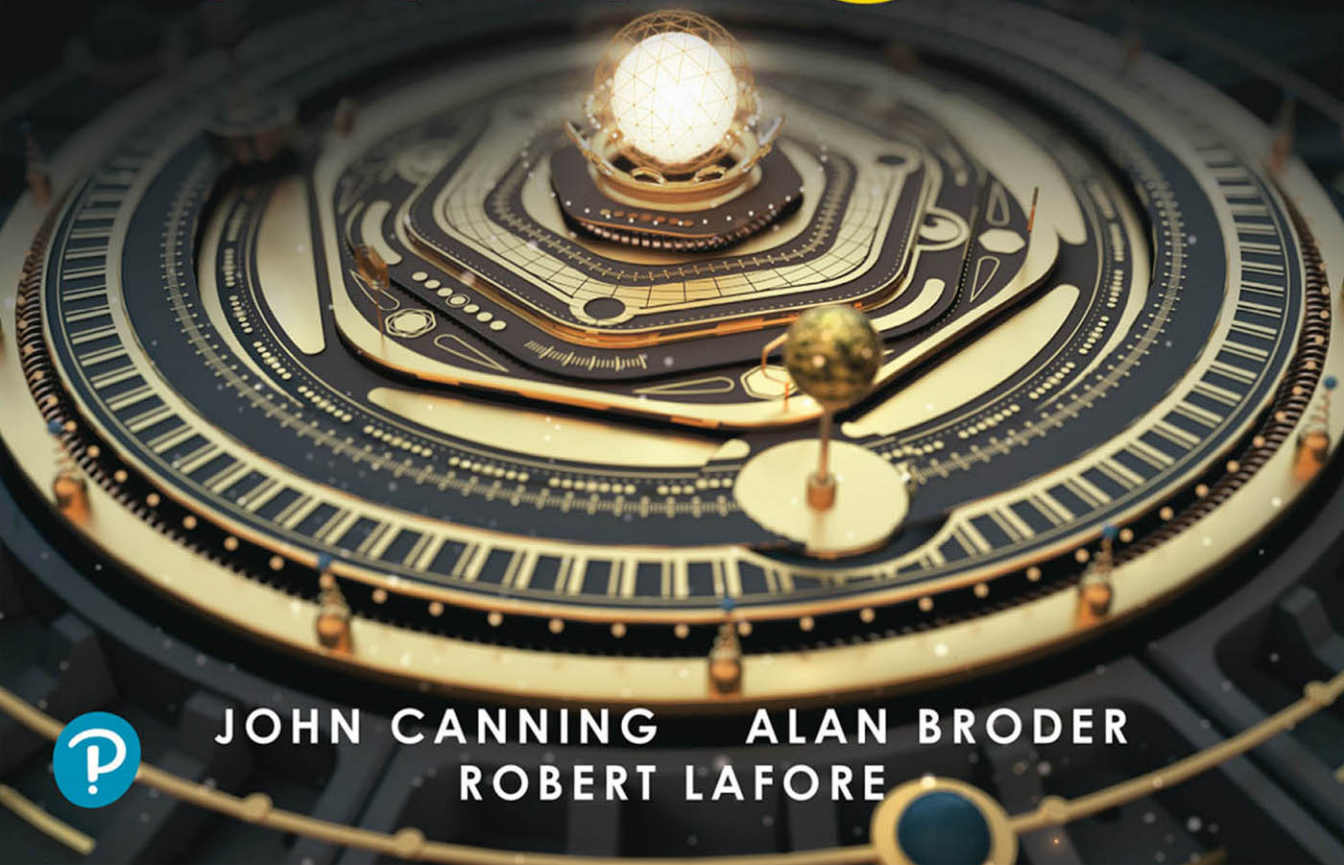




DATA STRUCTURES & ALGORITHMS *in* PYTHON



JOHN CANNING ALAN BRODER
ROBERT LAFORE

John Canning
Alan Broder
Robert Lafore

Data Structures & Algorithms in Python

◆ Addison-Wesley

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

Data Structures & Algorithms in Python

Table of Contents

Cover

Title Page

Copyright Page

Table of Contents

1 Overview

What Are Data Structures and Algorithms?

Overview of Data Structures

Overview of Algorithms

Some Definitions

Database

Record

Field

Key

Databases vs. Data Structures

Programming in Python

Interpreter

Dynamic Typing

Sequences

Looping and Iteration

Multivalued Assignment

Importing Modules

Functions and Subroutines

List Comprehensions

Exceptions

Table of Contents

Object-Oriented Programming

Summary

Questions

Experiments

2 Arrays

The Array Visualization Tool

Searching

The Duplicates Issue

Using Python Lists to Implement the Array Class

Creating an Array

Accessing List Elements

A Better Array Class Implementation

The OrderedArray Visualization Tool

Linear Search

Binary Search

Python Code for an OrderedArray Class

Binary Search with the find() Method

The OrderedArray Class

Advantages of Ordered Arrays

Logarithms

The Equation

The Opposite of Raising 2 to a Power

Storing Objects

The OrderedRecordArray Class

Big O Notation

Insertion in an Unordered Array: Constant

Linear Search: Proportional to N

Binary Search: Proportional to $\log(N)$

Table of Contents

Dont Need the Constant

Why Not Use Arrays for Everything?

Summary

Questions

Experiments

Programming Projects

3 Simple Sorting

How Would You Do It?

Bubble Sort

Bubble Sort on the Football Players

The SimpleSorting Visualization Tool

Python Code for a Bubble Sort

Invariants

Efficiency of the Bubble Sort

Selection Sort

Selection Sort on the Football Players

A Brief Description

A More Detailed Description

The Selection Sort in the SimpleSorting Visualization Tool

Python Code for Selection Sort

Invariant

Efficiency of the Selection Sort

Insertion Sort

Insertion Sort on the Football Players

Partial Sorting

The Marked Player

The Insertion Sort in the SimpleSorting Visualization Tool

Python Code for Insertion Sort

Table of Contents

- Invariants in the Insertion Sort
- Efficiency of the Insertion Sort
- Python Code for Sorting Arrays
- Stability

Comparing the Simple Sorts

Summary

Questions

Experiments

Programming Projects

4 Stacks and Queues

Different Structures for Different Use Cases

- Storage and Retrieval Pattern
- Restricted Access
- More Abstract

Stacks

- The Postal Analogy
- The Stack Visualization Tool
- Python Code for a Stack
- Stack Example 1: Reversing a Word
- Stack Example 2: Delimiter Matching
- Efficiency of Stacks

Queues

- A Shifty Problem
- A Circular Queue
- The Queue Visualization Tool
- Python Code for a Queue
- Efficiency of Queues
- Dequeues

Table of Contents

Priority Queues

- The PriorityQueue Visualization Tool

- Python Code for a Priority Queue

- Efficiency of Priority Queues

- What About Search and Traversal?

Parsing Arithmetic Expressions

- Postfix Notation

- Translating Infix to Postfix

- The InfixCalculator Tool

- Evaluating Postfix Expressions

Summary

Questions

Experiments

Programming Projects

5 Linked Lists

Links

- References and Basic Types

- Relationship, Not Position

The LinkedList Visualization Tool

- The Search Button

- The Delete Button

- The New Button

- The Other Buttons

A Simple Linked List

- The Basic Linked List Methods

- Traversing Linked Lists

- Insertion and Search in Linked Lists

- Deletion in Linked Lists

Table of Contents

Double-Ended Lists

Linked List Efficiency

Abstract Data Types and Objects

A Stack Implemented by a Linked List

A Queue Implemented by a Linked List

Data Types and Abstraction

ADT Lists

ADTs as a Design Tool

Ordered Lists

Python Code for Ordered Lists

Efficiency of Ordered Linked Lists

List Insertion Sort

Doubly Linked Lists

Insertion and Deletion at the Ends

Insertion and Deletion in the Middle

Doubly Linked List as Basis for Deques

Circular Lists

Iterators

Basic Iterator Methods

Other Iterator Methods

Iterators in Python

Summary

Questions

Experiments

Programming Projects

6 Recursion

Triangular Numbers

Finding the nth Term Using a Loop

Table of Contents

Finding the nth Term Using Recursion

Whats Really Happening?

Characteristics of Recursive Routines

Is Recursion Efficient?

Mathematical Induction

Factorials

Anagrams

A Recursive Binary Search

Recursion Replaces the Loop

Divide-and-Conquer Algorithms

The Tower of Hanoi

The TowerofHanoi Visualization Tool

Moving Pyramids

The Recursive Implementation

Sorting with mergesort

Merging Two Sorted Arrays

Sorting by Merging

Merging Subranges

Testing the Code

The Mergesort Visualization Tool

Efficiency of the mergesort

Eliminating Recursion

Recursion and Stacks

Simulating a Recursive Function: Triangular

Rewriting a Recursive Procedure: mergesort

Some Interesting Recursive Applications

Raising a Number to a Power

The Knapsack Problem

Combinations: Picking a Team

Table of Contents

Summary

Questions

Experiments

Programming Projects

7 Advanced Sorting

Shellsort

Insertion Sort: Too Many Copies

N-Sorting

Diminishing Gaps

The AdvancedSorting Visualization Tool

Python Code for the Shellsort

Other Interval Sequences

Efficiency of the Shellsort

Partitioning

The Partition Process

The General Partitioning Algorithm

Efficiency of the Partition Algorithm

Quicksort

The Basic Quicksort Algorithm

Choosing a Pivot Value

A First Quicksort Implementation

Running Quicksort in the AdvancedSorting Visualization Tool

The Details

Degenerates to $O(N^2)$ Performance

Median-of-Three Partitioning

Handling Small Partitions

The Full Quicksort Implementation

Removing Recursion

Efficiency of Quicksort

Table of Contents

Radix Sort

- Algorithm for the Radix Sort
- Designing a Radix Sort Program
- Efficiency of the Radix Sort
- Generalizing the Radix Sort
- Using a Counting Sort

Timsort

- Efficiency of Timsort

Summary

Questions

Experiments

Programming Projects

8 Binary Trees

Why Use Binary Trees?

- Slow Insertion in an Ordered Array
- Slow Searching in a Linked List
- Trees to the Rescue
- What Is a Tree?

Tree Terminology

- Root
- Path
- Parent
- Child
- Sibling
- Leaf
- Subtree
- Visiting
- Traversing

Table of Contents

Levels

Keys

Binary Trees

Binary Search Trees

An Analogy

How Do Binary Search Trees Work?

The Binary Search Tree Visualization Tool

Representing the Tree in Python Code

Finding a Node

Using the Visualization Tool to Find a Node

Python Code for Finding a Node

Tree Efficiency

Inserting a Node

Using the Visualization Tool to Insert a Node

Python Code for Inserting a Node

Traversing the Tree

In-order Traversal

Pre-order and Post-order Traversals

Python Code for Traversing

Traversing with the Visualization Tool

Traversal Order

Finding Minimum and Maximum Key Values

Deleting a Node

Case 1: The Node to Be Deleted Has No Children

Case 2: The Node to Be Deleted Has One Child

Case 3: The Node to Be Deleted Has Two Children

The Efficiency of Binary Search Trees

Trees Represented as Arrays

Table of Contents

Tree Levels and Size

Printing Trees

Duplicate Keys

The BinarySearchTreeTester.py Program

The Huffman Code

Character Codes

Decoding with the Huffman Tree

Creating the Huffman Tree

Coding the Message

Summary

Questions

Experiments

Programming Projects

9 2-3-4 Trees and External Storage

Introduction to 2-3-4 Trees

Whats in a Name?

2-3-4 Tree Terminology

2-3-4 Tree Organization

Searching a 2-3-4 Tree

Insertion

Node Splits

Splitting the Root

Splitting on the Way Down

The Tree234 Visualization Tool

The Random Fill and New Tree Buttons

The Search Button

The Insert Button

Zooming and Scrolling

Table of Contents

Experiments

Python Code for a 2-3-4 Tree

The __Node Class

The Tree234 Class

Traversal

Deletion

Efficiency of 2-3-4 Trees

Speed

Storage Requirements

2-3 Trees

Node Splits

Promoting Splits to Internal Nodes

Implementation

Efficiency of 2-3 Trees

External Storage

Accessing External Data

Sequential Ordering

B-Trees

Indexing

Complex Search Criteria

Sorting External Files

Summary

Questions

Experiments

Programming Projects

10 AVL and Red-Black Trees

Our Approach to the Discussion

Balanced and Unbalanced Trees

Table of Contents

Degenerates to $O(N)$

Measuring Tree Balance

How Much Is Unbalanced?

AVL Trees

The AVLTree Visualization Tool

Inserting Items with the AVLTree Visualization Tool

Python Code for the AVL Tree

The Efficiency of AVL Trees

Red-Black Trees

Conceptual

Top-Down Insertion

Bottom-Up Insertion

Red-Black Tree Characteristics

Using the Red-Black Tree Visualization Tool

Flipping a Nodes Color

Rotating Nodes

The Insert Button

The Search Button

The Delete Button

The Erase & Random Fill Button

Experimenting with the Visualization Tool

Experiment 1: Inserting Two Red Nodes

Experiment 2: Rotations

Experiment 3: Color Swaps

Experiment 4: An Unbalanced Tree

More Experiments

The Red-Black Rules and Balanced Trees

Null Children

Rotations in Red-Black Trees

Table of Contents

Subtrees on the Move

Inserting a New Node

Preview of the Insertion Process

Color Swaps on the Way Down

Rotations After the Node Is Inserted

Rotations on the Way Down

Deletion

The Efficiency of Red-Black Trees

2-3-4 Trees and Red-Black Trees

Transformation from 2-3-4 to Red-Black

Operational Equivalence

Red-Black Tree Implementation

Summary

Questions

Experiments

Programming Projects

11 Hash Tables

Introduction to Hashing

Bank Account Numbers as Keys

A Dictionary

Hashing

Collisions

Open Addressing

Linear Probing

Python Code for Open Addressing Hash Tables

Quadratic Probing

Double Hashing

Separate Chaining

Table of Contents

The HashTableChaining Visualization Tool

Python Code for Separate Chaining

Hash Functions

Quick Computation

Random Keys

Nonrandom Keys

Hashing Strings

Folding

Hashing Efficiency

Open Addressing

Separate Chaining

Open Addressing Versus Separate Chaining

Hashing and External Storage

Table of File Pointers

Nonfull Blocks

Full Blocks

Summary

Questions

Experiments

Programming Projects

12 Spatial Data Structures

Spatial Data

Cartesian Coordinates

Geographic Coordinates

Computing Distances Between Points

Distance Between Cartesian Coordinates

Circles and Bounding Boxes

Clarifying Distances and Circles

Table of Contents

Bounding Boxes

The Bounding Box of a Query Circle in Cartesian Coordinates

The Bounding Box of a Query Circle in Geographic Coordinates

Implementing Bounding Boxes in Python

The CircleBounds Subclass

Determining Whether Two Bounds Objects Intersect

Determining Whether One Bounds Object Lies Entirely Within Another

Searching Spatial Data

Lists of Points

Creating an Instance of the PointList Class

Inserting Points

Finding an Exact Match

Deleting a Point

Traversing the Points

Finding the Nearest Match

Grids

Implementing a Grid in Python

Creating an Instance of the Grid Class

Inserting Points

Finding an Exact Match

Big O and Practical Considerations

Deleting and Traversing

Finding the Nearest Match

Does the Query Circle Fall Within a Layer?

Does the Query Circle Intersect a Grid Cell?

Generating the Sequence of Neighboring Cells to Visit

Pulling It All Together: Implementing Grids findNearest()

Quadtrees

Creating an Instance of the QuadTree Class

Table of Contents

Inserting Points: A Conceptual Overview

Avoiding Ambiguity

The QuadTree Visualization Tool

Implementing Quadtrees: The Node Class

The insert Method

Efficiency of Insertion

Finding an Exact Match

Efficiency of Exact Search

Traversing the Points

Deleting a Point

Finding the Nearest Match

Finding a Candidate Node

Finding the Closest Node

Pulling It All Together: Implementing QuadTrees findNearest()

Efficiency of findNearest()

Theoretical Performance and Optimizations

Practical Considerations

Further Extensions

Other Operations

Higher Dimensions

Summary

Questions

Experiments

Programming Projects

13 Heaps

Introduction to Heaps

Priority Queues, Heaps, and ADTs

Partially Ordered

Table of Contents

Insertion

Removal

Other Operations

The Heap Visualization Tool

The Insert Button

The Make Random Heap Button

The Erase and Random Fill Button

The Peek Button

The Remove Max Button

The Heapify Button

The Traverse Button

Python Code for Heaps

Insertion

Removal

Traversal

Efficiency of Heap Operations

A Tree-Based Heap

Heapsort

Sifting Down Instead of Up

Using the Same Array

The heapsort() Subroutine

The Efficiency of Heapsort

Order Statistics

Partial Ordering Assists in Finding the Extreme Values

The Efficiency of K Highest

Summary

Questions

Experiments

Programming Projects

Table of Contents

14 Graphs

Introduction to Graphs

Definitions

The First Uses of Graphs

Representing a Graph in a Program

Adding Vertices and Edges to a Graph

The Graph Class

Traversal and Search

Depth-First

Breadth-First

Minimum Spanning Trees

Minimum Spanning Trees in the Graph Visualization Tool

Trees Within a Graph

Python Code for the Minimum Spanning Tree

Topological Sorting

Dependency Relationships

Directed Graphs

Sorting Directed Graphs

The Graph Visualization Tool

The Topological Sorting Algorithm

Cycles and Trees

Python Code for the Basic Topological Sort

Improving the Topological Sort

Connectivity in Directed Graphs

The Connectivity Matrix

Transitive Closure and Warshalls Algorithm

Implementation of Warshalls Algorithm

Summary

Questions

Table of Contents

Experiments

Programming Projects

15 Weighted Graphs

Minimum Spanning Tree with Weighted Graphs

An Example: Networking in the Jungle

The WeightedGraph Visualization Tool

Building the Minimum Spanning Tree: Send Out the Surveyors

Creating the Algorithm

The Shortest-Path Problem

Travel by Rail

Dijkstras Algorithm

Agents and Train Rides

Finding Shortest Paths Using the Visualization Tool

Implementing the Algorithm

Python Code

The All-Pairs Shortest-Path Problem

Efficiency

Intractable Problems

The Knights Tour

The Traveling Salesperson Problem

Hamiltonian Paths and Cycles

Summary

Questions

Experiments

Programming Projects

16 What to Use and Why

Analyzing the Problem

What Kind of Data?

Table of Contents

How Much Data?

What Operations and How Frequent?

Who Will Maintain the Software?

Foundational Data Structures

Speed and Algorithms

Libraries

Arrays

Linked Lists

Binary Search Trees

Balanced Search Trees

Hash Tables

Comparing the General-Purpose Storage Structures

Special-Ordering Data Structures

Stack

Queue

Priority Queue

Comparison of Special-Ordering Structures

Sorting

Specialty Data Structures

Quadtrees and Grids

Graphs

External Storage

Sequential Storage

Indexed Files

B-trees

Hashing

Choosing Among External Storage Types

Virtual Memory

Onward

Table of Contents

Appendixes

A Running the Visualizations

For Developers: Running and Changing the Visualizations

Getting Python

Getting Git

Getting the Visualizations

For Managers: Downloading and Running the Visualizations

For Others: Viewing the Visualizations on the Internet

Using the Visualizations

B Further Reading

Data Structures and Algorithms

Object-Oriented Programming Languages

Object-Oriented Design (OOD) and Software Engineering

C Answers to Questions

Chapter 1, "Overview"

Chapter 2, "Arrays"

Chapter 3, "Simple Sorting"

Chapter 4, "Stacks and Queues"

Chapter 5, "Linked Lists"

Chapter 6, "Recursion"

Chapter 7, "Advanced Sorting"

Chapter 8, "Binary Trees"

Chapter 9, "2-3-4 Trees and External Storage"

Chapter 10, "AVL and Red-Black Trees"

Chapter 11, "Hash Tables"

Chapter 12, "Spatial Data Structures"

Chapter 13, "Heaps"

Chapter 14, "Graphs"

Chapter 15, "Weighted Graphs"

Index

Table of Contents