UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria
"Enzo Ferrari"

Algoritmi e Strutture Dati

Complessità computazionale degli Algoritmi

AGENDA

- **Valutazione della “bontà” di un algoritmo**
- **Complessità temporale**
- **Calcolo della complessità in numero di passi base**
- **Comportamento asintotico di una funzione**
- **Notazione O e algebra degli O**
- **Notazione Ω**

VALUTAZIONE DELLA “BONTÀ” DI UN ALGORITMO

Un problema può essere risolto da algoritmi diversi. È quindi necessario trovare un metodo per confrontare la “bontà” degli algoritmi.

<u>*Complessità computazionale*</u>

Costo di un algoritmo in termini di quantità di risorsa richiesta per il calcolo

Complessità temporale (risorsa = tempo)

Complessità spaziale (risorsa = spazio)

Complessità di Input/Output (risorsa = tempo per l’accesso alle periferiche)

Tempo Accesso RAM (core i9): 2.4 GHz > 4,2 10^{-10} sec

Tempo Accesso SSD: 1,0 10^{-5} sec (fino a 540 MB/s in lettura)

LA COMPLESSITÀ TEMPORALE

Complessità temporale

Costo di un algoritmo in termini di quantità di tempo richiesto per il calcolo

Si supponga di avere a disposizione due algoritmi diversi per ordinare n numeri interi.

- Il primo algoritmo ordina gli n numeri con n^2 istruzioni
- il secondo ordina gli n numeri con $n*\log_2 n$ istruzioni.

Supponiamo di avere un processore i9 a 2.9 GHz > 3.4 10^{-10} sec.

	$n= 10^4$	$n= 10^6$
n^2	34 msec	340 sec
$n*\log_2 n$	45 µsec	6.7 msec

FATTORI PER LA VALUTAZIONE DEL TEMPO

Scelto un algoritmo per risolvere il problema, i fattori che influenzano il tempo richiesto per il calcolo sono:

- la dimensione dell'input
- la configurazione dell’input
- la velocità della macchina usata
- il linguaggio di programmazione

Vogliamo un modello di calcolo per il calcolo della complessità temporale che consenta di confrontare algoritmi a prescindere

- dal linguaggio con cui sono implementati
- dalla potenza della macchina su cui vengono eseguiti

A seconda del problema, per dimensione dell'input si indicano cose diverse:

- La grandezza di un numero (es.: problemi di calcolo)
- Quanti elementi sono in ingresso (es.: ordinamento)
- Quanti bit compongono un numero

Indichiamo con "n" la dimensione dell'input.

MODELLO DI COSTO

Sono operazioni di **costo unitario** le istruzioni semplici:

- Lettura da file. Es: scanf();
- Scrittura su file. Es: printf();
- Assegnamento. Es. x=x+y;

Si indicherà una operazione di costo unitario con il termine passo base.

- Un blocco ha un costo pari alla somma dei costi delle istruzioni che contiene.
- Le espressioni condizionali hanno un costo pari alla somma del costo di valutazione della condizione più il costo del ramo selezionato
- Le strutture di controllo hanno un costo pari alla somma dei costi dell'esecuzione delle istruzioni interne, più la somma dei costi delle condizioni
- Le chiamate di funzioni hanno un costo pari al costo del blocco. Il passaggio di parametri ha un costo nullo.

CALCOLO DELLA COMPLESSITÀ IN NUMERO DI PASSI BASE

ESEMPIO 1 (somma fino a n)

```
int i = 1, s = 0;
while (i <= n) {
    s = s + i;
    i = i + 1;
}
```

Assegnamento esterno (i=1, s=0)	2
Numero di test (i <= n)	n+1
Assegnamento interno (s=s+i, i=i+1)	n*2
Numero totale di passi base:	3+3*n

(n è il limite della somma)

CALCOLO DELLA COMPLESSITÀ IN NUMERO DI PASSI BASE

ESEMPIO 2 (potenza m^{2*n})

```
int i = 1, p = 1;
while (i <= 2 * n) {
    p = p * m;
    i = i + 1;
}
```

Assegnamento esterno (i=1, p=1)	2
Numero di test (i <= 2*n)	2*n+1
Assegnamento interno (p=p*m, i=i+1)	2*(2*n)
Numero totale di passi base:	3+6*n

(2*n è l'esponente)

CALCOLO DELLA COMPLESSITÀ IN NUMERO DI PASSI BASE

ESEMPIO 3 (ciclo condizionale)

```
int i = 0;
while(i < n){
    for (int j=0;j<n;j++)
        if(i<j) printf("Ciao");
        else printf("a tutti");
    i = i + 1;}
```

Assegnamento esterno (i=0)	1 +
Test while (i < n)	n+1 +
Numero blocchi while (	n*(
Test nel for (j<n)	n + 1 +
Corpo for ((i<j), printf()e j++):	3*n +
Assegnamenti (j=0; e i=i+1;):	2)
Numero totale di passi base:	$2+4*n+4*n^2$

CALCOLO DELLA COMPLESSITÀ IN NUMERO DI PASSI BASE

ESEMPIO 4 (chiamata procedura)

```
void Stampastelle(int ns){
   for(int i=0;i<ns;i++)
     printf("*");
}

void main(){
int n,m;
   printf("Quante stelle per riga?");
   scanf("%d",&n);
   printf("Quante righe di stelle?");
   scanf("%d",&m);
   for(j=0;j<m;j++) Stampastelle(n);}
```

Complessità della procedura:

$$2 + 3*ns$$

Complessità del main:

$$6+m+m*((2+3*n)+1) =$$

$$6+4*m+3*m*n$$

Nel calcolo della complessità di un algoritmo bisogna tener conto di tutti i parametri che definiscono la dimensione dell’input. Il costo di esecuzione di una procedura o funzione può dipendere dai parametri che le vengono dati in ingresso.

CALCOLO DELLA COMPLESSITÀ IN NUMERO DI PASSI BASE

ESEMPIO 5 (variante chiamata procedura)

```
void Stampastelle(int ns){
   for(int i=0;i<ns;i++)
    printf("*");
}

void main(){
int m;
    printf("Quante righe di
              stelle?");
   scanf("%d",&m);
   for(j=0;j<m;j++)
Stampastelle(j);}
```

Complessità della procedura:

$2 + 3*ns$

Complessità del main:

$4+m+\sum_{j=0}^{j=m-1}((2+3*j)+1) =$

$4+m+3*m+3*m*(m-1)/2 =$

$4+5/2*m+3/2*m^2$

Una stessa procedura/funzione può essere chiamata con dati diversi (input di dimensione diversa).

COSTO IN FUNZIONE DELLA CONFIGURAZIONE DELL'INPUT

```
int ricercaInVettore(int V[], int e){
    for(int i=0;i<n;i++)
        if(V[i]==e) return i;
    return -1;}
```

ESEMPIO 6

Supponiamo che V abbia la seguente configurazione:

3	-2	0	7	5	4	0	8	-3	-1	9	12	20	5

Cerchiamo l'elemento 8 (e=8):

1 ass + 8 test + 8 test if + 7 incr. + 1 return = 2+2*8+(8-1) = 25

Cerchiamo l'elemento 6 (e=6):

Poiché tale elemento non è presente, dobbiamo scorrere l'array per intero, con complessità:

1 ass. + (n+1) test + n test if + n incr. +1 return =

3+3*n passi base (3+3*14 passi nell'esempio)

CASO MIGLIORE, MEDIO, PEGGIORE

Questo è un classico esempio in cui il costo del programma dipende non solo dalla dimensione dell'input ma anche dai **particolari valori dei dati stessi**.

Nel caso in cui il costo di esecuzione del programma dipende dalla configurazione dell'input, è possibile distinguere diversi casi:

- **Caso migliore**: si fa riferimento alla configurazione che comporta il costo minore
- **Caso peggiore**: si fa riferimento alla configurazione che comporta il costo maggiore
- **Caso medio**: si calcola la somma dei costi delle esecuzioni rispetto a tutti i possibili dati di ingresso

CASO MIGLIORE, MEDIO, PEGGIORE (2)

Caso migliore: l'elemento cercato è il primo. Il costo in numero di passi base è 4.

Caso peggiore: l'elemento cercato non appartiene all'array.

Il costo in numero di passi base è 3+3*n.

Caso medio:

Ipotesi di distribuzione uniforme dei valori nell'array. La probabilità di trovare l'elemento e cercato in posizione i-esima è: **P(V[i]=e) = 1/n**

Se l'elemento e è nella posizione i-esima, Il costo in numero di passi base è: C(V[i]=e)=2+2*i+(i-1)=1+3*i

Il numero medio di confronti da effettuare è:

$$\sum_{i=1}^{j=n} \frac{1}{n} * (1+3*i) = 1 + \frac{3}{n} * \frac{n*(n+1)}{2} = \frac{5+3*n}{2}$$

FUNZIONE DI COSTO

Usiamo il termine funzione di costo per indicare una funzione $f\colon \mathbb{N} \rightarrow \mathbb{R}$ dall'insieme dei numeri naturali ai reali.

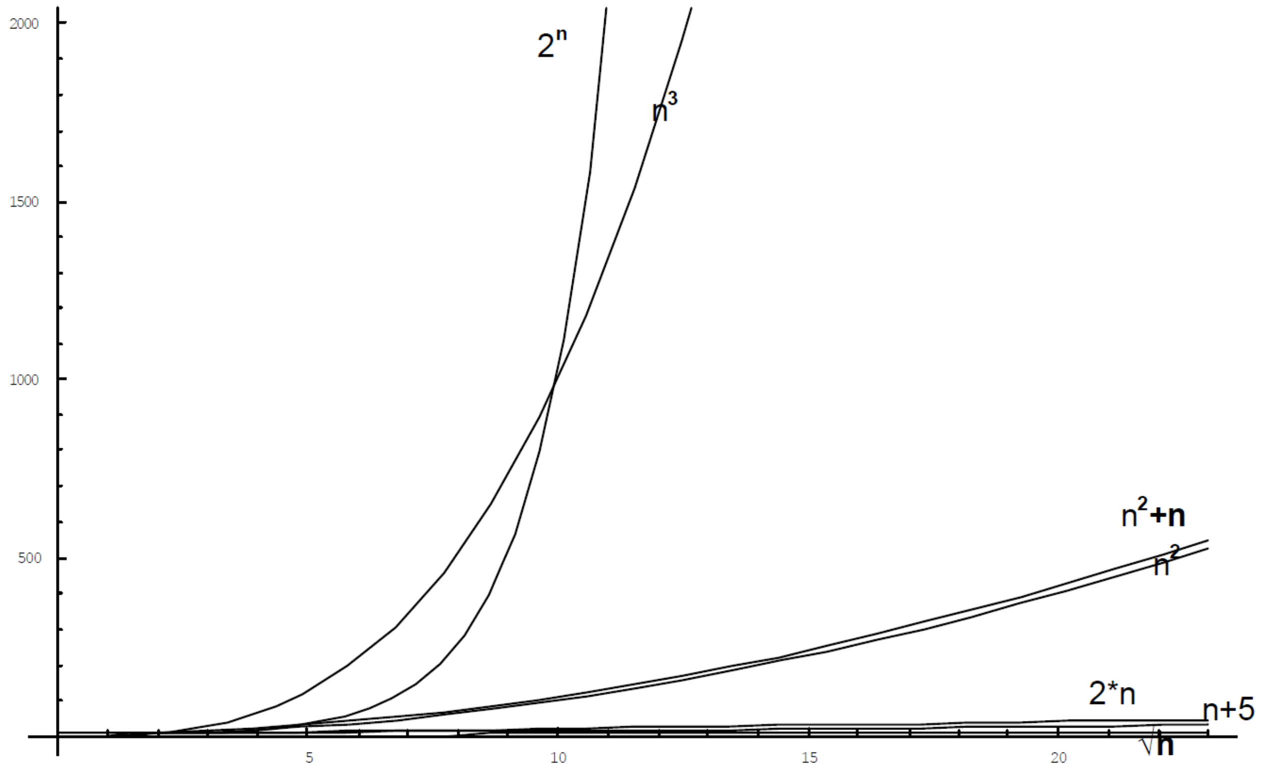
Come possiamo confrontare le funzioni di costo?

Ricerca, caso ottimo: $f(n)$: a_1

Ricerca, caso medio: $f(n)$: $a_2 n + b_2$

Stampastelle: $f(n)$: $a_3 n^2 + b_3 n + c_3$

COMPORTAMENTO ASINTOTICO DI UNA FUNZIONE



Per n che tende all'infinito le funzioni 2*n e n+5 hanno un comportamento simile. Lo stesso vale per n^2 e n^2+n.

La funzione 2^n diverge molto rapidamente

COMPORTAMENTO ASINTOTICO DI UNA FUNZIONE (2)

Non è sempre facile quantificare con esattezza la complessità di un algoritmo in numero di passi base.

Per l'analisi della complessità

dei problemi e degli algoritmi che li risolvono

si fa spesso riferimento al comportamento asintotico.

Comportamento asintotico di una funzione

Comportamento al crescere della dimensione n all'infinito trascurando le costanti moltiplicative ed additive e tutti i termini di ordine inferiore

La notazione asintotica si basa su 3 notazioni principali:

- **Notazione O e algebra degli O**
- **Notazione Ω**
- **Notazione Θ**

NOTAZIONE O

Algoritmi diversi per la soluzione dello stesso problema possono essere confrontati valutando il numero di operazioni in ordine di grandezza.

Notazione O

Sia g(n) una funzione di costo, indichiamo con O(g(n)) l'insieme delle funzioni f(n) tali che:

$$\exists\, c > 0, \exists\, m \geq 0 : f(n) \leq cg(n), \forall\, n \geq m$$

Come si legge: $f(n)$ è «O grande» (big-O) di $g(n)$

Come si scrive: $f(n) = O(g(n))$

$g(n)$ è un limite asintotico superiore di $f(n)$

$f(n)$ cresce al più come $g(n)$

NOTAZIONE Ω

Algoritmi diversi per la soluzione dello stesso problema possono essere confrontati valutando il numero di operazioni in ordine di grandezza.

Notazione Ω

Sia g(n) una funzione di costo, indichiamo con Ω(g(n)) l'insieme delle funzioni f(n) tali che:

$$\exists\, \boldsymbol{c} > \mathbf{0}, \exists\, \boldsymbol{m} \geq \mathbf{0} : \boldsymbol{f}(\boldsymbol{n}) \geq \boldsymbol{c}\boldsymbol{g}(\boldsymbol{n}), \forall\, \boldsymbol{n} \geq \boldsymbol{m}$$

Come si legge: $\boldsymbol{f}(\boldsymbol{n})$ è «Omega grande» di $\boldsymbol{g}(\boldsymbol{n})$

Come si scrive: $\boldsymbol{f}(\boldsymbol{n}) = \Omega(\boldsymbol{g}(\boldsymbol{n}))$

$\boldsymbol{g}(\boldsymbol{n})$ è un limite asintotico inferiore di $\boldsymbol{f}(\boldsymbol{n})$

$\boldsymbol{f}(\boldsymbol{n})$ cresce almeno quanto $\boldsymbol{g}(\boldsymbol{n})$

NOTAZIONE Θ

Algoritmi diversi per la soluzione dello stesso problema possono essere confrontati valutando il numero di operazioni in ordine di grandezza.

Notazione Θ

Sia g(n) una funzione di costo, indichiamo con Θ(g(n)) l'insieme delle funzioni f(n) tali che:

$$\exists\, c_1 > 0, \exists\, c_2 > 0, \exists\, m \geq 0 : c_1 g(n) \geq f(n) \geq c_2 g(n), \forall\, n \geq m$$

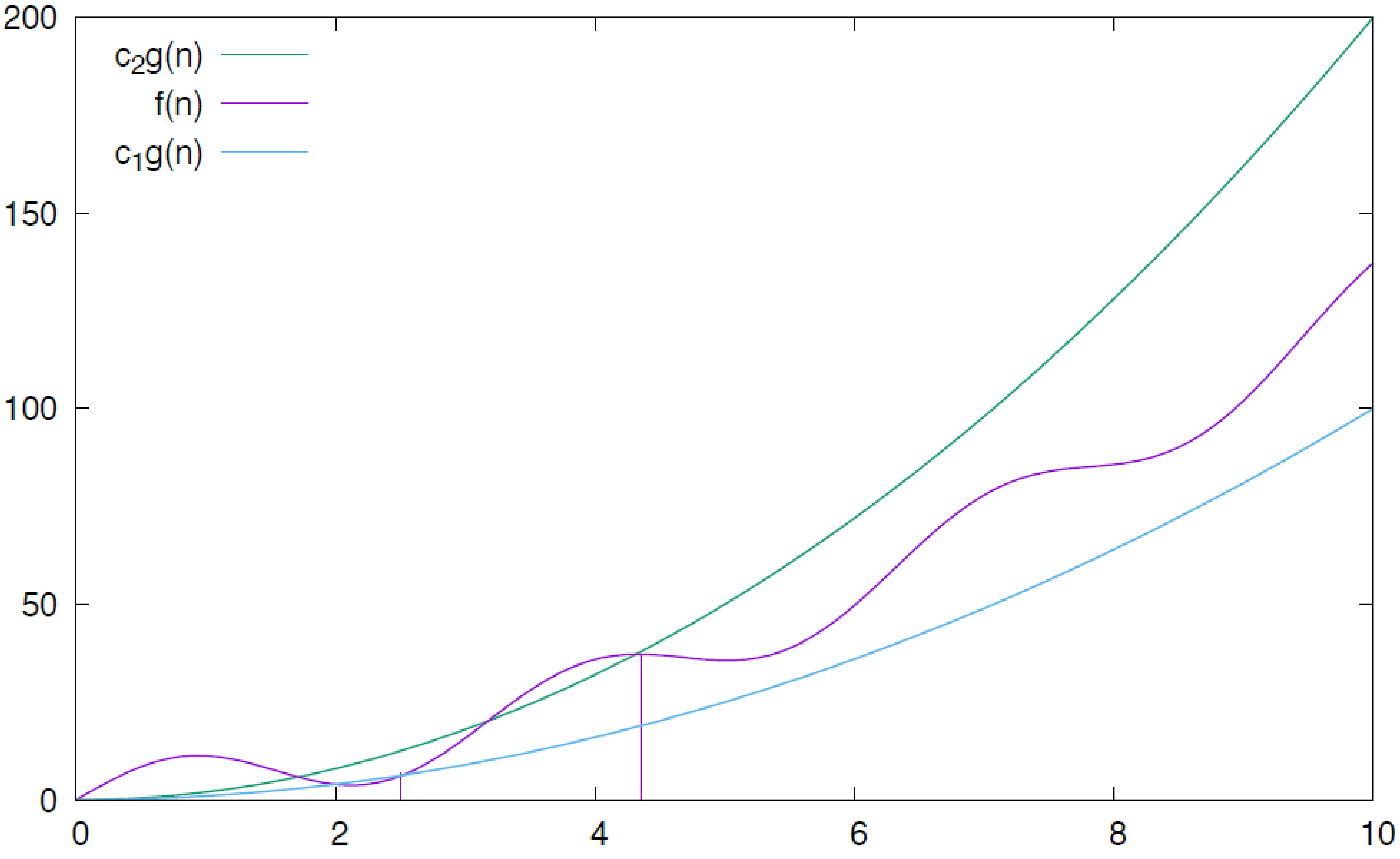
Come si legge: $f(n)$ è «Theta» di $g(n)$

Come si scrive: $f(n) = \Theta(g(n))$

$f(n)$ cresce esattamente quanto $g(n)$

$f(n) = \Theta(g(n))$ se e solo se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

Graficamente



Regola generale - Espressioni polinomiali

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \ ... + \ a_1 n^1 + a_0 n^0, a_k > 0 \ \rightarrow \ \ f(n) = \Theta(n^k)$$

Limite superiore: $\exists c > 0, \exists m \geq 0 : f(n) \leq cn^k, \forall n \geq m$

$$\begin{aligned}
f(n) &= a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \\
&\leq a_k n^k + |a_{k-1}| n^{k-1} + \ldots + |a_1| n + |a_0| \\
&\leq a_k n^k + |a_{k-1}| n^k + \ldots + |a_1| n^k + |a_0| n^k \qquad \forall n \geq 1 \\
&= (a_k + |a_{k-1}| + \ldots + |a_1| + |a_0|) n^k \\
&\stackrel{?}{\leq} cn^k
\end{aligned}$$

che è vera per $c \geq (a_k + |a_{k-1}| + \ldots + |a_1| + |a_0|) > 0$ e per $m = 1$.

Regola generale - Espressioni polinomiali

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n^1 + a_0 n^0, a_k > 0 \rightarrow \quad f(n) = \Theta(n^k)$$

Limite inferiore: $\exists d > 0, \exists m \geq 0 : f(n) \geq d n^k, \forall n \geq m$

$$\begin{aligned} f(n) &= a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \\ &\geq a_k n^k - |a_{k-1}| n^{k-1} - \ldots - |a_1| n - |a_0| \\ &\geq a_k n^k - |a_{k-1}| n^{k-1} - \ldots - |a_1| n^{k-1} - |a_0| n^{k-1} \qquad \forall n \geq 1 \\ &\stackrel{?}{\geq} d n^k \end{aligned}$$

L'ultima equazione è vera se:

$$d \leq a_k - \frac{|a_{k-1}|}{n} - \frac{|a_{k-2}|}{n} - \ldots - \frac{|a_1|}{n} - \frac{|a_0|}{n} > 0 \Leftrightarrow n > \frac{|a_{k-1}| + \ldots + |a_0|}{a_k}$$

Proprietà

Dualità

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

Dimostrazione:

$$\begin{aligned}
f(n) = O(g(n)) &\Leftrightarrow f(n) \leq cg(n), \forall n \geq m \\
&\Leftrightarrow g(n) \geq \frac{1}{c} f(n), \forall n \geq m \\
&\Leftrightarrow g(n) \geq c' f(n), \forall n \geq m, c' = \frac{1}{c} \\
&\Leftrightarrow g(n) = \Omega(f(n))
\end{aligned}$$

Proprietà

Eliminazione delle costanti

$$f(n) = O(g(n)) \Leftrightarrow af(n) = O(g(n)), \forall a > 0$$
$$f(n) = \Omega(g(n)) \Leftrightarrow af(n) = \Omega(g(n)), \forall a > 0$$

Dimostrazione:

$$\begin{aligned} f(n) = O(g(n)) &\Leftrightarrow f(n) \leq cg(n), \forall n \geq m \\ &\Leftrightarrow af(n) \leq acg(n), \forall n \geq m, \forall a \geq 0 \\ &\Leftrightarrow af(n) \leq c'g(n), \forall n \geq m, c' = ac > 0 \\ &\Leftrightarrow af(n) = O(g(n)) \end{aligned}$$

Proprietà

Simmetria

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

Dimostrazione

Grazie alla proprietà di dualità:

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

Proprietà

Transitività

$$f(n) = O(g(n)), g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$

Dimostrazione

$$\begin{aligned} f(n) = O(g(n)) \wedge g(n) = O(h(n)) &\quad \Rightarrow \\ f(n) \leq c_1 g(n) \wedge g(n) \leq c_2 h(n) &\quad \Rightarrow \\ f(n) \leq c_1 c_2 h(n) &\quad \Rightarrow \\ f(n) = O(h(n)) & \end{aligned}$$

Esempi

$$f(n) = 10n^3 + 2n^2 + 7 \stackrel{?}{=} O(n^3)$$

Dobbiamo provare che $\exists c > 0, \exists m \geq 0 : f(n) \leq cn^3, \forall n \geq m$

$$\begin{aligned} f(n) &= 10n^3 + 2n^2 + 7 \\ &\leq 10n^3 + 2n^3 + 7 && \forall n \geq 1 \\ &\leq 10n^3 + 2n^3 + 7n^3 && \forall n \geq 1 \\ &= 19n^3 \\ &\stackrel{?}{\leq} cn^3 \end{aligned}$$

che è vera per ogni $c \geq 19$ e per ogni $n \geq 1$, quindi $m = 1$.

Esempi

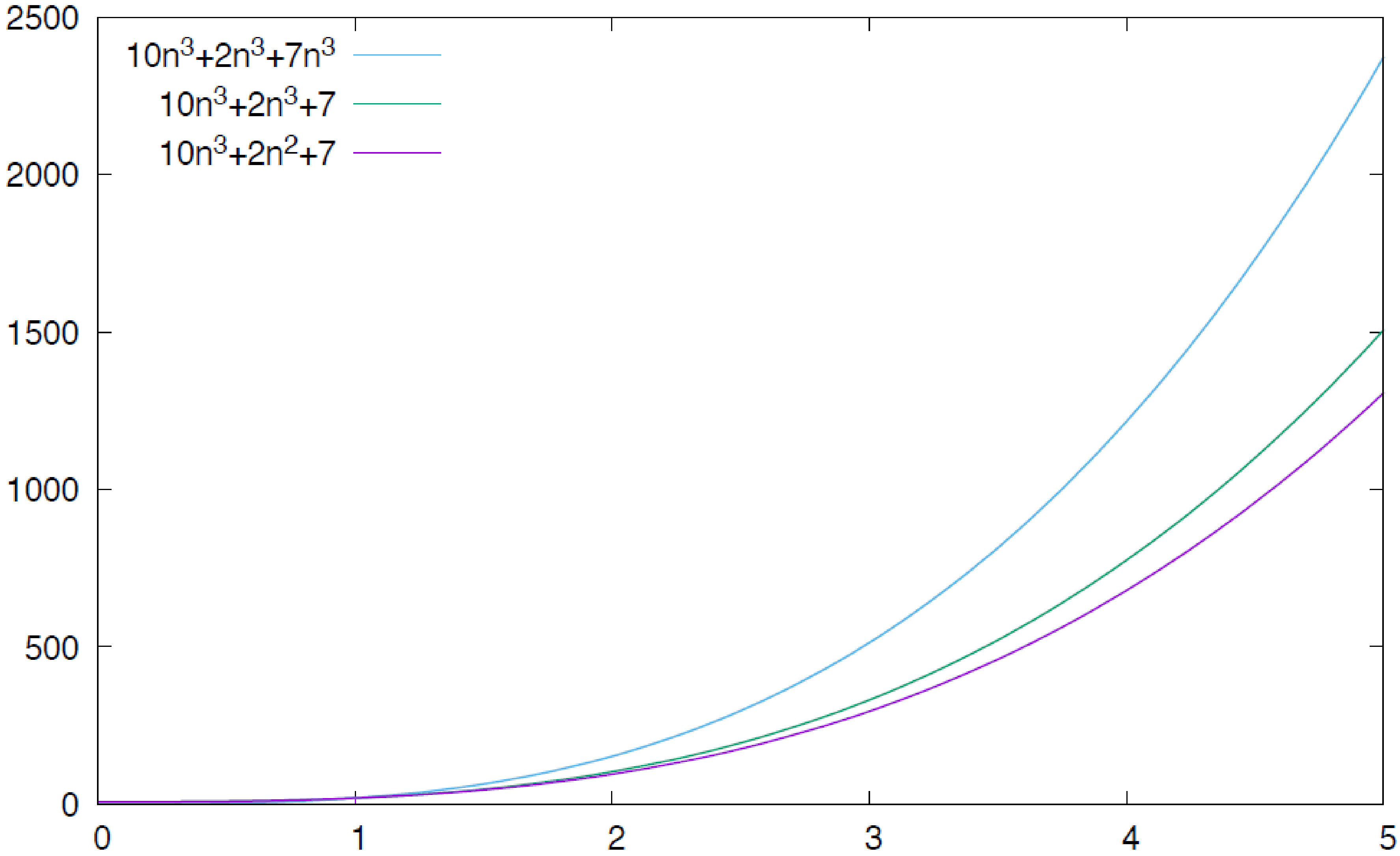
$$f(n) = 10n^3 + 2n^2 + 7 \stackrel{?}{=} O(n^3)$$

Dobbiamo provare che $\exists c > 0, \exists m \geq 0 : f(n) \leq cn^3, \forall n \geq m$

$$\begin{aligned}
f(n) &= 10n^3 + 2n^2 + 7 \\
&\leq 10n^3 + 2n^3 + 7 && \forall n \geq 1 \\
&\leq 10n^3 + 2n^3 + 7n^3 && \forall n \geq 1 \\
&= 19n^3 \\
&\stackrel{?}{\leq} cn^3
\end{aligned}$$

che è vera per ogni $c \geq 19$ e per ogni $n \geq 1$, quindi $m = 1$.

Esempi - graficamente



Esempi

$$f(n) = 3n^2 + 7n \stackrel{?}{=} \Theta(n^2)$$

Limite inferiore: $\exists c_1 > 0, \exists m_1 \geq 0 : f(n) \geq c_1 n^2, \forall n \geq m_1$

$$\begin{aligned} f(n) &= 3n^2 + 7n \\ &\geq 3n^2 && \text{Per } n \geq 0 \\ &\stackrel{?}{\geq} c_1 n^2 \end{aligned}$$

che è vera per ogni $c_1 \leq 3$ e per ogni $n \geq 0$, quindi $m_1 = 0$

Esempi

$$f(n) = 3n^2 + 7n \stackrel{?}{=} \Theta(n^2)$$

Limite superiore: $\exists c_2 > 0, \exists m_2 \geq 0 : f(n) \leq c_2 n^2, \forall n \geq m_2$

$$\begin{aligned} f(n) &= 3n^2 + 7n \\ &\leq 3n^2 + 7n^2 && \text{Per } n \geq 1 \\ &= 10n^2 \\ &\stackrel{?}{\leq} c_2 n^2 \end{aligned}$$

che è vera per ogni $c_2 \geq 10$ e per ogni $n \geq 1$, quindi $m_2 = 1$

Esempi

$$f(n) = 3n^2 + 7n \stackrel{?}{=} \Theta(n^2)$$

Notazione Θ:

$$\exists c_1 > 0, \exists c_2 > 0, \exists m \geq 0 : c_1 n^2 \leq f(n) \leq c_2 n^2, \forall n \geq m$$

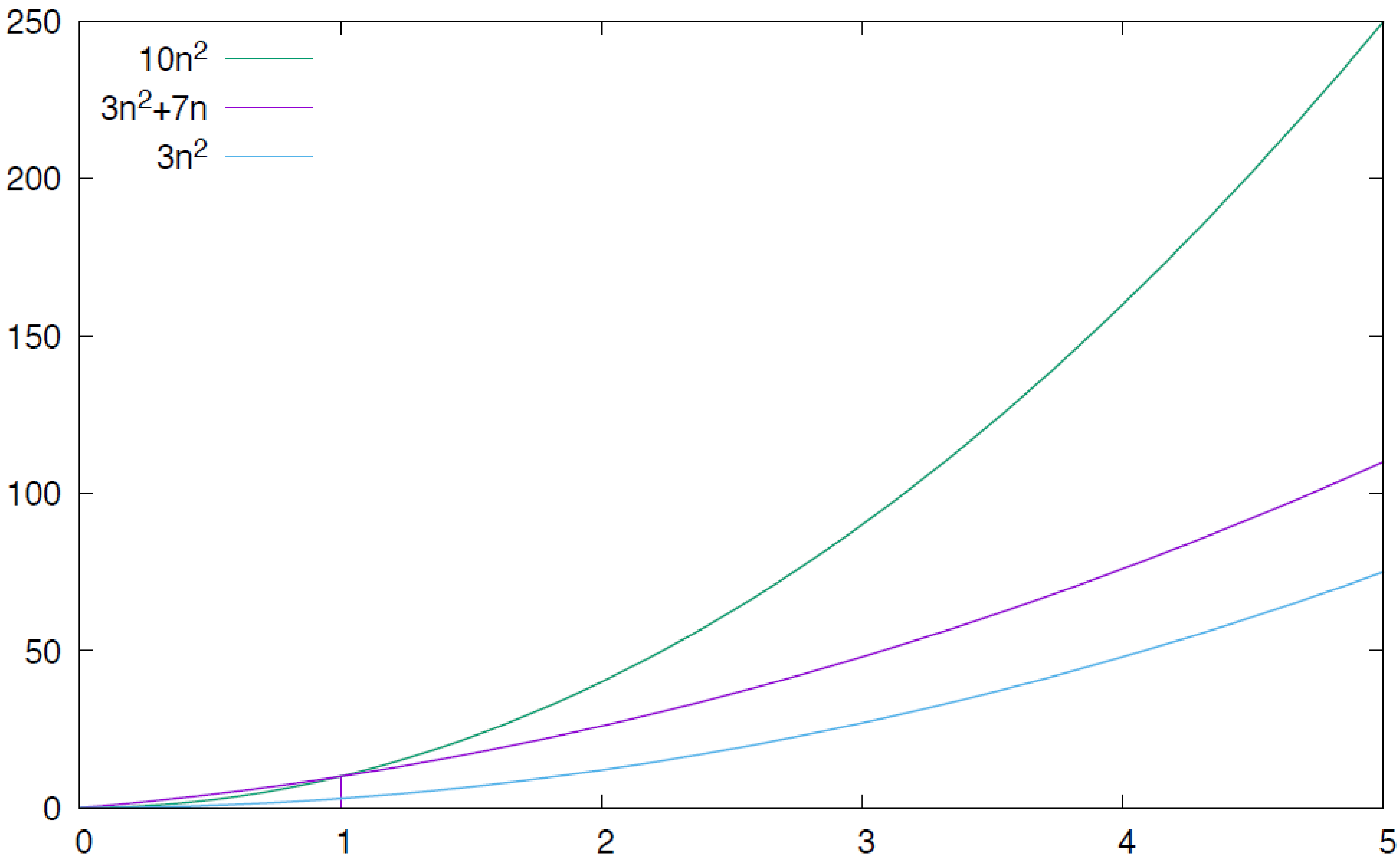
Con questi parametri:

$$c_1 = 3$$

$$c_2 = 10$$

$$m = \max\{m_1, m_2\} = \max\{0, 1\} = 1$$

Esempi - graficamente



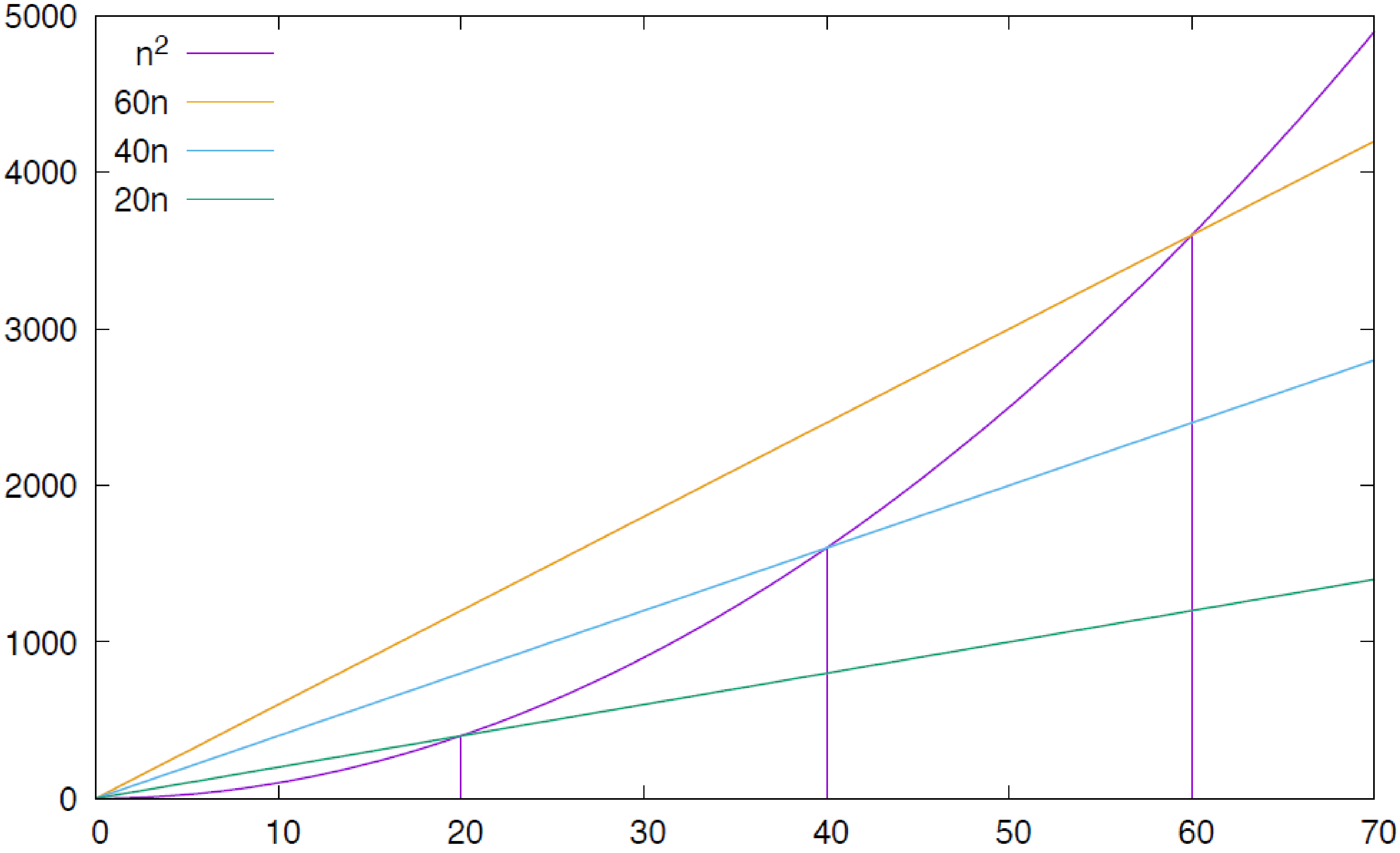
Esempi

$$n^2 \stackrel{?}{=} O(n)$$

Dobbiamo dimostrare che $\exists c > 0, \exists m > 0 : n^2 \leq cn, \forall n \geq m$

- Otteniamo questo: $n^2 \leq cn \Leftrightarrow c \geq n$
- Questo significa che c cresce con il crescere di n, ovvero che non possiamo scegliere una costante c

Esempi - graficamente



Proprietà

Sommatoria (sequenza di algoritmi)

$$f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n)) \Rightarrow f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$$

$$f_1(n) = \Omega(g_1(n)), f_2(n) = \Omega(g_2(n)) \Rightarrow f_1(n) + f_2(n) = \Omega(\max(g_1(n), g_2(n)))$$

Dimostrazione (Lato O)

$$\begin{aligned}
f_1(n) = O(g_1(n)) \wedge f_2(n) = O(g_2(n)) & \quad \Rightarrow \\
f_1(n) \leq c_1 g_1(n) \wedge f_2(n) \leq c_2 g_2(n) & \quad \Rightarrow \\
f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) & \quad \Rightarrow \\
f_1(n) + f_2(n) \leq \max\{c_1, c_2\}(2 \cdot \max(g_1(n), g_2(n))) & \quad \Rightarrow \\
f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n))) &
\end{aligned}$$

Proprietà

Prodotto (Cicli annidati)

$$f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n)) \Rightarrow f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$$
$$f_1(n) = \Omega(g_1(n)), f_2(n) = \Omega(g_2(n)) \Rightarrow f_1(n) \cdot f_2(n) = \Omega(g_1(n) \cdot g_2(n))$$

Dimostrazione

$$\begin{aligned} f_1(n) = O(g_1(n)) \wedge f_2(n) = O(g_2(n)) \quad &\Rightarrow \\ f_1(n) \leq c_1 g_1(n) \wedge f_2(n) \leq c_2 g_2(n) \quad &\Rightarrow \\ f_1(n) \cdot f_2(n) \leq c_1 c_2 g_1(n) g_2(n) \end{aligned}$$

ALGEBRA DEGLI "O" GRANDI

Primo blocco P

Secondo blocco Q

```
i=0;
while (i<n){
    stampastelle(i);
     i++;}
for(i=0;i<2*n;i++)
    printf("ciao");
```

$f_P(n)$: complessità del primo blocco

$f_Q(n)$: complessità del secondo

$$O(f_P(n)+ f_Q(n)) = O(\max\{ f_P(n), f_Q(n)\})$$

ALGEBRA DEGLI "O" GRANDI

Blocchi annidati

Primo blocco P

Secondo blocco Q

```
for(i=0;i<n;i++){
    s[i]=0;
    for(j=0;j<m;j++)
      s[i]=s[i]+j;
}
```

$f_P(n)$: complessità del primo blocco

$f_Q(n)$: complessità del secondo

$$O(f_P(n) * f_Q(n)) = O(f_P(n)) * O(f_Q(n))$$

ESEMPIO

```
#define N 7
int ricercaInVettore(int V[], int e){
   for(int i=0;i<n;i++)
        if(V[i]==e) return i;
   return -1;}
void main(){
    int vett[N];
    int elem;
    for(int i=0;i<N;i++)
         scanf("%d",&vett[i]);
    printf("Elemento cercato");
    scanf("%d",&elem)
    if(ricercaInVettore(vett,elem)!=-1)
         printf("trovato!");
    else printf("non trovato!");
}
```

Due blocchi in sequenza

Acquisizione vettore

$f_P(n) = n$

Ricerca elemento

$f_Q(n) = 3 + 3*n$

$O(f_P(n) + f_Q(n)) =$

$O(f_Q(n)) = n$

Complessità di un problema

Obiettivo: riflettere su complessità di problemi/algoritmi

- In alcuni casi, si può migliorare quanto si ritiene "normale"
- In altri casi, è impossibile fare di meglio
- Qual è il rapporto fra un problema computazionale e l'algoritmo?

Notazione $O(f(n))$ – Limite superiore

Un problema ha complessità $O(f(n))$ se esiste almeno un algoritmo che ha complessità $O(f(n))$

Notazione $\Omega(f(n))$ – Limite inferiore

Un problema ha complessità $\Omega(f(n))$ se tutti i possibili algoritmi che lo risolvono hanno complessità $\Omega(f(n))$.

Algoritmi vs problemi

Complessità in tempo di un algoritmo

La quantità di tempo richiesta per input di dimensione n

- $O(f(n))$: Per tutti gli input, l'algoritmo costa al più $f(n)$
- $\Omega(f(n))$: Per tutti gli input, l'algoritmo costa almeno $f(n)$
- $\Theta(f(n))$: L'algoritmo richiede $\Theta(f(n))$ per tutti gli input

Complessità in tempo di un problema computazionale

La complessità in tempo relative a tutte le possibili soluzioni

- $O(f(n))$: Complessità del miglior algoritmo che risolve il problema
- $\Omega(f(n))$: Dimostrare che nessun algoritmo può risolvere il problema in tempo inferiore a $\Omega(f(n))$
- $\Theta(f(n))$: Algoritmo ottimo