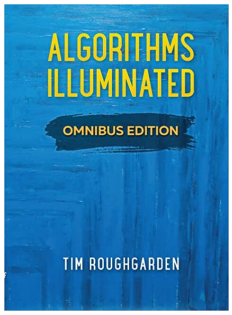
Review of[1]



Algorithms Illuminated, Omnibus Edition
by Tim Roughgarden
SoundLikeYourself Publishing, 2022
(distributed by Cambridge University Press)
690 pages, Hardcover, $60

Review by
Nicholas Tran (ntran@scu.edu)
Department of Mathematics & Computer Science
Santa Clara University

1 Overview

Algorithms Illuminated is the text version of a sequence of four Coursera algorithms courses taught by the author in the last ten years to a large-scale and diverse audience. This book covers topics in data structures (hash tables, Bloom filters, binary heaps, binary search trees, union-find, graphs), analysis techniques (asymptotic notations, the master method, expected values), design paradigms (divide-conquer, greedy, dynamic programming), and coping with hardness (NP-completeness, approximate algorithms, local search, integer programming and satisfiability solvers). Two appendices on proof techniques and probability, a short field guide to algorithm design, and an in-depth case study of the 2016-17 Federal Communications Commission's reverse auction of wireless spectrum are also included. The materials have been used to teach an undergraduate-level as well as a master's level course in algorithms at Stanford and other universities. Extensive learning resources (YouTube videos, slides, math supplements, test data sets, and discussion forums) are available at the eponymous web site.

Keeping with its single goal "*to teach the basics of algorithms in the most accessible way possible,*" the book maintains a conversational style and interacts effectively with the reader through the use of quizzes and footnotes sprinkled throughout the text where elaboration may be required or desired. The quizzes, whose answers with explanation appear a few pages later, force the reader to master a point needed in the subsequent discussion, while the footnotes provide optional additional insights and historical notes. Algorithms are described in very high-level pseudocode that omit implementational details to focus on the big computational picture. The standard treatment of covered topics at many places has been updated with modern problems, tweaked solutions, admonitions against common pitfalls, and references to latest developments. Each chapter ends with a summary, followed by a small number of problems classified into three types: routine, challenge, and programming. Solutions or hints are provided for all non-programming problems.

2 Summary of Contents

Part I: The Basics

Chapter 1 motivates the study of design and analysis of algorithms with a gentle development of Karatsuba multiplication and merge sort. The conventions of the field are presented: how to describe a problem, how to express algorithms in pseudocode, how to count the number of primitive operations, what theorems, lemmas, Q.E.D., and "fast" mean, and why focusing on analyzing the worst-case and long-term behavior of algorithms yields the right balance between mathematical tractability and accurate prediction of running times.

Chapter 2 explains asymptotic notations in English and pictorially with familiar examples and then formally defines them mathematically with additional examples. Common pitfalls are mentioned, such as using O to mean Θ and not realizing that 2^n and 4^n have different growth rates.

Chapter 3 illustrates the divide-and-conquer design paradigm with three classic problems: counting inversions in an array, matrix multiplication, and finding the closest pair of points in the plane. Exhaustive solutions to these problems are presented first, followed by improvements based on the divide-conquer-combine paradigm. Asymptotic analysis of these recursive algorithms is given informally, to be made precise using the master theorem in the next chapter. The closest-pair algorithm given in this chapter is a nice variation of the standard treatment that presorts the array of points twice (by x-coordinates and independently by y-coordinates) to achieve a clean conquer step.

Chapter 4 states the master theorem and applies it to recursive algorithms discussed in the previous chapters among others. A carefully explained proof of the theorem is also provided.

Chapter 5 is an in-depth study of quicksort implemented using Lomuto's version of the partition algorithm. Randomized algorithms are introduced, and analysis of the expected running time of randomized quicksort using indicator random variables is given. The chapter ends with a proof of the lower bound of $\Omega(n \log n)$ for comparison-based sorting algorithms and a discussion of faster sorting algorithms such as bucket sort, counting sort, and radix sort that are not comparison-based.

Chapter 6 describes the straightforward application of the partition algorithm to solve the selection problem (commonly known as quickselect) and shows that the expected running time of randomized quickselect is $O(n)$. The chapter ends with a detailed description and analysis of the $O(n)$ median-of-medians algorithm for selection.

Part II: Graph Algorithms and Data Structures

Chapter 7 defines graphs and graph terminology, provides examples of graph applications, and compares the adjacency matrix and adjacency list representations of graphs.

Chapter 8 proves the correctness of the generic graph search algorithm and then implements and analyzes breadth-first search and depth-first search as special cases. Finding least-edge paths and connected components are discussed as applications of breadth-first search; similarly, finding topological ordering and strongly connected components with Kosaraju's algorithm are discussed as applications of depth-first search. The chapter ends with a discussion of structural properties of strongly connected components of the Web graph, whose vertices are web pages and whose edges are hyperlinks.

Chapter 9 defines the single-source shortest path problem and gives some examples of its applications. Dijkstra's algorithm is then presented in data structure-agnostic pseudocode, shown

to run in polynomial time, and proven correct. The need for a correctness proof is motivated by two observations that i) Dijkstra's algorithm fails on some graphs with negative weights; and ii) adding a constant positive offset to all edge weights changes the shortest paths.

Chapter 10 introduces the heap data structure, its supported operations and their running times, and its applications. A heap-based version of Dijkstra's algorithm is then presented and analyzed. The chapter ends with an array-based implementation of heaps.

Chapter 11 defines the search tree data structure, its supported operations and their running times for the balanced variant. Implementation of basic binary search trees is discussed, with a hint on how to correct imbalance using rotations.

Chapter 12 defines the hash table data structure, its supported operations and their typical running times, and its applications. Implementations of hash tables using chaining and open addressing with linear probing and double hashing are explained, and their performances are analyzed in terms of the load factor. There is a nice discussion of pathological data sets causing poor performance that exist for any hash function, and a brief mention of universal hashing functions as basic good choices for everyday hashing. The chapter also defines Bloom filters, their supported operations and applications. It ends with discussions of their implementation as well as heuristic analysis of their performance.

Part III: Greedy Algorithms and Dynamic Programming

Chapter 13 introduces the greedy algorithm design paradigm and illustrates it with a scheduling problem that seeks to minimize the sum of weighted completion times. Two appealing greedy strategies that yield conflicting outcomes are compared to motivate the need for a correctness proof, which is based on an exchange argument.

Chapter 14 motivates the problem of finding optimal prefix-free code and rephrases it as finding a Σ-tree with the minimum average leaf depth. Huffman's algorithm is then presented in pseudocode and analyzed, followed by a discussion of speeding it up using either a heap or two queues. The chapter ends with a careful induction proof of correctness for Huffman's algorithm.

Chapter 15 defines the minimum spanning tree problem and presents and analyzes Prim's algorithm, first in high-level pseudocode, and then enhanced with the use of heaps. A proof of correctness based on the minimum bottleneck property is provided; the more traditional proof based on the cut property is covered in the exercises. Next, it presents and analyzes Kruskal's algorithm, first in high-level pseudocode, and then enhanced with the use of the union-find data structure. A tree-based implementation of union-find that uses union by rank but not path compression is sketched. A proof of correctness of Kruskal's algorithm based on the minimum bottleneck property is provided. The chapter ends with a discussion of the clustering problem and how to solve it using Kruskal's algorithm.

Chapter 16 gives a fresh introduction to the dynamic programming design paradigm and illustrates it with two problems: weighted independent set and knapsack. After showing that the problems are not susceptible to greedy strategies, it enumerates a three-step recipe for designing dynamic programming algorithms and points out the differences between this and the divide-and-conquer recipe. It also includes an amusing anecdote on the origin of the somewhat confusing paradigm name.

Chapter 17 reinforces the three-step recipe for designing dynamic programming algorithms with two classic problems: sequence alignment and construction of binary search trees with minimum average search time. The characteristic activities of finding a recurrence for the dependence

on optimal solutions to subproblems, caching, and reconstructing an actual solution that achieves the optimal objective function value are illustrated in detail.

Chapter 18 presents Bellman-Ford and Floyd-Warshall algorithms, which are dynamic programming solutions to single-source and all-pairs shortest path problems respectively. Again, the dependence on optimal solutions to subproblems is elaborated with examples. Notable discussions in this chapter include two ways to define shortest paths in the presence of negative-weight cycles and application of Bellman-Ford algorithm to internet routing.

Part IV: Algorithms for NP-hard Problems

Chapter 19 brings forth the existence of NP-hard problems such as traveling salesman with the following informally defined property: they have no known fast solutions, and if indeed no such solutions exist, then verifying a solution to a problem is fundamentally easier than finding one from scratch. Three strategies are suggested for coping with NP-hard problems: compromising on generality, correctness, or speed of their solutions. Several useful asides briefly address randomized and quantum solutions to NP-hard problems, the exponential time hypothesis, and common rookie mistakes.

Chapter 20 illustrates examples of approximately correct solutions to three NP-hard problems: schedule makespan minimization, maximum coverage, and influence maximization. These are fast greedy algorithms that produce solutions guaranteed to be within a constant factor of the optimal solutions. For traveling salesman, even existence of such approximately correct solutions would imply P = NP; high-quality heuristics based on local search are discussed instead.

Chapter 21 presents dynamic-programming solutions that perform better than the brute-force solutions to two NP-hard problems: traveling salesman and minimum-cost k-path. More generally it is shown how to use state-of-the-art solvers for mixed integer programming and satisfiability to obtain optimal solutions for medium-size instances of a variety of NP-hard problems.

Chapter 22 presents examples of NP-hard proofs. Starting with the assumption that 3-SAT is NP-hard, it is shown that independent set, hamiltonian path, traveling salesman, subset sum are NP-hard as well through a series of reductions.

Chapter 23 gives formal definitions of NP, NP-hard, and NP-complete problems. It also discusses the (strong) exponential time hypothesis and the unexpected relationship between it and sequence alignment.

Chapter 24 is an in-depth case study of the reallocation of spectrum previously licensed to television broadcasters to wireless broadband providers. Between 2016 and 2017 the FCC implemented a reverse auction to determine which television stations would release their spectrum licenses and at what price, an NP-hard problem even in its simplified form. Drawing together concepts and results discussed previously, this chapter walks through the details of a greedy heuristic solution which itself depends on multiple state-of-the-art satisfiability solvers working together plus various optimizing techniques. It ends with a discussion of the auction's technical and economic outcomes.

3 My opinion

This book sets high and specific goals for itself in the preface and to a high degree delivers on its promises. To make the materials accessible, it keeps a laser focus on algorithmic ideas in the text and renders analysis and implementation in broad strokes. The discussions and the footnotes in the book do feel like chats you would have with an expert colleague on their favorite topics. The book

does a great job in presenting greatest hits of algorithms, from the time-tested such as linear-time selection, randomized algorithms and their analysis, fast solutions for minimum spanning trees and shortest paths, to new ones such as influence maximization and fine-grained complexity. Finally, the chapter on the FCC spectrum reverse auction is a unique and convincing demonstration of bringing the algorithmic tools discussed in this book to bear on a real-world difficult problem.

I particularly enjoy the book's positive attitude about coping with NP-hardness and discussion on the different strategies including using integer programming and satisfiability solvers. Overall, I think this book is a modern, accessible but not lightweight textbook on algorithms that will be greatly enjoyed by a wide range of undergraduate and beginning graduate students and practitioners who want to learn how to design algorithmic solutions to problems in their areas.