

Algoritmi e Strutture Dati

Capitolo 2

Modelli di calcolo e metodologie di analisi

Camil Demetrescu, Irene Finocchi,
Giuseppe F. Italiano

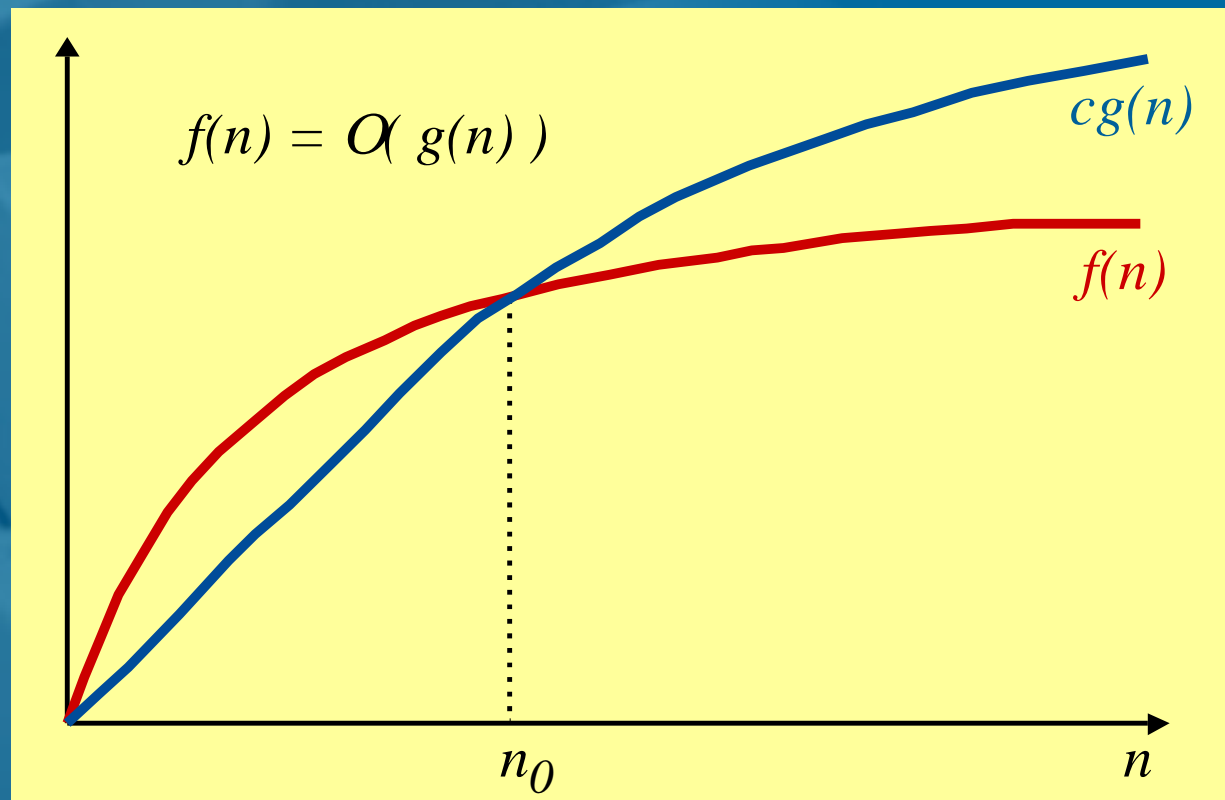
Notazione asintotica

Notazione asintotica

- $f(n)$ = tempo di esecuzione / occupazione di memoria di un algoritmo su input di dimensione n
- La notazione asintotica è un'astrazione utile per descrivere l'ordine di grandezza di $f(n)$ ignorando i dettagli non influenti, come costanti moltiplicative e termini di ordine inferiore

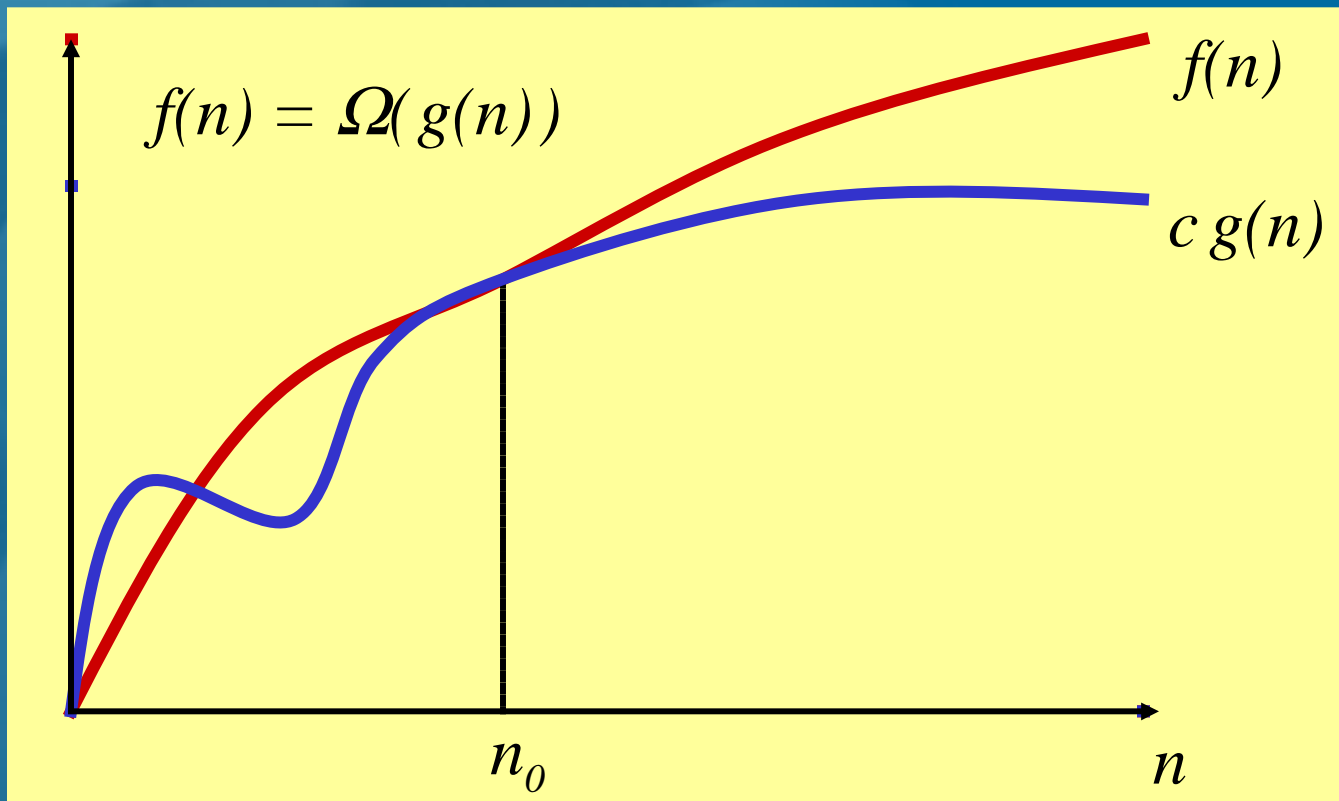
Notazione asintotica O

$f(n) = O(g(n))$ se \exists due costanti $c > 0$ e $n_0 \geq 0$ tali che $f(n) \leq c g(n)$ per ogni $n \geq n_0$



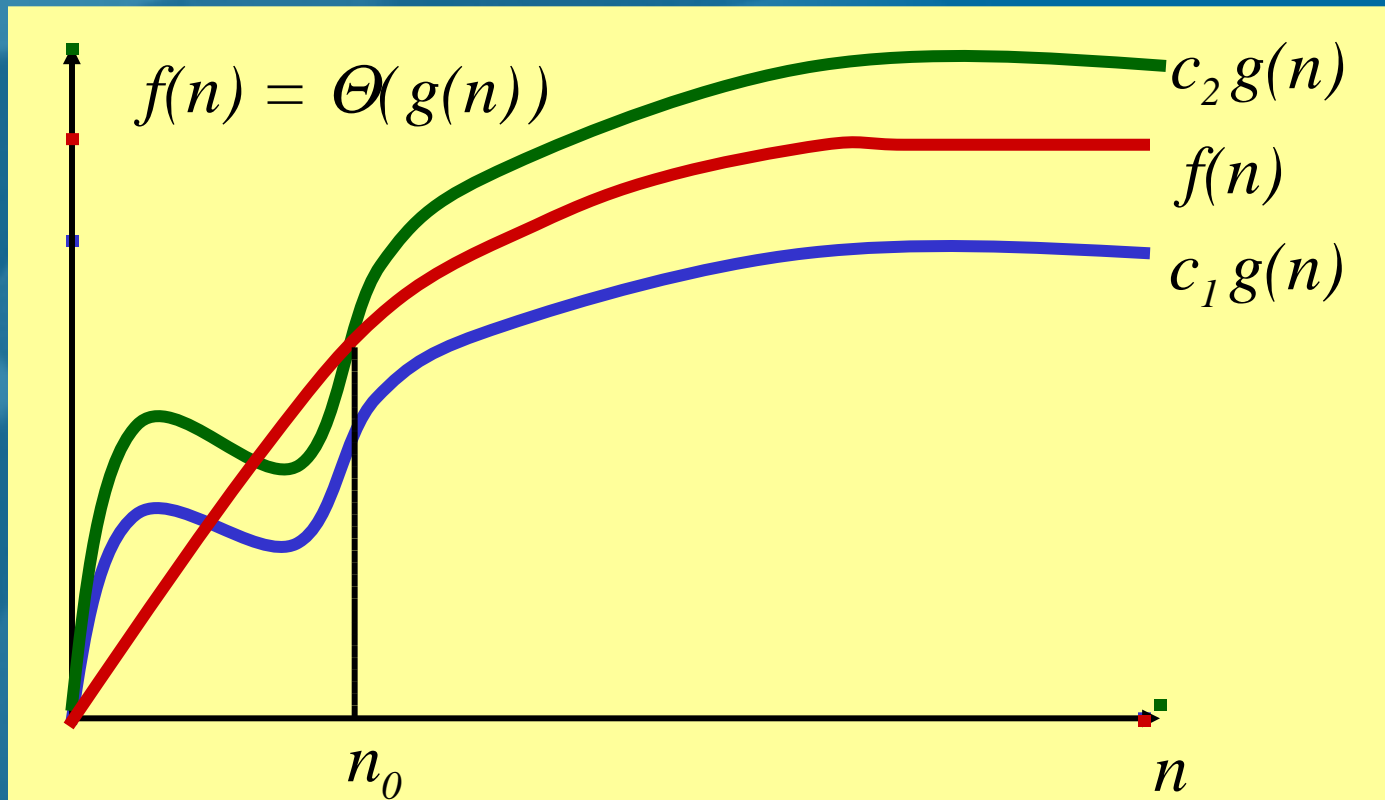
Notazione asintotica Ω

$f(n) = \Omega(g(n))$ se \exists due costanti $c > 0$ e $n_0 \geq 0$ tali che $f(n) \geq c g(n)$ per ogni $n \geq n_0$



Notazione asintotica Θ

$f(n) = \Theta(g(n))$ se \exists tre costanti $c_1, c_2 > 0$ e $n_0 \geq 0$ tali che $c_1 g(n) \leq f(n) \leq c_2 g(n)$ per ogni $n \geq n_0$



Notazione asintotica: esempi

- Sia $g(n)=3n^2+10$
- $g(n)=O(n^2)$: scegliere $c=4$ e $n_0=10$
- $g(n)=\Omega(n^2)$: scegliere $c=1$ e $n_0=0$
- $g(n)=\Theta(n^2)$: infatti $g(n)=\Theta(f(n))$ se e solo se $g(n)=O(f(n))$ e $g(n)=\Omega(f(n))$
- $g(n)=O(n^3)$ ma $g(n)\neq\Theta(n^3)$

Funzioni polinomiali

Se $g(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$, allora $g(n) = O(n^k)$

Dim:

$$g(n) \leq |g(n)| = |a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0| \leq$$

$$\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0| =$$

$$= n^k (|a_k| + |a_{k-1}|/n + \dots + |a_1|/n^{k-1} + |a_0|/n^k) \leq c n^k$$

Delimitazioni inferiori e superiori

La notazione asintotica introdotta ci permette di esprimere delimitazioni inferiori e superiori alla complessità di un problema rispetto a una data risorsa di calcolo.

La notazione asintotica O è adatta ad esprimere delimitazioni superiori, mentre la notazione Ω permette di esprimere le delimitazioni inferiori

Upper bound

- Un algoritmo A ha costo di esecuzione $O(f(n))$ su un'istanza di dimensione n , rispetto a una certa risorsa di calcolo, se la quantità r di risorsa sufficiente per eseguire A su una qualunque istanza di dimensione n verifica la relazione $r(n)=O(f(n))$

Upper bound di un problema

- Un problema P ha una complessità $O(f(n))$ rispetto a una data risorsa di calcolo se esiste un algoritmo che risolve P il cui costo di esecuzione rispetto a quella risorsa è $O(f(n))$

Lower bound

- Un algoritmo A ha costo di esecuzione $\Omega(f(n))$ su un'istanza di ingresso di dimensione n , rispetto a una certa risorsa di calcolo, se la quantità r di risorsa necessaria per eseguire A su una qualunque istanza di dimensione n verifica la relazione $r(n) = \Omega(f(n))$

Lower bound per un problema

Un problema P ha una complessità $\Omega(f(n))$ rispetto a una data risorsa di calcolo se ogni algoritmo che risolve P ha costo di esecuzione $\Omega(f(n))$ rispetto a quella risorsa.

Dato un problema P con complessità $\Omega(f(n))$ rispetto a una data risorsa di calcolo, un algoritmo che risolve P è ottimo se ha costo di esecuzione $O(f(n))$ rispetto a quella risorsa

Metodi di analisi

Caso peggiore, migliore e medio

- Misureremo le risorse di calcolo usate da un algoritmo (tempo di esecuzione / occupazione di memoria) in funzione della dimensione n delle istanze
- Istanze diverse, a parità di dimensione, potrebbero però richiedere risorse diverse
- Distinguiamo quindi ulteriormente tra analisi nel caso peggiore, migliore e medio

Caso peggiore

- Sia $\text{tempo}(I)$ il tempo di esecuzione di un algoritmo sull'istanza I
- $T_{\text{worst}}(n) = \max_{\text{istanze } I \text{ di dimensione } n} \{\text{tempo}(I)\}$
- Intuitivamente, $T_{\text{worst}}(n)$ è il tempo di esecuzione sulle istanze di ingresso che comportano più lavoro per l'algoritmo

Caso migliore

- Sia $\text{tempo}(I)$ il tempo di esecuzione di un algoritmo sull'istanza I
- $T_{\text{best}}(n) = \min_{\text{istanze } I \text{ di dimensione } n} \{\text{tempo}(I)\}$
- Intuitivamente, $T_{\text{best}}(n)$ è il tempo di esecuzione sulle istanze di ingresso che comportano meno lavoro per l'algoritmo

Caso medio

- Sia $\mathcal{P}(I)$ la probabilità di avere in ingresso un'istanza I
- $T_{\text{avg}}(n) = \sum_{\text{istanze } I \text{ di dimensione } n} \{ \mathcal{P}(I) \text{ tempo}(I) \}$
- Intuitivamente, $T_{\text{avg}}(n)$ è il tempo di esecuzione nel caso medio, ovvero sulle istanze di ingresso “tipiche” per il problema
- Richiede conoscenza di una distribuzione di probabilità sulle istanze

Esempio 1

Ricerca di un elemento x in una lista \mathcal{L} non ordinata

algoritmo *ricercaSequenziale*(*lista* L , *elem* x) \rightarrow *booleano*

1. **for each** ($y \in L$) **do**
2. **if** ($y = x$) **then return** trovato
3. **return** non trovato

$$T_{\text{best}}(n) = 1$$

x è in prima posizione

$$T_{\text{worst}}(n) = n$$

$x \notin \mathcal{L}$ oppure è in ultima posizione

$$T_{\text{avg}}(n) = (n+1)/2$$

assumendo che le istanze siano
equidistribuite

Esempio 2 (1/2)

Ricerca di un elemento x in un array \mathcal{L} ordinato

algoritmo ricercaBinariaIter(*array* L , *elem* x) \rightarrow *booleano*

1. $a \leftarrow 1$
2. $b \leftarrow$ lunghezza di L
3. **while** ($L[(a + b)/2] \neq x$) **do**
4. $m \leftarrow (a + b)/2$
5. **if** ($L[m] > x$) **then** $b \leftarrow m - 1$
6. **else** $a \leftarrow m + 1$
7. **if** ($a > b$) **then return** non trovato
8. **return** trovato

Confronta x con l'elemento centrale di \mathcal{L} e prosegue nella metà sinistra o destra in base all'esito del confronto

Esempio 2 (2/2)

$$T_{\text{best}}(n) = 1$$

l'elemento centrale è uguale a x

$$T_{\text{worst}}(n) = \log n$$

$x \notin \mathcal{L}$ oppure viene trovato
all'ultimo confronto

Poiché la dimensione del sotto-array su cui
si procede si dimezza dopo ogni confronto,
dopo l' i -esimo confronto il sottoarray di
interesse ha dimensione $n/2^i$

Risulta $n/2^i = 1$ per $i = \log_2 n$

$$T_{\text{avg}}(n) = \log n - 1 + 1/n \quad \text{assumendo che le istanze siano equidistribuite}$$

Analisi di algoritmi ricorsivi

Esempio

L'algoritmo di ricerca binaria può essere riscritto ricorsivamente come:

algoritmo ricercaBinariaRic(*array* L , *elemento* x) \rightarrow *booleano*

1. $n \leftarrow$ lunghezza di L
2. **if** ($n = 0$) **then return** non trovato
3. $i \leftarrow \lceil n/2 \rceil$
4. **else if** ($L[i] = x$) **then return** trovato
5. **else if** ($L[i] > x$) **then return** ricercaBinariaRic($x, L[1; i - 1]$)
6. **else return** ricercaBinariaRic($x, L[i + 1; n]$)

Come analizzarlo?

Equazioni di ricorrenza

Il tempo di esecuzione può essere descritto tramite una equazione di ricorrenza:

$$T(n) = \begin{cases} c + T(\lceil n-1 \rceil/2) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

Vari metodi per risolvere equazioni di ricorrenza:
iterazione, sostituzione, teorema Master...

Metodo dell'iterazione

Idea: “srotolare” la ricorsione, ottenendo una sommatoria dipendente solo dalla dimensione n del problema iniziale

Esempio: $T(n) = c + T(n/2)$

$$T(n/2) = c + T(n/4) \quad \dots$$

$$\begin{aligned} \Rightarrow T(n) &= c + T(n/2) = 2c + T(n/4) = \\ &= \left(\sum_{j=1 \dots i} c \right) + T(n/2^i) = ic + T(n/2^i) \end{aligned}$$

Per $i = \log_2 n$: $T(n) = c \log n + T(1) = O(\log n)$

Metodo della sostituzione

Idea: “indovinare” una soluzione, ed usare induzione matematica per provare che la soluzione dell’equazione di ricorrenza è effettivamente quella intuita

Esempio: $T(n) = n + T(n/2)$, $T(1)=1$

Assumiamo che la soluzione sia $T(n) \leq cn$ per una costante c opportuna

Passo base: $T(1)=1 \leq c \cdot 1$ per ogni c

Passo induttivo: $T(n) = n + T(n/2) \leq n + c(n/2) = (c/2 + 1)n$

Quindi $T(n) \leq cn$ per $c \geq 2$

Teorema Master

Permette di analizzare algoritmi basati sulla tecnica del *divide et impera*:

- dividi il problema (di dimensione n) in **a sottoproblemi** di **dimensione n/b**
- risolvi i sottoproblemi ricorsivamente
- ricombina le soluzioni

Sia **$f(n)$** il tempo per dividere e ricombinare istanze di dimensione n . La relazione di ricorrenza è data da:

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

Teorema Master

La relazione di ricorrenza:

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

ha soluzione:

1. $T(n) = \Theta(n^{\log_b a})$ se $f(n) = O(n^{\log_b a - \epsilon})$ per $\epsilon > 0$
2. $T(n) = \Theta(n^{\log_b a} \log n)$ se $f(n) = \Theta(n^{\log_b a})$
3. $T(n) = \Theta(f(n))$ se $f(n) = \Omega(n^{\log_b a + \epsilon})$ per $\epsilon > 0$ e
 $a f(n/b) \leq c f(n)$ per $c < 1$ e n sufficientemente grande

Esempi

1) $T(n) = n + 2T(n/2)$

$a=2, b=2, f(n)=n=\Theta(n) \quad T(n)=\Theta(n \log n)$

(caso 2 del teorema master)

$\log_2 2$



2) $T(n) = c + 3T(n/9)$

$a=3, b=9, f(n)=c=O(n^{\log_9 3 - \epsilon}) \rightarrow T(n)=\Theta(\sqrt{n})$

(caso 1 del teorema master)

3) $T(n) = n + 3T(n/9)$

$a=3, b=9, f(n)=n=\Omega(n^{\log_9 3 + \epsilon})$

$3(n/9) \leq c n \quad \text{per } c=1/3$

(caso 3 del teorema master)



$T(n)=\Theta(n)$

Riepilogo

- Esprimiamo la quantità di una certa risorsa di calcolo (tempo, spazio) usata da un algoritmo in funzione della dimensione n dell'istanza di ingresso
- La notazione asintotica permette di esprimere la quantità di risorsa usata dall'algoritmo in modo sintetico, ignorando dettagli non influenti
- A parità di dimensione n , la quantità di risorsa usata può essere diversa, da cui la necessità di analizzare il caso peggiore o, se possibile, il caso medio
- La quantità di risorsa usata da algoritmi ricorsivi può essere espressa tramite relazioni di ricorrenza, risolubili tramite vari metodi generali