ORDINAMENTO DI UN GRANDE NUMERO DI OGGETTI

irreversibilità sospesa

L’entropia era già molta ma ... il risultato finale sarà irreversibile a dispetto di qualsivoglia

ALGORITMO DI ORDINAMENTO

SELECTION SORT **INSERTION SORT** **MERGE SORT** **QUICK SORT**

QUICKSORT: ORDINAMENTO VELOCE

Uno degli algoritmi di ordinamento più diffusi è il Quicksort.

L'algoritmo Quicksort lavora partizionando l'array da ordinare, e quindi **ordinandone ricorsivamente ogni partizione.**

Uno degli elementi dell'array viene scelto come valore di **pivot** (di perno). I valori inferiori al pivot vengono messi alla sua sinistra, mentre i valori superiori vengono messi alla sua destra.

Se siamo fortunati il pivot scelto corrisponde alla mediana di tutti i valori, cioè divide l'array in parti uguali. Per un momento assumiamo di trovarci in questo caso.

Siccome l'array viene diviso in due ad ogni passo e devono comunque essere esaminati tutti gli n elementi, **il tempo di esecuzione risulta $O(n \lg n)$**

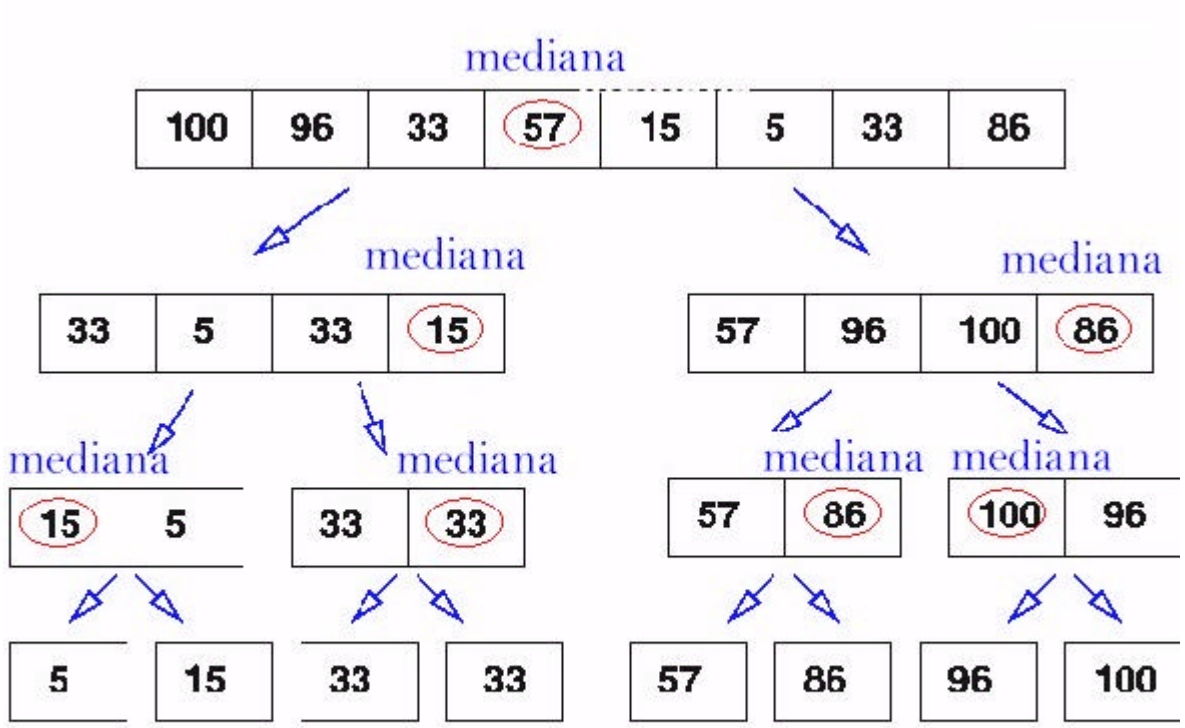
Supponiamo di avere una pila di libri da mettere in ordine alfabetico per autore.

Possiamo iniziare scegliendo uno dei libri (ad esempio un autore che inizia con la "L" o con la "M") e dividendo i libri in due pile: una con tutti i libri che precedono in ordine alfabetico l'autore scelto, ed una con tutti i libri che seguono (il libro scelto come "elemento separatore" può appartenere ad una qualsiasi delle due pile).

A questo punto possiamo ripetere la procedura su ciascuna delle due pile ottenute.

Continuando a dividere rimarremo con gruppetti di due o tre libri, ed infine con libri singoli, che a questo punto saranno in ordine alfabetico.

Ecco un altro esempio con i numeri (i numeri non sono tutti diversi) che ci aiuta a capire come questa strategia possa essere convertita in un algoritmo:



Quicksort (nel caso migliore ed ottimale!)

Osservando bene si nota che il caso nella figura è assolutamente ottimale! Ogni volta siamo riusciti a scegliere un elemento "separatore" che divide la collezione esattamente in due.

Comunque, c'è un caso che fallisce miserabilmente.

Supponiamo che l'array fosse originalmente ordinato. Se si decide di scegliere come pivot il primo elemento della sequenza, e cioè l'elemento minore, si avrebbe che l'array ha un elemento nella partizione di sinistra, e gli altri elementi nell'altra, perché sarebbero tutti maggiori del pivot.

Ogni chiamata ricorsiva di quicksort ridurrebbe solo di un'unità la dimensione dell'array da ordinare. Sarebbero quindi necessarie n chiamate ricorsive per effettuare l'ordinamento, portando a un tempo di esecuzione di $\boldsymbol{O}(n^2)$.

Una soluzione a questo problema si può ottenere scegliendo a caso il pivot tra gli elementi della sequenza.

La probabilità di selezionare proprio la mediana e' ad ogni passo uguale ad 1/n, quindi e' inversamente proporzionale alla lunghezza dell'array, ma la bontà del QS risiede proprio nell'avere un comportamento ottimo anche nei casi sfavorevoli (ma non troppo).

L'unico punto debole di QS risiede nella degenerazione del caso peggiore.

QS preferisce sequenze disordinate e da luogo a comportamenti " poco intelligenti" nel caso di sequenze già ordinate.

Scegliendo a caso il pivot, anche se la sequenza è già ordinata, e se gli elementi sono sufficientemente numerosi, risulta altamente improbabile che il pivot scelto sia il primo.

Può sembrare sorprendente che uno dei sistemi migliori per effettuare tale scelta sia quello di ***affidarsi al caso.***

SELECTION SORT

Ordinamento per selezione, la strategia:

finché l'array non è ordinato

1. seleziona l'elemento di valore minimo dell'array tra quelli che non sono stati ancora ordinati

2. disponi l'elemento di valore minimo nella sua posizione definitiva.

Nell'ordinamento per selezione inizialmente tutti gli elementi sono non ordinati. Dopo la prima *passata* il primo elemento viene ordinato con n-1 confronti. Per ordinare l'array si devono eseguire tante *passate* fino a quando tutti gli elementi degli array non risultano ordinati.
Ad ognuna di tali *passate* il numero dei passi aumenta di k-1, dove k è il numero degli oggetti rimasti, si trova infatti il più grande (o il più piccolo) degli elementi ancora da ordinare e in questo modo si ordinano dal più grande al più piccolo. Anche se ce ne sono due uguali, la risposta al confronto con un elemento uguale è in ogni caso FALSO, dato che A<B e anche A>B è falso, ma questo non è comunque importante perché l'elemento considerato per secondo, tra i due uguali, risulterà il maggiore nella passata successiva.

La strategia di Selection Sort è la più ingenua

Naturalmente è lo stesso se invece di determinare il minimo ad ogni *passata,* viene determinato il massimo dell'insieme.

Il comportamento dell'ordinamento per selezione è indipendente dall'ordine preesistente nell'array da ordinare, il caso migliore e il caso peggiore coincidono,

Per n-1 volte bisogna comunque determinare il minimo tra gli elementi non ordinati ed effettuare uno scambio.

Per ottenere il numero totale dei confronti, nel caso di Selection Sort, il calcolo da fare è **n – 1 + (n –1) – 1 + (n – 2) – 1 + (n – 3) – 1 + ... + 1 = n – 1 + n – 2 + n – 3 + n – 4 + ... + 1 = = n * (n – 1) – (n – 1) * n/2 = n(n – 1)/2**

In tutti gli algoritmi di ordinamento esaminati, l'operazione dominante è il confronto tra coppie di elementi. Ciascuna esecuzione dell'operazione dominante, il confronto, ha costo unitario.

L'ordinamento per selezione ha complessità quadratica rispetto alla lunghezza dell'array da ordinare (complessità asintotica $O(n^2)$)

MERGE SORT: ORDINAMENTO PER FUSIONE

Si procede così :

1. Gli elementi del vettore vengono divisi in due sotto-sequenze. Quindi è necessario individuare una strategia per "partizionare" gli elementi del vettore: gli elementi vengono decomposti in due sottosequenze contigue. Questa scelta rende possibile una decomposizione ricorsiva di sequenze contigue di un array in sotto-sequenze contigue.

2. Ciascuna sotto-sequenza viene ordinata separatamente. Quindi è necessario implementare un algoritmo di ordinamento, ed è abbastanza ovvio che utilizzeremo ancora l'algoritmo per fusione, il Merge Sort. **L'ordinamento per fusione è un algoritmo ricorsivo**

3. Le due sottosequenze ordinate vengono **fuse in un'unica sequenza ordinata.** quindi è necessario avere a disposizione un algoritmo di fusione che risolva il seguente problema: date due sequenze S1 e S2 ordinate in modo non decrescente, costruire una nuova sequenza S che contiene tutti gli elementi di S1 e S2 ed è ordinata in modo non decrescente.

4. Basterà 1) selezionare gli elementi di S uno alla volta, in sequenza e poi il prossimo elemento di S verrà scelto tra l'elemento più piccolo di S1 e quello più piccolo di S2 tra quelli che non sono stati ancora inseriti in S. Si procederà confrontando tra loro i primi due delle due successioni, uno dei due verrà inserito nella sequenza S, successivamente si confronterà quello non inserito, con il secondo della sequenza che ha ceduto un elemento ad S, e così via.

L'ordinamento potrà iniziare perché le successive partizioni degli elementi porteranno a sequenze di due soli elementi che potranno facilmente essere confrontati e ordinati, e successivamente verrà fatta la fusione e così di seguito.

Per esempio consideriamo A, B, C, D, E.

AB CDE

CDE verrà diviso in

CD E

C e D sono confrontati in un passaggio

Successivamente la coppia CD viene fusa con E, poi la terna CDE viene fusa con AB, in tutto 8 passi basati sul confronto (un passaggio in più rispetto al metodo precedentemente trovato)

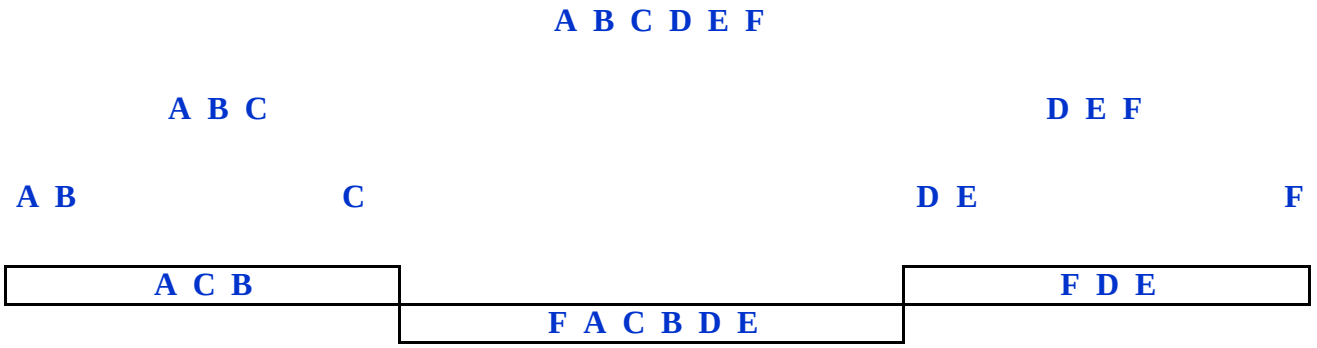
Con 8 passi si ordinano i 5 elementi.

Nella prima parte è stato adoperato, senza una piena consapevolezza, questo metodo.

Si noti quindi che con questo metodo non si ottiene l'ottimizzazione dei confronti per pochi oggetti, il metodo che con tanta testardaggine si è cercato per i cinque oggetti è in questo caso "snobbato"

.

Per capire meglio proviamo ancora con 6 elementi. Dovremo fare 11 passaggi.



1 passaggio per ordinare gli insiemi di due elementi, 2 passaggi per fondere il terzo elemento con la coppia ordinata. Successivamente la fusione tra le due terne ordinate. (5 passi,n-1 passi).

L'operazione dominante dell'ordinamento per fusione è la fusione di sottosequenze ordinate.

Il costo di questo algoritmo di ordinamento dipende quindi da quante volte viene eseguita la fusione e da quanto costa ciascuna operazione di fusione

COMPLESSITÀ DELL'ORDINAMENTO PER FUSIONE

Analisi del tempo di esecuzione dell' ordinamento per fusione

Sia M(n) il numero di confronti, nel caso peggiore, che MergeSort opera con un input di dimensione n.

Definiamo, ora una funzione T(n) per il tempo di esecuzione, tale che **M(n) <= T(n)** Consideriamo, consistentemente con la definizione di MergeSort, due casi:

T(n) = 0, se n<2 questo e' il caso base

Il caso induttivo deve stabilire che il numero dei confronti fatti per ordinare n elementi e' al piu' uguale alla somma del numero di confronti fatti nel caso peggiore per ciascuno dei tre passi descritti sopra, nel caso n>1.

1. Ordina la parte sinistra dell'array

2. Ordina la parte destra dell'array

3. Unisci (merge) le due parti in un array ordinato

Quindi:

$T(n) = T(n/2) + T(n/2) + n$, se $n>1$

Consideriamo ciascun termine.

Il primo termine fissa un limite superiore al numero di confronti necessari ad ordinare la parte sinistra dell'array, che consiste della meta' degli elementi dell'intero array.

Il secondo termine fissa un limite superiore al numero di confronti necessari ad ordinare la parte destra dell'array, che consiste della meta' degli elementi dell'intero array.

L'ultimo termine, n, fissa un limite superiore ai confronti necessari a comporre i due array (più precisamente i confronti necessari sono n-1)

Esempio, sia n=8, come calcoliamo T(8)?

Dobbiamo scomporre la relazione fintanto che non arriviamo al passo base:

$T(8) = 2T(4) + 8$

$T(4) = 2T(2) + 4$

$T(2) = 2T(1) + 2$

$T(1) = 0$

A questo punto possiamo fare le opportune sostituzioni come segue:

$T(1) = 0$

$T(2) = 2T(1) + 2 = 0 + 2$

$T(4) = 2T(2) + 4 = 4 + 4 = 8$

$T(8) = 2T(4) + 8 = 16 + 8 = 24$

In questo caso, mergeSort richiede al piu' 24 confronti per ordinare 8 elementi.

Da questa osservazione possiamo calcolare che M(n)= n log n.

Quindi possiamo porre, analogamente, che n log n = M(n) <= T(n) .

L'operazione dominante nell'ordinamento per fusione è la fusione di sottosequenze ordinate.

Quante volte viene esguita questa operazione e per quale motivo la fusione ha un costo minore o uguale a n?

Un esempio numerico

Partizioni

74 20 50 45 15 30 80 47

74 20 50 45 | 15 30 80 47

74 20 | 50 45 | 15 30 | 80 47

Fusioni

20 74 | 45 50 | 15 30 | 47 80

20 45 50 74 | 15 30 47 80

15 20 30 45 47 50 74 80

L'ordinamento per fusione ha quindi complessità asintotica O(n log n) rispetto alla lunghezza dell'array da ordinare.

L'ordinamento per fusione è asintoticamente più efficiente degli altri algoritmi di ordinamento.

È stato dimostrato che non esistono algoritmi di ordinamento di array basati su confronti che abbiano complessità asintotica migliore di n log n.

INSERTION SORT

Supponiamo di avere una sequenza di carte numerate e di voler costruire su di un tavolo una sequenza ordinata crescente da sinistra a destra estraendo una carta alla volta dal mazzo.

Otterremo un procedimento del tipo:

> **"prendi la carta in cima al mazzo ed inseriscila al suo posto sul tavolo fino ad esaurimento delle carte"**

Questo significa quanto segue.

1. Gli elementi sono divisi in due insiemi: un insieme di elementi "ordinati", memorizzati ad esempio nelle posizioni più a sinistra dell'array (sono elementi ordinati tra di loro, ma che non è detto siano stati necessariamente collocati nelle loro posizioni definitive,), un insieme di elementi "non ordinati" memorizzati nelle posizioni più a destra dell'array.

2. Ad ogni passata il primo tra gli elementi non ordinati viene collocato tra gli elementi "ordinati", inserendolo nella posizione corretta, corretta relativamente agli elementi già "ordinati"

Passiamo ad un'analisi di massima delle prestazione dell' Ordinamento per inserzione.

L'operazione dominante per l'algoritmo **IS** è il confronto tra coppie di elementi.

L'ordinamento per inserimento diretto si comporta in modo naturale poiché inserisce gli elementi lavorando meno se essi sono gia' parzialmente ordinati. Infatti basta eseguire il primo confronto con la penultima carta estratta.

Se sono ordinati nell'ordine giusto allora la carta estratta sarà messa al posto giusto con un solo confronto. Il caso peggiore si ottiene invece nel caso in cui la sequenza di elementi si presenta in

ordine inverso, perché in questo caso il numero di confronti sarà pari al numero di carte già ordinate.

L'AO durante la prima passata esegue un confronto con l'unica carta che è posta sul tavolo, o con la carta più a sinistra nell'array.

Durante la seconda passata esegue due confronti, con le due carte che ci sono sul tavolo, oppure con le due carte più a sinistra dell'array.

Durante la n-1 esima passata vengono eseguiti al più n-1 confronti

Si può concludere che Insertion Sort ha complessità asintotica $O(n^2)$ rispetto alla lunghezza dell'array da ordinare infatti: $1 + 2 + 3 + ... + n-1 = n(n-1)/2$