Relazioni di ricorrenza

19 marzo 2021

Ricorrenze

- Per analizzare il tempo di esecuzione di un algoritmo, dobbiamo distinguere il caso degli algoritmi ricorsivi (che sono tanti, per esempio tutti gli algoritmi basati sul paradigma divide-et-impera)
- L'analisi di un algoritmo ricorsivo ci porta a dover risolvere una relazione di ricorrenza per T(n) (o S(n)).
- Dobbiamo imparare come risolverle.
- Esistono vari metodi :
  - Unrolling (o iterazione)
  - Alberi di ricorsione
  - Sostituzione (per induzione)
- Esistono poi un Teorema generale e il Master Theorem

Tempo di esecuzione di Mergesort

Dividi

```
MERGE-SORT(A, p, r)
1 if p < r
2    then q ← ⌊(p + r)/2⌋
3         MERGE-SORT(A, p, q)
4         MERGE-SORT(A, q + 1, r)
5         MERGE(A, p, q, r)
```

Sia **T(n)** il tempo di esecuzione di MERGE−SORT su un array di taglia n.

T(n) = ?

C'è 1 confronto nella linea 1: $\Theta(1)$.
Poi nel caso generale:
un assegnamento $\Theta(1)$ ……. oppure $O(n)$?
e l'esecuzione del MERGE: $\Theta(n)$
per un totale di al più: $\Theta(1) + \Theta(1) + \Theta(n)$ e poi
2 esecuzioni di MERGE−SORT su due array di circa **n/2** elementi, cioè?

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

Relazioni di ricorrenza

Una relazione di ricorrenza per una funzione T(n) è un'equazione o una disequazione che esprime T(n) rispetto a valori di T su variabili più piccole, completata dal valore di T nel caso base (o nei casi base).

Esempio:

$$\begin{cases} T(2) = 5 & \\ T(n) = 2\,T(n/2) + 3n & \text{per } n=2^k,\ n > 2 \end{cases}$$

$T(8) = ???$

$T(8) = 2T(4) + 3 \cdot 8$

$T(4) = 2T(2)+ 3 \cdot 4$

$T(2) = 5$

$T(4) = 10 + 3 \cdot 4 = 22$

$T(8) = 44 + 3 \cdot 8 = 68$

E se $n = 64$, $T(n) = ?$

$T(n) = 5\,n/2 + 3\,n\,(\log_2 n - 1)$ per $n=2^k$

NOTA: in generale non avremo costanti esatte, e ci basterà

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

A Recurrence Relation for Mergesort

Def. T(n) = number of comparisons to mergesort an input of size n.

Mergesort recurrence.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 2 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{\Theta(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$

Solution. $T(n) = \Theta(n \log_2 n)$.

Assorted proofs.

We will describe several ways to prove this recurrence.

First, we will solve a simplified recurrence:

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

MergeSort

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

$$\begin{cases} T(n) = 2\,T(n/2) + f(n) & \text{dove } f(n)= \Theta(n)\text{, cioè } a_1 n \leq f(n) \leq c_1 n \text{ per } n>2 \\ T(2)=g(n) & \text{dove } g(n)= \Theta(1)\text{, cioè } a_2 \leq g(n) \leq c_2 \end{cases}$$

Scegliendo $c=\max\{c1, c2\}$ e $a =\min\{a1, a2\}$

$$\begin{cases} T(n) \leq 2\,T(n/2) + cn & \text{con } T(2) \leq c \\ T(n) \geq 2\,T(n/2) + an & \text{con } T(2) \geq a \end{cases}$$

Da $T(n) \leq 2\,T(n/2) + cn$ ricaveremo $T(n) =O(n \log_2 n)$, col metodo unrolling.

Da $T(n) \geq 2\,T(n/2) + an$ ricaveremo $T(n) =\Omega(n \log_2 n)$, per sostituzione.

Da cui $T(n) = \Theta(n \log_2 n)$.

Metodo di sostituzione

Mergesort: Sia $n=2^k$ $T(n) \geq 2T(n/2)+ an$

$T(2) \geq a$

- ipotizziamo che la soluzione sia $T(n) \geq d\, n \log_2 n$ per una costante d opportuna
- verifichiamolo con l'induzione (forte)

Base: $T(2) \geq a \geq d\ 2 \log_2 2 = 2d$ per ogni $2d \leq a$, $d \leq a/2$

Ipotesi induttiva: supponiamo che $T(n/2) \geq d\ n/2 \log_2(n/2)$

Passo induttivo: $T(n) \geq 2T(n/2) + an \geq 2\ d\ n/2 \log_2(n/2) + an =$

$= dn (\log_2 n - 1) + an =$

$= dn \log_2 n + (a - d)\, n \geq dn \log_2 n$ sse

$a - d \geq 0$, sse $d \leq a$

quindi $T(n) \geq d\, n \log_2 n$ per $d \leq a/2$, da cui $T(n)= \Omega(n \log_2 n)$.

Metodo di iterazione (Unrolling)

Esempio MergeSort: Sia $n=2^k$ $T(n) \leq 2\,T(n/2) + c\,n$

$T(2) \leq c$

$T(n) \leq \mathbf{2\,T(n/2) + cn}$

ma $T(n/2) \leq 2T(n/2^2) + c\,n/2$

$\leq 2\,(2T(n/2^2) + c\,n/2\,) + cn$

$\leq 2^2\,T(n/2^2) + 2 \cdot c\,n/2 + cn$

$= 2^2\,T(n/2^2) + c\,n + c\,n = \mathbf{2^2\,T(n/2^2) + 2c\,n}$

ma $T(n/2^2) \leq 2T(n/2^3) + c\,n/2^2$

$\leq 2^2\,(2T(n/2^3) + c\,n/2^2) + 2c\,n$

$= 2^3\,T(n/2^3) + 2^2 \cdot c\,n/2^2 + 2c\,n$

$= 2^3\,T(n/2^3) + c\,n + 2c\,n = \mathbf{2^3\,T(n/2^3) + 3c\,n}$

....

Metodo di iterazione (Unrolling)

$T(n) \leq \mathbf{2^3\, T(n/2^3) + 3\, c\, n}$

Andrebbe provato per induzione!

....

$\leq \mathbf{2^i\, T(n/2^i) + i\, c\, n}$

proseguendo in questo modo dopo i iterazioni avremo

Finché raggiungiamo T(2), per cui $T(2) \leq c$

$= n/2 \bullet T(2) + c\, n\, (\log_2 n - 1)$

$\leq c\, n/2 + c\, n\, (\log_2 n - 1)$

$= cn\,/2 + c\, n \log_2 n - cn$

$= c\, n \log_2 n - c/2\, n$

$\leq \mathbf{c\, n \log_2 n}$

$n/2^i = 2$ quando $n = 2^{i+1}$

$i+1 = \log_2 n$

$i = \log_2 n - 1$

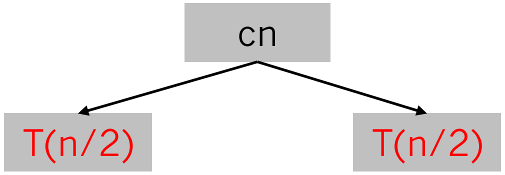
$2^i = n/\ 2$

Quindi $\mathbf{T(n) = O(\, n \log n)}$

Albero di ricorsione per MergeSort

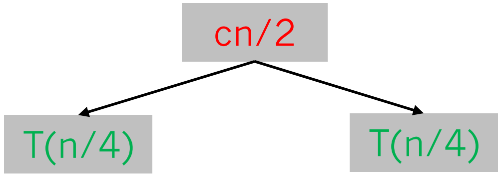
$T(n) = 2\,T(n/2) + cn$ con $T(2) = c$

Albero per T(n):

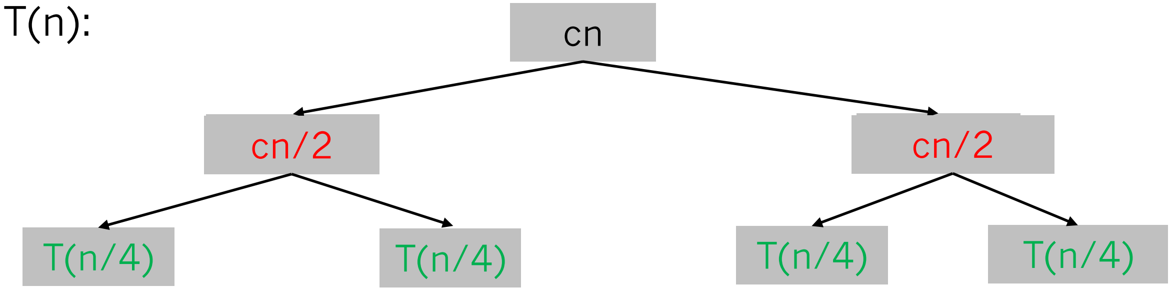


Albero per T(n/2):

$T(n/2) = 2\,T(n/4) + cn/2$



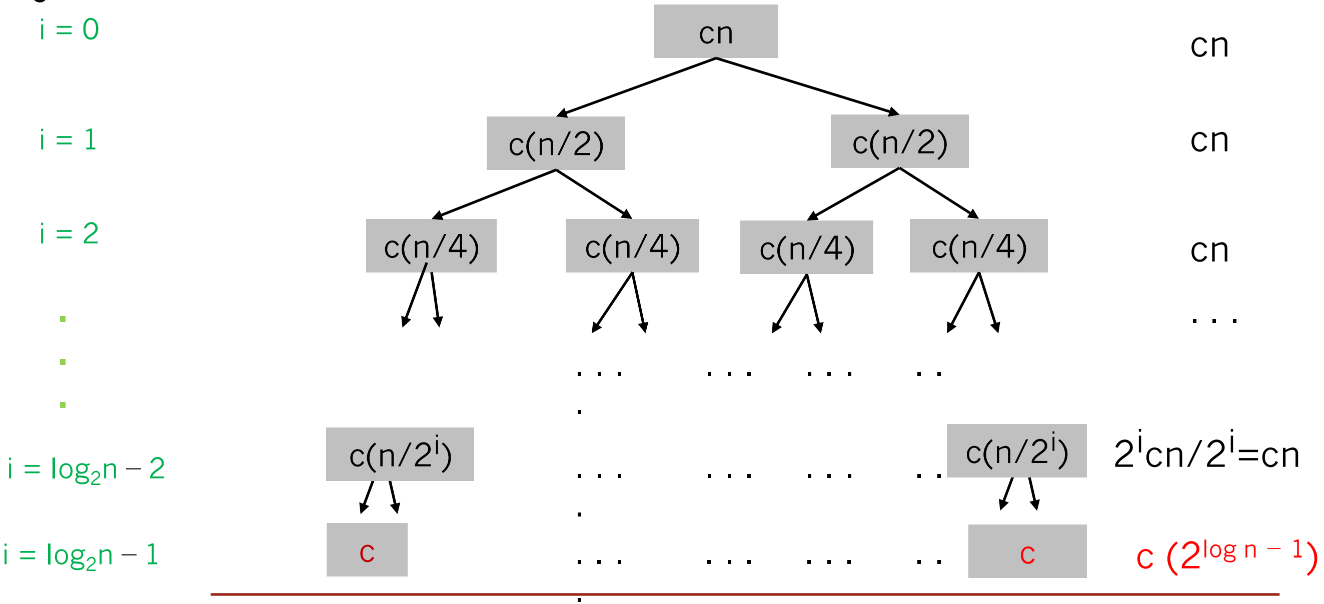
Albero per T(n):



Albero di ricorsione per MergeSort

$T(n) = 2\,T(n/2)+cn$

$T(2) = c$



$$T(n) = c\,(2^{\log n - 1}) + c\,n\,(\log_2 n - 1)$$

$$= c\,n/2 + c\,n\,(\log_2 n - 1)$$

$n/2^i = 2$ sse $n = 2^{i+1}$ sse $i+1 = \log_2 n$

sse $i = \log_2 n - 1$ sse $2^i = n/2$

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

Analisi ricorrenza esatta per MergeSort

$$T(n) = \begin{cases} c & \text{if } \quad n = 2 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{\Theta(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$

$\boxed{x \le \lceil x \rceil < x+1}$ $\boxed{\lfloor x \rfloor \le x \le \lceil x \rceil}$ $\boxed{x-1 < \lfloor x \rfloor \le x}$

$\boxed{3{,}45 \le \lceil 3{,}45 \rceil = 4 < 4{,}45}$ $\boxed{3 \le \lceil 3{,}45 \rceil \le 4}$ $\boxed{2{,}45 < \lfloor 3{,}45 \rfloor = 3 \le 3{,}45}$

Ogni intero n, anche se non è una potenza di 2, è comunque compreso fra due potenze di 2:

$n=2^{\log_2 n}$ dove $\log_2 n$ potrebbe non essere intero.

Però: $\lfloor \log_2 n \rfloor \le \log_2 n \le \lceil \log_2 n \rceil$

E quindi $2^{\lfloor \log_2 n \rfloor} \le n = 2^{\log_2 n} \le 2^{\lceil \log_2 n \rceil}$ (per esempio $2^5 \le 48 \le 2^6$).

Analisi ricorrenza esatta per MergeSort

$$T(n) \leq \begin{cases} c & \text{if } n = 2 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{\Theta(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$

$$\boxed{x \leq \lceil x \rceil < x+1} \qquad \boxed{x-1 < \lfloor x \rfloor \leq x}$$

Ho dimostrato che per $n = 2^k$ si ha $T(n) \leq c\ n \log_2 n = c2^k\ k$.

Per n qualsiasi:

$T(n) \leq T(2^{\lceil \log 2\ n \rceil}) \leq c\ 2^{\lceil \log 2\ n \rceil} \lceil \log_2 n \rceil \leq c\ 2^{(\log 2\ n+1)} (\log_2 n + 1) =$

$= 2c\ n (\log_2 n+1) = 2c\ n \log_2 n + 2cn \leq 4c\ n \log_2 n$ (se $1 \leq \log_2 n$, cioè $n \geq 2$)

da cui $T(n) = O(n\log n)$.

Analogamente per $T(n) = \Omega(n \log_2 n)$.

Analisi ricorrenza esatta per MergeSort (induzione)

$$T(n) \le \begin{cases} c & \text{if } n = 2 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{c\,n}_{\text{merging}} & \text{otherwise} \end{cases}$$

In alternativa, è anche possibile dimostrare per induzione su n che

$T(n) \le cn\lceil \log_2 n \rceil$

per ogni $n \ge 2$.

Prova (per esercizio)

Altre relazioni di ricorrenza

Consideriamo la sotto-famiglia

- $T(n) = qT(n/2) + cn$ con $T(2)=c$

  per $q=1$ allora $T(n)= ?$

  $q=2$ allora $T(n)= \Theta(n \log_2 n)$ (MergeSort)

  $\boxed{q>2 \text{ allora } T(n)= ?}$

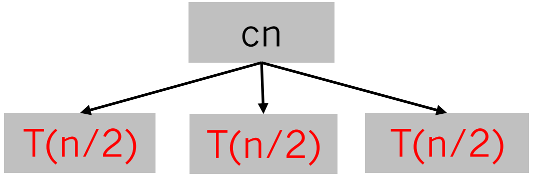
- $T(n)=2T(n/2) + cn^2$

  $T(n)= ?$

Albero di ricorsione per il caso q=3

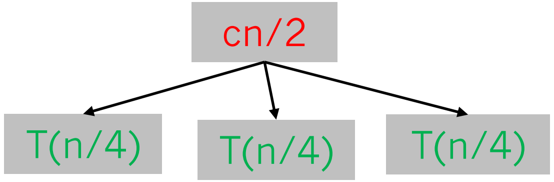
$T(n) = 3\ T(n/2) + cn$
$T(2) = c$

Albero per T(n):

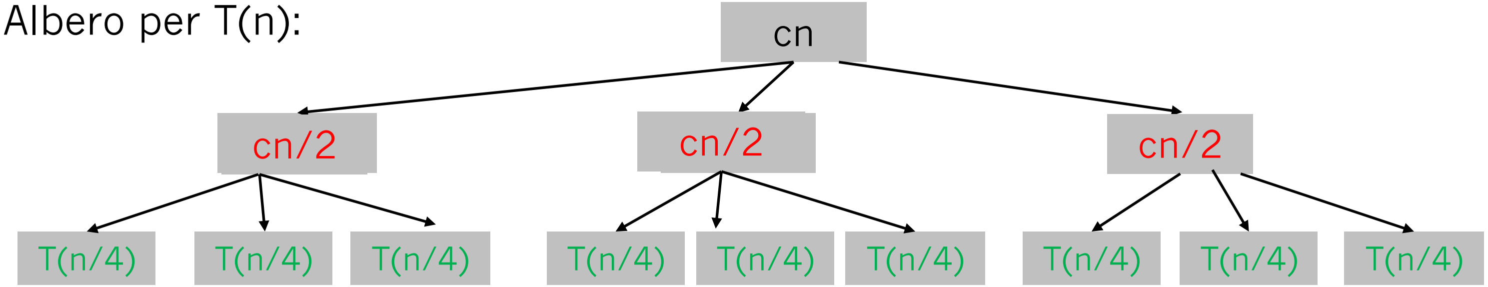


Albero per T(n/2):

$T(n/2) = 3\ T(n/4) + cn/2$



Albero per T(n):



Albero di ricorsione per il caso q=3

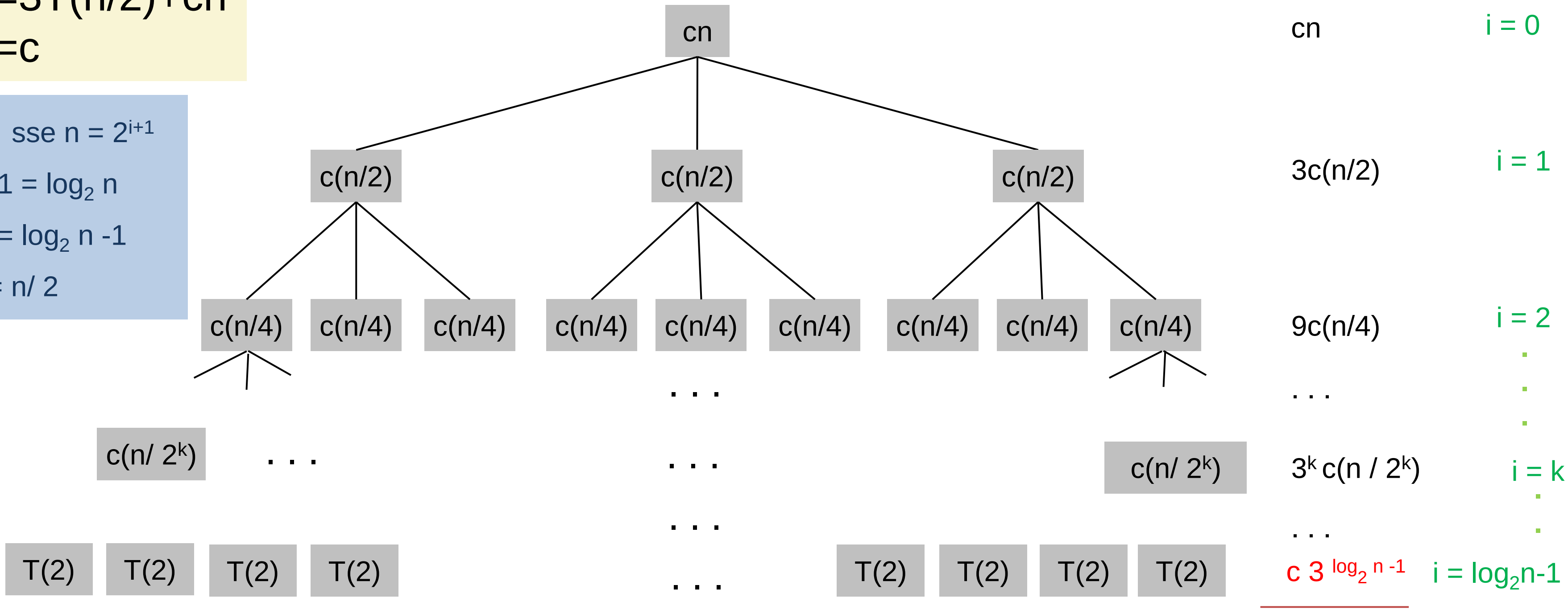
$T(n)=3T(n/2)+cn$
$T(2)=c$

$n/2^i = 2$ sse $n = 2^{i+1}$

sse $i+1 = \log_2 n$

sse $i = \log_2 n - 1$

sse $2^i = n/2$



$$T(n) = c\,(3^{\log_2 n - 1}) + c\,n\,(1+3/2+ (3/2)^2 + \ldots + (3/2)^{\log_2 n - 2})$$

Da ricordare: serie geometrica

Somme finite: Se $\alpha \neq 1$ allora

$$\sum_{i=0}^{n} \alpha^i = \frac{\alpha^{n+1} - 1}{\alpha - 1} \qquad (1)$$

Somme infinite: Se $0 < \alpha < 1$. Allora

$$\sum_{i=0}^{\infty} \alpha^i = \frac{1}{1 - \alpha} \qquad (2)$$

T(n)=3T(n/2)+cn
T(2)=c

Caso q=3

$$T(n) = c\,(3^{\log_2 n - 1}) + c\,n\,(1+3/2+(3/2)^2 + \dots + (3/2)^{\log_2 n - 2})$$

Poiché $c\,(3^{\log_2 n - 1}) \leq c\,n\,(3/2)^{\log_2 n - 1}$ (*verifica per esercizio*)

$$T(n) \leq c\,n \sum_{i=0}^{\log_2 n - 1} \left(\frac{3}{2}\right)^i$$

Pongo $r = \frac{3}{2}$

$$T(n) \leq c\,n \left(\frac{r^{\log_2 n} - 1}{r-1}\right) \leq c\,n \left(\frac{r^{\log_2 n}}{r-1}\right) = \left(\frac{c}{r-1}\right) n r^{\log_2 n}$$

$$r^{\log_2 n} = n^{\log_2 r} = n^{\log_2 \frac{3}{2}} = n^{(\log_2 3 - 1)}$$

$$T(n) \leq \left(\frac{c}{r-1}\right) n\, n^{(\log_2 3 - 1)} = \left(\frac{c}{r-1}\right) n^{\log_2 3}$$

$$\boldsymbol{T(n) = O(n^{\log_2 3})}$$

Albero di ricorsione per il caso q>2

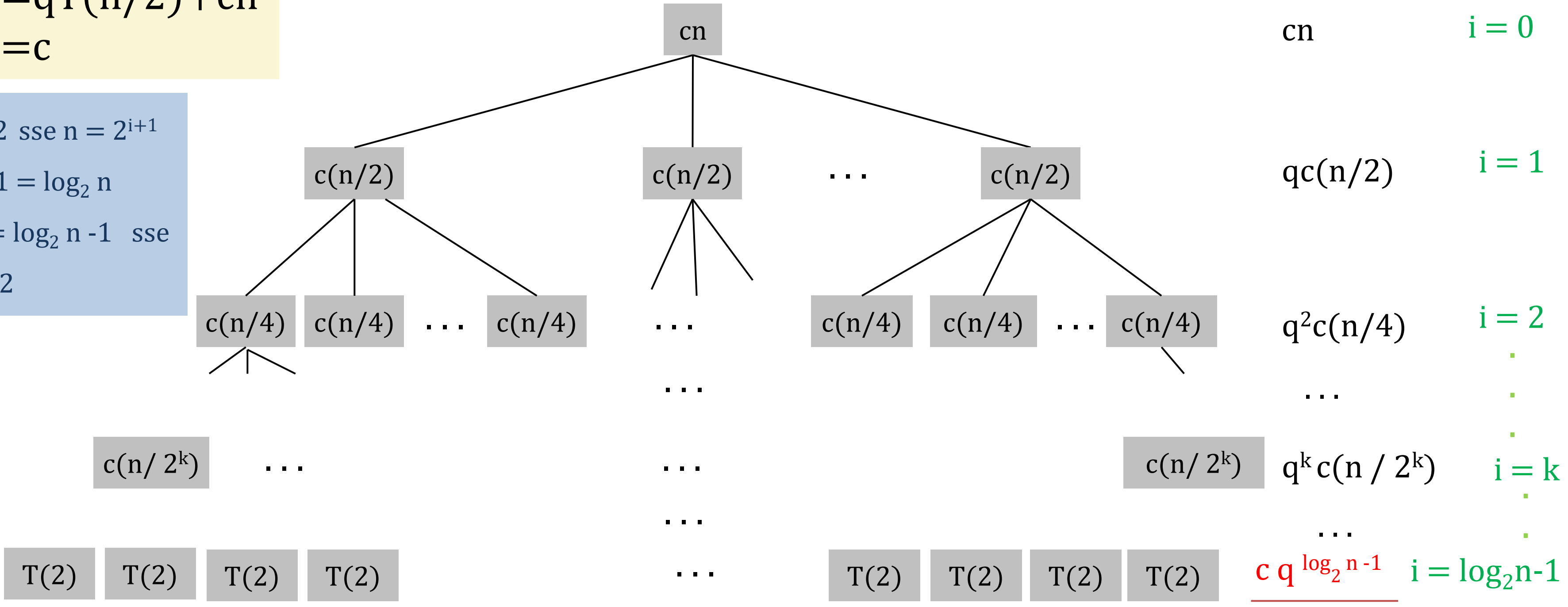
$T(n)=qT(n/2)+cn$
$T(2)=c$

$n/2^i = 2$ sse $n = 2^{i+1}$

sse $i+1 = \log_2 n$

sse $i = \log_2 n - 1$ sse

$2^i = n/2$



$$T(n) = c\,(q^{\log_2 n - 1}) + c\,n\,(1+q/2+ (q/2)^2 + \ldots + (q/2)^{\log_2 n - 2})$$

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

Caso q>2

$$T(n) = c\,(q^{\log_2 n - 1}) + c\,n\,(1+q/2+ (q/2)^2 + \ldots + (q/2)^{\log_2 n - 2})$$

Poiché $c\,(q^{\log_2 n - 1}) \le c\,n\,(q/2)^{\log_2 n - 1}$ (*verifica per esercizio*)

$$T(n) \le c\,n \sum_{i=0}^{\log_2 n - 1} \left(\frac{q}{2}\right)^i$$

Pongo $r = \frac{q}{2}$

$$T(n) \le c\,n \left(\frac{r^{\log_2 n} - 1}{r-1}\right) \le c\,n \left(\frac{r^{\log_2 n}}{r-1}\right) = \left(\frac{c}{r-1}\right) n r^{\log_2 n}$$

$$r^{\log_2 n} = n^{\log_2 r} = n^{\log_2 \frac{q}{2}} = n^{(\log_2 q - 1)}$$

$$T(n) \le \left(\frac{c}{r-1}\right) n\, n^{(\log_2 q - 1)} = \left(\frac{c}{r-1}\right) n^{\log_2 q}$$

$$\boldsymbol{T(n) = O(n^{\log_2 q})}$$

Altre relazioni di ricorrenza

Abbiamo considerato una sotto-famiglia

- $T(n) = qT(n/2) + cn$ con $T(2)=c$

  per q=1 allora $T(n)= \Theta(n)$

  q=2 allora $T(n)= \Theta(n \log_2 n)$

  q>2 allora $T(n)= O(n^{\log_2 q})$

- $T(n)=2T(n/2) +cn^2$

  $T(n)= \Theta(n^2)$

Altri esempi

Sia $T(1) = 1$. Valutate

- $T(n) = 2T(n/2) + n^3$
- $T(n) = T(9n/10) + n$
- $T(n) = 16T(n/4) + n^2$
- $T(n) = 7T(n/3) + n^2$
- $T(n) = 7T(n/2) + n^2$
- $T(n) = 2T(n/3) + \sqrt{n}$
- $T(n) = T(n-1) + n$
- $T(n) = T(\sqrt{n}) + 1$

Tempo di esecuzione 2

Qual è il tempo di esecuzione del seguente frammento di pseudocodice?

```
for i=1 to n/2
   for j=1 to logn
        x=i*j
 return x
```

A. $O(\log n)$

B. $o(n \log n)$

C. $\Theta(n^2)$

D. Nessuna delle risposte precedenti

Un esercizio

Calcolare il tempo di esecuzione del seguente algoritmo

Alg$(A[1...n])$

for $i = 1$ to n do
    $B[i] = A[i]$
for $i = 1$ to n do
    $j = n$
    while $j > i$ do
        $B[i] = B[i] + A[i], j = j - 1$
for $i = 1$ to n do
    $t = t + B[i]$

Esercizio

Determinare la relazione di ricorrenza per il tempo di esecuzione dell’algoritmo ricorsivo per il fattoriale

```
int ric-fact (int n)
      if (n==0) return 1
            else return n * ric-fact(n-1)
```

Esercizi

Scrivere la relazione di ricorrenza soddisfatta dal tempo di esecuzione degli algoritmi nelle prossime slides.

Si noti che non ci interessa cosa calcolino gli algoritmi, ma soltanto quanto tempo impieghino.

Supponiamo che il tempo per eseguire **qualcosa** sia c

Esempi

procedure daffy(n)
  if $n = 1$ or $n = 2$ **then do** qualcosa
    else
      daffy($n - 1$);
      for $i = 1$ **to** n **do**
        qualcosa di nuovo
      daffy($n - 1$)

Esempi

```
procedure elmer(n)
  if n = 1 then do  qualcosa
    else if n = 2 then do  qualcos'altro
    else
      for i = 1 to n do
        elmer(n − 1)
        fa qualcosa di differente
```

Esempi

procedure bar(n)
  if $n = 1$ **then do** qualcosa
    else
      for $i = 1$ **to** *n-1* **do**
        bar(i)
        fa qualcosa di differente

Esercizio: ricerca ternaria

- Progettare un algoritmo per la ricerca di un elemento key in un array ordinato A[1..n], basato sulla tecnica Divide-et-impera che nella prima fase divide l'array in 3 parti «uguali» (le 3 parti differiranno di al più 1 elemento).
- Scrivere la relazione di ricorrenza per il tempo di esecuzione dell'algoritmo proposto.
- *Risolvere la relazione di ricorrenza.*
- *Confrontare il tempo di esecuzione con quello della ricerca binaria.*