RANDOM ACCESS MACHINE (RAM)

- Le istruzioni di un algoritmo sono eseguite in maniera sequenziale

- Ogni istruzione elementare e accesso alla memoria impiega tempo costante.

ISTRUZIONI ELEMENTARI NEL MOD RAM:

- operazioni aritmetiche (+, -, x, ÷, %, floor ⌊ ⌋, ceiling ⌈ ⌉);
- spostamento dei dati (upload, save, copy);
- controllo (branching condizionale e incondizionale, chiamata di subroutine e return).

FORMATO DEI DATI:
- interi
- floating point
- caratteri

Ogni stringa di dati è codificata da un numero limitato di bit.

es: se devo codificare una stringa di lunghezza n posso assumere che essa sia rappresentata con $c \cdot \log_2 n$, $\exists c \in \mathbb{Q}$, $c \geq 1$.

Il modello RAM non tiene conto della gerarchia di memoria

misura dell'input dipende dal tipo di dato.

- lunghezza

TEMPO COMPUTAZIONALE (RUNTIME): numero di istruzioni elementari e di accessi ai dati eseguiti, in funzione della lunghezza dell'input.

Lezione 1

CORRETTEZZA e ANALISI DELLE PERFORMANCE

$\Updownarrow$

utilizzo delle risorse:

- memoria
- tempo computazionale (runtime)
- larghezza della banda di comunicazione
- energia consumata

RANDOM-ACCESS MACHINE (RAM)

assumeremo che gli algoritmi siano formalizzati come programmi informatici scritti in pseudocodice

- le istruzioni sono eseguite in maniera sequenziale, senza operazioni contemporanee
- ogni singola istruzione elementare e ogni accesso alla memoria avvengono in tempo costante.

Istruzioni elementari nel modello RAM:

- operazioni aritmetiche ($+, -, \times, \div, \%$, floor $\lfloor\ \rfloor$, ceiling $\lceil\ \rceil$);
- spostamento dei dati (upload, save, copy);
- controllo (branching condizionale e incondizionale, le chiamate di subroutine e return)

FORMATO DEI DATI: interi, floating point e caratteri.
Ogni stringa di dati può essere codificata da un numero limitato di bit.

Il modello RAM non tiene conto delle gerarchie di memoria.

La nozione di misura dell'input dipende dal problema (task) considerato e dalle strutture dati che impiegheremo.

es:

- array ⇝ # elementi dell'array;
- albero ⇝ # nodi;
- grafi ⇝ # nodi e # archi.

Tempo computazionale: numero di istruzioni elementari e accessi ai dati eseguiti, in funzione della misura dell'input.

ESEMPIO: INSERTION SORT

$A[1, \dots, n]$

	costo	ripetizioni
for $i = 2$ to n	c_1	n
$key = A[i]$	c_2	$n-1$
$j = i-1$	c_3	$n-1$
while $j > 0$ and $A[j] > key$	c_4	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_5	$\sum_{i=2}^{n} (t_i - 1)$
$j = j-1$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_7	$n-1$

t_i = # di volte che il ciclo while alla linea 4 viene eseguito, $i \in \{2, \dots, n\}$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i + c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7(n-1).$$

BEST CASE : l'input è già ordinato $\rightarrow t_i = 1 \quad \forall\, i \in \{2, \dots, n\}$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_5 \cdot 0 + c_6 \cdot 0 + c_7(n-1) =$$

$$= \underbrace{(c_1 + c_2 + c_3 + c_4 + c_7)}_{a} n + \underbrace{(-c_2 - c_3 - c_4 - c_7)}_{b} = an + b$$

tempo lineare

WORST CASE: l'input è ordinato in maniera inversa $\rightsquigarrow t_i = i$, per $i \in \{2, \dots, n\}$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} i + c_5 \sum_{i=2}^{n} (i-1) + c_6 \sum_{i=2}^{n} (i-1) + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5 \frac{(n-1)n}{2} + c_6 \frac{(n-1)n}{2} + c_7(n-1) =$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

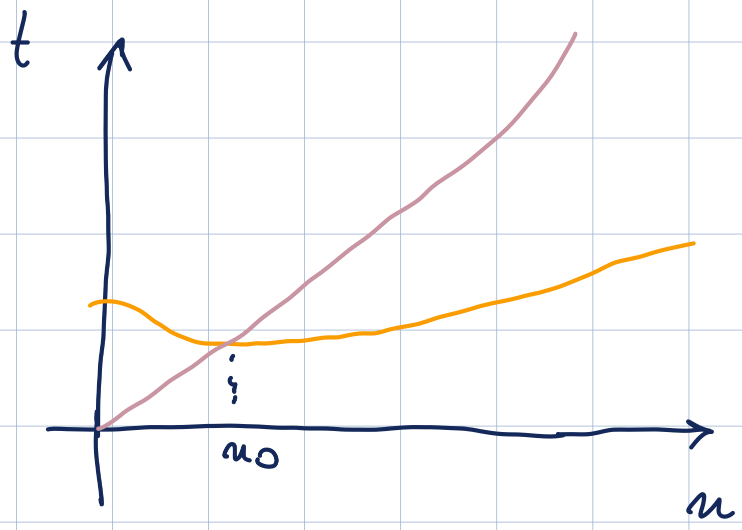
$$\sum_{i=2}^{n} i = \left(\sum_{i=1}^{n} i\right) - 1 = \frac{n(n+1)}{2} - 1 \quad , \quad \sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

$$= \underbrace{\left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)}_{a} n^2 + \underbrace{\left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)}_{b} n + \underbrace{\left(-c_2 - c_3 - c_4 - c_7\right)}_{c} =$$

$$= an^2 + bn + c \quad \rightsquigarrow \quad \text{tempo quadratico}$$

Ci interessa l'ordine di grandezza del tempo computazionale.

Un algoritmo è più efficiente di un altro se il runtime nel caso peggiore cresce più lentamente al crescere della lunghezza dell'input.



La notazione asintotica

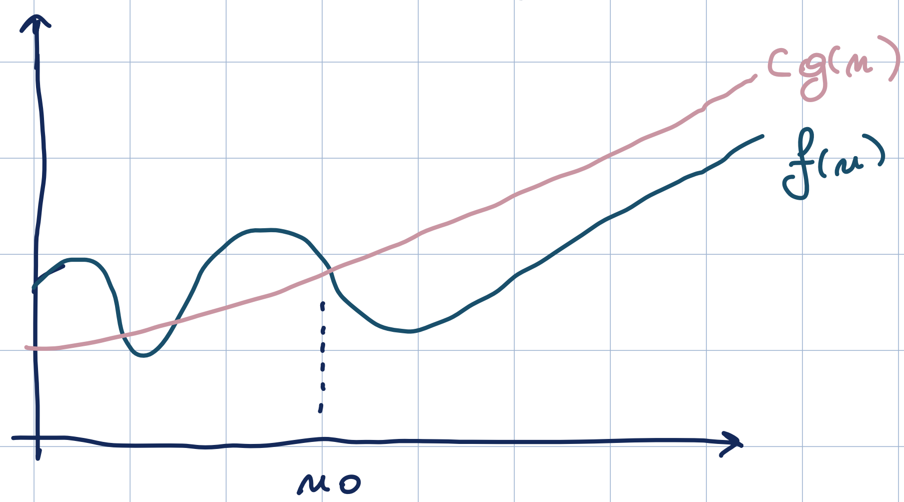
la notazione big-Oh

O : denota un upper bound (limite superiore) al comportamento asintotico di una funzione.

es: $f(n) = 7n^3 + 10n^2 - 20n + 6$ $\rightsquigarrow$ questa funzione (asintoticamente) non cresca più velocemente di $8n^3$

$\hookrightarrow f(n) = O(n^3)$

$f(n) = O(n^4)$, $f(n) = O(n^5)$, $f(n) = O(n^3 \log n)$.



$$O(g(n)) = \{ f(n) \mid \exists c \in \mathbb{Q}_{>0}, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 \quad 0 \leq f(n) \leq c \cdot g(n) \}$$

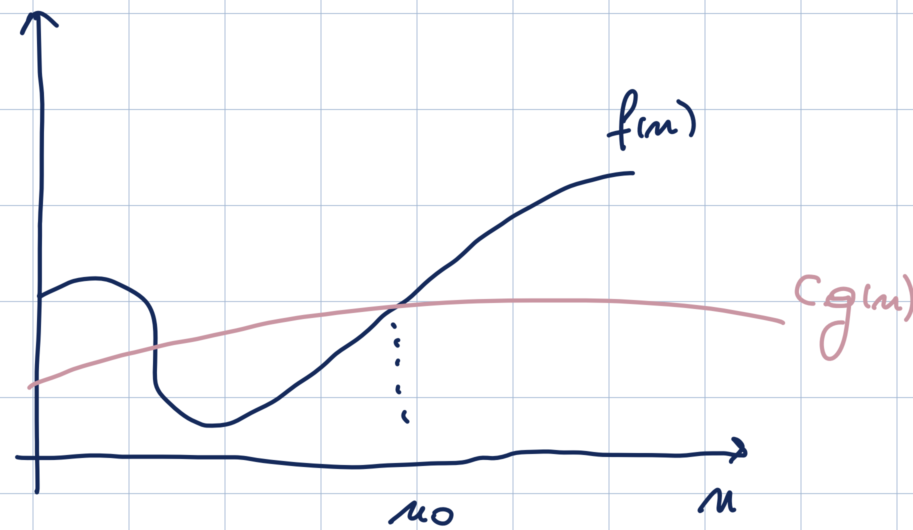
f(n), g(n) sono non negative

- Ω: denota un lower bound (limite inferiore) asintotico

es: $f(n) = 7n^3 + 10n^2 - 20n + 6$ → questa funzione non cresce più lentamente di $7n^3$

$\rightarrow f(n) = \Omega(n^3)$

$f(n) = \Omega(n^2)$, $f(n) = \Omega(n)$



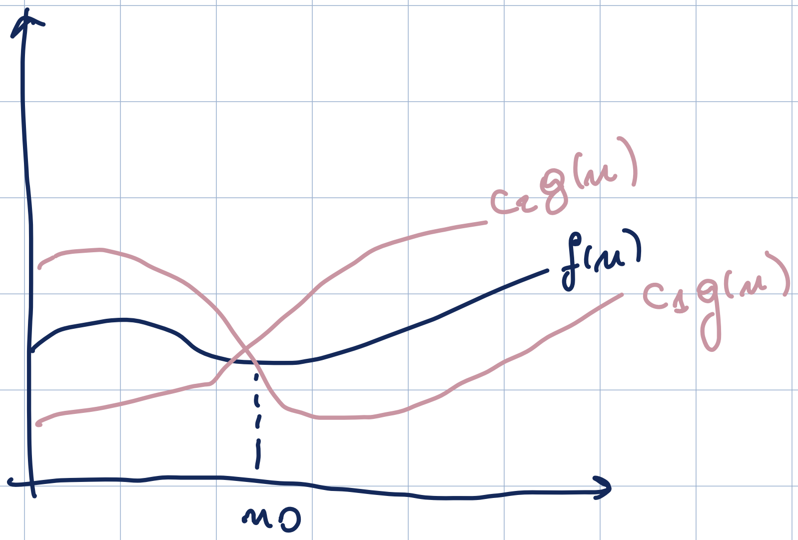
$$\Omega(g(n)) = \{ f(n) \mid \exists c \in \mathbb{Q}_{>0}, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 \quad 0 \leq c \cdot g(n) \leq f(n) \}$$

PROP: $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$

Θ: denota limiti (superiori e inferiori, tight bound)

es: $f(n) = 7n^3 + 10n^2 - 20n + 6$ è $\Theta(n^3)$

$$\Theta(g(n)) = \left\{ f(n) \;\middle|\; \begin{array}{l} \exists\, c_1, c_2 \in \mathbb{Q}_{>0},\ \exists\, n_0 \in \mathbb{N} : \forall\, n \geq n_0 \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \end{array} \right\}$$



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

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

PROPRIETÀ: $f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$ (proprietà simmetrica)

PROPRIETÀ RIFLESSIVE: $f(n) = O(f(n))$, $f(n) = \Omega(f(n))$, $f(n) = \Theta(f(n))$

Il tempo computazionale di insertion sort:

- caso migliore $T(n) = O(n)$, $T(n) = \Omega(n)$, $T(n) = \Theta(n)$
- caso peggiore $T(n) = O(n^2)$, $T(n) = \Omega(n^2)$, $T(n) = \Theta(n^2)$

insertion sort in generale $T(n) = O(n^2)$, $T(n) = \Omega(n)$.

<u>little-oh notation</u>

o: denota upper bound non asintoticamente stretto (o precisi)
ω: denota lower bound non asintoticamente stretto (o precisi)

<u>Es:</u>

$2n = O(n^2) \wedge 2n \neq \Theta(n^2) \rightarrow 2n = o(n^2)$

$2n^2 = O(n^2) \wedge 2n^2 = \Theta(n^2) \rightarrow 2n^2 \neq o(n^2)$

$$o(g(n)) = \left\{ f(n) \;\middle|\; \begin{array}{l} \forall c \in \mathbb{Q}_{>0},\ \exists n_0 \in \mathbb{N} : \ \forall n \geq n_0 \\ 0 \leq f(n) < c\, g(n) \end{array} \right\}$$

PROP. :

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

es :

$$2n = \Omega(n), \quad 2n = \Theta(n) \quad \rightarrow \quad 2n \neq \omega(n)$$

$$2n^2 = \Omega(n), \quad 2n^2 \neq \Theta(n) \quad \rightarrow \quad 2n^2 = \omega(n)$$

$$\omega(g(n)) = \left\{ f(n) \;\middle|\; \begin{array}{l} \forall c \in \mathbb{Q}_{>0},\ \exists n_0 \in \mathbb{N} : \ \forall n \geq n_0 \\ 0 \leq c\, g(n) < f(n) \end{array} \right\}.$$

PROPR:

$$f(n) = \omega(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \quad (\text{se } \exists)$$

PROPRIETÀ DI SIMMETRIA TRASPOSTA:

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \iff g(n) = \omega(f(n))$$

PROPR. TRANSITIVE

$$f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = o(g(n)) \wedge g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \wedge g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

Es: n e $n^{1+\sin n}$ non sono comparabili → non sempre posso usare la notazione asintotica per confrontare due funzioni