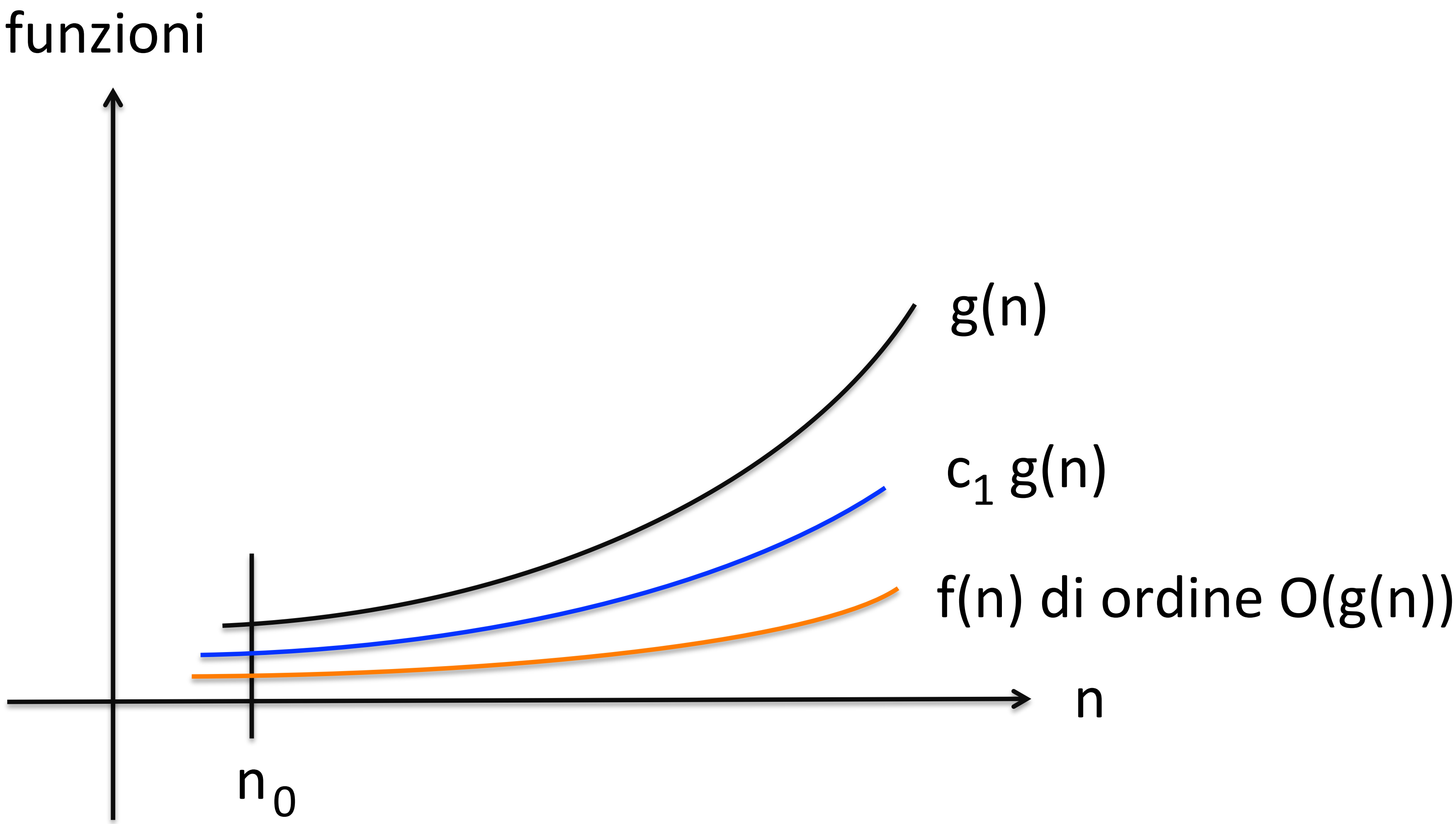
Informatica per le Biotecnologie
Algoritmica Lezione 2

ORDINI DI GRANDEZZA DELLE FUNZIONI

f(n)
La variabile indipendente $n > 0$ indica la dimensione dei dati di un problema.
La funzione f di cui si studia l'ordine di grandezza indica la complessità di un algoritmo di risoluzione.

Notazione O

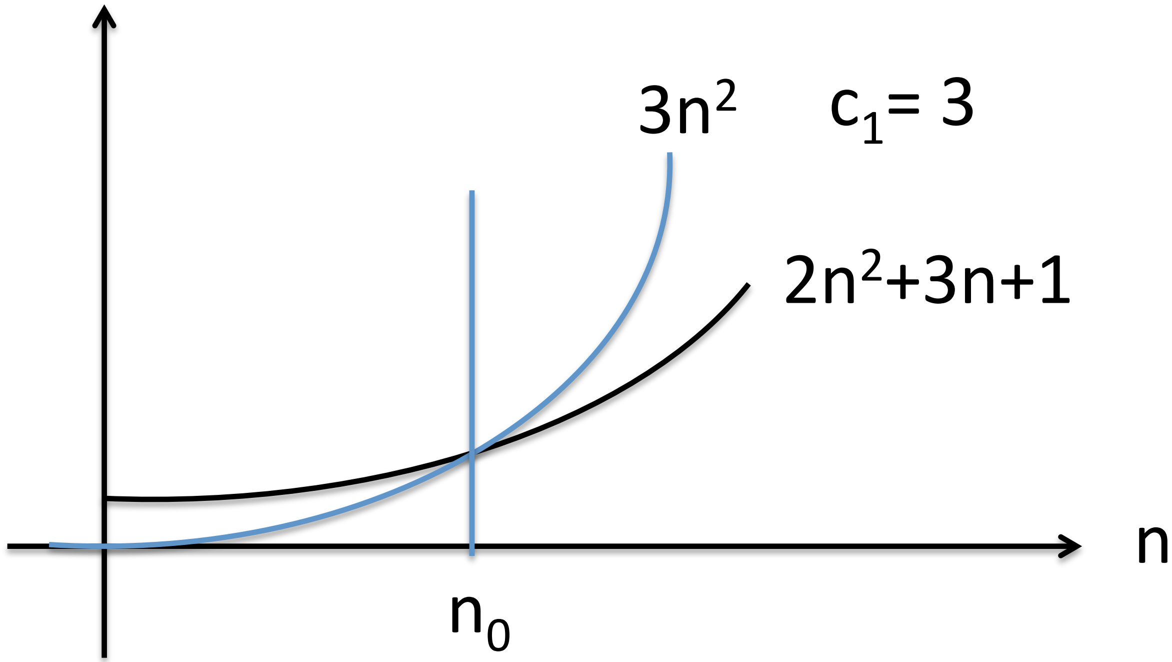
f(n) **è di ordine O**(g(n)) se esistono due costanti positive c_1, n_0, tali che $0 \le f(n) \le c_1 g(n)$ per ogni $n \ge n_0$.



Cioè, al crescere di n e a partire da un valore n_0 , la funzione $f(n)$ non sale al di sopra di $g(n)$ a meno di una costante moltiplicativa c_1:

dunque nella funzione $g(n)$ non si indicano le costanti moltiplicative

Per esempio $f(n) = 2n^2+3n+1$ è di ordine $\mathbf{O}(n^2)$
(contano solo i termini di grado massimo)



Ma la $2n^2+3n+1$ è anche di ordine $\mathbf{O}(n^3)$!

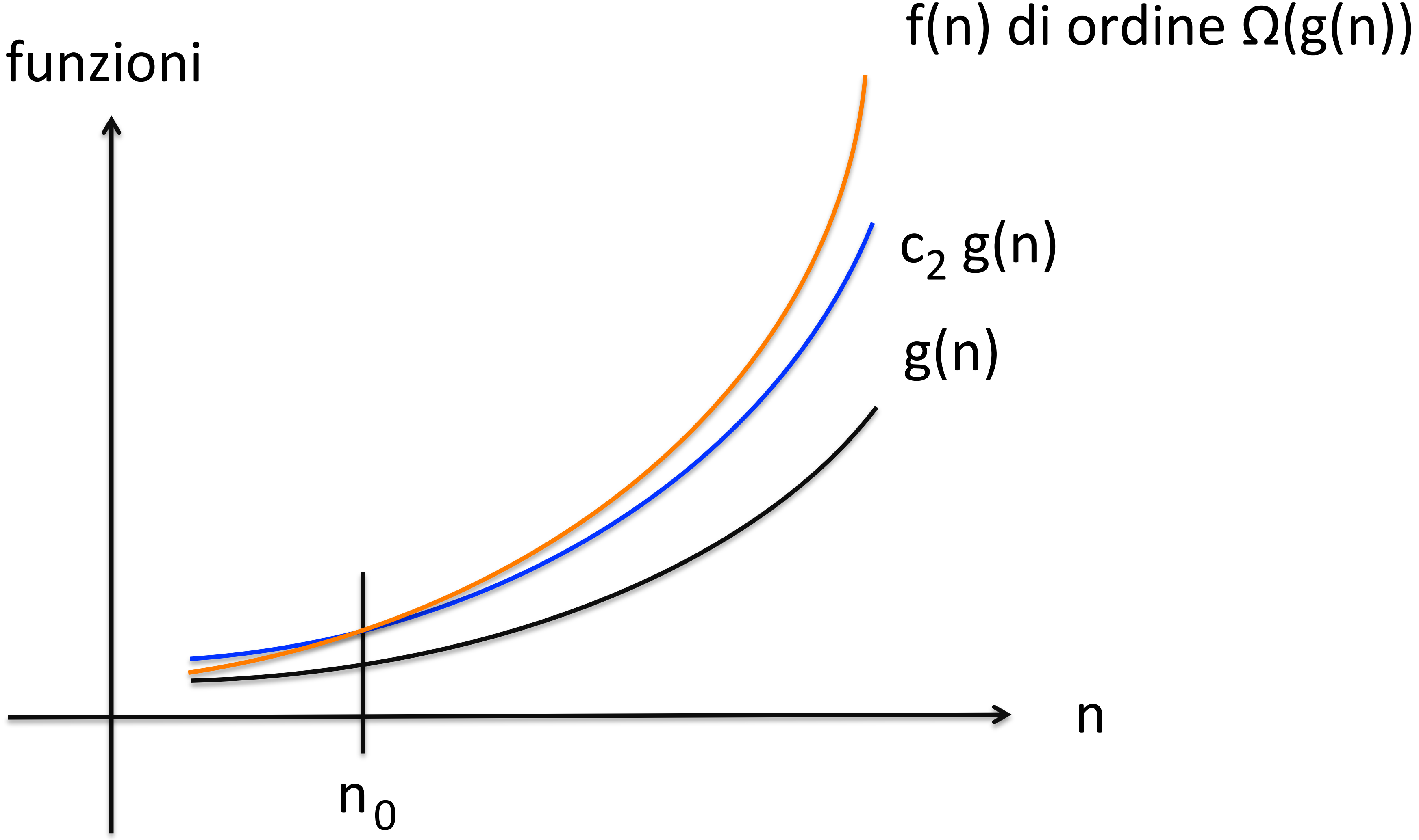
La notazione O(g(n)) si impiega per indicare il tempo di un algoritmo:

- di cui **non** si conosce compiutamente il comportamento, ma che si sa che non può superare g(n);

- oppure che **non** si comporta allo stesso modo per tutti gli insiemi di dati di dimensione n che gli si presentano, ma per alcuni richiede tempo proporzionale a g(n)), per altri meno.

Notazione Ω

f(n) **è di ordine Ω**(g(n)) se esistono due costanti positive c_2, n_0, tali che $0 \le c_2 g(n) \le f(n)$ per ogni $n \ge n_0$.



Cioè, al crescere di n e a partire da un valore n_0, la funzione $f(n)$ non scende al di sotto di $g(n)$ a meno di una costante moltiplicativa.

Anche qui contano solo i termini di grado massimo.

Per esempio $f(n) = 2n^2-3n+1$ è di ordine $\Omega(n^2)$, ma anche $\Omega(n)$ ecc..

La notazione Ω si impiega per indicare il limite inferiore al tempo di soluzione di un problema.

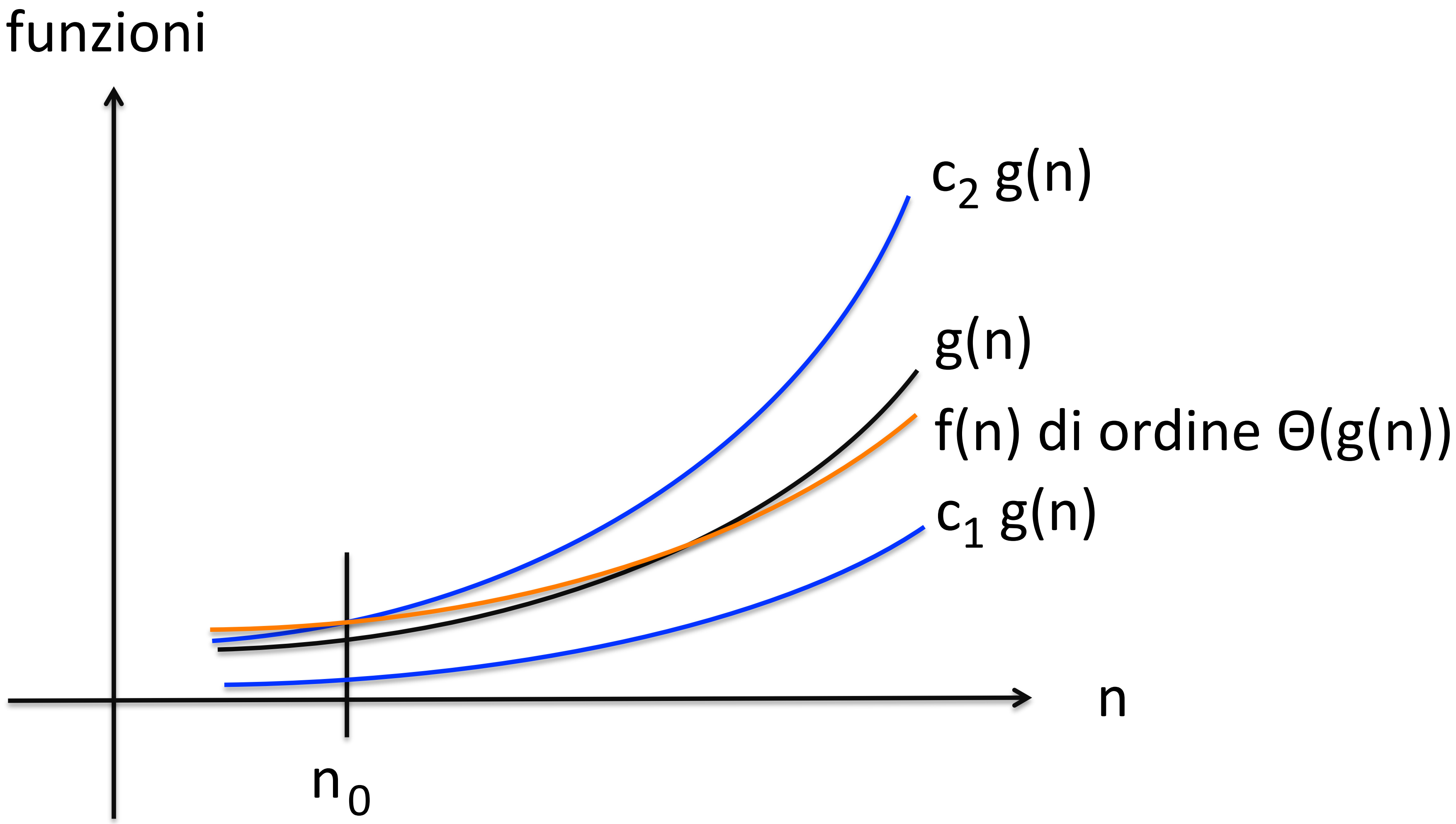
Notare l'importante differenza nell'impiego dei due ordini O e Ω. Il primo è relativo al comportamento di un particolare algoritmo di soluzione, il secondo alla natura intrinseca del problema e si applica quindi a *tutti i suoi algoritmi* di soluzione

Notazione Θ

f(n) **è di ordine** **Θ**(g(n)) se esistono tre costanti positive c_1, c_2, n_0 ,tali che $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ per ogni $n \ge n_0$.

Notazione Θ

$f(n)$ **è di ordine** $\mathbf{\Theta}(g(n))$ se esistono tre costanti positive c_1, c_2, n_0, tali che $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ per ogni $n \ge n_0$.



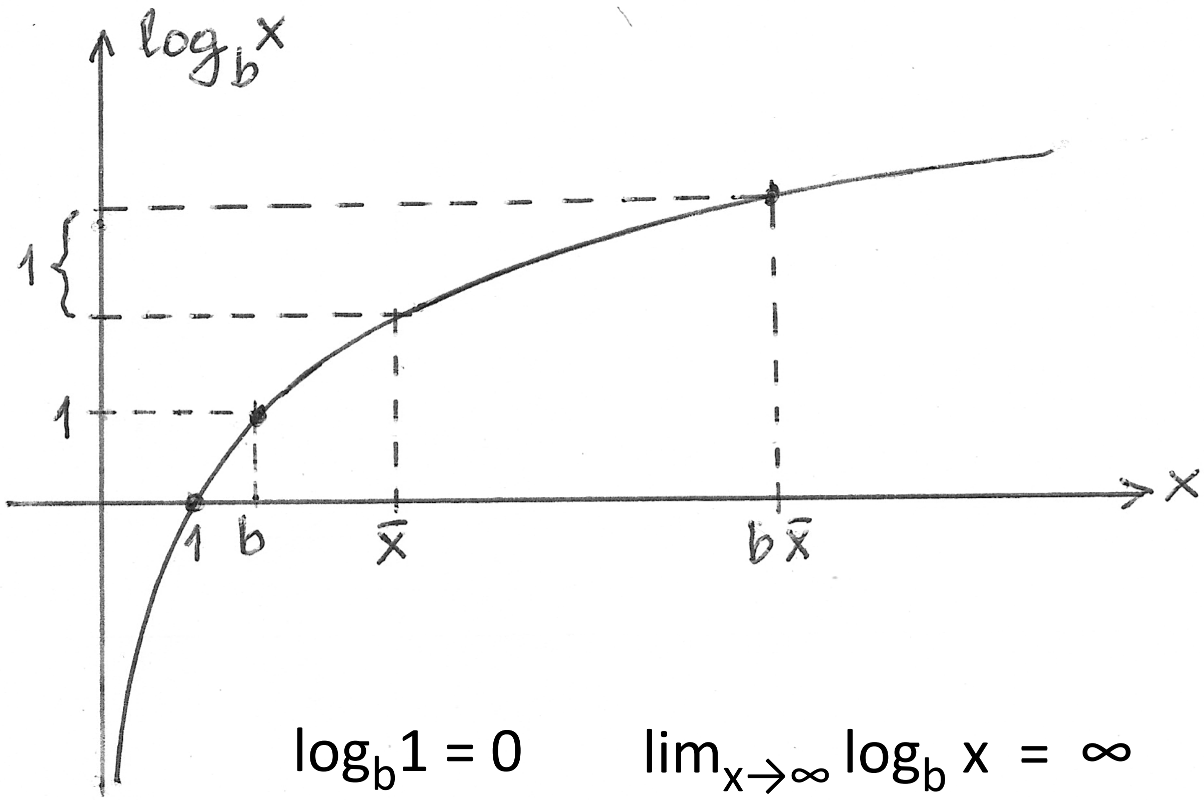
Cioè, al crescere di n e a partire da un valore n_0 , le funzioni $f(n)$ e $g(n)$ hanno lo stesso andamento a meno di costanti moltiplicative.

Contano solo i termini di grado massimo.
Per esempio $f(n) = n^2-3n+1$ è di ordine $\Theta(n^2)$, ma anche $f(n)=3n^2-5$ è di ordine $\Theta(n^2)$ perché non si considerano le costanti moltiplicative.

La somma dei primi n interi $1+2+ . . . +n = n(n+1)/2$
è di ordine $\Theta(n^2)$.

La notazione Θ si impiega per indicare il tempo di un algoritmo di cui si conosce compiutamente il comportamento e che, a pari valore di n, si comporta allo stesso modo per tutti gli insiemi di dati di dimensione n che si presentano.

LA FUNZIONE LOGARITMO $y = \log_b x$



$\log_b 1 = 0$ $\lim_{x \to \infty} \log_b x = \infty$

$\log_b b = 1$ $\lim_{x \to 0} \log_b x = -\infty$

Per definizione $y = \log_b x$ è l'esponente che bisogna dare alla base b per ottenere x:

$$\log_2 8 = 3 \leftrightarrow 2^3 = 8$$

Dunque le funzioni logaritmo ed esponenziale sono una inversa dell'altra.

In informatica hanno sono importanti le basi $b = 2$ e $b = 10$.
Il numero di Eulero $e = 2.718...$ è la base del logaritmo naturale $\ln(x)$.

Per x che tende all'infinito, $\log_b x$ tende all'infinito più lentamente di x^c per qualunque esponente $c > 0$ per quanto piccolo (in particolare per $0 < c < 1$).

Proprietà della funzione logaritmo $y = \log_b x$

$\log_b(xy) = \log_b x + \log_b y$

$\log_b(x^y) = y \cdot \log_b x$

$\log_b 1/x = - \log_b x$

$\log_b(bx) = \log_b x + 1$

Per cambiare la base di un logaritmo da b ad a si deve moltiplicare per il logaritmo $\log_a b$ tra le basi, cioè:

$$\log_a x = \log_a b \cdot \log_b x$$

Poiché il termine $\log_a b$ è una costante (cioè è indipendente da x), i logaritmi di x in basi diverse sono proporzionali tra loro.
Ciò implica che le basi dei logaritmi non si specificano negli ordini di grandezza perché i diversi logaritmi dello stesso argomento differiscono solo per una costante moltiplicativa. Scriveremo $O(\log x)$ ecc..

RAPPRESENTAZIONE E VALORE DEI NUMERI

Esprimendo i numeri naturali in una base b , con n cifre si rappresentano b^n numeri differenti (le disposizioni con ripetizione di b elementi in gruppi di n):

Esempio: i numeri di $n = 3$ cifre in base $b = 10$

000, 001, 002, ... , 999 sono in totale $10^3 = 1000$

Dunque il valore V di un numero è di ordine $O(b^n)$, cioè esponenziale nel numero di cifre n ; invrrsamente il numero di cifre di V è di ordine $O(\log_b V)$ (ovvero $O(\log V)$ omettendo la base), dunque logaritmico nel valore del numero.

Abbiamo visto che $\log_b(xy) = \log_b x + \log_b y$

```
345 × 98 = 33810
 3  + 2  =   5 cifre
```

Sicurezza di un lucchetto con display di n = 4 cifre

Impostazione 3 7 2 9 tempo $O(n)$

Effrazione: tentativo su tutte le combinazioni tempo $O(10^n)$

In crittografia si utilizzano funzioni che si calcolano in tempo polinomiale per eseguire o interpretare una comunicazione cifrata, ma si possono invertire solo in tempo esponenziale per decifrare una comunicazione senza autorizzazione

RAPPRESENTAZIONE DELL'INFORMAZIONE

In generale per rappresentare (o dare un nome a) gli N oggetti di un insieme mediante sequenze lunghe k costruite con un alfabeto di n caratteri

Per $n=1$ (alfabeto unario) almeno una sequenza deve contenere $k=N$ caratteri

Per $n=2$ (per esempio con l'alfabeto {0,1}) N oggetti

si possono rappresentare con $k = \log_2 N$ caratteri

inversamente, $N = 2^k$

con 1 byte (8 bit) si rappresentano $2^8 = 256$ oggetti

Per n generico si rappresentano N oggetti con $\log_n N$ caratteri.

Come già detto impiegando $n=10$ cifre decimali si rappresentano N numeri mediante $\log_{10} N$ cifre.

SEQUENZE, FUNZIONI E ALGORITMI

Le sequenze finite su un alfabeto finito sono numerabili, cioè possono essere poste in corrispondenza biunivoca con i numeri naturali

ALFABETO $A = \{a,b,c, \ldots,z\}$

Ordinamento canonico delle sequenze

ε

a, b, …, z

aa, ab, …, az, ba, bb, …, bz, …, za, zb, …, zz,

aaa, aab, …, aaz, aba, …, … zzz,

…….

Le sequenze infinite non sono numerabili

S_0 s_{00} s_{10} s_{20} s_{30} s_{40} s_{50} ....

S_1 s_{01} s_{11} s_{21} s_{31} s_{41} s_{51} ....

S_2 s_{02} s_{12} s_{22} s_{32} s_{42} s_{52} ....

....

Consideriamo la sequenza S_k

ove, per ogni valore di i, $s_{i\,k} \neq s_{i\,i}$

Le sequenze infinite non sono numerabili

S_0 s_{00} s_{10} s_{20} s_{30} s_{40} s_{50} ....

S_1 s_{01} s_{11} s_{21} s_{31} s_{41} s_{51} ....

S_2 s_{02} s_{12} s_{22} s_{32} s_{42} s_{52} ....

....

S_i s_{0i} s_{1i} . . . . . s_{ii} ....

Consideriamo la sequenza S_k

ove, per ogni valore di i, $s_{i\,k} \neq s_{i\,i}$

cioè S_k *è diversa da ogni* S_i *per il valore sulla diagonale*

S_k *non ha posto nella tabella*

Gli algoritmi sono sequenze finite (anche se di lunghezza a priori non limitata)

Le funzioni sono sequenze infinite

x	0	1	2	3	4	5	.....
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	$f_0(4)$	$f_0(5)$	.....
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	$f_1(4)$	$f_1(5)$	.....
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	$f_2(4)$	$f_2(5)$	.....
.....							

Esistono infinitamente più funzione che algoritmi, quindi vi sono funzioni non calcolabili, cioè per cui non esiste alcun algoritmo di calcolo.

In informatica ciò significa che vi sono infiniti problemi per cui non esiste algoritmo di soluzione. Il primo fu indicato da Turing nel 1936 costruendo una antinomia, cioè un'affermazione autocontraddittoria.