MergeSort
Esercizi

22 marzo 2023

Divide et Impera

– Dividi il problema in sottoproblemi

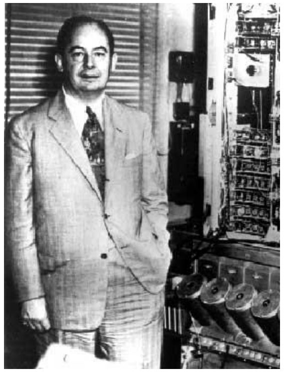
– Risolvi ogni sottoproblema ricorsivamente

– Combina le soluzioni ai sottoproblemi per ottenere la soluzione al problema

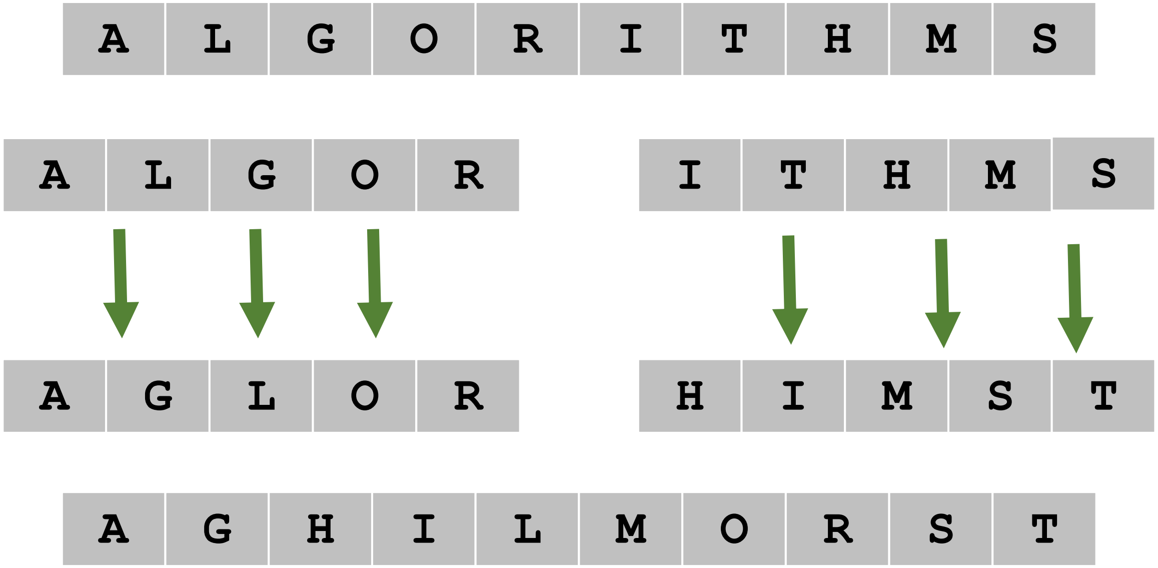
Mergesort

Mergesort.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.



Jon von Neumann (1945)



Recursively sort: as a black box

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)
```

Qui supponiamo che MERGE(A, p, q, r) fonda le sequenze A[p,…, q] e A[q+1, …, r] in loco. Quindi la divisione dell’array in due metà si fa semplicemente calcolando q nella linea 2 (con costo costante). Altrimenti, la divisione avrebbe comportato l’allocazione di due sotto-array e la copia degli elementi (con costo lineare).
Anche MERGE-SORT sarà in loco, non avrà quindi bisogno di memoria ausiliaria.

Correttezza di Mergesort

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

Perché funziona?

Nel caso base (n=1), un array di 1 elemento è già ordinato e l'algoritmo correttamente non esegue nessuna operazione su di esso.

Nel caso generale di una chiamata principale su n elementi, l'algoritmo ordina correttamente la porzione di array perché:

- Possiamo supporre per induzione che le due chiamate di MERGE-SORT su n/2 elementi restituiscano gli array ordinati
- MERGE chiamato su due array ordinati, fonde correttamente i due array ordinati in un solo array ordinato

Tempo di esecuzione di Mergesort

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

Sia T(n) il tempo di esecuzione di Mergesort su un array di taglia n.

T(n) = ?

Come si calcola il tempo di esecuzione di un algoritmo ricorsivo?

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 $\Theta(1)$ 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?

Non 2 Θ(n/2) !

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

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 $n=2^k$

Relazioni di ricorrenza per alcuni algoritmi Divide-et-Impera

Se un algoritmo Divide-et-Impera

- Divide il problema di taglia n in a sotto-problemi di taglia n/b
- Ricorsione sui sottoproblemi
- Combinazione delle soluzioni

allora

$T(n)=$ tempo di esecuzione su input di taglia n

$$T(n) = D(n) + a\,T(n/b) + C(n)$$

Esercizi dalla piattaforma (svolti)

Notazioni asintotiche 1

La relazione $\mathbf{3n(n\text{-}1) \leq c\ n^2}$ è vera per ogni $n \geq n_0$ con

A. $c = 4$ e $n_0 = 0$

B. $c = 2$ e $n_0 = 0$

C. $c = 1/3$ e $n_0 = 1$

D. Nessuna delle risposte precedenti

	A	B	C	D
Numero di risposte	26	4	4	16

Notazioni asintotiche 2

Date due funzioni $f(n)$ e $g(n)$ si ha che $\boldsymbol{f(n) = O(g(n))}$ se

A. $f(n) \leq g(n)$ per n che tende ad infinito

B. Per ogni $c > 0$, $f(n) \leq c\, g(n)$ per ogni $n \geq n_0$

C. Esistono $c > 0$, $n_0 \geq 0$ tali che $f(n) \leq c\, g(n)$ per ogni $n \geq n_0$ D. Nessuna delle risposte precedenti

	A	B	C	D
Numero di risposte	0	0	50	4

Notazioni asintotiche 3

Qual è la corretta successione delle funzioni seguenti affinché compaiano da sinistra a destra in ordine crescente di crescita asintotica: $\mathbf{n^2}$, $\mathbf{n\ (log\ n)^2}$, $\sqrt{\boldsymbol{n}^{\mathbf{3}}}$?

A. $\sqrt{\boldsymbol{n}^{\mathbf{3}}}$, n^2, $n\ (\log n)^2$

B. $n\ (\log n)^2$, $\sqrt{\boldsymbol{n}^{\mathbf{3}}}$, n^2

C. $n\ (\log n)^2$, n^2 , $\sqrt{\boldsymbol{n}^{\mathbf{3}}}$

D. Nessuna delle risposte precedenti.

	A	B	C	D
Numero di risposte	2	37	0	16

Tempo di esecuzione 1

Il tempo di esecuzione del seguente frammento di pseudocodice è

```
for i=1 to n/2
      if i > 0 then
            x=2x
return x
```

A. O(log n)
B. Θ(n log n)
C. Θ (n^2)
D. Nessuna delle risposte precedenti

	A	B	C	D
Numero di risposte	3	1	1	48

Tempo di esecuzione 2

Il tempo di esecuzione del seguente frammento di pseudocodice è

```
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. ϴ (n^2)
D. Nessuna delle risposte precedenti

	A	B	C	D
Numero di risposte	1	9	0	41

Tempo di esecuzione 3

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
    $t = t + B[i]$

A. $\Theta(n^2)$
B. $\omega(n^2)$
C. $\Theta(n)$
D. Nessuna delle risposte precedenti

	A	B	C	D
Numero di risposte	4	1	0	0

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

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 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?

Numero ripetizioni (6)= (n-1) + (n-2) + … + 2 + 1 =
= 1 + 2 + … + (n-1) =
=(n-1) n/2 = $(n^2 - n)/2 = \Theta(n^2)$.

Analogamente il test j > i è eseguito n + (n-1) + … + 2 + 1 = $\Theta(n^2)$ volte.
L'assegnamento (4) è eseguito n volte.
In totale, il tempo di esecuzione del for è $\Theta(n^2)$.

Tempo di esecuzione 4

Il tempo di esecuzione del seguente frammento di pseudocodice è

```
for i=1 to logn
    for j=1 to logn
        x=i*j
return x
```

A. O(log n)
B. O(n log n), ma non Θ (n log n)
C. Θ (n^2)
D. Nessuna delle risposte precedenti

	A	B	C	D
Numero di risposte	4	21	1	23

Merge

Quanti confronti effettua l'algoritmo MERGE per la fusione dei due array ordinati

[1,5,6,7] e [2,3,4]?

A. 4 B. 6 C. 12

D. Nessuna delle precedenti

	A	B	C	D
Numero di risposte	22	0	0	0

Funzioni da ordinare (KT3)

Es. n. 3 pag. 67 del libro [KT].

Indicare la corretta successione delle funzioni seguenti affinché compaiano da sinistra a destra in ordine crescente di crescita asintotica, motivando adeguatamente la successione proposta:

$$f_1(n) = n^{2.5}$$

$$f_2(n) = \sqrt{2n}$$

$$f_3(n) = n + 10$$

$$f_4(n) = 10^n$$

$$f_5(n) = 100^n$$

$$f_6(n) = n^2 \log n$$