

# Ordini di calcolo

Luca Tagliavini

February 24-25, 2021

## Contents

0.1	Modello di calcolo . . . . .	2
0.2	Costo computazionale . . . . .	2
0.2.1	Esempio . . . . .	2
0.3	Notazione asintotica $O$ (Omicron) . . . . .	2
0.4	Operazione asintotica $\Omega$ (Omega) . . . . .	3
0.5	Operazione asintotica $\Theta$ (Theta) . . . . .	3
0.6	Teoremi . . . . .	4
0.7	Tabella degli ordini di grandezza . . . . .	4
0.8	Spiegazione di alcuni ordini di grandezza . . . . .	5
0.9	Confronto con limite . . . . .	5

## 0.1 Modello di calcolo

Immaginiamo di avere una macchina *a registri* con le seguenti caratteristiche:

- La macchina ha  $n$  locazioni di memoria, indicizzate da 1 a  $n$
- L'accesso in lettura/scrittura richiede sempre *tempo costante*
- La macchina ha accesso a operazioni base come somma/moltiplicazione che vengono eseguite in *tempo costante*

## 0.2 Costo computazionale

Indichiamo con  $f(n)$  la quantità di risorse (tempo, memoria) necessaria al fine dell'esecuzione di un algoritmo con un input  $n$ , operante sulla macchina a registri sopra descritta.

Siamo interessati a studiare *l'ordine di grandezza* di  $f(n)$ , ignorando le costanti numeriche o i termini di ordine inferiore.

Oltretutto non andremo a quantificare un tempo in secondi, ma bensì il numero di operazioni elementari svolte dall'algoritmo.

### 0.2.1 Esempio

Consideriamo due algoritmi:  $A$  e  $B$ . Assumiamo le seguenti tempistiche:

- $f_A(n) = 10^3 n$
- $f_B(n) = 10^{-3} n^2$

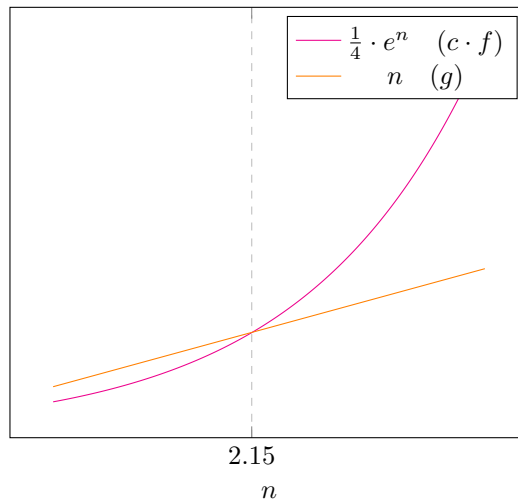
## 0.3 Notazione asintotica $O$ (Omicron)

Data una funzione  $f(n)$  che quantifica il costo di un algoritmo con input  $n$ , usiamo  $O(f(n))$  per indicare l'insieme di funzioni  $g(n)$  che al limite stanno sempre sotto  $f(n)$ , ovvero.

$$g(n) \in O(f(n)) \text{ quando } \exists c > 0, n_0 \geq 0. \forall n \geq n_0. \quad g(n) \leq c \cdot f(n)$$

Esiste un  $n_0$  dopo il quale la funzione  $g$  rimane inferiore rispetto a  $f$ .  $c$  è una costante che eventualmente devo applicare a  $f$  per renderla *sempre* maggiore di  $g$ . Abuso di notazione:  $g = O(f(n))$  come  $g \in O(f(n))$ .

Esempio dove si ha  $g(n) \in O(f(n))$ , scegliendo  $n_0 = 2.15, c = \frac{1}{4}$

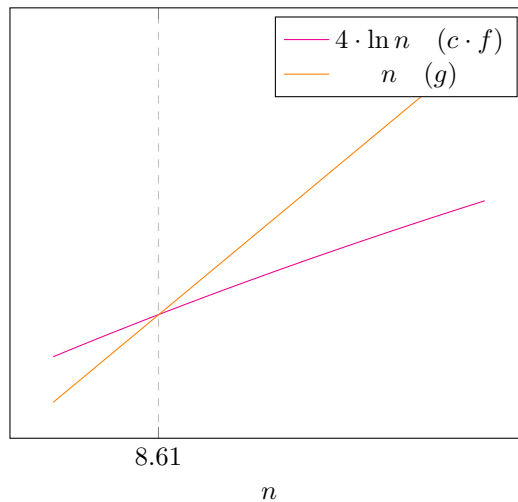


#### 0.4 Operazione asintotica $\Omega$ (Omega)

Data una funzione  $f(n)$  che quantifica il costo di un algoritmo con input  $n$ , usiamo  $\Omega(f(n))$  per indicare l'insieme di funzioni  $g(n)$  che al limite si stanno sempre sopra  $f(n)$ , ovvero.

$$g(n) \in \Omega(f(n)) \text{ quando } \exists c > 0, n_0 \geq 0. \forall n \geq n_0. \quad g(n) \geq c \cdot f(n)$$

Esempio dove si ha  $g(n) \in \Omega(f(n))$ , scegliendo  $n_0 = 8.61, c = \frac{1}{4}$



#### 0.5 Operazione asintotica $\Theta$ (Theta)

Data una funzione  $f(n)$  che quantifica il costo di un algoritmo con input  $n$ , usiamo  $\Theta(f(n))$  per indicare l'insieme di funzioni  $g(n)$  che al limite si comportano

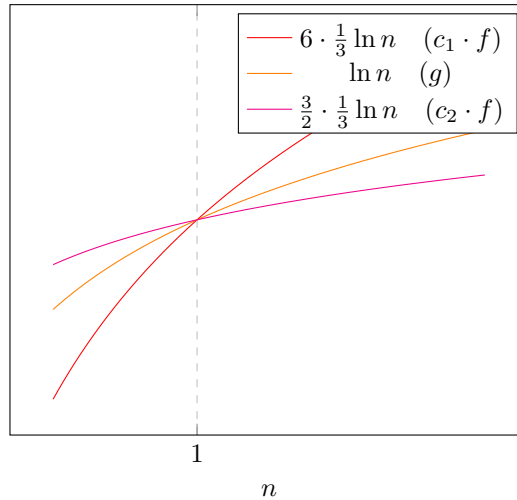
come  $f(n)$ , ovvero.

$$g(n) \in \Theta(f(n)) \text{ quando } \exists c_1 > 0, c_2 > 0, n_0 \geq 0. \forall n \geq n_0.$$

$$c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$$

Le funzioni  $g$  e  $f$  crescono esattamente con lo stesso ordine di grandezza.  
Ossia due funzioni espresse in polinomi saranno in relazione  $\Theta$  sse il polinomio di grado maggiore e' lo stesso.

Esempio dove si ha  $g(n) \in \Theta(f(n))$ , scegliendo  $n_0 = 1, c_1 = 6, c_2 = \frac{3}{2}$



## 0.6 Teoremi

- **simmetria:**  $g(n) = \Theta(f(n))$  sse  $f(n) = \Theta(g(n))$
- **simmetria trasposta:**  $g(n) = O(f(n))$  sse  $f(n) = \Omega(g(n))$
- **transitivita':**  $g(n) = O(f(n)) \wedge f(n) = O(h(n))$  allora  $g(n) = O(h(n))$ .  
Lo stesso vale per  $\Theta$  e  $\Omega$ .

## 0.7 Tabella degli ordini di grandezza

$O(f(n))$	Ordine	Esempio
$O(1)$	costante	numero pari, somma, moltiplicazione
$O(\log n)$	logaritmico	ricerca binaria (array ordinato)
$O(n)$	lineare	ricerca in un array disordinato
$O(n \log n)$	pseudo-lineare	ordinamento di un array (merge sort)
$O(n^2)$	quadratico	ordinamento di un array (bubble sort)
$O(n^3)$	cubico	prodotto di due matrici $n \times n$
$O(c^n)$	esponenziale, $c > 1$	<i>subset-sum problem</i> con forza bruta
$O(n!)$	fattoriale	<i>commesso viaggiatore</i> con forza bruta
$O(n^n)$	esponenziale, base $n$	<i>n-queens</i> tramite forza bruta

- **subset-sum**: abbiamo un insieme di numeri, dobbiamo trovare un sottoinsieme al quale, applicando una sommatoria si ottiene un valore desiderato. L'approccio forza bruta considera *ogni sottoinsieme possibile*.
- **commesso viaggiatore**: abbiamo una mappa e delle strade (rappresentate con grafi) e vogliamo trovare il modo migliore per il commesso di viaggiare da A a B. L'approccio forza bruta consiste nel valutare *ogni strada possibile*.
- **n-queens**: problema che ci chiede di porre le regine in una scacchiera  $n \times n$  in modo che esse non si mangino a vicenda. L'approccio forza bruta prova *ogni possibile combinazione* di piazzamento in  $n \times n$ .

## 0.8 Spiegazione di alcuni ordini di grandezza

- **binary search**: Andiamo ad analizzare gli elementi dell'array considerando meta' array alla volta, partendo da  $n$  elementi, guardando poi  $\frac{n}{2}$ , poi  $\frac{n}{4}$  e cosi' via. Facendo questa procedura si svolgono  $\log n$  (dove  $\log$  e' sempre  $\log_2$  in algoritmi). Eccone una spiegazione:

$$\frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8}$$

fino ad arrivare ad avere una frazione che vale 1

$$1 = \frac{n}{2^{\#}}$$

$$2^{\#} = n$$

$$\# = \log_2 n$$

- **merge sort**: Analizziamo gli array a meta' come nelle binary search, e ogni operazione di ordinamento sulle sottoparti richiede  $O(\frac{n}{2})$  tempo per svolgere i confronti; Visto che dovremo ordinare entrambe le meta' dell'array, svolgeremo  $O(\frac{n}{2}) \cdot 2$  volte l'operazione di ordinamento, ovvero  $O(n)$ .

Il che ci da un costo di  $O(n) \cdot O(\log n) = O(n \log n)$

## 0.9 Confronto con limite

Per confrontare l'ordine di grandezza asintotica di due funzioni  $g(n)$  e  $f(n)$ , si puo' svolgere il:

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

- $\infty$ :  $g(n) = \Omega(f(n))$  poiche'  $g(n)$  ha una crescita superiore
- $k \in \mathbb{R}$ :  $g(n) = \Theta(f(n))$  poiche'  $g(n)$  cresce come  $f(n) + k$
- $0$ :  $g(n) = O(f(n))$  poiche'  $g(n)$  ha una crescita inferiore