USI Lugano
Computer Science Department

Algoritms & data structures book

Professor
Prof. Carzaniga

TA's
F. Di Lauro, S. Mangipudi

Student
Krit Pio Nicol

Year 2024

Abstract

Not applicable

Contents

Chapter 1

The role of algorithms in computing (1-38)

Lecture 1: 1-27

Lecture 2: 27-54

19 Feb

1.1 Algoriths

21 Feb

1.1.1 Data structures

Definition 1.1.1 (Data structure). A way to store and organize data in order to facilitate access and modifications.

Definition 1.1.2 (Efficiency). Different algorithms solve the same problem but have different level of efficiency.

Example (Insertion sort vs merge sort). We note that the two sorting algorithms have different efficiency

Note (Insertion sort). $C_1 * n^2$ where C_1 is constant independent of n

Note (Merge sort). $C_2 * n * \log_2 n$ where C_2 is constant independent of n

Remark (Constant factor). insertion sort has a smaller constant factor than merge sort $C_1 < C_2$. The majority of the times constant factor has less influence than input size n

Definition 1.1.3 (Constant factor). Anything that doesn't depend on the input parameter(s)

Definition 1.1.4 (Input size). The input size can be the following:

- Number of items in the input, eg the number of items to sort
- Total number of bits, eg bitwise multiplication to multiply 2 integers
- Input size in term of 2 numbers, eg for finding the shortest path in a graph from a given source

Chapter 2

Getting started (39-49)

2.1 Insertion sort

Definition 2.1.1 (keys). The numbers to be sorted. The input comes in the form of an array with "n" elements. The keys are often associated with other data, called "satellite data". Together they form a "record"

Definition 2.1.2 (Arrays). A data structure. A collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

Example (How it works). To sort an array of size N in ascending order iterate over the array and compare the current element (key) keys to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Algorithm 2.1: Insertion sort

for $i \leftarrow 2$ **to** n **do**
    $key = A[1]$; `// Assign variable "key" to the index "i" of array "A"`
    $j = i - 1$; `// Insert` $A[1]$ `into the sorted sub-array` $A[1 : i - 1]$
    `/* Conditional check while true */`
    while $j > 0$ *and* $A[j] > key$ **do**
        $A[j + 1] = A[j]$; `// Retrive value at index "j" and assign this value to the element at index` $j + 1$
        $j = j - 1$
    ; `// Swap` j `with the value at index` $j - 1$
    $A]j + 1] = key$

Loop invariant

Definition. Loop invariant

Definition 2.1.3 (Loop invariant). A loop invariant is a condition (among program variables) that is necessarily true immediately before and immediately after each iteration of a loop.

Note (Boolean state thorugh the iteration). Note that this says nothing about its truth or falsity part way through an iteration.

2.2 Analyzing algorithms

Note (running time). The running time is dependent on various factor, such as input size, current status[a], machine hardware.

However you can evaluate a formula depending on the input size, which is a general explanation of the time complexity

[a]How many keys are already sorted

Definition 2.2.1 (Running time). Number of instruction and data accesess executed