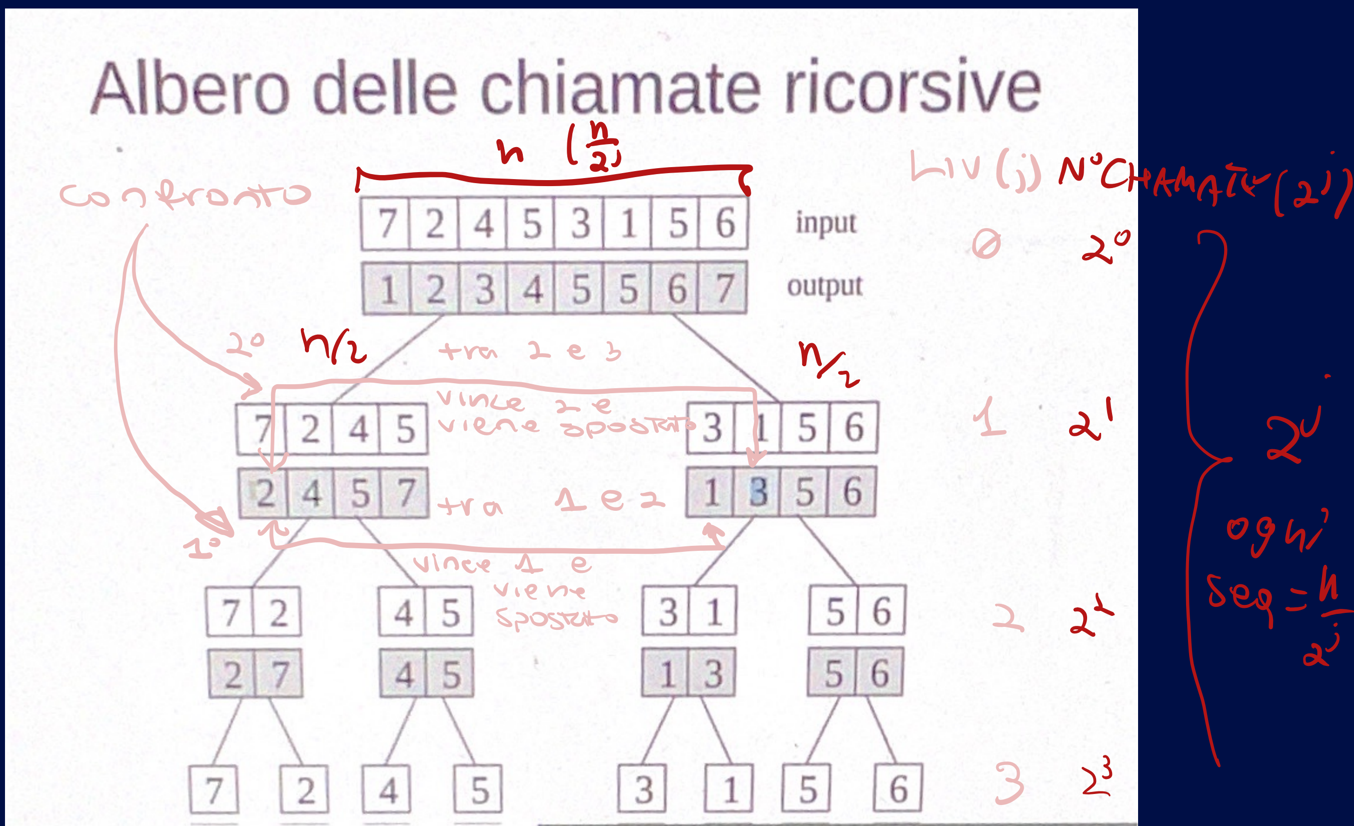
Complessità mergesort

quanto mi costa fondere 2 parti in un unico array?

On sia nel caso migliore che nel peggiore questo xk ogni elem delle sottosequenze è analizzato una volta sola



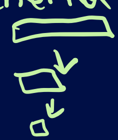
complessità di merge (fonti) On xk non confronto ogni elem di seq di sx con quello di di dx

dove n è num di elem della sequenza ordinata (dopo fusione)

mergesort ≠ RecBin

ad ogni chiamata ne ho $\boxed{2}$ ricorsive ex schema

ogni chiamata ne dava solo 1 ricorsiva ex schema



Livello 1: 2 chiamate ricorsive e poi fusione

Ad 1 livello quante operaz faccio?

LIVELLO		dimensione
0	(1 seq)	n $1 \cdot n = \Theta n$
1	(2 seq)	$n/2$ $\cancel{2} \cdot \frac{n}{\cancel{2}} = \Theta n$
2	(4 seq)	$n/4$ $\cancel{4} \cdot \frac{n}{\cancel{4}} = \Theta n$
3	(8 seq)	$n/8$ $8 \cdot \frac{n}{8} = \Theta n$

complessità – $\Theta n \cdot$ n° LIVELLI

complex ad ogni livello

RIPETO

LIVELLO	n° di merge (chiamate)	costo
0	$1 = 2^0$	Θn
1	$2 = 2^1$	$\Theta(\frac{n}{2})$
2	$4 = 2^2$	$\Theta(\frac{n}{4})$
3		

RELAZIONE

per ogni livello 2^j chiamate merge che daranno come risultato una sottoseq lunga $n/2^j$

quindi complessità $= \Theta\, n/2^j$ (per ogni livello)

quindi $\cancel{2^j} \cdot (\Theta \frac{n}{\cancel{2^j}}) = \Theta n$

Quindi la domanda da porci è: qual è quel livello in cui le sotto sequenze sono lunghe uno?

$\frac{n}{2^j} = 1$? $\iff$ $j = \log_2 n$ $\qquad \Theta \log n$

Complessità tot $\Theta n \cdot \Theta \log_{\cancel{2}} n \cancel{+1} = \Theta(n \cdot \log n)$

caso migliore = caso peggiore

... livello della ricorsione

Costo su ogni livello:

sappiamo che merge ha costo $\Theta(n)$ lineare

Su ogni livello si hanno 2^j sottoproblemi di tipo merge, ognuno lungo $\frac{n}{2^j}$ e quindi risolvibile in $\Theta(\frac{n}{2^j})$

Per sapere il costo su ogni livello bisogna moltiplicare il costo di merge per il numero di volte che viene chiamato:

$$2^j \times \Theta\left(\frac{n}{2^j}\right) = \Theta\left(2^j \times \frac{n}{2^j}\right) = \Theta(n)$$

ogni livello costa $\Theta(n)$

numero dei livelli:

Sapendo che all'ultimo livello della ricorsione, i sottoproblemi assumono dimensione 1, e che su ogni livello i sottoproblemi sono di dimensione $\frac{n}{2^j}$, per quale j si ha che

$\frac{n}{2^j} = 1$? $\qquad n = 2^j \rightarrow \log_2 n = j$

Quindi, n° di livelli = $\boxed{\log_2 n + 1}$ ~ perché si parte dal livello 0

In conclusione, il costo di mergeSort è dato da:

$$\Theta(n) \times \log_2 n + \cancel{1} \rightarrow \Theta(n \log n)$$

... sia nel caso migliore sia nel caso peggiore

Risposta esatta per spiegare ms

all'esame:

- dire se caso migliore = peggiore
- complessità
- operaz per livello
- dire che vengono fatte operazioni di tipo merge
- profondità albero (xk log n?)

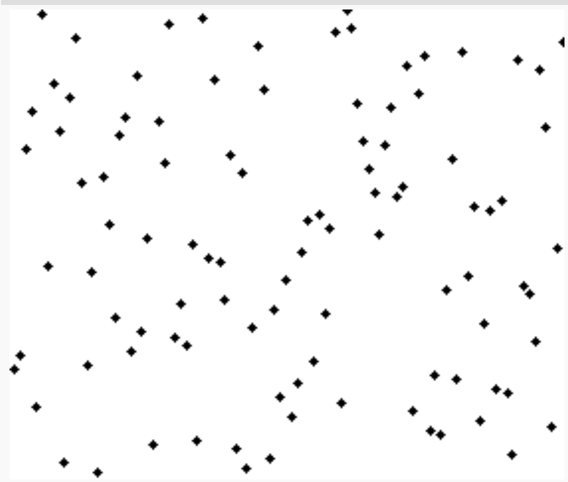
Merge sort

Da Wikipedia, l'enciclopedia libera.

Questa voce o sezione sull'argomento programmazione non cita le fonti necessarie o quelle presenti sono insufficienti.

Il **merge sort** è un algoritmo di ordinamento basato su confronti che utilizza un processo di risoluzione ricorsivo, sfruttando la tecnica del Divide et Impera, che consiste nella suddivisione del problema in sottoproblemi della stessa natura di dimensione via via più piccola. Fu inventato da John von Neumann nel 1945. Una descrizione dettagliata e un'analisi della versione bottom-up dell'algoritmo apparve in un articolo di Goldstine e Neumann già nel 1948.

Merge sort



Esempio di merge sort con una lista di numeri casuali.

Classe	Algoritmo di ordinamento
Struttura dati	Array
Caso peggiore temporalmente	$\Theta(n \log n)$
Caso ottimo temporalmente	$\Theta(n \log n)$
Caso medio temporalmente	$\Theta(n \log n)$
Caso peggiore spazialmente	$\Theta(n)$
Ottimale	In alcuni casi

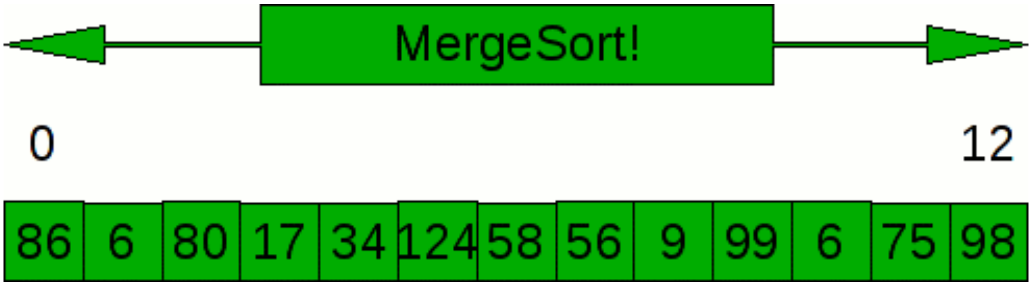
Indice

Descrizione dell'algoritmo

Concettualmente, l'algoritmo funziona nel seguente modo:

1. Se la sequenza da ordinare ha lunghezza 0 oppure 1, è già ordinata. Altrimenti:
2. La sequenza viene divisa (*divide*) in due metà (se la sequenza contiene un numero dispari di elementi, viene divisa in due sottosequenze di cui la prima ha un elemento in più della seconda)
3. Ognuna di queste sottosequenze viene ordinata, applicando ricorsivamente l'algoritmo (*impera*)
4. Le due sottosequenze ordinate vengono fuse (*combina*). Per fare questo, si estrae ripetutamente il minimo delle due sottosequenze e lo si pone nella sequenza in uscita, che risulterà ordinata

Esempio di funzionamento



Simulazione del merge sort in esecuzione su un array

Supponendo di dover ordinare la sequenza [10 3 15 2 1 4 9 0], l'algoritmo procede ricorsivamente dividendola in metà successive, fino ad arrivare agli elementi

[10] [3] [15] [2] [1] [4] [9] [0]

A questo punto si fondono (merge) in maniera ordinata gli elementi, riunendoli in coppie:

[3 10] [2 15] [1 4] [0 9]

Al passo successivo, si fondono le coppie di array di due elementi:

[2 3 10 15] [0 1 4 9]

Infine, fondendo le due sequenze di quattro elementi, si ottiene la sequenza ordinata:

[0 1 2 3 4 9 10 15]

L'esecuzione ricorsiva all'interno del calcolatore non avviene nell'ordine descritto sopra. Tuttavia, si è formulato l'esempio in questo modo per renderlo più comprensibile.

Implementazione

L'algoritmo può essere implementato fondamentalmente tramite due tecniche:

1. **Top-Down**, che è quella presentata in questa pagina. Opera da un insieme A e lo divide in sotto insiemi (A_1, A_2) fino ad arrivare all'insieme contenente un solo elemento, per poi riunire le parti scomposte;
2. **Bottom-Up**, che consiste nel considerare l'insieme A come composto da un vettore di n sequenze. Ad ogni passo vengono fuse due sequenze.

Una possibile implementazione dell'algoritmo in forma di pseudocodice tramite una tecnica top-down è la seguente:

```
function mergesort (a[], left, right)
    if left < right then
        center ← (left + right) / 2
        mergesort(a, left, center)
        mergesort(a, center+1, right)
        merge(a, left, center, right)
```

Una possibile implementazione della funzione merge (unione di due sottosequenze ordinate) è la seguente:

```
function merge (a[], left, center, right)
   i ← left
   j ← center + 1
   k ← 0
   b ← array temp size= right-left+1

   while i ≤ center and j ≤ right do
      if a[i] ≤ a[j] then
         b[k] ← a[i]
         i ← i + 1
      else
         b[k] ← a[j]
         j ← j + 1
      k ← k + 1
   end while

   while i ≤ center do
      b[k] ← a[i]
      i ← i + 1
      k ← k + 1
   end while

   while j ≤ right do
      b[k] ← a[j]
      j ← j + 1
      k ← k + 1
   end while

   for k ← left to right do
      a[k] ← b[k-left]
```

Analisi

L'algoritmo Merge Sort, per ordinare una sequenza di n oggetti, ha complessità temporale $T(n) = \Theta(n \log n)$ sia nel caso medio che nel caso pessimo. Infatti:

- la funzione merge qui presentata ha complessità temporale $\Theta(n)$
- mergesort richiama se stessa due volte, e ogni volta su (circa) metà della sequenza in input

Da questo segue che il tempo di esecuzione dell'algoritmo è dato dalla ricorrenza:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

la cui soluzione in forma chiusa è $\Theta(n \log n)$, per il secondo caso del teorema principale.

Raffigurazione grafica delle versioni iterativa (bottom-up) e ricorsiva (top-down) dell'algoritmo

Esistono implementazioni più efficienti della procedura merge, che hanno nel caso migliore complessità $O(1)$. Infatti, se i due array da fondere sono già ordinati, è sufficiente confrontare l'ultimo elemento del primo array con il primo elemento del secondo array per sapere che si può fonderli senza effettuare ulteriori confronti. Per cui si può implementare l'algoritmo mergesort in modo che abbia complessità O(nlogn) nel caso peggiore, e O(n) nel caso migliore, cioè quando l'array è già ordinato.

Bibliografia

- Thomas Cormen, *Introduction to Algorithms*, 3ª ed..

Altri progetti

- Wikibooks contiene implementazioni di **merge sort**
- Wikimedia Commons (https://commons.wikimedia.org/wiki/?uselang=it) contiene immagini o altri file su **merge sort (https://commons.wikimedia.org/wiki/Category:Merge_sort?uselang=it)**