Algoritmi e Strutture Dati

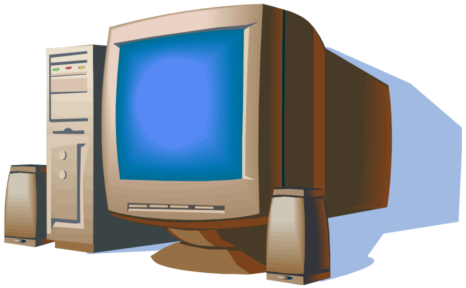
Luciano Gualà
guala@mat.uniroma2.it
www.mat.uniroma2.it/~guala

riassunto puntate precedenti

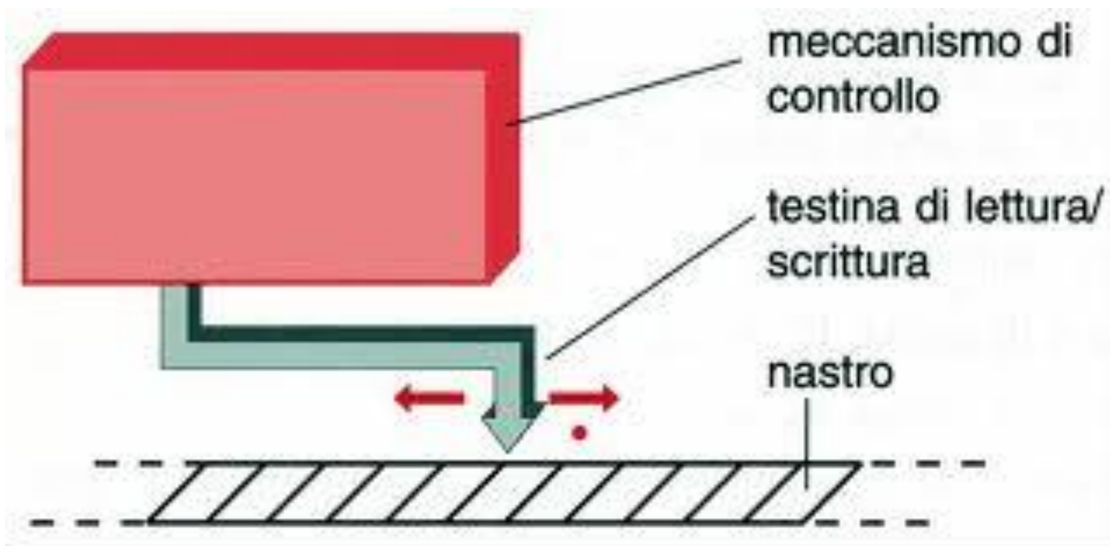
Abbiamo un problema a cui sono associate diverse (infinite) istanze di diversa dimensione. Vogliamo risolvere (automaticamente) il problema progettando un algoritmo. L'algoritmo sarà eseguito su un modello di calcolo e deve descrivere in modo non ambiguo (utilizzando appositi costrutti) la sequenza di operazioni sul modello che risolvono una generica istanza. La velocità dell'algoritmo è misurata come numero di operazioni eseguite sul modello e dipende dalla dimensione e dall'istanza stessa. Analizzare la complessità computazionale di un algoritmo vuol dire stimare il tempo di esecuzione dell'algoritmo nel caso peggiore in funzione della dimensione dell'istanza.
Sappiamo progettare un algoritmo veloce? Fin dove possiamo spingerci con la velocità? A volte si può dimostrare matematicamente che oltre una certa soglia di velocità non si può andare.

È sensato misurare la complessità di un algoritmo contando il numero di linee di codice eseguite?

modelli di calcolo



Un modello storico: la macchina di Turing



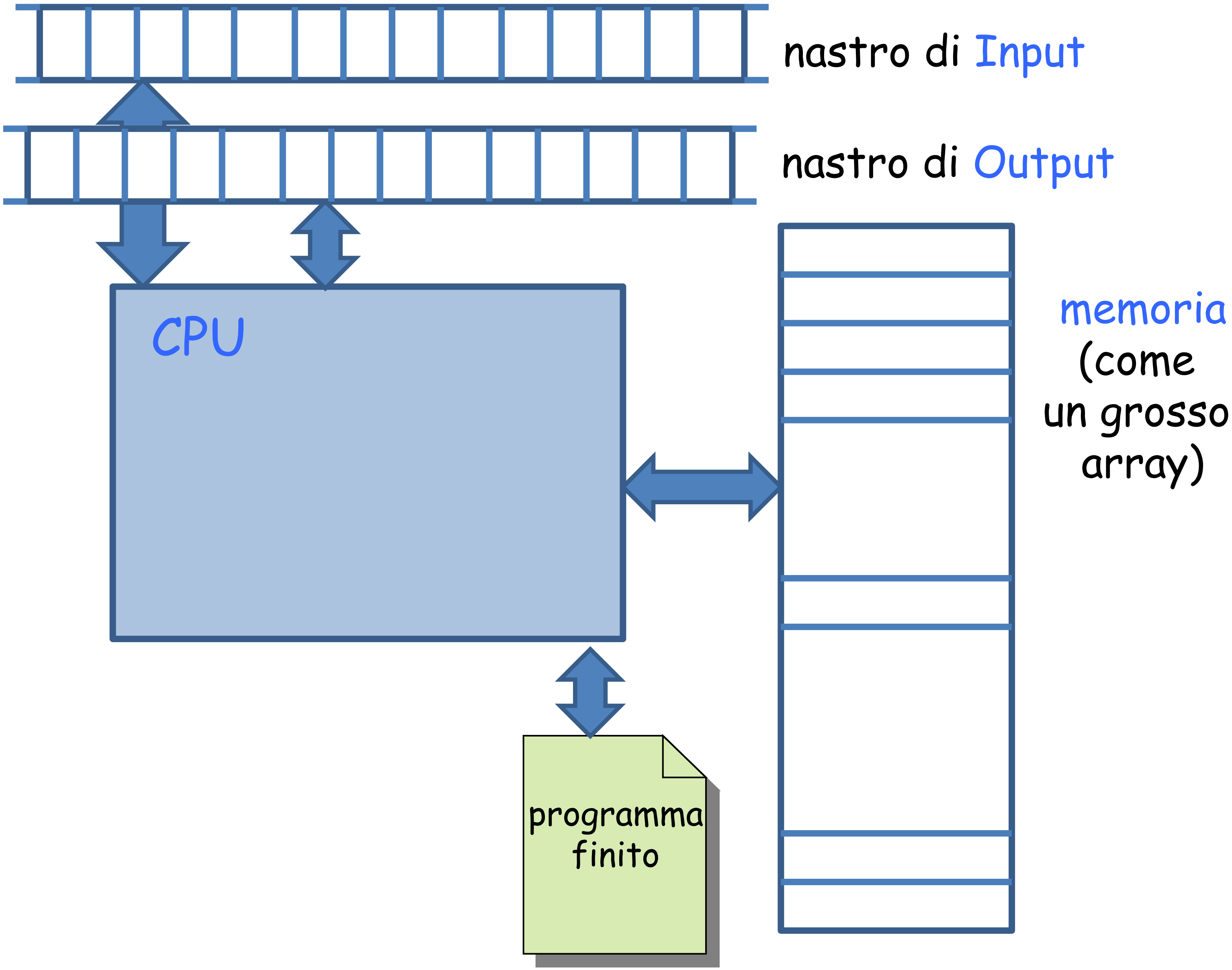
- troppo di basso livello: somiglia troppo poco ai calcolatori reali su cui girano i programmi
- utile per parlare di calcolabilità ma meno utile per parlare di efficienza

un modello più realistico

- Macchina a registri (RAM: *random access machine*)
  - un programma finito
  - un nastro di ingresso e uno di uscita
  - una memoria strutturata come un array
    - ogni cella può contenere un qualunque valore intero/reale
  - una CPU per eseguire istruzioni

- la RAM è un'astrazione dell'architettura di von Neumann

Macchina a registri
RAM: random access machine



Modello di calcolo: cosa posso fare

- L'analisi della complessità di un algoritmo è basata sul concetto di passo elementare

- passi elementari su una RAM
  - istruzione ingresso/uscita (accesso nastri I/O)
  - operazione aritmetico/logica
  - accesso/modifica del contenuto della memoria

Criteri di costo: quanto mi costa

- Criterio di costo uniforme:
  - tutte le operazioni hanno lo stesso costo
  - complessità temporale misurata come numero di passi elementari eseguiti

- Criterio di costo logaritmico
  - Il costo di una operazione dipende dalla dimensione degli operandi dell'istruzione
  - Un'operazione su un operando di valore x ha costo $\log x$
  - È un criterio di costo che modella meglio la complessità di algoritmi "numerici"

criterio di costo generalmente usato è quello uniforme

Caso peggiore e caso medio

- Misureremo il tempo di esecuzione di un algoritmo in funzione della dimensione n delle istanze
- Istanze diverse, a parità di dimensione, potrebbero però richiedere tempo diverso
- Distinguiamo quindi ulteriormente tra analisi nel caso peggiore e medio

Caso peggiore

- Sia $tempo(I)$ il tempo di esecuzione di un algoritmo sull'istanza I

$$T_{worst}(n) = \max_{\text{istanze } I \text{ di dimensione } n} \{tempo(I)\}$$

- Intuitivamente, $T_{worst}(n)$ è il tempo di esecuzione sulle istanze di ingresso che comportano più lavoro per l'algoritmo
- rappresenta una garanzia sul tempo di esecuzione di ogni istanza

Caso medio

- Sia $P(I)$ la probabilità di occorrenza dell'istanza I

$$T_{avg}(n) = \sum_{\text{istanze } I \text{ di dimensione } n} \{P(I)\ \text{tempo}(I)\}$$

- Intuitivamente, $T_{avg}(n)$ è il tempo di esecuzione nel caso medio, ovvero sulle istanze di ingresso "tipiche" per il problema
- Come faccio a conoscere la distribuzione di probabilità sulle istanze?
- Semplice: (di solito) non posso conoscerla
- -> faccio un'assunzione.
- spesso è difficile fare assunzioni realistiche

Esercizio

Analizzare la complessità nel caso medio del primo algoritmo di pesatura (Alg1) presentato nella prima lezione. Rispetto alla distribuzione di probabilità sulle istanze, si assuma che la moneta falsa possa trovarsi in modo equiprobabile in una qualsiasi delle n posizioni.

Una grande idea: notazione asintotica

Notazione asintotica: intuizioni

complessità computazionale di un algoritmo espressa con una funzioneT(n)

T(n): # passi elementari eseguiti su una RAM nel caso peggiore
su un'istanza di dimensione n

Idea: descrivere T(n) in modo qualitativo. Ovvero: perdere un po' in precisione (senza perdere l'essenziale) e guadagnare in semplicità

Notazione asintotica: intuizioni

T(n): # passi elementari eseguiti su una RAM nel caso peggiore su un'istanza di dimensione n

un esempio:

$$T(n)= \begin{cases} 71\, n^2 + 100 \lfloor n/4 \rfloor + 7 & \text{se } n \text{ è pari} \\ 70\, n^2 + 150 \lceil (n+1)/4 \rceil + 5 & \text{se } n \text{ è dispari} \end{cases}$$

scriveremo: $T(n)= \Theta(n^2)$

intuitivamente vuol dire: T(n) è proporzionale a n^2

cioè ignoro:
- costanti moltiplicative
- termini di ordine inferiore
  (che crescono più lentamente)

Nota:
l'assunzione implicita è che guardo come si comporta l'algoritmo su istanze grandi

…una vecchia tabella: numero asintotico di pesate

assunzione: ogni pesata richiede un minuto

TABELLA

n	10	100	1.000	10.000	100.000
Alg1	9m	~ 1h, 39m	~16 h	~6gg	~69gg
Alg2	5 m	~ 50 m	~8 h	~3,5gg	~35gg
Alg3	3 m	6m	9m	13m	16m
Alg4	3 m	5m	7m	9m	11m

$\Theta(n)$ pesate

$\Theta(\log n)$ pesate

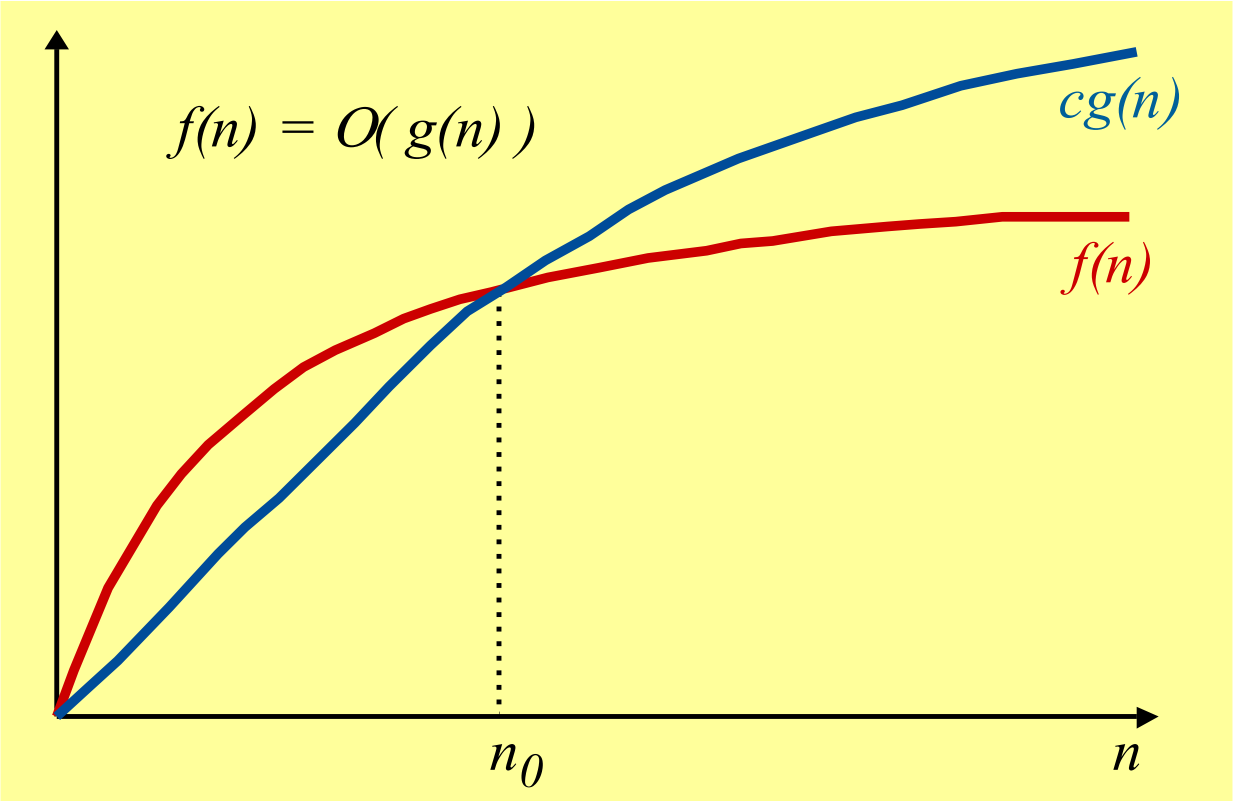
Un'altra tabella: dalla bilancia al computer

Tempi di esecuzione di differenti algorimi per istanze di dimensione crescente su un processore che sa eseguire un milione di istruzioni di alto livello al secondo.
L'indicazione very long indica che il tempo di calcolo supera 10^{25} anni.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Notazione asintotica O

$f(n) = O(g(n))$ se $\exists$ due costanti $c>0$ e $n_0 \geq 0$ tali che $0 \leq f(n) \leq c\, g(n)$ per ogni $n \geq n_0$



Esempi:

Sia $f(n) = 2n^2 + 3n$, allora

- $f(n)=O(n^3)$ $(c=1, n_0=3)$
- $f(n)=O(n^2)$ $(c=3, n_0=3)$
- $f(n) \neq O(n)$

Notazione asintotica O

$$O(\ g(n)\ )=\{f(n) \mid \exists\ c>0 \text{ e } n_0\geq 0 \text{ tali che } 0 \leq f(n) \leq c\ g(n) \text{ per ogni } n \geq n_0\}$$

- La scrittura:

$$2n^2+4=O(n^3)$$

- è un abuso di notazine per:

$$2n^2+4 \in O(n^3)$$

Notare:

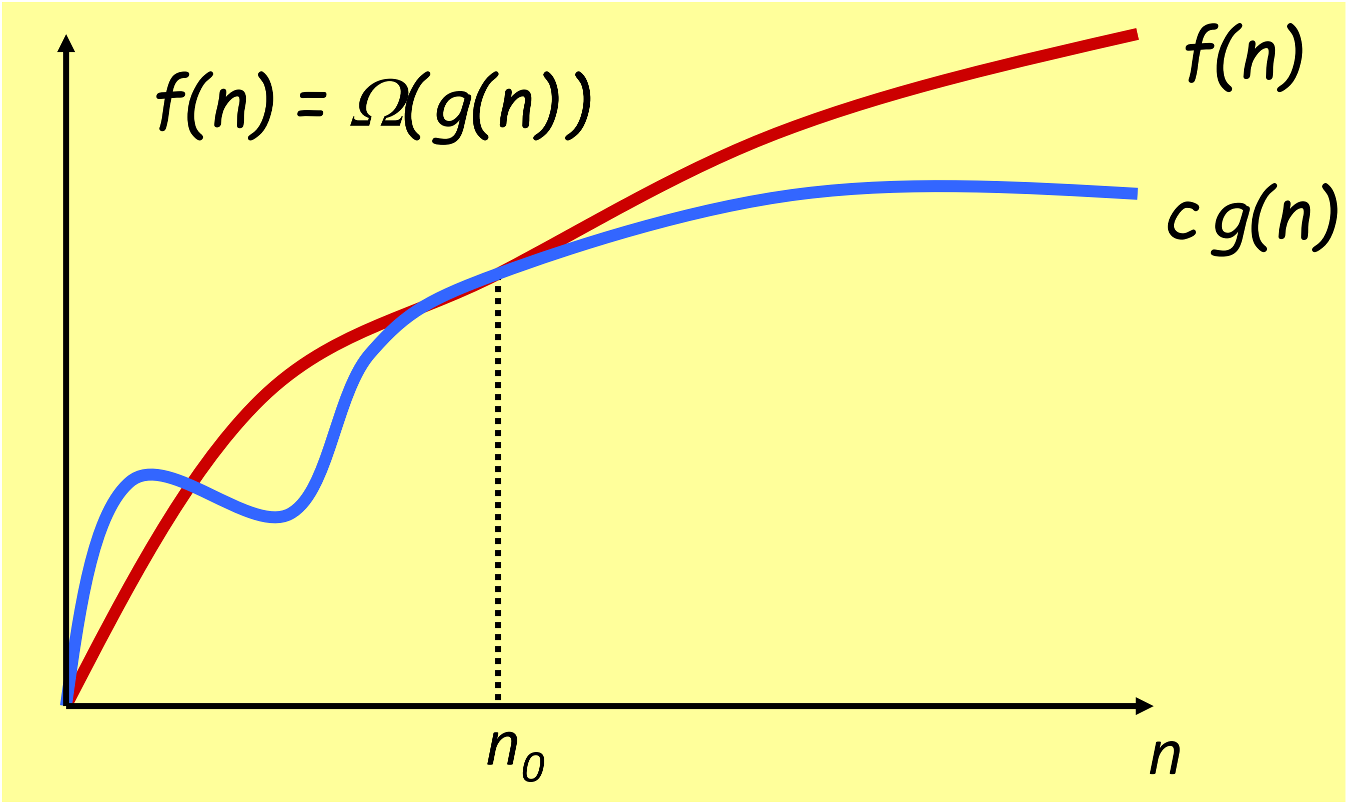
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \quad \Longrightarrow \quad f(n) = O(g(n))$$

$$f(n) = O(g(n)) \quad \not\Longrightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) = O(g(n)) \quad \Longrightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} \quad \text{(se esiste)} \quad < \infty$$

Notazione asintotica Ω

$f(n) = \Omega(g(n))$ se $\exists$ due costanti $c>0$ e $n_0 \geq 0$ tali che $f(n) \geq c\, g(n) \geq 0$ per ogni $n \geq n_0$



Esempi:

Sia $f(n) = 2n^2 - 3n$, allora

- $f(n) = \Omega(n)$ $(c=1,\ n_0=2)$
- $f(n) = \Omega(n^2)$ $(c=1,\ n_0=3)$
- $f(n) \neq \Omega(n^3)$

Notazione asintotica Ω

$\Omega(g(n))=\{f(n) \mid \exists\ c>0 \text{ e } n_0\geq 0$ tali che
$0 \leq c\, g(n) \leq f(n)$ per ogni $n \geq n_0\}$

- La scrittura:

  $2n^2+4= \Omega(n)$

- è un abuso di notazine per:

  $2n^2+4 \in \Omega(n)$

Notare:

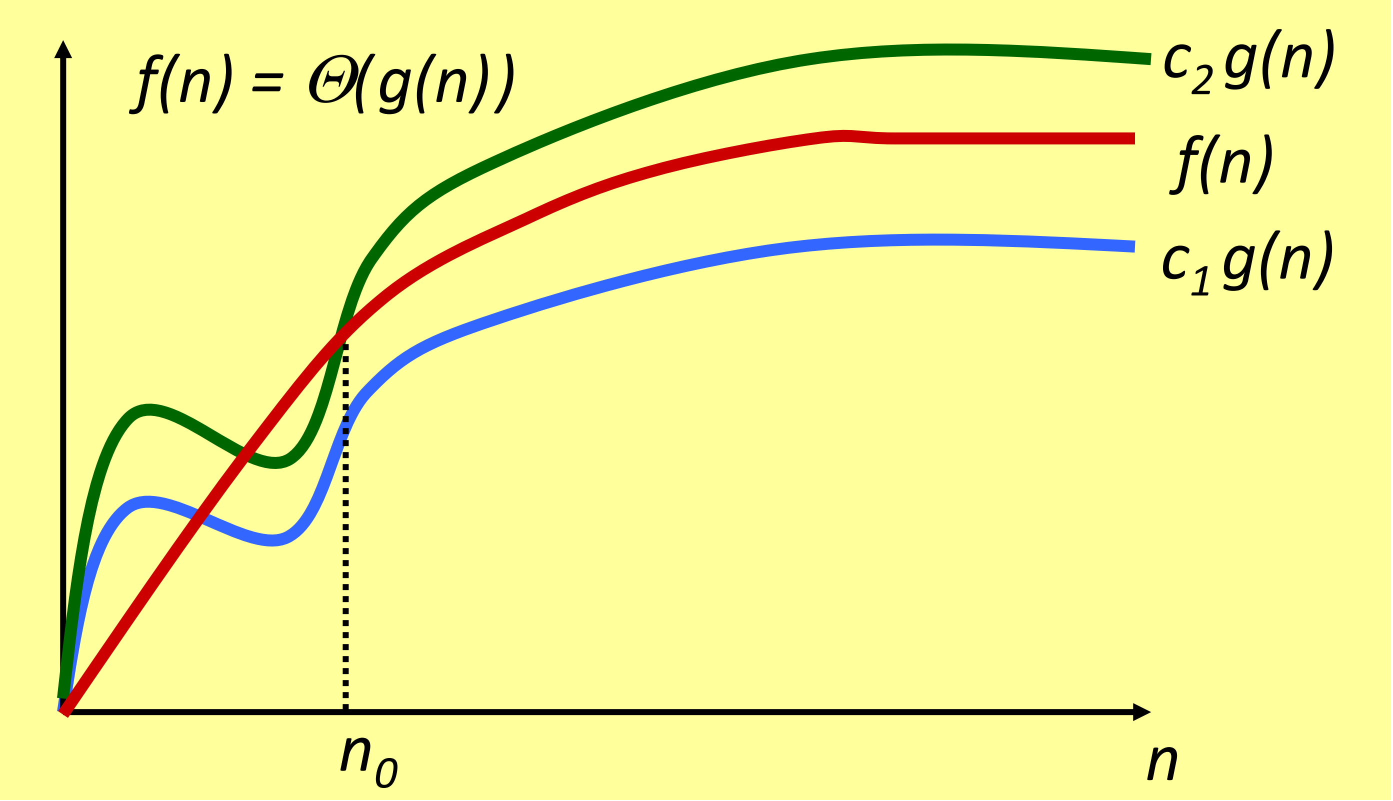
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \quad \Longrightarrow \quad f(n) = \Omega(g(n))$$

$$f(n) = \Omega(g(n)) \quad \not\Longrightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

$$f(n) = \Omega(g(n)) \quad \Longrightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} \quad \text{(se esiste)} \quad > 0$$

Notazione asintotica Θ

$f(n) = \Theta(\ g(n)\ )$ se $\exists$ tre costanti $c_1, c_2 > 0$ e $n_0 \geq 0$ tali che $c_1\ g(n) \leq f(n) \leq c_2\ g(n)$ per ogni $n \geq n_0$



Esempi:

Sia $f(n) = 2n^2 - 3n$, allora

- $f(n) = \Theta(n^2)$ $(c_1=1, c_2=2, n_0=3)$
- $f(n) \neq \Theta(n)$
- $f(n) \neq \Theta(n^3)$

Notazione asintotica Θ

$\Theta(g(n))=\{f(n) \mid \exists\ c_1,c_2>0$ e $n_0\geq 0$ tali che
$c_1\, g(n) \leq f(n) \leq c_2\, f(n)$ per ogni $n\geq n_0\}$

- La scrittura:

  $2n^2+4= \Theta(n^2)$

- è un abuso di notazine per:

  $2n^2+4 \in \Theta(n^2)$

Notare che:

$$f(n) = \Theta(g(n)) \quad \Rightarrow \quad f(n) = O(g(n))$$

$$f(n) = O(g(n)) \quad \not\Rightarrow \quad f(n) = \Theta(g(n))$$

$$f(n) = \Theta(g(n)) \quad \Rightarrow \quad f(n) = \Omega(g(n))$$

$$f(n) = \Omega(g(n)) \quad \not\Rightarrow \quad f(n) = \Theta(g(n))$$

$$\boxed{f(n) = \Theta(g(n)) \quad \Leftrightarrow \quad f(n) = \Omega(g(n)) \quad e \quad f(n) = O(g(n))}$$

Notazione asintotica o

Data una funzione $g(n)$: $N \rightarrow R$, si denota con $o(g(n))$ l' insieme delle funzioni $f(n)$: $N \rightarrow R$:

$$o(g(n)) = \{f(n) : \forall c > 0, \exists n_0 \text{ tale che } \forall n \geq n_0 \quad 0 \leq f(n) < c\, g(n) \}$$

Notare:

$$o(g(n)) \subset O(g(n))$$

definizione alternativa:

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

Notazione asintotica ω

Data una funzione $g(n)$: $N \rightarrow R$, si denota con $\omega(g(n))$ l' insieme delle funzioni $f(n)$:

$\omega(g(n)) = \{f(n) : \forall\ c > 0,\ \exists\ n_0$ tale che
$\forall\ n \geq n_0 \quad 0 \leq c\ g(n) < f(n)\ \}$

Notare:

$$\omega(g(n)) \quad \subset \quad \Omega(g(n))$$

definizione alternativa:

$$f(n) = \omega(g(n)) \quad \Leftrightarrow \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

Riassumendo ……

$$f(n)=\Theta(g(n)) \quad \Leftrightarrow \quad 0<c_1\leq \frac{f(n)}{g(n)}\leq c_2<\infty \quad \text{asintoticamente}$$

$$f(n)=O(g(n)) \quad \Leftrightarrow \quad \frac{f(n)}{g(n)}\leq c_2<\infty \quad \text{asintoticamente}$$

$$f(n)=\Omega(g(n)) \quad \Leftrightarrow \quad 0<c_1\leq \frac{f(n)}{g(n)} \quad \text{asintoticamente}$$

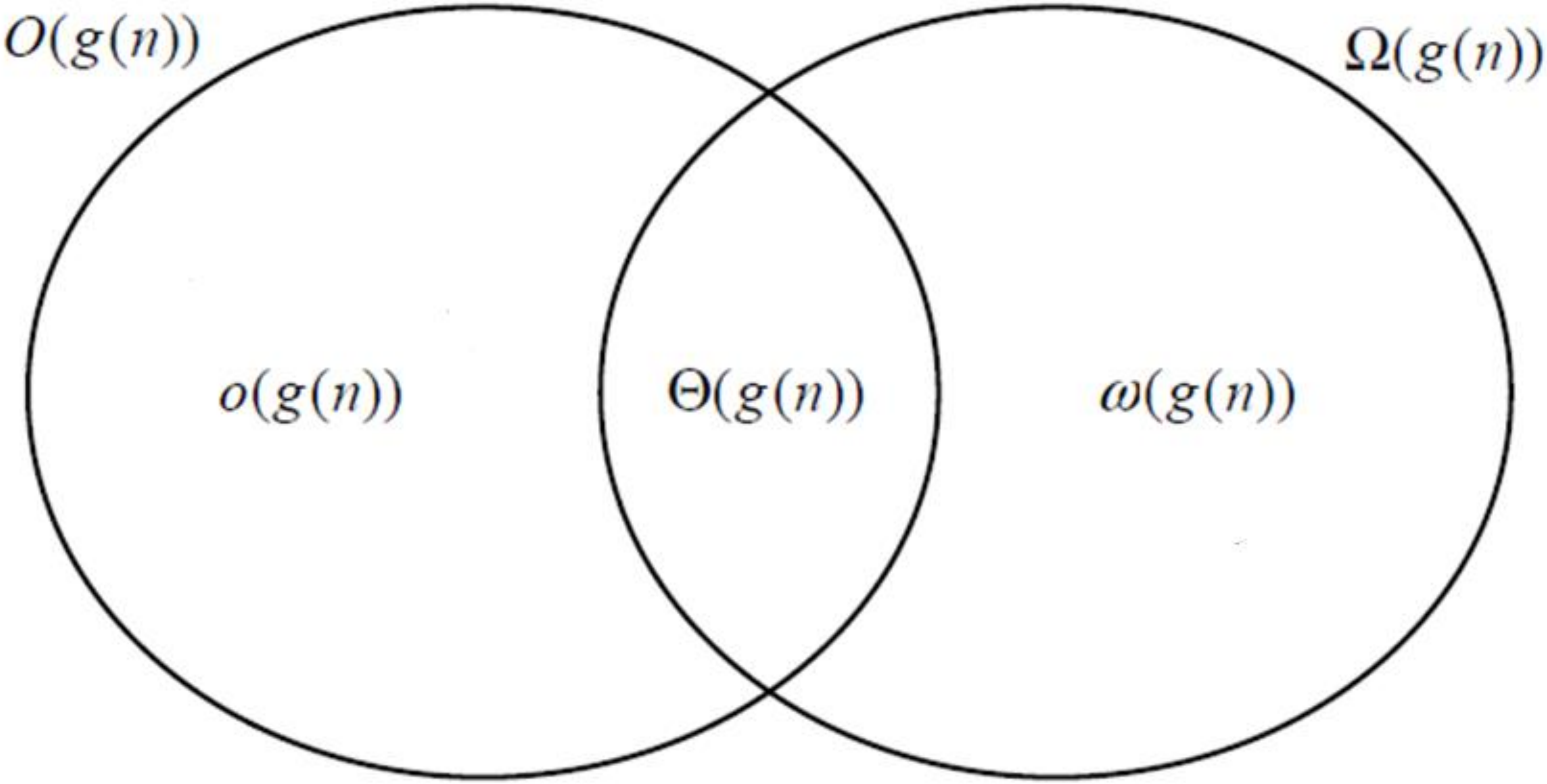
$$f(n)=o(g(n)) \quad \Leftrightarrow \quad \lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

$$f(n)=\omega(g(n)) \quad \Leftrightarrow \quad \lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

Analogie

O	Ω	Θ	o	ω
$\leq$	$\geq$	$=$	$<$	$>$

Graficamente



Proprietà della notazione asintotica

Transitività

$$f(n) = \Theta(g(n)) \quad e \quad g(n) = \Theta(h(n)) \quad \Longrightarrow \quad f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \quad e \quad g(n) = O(h(n)) \quad \Longrightarrow \quad f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \quad e \quad g(n) = \Omega(h(n)) \quad \Longrightarrow \quad f(n) = \Omega(h(n))$$
$$f(n) = o(g(n)) \quad e \quad g(n) = o(h(n)) \quad \Longrightarrow \quad f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \quad e \quad g(n) = \omega(h(n)) \quad \Longrightarrow \quad f(n) = \omega(h(n))$$

Riflessività

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

Simmetria

$$f(n) = \Theta(g(n)) \quad \Longleftrightarrow \quad g(n) = \Theta(f(n))$$

Simmetria trasposta

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

Ancora una convenzione

Un insieme in una formula rappresenta un'anonima funzione dell'insieme.

Esempio 1:

$$f(n)=n^3 + O(n^2)$$

sta per: c'è una funzione $h(n) \in O(n^2)$ tale che

$$f(n)=n^3 + h(n)$$

Esempio 2:

$$n^2 + O(n) = O(n^2)$$

sta per: per ogni funzione $f(n) \in O(n)$,
c'è una funzione $h(n) \in O(n^2)$ tale che

$$n^2 + f(n) = h(n)$$

…una semplice ma utile proprietà per capire la velocità di una funzione

Se $\lim_{n\to\infty} f(n)/g(n) = c > 0$

allora $f(n)=\Theta(g(n))$

Infatti:

$$c/2 < f(n)/g(n) < 2c$$

per n suff. grande

Esempio:

Se $T(n) = a_d n^d + a_{d-1} n^{d-1} + \dots + a_0$ è un polinomio di grado d (con $a_d>0$), allora $T(n) = \Theta(n^d)$

Infatti:

$T(n) / n^d = a_d + a_{d-1} n^{-1} + \dots + a_0 n^{-d}$

che tende a a_d quando $n \to\infty$:

Polinomi ……

$P(n) = a_d\, n^d + a_{d-1}\, n^{d-1} + \dots + a_0$

$a_d > 0$

$\longrightarrow$

$P(n) = \Theta(n^d)$

$P(n) = O(n^d)$

$P(n) = \Omega(n^d)$

Esponenziali ……

$f(n) = a^n$

$a > 1$

$\longrightarrow$

$a^n = \omega(n^d)$

$a^n = \Omega(n^d)$

$$\lim_{n \to \infty} \frac{a^n}{n^d} = \infty$$

Logaritmi ……

$f(n) = \log_b(n) \quad b>1$

$\longrightarrow$

$[\log_b(n)]^c = o(n^d)$

$[\log_b(n)]^c = O(n^d)$

$$\lim_{n \to \infty} \frac{[\log_b(n)]^c}{n^d} = 0, \forall c, d > 0$$

Fattoriali ……

$f(n) = n! = n*(n-1)*\dots\dots*2*1$

$\longrightarrow$

$n! = o(n^n)$

$n! = \omega(a^n)$

velocità asintotica di funzioni composte

Velocità delle funzioni composte

date $f(n)$ e $g(n)$,
la velocità ad andare a infinto della funzione $f(n)+g(n)$
è la velocità della più veloce fra $f(n)$ e $g(n)$

Esempi:

$$n^3+n = \Theta(n^3)$$

$$n+\log^{10} n = \Theta(n)$$

Velocità delle funzioni composte

date $f(n)$ e $g(n)$,

la velocità ad andare a infinito della funzione $f(n)\,g(n)$
e la velocità di $f(n)$ "più" la velocità di $g(n)$

la velocità ad andare a infinito della funzione $f(n)/g(n)$
e la velocità di $f(n)$ "meno" la velocità di $g(n)$

Esempio:

$$\frac{n^3 \log n + \sqrt{n}\, \log^3 n}{n^2 + 1} = \Theta(n \log n)$$

Usare la notazione asintotica nelle analisi

Analisi complessità fibonacci3: un Upper Bound

algoritmo `fibonacci3`*(intero n)* $\rightarrow$ *intero*

1 *sia Fib un array di n interi*
2 *Fib[1]* $\leftarrow$ *Fib[2]* $\leftarrow$ *1*
3 **for** i = 3 **to** n **do**
4   *Fib[i]* $\leftarrow$ *Fib[i-1] + Fib[i-2]*
5 **return** *Fib[n]*

T(n): complessità computazionale nel caso peggiore con input n

c_j: #passi elementari eseguiti su una RAM quando è esguita la linea di codice j

- linea 1, 2 e 5 eseguite una vota
- linee 3 e 4: eseguite al più n volte

$$T(n) \leq c_1+c_2+c_5+(c_3+c_4)n = \Theta(n)$$

$$T(n)=O(n)$$

Analisi complessità fibonacci3: un Lower Bound

algoritmo `fibonacci3`*(intero n)* $\rightarrow$ *intero*

```
1   sia Fib un array di n interi
2   Fib[1] ← Fib[2] ← 1
3   for i = 3 to n do
4       Fib[i] ← Fib[i-1] + Fib[i-2]
5   return Fib[n]
```

Nota: poiché ogni istruzione di alto livello esegue un #costante di passi elementari posso contare # di istruzioni

$T(n)$: complessità computazionale nel caso peggiore con input n

c_j: #passi elementari eseguiti su una RAM quando è esguita la linea di codice j

la linea 4 è eseguita almeno n-3 volte

$T(n) \geq c_4(n-3)= c_4 n -3c_4 =\Theta(n)$

$T(n)=\Omega(n)$

$$\boxed{T(n)=\Theta(n)}$$

Notazione asintotica: perché è una grande idea

- misura indipendente dall'implementazione dell'algoritmo e dalla macchina reale su cui è eseguito
- il "dettagli" nascosti (costanti moltiplicative e termini di ordine inferiore) sono poco rilevanti quando n è grande per funzioni asintoticamente diverse (guarda tabella)
- analisi dettagliata del numero di passi realmente eseguiti sarebbe difficile, noiosa e non direbbe molto di più (come si possono conoscere per esempio i costi reali di un'istruzione di alto livello?)
- si è visto che descrive bene in pratica la velocità degli algoritmi