LABORATORIO E ANALISI DI ALGORITMI

Lezione 1

1.1 STRUMENTI PER L'ANALISI DEGLI ALGORITMI

Analizzare un algoritmo vuol dire predire e quantificare le risorse utilizzate dallo stesso.

Risorse: spazio (memoria), consumo di energia, larghezza della banda, soprattutto il tempo computazionale (RUNTIME)

Adotteremo il modello RAM (Random-Access Machine)

Caratteristiche del modello RAM:

- le istruzioni di un algoritmo sono eseguite in maniera sequenziale, senza operazioni contemporanee.
- ogni istruzione e ogni accesso alla memoria avviene in tempo costante.

ISTRUZIONI ELEMENTARI NEL MODELLO RAM:

- ARITMETICHE
- SPOSTAMENTO DEI DATI (carica, salva, copia)
- CONTROLLO (branching condizionale o incondizionale, subroutine, return).

DATA TYPES nel modello RAM:
- interi;
- floating point;
- caratteri.

- Ogni stringa di dati ha un numero limitato di bit.
- Non si tiene conto della gerarchia di memoria.

Vogliamo espressioni che descrivono l'andamento dell'algoritmo che possano essere formule che dipendono dall'input.

Es: Quanto tempo impiega INSERTION SORT?
Esaminiamo il numero di volte che l'algoritmo esegue ogni riga di pseudocodice e quanto tempo richiede ciascuna riga per essere eseguita.

Il TEMPO COMPUTAZIONALE di un algoritmo su un input di fissata misura n è il numero di istruzioni e accessi ai dati eseguiti. Nel modello RAM, assumiamo che sia necessaria una quantità costante di tempo per eseguire ogni riga.

La MISURA DELL'INPUT dipende dal problema da risolvere.

1.2. ANALISI DI INSERTION SORT

		costo	ripetizioni
1.	for $i = 2$ to n	c_1	n
2.	$key = A[i]$	c_2	$n-1$
3.	// insert into the sorted subarray		
4.	$j = i-1$	c_4	$n-1$
5.	while $j>0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6.	$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i-1)$
7.	$j = j-1$	c_7	$\sum_{i=2}^{n} (t_i-1)$
8.	$A[j+1] = key$	c_8	$n-1$

t_i: # volte che il ciclo while alla riga 5 viene eseguito, per i.

$$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_4 (n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8 (n-1)$$

Best case $\quad \forall i, \; t_i = 1$

$$T(n) = c_1 \cdot n + c_2 n - c_2 + c_4 n - c_4 + c_5 (n-1) + c_8 (n-1)$$

$$= \underbrace{(c_1 + c_2 + c_4 + c_5 + c_8)}_{a} n + \underbrace{[-(c_2 + c_4 + c_5 + c_8)]}_{b}$$

$$T(n) = an + b \quad \rightsquigarrow \quad T(n) = O(n)$$

Worst case

$\forall i \in \{2,3,\dots,n\}$, $t_i = i$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right)$$

$$+ c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \underbrace{\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)}_{a} n^2 + \underbrace{\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)}_{b} n + \underbrace{\left[-(c_2 + c_4 + c_5 + c_8)\right]}_{c} = an^2 + bn + c \qquad a > 0$$

$\in O(n^2)$

Un algoritmo è più efficiente di un altro se il suo tempo computazionale nel caso peggiore è di un ordine di grandezza inferiore, ovvero cresce più lentamente al crescere della misura dell'input.

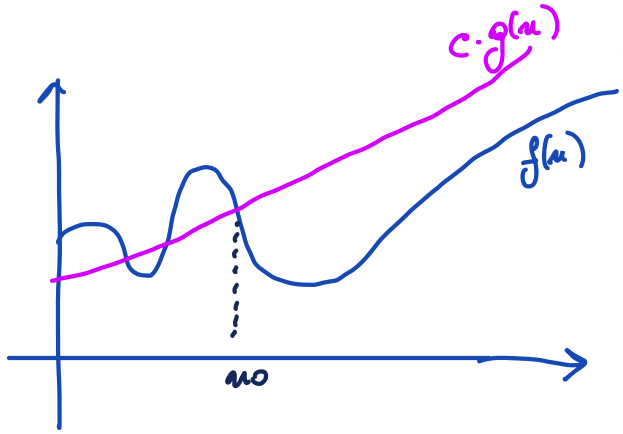
> CONVENZIONE:
> $T(n) = f(n)$

1.3 NOTAZIONI ASINTOTICHE

(big-Oh) **O-NOTATION** $(f(n) \in O(n^3))$

es: $f(n) = 7n^3 + 10n^2 - 20n + 6$, $f(n) = O(n^3)$, $f(n) = O(n^4)$, $f(n) = O(n^5)$

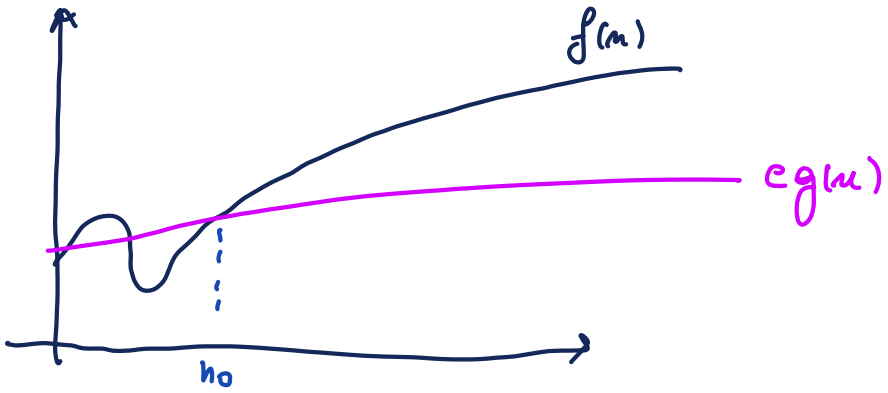
upper bound



$$O(g(n)) = \{ f(n) \mid \exists\ c>0 \text{ costante e } n_0 \in \mathbb{N} \text{ tale che } 0 \le f(n) \le c\,g(n) \quad \forall\ n \ge n_0 \}$$

Ω-NOTATION

es: $7n^3 + 10n^2 - 20n + 6 = \Omega(n^3) = \Omega(n^2) = \Omega(n)$ } lower bound

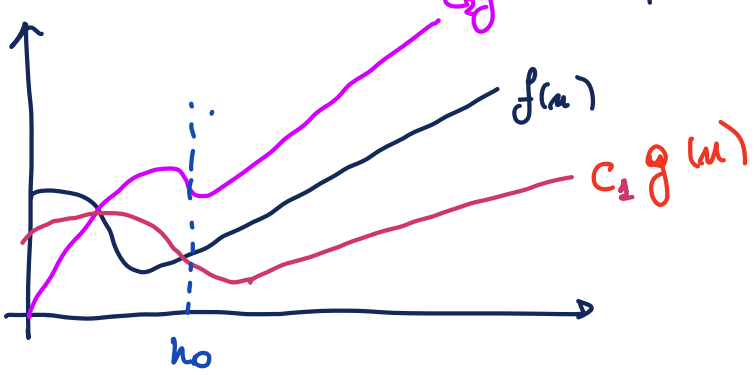


$$\Omega(g(n)) = \{ f(n) \mid \exists \text{ costanti } c>0 \text{ e } n_0 \in \mathbb{N} : 0 \le cg(n) \le f(n) \quad \forall\ n \ge n_0 \}$$

Θ-NOTATION

es. $7n^3 + 10n^2 - 20n + 6 = \Theta(n^3) \neq \Theta(n^2), \neq \Theta(n^4)$

Tight bound



$$\Theta(g(n)) = \left\{ f(n) \;\middle|\; \begin{array}{l} \exists\, c_1, c_2 > 0 \text{ costanti e } n_0 \in \mathbb{N} : \\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \quad \forall n \ge n_0 \end{array} \right\}$$

THM: $\forall\, f(n), g(n)$

$$f(n) = \Theta(g(n)) \iff \begin{cases} f(n) = O(g(n)), \\ f(n) = \Omega(g(n)). \end{cases}$$

(Little-oh notation) o-notation

Per notazioni non asintoticamente precise.

es: $2n^2 = O(n^2)$ → asintoticamente precise (asymptotically tight)

$2n = O(n^2)$ → non asintoticamente precise

$2n = o(n^2)$, mentre $2n^2 \neq o(n^2)$

$$o(g(n)) = \left\{ f(n) \;\middle|\; \begin{array}{l} \forall\, c > 0 \text{ costante}, \quad \exists\, n_0 \in \mathbb{N} \text{ s.t.} \\ 0 \le f(n) < c\, g(n) \end{array} \right\}$$

↘ $f(n)$ è asintoticamente più piccola di $g(n)$

Oss: $f(n) = o(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.

little-omega

$$\omega(g(n)) = \left\{ f(n) \;\middle|\; \begin{array}{l} \forall\, c > 0 \text{ costante}, \exists\, n_0 \in \mathbb{N} : \\ 0 \le c\, g(n) < f(n) \quad \forall n > n_0 \end{array} \right\}$$

$f(n)$ è asintoticamente più grande di $g(n)$

Oss: $f(n) = \omega(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = +\infty$ (se esiste)

OSS ? : $f(n) = \omega(g(n)) \iff g(n) = O(f(n))$

PROPRIETÁ

- Transitive: $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n)) \to f(n) = \Theta(h(n))$
  (vale pure per O, Ω, o, ω)

- Riflessiva: $f(n) = \Theta(f(n))$, $f(n) = O(f(n))$, $f(n) = \Omega(f(n))$
- Simmetria: $f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$
- Simmetria trasposta: $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$
  $f(n) = o(g(n)) \iff g(n) = \omega(f(n))$

NON vale il totale ordinamento:

$\exists \; f(n) \neq O(g(n))$, $f(n) \neq \Omega(g(n))$

es: non possiamo confrontare n e $n^{1+\sin n}$.