ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

A.A. 2022/23

NOTAZIONE ASINTOTICA

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Notazione Asintotica

La notazione asintotica ci permette di esprimere limiti superiori e inferiori al costo computazionale di algoritmi e problemi

A COSA CI SERVE?

- Stimare il tempo massimo necessario per l'esecuzione di un algoritmo
- Conoscere il tipo di crescita del tempo di esecuzione al crescere della dimensione delle istanze di input.
- Confrontare le prestazioni di algoritmi diversi per lo stesso problema.
- Stabilire la difficolta' dei problemi.

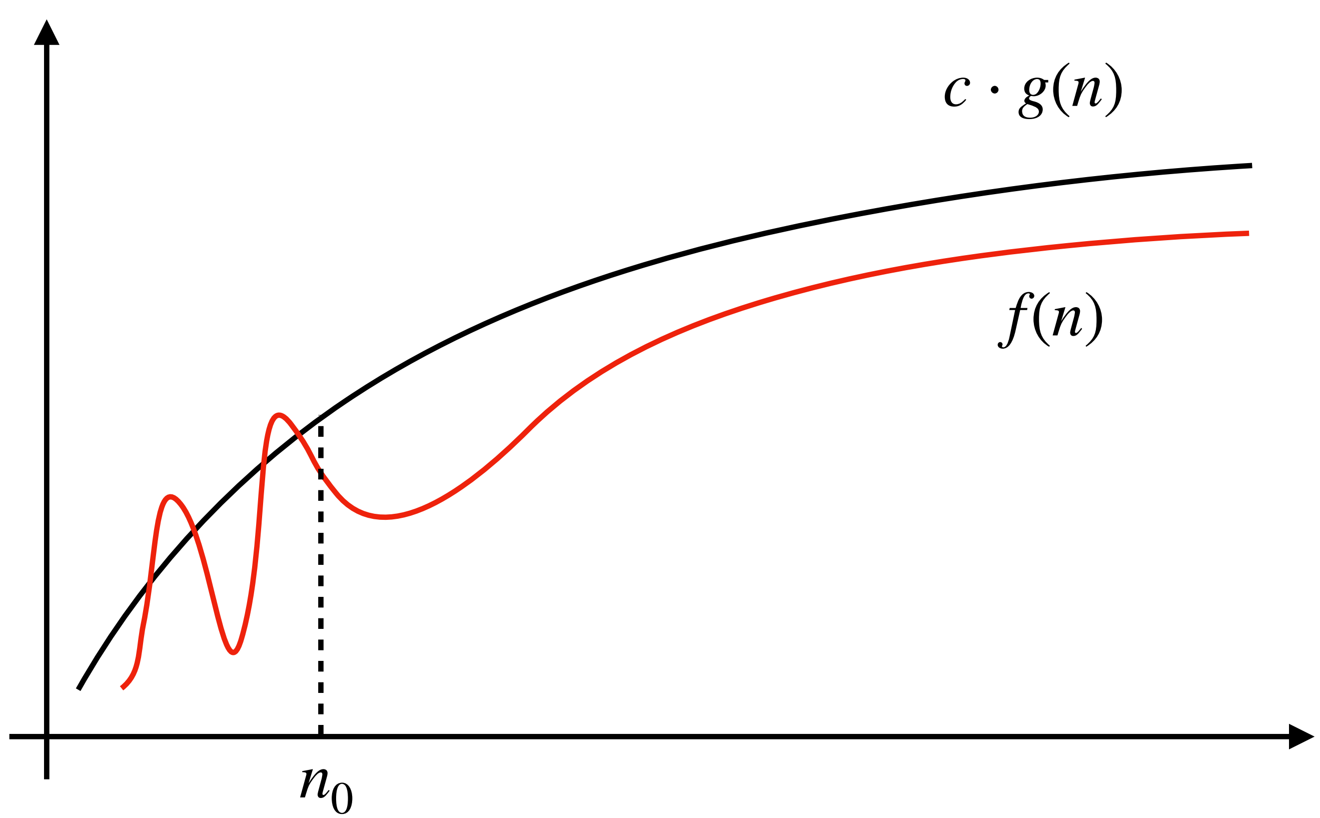
Notazione "O-grande" $O(\cdot)$

DEFINIZIONE

Date due funzioni $f : \mathbb{N} \to \mathbb{R}_+$ e $g : \mathbb{N} \to \mathbb{R}_+$ diremo che $f(n) \in O(g(n))$

SE E SOLO SE

esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che $f(n) \leq cg(n)$ per ogni $n \geq n_0$.



Intuizione

A partire da $n = n_0$ la funzione $f(n)$ non sta MAI sopra la funzione $cg(n)$

UNIMORE

Notazione "O-grande" $O(\cdot)$

ESEMPIO

Date le funzioni $f(n) = 2n^2 + 3n + 6$ e $g(n) = n^2$,

dimostriamo che $2n^2 + 3n + 6 \in O(n^2)$

Ovvero, facciamo vedere che esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, vale che

$$2n^2 + 3n + 6 \leq cn^2$$

Osserviamo che

$$2n^2 + 3n + 6 \leq 2n^2 + 3n^2 + 6 \leq 2n^2 + 3n^2 + n^2 = 6\ n^2$$

perché $n \leq n^2 \forall n \geq 0$

assumendo che $n^2 \geq 6$, ovvero che $n \geq 3$

Notazione "O-grande" $O(\cdot)$

ESEMPIO

Date le funzioni $f(n) = 2n^2 + 3n + 6$ e $g(n) = n^2$,

dimostriamo che $2n^2 + 3n + 6 \in O(n^2)$

Ovvero, facciamo vedere che esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, vale che

$$2n^2 + 3n + 6 \leq cn^2$$

Osserviamo che

$$2n^2 + 3n + 6 \leq 2n^2 + 3n^2 + 6 \leq 2n^2 + 3n^2 + n^2 = 6n^2$$

assumendo che $n^2 \geq 6$, ovvero che $n \geq 3$

Notazione "O-grande" $O(\cdot)$

ESEMPIO

Date le funzioni $f(n) = 2n^2 + 3n + 6$ e $g(n) = n^2$,

dimostriamo che $2n^2 + 3n + 6 \in O(n^2)$

Ovvero, facciamo vedere che esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, vale che

$$2n^2 + 3n + 6 \leq cn^2$$

Osserviamo che

$$2n^2 + 3n + 6 \leq 2n^2 + 3n^2 + 6 \leq 2n^2 + 3n^2 + n^2 = 6n^2$$

$n_0 = 3$

assumendo che $n^2 \geq 6$, ovvero che $n \geq 3$

Notazione "O-grande" $O(\cdot)$

ESEMPIO

Date le funzioni $f(n) = 2n^2 + 3n + 6$ e $g(n) = n^2$,

dimostriamo che $2n^2 + 3n + 6 \in O(n^2)$

Ovvero, facciamo vedere che esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, vale che

$$2n^2 + 3n + 6 \leq cn^2$$

Quindi, avendo trovato le due costanti

$c = 6$ e $n_0 = 3$

abbiamo dimostrato che $f(n) \in O(n^2)$

OSSERVAZIONE: le costanti non sono necessariamente UNICHE, ma trovarne una coppia e' sufficiente.

Notazione "O-grande" $O(\cdot)$

PROPOSIZIONE

Per ogni costante $k \geq 1$ abbiamo che

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \in O(n^k)$$

Ovvero, un polinomio di grado massimo k e' $O(n^k)$.

Notazione "O-grande" $O(\cdot)$

PROPOSIZIONE

Per ogni costante $k \geq 1$ abbiamo che

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 = O(n^k)$$

Ovvero, un polinomio di grado massimo k e' $O(n^k)$.

INTUIZIONE

Ogni polinomio di grado massimo k cresce come il polinomio n^k (a meno di una costante moltiplicativa)

ESEMPIO

$$\frac{7n^{10} + 8n^5 + 5}{n^{10}} = 7 + \frac{8}{n^5} + \frac{5}{n^{10}}$$

Costante

Trascurabili al crescere di n

Dimostrazioni

PROPOSIZIONE

Per ogni costante $k \geq 1$ abbiamo che

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \in O(n^k)$$

DIMOSTRAZIONE

$\forall\ n \geq 0$

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$$

$$\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \ldots + |a_1| n + |a_0|$$

per n>0 $\leq |a_k| n^k + |a_{k-1}| n^k + \ldots + |a_1| n^k + |a_0| n^k$

$$= \underbrace{(|a_k| + |a_{k-1}| + \ldots + |a_1| + |a_0|)}_{c} n^k$$

$$= cn^2$$

Notazione "O-grande" $O(\cdot)$

Cosa possiamo dire per altre funzioni?

PROPOSIZIONE

$$\log n \in O(n)$$

Dobbiamo fare vedere che esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, vale che

$$\log n \leq cn$$

ATTENZIONE

In questo corso quando scriveremo $\log n$ intendiamo il logaritmo in BASE 2 (se non specificato diversamente), quindi $\log_2 n$

UNIMORE

Notazione "O-grande" $O(\cdot)$

PROPOSIZIONE

$$\log n \in O(n)$$

Dobbiamo fare vedere che esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, vale che

$$\log n \leq cn$$

DIMOSTRAZIONE PER INDUZIONE: con $c = 1$ e $n_0 = 1$ $\Rightarrow$ $\boxed{\log n \leq n, \forall n \geq 1}$

CASO BASE: $n = 1$ $\log n = \log 1 = 0 \leq 1 = c \cdot n$

Notazione "O-grande" $O(\cdot)$

PROPOSIZIONE

$$\log n \in O(n)$$

Dobbiamo fare vedere che esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, vale che

$$\log n \leq cn$$

DIMOSTRAZIONE PER INDUZIONE: con $c = 1$ e $n_0 = 1$ ⇨ $\boxed{\log n \leq n, \forall n \geq 1}$

Passo induttivo

IPOTESI INDUTTIVA: $\log n \leq n$

TESI DA DIMOSTRARE:

$$\log(n+1) \leq (n+1)$$

DIMOSTRAZIONE:

$$\begin{aligned}\log(n+1) &\leq \log(n+n)\\ &= \log(2n)\\ &= \log 2 + \log n\\ &\leq 1 + n \quad \text{(per l'ipotesi induttiva)}\end{aligned}$$

Notazione "O-grande" $O(\cdot)$

Si possono dimostrare altri risultati che sono riassunti nella seguente tabella

Espressione O	nome
$O(1)$	(sublineare) costante
$O(\log\log n)$	(sublineare) log log
$O(\log n)$	(sublineare) logaritmico
$O(\sqrt[c]{n}),\ c > 1$	sublineare
$O(n)$	lineare
$O(n \log n)$	$n \log n$
$O(n^2)$	quadratico
$O(n^3)$	cubico
$O(n^k)\ (k \geq 1)$	polinomiale
$O(a^n)\ (a > 1)$	esponenziale
$O(n!)$	fattoriale

perfetto buono accettabile inaccettabile

INTERPRETAZIONE

Una funzione
$f(n)$ in ALTO
e' O-grande
di una qualsiasi funzione
$g(n)$ più in BASSO

ESEMPI

$\sqrt[2]{n} \in O(n)$

$n^{10} \in O(2^n)$

$7^n \in O(n!)$

Notazione "O-grande" $O(\cdot)$

PROPRIETÀ UTILI

per determinare l'ordine di grandezza delle funzioni

PROPOSIZIONE

Per ogni costante $a > 0$ vale che

SE $f(n) \in O(g(n))$ ALLORA $a \cdot f(n) \in O(g(n))$

ESEMPIO

$$\log n \in O(n) \Rightarrow 7 \log n \in O(n)$$

Notazione "O-grande" $O(\cdot)$

PROPRIETÀ UTILI

per determinare l'ordine di grandezza delle funzioni

PROPOSIZIONE (somma di funzioni)

SE $d(n) \in O(f(n))$, e $e(n) \in O(g(n))$ ALLORA

$$d(n) + e(n) \in O(\max\{f(n), g(n)\})$$

ESEMPIO

$$\log n \in O(n), \sqrt{n} \in O(n) \Rightarrow \log n + \sqrt{n} \in O(n)$$

$$\log n \in O(n), n \in O(n^2) \Rightarrow \log n + n^2 \in O(n^2)$$

Notazione "O-grande" $O(\cdot)$

PROPRIETÀ UTILI

per determinare l'ordine di grandezza delle funzioni

PROPOSIZIONE (prodotto di funzioni)

SE $d(n) \in O(f(n))$, e $e(n) \in O(g(n))$ ALLORA

$$d(n) \cdot e(n) \in O(f(n) \cdot g(n))$$

ESEMPIO

$$\log n \in O(\sqrt{n}), \sqrt{n} \in O(\sqrt{n}) \Rightarrow \sqrt{n} \log n \in O(n)$$

Notazione "O-grande" $O(\cdot)$

PROPRIETÀ UTILI

per determinare l'ordine di grandezza delle funzioni

PROPOSIZIONE (transitività)

SE $d(n) \in O(f(n))$, e $f(n) \in O(g(n))$ ALLORA

$$d(n) \in O(g(n))$$

ESEMPIO

$$\log n \in O(\sqrt{n}), \sqrt{n} \in O(n) \Rightarrow \log n \in O(n)$$

Notazione "O-grande" $O(\cdot)$

PROPRIETÀ UTILI

per determinare l'ordine di grandezza delle funzioni

PROPOSIZIONE

Per ogni coppia di costanti $a > 1$ e $x > 0$ vale che

$$n^x \in O(a^n)$$

ESEMPIO

$$n^{100} = O(2^n)$$

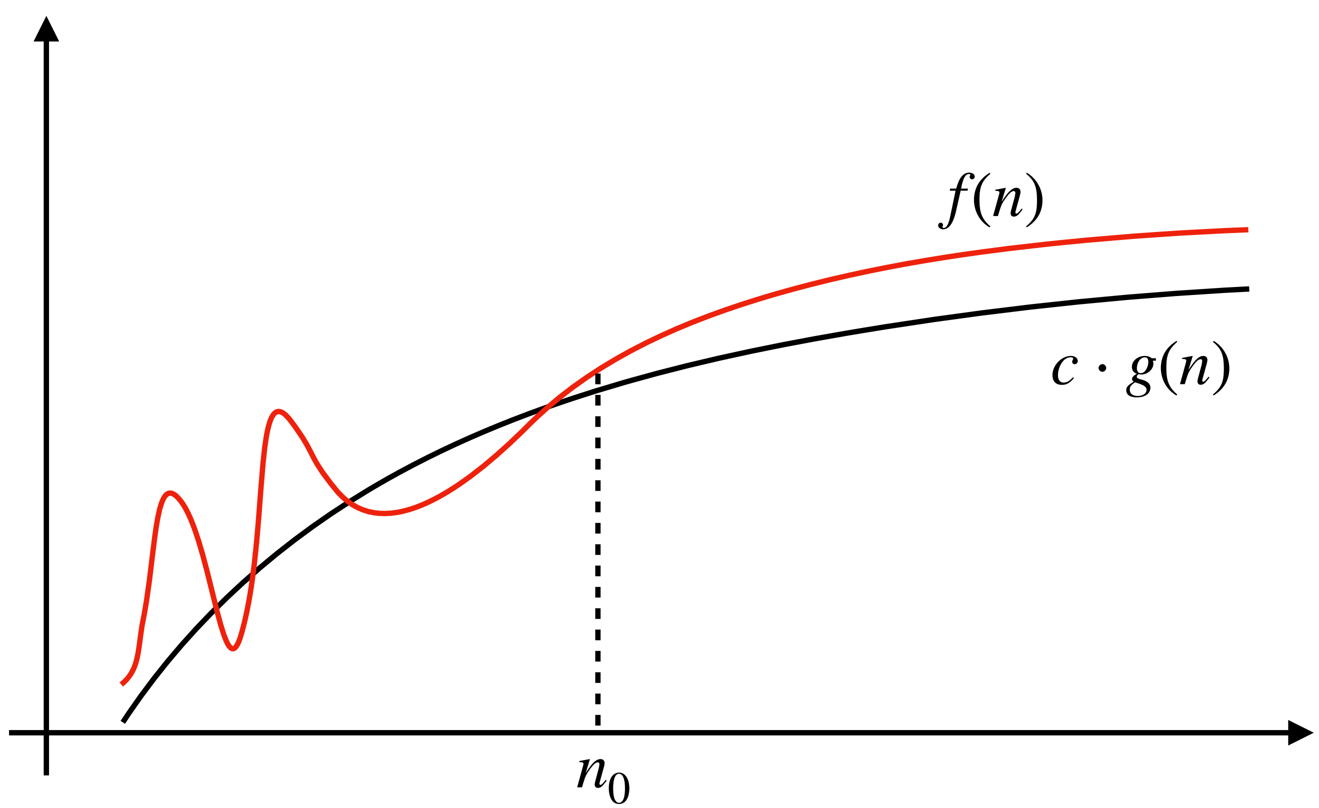
Notazione "Omega" $\Omega(\cdot)$

DEFINIZIONE

Date due funzioni $f : \mathbb{N} \to \mathbb{R}_+$ e $g : \mathbb{N} \to \mathbb{R}_+$ diremo che $f(n) \in \Omega(g(n))$

SE E SOLO SE

esistono due costanti $c > 0$ e $n_0 \in \mathbb{N}$ tali che $f(n) \geq cg(n)$ per ogni $n \geq n_0$.



Intuizione

A partire da $n = n_0$ la funzione $f(n)$ non sta MAI sotto la funzione $cg(n)$

Notazione "Omega" $\Omega(\cdot)$

PROPOSIZIONE

$$f(n) \in \Omega(g(n)) \Leftrightarrow g(n) \in O(f(n))$$

DIMOSTRAZIONE

$f(n) \in \Omega(g(n)) \Leftrightarrow \exists c > 0, n_0 \geq : f(n) \geq cg(n), \forall n \geq n_0$ per definizione di $\Omega(\cdot)$

$\Leftrightarrow \exists\, c' = 1/c > 0, n_0 \geq 0 : g(n) \leq \dfrac{1}{c} f(n), \forall n \geq n_0$ dividendo ambo i lati per c

$\Leftrightarrow g(n) \in O(f(n))$ per definizione di $O(\cdot)$

Notazione "Omega" $\Omega(\cdot)$

La proposizione precedence permette di enunciare i risultati
che sono riassunti nella seguente tabella

Espressione O	nome
$\Omega(1)$	(sublineare) costante
$\Omega(\log\log n)$	(sublineare) log log
$\Omega(\log n)$	(sublineare) logaritmico
$\Omega(\sqrt[c]{n}),\ c > 1$	sublineare
$\Omega(n)$	lineare
$\Omega(n \log n)$	$n \log n$
$\Omega(n^2)$	quadratico
$\Omega(n^3)$	cubico
$\Omega(n^k)\ (k \geq 1)$	polinomiale
$\Omega(a^n)\ (a > 1)$	esponenziale
$\Omega(n!)$	fattoriale

perfetto buono accettabile inaccettabile

INTERPRETAZIONE

Una funzione
$f(n)$ in BASSO
e' Omega
di una qualsiasi funzione
$g(n)$ più in ALTO

ESEMPI

$$\sqrt{n} \in \Omega(\log n)$$
$$n^{k+1} = \Omega(n^k)$$
$$n! = \Omega(2^n)$$

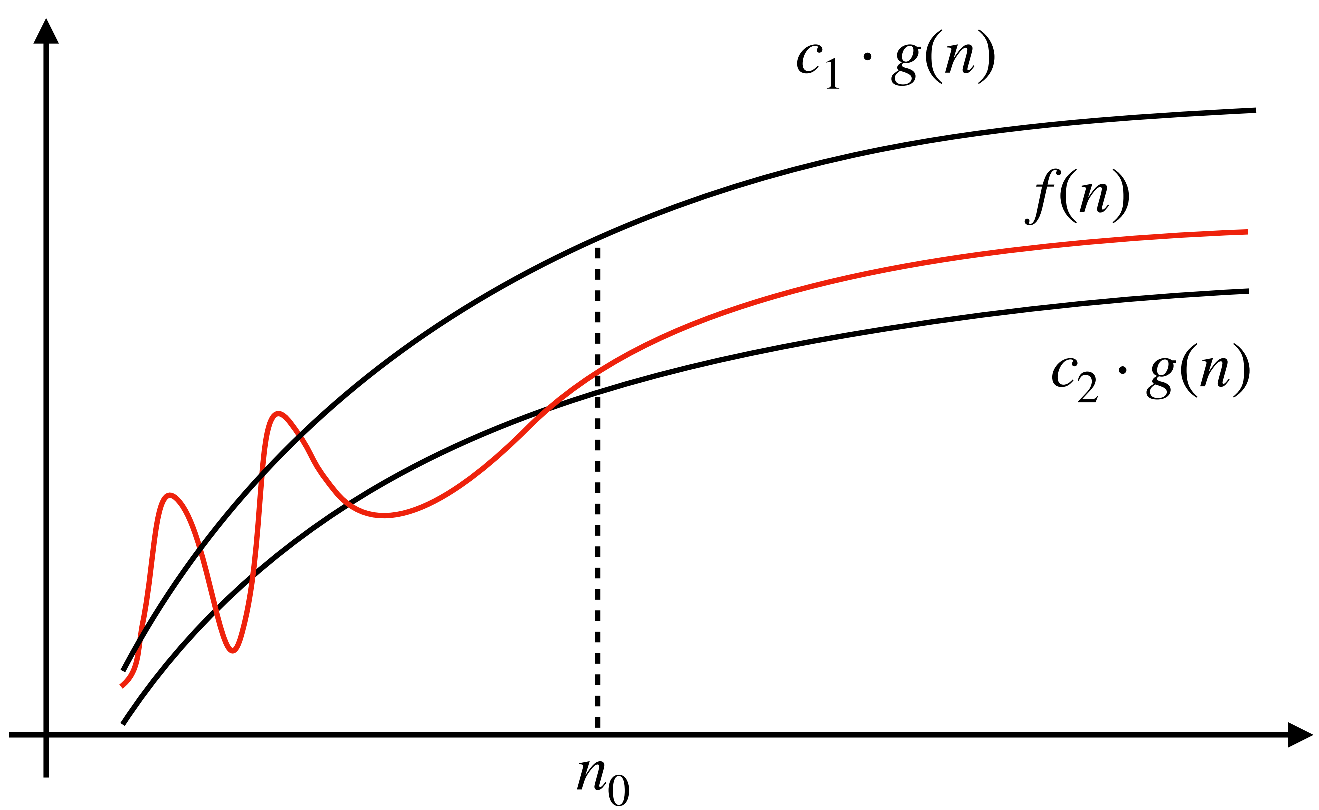
Notazione "Teta" $\Theta(\cdot)$

DEFINIZIONE

Date due funzioni $f : \mathbb{N} \to \mathbb{R}_+$ e $g : \mathbb{N} \to \mathbb{R}_+$ diremo che $f(n) \in \Theta(g(n))$

SE E SOLO SE

esistono tre costanti $c_1, c_2 > 0$ e $n_0 \in \mathbb{N}$ tali che $c_1 g(n) \leq f(n) \leq c_2 g(n)$ per ogni $n \geq n_0$.



Intuizione

A partire da $n = n_0$

la funzione $f(n)$

non sta

MAI sopra

la funzione $c_1 g(n)$

e

MAI sotto

la funzione $c_2 g(n)$

Notazione "Teta" $\Theta(\cdot)$

ESEMPIO

Per dimostrare che $4n^2 + \log n \in \Theta(n^2)$

dobbiamo fare vedere che

$4n^2 + \log n \in \Omega(n^2)$ e $4n^2 + \log n \in O(n^2)$

SI perche'

$$4n^2 + \log n \geq 4n^2$$

$$\forall n \geq 1 = n_0, c = 4$$

SI perche'

$$4n^2 + \log n \leq 4n^2 + n \leq 5n^2$$

$$\forall n \geq 1 = n_0, c = 5$$

$$c_1 = 4, c_2 = 5, n_0 = 1$$

Costo computazionale

Siamo interessati a contare il numero di operazioni elementari
(assegnamenti e op. logico-aritmetiche)
eseguite durante l'esecuzione di un algoritmo

ANALISI DEL CASO PEGGIORE (**worst case**)

- La più importante
- E' un limite superiore (**upper bound**) per una qualsiasi istanza di input
- Si cerca la funzione più piccola possibile

ANALISI DEL CASO MEDIO (**average case**)

- Spesso difficile: qual e' il caso medio?
- Distribuzione delle istanze di input

ANALISI DEL CASO MIGLIORE (**best case**)

- Poco significativa
- E' un limite superiore (**upper bound**) solo per una istanza di input (o un sottoinsieme proprio delle istanze)

Costo computazionale

Contiamo gli assegnamenti e le operazioni logico-aritmetiche

ESEMPIO

$T(n)$ = numero massimo di operazioni eseguito dalla porzione di codice, in funzione di n

Trovare $f(n)$ più piccola possibile tale che $T(n) \in O(f(n))$

```
for i:=0 to n-1
  c := c+1
  d := d+1
c := c+d
```

Costo computazionale

Contiamo gli assegnamenti e le operazioni logico-aritmetiche

ESEMPIO

$T(n) =$ numero massimo di operazioni eseguito dalla porzione di codice, in funzione di n

Trovare $f(n)$ più piccola possibile tale che $T(n) \in O(f(n))$

```
for i=0 to n-1
  c := c+1
  d := d+1
c := c+d
```

istruzioni nel corpo del ciclo for eseguite n volte

4 operazioni

Tot = 4n

2 operazioni

$T(n) = 4n + 2 \in O(n)$ LINEARE

UNIMORE

Costo computazionale

ESEMPIO

```
for i=0 to n-1
  for j=i to n-1
    x := x+1
  y := y+1
```

Costo computazionale

ESEMPIO

```
for i=0 to n-1
  for j=i to n-1
    x := x+1
  y := y+1
```

Fissato `i`, le istruzioni nel corpo del for sono eseguite $n - i$ volte

2 operazioni

Tot = $2 \cdot (n - i)$

Il ciclo viene eseguito la prima volta quando `j = i`
e l'ultima volta quando `j = n-1` (compresi gli estremi).

Quindi viene ripetuto $(n-1) - i + 1 = n - 1$

Costo computazionale

ESEMPIO

```
for i=0 to n-1
  for j=i to n-1
    x := x+1
  y := y+1
```

2 operazioni

Fissato `i`, le istruzioni nel corpo del for sono eseguite $n - i$ volte $\Rightarrow$ $\text{Tot} = 2 \cdot (n - i)$

Prima esecuzione del ciclo interno (per `i=0`): $2 \cdot n$ operazioni +

Seconda esecuzione del ciclo interno (per `i=1`): $2 \cdot (n-1)$ operazioni +

Terza esecuzione del ciclo interno (per `i=2`): $2 \cdot (n-2)$ operazioni +

Quarta esecuzione del ciclo interno (per `i=3`): $2 \cdot (n-3)$ operazioni +

……. …

(n-1)-esima esecuzione del ciclo interno (per `i= n-2`): $2 \cdot (n-(n-2)) = 2 \cdot 2$ operazioni +

n-esima esecuzione del ciclo interno (per `i= n-1`): $2 \cdot (n-(n-1)) = 2 \cdot 1$ operazioni +

$$\Rightarrow \text{Tot} = 2 \cdot (n + (n-1) + (n-2) + (n-3) + \ldots + 2 + 1) = 2\sum_{i=1}^{n} i = 2\frac{n(n+1)}{2} = n(n+1)$$

Costo computazionale

ESEMPIO

```
for i=0 to n-1
  for j=i to n-1
    x := x+1
  y := y+1
```

TUTTE le iterazioni del ciclo interno, complessivamente eseguono $n(n+1)$ istruzioni

l'assegnamento a y viene eseguito 1 volta ad ogni iterazione del ciclo esterno, il ciclo esterno viene ripetuto esattamente n volte

$\Rightarrow$ $\text{Tot} = 2 \cdot n$

TOTALE

$$T(n) = n(n+1) + 2n = n^2 + 3n \in O(n^2)$$

QUADRATICO

UNIMORE

Costo computazionale

ESEMPIO

```
x := 0
y := 1
while y < n+1
  x := x+1
  y := 2y
return x
```

$O(1)$

$O(1)$

Come fare a capire quante volte viene seguito il ciclo while?

Prima del while —> y = 1

Costo computazionale

ESEMPIO

```
x := 0
y := 1
while y < n+1
  x := x+1
  y := 2y
return x
```

$O(1)$

$O(1)$

Come fare a capire quante volte viene seguito il ciclo while?

Prima del while —> y = 1

Dopo la prima iterazione del ciclo while —> y = 2

Dopo la seconda iterazione del ciclo while —> y = 2 * 2

Dopo la terza iterazione del ciclo while —> y = 2 * 2 * 2

....

Dopo l'i-esima iterazione del ciclo while —> $y = 2^i$

Dopo quanti cicli k abbiamo che $2^{k+1} = y \geq n + 1 > 2^k$?

…ovvero k e' stata l'ultima iterazione del ciclo? $\Rightarrow$ $2^k \leq n$, $k \leq \log n$ $\Rightarrow$ $T(n) \in O(\log n)$ LOGARITMICO

Costo computazionale

Qualche considerazione generale

SCANSIONE di UNA SEQUENZA DI ELEMENTI

- n elementi, ciascuno analizzato $f(n)$ volte
- per ogni elemento scansionato, max num operazioni $g(n)$

$\Rightarrow$ $T(n) \in O(n \cdot f(n) \cdot g(n))$

SE $f(n), g(n) \in O(1) \Rightarrow T(n) \in O(n)$ LINEARE

SCANSIONE di TUTTE LE COPPIE

- n elementi nell'insieme, n^2 coppie
- ogni coppia analizzata $f(n)$ volte
- per ogni coppia, max num operazioni $g(n)$

$\Rightarrow$ $T(n) \in O(n^2 \cdot f(n) \cdot g(n))$

SE $f(n), g(n) \in O(1) \Rightarrow T(n) \in O(n^2)$ QUDRATICO

Costo computazionale

Qualche considerazione generale

SCANSIONE di TUTTE LE TERNE

- n elementi nell'insieme, n^3 terne
- ogni terna analizzata $f(n)$ volte
- per ogni terna, max num operazioni $g(n)$

$\Rightarrow$ $T(n) \in O(n^3 \cdot f(n) \cdot g(n))$

SE $f(n), g(n) \in O(1) \Rightarrow T(n) \in O(n^3)$ CUBICO

SCANSIONE di TUTTI I SOTTOINSIEMI

- n elementi nell'insieme, 2^n sottoinsiemi diversi
- ogni sottoinsieme analizzato $f(n)$ volte
- per ogni sottoinsieme, max num operazioni $g(n)$

$\Rightarrow$ $T(n) \in O(2^n \cdot f(n) \cdot g(n))$

SE $f(n), g(n) \in O(1) \Rightarrow T(n) \in O(2^n)$ ESPONENZIALE

Costo computazionale

Qualche considerazione generale

SCANSIONE di TUTTE LE PERMUTAZIONI

- n elementi nell'insieme, $n!$ permutazioni diverse
- ogni permutazione analizzata $f(n)$ volte
- per ogni permutazione, max num operazioni $g(n)$

$\Rightarrow$ $T(n) \in O(n! \cdot f(n) \cdot g(n))$

SE $f(n), g(n) \in O(1) \Rightarrow T(n) \in O(n!)$ FATTORIALE

Algoritmi e Problemi

Costo computazionale (worst case) di ALGORITMI:

Per input di dimensione n

- $O(f(n))$: per NESSUN input l'algoritmo costa PIÙ di $f(n)$ (UPPER BOUND)
- $\Omega(f(n))$: ESISTE un input per cui l'algoritmo costa ALMENO $f(n)$ (LOWER BOUND)
- $\Theta(f(n))$: l'algoritmo ha costo $O(f(n))$ e $\Omega(f(n))$

Costo computazionale di PROBLEMI:

Per input di dimensione n

- $O(f(n))$: costo computazionale del MIGLIORE algoritmo noto per il problema (UPPER BOUND)
- $\Omega(f(n))$: dimostrazione che NESSUN algoritmo per il problema può costare meno di $\Omega(f(n))$ (LOWER BOUND) nel caso peggiore
- $\Theta(f(n))$: esiste un algoritmo ottimo per il problema (TIGHT BOUND), ovvero l'algoritmo ha un upper bound uguale al lower bound per il problema

Problemi facili e problemi difficili

In base a risultati sul costo computazionale,
i problemi possono essere classificati per difficoltà

La difficoltà dipende dal fatto
di essere in grado o meno di trovare un
algoritmo efficiente (in termini di TEMPO)
per il problema,
non a quanto sia difficile trovare
UN algoritmo qualsiasi

Problemi facili e problemi difficili

In base a risultati sul costo computazionale,
i problemi possono essere classificati per difficoltà

- PROBLEMI TRATTABILI (facili): esiste un algoritmo per il problema di **costo computazionale polinomiale** (i.e., $T(n) \in O(n^k), k \geq 0$).
- PROBLEMI PRESUMIBILMENTE INTRATTABILI (difficili): NON abbiamo un algoritmo di **costo polinomiale**, ma NON e' stato dimostrato che non esiste.
- PROBLEMI INTRATTABILI: si può dimostrare che NON esiste un algoritmo per il problema di **costo polinomiale**.
- PROBLEMI IRRISOLVIBILI: si può dimostrare che NON esiste un algoritmo per il problema (**indipendentemente dal costo**).

Dimostrazioni

di proposizioni sulle dispense

Notazione "O-grande" $O(\cdot)$

PROPOSIZIONE

Per ogni costante $k \geq 0$ abbiamo che

$$n^k \in O(2^n)$$

$$n^k \leq n^{n/(\log n)} = (2^{\log n})^{n/(\log n)} = 2^n$$

Fissato k,
esiste n_0 tale che
$\forall\ n \geq n_0 : k \leq n/\log n$

$2^{\log n} = n$
Per definizione
di logaritmo

Per le proprietà
delle potenze

$\forall\ n \geq n_0$
$c = 1$

Esempio: $k = 3 \rightarrow n_0 = 16$
infatti $k = 3 \leq 4 = 16/4 = 16/\log 16$

Notazione "O-grande" $O(\cdot)$

PROPOSIZIONE

Abbiamo che

$$2^n \in O(n!)$$

$$2^n = \underbrace{2 \times 2 \times \ldots \times 2}_{n \text{ volte}} \leq \underbrace{1 \times 2 \times 3 \times \ldots \times n}_{n \text{ volte}} = n!$$

Per definizione di potenza

$\forall\ n \geq 4$

Per definizione di fattoriale

$$\boxed{\begin{array}{l} \forall\ n \geq n_0 = 4 \\ c = 1 \end{array}}$$

Notazione "O-grande" $O(\cdot)$

PROPOSIZIONE

Abbiamo che

$$n! \in O(n^n)$$

$$n! = \underbrace{1 \times 2 \times 3 \times \ldots \times n}_{n \text{ volte}} \leq \underbrace{n \times n \times n \times \ldots \times n}_{n \text{ volte}} = n^n$$

Per definizione di fattoriale

$\forall\ n \geq 1$

Per definizione di potenza

$$\boxed{\begin{array}{l} \forall\ n \geq n_0 = 1 \\ c = 1 \end{array}}$$

UNIMORE