

A dark blue, irregular ink splash or blotch serves as the background for the text. It has a textured, painterly appearance with some lighter blue and white speckles around its edges.

Relazioni di ricorrenza

16 marzo 2022

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

```
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:

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

$$T(8) = ???$$

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

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

$$T(2) = 5$$

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

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

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

$$T(n) = 5 n/2 + 3 n (\log_2 n - 1) \quad \text{per } n=2^k$$

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:

$$T(n) = 2T(n/2) + \Theta(n) \quad \text{con } T(2) = \Theta(1) \quad \text{per } n=2^k$$

MergeSort

$$T(n) = 2 T(n/2) + \Theta(n) \quad \text{con } T(2) = \Theta(1) \quad \text{per } n=2^k$$

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

Scegliendo $c = \max\{c_1, c_2\}$ e $a = \min\{a_1, a_2\}$

$$\left\{ \begin{array}{ll} T(n) \leq 2 T(n/2) + cn & \text{con } T(2) \leq c \\ T(n) \geq 2 T(n/2) + an & \text{con } T(2) \geq a \end{array} \right.$$

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: $T(2) \geq a \geq d \cdot 2 \log_2 2 = 2d$ 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 $n=2^k$ $T(n) \leq 2 T(n/2) + c n$

$$T(2) \leq c$$

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

$$\text{ma } T(n/2) \leq 2T(n/2^2) + c n/2$$

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

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

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

$$\text{ma } T(n/2^2) \leq 2T(n/2^3) + c n/2^2$$

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

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

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

.....

Metodo di iterazione (Unrolling)

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

Andrebbe
provato per
induzione!

proseguendo in questo modo
dopo i iterazioni avremo

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

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

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

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

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

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

$$\leq c n \log_2 n$$

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

$$i+1 = \log_2 n$$

$$i = \log_2 n - 1$$

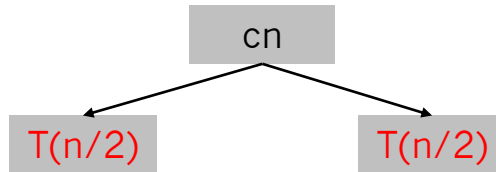
$$2^i = n/2$$

Quindi $T(n) = O(n \log n)$

Albero di ricorsione per MergeSort

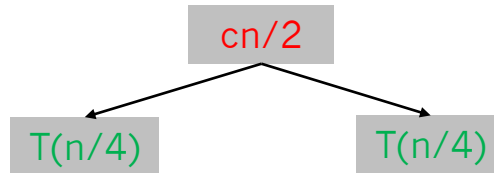
$$T(n) = 2 T(n/2) + cn \quad \text{con } T(2) = c$$

Albero per $T(n)$:

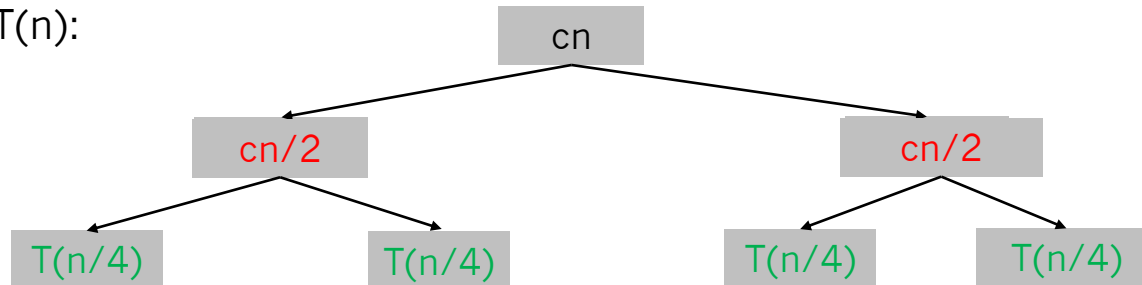


Albero per $T(n/2)$:

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



Albero per $T(n)$:



Albero di ricorsione per MergeSort

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

$$T(2) = c$$

$$i = 0$$

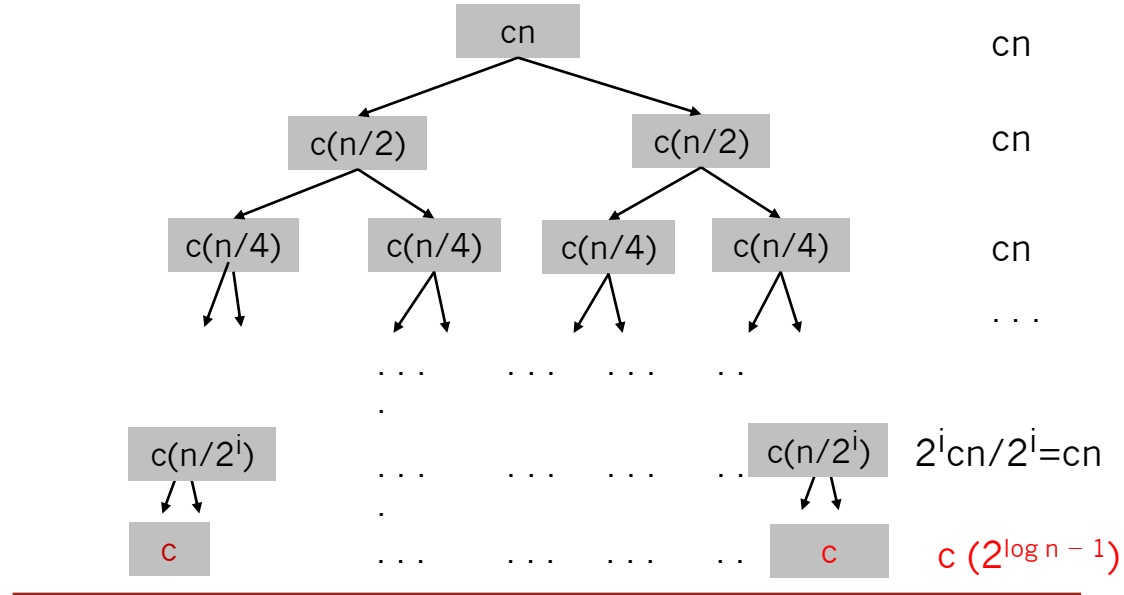
$$i = 1$$

$$i = 2$$

...

$$i = \log_2 n - 2$$

$$i = \log_2 n - 1$$



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

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

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

$$\begin{aligned} n/2^i &= 2 \text{ sse } n = 2^{i+1} \text{ sse } i+1 = \log_2 n \\ \text{sse } i &= \log_2 n - 1 \text{ sse } 2^i = n/2 \end{aligned}$$

Analisi ricorrenza esatta per MergeSort

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

$$x \leq \lceil x \rceil < x+1$$

$$\lfloor x \rfloor \leq x \leq \lceil x \rceil$$

$$x-1 < \lfloor x \rfloor \leq x$$

$$3,45 \leq \lceil 3,45 \rceil = 4 < 4,45$$

$$3 \leq \lceil 3,45 \rceil \leq 4$$

$$2,45 < \lfloor 3,45 \rfloor = 3 \leq 3,45$$

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: $\lfloor \log_2 n \rfloor \leq \log_2 n \leq \lceil \log_2 n \rceil$

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

Analisi ricorrenza esatta per MergeSort

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

$$x \leq \lceil x \rceil < x+1$$

$$x-1 < \lfloor x \rfloor \leq x$$

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

Per n qualsiasi:

$$\begin{aligned} T(n) &\leq T(2^{\lceil \log_2 n \rceil}) \leq c 2^{\lceil \log_2 n \rceil} \lceil \log_2 n \rceil \leq c 2^{(\log_2 n + 1)} (\log_2 n + 1) = \\ &= 2c n (\log_2 n + 1) = 2c n \log_2 n + 2cn \leq 4c n \log_2 n \quad (\text{se } 1 \leq \log_2 n, \text{ cioè } n \geq 2) \end{aligned}$$

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

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

Analisi ricorrenza esatta per MergeSort (induzione)

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

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

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

per ogni $n \geq 2$.

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)

$q>2$ allora $T(n)= ?$

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

$T(n)= ?$

Un esercizio (svolto)

Calcolare il tempo di esecuzione del seguente algoritmo

Alg($A[1\dots n]$)

1. for $i = 1$ to n do
2. $B[i] = A[i]$
3. for $i = 1$ to n do
4. $j = n$
5. while $j > i$ do
6. $B[i] = B[i] + A[i], j = j - 1$
7. for $i = 1$ to n do
8. $t = t + B[i]$

Analisi for (linee 3-6)

```
3.  for  $i = 1$  to  $n$  do
4.       $j = n$ 
5.      while  $j > i$  do
6.           $B[i] = B[i] + A[i], j = j - 1$ 
```

$i = 1$, $j = n, n-1, \dots, 2$, quindi $(n-1)$ volte

$i = 2$, $j = n, n-1, \dots, 3$, quindi $(n-2)$ volte

... ..

$i = n-1$, $j = n$, quindi 1 volta

$i = n$, quindi 0 volte

Primo modo:

Cominciamo dall'analisi del **while** (5-6):

Ripete due istruzioni di tempo costante (6) un numero di volte variabile, ma $\leq n-1$, quindi prende tempo $O(n)$.

Il **for** itera n volte blocco 4-6

Blocco 4-6 prende tempo $O(1)+O(n) = O(n)$.

Quindi il **for** prende tempo $O(n \cdot n) = O(n^2)$.

Analisi for (linee 3-6)

```
3.  for i = 1 to n do
4.      j = n
5.      while j > i do
6.          B[i] = B[i] + A[i], j = j - 1
```

i = 1, j = n, n-1, ... , 2, quindi (n-1) volte

i = 2, j = n, n-1, ... , 3, quindi (n-2) volte

... ..

i = n-1, j = n, quindi 1 volta

i = n, quindi 0 volte

Secondo modo:

Quante volte **in tutto** è ripetuta ciascuna istruzione di tempo costante, all'interno del **for**?

$$\begin{aligned}\text{Numero ripetizioni (6)} &= (n-1) + (n-2) + \dots + 2 + 1 = \\ &= 1 + 2 + \dots + (n-1) = \\ &= (n-1)n/2 = (n^2 - n)/2 = \Theta(n^2).\end{aligned}$$

Analogamente il test $j > i$ è eseguito $n + (n-1) + \dots + 2 + 1 = \Theta(n^2)$ volte.

L'**assegnamento** (4) è eseguito n volte.

In totale, il tempo di esecuzione del **for** è $\Theta(n^2)$.

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
```

Esempi

```
procedure bugs( $n$ )  
  if  $n = 1$  then do qualcosa  
  else  
    bugs( $n - 1$ );  
    bugs( $n - 2$ );  
    for  $i = 1$  to  $n$  do qualcosa
```


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.*