timely insights throughout this book!)

About the Author

Kirupa Chinnathambi has spent most of his life teaching others to love web development as much as he does. He founded KIRUPA, one of the Web’s most popular free web development education resources, serving 210,000+ registered members. Now a product manager at Google, he has authored several books, including Learning React. He holds a B.S. in computer science from MIT.
Tech Editors

**Cheng Lou** is a software engineer who has worked on various projects, such as ReactJS, Meta Messenger and ReScript. He’s been passionate about graphics and general programming since the early Flash days, and is eager to keep its spirit alive.

Personal site: chenglou.me

Twitter / X: twitter.com/_chenglou

**Aswhin Raghav** serves as the Engineering lead for Project IDX at Google. He’s also to blame for those pesky Firebase APIs. He’s been building software and software teams for two decades at Twitter, Zynga, Thoughtworks, and Intel. He considers himself a specialist at building developer tools and facing the wrath of unhappy developers around the world. He lives with his wife and two kids.

Personal site: ashwinraghav.me

Twitter / X: twitter.com/ashwinraghav
This page intentionally left blank
INTRODUCTION TO DATA STRUCTURES

Programming is all about taking data and manipulating it in all sorts of interesting ways. Now, depending on what we are doing, our data needs to be represented in a form that makes it easy for us to actually use. This form is better known as a data structure. As we will see shortly, data structures give the data we are dealing with a heavy dose of organization and scaffolding. This makes manipulating our data easier and (often) more efficient. In the following sections, we find out how that is possible!

Onward!
Right Tool for the Right Job

To better understand the importance of data structures, let’s look at an example. Here is the setup. We have a bunch of tools and related gadgets (Figure 1-1).

![Figure 1-1](image)

**FIGURE 1-1**

*Tools, tools, tools*

What we want to do is store these tools for easy access later. One solution is to simply throw all of the tools in a giant cardboard box and call it a day (Figure 1-2).

![Figure 1-2](image)

**FIGURE 1-2**

*Tools, meet box!*
If we want to find a particular tool, we can rummage through our box to find what we are looking for. If what we are looking for happens to be buried deep in the bottom of our box, that’s cool. With enough rummaging (Figure 1-3)—and possibly shaking the box a few times—we will eventually succeed.

FIGURE 1-3
A rummager!

Now, there is a different approach we can take. Instead of throwing things into a box, we could store them in something that allows for better organization. We could store all of these tools in a toolbox (Figure 1-4).
A toolbox is like the Marie Kondo of the DIY world, with its neat compartments and organized bliss. Sure, it might take a smidge more effort to stow things away initially, but that’s the price we pay for future tool-hunting convenience. No more digging through the toolbox like a raccoon on a midnight snack raid.

We have just seen two ways to solve our problem of storing our tools. If we had to summarize both approaches, it would look as follows:

- **Storing Tools in a Cardboard Box**
  - Adding items is very fast. We just drop them in there. Life is good.
  - Finding items is slow. If what we are looking for happens to be at the top, we can easily access it. If what we are looking for happens to be at the bottom, we’ll have to rummage through almost all of the items.
  - Removing items is slow as well. It has the same challenges as finding items. Things at the top can be removed easily. Things at the bottom may require some extra wiggling and untangling to safely get out.

- **Storing Tools in a Toolbox**
  - Adding items to our box is slow. There are different compartments for different tools, so we need to ensure the right tool goes into the right location.
  - Finding items is fast. We go to the appropriate compartment and pick the tool from there.
• Removing items is fast as well. Because the tools are organized in a good location, we can retrieve them without any fuss.

What we can see is that both our cardboard box and toolbox are good for some situations and bad for other situations. There is no universally right answer. If all we care about is storing our tools and never really looking at them again, stashing them in a cardboard box is the right choice. If we will be frequently accessing our tools, storing them in the toolbox is more appropriate.

**Back to Data Structures**

When it comes to programming and computers, deciding which data structure to use is similar to deciding whether to store our tools in a cardboard box or a toolbox. Every data structure we will encounter is good for some situations and bad for other situations (Figure 1-5).

**FIGURE 1-5**

*A good fit in this case*
Knowing which data structure to use and when is an important part of being an effective developer, and the data structures we need to deeply familiarize ourselves with are

- Arrays
- Linked lists
- Stacks
- Queues
- Introduction to trees
- Binary trees
- Binary search trees
- Heap data structure
- Hashtable (aka hashmap or dictionary)
- Trie (aka prefix tree)

**Conclusion**

Over the next many chapters, we’ll learn more about what each data structure is good at and, more important, what types of operations each is not very good at. By the end of it, you and I will have created a mental map connecting the right data structure to the right programming situation we are trying to address.

**SOME ADDITIONAL RESOURCES**

- Ask a question: [https://forum.kirupa.com](https://forum.kirupa.com)
Index

A
accessing items in arrays, 24, 31
active numbers, insertion sort, 347
acyclic graphs, 188–189
adding
items
to arrays, 19–21, 31
to hashtables (dictionaries), 146–147
nodes
to binary search trees, 93–97, 103
to linked lists, 38–39
arrays, 17, 32–33
accessing items in arrays, 24, 31
adding items, 19–21, 31
defined, 18
deleting items, 21–22, 32
heaps as, 126–132
implementing, 24–25
inserting items into arrays, 19–21, 31
memory, 26–30
performance, 30–31
searching for items, 22–23, 32
autocomplete
element of, 155–156
tries (prefix trees), 169

B
balanced binary trees, 85, 116
BFS (Breadth-First Searches), 281–283, 308
coding, 300–306
implementing, 306–307
JavaScript, 300–306
memory complexity, 307–308
overview, 284
performance, 307–308
runtime complexity, 307
walkthrough, 293–298
when to use, 298–300
bidirectional graphs, 185
Big-O notation, 7, 11, 15
complexity graphs, 12–14
constants, 9–10
examples, 8–10
input graphs, 12–14
linear relationships, 11–12
linearly, scaling, 8–9
O(1)—Constant
Complexity, 13
O(2^n)—Exponential
Complexity, 14
odd/even numbers, 8–9
O(log n)—Logarithmic
Complexity, 13
O(n log n)—Linearithmic
Complexity, 14
O(n^2)—Quadratic
Complexity, 14
O(n!)—Factorial
Complexity, 14
O(n)—Linear Complexity,
13–14
upper bound scenarios, 11
worst-case scenarios, 11
Big-Omega notation, 14
Big-Theta notation, 14
binary searches, 32, 243, 257
coding, 250–253
dividing operations, 247–250
iteration, 250–251
JavaScript, 250–253
middle elements, 245–246, 252–253
operations, 244
recursive operations, 251
runtime, 254–257
sorted items, 244
walkthrough, 252–253
binary search trees, 93, 112
coding, 103–109
defined, 91–92
example of, 91–92
implementing, 103–109
JavaScript, 103–109
memory, 110–111
nodes
adding, 93–97, 103
leaf nodes, 97–98
removing, 97–103
single child nodes, 99–100
two children nodes, 101–103
performance, 110–111
binary trees, 79, 259–260
balanced binary trees, 85, 116

2n space, mergesort, 379

Numbers

accessing items in arrays, 24, 31
active numbers, insertion sort, 347
acyclic graphs, 188–189
adding
items
to arrays, 19–21, 31
to hashtables (dictionaries), 146–147
nodes
to binary search trees, 93–97, 103
to linked lists, 38–39
arrays, 17, 32–33
accessing items in arrays, 24, 31
adding items, 19–21, 31
defined, 18
deleting items, 21–22, 32
heaps as, 126–132
implementing, 24–25
inserting items into arrays, 19–21, 31
memory, 26–30
performance, 30–31
searching for items, 22–23, 32
autocomplete
element of, 155–156
tries (prefix trees), 169
child nodes, 71–73
single child nodes, 99–100
two children nodes, 101–103
circular linked lists, 42–43
coding
arrays, implementing, 24–25
BFS, 300–306
binary search, 250–253
binary search trees, 103–109
binary tree traversals, 273–274
breadth-first traversals, 274–276
depth-first traversals, 276–278
binary trees, 86–89
breadth-first traversals, 274–276
bubblesort, 333
depth-first traversals, 276–278
DFS, 300–306
Fibonacci sequences calculating, 207–208
iteration, 215–216
recursive operations, 209–212
graphs, 192–196
hashtables (dictionaries), 148–150
heaps, 126–132
insertion sort, 349–351
JavaScript, 229–232
linear search, 238–239
linked lists, 44–51
mergesort, 380–381
queues, 64–66
quicksort, 319–322
recursion
function calls, 202
terminating conditions, 203–205
selection sort, 366–369
stacks, 56–58
tries (prefix trees), 173–179

collisions, hashtables (dictionaries), 150–151
complete binary trees, 83–84
complexity graphs, Big-O notation, 12–14
connected trees, graphs, 188–189
connecting nodes, 70–71
Constant Complexity O(1), 13
constants, Big-O notation, 9–10
cycles/cyclic graphs, 187–188

dags (directed acyclic graphs), 188–189
data structures, 1
importance of, 2–5
toolbox metaphor, 4–5
types of, 5–6
degenerate binary trees, 85
deleting items
from arrays, 21–22, 32
from tries (prefix trees), 165–167
nodes from linked lists, 39–40
depth-first traversals, 279
implementing, 270, 274–276
node exploration, 270–271
performance, 278
walkthrough, 260–265
bubblesort, 325
coding, 333
JavaScript, 333–334
memory, 324, 334, 353, 369, 382
performance, 324, 334
speed, 353, 369, 382
walkthrough, 326–332
cache performance, queues, 67
calculating Fibonacci sequences, 217–218
dequeue (deletion), 66
DFS (Depth-First Searches), 281–283, 308
coding, 300–306
implementing, 306–307
JavaScript, 300–306
memory complexity, 307
overview, 283
performance, 307
runtime complexity, 307
walkthrough, 285–291
when to use, 298–300
dictionaries (hashables), 137, 153
  adding items, 146–147
coding, 148–150
collisions, 150–151
defined, 145–146
  hashing functions, 142–145
  implementing, 148–150
JavaScript, 148–150
memory, 151–153
performance, 151–153
reading items, 147–148
robot example, 138–142
digraphs (directional graphs), 187
divide-and-conquer algorithms
mergesort, 372
quicksort, 309
dividing operations, binary search, 247–250
doubly linked lists, 42
dynamic memory allocation, queues, 67

E
edges, 70–71, 185
tenque (insertion), 66
even/odd numbers, Big-O notation, 8–9
Exponential Complexity
$O(2^n)$, 14

F
Factorial Complexity $O(n!)$, 14
Fibonacci sequences, 218–219
  calculating, 207–208, 217–218
  function calls, expense of, 212–213
  iteration, 215–216, 217–218
  memoization, 213–215, 218
  recursively solving, 209–212
FIFO (First In, First Out), 62
  finding
  items in tries (prefix trees), 162–165
  values in linked lists, 36–37
  full binary trees, 82–83
  function calls
  expense of, 212–213
  recursive function calls, 202
  giant cookie example,
  recursion, 200–202
  global linear searches, 240–241
  graphs, 183, 196
  acyclic graphs, 188–189
  BFS, 281–283, 308
  coding, 300–306
  implementing, 306–307
  JavaScript, 300–306
  memory complexity, 307–308
  overview, 284
  performance, 307–308
  runtime complexity, 307
  walkthrough, 293–298
  when to use, 298–300
  bidirectional graphs, 185
coding, 192–196
connected trees, 188–189
cycles/cyclic graphs, 187–188
dags, 188–189
defined, 184–189
DFS, 281–283, 308
  coding, 300–306
  implementing, 306–307
  JavaScript, 300–306
  memory complexity, 307
  overview, 283
  performance, 307
  runtime complexity, 307
  walkthrough, 285–291
  when to use, 298–300
digraphs, 187
direct, 185
example of, 184–189
implementing, 190, 192–196
JavaScript, 192–196
nodes, 184–185, 190–191
relationships, 187
unconnected trees, 188–189
undirected graphs, 185–186

H
Hanoi, Towers of, 221, 234
algorithm, 228–229
  calculating number of moves, 232–234
coding, 229–232
JavaScript, 229–232
playing, 222–223
recursive operations, 231–232
single disk game play, 223
two disk game play, 224–225
two disk game play, 224–225

G
hashtables (dictionaries), 137, 153
  adding items, 146–147
coding, 148–150
collisions, 150–151
defined, 145–146
  hashing functions, 142–145
  implementing, 148–150
JavaScript, 148–150
memory, 151–153
performance, 151–153
reading items, 147–148
robot example, 138–142

H
Heaps, 113–114, 134–135
as arrays, 126–132
coding, 126–132
defined, 114–116
example of, 114–116
implementing, 126–132
inserting items, 133–134
JavaScript, 126–132
mini-heaps, 115
nodes
  inserting, 116–121
  removing, 122–126, 132–133
  root nodes, 122–126, 132–133

I

performance, 132–134
properties of, 115
as task organizers, 134–135
heapsort
memory, 324, 334, 353, 369, 381
performance, 324, 334
speed, 353, 369, 381
height/depth of nodes, 75–76
input graphs, Big-O notation, 12–14
inserting items into arrays, 19–21, 31
insertion sort, 335–336
active numbers, 347
coding, 349–351
implementing, 349–351
JavaScript, 349–351
memory, 324, 334, 353, 369, 382
performance, 324, 334, 351–352
speed, 353, 369, 382
walkthrough, 336–348
IP routing, tries (prefix trees), 170–171
iteration
binary search, 250–251
Fibonacci sequences, 215–216, 217–218
JavaScript
arrays, implementing, 24–25
BFS, 300–306
binary searches, 250–253
binary search trees, 103–109
binary tree traversals, 273–274
breadth-first traversals, 274–276
depth-first traversals, 276–278
binary trees, 86–89
breadth-first traversals, 274–276
bubblesort, 333
depth-first traversals, 276–278
DFS, 300–306
Fibonacci sequences
calculating, 207–208
iteration, 215–216
recursive operations, 209–212
graphs, 192–196
hashtables (dictionaries), 148–150
heaps, 126–132
insertion sort, 349–351
linear search, 238–239
linked lists, 44–51
mergesort, 380–381
queues, 64–66
quicksort, 319–322
recursion
function calls, 202
terminating conditions, 203–205
selection sort, 366–369
stacks, 56–58
Towers of Hanoi, 229–232
tries (prefix trees), 173–179
linked lists, 35, 52
circular linked lists, 42–43
coding, 44–51
defined, 36
doubly linked lists, 42
finding values, 36–37
implementing, 44–51
JavaScript, 44–51
nodes, 36–37
adding, 38–39
deleting, 39–40
unreachable nodes, 40
performance, 41
singly linked lists, 42
skip lists, 44
space complexity, 41
time complexity, 41
variations, 41–44
Logarithmic Complexity
$O(\log n)$, 13
maps
BFS, 281–283, 308
coding, 300–306
implementing, 306–307
JavaScript, 300–306
memory complexity, 307–308
overview, 284
performance, 307–308
runtime complexity, 307
walkthrough, 293–298
when to use, 298–300
DFS, 281–283, 308
coding, 300–306
implementing, 306–307
JavaScript, 300–306
memory complexity, 307
overview, 283
performance, 307
runtime complexity, 307
walkthrough, 285–291
when to use, 298–300
memoization, Fibonacci
sequences, 213–215, 218
memory
arrays, 26–30
BFS, 307–308
binary search trees, 110–111
bubblesort, 324, 334, 353, 369, 382
DFS, 307
hashtables (dictionaries), 151–153
heapsort, 324, 334, 353, 369, 381
insertion sort, 324, 334, 353, 369, 382
memory overhead, queues, 67
mergesort, 324, 334, 353, 369, 381
queues, 67
quicksort, 324, 334, 353, 369, 381
selection sort, 324, 334, 353, 369, 382
stacks, 59
timsort, 324, 334, 353, 369, 382
mergesort, 371–372
2n space, 379
coding, 380–381
divide-and-conquer algorithms, 372
JavaScript, 380–381
memory, 324, 334, 353, 369, 381
operations, 379–380
performance, 324, 334
speed, 353, 369, 381
stable sorts, 379–380
tree depth, 371–372
walkthrough, 372–379
middle elements, binary search, 245–246, 252–253
mini-heaps, 115

N
network routing tables, tries (prefix trees), 170–171
nodes
binary search trees
adding nodes, 93–97, 103
removing nodes, 97–103
binary trees, 86, 89
child nodes, 71–73
    single child nodes, 99–100
two children nodes, 101–103
connecting, 70–71
deleting, 39–40
edges, 70–71
graphs, 184–185, 190–191
heaps
    inserting items, 133–134
    inserting nodes, 116–121
    removing nodes, 122–126, 132–133
height/depth, 75–76
leaf nodes, 74, 97–98
linked lists, 36–37, 38–39
parent nodes, 72–73
root nodes, 74, 122–126, 132–133
sibling nodes, 73
single child nodes, 99–100
traversing, 37
    breadth-first traversals, 270–271
    depth-first traversals, 272–273
trees, 70–74
    two children nodes, 101–103
unreachable nodes, 40

O
O(1)—Constant Complexity, 13
O(2^n)—Exponential Complexity, 14
odd/even numbers, Big-O notation, 8–9
O(log n)—Logarithmic Complexity, 13
O(n log n)—Linearithmic Complexity, 14
O(n^2)—Quadratic Complexity, 14
O(n!)—Factorial Complexity, 14
O(n)—Linear Complexity, 13–14
P
parent nodes, 72–73
peek operations
    queues, 66
    stacks, 59
perfect binary trees, 84–85
performance
    arrays, 30–31
    BFS, 307–308
binary search trees, 110–111
breadth-first traversals, 278
bubblesort, 324, 334
depth-first traversals, 278
DFS, 307
hashtables (dictionaries), 151–153
heaps, 132–134
heapsort, 324, 334, 353, 369, 381
insertion sort, 324, 334, 351–352
linked lists, 41
mergesort, 324, 334
queues, 66
caches, 67
memory, 67
runtime, 66–67
quicksort, 322, 324, 334
selection sort, 324, 334, 351–352
stacks, 58
memory, 59
runtime, 59
timsort, 324, 334
tries (prefix trees), 179–181
pop operations, stacks, 59
predictive text, tries (prefix trees), 169
autocomplete, 169
coding, 173–179
defined, 156–157
deleting items, 165–167
finding items, 162–165
implementing, 173–179
inserting words, 157–162
IP routing, 170–171
 JavaScript, 173–179
 network routing tables, 170–171
 performance, 179–181
 predictive text, 169
 spell checking/correction, 170
 word games/puzzles, 171–172
 push operations, stacks, 59
 puzzles/word games, tries (prefix trees), 171–172

R
 reading items from hash tables (dictionaries), 147–148
 recursion, 199, 206
 coding
 function calls, 202
 terminating conditions, 203–205
 function calls, 202
 giant cookie example, 200–202
 JavaScript
 function calls, 202
 terminating conditions, 203–205
 terminating conditions, 203–205
 recursive operations
 binary search, 251
 Fibonacci sequences, 209–212
 Towers of Hanoi, 231–232
 Redo/Undo, 53
 relationships, graphs, 187
 removing nodes
 from binary search trees, 97–103
 from heaps, 122–126
 robot example, hashtables (dictionaries), 138–142
 root nodes, 74, 122–126, 132–133
 routing tables, tries (prefix trees), 170–171
 runtime
 BFS, 307
 binary search, 254–257
 DFS, 307
 linear search, 239
 queues, 66–67
 stacks, 59

S
 searches
 BFS, 281–283, 308
 coding, 300–306
 implementing, 306–307
 JavaScript, 300–306
 memory complexity, 307–308
 overview, 284
 performance, 307–308
 runtime complexity, 307
 walkthrough, 293–298
 when to use, 298–300
 binary searches, 32, 243, 257
 coding, 250–253
 dividing operations, 247–250
 iteration, 250–251
 JavaScript, 250–253
 middle elements, 244–250
 operations, 244
 recursive operations, 251
 runtime, 254–257
 sorted items, 244
 walkthrough, 252–253
 DFS, 281–283, 308
 coding, 300–306
 implementing, 306–307
 JavaScript, 300–306
 memory complexity, 307–308
 overview, 284
 performance, 307–308
 runtime complexity, 307
 walkthrough, 293–298
 when to use, 298–300
 global linear searches, 240–241
 item searches in arrays, 22–23, 32
 linear searches, 32, 235–236, 241
 coding, 238–239
 global linear searches, 240–241
 implementing, 238–239
 JavaScript, 238–239
 runtime, 239
 walkthrough, 236–238
 queues, 66–67
 search/contains operations, stacks, 59

Quadratic Complexity
O(n^2), 14
 queues, 61, 67–68
 cache performance, 67
 coding, 64–66
 defined, 62–63
 dequeue (deletion), 66
 dynamic memory allocation, 67
 enqueue (insertion), 66
 FIFO, 62
 JavaScript, 64–66
 memory overhead, 67
 peek operations, 66
 performance, 66
 caches, 67
 memory, 67
 runtime, 66–67
 searches, 66–67
 space complexity, 66–67
 time complexity, 66–67
 quicksort
 coding, 319–322
 divide-and-conquer algorithms, 309, 323–324
 implementing, 319–322
 JavaScript, 319–322
 memory, 324, 334, 353, 369, 381
 performance, 322, 324, 334
 space complexity, 323
 speed, 353, 369, 381
 stability, 323
 time complexity, 323
 walkthrough, 310–319
selection sort, 355
  coding, 366–369
  implementing, 366–369
  JavaScript, 366–369
  memory, 324, 334, 353, 369, 382
  performance, 324, 334
  sorted regions, 355–356
  speed, 353, 369, 382
  walkthrough, 356–366
sibling nodes, 73
single child nodes, 99–100
singly linked lists, 42
skip lists, 44
sorted regions, selection sort, 355–356
sorting algorithms
  bubblesort, 325
  coding, 333
  JavaScript, 333–334
  memory, 324, 334, 353, 369, 382
  performance, 324, 334
  speed, 353, 369, 382
  walkthrough, 326–332
heapsort
  memory, 324, 334, 353, 369, 381
  performance, 324, 334
  speed, 353, 369, 381
insertion sort, 335–336
active numbers, 347
coding, 349–351
implementing, 349–351
JavaScript, 349–351
memory, 324, 334, 353, 369, 382
performance, 324, 334
351–352
speed, 353, 369, 382
walkthrough, 336–348
mergesort, 371–372
2n space, 379
coding, 380–381
divide-and-conquer algorithms, 372
JavaScript, 380–381
memory, 324, 334, 353, 369, 381
operations, 379–380
performance, 324, 334
speed, 353, 369, 381
stable sorts, 379–380
tree depth, 371–372
walkthrough, 372–379
quicksort
  coding, 319–322
  divide-and-conquer algorithms, 309, 323–324
  implementing, 319–322
  JavaScript, 319–322
  memory, 324, 334, 353, 369, 381
  performance, 322, 324, 334
  space complexity, 323
  speed, 353, 369, 381
  stability, 323
time complexity, 323
  walkthrough, 310–319
selection sort, 355
  coding, 366–369
  implementing, 366–369
  JavaScript, 366–369
  memory, 324, 334, 353, 369, 382
  performance, 324, 334
  sorted regions, 355–356
  speed, 353, 369, 382
  walkthrough, 356–366
timsort
  memory, 324, 334, 353, 369, 382
  performance, 324, 334
  speed, 353, 369, 382
  space complexity, 323
  time complexity, 323
  walkthrough, 310–319
  space complexity, 58–59
  time complexity, 58–59
  Und奠Redo, 53

T

task boards, 113–114
task organizers, heaps as, 134–135
terminating conditions, recursion, 203–205
time complexity
  linked lists, 41
  queues, 66–67
  quicksort, 323
  stacks, 58–59
timsort
  memory, 324, 334, 353, 369, 382
  performance, 324, 334
  speed, 353, 369, 382
toolbox metaphor, data structures, 4–5
Towers of Hanoi, 221, 234
algorithm, 228–229
  calculating number of moves, 232–234
coding, 229–232
JavaScript, 229–232
playing, 222–223
recursive operations, 231–232
single disk game play, 223
two disk game play, 225–228
two children nodes, 101–103

traversals, binary tree, 259–260
breadth-first traversals, 279
  implementing, 270, 274–276
  node exploration, 260–265
  performance, 278
  walkthrough, 260–265
depth-first traversals, 279
  implementing, 270, 276–278
  node exploration, 272–273
  performance, 278
  stack behaviors, 273
  walkthrough, 265–270

traversing nodes, 37
  breadth-first traversals, 270–271
  depth-first traversals, 272–273
trees, 69–70
  binary search trees, 93, 112
    adding nodes, 93–97, 103
coding, 103–109
defined, 91–92
example of, 91–92
implementing, 103–109
JavaScript, 103–109
leaf nodes, 97–98
memory, 110–111
performance, 110–111
removing nodes, 97–103
single child nodes, 99–100
two children nodes, 101–103

binary trees, 79
  balanced binary trees, 85, 116
coding, 86–89
  complete binary trees, 83–84
defined, 80
degenerate binary trees, 85
element of, 80
full binary trees, 82–83
  implementing, 86–89
JavaScript, 86–89
nodes, 86, 89
perfect binary trees, 84–85
rules, 80–82
variants, 82–85
connected trees, 188–189
  defined, 70–74
depth, mergesort, 379
  edges, 70–71
examples, 70–74
nodes, 70–71
child nodes, 71–73
defined, 91–92
elements of, 91–92
implementing, 103–109
JavaScript, 103–109
leaf nodes, 97–98
memory, 110–111
performance, 110–111
removing nodes, 97–103
single child nodes, 99–100
two children nodes, 101–103

binary trees, 79
  balanced binary trees, 85, 116
coding, 86–89
  complete binary trees, 83–84
defined, 80
degenerate binary trees, 85
element of, 80
full binary trees, 82–83
  implementing, 86–89
JavaScript, 86–89
nodes, 86, 89
perfect binary trees, 84–85
rules, 80–82
variants, 82–85
connected trees, 188–189
  defined, 70–74
depth, mergesort, 379
  edges, 70–71
examples, 70–74
nodes, 70–71
child nodes, 71–73
defined, 91–92
elements of, 91–92
implementing, 103–109
JavaScript, 103–109
leaf nodes, 97–98
memory, 110–111
performance, 110–111
removing nodes, 97–103
single child nodes, 99–100
two children nodes, 101–103

unconnected trees, graphs, 188–189
undirected graphs, 185–186
Undo/Redo, 53
unreachable nodes, 40
upper bound scenarios, Big-O notation, 11

word games/puzzles, 171–172
two children nodes, 101–103

U - V

unconnected trees, graphs, 188–189
undirected graphs, 185–186
Undo/Redo, 53
unreachable nodes, 40
upper bound scenarios, Big-O notation, 11

W - X - Y - Z

word games/puzzles, tries (prefix trees), 171–172
worst-case scenarios, Big-O notation, 11