Review of[1]
How to Think about Algorithms
by Jeff Edmonds
Cambridge University Press, 2008
xiv+450 pages, ISBN: 9780521849319 (Hardback, £55.00, $99.00),
9780521614108 (Paperback, £19.99, $36.99)

Review by
Kyriakos N. Sgarbas (sgarbas@upatras.gr)
Electrical & Computer Engineering Department, University of Patras, Greece

1 Overview

This is a book on algorithms. Or rather a book on applied abstract thinking about algorithms. The algorithms are grouped into three broad categories (iterative, recursive, and algorithms for optimization problems) and each category is presented in a separate part of the book. The book does not focus into the formal aspects of each algorithm. Instead, it aims to explain how one thinks in order to devise such an algorithm, why the algorithm works, and how it relates to other algorithms of the same category. It uses a meta-algorithmic approach to reveal that all algorithms of the same category work more or less the same way. Of course, pseudocode, asymptotic notations, proofs of correctness, are all there, but they are used to explain rather than formally define each algorithm. The language used is simple and approachable and the author prefers to use words to explain how things work instead of overloading the text with mathematical formulas. The layout of the book is quite original (for an algorithm book); in some places it even contains pictures (obviously of the author's family) remotely relevant to some notion mentioned in the page, but giving a nice artistic touch nonetheless.

2 Summary of Contents

Just after the Table of Contents the book starts with a 2.5-page Preface where its goals are explained and a 2-page Introduction with some non-formal definitions concerning computational problems,

[1]©2009, KYRIAKOS N. SGARBAS

algorithms, abstract data types, correctness, running time and meta-algorithms. After that it is organized in 5 parts and 28 chapters (each with its own set of exercises), as follows:

Part I "Iterative Algorithms and Loop Invariants" discusses problems that can be solved using iterative algorithms and analyzes such algorithms using loop invariants (i.e. assertions that must hold true every time the computation returns to the top of the loop). Part I consists of Chapters 1 to 7:

Chapter 1 *"Iterative Algorithms: Measures of Progress and Loop Invariants"* (24 pages) on a first reading it seems like a set of advices on how to write algorithms that work. But seen as a set these advices propose a different way of abstract thinking, shifting from the one-instruction-per-step model to the one-instance-of-the-solution-per-step model, by defining appropriate loop invariants and measures of progress and applying them promptly. Examples used: insertion/selection/bubble sort and binary search.

Chapter 2 *"Examples Using More-of-the-Input Loop Invariants"* (14 pages) presents some more complex problems solved using more-of-the-input loop invariants (i.e. loop invariants where the measure of progress is related to the amount of the input considered) and a comparison with problems more easily solved with more-of-the-output invariants (i.e. loop invariants where the measure of progress is related to the amount of the output constructed). Examples used: Plane coloring, parsing with DFA, arithmetic operations, Euler cycle, etc.

Chapter 3 *"Abstract Data Types"* (17 pages) briefly introduces lists, stacks, queues, priority queues, sets, graphs, and trees, their implementation and basic operations on them.

Chapter 4 *"Narrowing the Search Space: Binary Search"* (11 pages) as its title suggests, presents loop invariants using the principle of narrowing the search space of a problem after ensuring that the target element always remains in the reduced search space. It uses two very nice examples: Magic Sevens (a card trick) and VLSI Chip Testing (not really a hardware problem, just a problem of element comparison). These examples are intentionally flawed in their definition, so the analysis does not stop to their solution but extends to some variants as well.

Chapter 5 *"Iterative Sorting Algorithms"* (8 pages) discusses bucket sort, counting sort, radix sort and radix counting sort and explains how they relate to each other.

Chapter 6 *"Euclid's GCD Algorithm"* (6 pages) is a short chapter on the well-known algorithm presented separately obviously due to its 'strange' loop invariant. Two other problems using similar-type invariants are mentioned in the chapter exercises.

Chapter 7 *"The Loop Invariant for Lower Bounds"* (10 pages) discusses how to prove the lower bound of an iterative algorithm. The author points out that the process of calculation of a lower bound can be seen as an algorithm itself, therefore the same model based on loop invariants can be used to evaluate it. Examples used: sorting, binary search, parity, and multiplexer.

Part II "Recursion" discusses problems that can be solved using recursive algorithms and explains some methods for devising them. Part II consists of Chapters 8-12:

Chapter 8 *"Abstractions, Techniques, and Theory"* (17 pages) provides the basic thinking tools for confronting problems by recursion. Stack frames, strong induction and a framework for a general-purpose blueprint for recursive algorithms are discussed. The problem of the Towers of Hanoi is used as example and analyzed thoroughly.

Chapter 9 *"Some Simple Examples of Recursive Algorithms"* (16 pages) provides some more examples of recursive algorithms explained with the model of Chapter 8. Examples used: merge sort, quick sort, calculation of integer powers, matrix multiplication, etc.

Chapter 10 *"Recursion on Trees"* (23 pages) presents and analyzes recursive algorithms for operations on trees: counting the number of nodes in binary trees, perform tree traversals, return the maximum of data fields of tree-nodes, calculate the height of a tree, count the number of its leaves, making a copy of the whole tree. A thorough analysis of the heap sort algorithm continues and the chapter concludes with some algorithms for tree-representations of algebraic expressions: expression evaluation, symbolic differentiation, expression simplification.

Chapter 11 *"Recursive Images"* (6 pages) is a small chapter explaining how to use recursive algorithms to produce fractal-like images and random mazes.

Chapter 12 *"Parsing with Context-Free Grammars"* (9 pages): Context-Free Grammars (CFGs) are used in compiler development and computational linguistics to express the syntactic rules of an artificial or natural language. This chapter discusses how to develop algorithms for parsing expressions (strings) according to a set of CFG-rules and produce structured representations of the input expressions.

Part III "Optimization Problems" presents methodologies for building algorithms (iterative and recursive) to confront several classes of optimization problems. Part III consists of Chapters 13-21:

Chapter 13 *"Definition of Optimization Problems"* (2 pages) presents the definition of what an optimization problem is and gives short examples (only the definition) of different degrees of complexity: longest common subsequence, course scheduling and airplane construction.

Chapter 14 *"Graph Search Algorithms"* (25 pages) presents and analyzes several algorithms on graphs: a generic search algorithm, a breadth-first search algorithm for shortest paths, Dijkstra's shortest-weighted-path algorithm, iterative and recursive depth-first search algorithms, and topological sorting.

Chapter 15 *"Network Flows and Linear Programming"* (27 pages) starts with the definition of the Network Flow and Min Cut optimization problems and presents gradual solutions introducing several variations of hill-climbing algorithms (simple, primal-dual, steepest-ascent). The chapter concludes with a discussion on linear programming.

Chapter 16 *"Greedy Algorithms"* (26 pages) uses the Making Change problem (i.e. finding the minimum number of coins that sum to a certain amount) to explain the basics of greedy algorithms and then applies them in some more examples on job/event scheduling, interval covering, and producing the minimum-spanning-tree of a graph.

Chapter 17 *"Recursive Backtracking"* (16 pages) explores the idea of finding an optimal solution for one instance of the problem using a recurrence relation that combines optimal solutions for some smaller instances of the same problem that are computed recursively. Examples used: searching a maze, searching a classification tree, n-Queens problem, SAT problem (Davis-Putnam algorithm).

Chapter 18 *"Dynamic Programming Algorithms"* (28 pages): Dynamic Programming resembles Recursive Backtracking except that the optimal solutions for the smaller instances of the problem are not computed recursively but iteratively and are stored in a table. The chapter uses the problem of finding the shortest weighted path within a directed leveled graph to explain the basics on developing a dynamic programming algorithm and then elaborates in several optimization aspects.

Chapter 19 *"Examples of Dynamic Programs"* (29 pages) provides some additional examples of problems solved using dynamic programming algorithms, i.e. the longest-common-subsequence problem (with several variations), the weighted job/event scheduling problem, matrix multiplication, CFG-parsing, etc.

Chapter 20 *"Reductions and NP-Completeness"* (22 pages) begins with the definition of problem reduction and then uses it to classify problems according to this relation. It discusses NP-completeness and satisfiability and explains with great detail how to prove that a problem is NP-complete. Examples used: 3-coloring, bipartite matching.

Chapter 21 *"Randomized Algorithms"* (8 pages) points out that sometimes it is a good idea to use randomness (e.g. to avoid a worst case) and presents two models for analysis of randomized algorithms: Las Vegas (guaranteed to give the correct answer but the running time is random), and Monte Carlo (guaranteed to stop within a certain time limit but may not give the right answer). These models, along with the classic (deterministic) worst case analysis are used in some comparative examples.

Part IV "Appendix": These are not simple appendices, they are normal chapters just like all the previous ones. Some of them have exercises too! Part IV consists of Chapters 22-28:

Chapter 22 *"Existential and Universal Quantifiers"* (9 pages) provides the basic definitions for relations, quantifiers, free and bound variables and discusses some proof strategies involving universal and existential quantifiers.

Chapter 23 *"Time Complexity"* (8 pages) provides definitions of time and space complexity of algorithms and examples on their estimation.

Chapter 24 *"Logarithms and Exponentials"* (3 pages) summarizes the definitions, the properties and some frequent uses of logarithms and exponentials.

Chapter 25 *"Asymptotic Growth"* (11 pages): Formal definitions, estimations and examples of Big Oh, Omega, Theta, etc.

Chapter 26 *"Adding-Made-Easy Approximations"* (10 pages) contains formulas and methods to calculate or approximate sums of series.

Chapter 27 *"Recurrence Relations"* (10 pages) contains formulas and methods for solving recurrence relations.

Chapter 28 *"A Formal Proof of Correctness"* (2 pages) sketches the required steps in order to formally proof the correctness of an algorithm. It does not include any examples but one can compare the process with some of the formal proofs of correctness provided in previous chapters.

Part V "Exercise Solutions" (25 pages) contains solutions of many of the exercises.

The book concludes with a few lines Conclusion and a 10-page Index. It does not have any list of references.

3 Opinion

Since page one, the reader can realize that this is not an ordinary book about algorithms. Its layout, its structure, and its language, all indicate that it is something uncommon, probably unique. Reading it is an experience much different that what one gets from reading a common algorithm textbook. Instead of presenting and analyzing algorithms formally down to every single mathematical detail, instead of stressing the left part of your brain with lengthy proofs and dive deep into algorithm science, this book resembles more than a practical guide on algorithm craftsmanship.

Approachable and informal, although it uses pseudocode to express algorithms and meta-algorithms, it does not formally define the syntax of the pseudo language used.

Its structure is uncommon as well. There are chapters with 2 pages each and chapters with more than 20 pages each. Most of them have exercises, usually in the last section of the chapter, but some have exercises also inside the sections (in appendices too).

Furthermore you will see mathematical formulas with no explanation of what the variables stand for (e.g. in the beginning of Chapter 7). You either already know from previous experience, or you may try to deduce from the context, or -finally- you might think to skip into an appropriate appendix (sometimes the text prompts you to do so, sometimes not).

Actually, the text has many self references to other chapters but not any external references. Regardless whether it presents some well-known algorithm (like Dijkstra's algorithm), or provides additional information (e.g. in Chapter 20 where it mentions that the relation of any optimization problem to CIR-SAT was proved by Steve Cook in 1971), or discussing the current state of the art (like in Chapter 7 on proving lower bounds) there is not a single external reference, neither in any footnote nor in any list of collected bibliography at the end of the book. Strange thing indeed.

The careful reader will also observe a few typos, but they are not serious enough to hinder the reading or alter the semantics of the text.

Reading the text sometimes feels like reading the slides of a class lecture, along with the actual words of what the teacher said, with all the redundant information, the not-so-relevant comments and the jokes the speaker usually adds to make his speech more interesting (e.g. you'll be surprised on the ingredients of hot dogs when you read the linear programming section). Sometimes it uses metaphors to explain something, and keeps the metaphor along several chapters, thus resulting into a unique pictorial terminology framework (e.g. friends, little birds, and magic denote algorithmic components).

Of course, whether you like reading about algorithms under these terms or not, I believe is a matter of personal taste. Personally I loved reading this book, but I suppose one might just as easily hate it for exactly the same reasons. But surprisingly enough, this model works well when explaining difficult subjects. It tends to communicate effectively the whole picture without much details, while the examples fill in the gaps. For instance, Chapter 20 on Reductions and NP-completeness is probably the most well-explained introduction text I have read on the subject.

In conclusion, I believe this book could be considered a must-read for every teacher of algorithms. Even if he reads things he already knows, he will be able to view them from different angles and in the process get some very useful ideas on how to explain algorithms in class. The book would also be invaluable to researchers who wish to gain a deeper understanding on how algorithms work, and to undergraduate students who wish to develop their algorithmic thought. However, I would not recommend someone to try to learn algorithms starting from this book alone. I believe one has to know a great deal on the subject already, in order to fully appreciate the book and benefit from it. For this is not really an algorithm textbook; it's more like the right-brain counterpart of an ordinary algorithm textbook. And as such, it has the potential to be considered a classic.