Corso di Perfezionamento

Analisi Asintotica

Maria Rita Di Berardini[1]

10 febbraio 2009

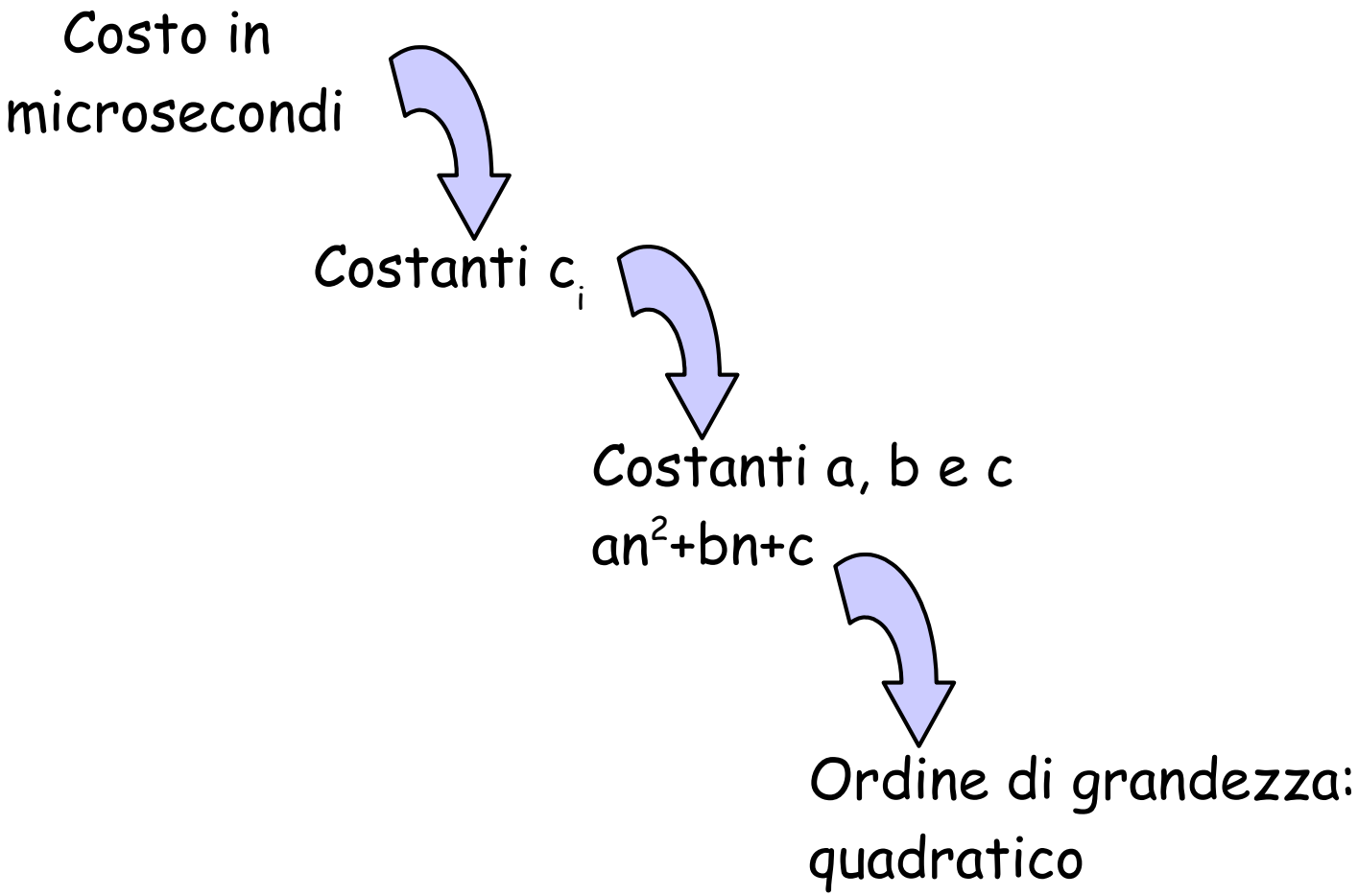
Un graduale processo di astrazione

Passo 1: abbiamo ignorato il costo effettivo delle singole istruzioni introducendo delle costanti per rappresentare i loro (ad esempio le costanti $c_1, ..., c_8$ nel caso dell'insertion sort)
Passo 2: ci siamo resi conto che anche queste costanti forniscono dettagli non necessari

- il tempo di esecuzione dell'insertion sort nel caso peggiore è $an^2 + bn + c$, dove a, b e c sono altre costanti che dipendono dai costi c_i

Passo 3: un ulteriore passo di astrazione consiste nel considerare l'ordine di grandezza o tasso di crescita del tempo di esecuzione

Un graduale processo di astrazione



Analisi Asintotica

- Supponiamo di aver determinato che il tempo necessario a completare un algoritmo sia $T(n) = 4n^2 - 2n + 2$
- Per grandi valori di n, il termine $4n^2$ è preponderante rispetto agli altri, che potranno non essere considerati
- Ex per $n = 500$, $4n^2 = 1.000.000$ è pari a 1000 volte il valore di $2n = 1000$
- Anche i coefficienti diventano irrilevanti se compariamo $T(n)$ con una funzione di ordine superiore, come n^3 oppure 2^n
- Ex: snche se $T(n) = 1.000.000n^2$, $U(n) = n^3$ sarà maggiore di $T(n)$ per ogni $n > 1.000.000$
  $T(1.000.000) = 1.000.000^3 = U(1.000.000)$

Analisi Asintotica

- Astraiamo da alcuni dettagli irrilevanti (ex: termini di ordine diverso, costanti moltiplicative) e ci interessiamo solo al tasso di crescita; analizziamo come il tempo di esecuzione cresce asintoticamente in funzione della dimensione dell'input
- Obiettivo
  - semplificare l'analisi del tempo di esecuzione di un algoritmo (proprio perchè presciamo da dettagli non rilevanti)
  - classificare le funzioni in base al loro comportamento asintotico
- Asintoticamente non significa per tutti gli input
  un algoritmo asintoticamente più efficiente di un altro sarà migliore pet tutti gli input tranne, eventualmente, per quelli molto piccoli

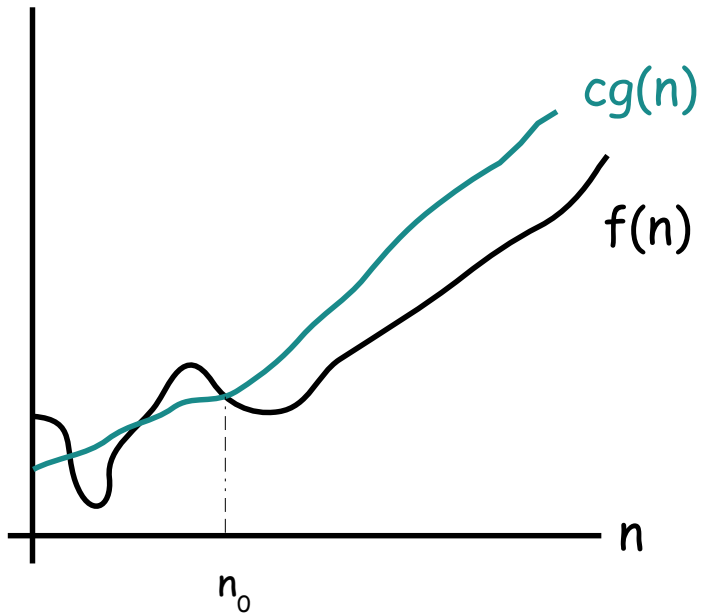
Analisi Asintotica

Per determinare la il comportamento asintotico degli algoritmi importiamo alcune notazioni dalla matematica:

- O (o grande): ci consente di fornire delle **delimitazioni superiori** alla complessità di un algoritmo
- Ω (omega grande): fornisce delle **delimitazione inferiori**
- Θ (theta grande): fornisce delle **delimitazioni strette**, sia superiori che inferiori

Definition (la notazione O grande)

Siano $f, g : \mathbf{N} \to \mathbf{R}^{\geq 0}$. Diciamo che $f(n) = O(g(n))$ se $\exists$ due costanti $c, n_0 > 0$ tali che $0 \leq f(n) \leq cg(n)$ per ogni $n \geq n_0$



- Se $f(n) = O(g(n))$ allora $g(n)$ è un limite asintotico superiore per $f(n)$: $f(n)$, a meno di un fattore costante, cresce **al più** come $g(n)$
- Ex: il limite superiore per la ricerca in un array non ordinato è $O(n)$
- O viene usata nell'analisi del costo computazionale nel caso pessimo
- $O(g(n))$ rappresenta in realtà l'insieme di funzioni definito come:

$$O(g(n)) = \{f(n) | \quad \exists \text{ delle costanti } \textbf{positive } c \text{ ed } n_0 \text{ tali che } 0 \leq f(n) \leq cg(n) \text{ per ogni } n \geq n_0 \}$$

- Scrivere $f(n) = O(g(n))$ è un "abuso" di notazione; avremmo dovuto scrivere $f(n) \in O(g(n))$

Provare che $f(n) = 3n^2 + 10n = O(n^2)$

Dobbiamo dimostrare l'esistenza di costanti positive c ed n_0 tali che $0 \leq f(n) \leq cn^2$ per ogni $n \geq n_0$

Basta scegliere $c = 13$ e $n_0 = 1$; infatti:

$$\begin{array}{rll} & 3n^2 + 10n & \text{per ogni } n \geq 1 \\ \leq & 3n^2 + 10n^2 & \\ = & 13n^2 & \end{array}$$

Vero o falso $n^2 = O(n)$?

Chiaramente è falsa: n non può essere un limite superiore per n^2. Proviamo a dimostrarlo **per assurdo**.

Se $n^2 = O(n)$, allora esistono delle costanti positive c ed n_0 tali che

$$0 \leq n^2 \leq cn \text{ per ogni } n \geq n_0$$

ossia tali che (dividendo tutto per n)

$$0 \leq n \leq c \text{ per ogni } n \geq n_0$$

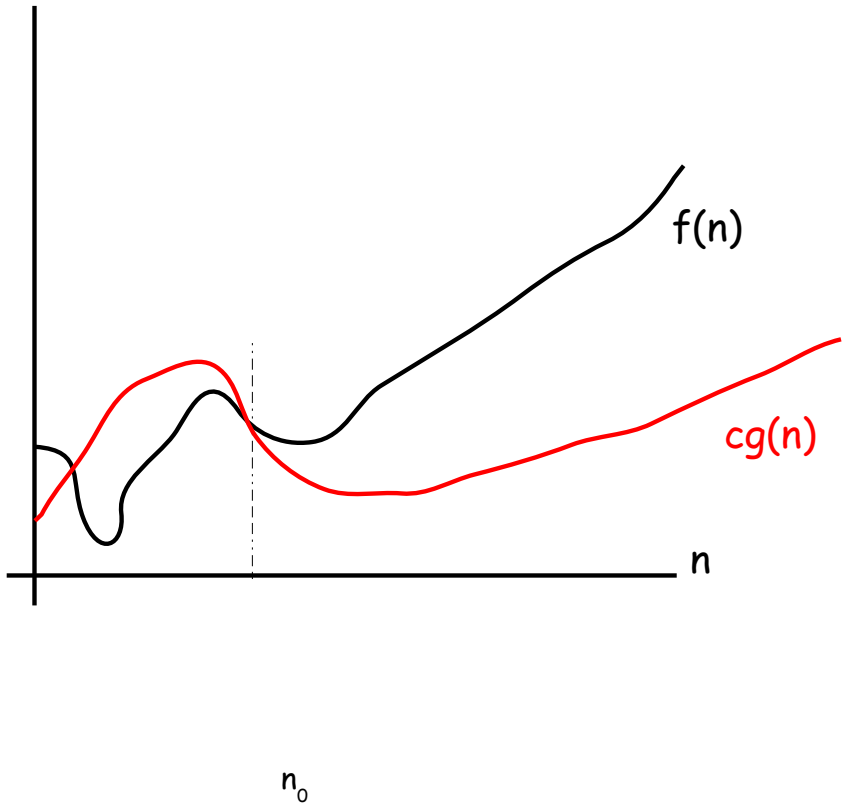
il che è impossibile visto che c è una costante

Classi di complessità asintotica degli algoritmi

- $T(n) = O(1)$: complessità costante (cioè $T(n)$ non dipende dalla dimensione n dei dati di ingresso)
- $T(n) = O(\log n)$: complessità logaritmica.
- $T(n) = O(n)$: complessità lineare.
- $T(n) = O(n \log n)$: complessità pseudolineare (così detta da $n \log n = O(n^{1+\epsilon})$ per ogni $\epsilon > 0$
- $T(n) = O(n^2)$: complessità quadratica
- $T(n) = O(n^3)$: complessità cubica
- $T(n) = O(n^k)$, $k > 0$: complessità polinomiale
- $T(n) = O(\alpha^n)$, $\alpha > 1$: complessità esponenziale

Definition (la notazione Ω grande)

Siano $f, g : \mathbf{N} \rightarrow \mathbf{R}^{\geq 0}$. Diciamo che $f(n) = \Omega(g(n))$ se $\exists$ due costanti $c, n_0 > 0$ tali che $0 \leq cg(n) \leq f(n)$ per ogni $n \geq n_0$



- Se $f(n) = \Omega(g(n))$ allora $g(n)$ è un limite asintotico inferiore per $f(n)$: $f(n)$, a meno di un fattore costante, cresce **almeno** come $g(n)$
- Esempio: il limite inferiore per la ricerca in un array non ordinato è $\Omega(1)$
- Ω viene usata nell'analisi del costo computazionale nel caso ottimo
- $\Omega(g(n))$ rappresenta l'insieme di funzioni definito come:

$$\Omega(g(n)) = \{f(n) | \quad \exists \text{ delle costanti } \textbf{positive } c \text{ ed } n_0 \text{ tali che } 0 \leq cg(n) \leq f(n) \text{ per ogni } n \geq n_0\}$$

- Di nuovo, scrivere $f(n) = \Omega(g(n))$ è un "abuso" di notazione; avremmo dovuto scrivere $f(n) \in \Omega(g(n))$

Provare che $f(n) = 3n^2 + 10n = \Omega(n^2)$

Dobbiamo dimostrare l'esistenza di costanti positive c ed n_0 tali che $0 \leq cn^2 \leq f(n)$ per ogni $n \geq n_0$

Basta scegliere $c = 3$ e $n_0 = 1$; infatti:

$$3n^2 + 10n \geq 3n^2 \text{ per ogni } n \geq n_0 = 1$$

Vero o falso $n^2 = \Omega(n^3)$?

Chiaramente è falsa: n^3 non può essere un limite inferiore per n^2. Proviamo a dimostrarlo **per assurdo**.

Se $n^2 = \Omega(n^3)$, allora esistono delle costanti positive c ed n_0 tali che

$$0 \leq cn^3 \leq n^2 \text{ per ogni } n \geq n_0$$

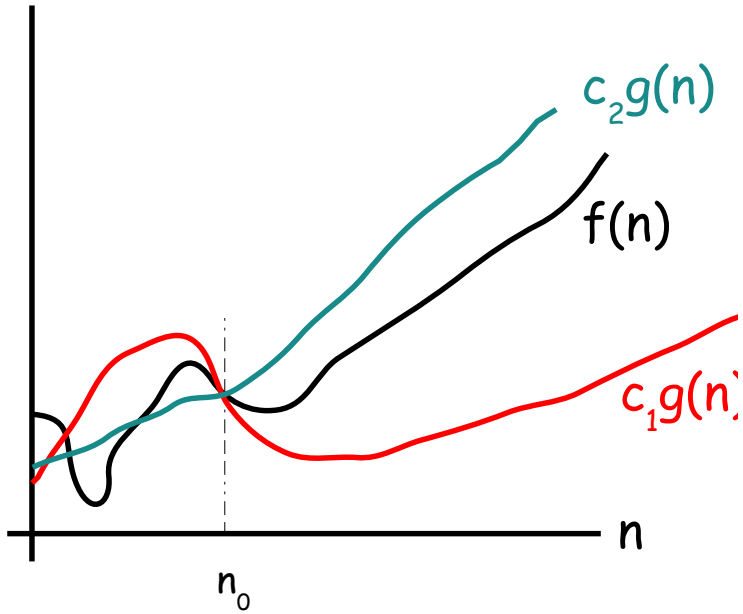
ossia tali che (dividendo tutto per n^3)

$$0 \leq c \leq \frac{1}{n} \text{ per ogni } n \geq n_0$$

il che è impossibile visto che c è una costante

Definition (la notazione Θ grande)

Siano $f, g : \mathbf{N} \rightarrow \mathbf{R}^{\geq 0}$. Diciamo che $f(n) = O(g(n))$ se $\exists$ due costanti $c_1, c_2, n_0 > 0$ tali che $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ per ogni $n \geq n_0$



- Se $f(n) = \Theta(g(n))$ allora $g(n)$ è un limite asintotico stretto per $f(n)$: $f(n)$, a meno di un fattore costante, cresce **esattamente** come $g(n)$
- $\Theta(g(n))$ rappresenta l'insieme di funzioni definito come:

$$\Theta(g(n)) = \{f(n) \mid \quad \exists \text{ costanti } \textbf{positive } c_1,\ c_2 \text{ ed } n_0 \text{ tali che } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ per ogni } n \geq n_0\}$$

- Scrivere $f(n) = \Theta(g(n))$ è un "abuso" di notazione

Provare che $f(n) = 3n^2 + 10n = \Theta(n^2)$

Theorem

Per ogni coppia di funzioni $f(n)$ e $g(n)$, si ha che $f(n) = \Theta(g(n))$ **iff** $f(n) = O(g(n))$ *e* $f(n) = \Omega(g(n))$

Provare che $f(n) = 3n^2 + 10n = \Theta(n^2)$

Theorem

Per ogni coppia di funzioni $f(n)$ e $g(n)$, si ha che $f(n) = \Theta(g(n))$ **iff** $f(n) = O(g(n))$ *e* $f(n) = \Omega(g(n))$

Abbiamo dimostrato che $f(n) = O(n^2)$ e $f(n) = \Omega(n^2)$. Questo basta per concludere che $f(n) = \Theta(n^2)$

Alcune utili proprietà

- Transitività

$$f(n) = O(g(n)) \text{ e } g(n) = O(h(n)) \Longrightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \text{ e } g(n) = \Omega(h(n)) \Longrightarrow f(n) = \Omega(h(n))$$
$$f(n) = \Theta(g(n)) \text{ e } g(n) = \Theta(h(n)) \Longrightarrow f(n) = \Theta(h(n))$$

- Riflessività

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

- Simmetria

$$f(n) = \Theta(g(n)) \textbf{ iff } g(n) = \Theta(f(n))$$

- Simmetria trasposta

$$f(n) = O(g(n)) \textbf{ iff } g(n) = \Omega(f(n))$$

Le notazioni "o piccolo" ed "ω piccolo"

- Siano $f, g : \mathbf{N} \to \mathbf{R}^{\geq 0}$. Diciamo che $f(n) = o(g(n))$ se **per ogni** costante $c > 0$ esiste $n_0 > 0$ tale che $0 \leq f(n) < cg(n)$ per ogni $n \geq n_0$
- Le definizioni di O ed o sono molto simili:
  - nella prima, la condizione $0 \leq f(n) \leq cg(n)$ vale per un qualche $c > 0$
  - nella seconda, $0 \leq f(n) < cg(n)$ vale per **ogni** $c > 0$
- $f(n) = o(g(n))$ implica $f(n) = O(g(n))$
- Il concetto intuitivo in questa notazione è che $f(n)$ diventa trascurabile rispetto a $g(n)$ quando n tende all'infinito, cioè $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Le notazioni "o piccolo" ed "ω piccolo"

- Siano $f, g : \mathbf{N} \to \mathbf{R}^{\geq 0}$. Diciamo che $f(n) = \omega(g(n))$ se per ogni costante $c > 0$ esiste $n_0 > 0$ tale che $0 \leq cg(n) < f(n)$ per ogni $n \geq n_0$
- Le definizioni di Ω ed ω sono molto simili:
  - nella prima, la condizione $0 \leq cg(n) \leq f(n)$ vale per un qualche $c > 0$
  - nella seconda, $0 \leq cg(n) < f(n)$ vale per **ogni** $c > 0$
- $f(n) = \omega(g(n))$ implica $f(n) = \Omega(g(n))$
- In questo caso, $f(n)$ diventa aribitrariamente grande rispetto a $g(n)$ quando n tende all'infinito, cioè $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Alcune utili proprietà

- Transitività

$$f(n) = o(g(n)) \text{ e } g(n) = o(h(n)) \Longrightarrow f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \text{ e } g(n) = \omega(h(n)) \Longrightarrow f(n) = \omega(h(n))$$

- Simmetria trasposta

$f(n) = o(g(n))$ **iff** $g(n) = \omega(f(n))$

$f(n) = O(g(n))$ implica $af(n) = O(g(n))$ per ogni $a > 0$

Se $f(n) = O(g(n))$, allora $\exists c, n_0 > 0$ tali che

$$0 \leq f(n) \leq cg(n) \text{ per ogni } n \geq n_0$$

Moltiplicando tutto per a otteniamo:

$$0 \leq af(n) \leq a \cdot (cg(n)) = (ac) \cdot g(n) = c'g(n) \text{ per ogni } n \geq n_0$$

ossia $af(n) = O(g(n))$

$\log n = O(n)$

Dimostriamo, per induzione su n, che

$$\log n \leq n \qquad \text{per ogni } n \geq 1$$

(questo corrispondere a scegliere $c = 1$ ed $n_0 = 1$ nella def di O).

- Caso base, $n = 1$: $\log 1 = 0 \leq 1$
- Passo induttivo: sia $n \geq 1$ ed assumiamo $\log n \leq n$. Allora:

$$\begin{array}{lll} & \log(n+1) & \text{per ogni } n \geq 1 \\ \leq & \log(n+n) & \\ = & \log(2n) & \text{poichè } \log(ab) = \log(a) + log(b) \\ = & \log 2 + \log n & \\ = & 1 + \log n & \text{per ipotesi induttiva} \\ \leq & 1 + n & \end{array}$$

$\log_b n = O(n)$ per ogni $b > 2$

Per la regola del cambiamento di base (i.e. $\log_b n = \log_b a \cdot \log_a n$), abbiamo che

$$\log_b n = \log_b 2 \cdot \log n \text{ con } \log_b 2 > \log_b 1 = 0$$

Poichè $\log n = O(n)$, abbiamo che (vedi dietro) $\log_b n = O(n)$

$\log_b n^k = O(n)$ per ogni $b > 1$ e $k > 0$

Usiamo un'altra proprietà degli algoritmi: $\log_a n^k = k \log_a n$.
Allora:

$$\log_b n^k = k \log_b n \text{ con } k > 0$$

Di nuovo, poichè $\log_b n = O(n)$, possiamo concludere che (vedi dietro) $\log_b n^k = O(n)$

Polinomi

Un polinomio in n di grado k è una funzione $p(n)$ della forma

$$p(n) = \sum_{i=0}^{k} a_i n^i = a_0 + a_1 n + a_2 n^2 + ... + a_k n^k$$

dove le costanti reali $a_0, a_1, ..., a_k$ sono i coefficienti del polinomio e $a_k \neq 0$. Un polinomio è **asintoticamente positivo** se $a_k > 0$.

Se $p(n)$ è un polinomio di grado k asintoticamente positivo, allora $p(n) = \Theta(n^k)$

$n^k = O(a^n)$ per ogni $k \geq 0$ e $a > 1$

- Per ogni $k \geq 0$ e $a > 1$,

$$\lim_{n \to \infty} \frac{n^k}{a^n} = 0$$

- Allora $n^k = o(a^n)$
- E quindi anche $n^k = O(a^n)$

Che relazione c'è tra 2^n e 3^n?

Banalmente, poichè $2^n \leq 3^n$ per ogni $n \geq 1$, si ha $2^n = O(3^n)$. Il viceversa **non** è vero, ossia $3^n \neq O(2^n)$.

Infatti, se esistessero $c, n_0 > 0$ tali che

$$0 \leq 3^n \leq c2^n \text{ per ogni } n \leq n_0$$

allora (dividendo tutto per 2^n) avremmo

$$(3/2)^n \leq c \text{ per ogni } n \leq n_0$$

con c costante – impossibile

Se $f_1(n) = O(g_1(n))$ ed $f_2(n) = O(g_2(n))$ allora $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$

- se $f_1(n) = O(g_1(n))$ allora $\exists c_1, n_0^1 > 0$ tali che

$$0 \leq f_1(n) \leq c_1 g_1(n) \text{ per ogni } n \geq n_0^1$$

- se $f_2(n) = O(g_2(n))$ allora $\exists c_2, n_0^2 > 0$ tali che

$$0 \leq f_2(n) \leq c_2 g_2(n) \text{ per ogni } n \geq n_0^2$$

- Sia $c = \max\{c_1, c_2\}$ ed $n_0 = \max\{n_0^1, n_0^2\}$. Per ogni $n \geq n_0$, $0 \leq f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \leq c(g_1(n) + g_2(n))$

Se $f_1(n) = O(g_1(n))$ ed $f_2(n) = O(g_2(n))$ allora $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

- se $f_1(n) = O(g_1(n))$ allora $\exists c_1, n_0^1 > 0$ tali che

$$0 \leq f_1(n) \leq c_1 g_1(n) \text{ per ogni } n \geq n_0^1$$

- se $f_2(n) = O(g_2(n))$ allora $\exists c_2, n_0^2 > 0$ tali che

$$0 \leq f_2(n) \leq c_2 g_2(n) \text{ per ogni } n \geq n_0^2$$

- Sia $c = c_1 \cdot c_2$ ed $n_0 = \max\{n_0^1, n_0^2\}$. Per ogni $n \geq n_0$,
$0 \leq f_1(n) \cdot f_2(n) \leq (c_1 g_1(n)) \cdot (c_2 g_2(n)) = c(g_1(n) \cdot g_2(n))$

Se $f(n) = O(g(n))$ allora $f(n)^k = O(g(n)^k)$ per ogni $k \geq 1$

- se $f(n) = O(g(n))$ allora $\exists c, n_0 > 0$ tali che

$$0 \leq f(n) \leq cg(n) \text{ per ogni } n \geq n_0$$

- Elevando tutto alla k e ponendo $c' = c^k > 0$ otteniamo

$$0 \leq f(n) \leq (cg(n))^k = (c^k)g(n)^k = c'g(n)^k \text{ per ogni } n \geq n_0$$

$f(n) = O(g(n))$ implica $2^f(n) = O(2^g(n))$, vero o falso?

- **Falso** troviamo un controesempio
- Sappiamo che $2n = O(n)$
- Ma, $2^{2n} = (2^2)^n = 4^n \neq O(2^n)$