METODI DI RISOLUZIONE DI EQUAZIONI DI RICORRENZA

per il calcolo della complessità computazionale asintotica di algoritmi ricorsivi.

METODO ITERATIVO

Questo è il metodo più meccanico di quelli che vedremo, però richiede molti calcoli algebrici e per questo motivo non è sempre il metodo da preferire, soprattutto nei casi in cui è applicabile il TEOREMA MASTER che sarà presentato nel seguito.

L'idea alla base di questo metodo è di sviluppare l'eq di ricorrenza ed esprimerla come somma di termini dipendenti da n e dal caso base.

ESEMPI

1) FATTORIALE
$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 0 \\ T(n-1) + \Theta(1) & \text{se } n > 0 \end{cases}$$

$$T(n) = T(n-1) + d = T(n-2) + \underbrace{d + d}_{2d} = T(n-3) + \underbrace{d + d + d}_{3d} =$$
$$= \underbrace{\dots}_{\text{dopo } k \text{ passi}} = T(n-k) + k\,d$$

Bisogna continuare ad applicare il metodo fino a raggiungere il caso base, cioè finché $T(n-k)$ diventa $T(0)$, questo avviene quando $K = n$.

Per $K = n$, abbiamo
$$T(n) = \underbrace{T(0)}_{d} + \underbrace{n \cdot d}_{\Theta(n)} = \Theta(1) + \Theta(n) = \Theta(n)$$

2) RICERCA DICOTOMICA
$$T(n) = \begin{cases} \Theta(1)\ d & \text{se } n = 0 \text{ o } n = 1 \\ T\left(\frac{n}{2}\right) + \Theta(1)\ d & \text{se } n > 1 \end{cases}$$

$$T(n) = T\Big(\underbrace{\tfrac{n}{2}}_{2^1}\Big) + d = T\Big(\underbrace{\tfrac{n}{4}}_{2^2}\Big) + \underbrace{d + d}_{2d} = T\Big(\underbrace{\tfrac{n}{8}}_{2^3}\Big) + \underbrace{d + d + d}_{3d} =$$
$$= T\Big(\underbrace{\tfrac{n}{16}}_{2^4}\Big) + \underbrace{d + d + d + d}_{4d} = \text{dopo } k \text{ passi} =$$
$$= T\left(\frac{n}{2^k}\right) + k \cdot d$$

$T\left(\frac{n}{2^k}\right)$ diventa $T(1)$ quando $\frac{n}{2^k} = 1$ cioè $k = \log_2 n$

Per $K = \log_2 n$ abbiamo:

$$T(n) = \underbrace{T(1)}_{=d} + \underbrace{\log_2 n \cdot d}_{=\Theta(\log_2 n)} = \Theta(1) + \Theta(\log_2 n) =$$

$$= \Theta(1) + \Theta(\log_2 n) = \Theta(\log_2 n)$$

3) FIBONACCI $\quad T(n) = \begin{cases} \Theta(1) \; d & \text{se } n = 1 \text{ o } n = 2 \\ T(n-1) + T(n-2) + \Theta(1)_d & \text{se } n \geq 3 \end{cases}$

Non si riesce a risolvere questa eq. di ricorrenza con il metodo iterativo, perché il n° di addendi cresce esponenzialmente ad ogni iterazione.

Possiamo però osservare che:

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

da questa disuguaglianza si può trovare un limite superiore O per $T(n)$.

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

da questa disug. si può trovare un limite inferiore Ω per $T(n)$.

<u>Calcolo O</u>

$$T(n) \leq 2T(n-1) + d = 2[2T(n-2) + d] + d \leq$$

$$\leq 2^2 T(n-2) + 2d + d \leq$$

$$\leq 2^2 [2T(n-3) + d] + 2d + d \leq$$

$$\leq 2^3 T(n-3) + 2^2 d + 2^1 d + 2^0 d \leq \ldots \leq$$ dopo k passi

$$\leq 2^k T(n-k) + \sum_{i=0}^{k-1} 2^i \cdot d$$

Bisogna continuare finché $T(n-k)$ diventa $T(1)$, cioè quando $k = n-1$.

Per $k = n-1$, abbiamo $T(n) \leq \underbrace{2^{n-1} \cdot T(1)}_{O(2^n) \quad d} + \sum_{i=0}^{n-2} 2^i \cdot d$

$$= O(2^n) + d \cdot \sum_{i=0}^{n-2} 2^i$$

SERIE GEOMETRICA (con $x \neq 1$)

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

$$= O(2^n) + d\left(\frac{2^{n-2+1}-1}{2-1}\right) = O(2^n) + O(2^n) = O(2^n)$$

prendo il max

$$(2^{n-1}-1) = O(2^n)$$

<u>Calcolo Ω</u>

$$T(n) \geq 2T(n-2) + d \geq 2[2T(n-4) + d] + d \geq$$

$$\geq 2^2 T(n-4) + 2d + d \geq$$

$$\geq 2^2[2T(n-6) + d] + 2d + d \geq$$

$$\geq 2^3 T(n-6) + 2^2 d + 2^1 d + 2^0 d \geq \dots \geq$$ dopo k passi

$$\geq 2^k T(n-2k) + \sum_{i=0}^{k-1} 2^i d$$

Bisogna continuare finché $T(n-2k)$ diventa $T(1)$, ciò quando $n-2k = 1 \Rightarrow k = \frac{n-1}{2}$

Per $k = \frac{n-1}{2}$, abbiamo

$$T(n) \geq \underbrace{2^{\frac{n-1}{2}} \cdot T(1)}_{\Omega(2^{n/2-1})\, d} + \sum_{i=0}^{\frac{n-1}{2}-1} 2^i \cdot d = \Omega(2^{n/2-1}) + d \cdot \frac{2^{\frac{n-1}{2}-1+1}-1}{2-1}$$

$$= \Omega(2^{n/2-1}) \dots = \Omega(2^{n/2-1}) + \Omega(2^{n/2-1}) = \Omega(2^{n/2-1})$$

Si può concludere che la complessità computazionale di Fibonacci ricorsivo è esponenziale in n, visto che

$$c_1 2^{n/2-1} \leq T(n) \leq c_2 2^n$$

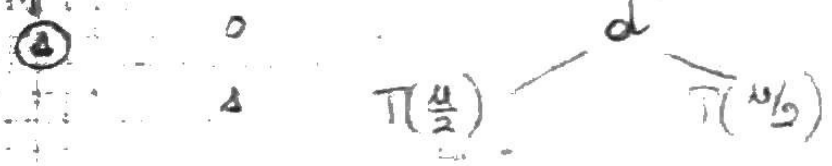
METODO DELL'ALBERO DI RICORSIONE

È una tecnica per rappresentare graficamente lo sviluppo della complessità generata da un algoritmo ricorsivo, così da fare un'ipotesi sulla complessità stessa, che però dovrà essere dimostrata in seguito, per es con il metodo di sostituzione che vedremo dopo.

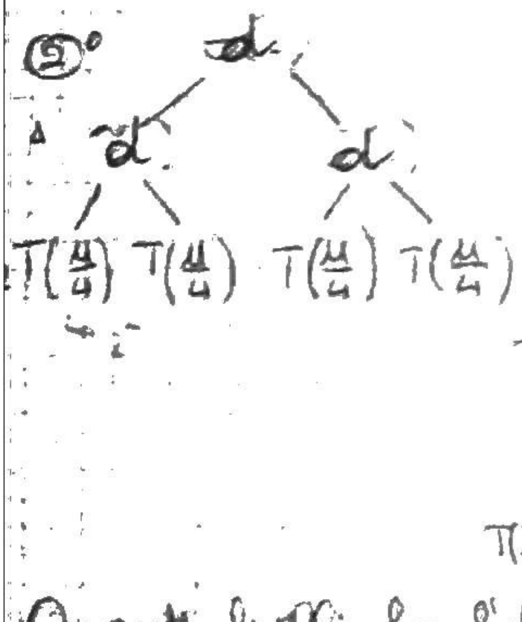
ESEMPI

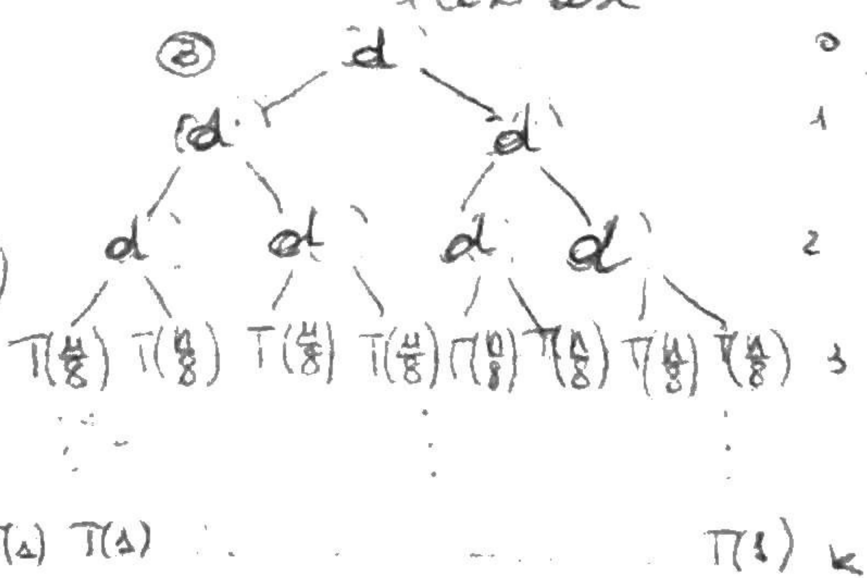
1) $$T(n) = \begin{cases} \Theta(1) \ d & \text{se } n = 1 \\ 2T\left(\frac{n}{2}\right) + \underbrace{\Theta(1)}_{d} & \text{altrim} \end{cases}$$

Costruzione albero: si parte dalla radice con i suoi figli, dove alla radice c'è il contributo (costo) di tutte le parti dell'algoritmo che non dipende dalle chiamate ricorsive [cioè il secondo addendo dell'eq. di ricorrenza]. La radice avrà un numero di figli pari al n° di chiamate ricorsive (in questo caso 2) e tengo traccia della dimensione del sottoproblema che devo risolvere.



Poi si itera il procedimento sui figli fino a raggiungere il livello ultimo delle foglie in cui tutte le foglie sono il caso base





Quanti livelli ha l'albero? Cioè, quanto vale k?

Arrivo al caso base $T(1)$ quando $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

Una volta completato l'albero, la complessità è data dalla somma delle complessità di tutti i livelli dell'albero.

Costo livello 0: d
" " 1: $d + d) = 2\,d$
" " 2: $d + d + d + d) = 2^2\,d$
" " 3: $= 2^3\,d$

Costo livello i: $2^i \cdot d$

~~Quanti livelli ha l'albero? Che poi sarebbe?~~
~~Arrivo al caso base T(1) quando ... O(1) ..., cioè quando~~

Il numero dei livelli come detto prima è $\log_2 n$, quindi sommando i costi di tutti i livelli si ottiene:

$$T(n) = \sum_{i=0}^{\log_2 n} 2^i \cdot d = d \cdot \sum_{i=0}^{\log_2 n} 2^i = d\left(\frac{2^{\log_2 n + 1} - 1}{2-1}\right) =$$

SERIE GEOMETRICA

$$2 \cdot 2^{\log_2 n} - 1 = 2 \cdot n - 1 = O(n)$$

$$= d\, O(n) = O(n)$$

2) $T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ 2T(\frac{n}{2}) + \Theta(n^2) & \text{se } n > 1 \end{cases}$

①
$$\begin{array}{ll} 0 & d n^2 \\ 1 & T(n/2) \quad T(n/2) \end{array}$$

②
$$\begin{array}{ll} d n^2 & 0 \\ d\left(\frac{n}{2}\right)^2 \quad d\left(\frac{n}{2}\right)^2 & 1 \\ T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad T(n/4) \quad T(n/4) & 2 \end{array}$$

③
$$\begin{array}{ll} d n^2 & 0 \\ d\left(\frac{n}{2}\right)^2 \quad d\left(\frac{n}{2}\right)^2 & 1 \\ d\left(\frac{n}{4}\right)^2 \quad d\left(\frac{n}{4}\right)^2 \quad d\left(\frac{n}{4}\right)^2 \quad d\left(\frac{n}{4}\right)^2 & 2 \\ T(\frac{n}{8}) \quad T(\frac{n}{8}) \quad T(\frac{n}{8}) \quad T(\frac{n}{8}) & 3 \\ \vdots & \\ T(1) \quad\quad T(1) & k \end{array}$$

Quanti livelli ha l'albero?
Arrivo al caso base quando $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

Che costo ha ogni livello?

livello 0: $d\,n^2$

livello 1: $d\left(\frac{n}{2}\right)^2 + d\left(\frac{n}{2}\right)^2 = 2\,d\left(\left(\frac{n}{2}\right)^2\right) = 2\,d\cdot\frac{n^2}{(2)^2}$

livello 2: $2^2\,d\left(\frac{n}{4}\right)^2 = 2^2\,d\,\frac{n^2}{(2^2)^2}$,

livello 3: $2^3\,d\left(\frac{n}{8}\right)^2 = 2^3\cdot d\,\frac{n^2}{(2^3)^2}$,

livello i: $2^i\cdot d\left(\frac{n^2}{(2^i)^2}\right) = 2^i d\,\frac{n^2}{(2^i)^2} = d\left(\frac{n^2}{2^i}\right)$

Somma il costo di tutti i livelli (che sono $\log_2 n + 1$, da 0 a $k = \log_2 n$).

$$T(n) = \sum_{i=0}^{\log_2 n} d\,\frac{n^2}{2^i} = \sum_{i=0}^{\log_2 n} (d\,n^2)\cdot\left(\frac{1}{2^i}\right) = d\,n^2\cdot\sum_{i=0}^{\log_2 n}\frac{1}{2^i}$$

$\Downarrow$

$$\sum_{i=0}^{\log_2 n}\left(\frac{1}{2}\right)^i \leq \sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i = \frac{1}{1-\frac{1}{2}} = 2$$

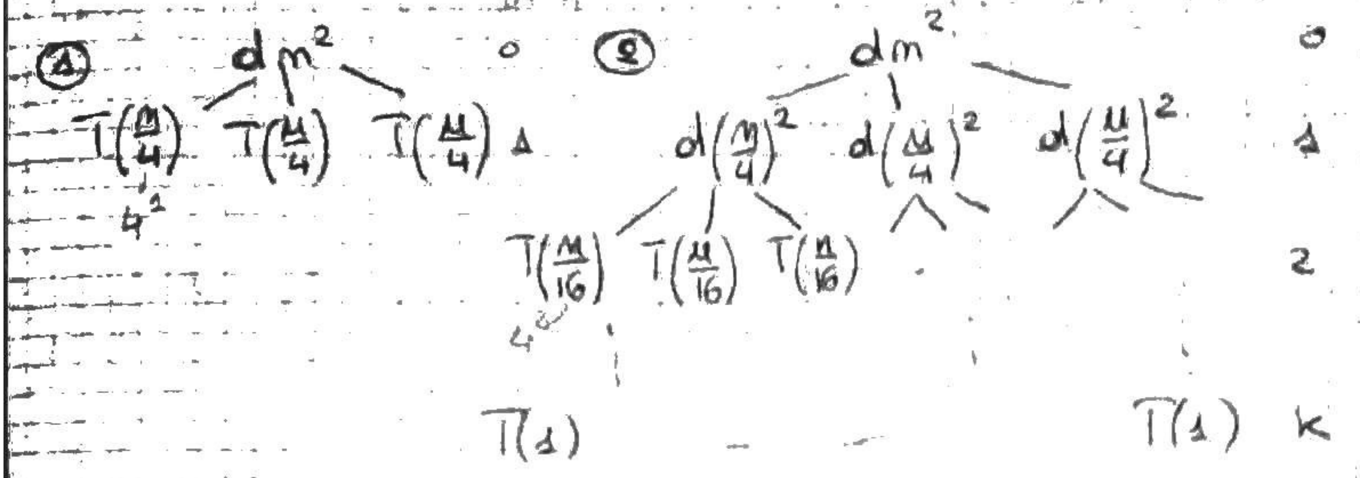
$|1/2| < 1$

$\uparrow$

SERIE GEOMETRICA DECRESCENTE INFINITA $\Rightarrow \sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ con $|x| < 1$

$$T(n) \leq 2\,d\,n^2 = O(n^2)$$

3) $T(n) = \begin{cases} \Theta(1) & \text{se } n=1 \\ 3T\left(\frac{n}{4}\right)+\Theta(n^2) & \text{altrim.} \end{cases}$



Quanti livelli ha l'albero? Cioè quanto vale k?
Arrivo al caso base $T(1)$ quando $\frac{n}{4^k} = 1 \Rightarrow k = \log_4 n$

Che costo ha ogni livello?

livello 0: dn^2

livello 1: $d\left(\frac{n}{4}\right)^2 + d\left(\frac{n}{4}\right)^2 + d\left(\frac{n}{4}\right)^2 = 3d\left(\frac{n^2}{4^2}\right)$

livello 2: $3^2 d\left(\frac{n^2}{(4^2)^2}\right)$

livello 3: $3^3 d\left(\frac{n^2}{(4^3)^2}\right)$

livello i: $3^i d\left(\frac{n^2}{(4^i)^2}\right) = 3^i \cdot d \cdot \left(\frac{n}{4^i}\right)^2 = 3^i\, d\, n^2 \underline{\left(\frac{1}{4^2}\right)^i} = \left(\frac{3}{16}\right)^i dn^2$

Sommo i costi dei $\log_4 n + 1$ livelli:

$$T(n) = \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i dn^2 \leq dn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i = dn^2 \frac{1}{1-\frac{3}{16}} = \frac{16}{13} dn^2 = O(n^2)$$

$\left|\frac{3}{16}\right| < 1$ Serie geom. decresc. infinita

Abbiamo detto che il metodo dell'albero di ricorsione ci permette di fare un'ipotesi su come si comporta la complessità degli algoritmi ricorsivi, dobbiamo poi verificare questa ipotesi usando, per esempio, il metodo di sostituzione.

METODO DI SOSTITUZIONE per O e Ω

Questo metodo serve soprattutto nelle dimostrazioni, perché occorre avere una buona ipotesi da andare a sostituire nella ricorrenza.

Per verificare se l'ipotesi è corretta si usa l'induzione matematica.

ESEMPI:

① $T(n) = 3T\left(\frac{n}{4}\right) + \underset{d n^2}{\Theta(n^2)}$ Ipotesi $T(n) = O(n^2)$

Dobbiamo dimostrare che

$\exists\, c$ e n_0 t.c. $0 \le T(n) \le c \cdot n^2 \quad \forall n \ge n_0$

Caso base: $n = 1 \quad T(1) = d \le c \cdot 1 \quad \forall c \ge d$

Passo induttivo: assumiamo che il limite $T(n') \le c n'^2$ sia vero $\forall n' < n$, quindi anche per $\frac{n}{4}$, e dim per n.

$$T(n) = 3T\left(\frac{n}{4}\right) + dn^2 \overset{\text{per ipotesi induttiva}}{\le} 3c\left(\frac{n}{4}\right)^2 + dn^2 = \frac{3}{16}cn^2 + dn^2 \le cn^2$$

dividiamo per n^2: $\frac{3}{16}c + d \le c \Rightarrow d \le c - \frac{3}{16}c$

$$\Rightarrow 16d \le 13c \Rightarrow c \ge \frac{16}{13}d \quad \text{e } n_0 = 0$$

METODO DI SOSTITUZIONE

$$T(n) = \begin{cases} \Theta(1) \; d & \text{se } n=0 \text{ o } n=1 \\ T\left(\frac{n}{2}\right)+\underset{d}{\Theta(1)} & \text{se } n>1 \end{cases}$$

ipotesi $T(n)=O(\log_2 n)$

Dobbiamo dire che:

$\exists c, n_0 : 0 \leq T(n) \leq c \log_2 n \quad \forall n \geq n_0$

<u>Caso base</u>. se $n=1$: $T(1) = d \leq c \log_2 1 \Rightarrow d \leq c \cdot 0 = 0$

è falso!

Si può facilmente superare questo probl. grazie alla notaz. asint. perché dobbiamo dire per $n \geq n_0$ con n_0 costante <u>arbitrariamente</u> scelta. Quindi proviamo per $n=2$ che diventa il nostro nuovo caso base.

<u>Caso base</u>: se $n=2$ ~~$T(2) = 2d \leq c \log_2 2 = c \cdot 1 \Rightarrow$ per $c \geq d$~~

$T(2) = T\left(\frac{2}{2}\right) + d = T(1) + d = d + d = 2d \leq c \log_2 2 = c \cdot 1$

$\Rightarrow c \geq 2d$

<u>Passo induttivo</u>: assumiamo che il limite $T(n') \leq c \log_2 n'$ sia vero $\forall n' < n$, quindi anche per $n/2$, e dim per n

$T(n) = T\left(\frac{n}{2}\right) + d \overset{\text{x ip induttiva}}{\leq} c \log_2\left(\frac{n}{2}\right) + d = c\left(\log_2 n - \log_2 2\right) + d =$

$= c \log_2 n - c + d \leq c \log_2 n$ vero se $c \geq d$

METODO ITERATIVO

$$T(n) = \begin{cases} \underbrace{\Theta(1)}_{d} & \text{se } n = 0 \\ T(n-1) + \underbrace{\Theta(n)}_{dn} & \text{altrim.} \end{cases}$$

$$\cancel{T(n) = T(n-1) + dn = T(n-2) + \underbrace{dn + dn}_{2dn} = T(n-3) + 3dn}$$

~~dopo k passi~~ $\cancel{= T(n-k) + kdn}$

$$T(n) = T(n-1) + dn = T(n-2) + d(n-1) + dn =$$

$$= T(n-3) + d(n-2) + d(n-1) + dn = \dots = \text{dopo } k \text{ passi}$$

$$= T(n-k) + \sum_{i=0}^{k-1} d(n-i) = T(n-k) + d \sum_{i=0}^{k-1} n-i$$

Bisogna continuare fino a che non si arriva al caso base $T(0)$, cioè quando $n-k=0 \to k=n$

Per $k=n$, abbiamo: $\to$ serie aritm. di Gauss

$$T(n) = \underbrace{T(0)}_{d} + d \sum_{i=0}^{n-1} n-i = d + d\left[\sum_{i=0}^{n} n - \sum_{i=0}^{n} i\right] =$$

$$\dots = \Theta(1) + \Theta(n^2) = \Theta(n^2)$$

ragione, commettendo un leggero abuso di notazione, ignoriamo il problema assumendo che *n* sia sempre una potenza di $b > 1$.

5.4.1 Enunciato del teorema principale

- Dati $a \geq 1$, $b > 1$, una funzione asintoticamente positiva *f(n)* ed un'equazione di ricorrenza di forma $T(n) = aT(n/b) + f(n)$, $T(1) = \Theta(1)$ valgono le seguenti proprietà:

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

Quanto sopra ci dice che in ciascuno dei tre casi vengono confrontati fra loro *f(n)* e $n^{\log_b a}$.

Per inciso, mentre già sappiamo che *f(n)* rappresenta il costo di ricombinazione delle soluzioni ai sottoproblemi, osserviamo ora che il valore $\log_b a$ è legato alla relazione che c'è fra il numero dei sottoproblemi in cui si suddivide un problema e la dimensione dei sottoproblemi in rapporto alla dimensione del problema.

Il teorema principale ci dice che "vince" il maggiore fra *f(n)* e $n^{\log_b a}$, ossia la complessità è governata dal maggiore dei due:

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

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.

In tali intervalli (casi 1 e 3) il metodo del teorema principale non può essere usato, come non può essere usato se (caso 3) non vale la condizione $a\,f(n/b) \leq c\,f(n)$.

Esempio 5.12

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

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

Poiché *f(n)* = $O(n^{\log_3 9-\varepsilon})$ con $\varepsilon = 1$, siamo nel caso 1, per cui $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

Esempio 5.13

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

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

Poiché $f(n) = \Theta(n^{\log_b a})$ siamo nel caso 2, per cui $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n)$.

Esempio 5.14

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

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

Poiché $f(n) = \Omega(n^{\log_4 3+\varepsilon})$ con $\varepsilon = 0,3$, 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 $T(n) = \Theta(n \log n)$.

EX. 3

Function $F(u)$

if $m>2$ then begin

$x := m$;

$m := m \text{ div } 2$;

log m volte

while ($m > 1$) do begin

$x := x \cdot m$;

$m := m \text{ div } 2$;

end

$F := x + F(u \text{ div } 4)$

end

Complessità:

$$T(u) = \begin{cases} \Theta(1) & \text{se } m \le 2 \\ T(u/4) + \log m & \text{se } m > 2 \end{cases}$$

non si può applicare il teo principale.

Metodo iterativo:

$$T(u) = \log m + T(u/4) = \log m + \log \frac{m}{4} + T(u/16) = \sum_{i=0}^{\log m - 1} \log(u/4^i) + \Theta(1) =$$

$$= \sum_{i=0}^{\log m-1} (\log m - \log 4^i) + \Theta(1) = \sum_{i=0}^{\log u - 1} \log m - \sum_{i=0}^{\log u - 1} i \cdot \log 4 + \Theta(1) =$$

$$\boxed{\sum_{i=0}^{k} t = \frac{k(k+1)}{2}}$$

$$= \log^2 m - \frac{\cancel{2}(\log(m-1))\log m}{\cancel{2}} + \Theta(1) = \log^2 u - \log^2 m + \log u = \Theta(\log m)$$

EX. 4

Scrivere una funzione che abbia complessità $\Theta(u^2 \log m)$.

2° caso del teo principale

$$T(u) = 4T(u/2) + \Theta(u^2)$$

$\log_b a = 2$ $\quad b = 2$, $a = 4$

$f(u) = \Theta(u^2)$

Function $F(u)$

if $m = 0$ then $F := 0$

else { for $i = 1$ to m do

for $j = 1$ to m do $x := x + 1$;

return $x + F(u/2) + F(u/2) * F(u/2) - F(u/2)$

}

Ex. 5

Scrivere una funzione ricorsiva la cui complessità verifica:

$T(n) = 7T(n/2) + \Theta(n \log n)$

```
Function F(n)

        if n<2 then F:=2

              else begin

n volte        for i:=1 to n do

               { m:=n div 2

log n            while m>1 do
volte
                     x:=i+1

                     m:=m div 2 }

               return x + F(n/2) * F(n/2) - 5F(n/2) * F(n/2) - 3F(n/2)
                      - 2F(n/2) * F(n/2)
```