Ordinamento (Sorting)

(da: wikipedia Algoritmo_di_ordinamento)

Gli **algoritmi di ordinamento** sono di vario tipo; di seguito alcune caratterizzazioni:

- ordinamento **interno**: dati memorizzati in memoria centrale
  (accesso diretto ai dati ➔ si ottimizza il numero di passi);
- ordinamento **esterno**: dati memorizzati su memoria di massa
  (tempi di attesa più lunghi ➔ si ottimizza il numero di accessi).
- ordinamento **in loco** (= ***in place)***: usa oltre al vettore da ordinare una quantità di memoria ausiliaria limitata che non dipende dal numero di dati da ordinare. Non crea una copia dell'input per raggiungere l'obiettivo, pertanto un algoritmo ***in place*** risparmia memoria rispetto ad un algoritmo non in place.
- ordinamento **stabile**: un metodo di ordinamento si dice stabile se preserva l'ordine relativo dei dati con chiavi uguali all'interno del file da ordinare (ad esempio se si ordina per anno di corso una lista di studenti già ordinata alfabeticamente un metodo stabile produce una lista in cui gli alunni dello stesso anno sono ancora in ordine alfabetico mentre un ordinamento instabile probabilmente produrrà una lista senza più alcuna traccia del precedente ordinamento)

Vi sono **varie classi di algoritmi di ordinamento**, i più noti ed utilizzati sono gli algoritmi di **ordinamento per confronto** (***comparison sort algorithms***), ma esistono altre classi caratterizzate da un tempo di esecuzione che, nel caso peggiore, hanno **complessità** inferiore a $O(n \cdot \log n)$.

Classificazione (per complessità) di alcuni algoritmi di Ordinamento

- *O* (**n^2**) : **quadratici**; semplici, iterativi, basati sul confronto;
  es. ***insertion sort, selection sort, bubble s*ort**;
- O (**$n \cdot \log_2 n$**): **linearitmici**; ottimi ma più complessi perché **ricorsivi**;
  es. ***quick sort****, merge sort, heap sort*;
- O (**n**) : **lineari**; basati sul calcolo, ma con ipotesi restrittive sui dati, cioè possono essere usati solo con determinati dati; es. *counting sort*.

Di seguito alcuni algoritmi di ordinamento, con relativa complessità al caso Migliore, Medio e Peggiore, in ordine di efficienza crescente.

Algoritmo	Caso migliore	Caso medio	Caso peggiore
SELECTIONSORT	n^2	–	n^2
INSERTIONSORT	n	–	n^2
BUBBLESORT	n	–	n^2
QUICKSORT	$n \log_2 n$	$n \log_2 n$	n^2
MERGESORT	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
HEAPSORT	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
COUNTINGSORT	n	n	n
BUCKETSORT	–	–	n

Confronto della complessità computazionale degli algoritmi esaminati

prof.ssa P.Grandillo

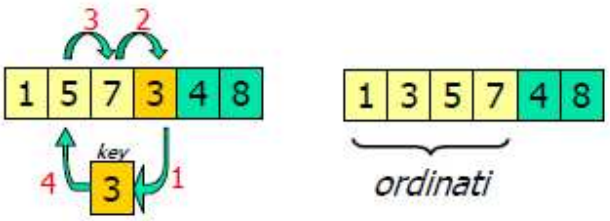
Descrizione di alcuni algoritmi

INSERTION SORT (da: wikipedia Insertion_sort)

L'**ordinamento a inserimento**, è un algoritmo relativamente semplice per ordinare un array (non è molto diverso dal modo in cui ordiniamo un mazzo di carte, un elenco di nomi, etc.); è un algoritmo ***in place***, cioè ordina l'array senza doverne creare una copia, risparmiando memoria. Pur essendo molto meno efficiente di algoritmi più avanzati, può avere alcuni vantaggi: ad esempio, è semplice da implementare ed è **efficiente per insiemi di partenza che sono quasi ordinati**.

Per ciascuno degli elementi a partire dal secondo (l'elemento da sistemare ad ogni iterazione si chiama *chiave=key*):

- si copia il valore della chiave in una variabile di appoggio *key* (nell'esempio il valore 3);
- **si scalano a destra** di una posizione tutti gli elementi precedenti alla chiave finché non ne viene trovato uno con valore inferiore o uguale a *key* (indichiamolo con MIN);



- nella posizione successiva a quella di MIN (nell'esempio, MIN ha valore 1) viene messo *key*;

Notare che gli elementi a sinistra della chiave sono sempre già ordinati in senso crescente grazie alle passate precedenti.

Nella iterazione successiva si pone in *key* il valore 4 e si procede a confrontarlo con i precedenti finchè viene trovato 3, nuovo valore di MIN. Si procede così fino all'ultimo elemento presente nell'array.

In C++ queste azioni vanno ripetute per **i = *1***, 2, … , ***n-1***

SELECTION SORT (da: wikipedia Selection_sort)

L'**ordinamento per selezione** è un algoritmo di ordinamento che opera ***in place*** ed in modo simile all'ordinamento per inserzione. L'algoritmo è di tipo *non adattivo*, ossia il suo tempo di esecuzione non dipende dall'input ma dalla dimensione dell'array.

L'algoritmo considera la sequenza suddivisa in due parti: la sottosequenza ordinata, che occupa le prime posizioni dell'array, e la sottosequenza da ordinare, che costituisce la parte restante dell'array.

L'algoritmo **seleziona di volta in volta il numero minore** nella sottosequenza da ordinare e lo sposta nella sequenza ordinata. Dovendo ordinare un array A di lunghezza n, in C++ si fa scorrere l'indice i da **0** a ***n-2*** ripetendo i seguenti passi:

1. si cerca il più piccolo elemento della sottosequenza A[**i+**1, .. , **n-1**];
2. si **scambia** questo elemento con l'elemento **i**-esimo.

Il primo passo è quello di individuare il **minimo** tra gli elementi del vettore e scambiarlo con quello nella **prima posizione**, il primo valore è ora al posto giusto. Se si ripete lo stesso identico procedimento per tutti gli elementi a partire dal secondo, si determina il secondo valore più piccolo e lo si colloca al secondo posto, etc.

(da: http://staff.polito.it/claudio.fornaro/CorsoC/12-Algoritmi.pdf)

EXCHANGE SORT (SELECTION SORT semplificato)

Si confronta il primo elemento del vettore con tutti gli altri, man mano che si trova un valore minore del primo, lo si scambia con il primo.

Dopo questa operazione il valore minimo è nella prima posizione (pos. 0) del vettore.

Se si ripete lo stesso identico procedimento per tutti gli elementi a partire dal secondo, si determina il secondo valore più piccolo e lo si colloca al secondo posto.

Se si ripete lo stesso identico procedimento per tutti gli elementi a partire dall'**i**-esimo, si determina l'**i**-esimo valore più piccolo e lo si colloca all'**i**-esimo posto.

Se si ripete questo procedimento per tutti i valori di i da 0 fino al penultimo (l'ultimo va a posto da sé) si ottiene l'ordinamento in senso crescente di tutto il vettore. In C++ si fa scorrere l'indice i da **0** a ***n-2***

BUBBLE SORT

Se si scorrono tutti gli elementi di un vettore e ogni volta che si trovano **due valori ADIACENTI** non in ordine (il più piccolo dei 2 a destra del più grande) li si scambia, il più grande di tutti risale a destra.

Ripetendo N-1 volte questa operazione, tutti i valori risalgono verso destra fino ad occupare la posizione corretta e quindi vengono ordinati in **senso crescente**.

Inefficienza: gli ultimi valori vengono in ogni caso confrontati, anche quando sono già stati collocati; per evitare perdita di tempo in questi controlli inutili si introduce un'ottimizzazione, fermando prima della fine il ciclo interno, sfruttando il ciclo esterno.

```
for (i=0 ; i<N-1; i++)
   for (j=0; j<N-1; j++)
      if (A[j] > A[j+1])
         scambia (A[j], A[j+1]);
```

```
for (i=N-1; i>0 ; i--)
   for (j=0; j<i ; j++)
      if (A[j] > A[j+1])
         scambia (A[j], A[j+1]);
```

Inefficienza: se pochi valori sono fuori posto e l'ordinamento si ottiene prima delle N-1 passate, i cicli continuano ad essere eseguiti. Si ottimizza l'algoritmo facendolo terminare se non ci sono stati scambi.

Bubble sort (ottimizzato)

```
char scambi = SI;              //#define SI 'S'
for (i= N-1; i>0 && scambi == SI; i--)
{ scambi = NO;                 //#define NO 'N'
  for (j=0; j<i ; j++)
     if (A[j] > A[j+1])
     { scambi = SI;
       tmp    = A[j];
       A[j]   = A[j+1];
       A[j+1] = tmp; }         // SCAMBIO completato
}
```

Insertion sort

```
for (j=1; j<N; j++)
{
   key = A[j];
   for (i=j-1; i>=0 && A[i]>key; i--)
      A[i+1] = A[i]; //avanzamento elementi
   A[i+1] = key;      //inserimento di key
}

//--- si parte dal secondo elemento
```

Selection sort semplificato (Exchange sort)

```
for (i=0; i<N-1; i++)
  for (j=i+1; j<N; j++)
     if (A[j] < A[i])
     {  tmp  = A[j];
        A[j] = A[i];
        A[i] = tmp;          // SCAMBIO completato
}

//-ad ogni iterazione di i,il num.scambi è >=0
```

Selection sort

```
for (i=0; i<N-1; i++)
{ jmin = i;
  for (j=i+1; j<N; j++)
     if (A[j] < A[jmin]) //aggiornam. della
        jmin = j;        //posizione di minimo
  tmp     = A[jmin];
  A[jmin] = A[i];
  A[i]    = tmp;         // SCAMBIO completato
} //-ad ogni iterazione 0 o 1 solo scambio
```