

Tempo di calcolo e complessità asintotica

Algoritmi e strutture dati

Ugo de'Liguoro, Andras Horvath

1

Sommario

- Obiettivi
 - introdurre le nozioni di tempo di calcolo e di confronto asintotico delle funzioni
- Argomenti
 - caso peggiore e caso migliore
 - notazione asintotica
 - inclusioni tra classi asintoticamente limitate
 - complessità di un problema

2

Complessità di un algoritmo

Quante risorse usa l'algoritmo?

- *tempo* = quanto tempo ci mette
- *spazio* = quanta memoria occorre per eseguire l'algoritmo
- *hardware* = numero di processori, numero dei componenti (porte) di un circuito, ecc...

3

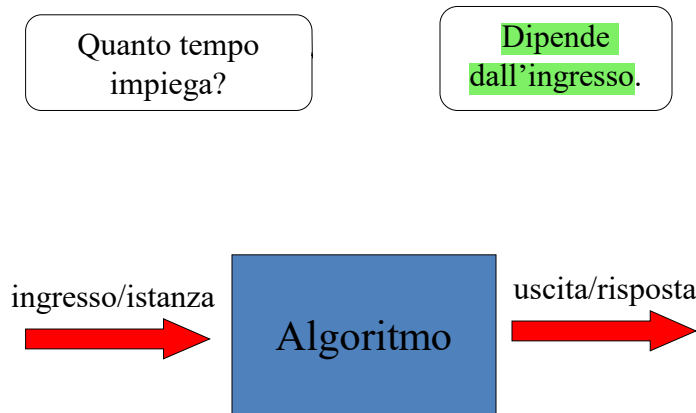
Complessità temporale

Ci interessa stabilirla per vari motivi

- per capire quanto tempo ci vuole per eseguire un programma che lo implementa
- per stimare la grandezza massima dell'ingresso di un'esecuzione ragionevole
- per confrontare l'efficienza di più algoritmi che risolvono lo stesso problema
- per sapere quale parte del codice sarà eseguita più volte ...

4

Il tempo di calcolo è una funzione



5

Definizione del tempo

Sotto certe condizioni queste misure differiscono di un fattore costante!

Possiamo seguire diversi approcci:

- il numero dei secondi (dipendente dalla macchina)
- il numero delle operazioni elementari, ciascuna con un proprio coefficiente
- il numero delle volte che una specifica operazione viene eseguita

6

Esempio: minimo in un vettore

	costo	volte
Minimo (A, j, k)	c_1	1
Pre: A vettore di dimensione $> k \geq j$		
Post: ritorna il minimo in $A[j..k]$		
$min \leftarrow A[j]$	c_2	1
for $i \leftarrow j + 1$ to k do	c_3	$k - j + 1$
if $A[i] < min$ then	c_4	$k - j$
$min \leftarrow A[i]$	c_5	$\leq k - j$
return min	c_6	1

valori di i : $j+1, j+2, j+3, \dots, k+1$
 $((k+1) - (j+1) + 1 = k - j + 1)$

7

Esempio: minimo in un vettore

Posto $n = k - j + 1$, vale a dire il numero degli elementi tra i quali cerchiamo il minimo, si ha

$$\begin{aligned}
 T(n) &\leq c_1 + c_2 + nc_3 + (n-1)c_4 + (n-1)c_5 + c_6 \\
 &= c_1 + c_2 + nc_3 + nc_4 - c_4 + nc_5 - c_5 + c_6 \\
 &= n(c_3 + c_4 + c_5) + (c_1 + c_2 - c_4 - c_5 + c_6) \\
 &= an + b \\
 T(n) &\geq c_1 + c_2 + nc_3 + (n-1)c_4 + c_6 \\
 &= c_1 + c_2 + nc_3 + nc_4 - c_4 + c_6 \\
 &= n(c_3 + c_4) + (c_1 + c_2 - c_4 + c_6) \\
 &= cn + d
 \end{aligned}$$

con a, b, c, d costanti.

Crescita lineare in funzione di n .

8

Esempio: minimo in un vettore

Riassumendo, posto $n = k - j + 1$, si ha

$$cn + d \leq T(n) \leq an + b$$

con

$$a = c_3 + c_4 + c_5, b = (c_1 + c_2 - c_4 - c_5 + c_6)$$

$$c = c_3 + c_4, d = (c_1 + c_2 - c_4 + c_6)$$

Crescita lineare in funzione di n .

9

La dimensione dell'ingresso

- nel caso di **Minimo**(A, j, k), per quanto riguarda l'ingresso, ciò che conta è il numero degli elementi in $A[j..k]$, non il loro valore (in ogni caso la crescita in funzione di n è lineare)
- in generale la dimensione dell'ingresso è una **misura della sua rappresentazione** (a meno di una costante moltiplicativa)

$$|m| = \text{dimensione di } m = \text{num. bit per rappresentare } m = \lfloor \log_2(m) \rfloor + 1$$

$$|A[0..n-1]| = \text{dimensione di } A[0..n-1] = n \cdot c$$

dove c = numero bit del generico elemento di A

- nel seguito useremo $c = 1$ perché **moltiplicare per un costante** (come vedremo) non conta dal punto di vista dell'analisi asintotica

10

Quale ingresso di dimensione n ?

Supponiamo di voler esprimere T in funzione della dimensione dell'istanza, invece che dell'istanza stessa:

$$|x| = |y| \text{ non implica } T(x) = T(y)$$

Come definire T
sulla dimensione?

11

Quale ingresso di dimensione n ?

Supponiamo di voler esprimere T in funzione della dimensione dell'istanza, invece che dell'istanza stessa:

Distinguiamo allora i casi.

$$T_{\text{migliore}}(n) = \min \{T(x) : |x| = n\}$$

caso migliore: x t.c. $T_{\text{migliore}}(|x|) = T(x)$

Minimo (A, j, k) : quando il minimo è $A[j]$

Insert-Sort: vettore non decrescente

12

Quale ingresso di dimensione n ?

Supponiamo di voler esprimere T in funzione della dimensione dell'istanza, invece che dell'istanza stessa:

Distinguiamo allora i casi.

$$T_{\text{peggiore}}(n) = \max \{T(x) : |x| = n\}$$

caso peggiore: x t.c. $T_{\text{peggiore}}(|x|) = T(x)$

Minimo (A, j, k) : quando $A[j..k]$ è ordinato in senso decrescente

Insert-Sort: vettore decrescente

13

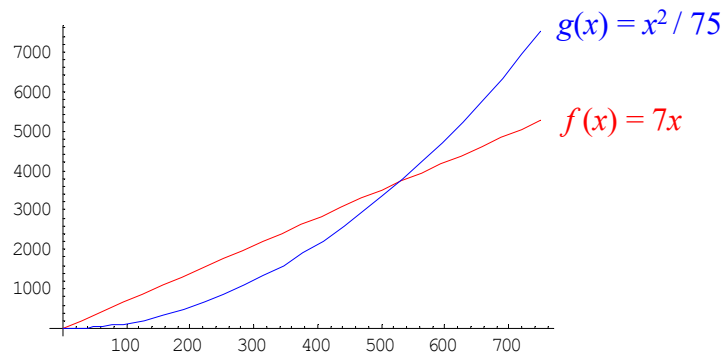
Come confrontare funzioni

- sappiamo che il tempo di calcolo non è un numero ma una funzione
- per confrontare il tempo di calcolo di due algoritmi dobbiamo confrontare tra loro funzioni

Come è possibile confrontare tra loro funzioni che hanno infiniti valori?

14

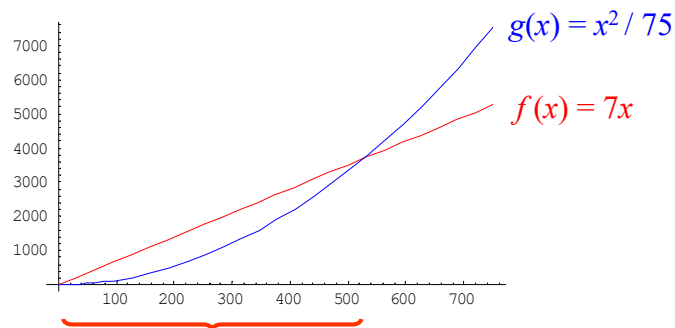
Come confrontare funzioni



$f(x) < g(x)$ quando $x > 525$

15

Trascuriamo un numero finito di casi



Se trascuriamo un
numero finito di casi
allora $g(x) > f(x)$

Sempre che non
siano proprio
quelli che ci
interessano!

16

Quanto contano le costanti?

- tempo di calcolo per un algoritmo implementato sul computer
 $C_1: T(n) = 2^n$
- D : dimensione massima di un problema trattabile col computer

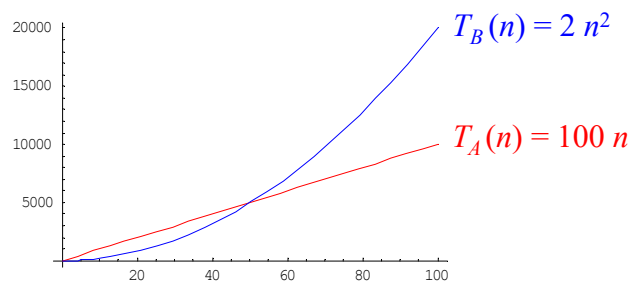
Costruiamo un computer C_2 1000 volte più veloce!

Quanto vale D' se
 $2^{D'}/1000 = 2^D$?

$$\begin{aligned}\frac{2^{D'}}{1000} &= 2^D \\ 2^{D'} &= 1000 \cdot 2^D \\ D' &= \log_2 1000 \cdot 2^D = \\ D' &= \log_2 1000 + \log_2 2^D \approx 10 + D\end{aligned}$$

17

Quanto contano le costanti?



- l'algoritmo A è migliore dell'algoritmo B per $n > 50$
- rimpiazzare B con A è meglio che raddoppiare la velocità del computer:

$$\frac{T_B(100)}{T_A(100)} = 2 \quad \frac{T_B(1000)}{T_A(1000)} = 20$$

18

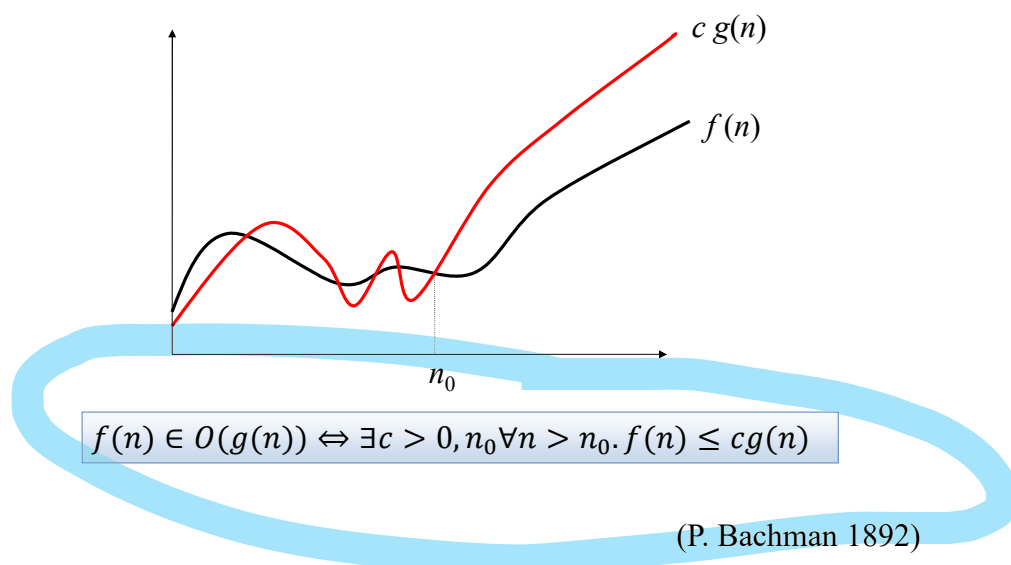
→ miglior 2

Le costanti contano poco perché

- moltiplicando per una costante il tempo di calcolo, la massima dimensione trattabile cambia poco
- il tipo di crescita di una funzione non dipende dalla costante moltiplicativa
- la stima esatta delle costanti è molto difficile in pratica

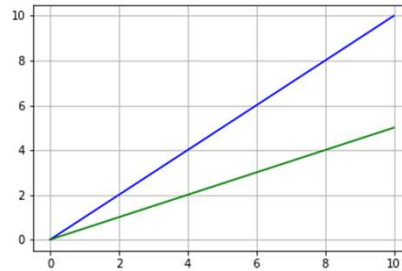
19

Ordini di grandezza: O-grande



20

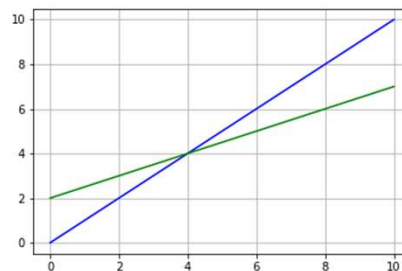
O-grande, esempio I



- $f(n) = \frac{1}{2} \cdot n, g(n) = n$
- $f(n) \in O(g(n))$? **Si!**
- $f(n) \leq g(n)$ con qualunque $n \geq 0$
- con $c = 1, n_0 = 0$ abbiamo $\forall n > n_0. f(n) \leq cg(n)$

21

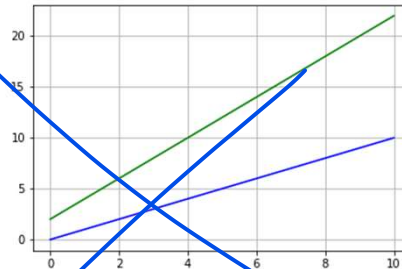
O-grande, esempio II



- $f(n) = \frac{1}{2} \cdot n + 2, g(n) = n$
- $f(n) \in O(g(n))$? **Si!**
- $f(n) \leq g(n)$ con qualunque $n \geq 4$
- con $c = 1, n_0 = 4$ abbiamo $\forall n > n_0. f(n) \leq cg(n)$

22

O-grande, esempio III

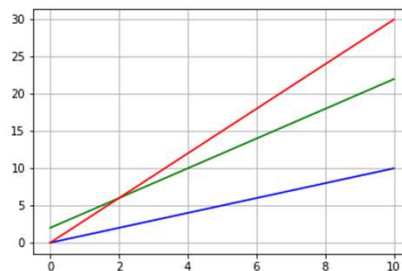


- $f(n) = 2n + 2, g(n) = n$
- $f(n) \in O(g(n))$?
- $f(n) > g(n)$ con qualunque $n \geq 0$
- "No." potrebbe sembrare la risposta giusta ma non è così

23

③

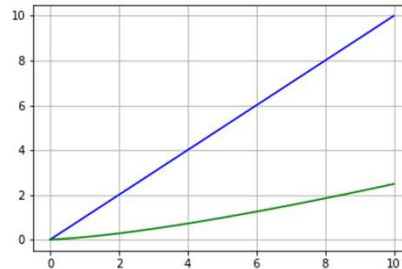
O-grande, esempio III (cont.)



- $f(n) = 2n + 2, g(n) = n, 3 \cdot g(n) = 3n$
- $f(n) \in O(g(n))$? **Sì!**
- $f(n) \leq 3g(n)$ con qualunque $n \geq 2$
- con $c = 3, n_0 = 2$ abbiamo $\forall n > n_0. f(n) \leq cg(n)$

24

O-grande, esempio IV



- $f(n) = n/10 \cdot \log(n+2)$, $g(n) = n$
- $f(n) \in O(g(n))$?
- la figura implica $f(n) \leq g(n)$ con qualunque $0 \leq n \leq 10$
- verrebbe da rispondere "Sì." ma non è così

25

O-grande, esempio IV (cont.)

- $f(n) = n/10 \cdot \log(n+2)$, $g(n) = n$
- per dimostrare $f(n) \in O(g(n))$ servirebbe una costante c con la quale oltre qualche n_0 (cioè $\forall n > n_0$)

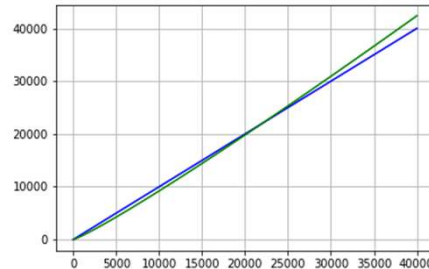
$$n/10 \cdot \log(n+2) \leq c \cdot n$$

$$1/10 \cdot \log(n+2) \leq c$$

- tale costante c non esiste e quindi
- $f(n) \in O(g(n))$? **No!**

26

O-grande, esempio IV (cont.)

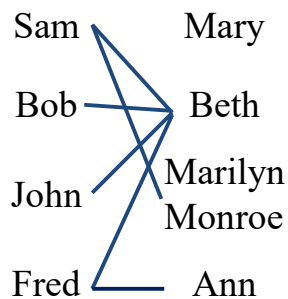


- $f(n) = \frac{n}{10} \cdot \log(n+2), g(n) = n$
- si vede anche graficamente ma bisogna plottare per grandi valori di n
- (non a caso $n \cdot \log n$ si chiama anche "quasi lineare")

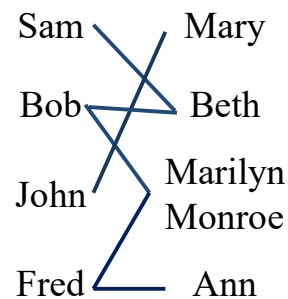
27

Ordine degli quantificatori conta!

$\exists g, \forall b, \text{loves}(b, g)$

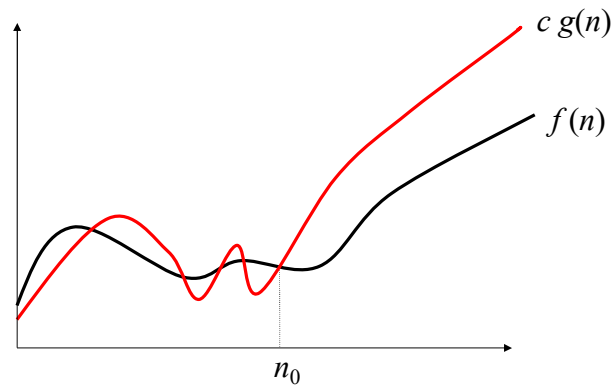


$\forall b, \exists g, \text{loves}(b, g)$



28

Ordini di grandezza: O-grande



$$f(n) \in O(g(n)) \Leftrightarrow \exists c > 0, n_0 \forall n > n_0. f(n) \leq c g(n)$$

(P. Bachman 1892)

29

Le costanti non contano

Per ogni f, g , e per ogni costante $c > 0$
 $f(n) \in O(g(n)) \Leftrightarrow c \cdot f(n) \in O(g(n))$

\Rightarrow per def. della O esistono $d > 0$ e n_0 tali che $f(n) \leq d \cdot g(n)$
 per ogni $n > n_0$

posto $b = c \cdot d > 0$ abbiamo

$$c \cdot f(n) \leq c \cdot d \cdot g(n) = b \cdot g(n) \text{ per ogni } n > n_0$$

e quindi $c \cdot f(n) \in O(g(n))$

\Leftarrow dimostrazione analoga

30

Le costanti non contano

Per ogni f, g , e per ogni costante $c > 0$
 $f(n) \in O(g(n)) \Leftrightarrow f(n) \in O(c \cdot g(n))$

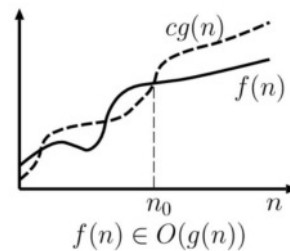
(con dimostrazione analoga a quella sul lucido precedente)

$O(1) = O(k)$ per ogni k e
 $O(1)$ è l'insieme delle funzioni superiormente limitate

31

Esempio pratico di O-grande

$$3n^2 + 7n + 8 \in O(n^2)$$



$$\exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$$

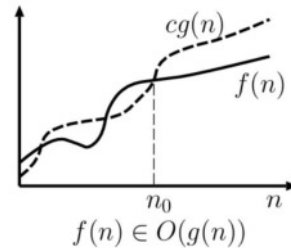
↑
?

$$3n^2 + 7n + 8 \leq c \cdot n^2$$

32

Esempio pratico di O-grande

$$3n^2 + 7n + 8 \in O(n^2)$$



$$\exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$$

↑
4

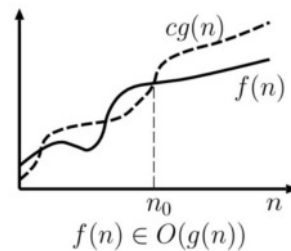
↑
?

$$3n^2 + 7n + 8 \leq 4 \cdot n^2$$

33

Esempio pratico di O-grande

$$3n^2 + 7n + 8 \in O(n^2)$$



$$\exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$$

↑
4

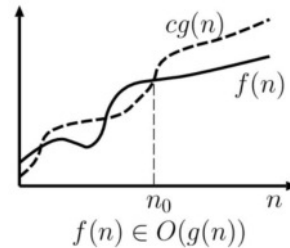
↑
1

$$3 \cdot 2^2 + 7 \cdot 2 + 8 \not\leq 4 \cdot 2^2$$

34

Esempio pratico di O-grande

$$3n^2 + 7n + 8 \in O(n^2)$$



$$\exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$$

↑
4

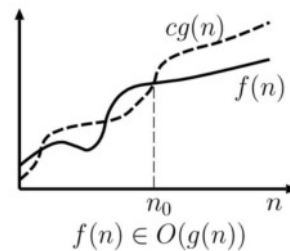
↑
8

$$\begin{aligned} 3 \cdot 8^2 + 7 \cdot 8 + 8 &\leq 4 \cdot 8^2 \\ \underbrace{3 \cdot 8^2 + (7+1) \cdot 8}_{314} &= 4 \cdot 8^2 \end{aligned}$$

35

Esempio pratico di O-grande

$$3n^2 + 7n + 8 \in O(n^2)$$



$$\exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$$

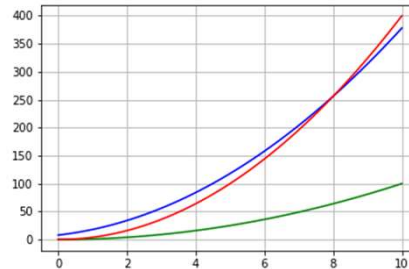
↑
4

↑
8

$$\begin{aligned} 3 \cdot 9^2 + 7 \cdot 9 + 8 &\leq 4 \cdot 9^2 \\ \underbrace{314}_{314} &\quad \quad \quad 324 \end{aligned}$$

36

Esempio pratico di O-grande



- $f(n) = 3n^2 + 7n + 8$, $g(n) = n^2$, $4 \cdot g(n) = 4n^2$

37

Ordini di grandezza: O-grande

$$n \geq 8 \Rightarrow 3n^2 + 7n + 8 \leq 4n^2$$

La tesi equivale a $7n + 8 \leq 4n^2 - 3n^2 = n^2$

Dividendo per n : $7 + \frac{8}{n} \leq n$

Ma $n \geq 8 \Rightarrow \frac{8}{n} \leq \frac{8}{8} = 1$ e quindi $7 + \frac{8}{n} \leq 7 + 1 = 8 \leq n$

38

Ordini di grandezza di polinomi

Teorema: Se $p(n)$ è un polinomio di grado k allora $p(n) \in O(n^k)$.

$$p(n) = \sum_{i=0}^k a_i n^i \text{ con } a = \max\{a_i : 0 \leq i \leq k\}:$$

$$p(n) \leq \sum_{i=0}^k a n^i = a \cdot \sum_{i=0}^k n^i \leq a(k+1)n^k \quad (\forall n > 1)$$

dunque con $c = a(k+1)$ e $n_0 = 1$

segue $p(n) \in O(n^k)$

I termini di grado inferiore si possono ignorare.

39

Definizione equivalente

Teorema:

$$\begin{aligned} f(n) \in O(g(n)) \\ \Leftrightarrow \\ \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \text{ esiste e } 0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \end{aligned}$$

La definizione equivalente fornita dal teorema precedente è spesso utile per dimostrare facilmente se $f(n) \in O(g(n))$ date $f(n)$ e $g(n)$.

40

Ancora limiti

Teorema:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

$$\Leftrightarrow$$

$$f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$$

- per esempio, per le funzioni n^2 e n^3 si ha
 $n^2 \in O(n^3) \wedge n^3 \notin O(n^2)$

41

Ordini di grandezza di polinomi

Teorema: Se $p(n)$ è un polinomio di grado $h > k$ e $a_h > 0$, allora $p(n) \notin O(n^k)$.

Con la def. equivalente precedente da

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{a_0 + a_1 n + \dots + a_h n^h}{n^k} = \infty$$

dunque $p(n) \notin O(n^k)$

Tutto ciò che conta in un polinomio è il grado.

42

Ordini di grandezza di polinomi

Teorema: Se $p(n)$ è un polinomio di grado $h > k$ e $a_h > 0$, allora $p(n) \notin O(n^k)$.

Esempio:

$$p(n) = 3n^3 + 5n + 18 \quad (h = 3)$$

implica

$$p(n) \text{ non è } O(n^2) \quad (k = 2)$$

Tutto ciò che conta in un polinomio è il grado.

43

Base dei logaritmi

$$O(\log_a n) = O(\log_b n), \quad a, b > 1$$

$$\log_a n = \frac{\log_b n}{\log_b a} = \frac{1}{\log_b a} \log_b n$$

Quindi scriviamo semplicemente

$$O(\log n)$$

44

Inclusioni

$$O(1) \subset O(\log n)$$

$\log n$ è superiormente illimitata, mentre
 $f(n) \in O(1) \Rightarrow \exists c > 0, n_0 \forall n > n_0. f(n) \leq c$

$$O(\log n) \subset O(n)$$

$\log_2 n \leq n \Leftrightarrow n = 2^{\log_2 n} \leq 2^n$ perché $n < 2^n$ per ogni
 $n > 0$,

$$O(n) \subset O(n \log n)$$

$$n > 2 \Rightarrow \log_2 n > 1 \Rightarrow n \log_2 n > n$$

(sul lucido si leggono ragionamenti informali, per avere una dimostrazione formale bisognerebbe trattare qualche dettaglio in più)

45

Inclusioni

$$O(n^p) \subset O(2^n)$$

Segue da

$$\lim_{n \rightarrow \infty} \frac{n^p}{2^n} = 0$$

grazie al teorema

Teorema:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

$$\Leftrightarrow$$

$$f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$$

46

Inclusioni

$$O(n^2) \subset O(2^n)$$

Segue che

Le funzioni esponenziali crescono più velocemente delle polinomiali.

47

Base di una potenza

$$O(2^n) \neq O(3^n)$$

Segue da

$$\lim_{n \rightarrow \infty} \frac{2^n}{3^n} = \lim_{n \rightarrow \infty} \left(\frac{2}{3}\right)^n = 0 \quad \text{e} \quad \lim_{n \rightarrow \infty} \frac{3^n}{2^n} = 1$$

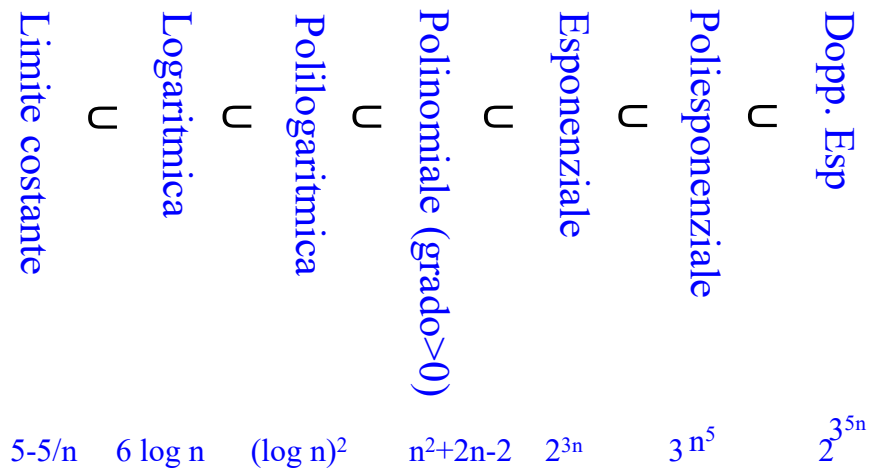
che $3^n \notin O(2^n) \wedge 3^n \in O(3^n)$ che implica $O(2^n) \neq O(3^n)$

Nel caso di una potenza la base conta!

(quindi anche $O(2^n) \neq O(2^{2n})$ perché $2^{2n} = 4^n$)
(e cmq $O(2^n) \subset O(3^n)$)

48

Classificazione delle funzioni



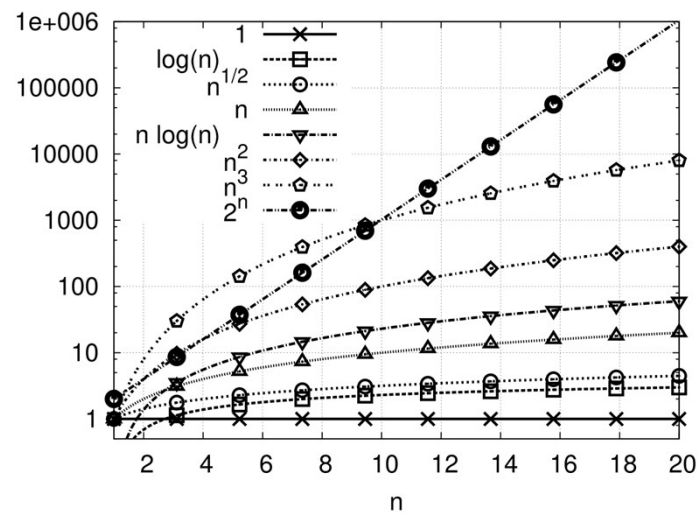
49

Funzioni ordinate per velocità di crescita

n	log n	\sqrt{n}	n	n log n	n^2	n^3	2^n
2	1	1,41	2	2	4	8	4
4	2	2	4	8	16	64	16
8	3	2,83	8	24	64	512	256
16	4	4	16	64	256	4.096	65.536
32	5	5,66	32	160	1.024	32.768	4.294.967.296
64	6	8	64	384	4.096	262.144	$1,84 \times 10^{19}$
128	7	11,31	128	896	16.384	2.097.152	$3,40 \times 10^{38}$
256	8	16	256	2.048	65.536	16.777.216	$1,15 \times 10^{77}$
512	9	22,63	512	4.608	262.144	134.217.728	$1,34 \times 10^{154}$
1.024	10	32	1.024	10.240	1.048.576	1.073.741.824	$1,79 \times 10^{308}$

50

Funzioni ordinate per velocità di crescita



51

Algebra informale di O-grande

Spesso leggiamo eguaglianze del tipo

$$\frac{1}{4}n^2 = O(n^2) \quad \frac{1}{2}n^2 + n = O(n^2) + O(n) = O(n^2)$$

Prese alla lettera conducono ad assurdità: $\frac{1}{2}n^2 + n = \frac{1}{4}n^2$

$f(n) = O(g(n))$ si interpreta come
un'equazione "a senso unico", con cui
rimpiazziamo una funzione con il suo
ordine di grandezza in senso O-grande

52

Algebra informale di O-grande

Definiamo:

$$f(n) + O(g(n)) = \{h : \exists g' \in O(g(n)) . h(n) \in O(f(n) + g'(n))\}$$

$$f(n) \cdot O(g(n)) = \{h : \exists g' \in O(g(n)) . h(n) \in O(f(n) \cdot g'(n))\}$$

Allora ne segue:

$$f(n) + O(g(n)) = O(f(n) + g(n))$$

$$f(n) \cdot O(g(n)) = O(f(n) \cdot g(n)).$$

Similmente definiamo:

$$O(f(n)) + O(g(n)) =$$

$$\{h : \exists f' \in O(f(n)) \exists g' \in O(g(n)) . h(n) \in O(f'(n) + g'(n))\},$$

ed $O(f(n)) \cdot O(g(n))$ analogamente.

53

Algebra informale di O-grande

Ne deriva:

$$f(n) = O(f(n))$$

$$c \cdot O(f(n)) = O(f(n)) \quad c \text{ costante}$$

$$O(f(n)) + O(g(n)) = O(f(n) + g(n))$$

$$O(f(n)) + O(f(n)) = O(f(n))$$

$$f(n) \in O(g(n)) \Rightarrow O(f(n)) + O(g(n)) = O(g(n))$$

$$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$$

Perciò ad esempio:

$$O(2) = O(1) + O(1) = O(1) \quad \text{ma}$$

$$n \cdot O(1) = O(n) \quad (n \text{ è una variabile!})$$

54

Una semplice applicazione

```

MatriceIdentità (Matrice A, int n)
// Post. A è la matrice identità nxn
{ for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        A[i][j] = 0;
    for (int i = 1; i <= n; i++)
        A[i][i] = 1;
}

```

Annotations in the original image:

- `A[i][j] = 0;` is annotated with `// T1`.
- The inner loop `for (int j = 1; j <= n; j++)` is grouped with `A[i][j] = 0;` by a green brace labeled `T2`.
- The outer loop `for (int i = 1; i <= n; i++)` is grouped with the inner loop by a green brace labeled `T3`.
- `A[i][i] = 1;` is annotated with `// T4`.
- The inner loop `for (int i = 1; i <= n; i++)` is grouped with `A[i][i] = 1;` by a green brace labeled `T5`.

$$T_1(n) \leq c_1 = O(1) \qquad T_2(n) = n \cdot T_1(n) = n \cdot O(1) = O(n)$$

$$T_3(n) = n \cdot T_2(n) = n \cdot O(n) = O(n^2) \qquad T_4(n) \leq c_2 = O(1)$$

$$T_5(n) = n \cdot T_4(n) = n \cdot O(1) = O(n)$$

$$T(n) = T_3(n) + T_5(n) = O(n^2) + O(n) = O(n^2 + n) = O(n^2)$$

55

Confini “stretti”?

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

Naturalmente si tratta di un'asserzione vera, ma dice poco perché il confine superiore 2^n **non è stretto**.

O è utilizzabile per fornire limiti asintotici **superiori**.

Come facciamo esprimere che un limite sia **stretto**?
Come facciamo esprimere limiti **inferiori**?

56

Definizione di O , Ω , Θ

O , limite (o confine) asintotico superiore:

$$f(n) \in O(g(n)) \Leftrightarrow \exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$$

Ω , limite (o confine) asintotico inferiore:

$$f(n) \in \Omega(g(n)) \Leftrightarrow \exists c > 0, n_0 \forall n > n_0. cg(n) \leq f(n)$$

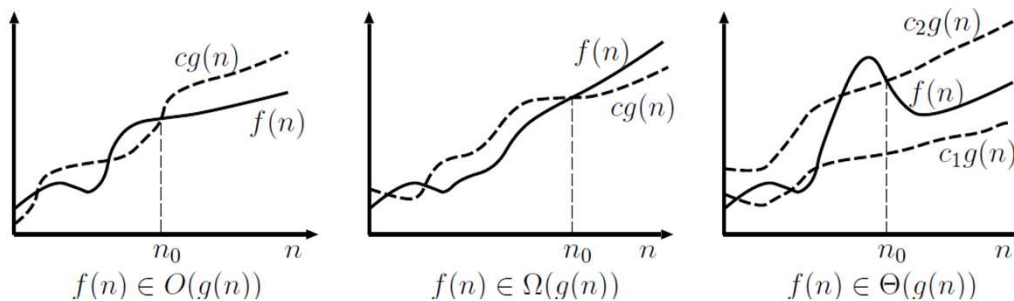
Θ , limite (o confine) asintotico sia inferiore sia superiore
(limite asintotico stretto):

$$\begin{aligned} f(n) \in \Theta(g(n)) \\ \Leftrightarrow \\ \exists c_1 > 0, c_2 > 0, n_0 \forall n > n_0. c_1 g(n) \leq f(n) \leq c_2 g(n) \end{aligned}$$

57

Notazione asintotica

graficamente:



le definizioni implicano direttamente che

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

58

O, Ω, Θ a parole

O : $f(n) \in O(g(n)) \Leftrightarrow \exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$

" $f(n)$ cresce al più come $g(n)$ "

Ω : $f(n) \in \Omega(g(n)) \Leftrightarrow \exists c > 0, n_0 \forall n > n_0. cg(n) \leq f(n)$

" $f(n)$ cresce almeno come $g(n)$ "

Θ : $f(n) \in \Theta(g(n)) \Leftrightarrow \exists c_1 > 0, c_2 > 0, n_0 \forall n > n_0. c_1g(n) \leq f(n) \leq c_2g(n)$

" $f(n)$ cresce come $g(n)$ "

(per tutti e tre "trascurando costanti moltiplicative e un numero finito di casi")

59

O-grande ed o-piccolo

O : $f(n) \in O(g(n)) \Leftrightarrow \exists c > 0, n_0 \forall n > n_0. f(n) \leq cg(n)$

o : $f(n) \in o(g(n)) \Leftrightarrow \forall c > 0 \exists n_0 \forall n > n_0. f(n) \leq cg(n)$

Se $f(n) \in o(g(n))$ allora f è un **infinitesimo** di g , quindi g non è un confine superiore "stretto" di f .

60

Definizione equivalente di O , Ω , Θ

Teorema:

$$0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \Leftrightarrow f(n) \in O(g(n))$$

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq \infty \Leftrightarrow f(n) \in \Omega(g(n))$$

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \Leftrightarrow f(n) \in \Theta(g(n))$$

La definizione equivalente fornita dal teorema è spesso utile per dimostrare facilmente la relazione fra $f(n)$ e $g(n)$.

61

Esercizi

- siano

$$f_1(n) = 3n^2 + 10n - 2$$

$$f_2(n) = \log_2 n + \frac{1}{4}n^{\frac{1}{2}}$$

$$f_3(n) = \left(\frac{3}{2}\right)^n + n^3 + 10$$

$$f_4(n) = n \log_2 n + 5n + 2$$

- quali proposizioni sono vere e quali false?

$$f_1(n) \in O(n^3), f_1(n) \in O(n), f_1(n) \in \Omega(n^2), f_1(n) \in \Theta(n^3), f_1(n) \in \Theta(n^2)$$

$$f_2(n) \in \Theta(\log n), f_2(n) \in \Theta(n^{1/2}), f_2(n) \in O(n), f_2(n) \in \Omega(n^{1/4})$$

$$f_3(n) \in O(10), f_3(n) \in \Omega(10), f_3(n) \in \Theta((3/2)^n + 10n^3), f_3(n) \in \Omega(n^3)$$

$$f_4(n) \in \Omega(n), f_4(n) \in \Omega(n \log n), f_4(n) \in \Theta(n \log n)$$

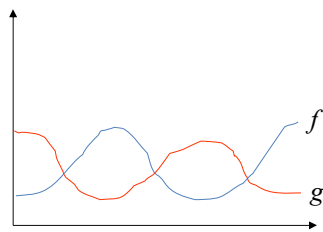
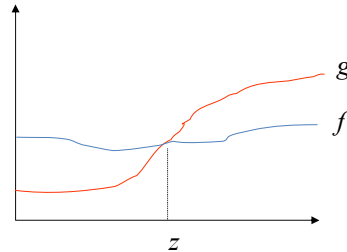
- stabilire per ciascuna funzione la sua ordine di grandezza, cioè la funzione più semplice $g_i(n)$ tale che $f_i(n) \in \Theta(g_i(n))$ con $i = 1, 2, 3, 4$

62

Confronto asintotico tra funzioni

$$f \leq g \Leftrightarrow \forall^\infty x. f(x) \leq g(x)$$

dove $\forall^\infty x. P(x) \Leftrightarrow \exists z \forall x \geq z. P(x)$.



$$\neg f \leq g \Leftrightarrow \exists^\infty x. \neg f(x) \leq g(x)$$

dove $\exists^\infty x. P(x) \Leftrightarrow \forall z \exists x \geq z. P(x)$.

Oss. $\forall^\infty x. P(x) \Leftrightarrow \neg \exists^\infty x. \neg P(x)$

$$\exists^\infty x. P(x) \Leftrightarrow \neg \forall^\infty x. \neg P(x)$$

63

Funzioni inconfrontabili

Ci sono funzioni inconfrontabili asintoticamente:

$$f(x) = \begin{cases} 1 & \text{se } x \text{ pari} \\ 0 & \text{se } x \text{ dispari} \end{cases} \quad g(x) = \begin{cases} 0 & \text{se } x \text{ pari} \\ 1 & \text{se } x \text{ dispari} \end{cases}$$

Con O :

$$f(x) \notin O(g(x)) \wedge g(x) \notin O(f(x))$$

e anche con la notazione introdotta prima:

$$f(x) \not\leq g(x) \wedge g(x) \not\leq f(x)$$

64

Complessità di un problema

Somma 17. Dato un vettore di n interi positivi decidere se ne contiene due la cui somma sia 17.

65

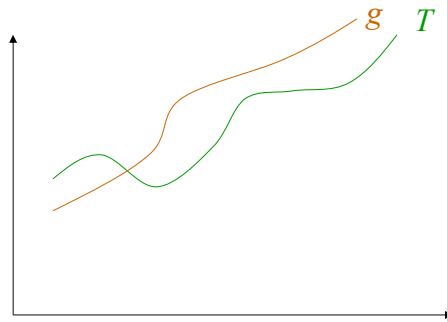
Complessità di un problema

Qual è un tempo di calcolo
sufficiente alla risoluzione del
problema “Somma 17”?

Confine superiore alla complessità di un problema:
un confine superiore per il tempo di calcolo (nel caso peggiore) di **un** algoritmo che risolve il problema.

66

Complessità di un problema



Se **un** algoritmo che risolve il problema ha tempo di calcolo $T(n)$ e $T(n) \in O(g(n))$ allora $g(n)$ è un confine superiore per la complessità del problema (in senso O).

Confine superiore alla complessità di un problema: un confine superiore per il tempo di calcolo (nel caso peggiore) di **un** algoritmo che risolve il problema.

67

Complessità di un problema

Somma 17. Dato un vettore di n interi positivi decidere se ne contiene due la cui somma sia 17.

```
bool Somma17 (int v[], int n)
{
    bool b = false;
    for (int i = 0; i < n; i++)
        for (int j = i+1; j < n; j++)
            if (v[i] + v[j] == 17) b = true;
    return b;
}
```

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \Rightarrow T_{\text{Somma17}}(n) \in O(n^2)$$

68

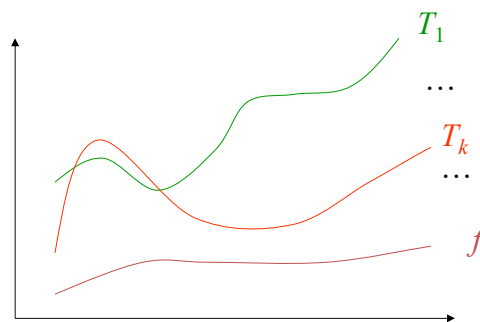
Complessità di un problema

Qual è un tempo di calcolo
necessario alla risoluzione del
problema “Somma 17”?

Confine inferiore alla complessità di un problema:
un confine inferiore per i tempi di calcolo (nel caso
peggiore) di **tutti** gli algoritmi che risolvono il problema

69

Complessità di un problema



Se **tutti** gli algoritmi T_i
che risolvono il
problema hanno tempo
di calcolo $T_i(n) \in$
 $\Omega(f(n))$ allora $f(n)$ è
un confine inferiore per
la complessità del
problema (in senso Ω).

Confine inferiore alla complessità di un problema:
un confine inferiore per i tempi di calcolo (nel caso
peggiore) di **tutti** gli algoritmi che risolvono il problema

70

Confini banali

- **Dimensione del input:** quando è necessario esaminare tutti i dati in ingresso.
Es. La moltiplicazione di due matrici quadrate di ordine n richiede l'ispezione di $2n^2 \in \Omega(n^2)$ entrate.
- **Dimensione del output:** numero di elementi da produrre in output.
Es. Se la soluzione è un vettore di n elementi allora tempo di calcolo richiesto è $\Omega(n)$.
- **Eventi contabili:** quando c'è un evento la cui ripetizione un numero di volte sia necessaria alla soluzione del problema.
Es. La determinazione del massimo tra n elementi richiede $n - 1 \in \Omega(n)$ confronti, in cui altrettanti elementi non massimi risultino minori.

71

Complessità di un problema

Contando il numero n degli elementi del vettore e poiché ognuno di essi deve essere considerato almeno una volta, un confine inferiore alla complessità del problema **Somma17** è $\Omega(n)$.

Questo è un esempio del criterio della "dimensione dei dati".

72

Una soluzione $O(n)$ per Somma17

1. Dato il vettore v , calcoliamo l'insieme $C = \{i \mid i \leq 17 \text{ e } i \text{ è presente in } v\}$
2. dato che v ha solo interi positivi, la risposta sarà true se e solo se esistono $i, j \in C$ tali che $i + j = 17$

73

Una soluzione $O(n)$ per Somma17

```
bool Somma17_veloce (int v[], int n)
{
    bool c[18]; int i, j;
    for(i = 0; i < 18; i++)
        c[i] = false;
    for(i = 0; i < n; i++)
        if (v[i] <= 17) c[v[i]] = true;
    for(i = 0, j = 17;
        i < j && !(c[i] && c[j]); i++, j--);
    return i < j;
}
```

74

Una soluzione $O(n)$ per Somma17

$v[5] =$

1	9	25	2	8
---	---	----	---	---

$c[18] =$

F	T	T	F	F	F	F	F	T	T	F	F	F	F	F	F	F	F
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

75

Una soluzione $O(n)$ per Somma17

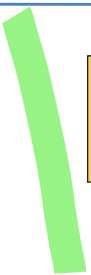
```
bool Somma17_veloce (int v[], int n)
{
    bool c[18]; int i, j;
    for (i = 0; i < 18; i++)
        c[i] = false;
    for (i = 0; i < n; i++)
        if (v[i] <= 17) c[v[i]] = true;
    for (i = 0, j = 17; i < j && !(c[i] && c[j]);
        i++, j--);
    return i < j;
}
```

Questa soluzione è ottima perché $O(n)$ essendo $\Omega(n)$ un confine inferiore per il problema.

$T_{\text{Somma17_veloce}}(n) \in O(18 + n + 9 + 1) \in O(n)$
 (tempo di esecuzione del primo ciclo è proporzionale a 18,
 tempo di esecuzione del secondo ciclo è proporzionale a n ,
 tempo di esecuzione del terzo ciclo è proporzionale a $18/2$,
 tempo di esecuzione del return è proporzionale a 1)

76

Complessità di un problema



Un algoritmo con tempo di calcolo $T(n) \in \mathcal{O}(g(n))$ è ottimo per un certo problema se $g(n)$ è un confine inferiore alla complessità del problema in termini di Ω .