Theoretical Basis

Nicolas Von Dolinger Moreira Rocha
2022035571
Federal University of Minas Gerais - UFMG

August 19, 2025

Abstract

This document presents a resume of the chapters 1, 2, 3 and 34 (34.1 and 34.2) of the book "Cormen et al., "Introduction to Algorithms", 2009".

1 Chapter 1: The Role of Algorithms in Computing

1.1 Algorithms

This subsection introduces the concept of algorithms. The author defines them as a well-defined computational procedure that takes a value, or a set of values, as input and produces a value, or a set of values, as output. There is a distinction between correct algorithms, which produce the expected output for every instance, and incorrect algorithms, which produce an incorrect output for one or more instances. However, this type of incorrect algorithm can be useful for some kinds of problems.

Many examples are given of the utility of algorithms to solve problems: sorting, scheduling, minimum path, etc. Every problem uses a data structure (a way to organize the data for the problem) and a technique (the way to model and solve the problem).

1.2 Algorithms as a technology

This subsection talks about the importance of studying algorithms and their efficiency, even if the computer running the code is fast and memory is inexpensive and large.

Moreover, the author describes the importance of algorithms and their presence in all kinds of technologies, even the ones that appear to be "simple" for the end user.

2 Chapter 2: Getting Started

2.1 Insertion Sort

In this subsection, the author introduces the concept of Insertion Sort, which is a sorting algorithm usually used to sort small sets of numbers. The concept of this algorithm is simple: for every key in the vector, it is taken and put in the right position. Throughout the process, the cards on the left will be sorted and the right side will be waiting to be sorted.

For the algorithm to be correct, a loop invariant needs to satisfy three conditions: Initialization (it is true prior to the first iteration of the loop), maintenance (if it is true before an iteration of the loop, it remains true before the next iteration) and termination (when the loop terminates, the invariant helps us show that the algorithm is correct).

2.2 Analyzing algorithms

In this subsection, the author introduces the idea of analyzing the cost of an algorithm, as he usually does in this book, using the RAM model, which is a simpler way to analyze a given algorithm.

Moreover, the author explains the difference between the input/instance size of a given problem and the running time (the sum of running times for each statement executed), which usually grows according to the input size.

For example, insertion sort's running time in the worst case is a quadratic function. Therefore, its complexity in the worst case is theta of n-squared. This means that the running time grows proportionally to n-squared, which describes a tight bound for the algorithm's growth rate.

2.3 Designing algorithms

In this section, the author introduces the concept of design algorithms to solve the same problem in different ways. To exemplify this, he introduces the concept of the divide-and-conquer approach, that is used in the merge sort.

2.3.1 The divide-and-conquer approach

The divide-and-conquer approach usually works recursively, dividing the given problem into smaller ones, solving them when it is possible, and combining the answers to get the answer to the previous problem.

Moreover, the author proves the initialization, the maintenance, and the termination of the merge sort, which proves its correctness while using the divide-and-conquer approach.

2.3.2 Analyzing divide-and-conquer algorithms

When an algorithm works recursively, we can describe its running time using a recurrence equation, based on how it works. To exemplify this, it uses the merge sort algorithm, which divides the problem on each recursive call, until it gets to the leaf of the recursive tree that was built, and merges the results. So, it proves that the cost of this algorithm is theta of log n.

3 Chapter 3: Growth of Functions

3.1 Asymptotic notation

The author introduces important notation that is often used in the book and in the study of computer science. The most important notations, related to the runtime of algorithms, are: Theta notation (), which "sandwiches" a function f using a function g; Big O (O), which creates an upper bound for a function f; and Big Omega (), which, like Big O,

creates a lower bound. There are also variations: little-o (o) and little-omega (), both used to provide strict bounds that are not asymptotically tight.

Moreover, it introduces properties for comparing functions, such as transitivity, reflexivity, and symmetry, that are useful for establishing relations between them.

3.2 Standard notations and common functions

This section reviews standard mathematical functions and notations, exploring their concepts. Examples include: monotonicity, floors and ceilings, modular arithmetic, polynomials, exponentials, and logarithms expressions, etc.

4 NP-Completeness

4.1 Polynomial time

Here, the author defines the concept of Class P, which is the class of decision problems that can be solved in polynomial time.

4.2 Polynomial-time verification

The author shows that, even when we have an NP problem, given a certificate for this problem, we can verify that certificate in polynomial time.

Bibliografia

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009.