University of Pisa
Algorithm Engineering

Enrico Fiasco

Year 2025

Contents

1 Sampling and Sorting

Definition 1.1 (Algorithm)**.** Downald Knuth defines Alrgorithm as a *finite, definite, effective procedure, with some output.*

- **Finite**: An algorithm must always terminate after a finite number of steps...a very finite numer, a reasonable numer.
- **Definite**: Each step of an algorithm must be precisely defined; the action to be carried out must be rigorously and unambiguously specified for each case.
- **Effective**: All of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite lenght of time by a man using paper and pencil.
- **Procedure**: The sequence of specific steps arranged in a logical order.
- **Input**: Quantities which are given to it initlially before the algorithm begins. These inputs are taken from specified sets of objects.
- **Output**: Quantities which have a specified relation to the inputs.

1.1 Time Complexity

Time complexity is measured over the number of steps $T(n)$, where n is the input size. We take the maximum $T(n)$ over all the inputs that induce in the worst behavior and we call it *worst case.*

1.2 The RAM model

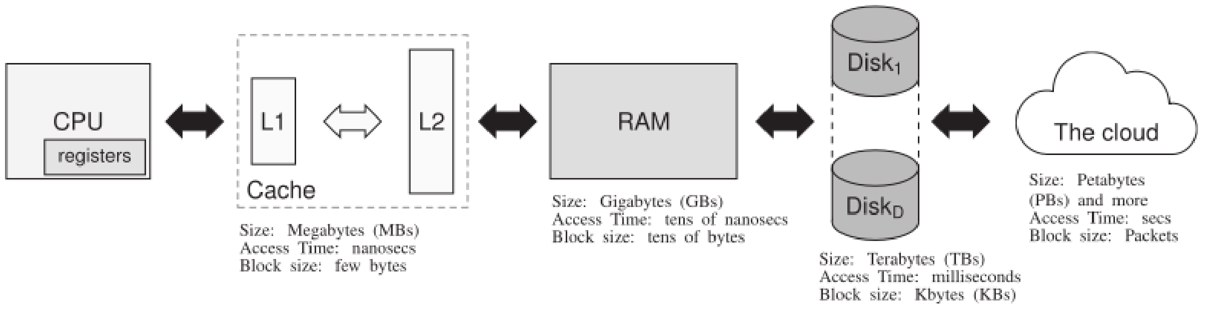

Figure 1: Memory Access Levels and the access Time

Random Access Memory model is a type of computing model where the memory size is assumed to be infinite, so every operation done on the `CPU` is assumed to be a finite value. This can lead out the importance of memory access, where as shown in Figures 1, if not handled correctly can drasticly reduce the efficiency of our algorithms due to the *time* needed to fetch new data from different levels of memory which can increase drasticly. If we try to read a value bigger than the capacity of our registers that can lead to use more than one `IOs` operation to execute a computational step which we assumed as one in the **RAM** model.

1.3 Why should we care about Asymthotics

Here we are gona show how the importance of algorithm complexity if much important than the speed of our machine by using this three examples.

$$\begin{cases} T_1(n) = n \rightarrow n = t \rightarrow n^{'} = k \times t \\ T_2(n) = n^2 \rightarrow n = \sqrt{t} \rightarrow n^{'} = \sqrt{k \times t} \\ T_3(n) = 2^n \rightarrow n = \log_2 t \rightarrow n^{'} = \log_2 k + \log_2 t \end{cases}$$

Here k rappresent a faster machine, as the complexity of the algorithm from T_1 to T_3 increase we can see that even if we use a k faster machine with the T_2 and T_3 algorithm the increase in speed and efficiency is not notable.

1.4 Summing items in an Array

A simple algorithm on which we can do a simple `IOs` complexity analisys can be the *Sum of elements in an Array.* We define an array as in Figures 2, and than we `SCAN` it (we read from left to right), block by block and sum the elements in the block to the accumulation variable.

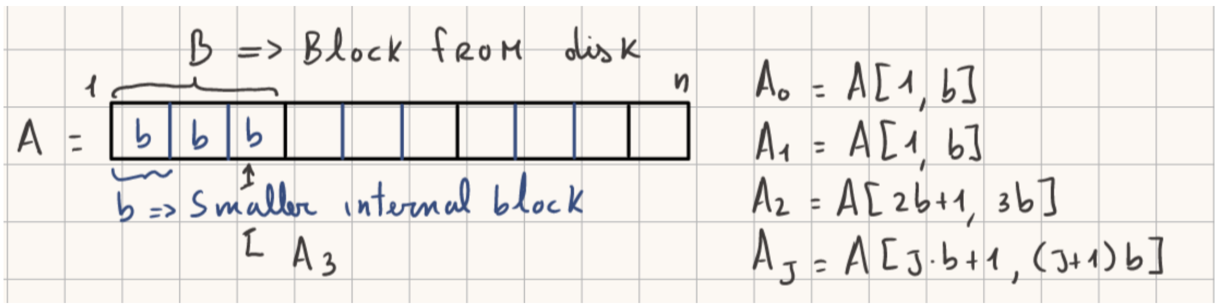

Figure 2: Sum of elements in an array

In this example we rappresent memory blocks, readed from the disk, by the B label, than the sub-blocks subdivinding the B in smaller blocks as b, and the access to those smaller block as $A_{0 \text{ or } 1} = A[1, b]$, $A_2 = A[2b + 1, 3b]$ and than the generic $A_j = A[jb + 1, (j + 1)b]$. The number of `IOs` operation is calculated as $\#\texttt{IOs} = \frac{n}{\#B}$. The following algorithm shows how to sum the the elements of all the blocks in the array.

Algorithm 1 Sum of elements

```
1: procedure SUM OF ELEMENTS($A, s, n, b$)
2:     $sum \leftarrow 0$
3:     for $i = 0$ to $(n/b) - 1$ do
4:         $j \leftarrow (i \times s) \bmod (n/b)$
5:         $sum \leftarrow sum +$ somma degli elementi in $A_j$
6:     end for
7:     return $sum$
8: end procedure
```

The algorith iterate over $\frac{n}{b}$ times which is equal to the total sub-blocks. Each iteration calculates j as the iteration time $(i \times s) \mod \frac{n}{b}$ where s is equal to the next step in the array.

1.4.1 `IO` Complexity

When $s = 1$ it scan the array jumping from $A_i \to A_{i+1}$ but when $s > 1$ than it can jump more than $+1$ position in the array. We can obtain the algorith `IO` Complexity as:

$$\texttt{IO}\ \text{Complexity} = \frac{n}{B} \times \min\{s, \frac{B}{b}\}$$

$$s \leq \frac{B}{b} \to s \times \frac{n}{B}$$

$$s > \frac{B}{b} \to \frac{n}{\cancel{B}} \times \frac{\cancel{B}}{b} = \frac{n}{b}$$

Where $\frac{B}{b}$ rappresent the number of total possible sub-blocks.

1.5 Binary Search