Preface

Analysis of algorithms plays an essential role in the education and training of any serious programmer preparing to deal with real world applications. It provides the tools for understanding which algorithm might yield better performance for solving a particular aspect of an application, or adapt best to the problem specifications. The goals of this field are: to provide a solid basis for the comparison of two or more algorithms available for the same problem, to give programmers an idea of how efficient a program is, and to allow them to estimate how well an algorithm will scale up with the data. Even if the students do not directly apply the knowledge acquired in such a class to the problems they need to solve for their professional work, they acquire structured thinking and deep understanding of the mechanism of an algorithm, which are always necessary to write good code. For this reason, algorithm analysis is typically included in the core curriculum of most bachelor's degrees in computer science or informatics.

The subject is a non-trivial one, however, and many students find it difficult to grasp. Part of the problem is the substantial mathematical background that is needed to understand the material. In order to provide a solid analysis of an algorithm, one must use rigorous proofs and other tools that can be challenging even for someone with good training. This is an aspect that makes a course like Analysis of Algorithms particularly demanding of both the students and the teacher.

This textbook addresses the needs of collegiate computing programs without assuming a strong mathematical background. Thus, our purpose is to provide proofs and explanations where all the steps are carefully detailed. We hope that they can be valuable assets for the student with the patience to read them and the will to understand them. For each of the topics discussed, many practical and detailed examples are provided, as well as similar exercises to allow the readers to test their comprehension.

Approach

We approach the field of analysis of algorithms on two fronts that are connected but can be studied separately. The first front concerns the fundamental notations in the big-Oh family, their definition, and how to prove that they apply to various functions. Chapters 2 and 3 of the book focus on this aspect. Chapter 2 provides the

tools for it, and Chap. 3 applies these tools to proofs and exercises related to these notations.

The second front concerns the study of algorithms to find the time and space complexities, and then determining the order of complexity in the best, worst, and average case. The next four chapters are focused on this aspect. We start with a chapter dedicated to recurrence relations, preparing the readers with a major tool for the remaining chapters. Then we follow with deterministic algorithms and study the best and worst cases for them. The following chapter introduces the role of probability theory in this field, and how it is used for finding the average complexity of some algorithms. The related topic of probabilistic algorithms is also covered in this chapter. The last chapter follows with graph theory, where various techniques seen before are applied.

We aimed for this book to be sufficient for an analysis of algorithms course without extensive references to other textbooks. It covers all the tools directly applied to analysis of algorithm in detail. It provides methods and examples to understand these tools and their use, such as the summation notation or recurrence relations. We dedicate a whole chapter to the latter and provide recipes to solve the major types encountered in analysis of algorithms. While most books on the topic discuss them, extensive coverage of recurrence relations can mostly found in discrete mathematics and related textbooks. Our goal is to be able to understand them fully for use within algorithms analysis and be able to derive complete solutions systematically. An appendix on probability theory reviews the major definitions and theorems that we use in the chapter Algorithms and Probabilities to make reading this chapter more seamless.

An aspect that we also try to attend to is the gap between theory and practice, a difficulty that many students encounter in courses of this nature. They understand the theory relatively well, but they are mystified when it comes to applying it to practical examples. Most textbooks on the subject are written at a level of abstraction that makes them difficult for the average computer science student to understand completely and to put them into practice. Many of the presented proofs are sketchy and leave the details to the self-study of the reader. Methods for solving related problems are introduced with few practical examples of how they are applied. These are mostly useful reference books for the class instructors, or for the computer science professionals who might need a quick access to an algorithm outline.

To address this, our book contains many fully worked examples and provides many of the proofs with enough details to be followed step by step. The chapter Fundamental Notations shows many examples of practically proving that a function is in one of the big-Oh relationship with another. For the same purpose, the previous chapter, Mathematical Preliminaries, covers the major functions that appear in proofs of the fundamental notations, such as the logarithms. We review properties of these functions that play a role in connection with the fundamental notations and show how they are used through many examples and exercises. In the chapter Recurrence Relations we compute many solutions down to the value of the coefficients, and the same is done afterwards with recurrence relations emerging

from analyzing specific algorithms. The graph theory algorithms are given with justification and with examples of execution on practical problems.

We present the material in order of difficulty, starting from the chapter that the students are probably the most familiar with. Each new chapter builds on the knowledge acquired in the previous ones and raises the bar higher with a new topic.

Book Outline

The book teaches the concepts of analysis of algorithms required in core courses in many undergraduate and graduate computer science degrees, as well as provides a review of the fundamental mathematical notions that are necessary to understand these concepts. For all of the chapters, the explanations are aimed to the level of understanding of the common upper level student, and not only the teacher. All the materials are accompanied by many examples explained in much detail and by exercises.

An introduction explains the purpose of studying this field and the major ideas behind the notion of complexity of algorithms. We make the connection between the execution time and the count of operations, and continue towards the more abstract notion of order of complexity. This leads the way to understanding the need for the fundamental notations, such as the big-Oh.

The second chapter, Mathematical Preliminaries, reviews the fundamental mathematical notions that the students need to understand analysis of algorithms. Even though these concepts are most likely taught in mathematical classes, the knowledge level may be non-uniform and this chapter serves the purpose of bringing them to a common ground. It also prepares the reader to understand the proofs and exercises in the Fundamental Notations chapter. We cover several functions that play a role in analysis of algorithms, as well as the summation operation, which is essential to loop counting.

The third chapter, Fundamental Notations, lays the foundation of the analysis of algorithms theory in terms of the big-Oh, Omega, and Theta notations. It also covers the related topics of the little-Oh and asymptotic functions. We use the properties of the functions introduced in the previous chapter to prove the relationships defined by the fundamental notations for a variety of expressions.

The fourth chapter, Recurrence Relations, introduces a very important tool used in analysis of algorithms. We go in further detail than other algorithms textbooks over the definition, properties, and procedures for solving various types of recurrence relations. We cover the major types of recurrence relations that occur often from analyzing algorithms and provide systematic procedures for solving them.

The next chapter introduces the students to the core of the analysis of algorithms theory, explaining the concept of basic operation, followed by traditional loop counting, and the ideas of best case and worst case complexities. We then provide a detailed analysis for the classes of algorithms that are most commonly seen in a data structures textbook or course. This chapter also provides a section on the theorem

setting the lower bound for the worst-case complexity of comparison-based sorting algorithms. It also discusses algorithms with unusual and interesting proofs of their complexity, such as Euclid's algorithm or KMP for pattern matching.

The chapter that follows, Algorithms and Probabilities, accomplishes two goals. First, it introduces the students to a number of algorithms of a probabilistic nature. Second, it uses elements of probability theory to compute the average complexity of some algorithms such as Quicksort. Textbooks rarely discuss the computation of the expected value of algorithms in such detail; many of them only cover the best and worst case complexities of Quicksort, for example.

The last chapter introduces a variety of classical finite graph algorithms, such as are taught in upper level computer science classes, together with an analysis of their complexity. These algorithms have many applications and provide an excellent playground for applying the various ideas introduced before.

http://www.springer.com/978-3-319-09887-6

Practical Analysis of Algorithms

Vrajitoru, D.; Knight, W.

2014, XII, 466 p. 245 illus., Softcover
ISBN: 978-3-319-09887-6