Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

Equazioni di ricorrenza

Giancarlo Bongiovanni
Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Queste dispense sono state realizzate sulla base delle slide
preparate da T. Calamoneri e G. Bongiovanni
per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Equazioni di ricorrenza (1)

- Valutare il costo computazionale di un algoritmo ricorsivo è, in genere, più laborioso che nel caso degli algoritmi iterativi.
- Infatti, la natura ricorsiva della soluzione algoritmica dà luogo a una funzione di costo che, essendo strettamente legata alla struttura dell'algoritmo, è anch'essa ricorsiva.
- Trovare la funzione di costo ricorsiva è piuttosto immediato. Essa però deve essere riformulata in modo che non sia più ricorsiva, altrimenti il costo asintotico non può essere quantificato, e questa è la parte meno semplice.
- La riformulazione della funzione di costo ricorsiva in una equivalente ma non ricorsiva si affronta impostando una ***equazione di ricorrenza***, costituita dalla **formulazione ricorsiva e dal caso base**.

Equazioni di ricorrenza (2)

Esempio: costo computazionale della ricerca sequenziale ricorsiva:

```
def ricercaBin(array A, int v, int i, int n):
  /* PREC: 1 ≤ i ≤ n+1, vettore da analizzare A[i .. n]

    if i > n return NULL
       /* caso base: finita scansione, non trovato */

    if A[i] = v return i
       /* caso base: valore trovato */

    return ricercaSeq(A, i+1, n)
```

In generale: $T(n)=\Theta(1)+T(n-1)$

Caso base: $T(1)=\Theta(1)$

Equazioni di ricorrenza (3)

- La parte generale dell'equazione di ricorrenza che definisce *T(n)* deve essere sempre costituita dalla somma di almeno due addendi, di cui almeno uno contiene la parte ricorsiva (nell'esempio *T(n-1)*) mentre uno rappresenta il costo computazionale di tutto ciò che viene eseguito al di fuori della chiamata ricorsiva.
- Anche se questa parte dovesse essere un solo confronto, il suo costo non può essere ignorato, altrimenti si otterrebbe che la funzione ha una costo computazionale indipendente dalla dimensione del suo input:

$$T(n)=T(n-1)$$

- Che risulterebbe quello del caso base. Non possiamo dire che ciò sia impossibile, ma è molto improbabile.
- Deve sempre essere presente un caso base.

Equazioni di ricorrenza (4)

Esistono alcuni metodi utili per risolvere le equazioni di ricorrenza, che illustreremo:

- Metodo iterativo
- Metodo di sostituzione
- Metodo dell'albero
- Metodo principale

Metodo iterativo (1)

Idea:

- Sviluppare l'equazione di ricorrenza ed esprimerla come somma di termini dipendenti da *n* e dal caso base

Difficoltà:

- Maggiore quantità di calcoli algebrici rispetto agli altri metodi

Aspetto positivo:

- Metodo piuttosto meccanico e applicabile quasi sempre

Metodo iterativo (2)

Esempio: Equazione relativa alla ricerca sequenziale ricorsiva

$T(n) = T(n-1) + \Theta(1)$

$T(1) = \Theta(1)$

$T(n) = T(n-1) + \Theta(1)$, ma $T(n-1)=T(n-2) + \Theta(1)$

Sostituiamo:

$T(n) = T(n-2) + \Theta(1) + \Theta(1)$, ma $T(n-2)=T(n-3) + \Theta(1)$

$T(n) = T(n-3) + \Theta(1) + \Theta(1) + \Theta(1)$, ma $T(n-3)=T(n-4) + \Theta(1)$

$T(n) = T(n-4) + \Theta(1) + \Theta(1) + \Theta(1) + \Theta(1)$, ecc.

fino a ottenere n addendi $\Theta(1)$

da cui: $\boldsymbol{T(n) = n\,\Theta(1) = \Theta(n)}$.

Metodo iterativo (3)

Esempio: Equazione relativa alla ricerca binaria ricorsiva

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

$T(1) = \Theta(1)$

$T(n) = T(n/2) + \Theta(1)$ ma $T(n/2) = T(n/2^2) + \Theta(1)$

Sostituiamo:

$T(n) = T(n/2^2) + \Theta(1) + \Theta(1)$ ma $T(n/2^2) = T(n/2^3) + \Theta(1)$

$T(n) = T(n/2^3) + \Theta(1) + \Theta(1) + \Theta(1)$ ma…

…

$T(n) = T(n/2^k) + k\,\Theta(1)$

Quando $k = \log n$ si ha $n/2^k = 1$, quindi per tale valore di k ci fermiamo ed otteniamo:

$\boldsymbol{T(n) = \Theta(1) + \log n\,\Theta(1) = \Theta(\log n)}$.

Metodo iterativo (4)

Formuliamo un altro algoritmo ricorsivo per la ricerca sequenziale su *n* elementi:

- ispezioniamo l' elemento centrale;
- se non è l'elemento cercato risolviamo il problema ricorsivamente sia sui primi *n/2 ch*e sugli ultimi *n/2* elementi.

```
Funzione Ricerca_seq_ric2(A: vettore; v, i_min, i_max: intero)
   if (i_min = i_max)                                              Θ(1)
      if A[i_min]=v return i_min else return null                  Θ(1)
   else m ← SUP(i_min + i_max)/2)                                  Θ(1)
        x ← Ricerca_seq_ric2(A, v, i_min, m)                       T(n/2)
        if x=null return Ricerca_seq_ric2(A, v, m + 1, i_max)      T(n/2)
```

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

$T(1) = \Theta(1)$

Questioni di stile…

```
def ricercaBin(array A, int v, int inf, int sup):
    if inf > sup return NULL
            /* caso base: sottovettore vuoto */

    m ← ⌊(inf + sup)/2⌋
    if A[m] = v return m
            /* caso base: elemento trovato */

    res ← ricercaBinSeq(A, v, inf, m-1)

    if res ≠ NULL return res
    return ricercaBinSeq(A, v, m+1, sup)
```

Metodo iterativo (5)

Risolviamo l'equazione relativa a questa versione dell'algoritmo:

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

$T(1) = \Theta(1)$

$T(n) = 2T(n/2) + \Theta(1)$ ma $T(n/2) = 2T(n/2^2) + \Theta(1)$

Proseguiamo con le sostituzioni:

$$
\begin{aligned}
T(n) &= 2[2T(n/2^2) + \Theta(1)] + \Theta(1) = \\
&= 2[2[2T(n/2^3) + \Theta(1)] + \Theta(1)] + \Theta(1) = \\
&= 2^3 T(n/2^3) + 4\Theta(1) + 2\Theta(1) + \Theta(1) = \\
&= 2^3 T(n/2^3) + 2^2\,\Theta(1) + 2^1\,\Theta(1) + 2^0\,\Theta(1) \ldots \\
&= 2^k\,T(n/2^k) + \sum_{i=0..k-1} 2^i\,\Theta(1)
\end{aligned}
$$

Metodo iterativo (6)

Come abbiamo già visto, proseguiamo finché

$$2^k = n, \text{ ossia } k = \log n$$

Ottenendo

$$T(n) = 2^{\log n}\,\Theta(1) + \sum_{i=0..\log n-1} 2^i\,\Theta(1)$$

Ricordiamo che

$$2^{\log n} = n = \Theta(n)$$

$$\sum_{i=0..\log n-1} 2^i = 2^{\log n} - 1 = n-1 = \Theta(n)$$

Otteniamo

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

Soluzioni utili da ricordare

C'è una sommatoria che capita spesso di dover risolvere:

$$\sum_{i=0..k} x^i$$

Se $0 < x < 1$ sappiamo che:

$$\sum_{i=0\,..\infty} x^i = \frac{1}{1-x}$$

E quindi la sommatoria $\sum_{i=0..k} x^i$ è $\Theta(1)$ dato che

$$1 \leq \sum_{i=0\,..\,k} x^i \leq \frac{1}{1-x}$$

Se invece $x > 1$ si può usare la soluzione:

$$\sum_{i=0..k} x^i = \frac{x^{k+1}-1}{x-1}$$

Metodo iterativo (7)

Esempio (Fibonacci, il metodo iterativo non è applicabile)

$T(n) = T(n-1) + T(n-2) + \Theta(1)$
$T(1) = \Theta(1)$

Non riusciamo a risolvere questa equazione di ricorrenza tramite il metodo iterativo, in quanto il numero di addendi cresce esponenzialmente.

Possiamo però fare le seguenti considerazioni:

1. $T(n) \leq 2T(n-1) + \Theta(1)$
2. $T(n) \geq 2T(n-2) + \Theta(1)$

Metodo iterativo (8)

Applicando il metodo iterativo alla prima disuguaglianza otteniamo:

$$T(n) = O(2^n)$$

Applicando il metodo iterativo alla seconda disuguaglianza otteniamo:

$$T(n) = \Omega(2^{n/2})$$

Anche se non siamo in grado di trovare una delimitazione stretta Θ, possiamo comunque concludere che calcolare i numeri di Fibonacci con una tecnica ricorsiva ha un costo esponenziale in n.

Si suggerisce di fare le due derivazioni, che sono comunque illustrate nelle dispense, come esercizio.

Metodo di sostituzione (1)

Idea:

- si ipotizza una soluzione per l'equazione di ricorrenza data;
- si verifica se essa "funziona".

Difficoltà:

- si deve trovare la funzione più vicina alla vera soluzione, perché tutte le funzioni più grandi (se stiamo cercando O) o più piccole (se stiamo cercando Ω) funzionano.
- In effetti questo metodo serve soprattutto nelle dimostrazioni mentre si sconsiglia di utilizzarlo nella pratica.

Metodo di sostituzione (2)

Esempio: Equazione relativa alla ricerca seq. ricorsiva

$T(n) = T(n - 1) + \Theta(1)$

$T(1) = \Theta(1)$

- Ipotizziamo la soluzione $T(n)=cn$, per una costante opportuna c. Sostituendo nell'equazione di ricorrenza otteniamo:

  $T(n)= cn = c(n-1) + \Theta(1)$, cioè $c = \Theta(1)$;

  $T(1) = c = \Theta(1)$
- Però non è detto che le due costanti nascoste nella notazione asintotica siano le stesse. Per questa ragione, è necessario eliminare la notazione asintotica dall'equazione di ricorrenza, così che le costanti non rimangano "nascoste" nella notazione, conducendo a risultati errati.

Metodo di sostituzione (3)

- Riformuliamo quindi l'equazione relativa alla ricerca sequenziale ricorsiva:

  $T(n) = T(n - 1) + c$

  $T(1) = d$

- Ipotizziamo la soluzione $T(n) = O(n)$, *ossia* $T(n) \leq kn$ dove k è una costante che va ancora determinata. La dimostrazione si fa per induzione su n.
- N.B. non si può sperare in una soluzione esatta, ma possiamo solo maggiorare o minorare.

1. Sostituiamo nel caso base:

   $T(1) \leq k$; poiché sapevamo che $T(1) = d$, la disuguaglianza è soddisfatta se e solo se $k \geq d$.

2. Sostituiamo nella formulazione ricorsiva:

   $T(n) \leq k(n-1) + c = kn - k + c \leq kn$; vera se e solo se $k \geq c$.

3. La soluzione $T(n) \leq kn$ è corretta per tutti i valori di k tali che $k \geq c$ e $k \geq d$. Poiché un tale k esiste sempre, una volta fissati c e d, $T(n)$ è un $O(n)$.

Metodo di sostituzione (4)

- Equazione relativa alla ricerca sequenziale ricorsiva:

  $T(n) = T(n - 1) + c$

  $T(1) = d$

- Che cosa sarebbe successo se avessimo ipotizzato $T(n) \leq kn^2$?

- Anche questa soluzione è corretta per tutti i valori di k tali che

  $k \geq c$ e $k \geq d$.

- A noi interessa stimare T(n) asintoticamente tramite la funzione più piccola, e questo è spesso un obiettivo difficile.

Metodo di sostituzione (5)

- Equazione relativa alla ricerca sequenziale ricorsiva:

  $T(n) = T(n - 1) + c$

  $T(1) = d$

- Ipotizziamo la soluzione $T(n) = \Omega(n)$, *ossia* $T(n) \geq hn$ dove h è una costante che va ancora determinata. Come prima, la dimostrazione si fa per induzione su n.

1. In modo assolutamente analogo al caso *O,* sostituiamo nel caso base ottenendo che la disuguaglianza è vera per $h \leq d$,
2. Sostituiamo nella formulazione ricorsiva:

   $T(n) \geq h(n - 1) + c = hn - h + c \geq hn$ che *è* vera se e solo se $h \leq c$.
3. Deduciamo dunque che $T(n)$ è un $\Omega(n)$ poiché è possibile trovare un valore h ($h \leq c$, $h \leq d$) per il quale si ha $T(n) \geq hn$.

Metodo di sostituzione (6)

Equazione relativa alla ricerca sequenziale ricorsiva:

$T(n) = T(n - 1) + c$

$T(1) = d$

Dalle due soluzioni

$$T(n) = O(n) \text{ e } T(n) = \Omega(n)$$

Si ottiene ovviamente che

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

Metodo di sostituzione (7)

Consideriamo ora un'altra equazione di ricorrenza:

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

$T(1) = \Theta(1)$

Dobbiamo eliminare per prima cosa la notazione asintotica:

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

$T(1)=d$

per due costanti positive **fissate** c e d.

Metodo di sostituzione (8)

Equazione di ricorrenza:

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

$T(1)=d$

Ipotizziamo la soluzione $T(n) \leq kn$ per qualche k **da determinare** e sostituiamo:

- nel caso base: $d \leq k$
- nel caso generale: $T(n) \leq 2\,k\,n/2 + c = kn + c$
  - che non è MAI $\leq kn$

Vuol dire che l'ipotesi è errata? non sempre.

Metodo di sostituzione (9)

Equazione di ricorrenza:

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

$T(1) = d$

Ipotizziamo una nuova soluzione T(n) ≤ $kn - h$ per qualche h e k **da determinare** e sostituiamo:

- nel caso base: $T(1) \leq k - h$, *che è vera per* $k - h \geq d$
- nel caso generale:

  $T(n) \leq 2\,k\,n/2 - h + c = kn - h + c \leq kn$ *che è vera per qualunque* k *se* $h \geq c$.

Da qui deduciamo che $T(n)=O(n)$.

Si lascia per esercizio dimostrare che T(n)= $\Omega(n)$

NOTA: la tecnica vista può essere necessaria in una delle due dimostrazioni ma mai in entrambe.

Metodo dell'albero (1)

Idea:

- Rappresentare graficamente lo sviluppo del costo computazionale dell'algoritmo, in modo da poterla valutare con una certa facilità

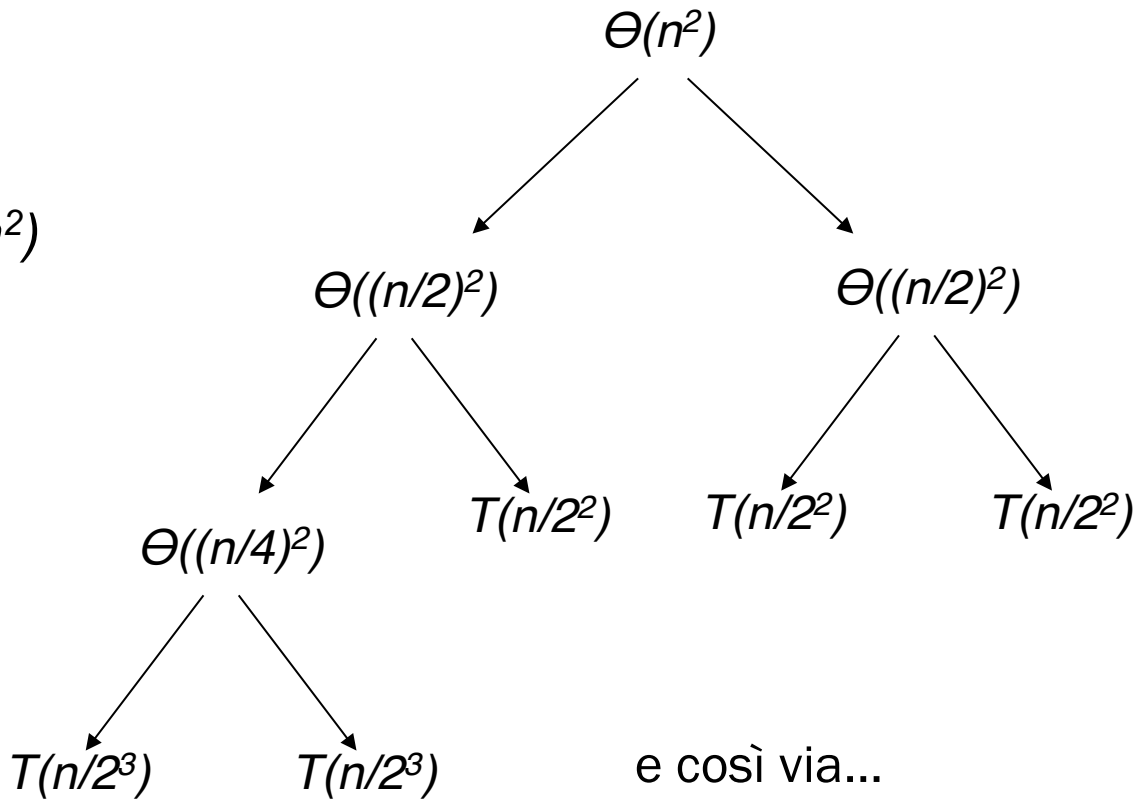
Difficoltà:

- Analoga a quella del metodo iterativo, i conteggi sono molto simili.

Metodo dell'albero (2)

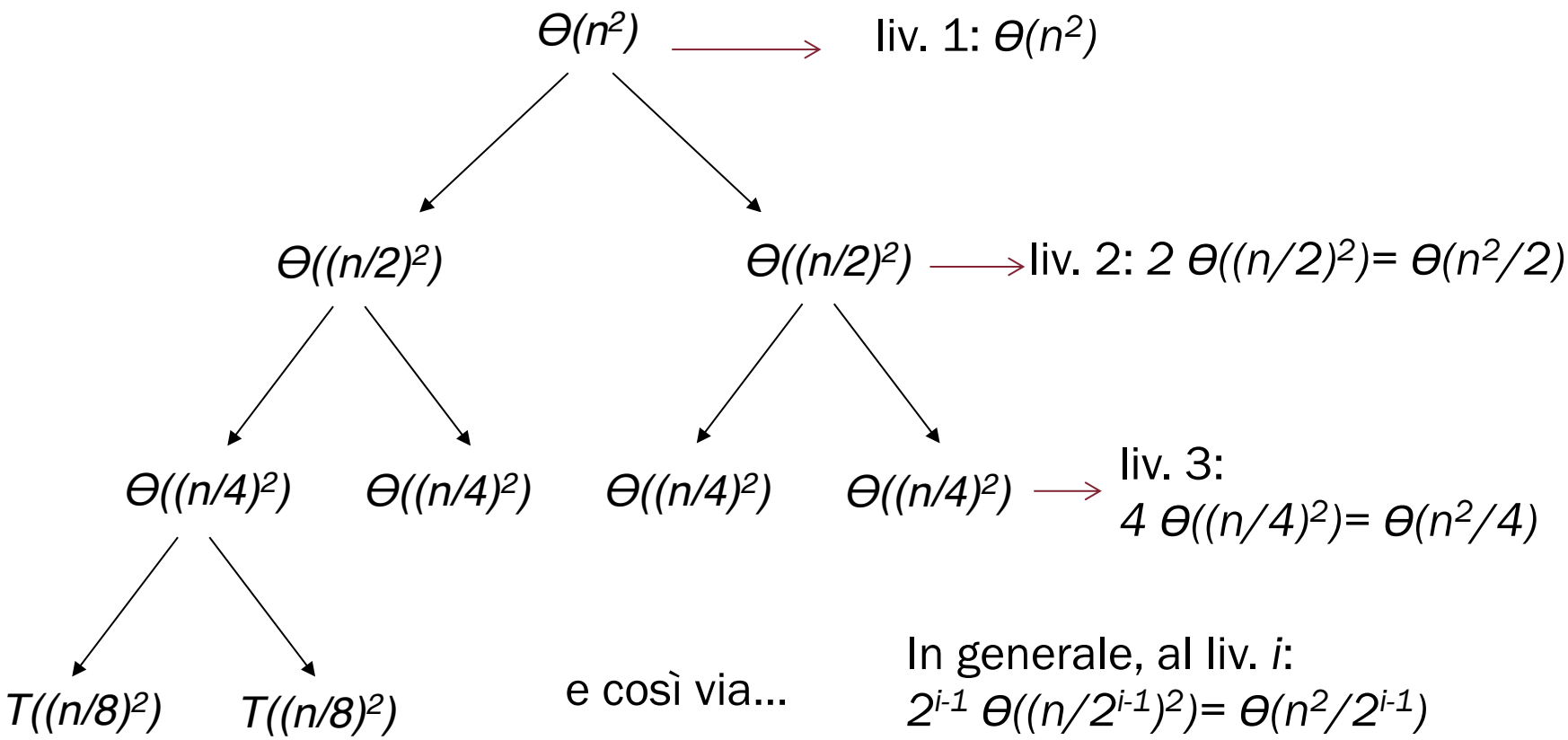
Esempio:

- $T(n) = 2T(n/2) + \Theta(n^2)$
- $T(1) = \Theta(1)$



Metodo dell'albero (3)

Una volta completato l'albero, il costo computazionale è dato dalla somma dei contributi di tutti i livelli (cioè le "righe" in cui sono disposti i nodi) di cui è costituito l'albero.



Metodo dell'albero (4)

In questo caso il numero di livelli dell'albero ha un valore tale che $n/2^{i-1} = 1$, ossia $i - 1 = \log n$ da cui $i = \log n + 1$.

Sommando i contributi di tutti i livelli otteniamo:

$$\sum_{i=1}^{\log n+1} \Theta\left(\frac{n^2}{2^{i-1}}\right) = n^2 \sum_{j=o}^{\log n} \Theta\left(\frac{1}{2^j}\right)$$

Ricordando che

$$1 \leq \sum_{j=0}^{\log n} \left(\frac{1}{2}\right)^j \leq \frac{1}{1-\frac{1}{2}} = 2$$

otteniamo:

$$n^2 \sum_{j=o}^{\log n} \Theta\left(\frac{1}{2^j}\right) = \Theta(n^2)$$

Metodo del teorema principale (1)

Idea:

- Avere una "ricetta" meccanica per risolvere un'equazione di ricorrenza

Difficoltà:

- Funziona **solo** quando l'equazione è della forma

$$T(n)=aT\left(\frac{n}{b}\right)+f(n)$$

$$T(1)=\Theta(1)$$

Metodo del teorema principale (2)

Enunciato del teorema principale:

Dati $a \geq 1$, $b > 1$, una funzione asintoticamente positiva $f(n)$ ed un'equazione di ricorrenza di forma $T(n) = aT(n/b) + f(n)$, $T(1) = \Theta(1)$ vale che:

- **Caso 1**: Se $f(n) = O(n^{\log_b a - \varepsilon})$ per qualche costante $\boldsymbol{\varepsilon} > 0$ allora $T(n) = \Theta(n^{\log_b a})$
- **Caso 2**: Se $f(n) = \Theta(n^{\log_b a})$ allora $T(n) = \Theta(n^{\log_b a} \log n)$
- **Caso 3**: Se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ per qualche costante $\boldsymbol{\varepsilon} > 0$ **E SE** $a*f(n/b) \leq c\, f(n)$ per qualche costante $c < 1$ e per n sufficientemente grande, allora $T(n) = \Theta(f(n))$.

Metodo del teorema principale (3)

Il teorema principale ci dice che in ciascuno dei tre casi vengono confrontati fra loro

$$f(n) \text{ e } n^{\log_b a}$$

ed il costo computazionale è governato dal maggiore dei due.

- Se (caso 1) il più grande dei due è $n^{\log_b a}$, allora il costo è $\Theta(n^{\log_b a})$;
- se (caso 3) il più grande dei due è *f(n)*, allora il costo è $\Theta(f(n))$;
- se (caso 2) sono uguali, allora si moltiplica *f(n)* per un fattore logaritmico.

Metodo del teorema principale (4)

Si noti che “più grande” e “più piccolo” in questo contesto significa **polinomialmente** più grande (o più piccolo), data la posizione all’esponente di ε.

In altre parole, *f(n)* deve essere asintoticamente più grande (o più piccola) rispetto a $n^{log_b a}$ di un fattore n^{ε} per qualche $\varepsilon > 0$.

In effetti fra i casi 1 e 2 vi è un intervallo in cui *f(n)* è più piccola di $n^{log_b a}$, ma non polinomialmente.

Analogamente, fra i casi 2 e 3 vi è un intervallo in cui *f(n)* è più grande di $n^{log_b a}$, ma non polinomialmente.

Metodo del teorema principale (5)

Esempio (caso 1)

$T(n) = 9T(n/3) + \Theta(n)$
$T(1) = \Theta(1)$

- $a = 9,\ b = 3$
- $f(n) = \Theta(n)$
- $n^{\log_b a} = n^{\log_3 9} = n^2$

Poiché

$$f(n) = \Theta(n^{\log_3 9 - \varepsilon}) \text{ con } \boldsymbol{\varepsilon} = 1$$

siamo nel **caso 1**, per cui

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2).$$

Metodo del teorema principale (6)

Esempio (caso 2)

$T(n) = T(\frac{2}{3}n) + \Theta(1)$
$T(1) = \Theta(1)$

- $a = 1, b = 3/2$
- $f(n) = \Theta(1)$
- $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Poiché

$$f(n) = \Theta(n^{\log_b a}) = \Theta(1)$$

siamo nel **caso 2**, per cui

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n).$$

Metodo del teorema principale (7)

Esempio (caso 3)

$T(n) = 3T(n/4) + \Theta(n \log n)$
$T(1) = \Theta(1)$

- $a = 3$, $b = 4$
- $f(n) = \Theta(n \log n)$
- $n^{\log_b a} = n^{\log_4 3} \approx n^{0,7}$

Poiché $f(n) = \Omega(n^{\log_4 3+\varepsilon})$ con, ad esempio, $\varepsilon = 0,2$, siamo nel caso 3 **se possiamo dimostrare** che $3\frac{n}{4} \log \frac{n}{4} \leq c\, n \log n$, per qualche $c < 1$ ed n abbastanza grande.
Ponendo $c = ¾$ otteniamo:

$$3\frac{n}{4} \log \frac{n}{4} \leq \frac{3}{4} n \log n$$

che è vera, quindi $T(n) = \Theta(n \log n)$.

NOTA: la presenza di log n nella f(n) è ininfluente!

Metodo del teorema principale (8)

Esempio (il teorema principale non è applicabile)

$T(n) = 2T(n/2) + \Theta(n \log n)$
$T(1) = \Theta(1)$

- $a = 2, b = 2$
- $f(n) = \Theta(n \log n)$
- $n^{\log_b a} = n^{\log_2 2} = n$

Ora, $f(n) = \Theta(n \log n)$ è asintoticamente più grande di $n^{\log_b a} = n$, **ma non polinomialmente più grande**.

Infatti, *log n* è asintoticamente minore di n^{ε} per qualunque valore di $\varepsilon > 0$. Di conseguenza non possiamo applicare il metodo del teorema principale.

Esercizi

Calcolare la soluzione delle seguenti equazioni di ricorrenza con tutti e quattro i metodi ove possibile:

- $T(n)=2T(n/2)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=3T(n/2)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=3T(n/4)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=2T(n/2)+\Theta(n^2)$ $T(1)=\Theta(1)$
- $T(n)=4T(n/2)+\Theta(n^2)$ $T(1)=\Theta(1)$
- $T(n)=2T(n/2)+\Theta(n^3)$ $T(1)=\Theta(1)$
- $T(n)=16T(n/4)+\Theta(n^2)$ $T(1)=\Theta(1)$
- $T(n)=T(n-1)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=3T(n/2)+\Theta(n \log n)$ $T(1)=\Theta(1)$