Introduction to Algorithms

Note

April 16, 2020

Contents

Part I

Foundations

Chapter 1

The Role of Algorithms in Computing

1. Algorithms: an *algorithm* is any well defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output*.

2. Data Structure: a *data structure* is a way to store and organize data in order to facilitate access and modifications.

3. **NP-complete Problems:**

   - Although no efficient algorithm for an NP-complete problem has ever been found, nobody has ever proven that an efficient algorithm for one cannot exist.
   - If an efficient algorithm exists for any one problem, then efficient algorithms exist for all of them.
   - Several NP-complete problems are similar, but not identical, to problems for which we do know of efficient algorithms. Hence a small change to the problem statement can cause a big change to the efficiency of the best known algorithm.

Chapter 2

Getting Started

2.1 Insertion sort

1. **Insertion Sort:**

   - **Pseudocode:**

     INSERTION-SORT(A)
     1 **for** $j = 2$ **to** $A.length$
     2 $key = A[j]$
     3 // Insert $A[j]$ into the sorted sequence $A[1 \ldots j-1]$.
     4 $i = j - 1$
     5 **while** $i > 0$ and $A[i] > key$
     6 $A[i+1] = A[i]$
     7 $i = i - 1$
     8 $A[i+1] = key$

   - IN PLACE: it rearranges the nunbers within the array A, with at most a constant nunber of them stored outside the array at any time.
   - **Loop invariants:**
     - Three things about a loop invariant:
       (a) **Initialization:** It is true prior to the first iteration of the loop.
       (b) **Maintenance:** If is true before an iteration of the loop, it remains true before the next iteration.
       (c) **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithms is correct.
     - When the loop is a **for** loop, the moment at which we check the loop invariant just prior to the first iteration is immediately after the initial assignment to the loop-counter variable and just before the first test in the loop header.

2.2 Analyzing algorithms

1. **Random-access machine (RAM) model:**
   - In the *RAM* model, instructions are executed one after another, with no concurrent operations.
   - The *RAM* model constains instructions commonly found in real computers: arithmetic, data movement, and control.
   - Such computers can compute 2^k in one constant-time instruction by shifting the integer 1 by k positions to the left, as long as k is no more than the number of bits in a computer word.
2. **Analysis of insertion sort:** INSERTION-SORT can take different amounts of time to sort two input sequences of the same size depending on how nearly sorted they already are. To analyze it, we need to define the terms "running time" and "size of input" more carefully:
   - The best notion for *input size* depends on the problem being studies.
   - The *running time* of an algorithm on a particular input is the number of primitive operations or "steps" executed.
3. **Worst-case, average-case, and best-case analysis**
   - **Order of growth:** Consider only the leading term of a formula and at the same time ignore the leading term's constant coefficient.

2.3 Designing algorithms

2.3.1 The divide-and-conquer approach