Algoritmi di ordinamento di un array

Problema: Data una sequenza di elementi in ordine qualsiasi, ordinarla.
Questo è un problema fondamentale, che si presenta in moltissimi contesti, ed in diverse forme:

- ordinamento degli elementi di un array in memoria centrale
- ordinamento di una collezione in memoria centrale
- ordinamento di informazioni memorizzate in un file su memoria di massa (file ad accesso casuale su disco, file ad accesso sequenziale su disco)

Noi consideriamo solo la variante base del problema dell'ordinamento.
Problema: Dato un **vettore V** di N elementi in **memoria centrale** non ordinato, ordinarne gli elementi in **ordine crescente**.

Sono molti gli algoritmi di ordinamento che possono essere utilizzati.
Noi ne analizziamo tre: **ORDINAMENTO PER SOSTITUZIONE**, **ORDINAMENTO PER SELEZIONE**, **ORDINAMENTO A BOLLE**.

Sono estremamente elementari e consentono un'implementazione assai semplice; il costo da pagare alla semplicità di questi algoritmi sta ovviamente nell'elevata ***complessità computazionale*** in termini del *numero di confronti* e *di scambi*, che in questi casi l'ordine è $O(N^2)$. Ve ne sono molti altri via via sempre più sofisticati (e dunque di complessità computazionale sempre più bassa in alcuni casi dell'ordine $O(N\ log_2 N)$ che risolvono il problema in modo più efficiente.

➢ Ordinamento per sostituzione

È il più semplice in assoluto, il primo metodo che viene in mente. Si confronta l'elemento nella prima posizione con gli elementi nelle posizioni successive effettuando uno scambio se, fra i due elementi confrontati, il primo è maggiore del secondo. Alla fine dei confronti l'elemento nella prima posizione risulta sicuramente minore di tutti gli elementi nelle posizioni successive. Successivamente si ripete il procedimento per l'elemento nella seconda posizione, nella terza e così via fino all'ultimo elemento.
Sia **N** la dimensione dell'array. Per inserire il valore minimo nella prima posizione si devono controllare **N-1** elementi effettuando nel caso peggiore **N-1** scambi. Per il secondo elemento si controllano **N-2** elementi, per il terzo **N-3** …., per cui, alla fine, si hanno:

n° di confronti : sempre	**N· (N – 1)/2**	[$(N-1)+(N-2)+\cdots+2+1 = N\cdot(N-1)/2$]
n° di scambi : (caso migliore)	**0**	(caso peggiore) **N· (N – 1)/2**

Il numero totale di confronti rimane inalterato anche in caso in cui il vettore all'inizio sia già ordinato o parzialmente ordinato. Ad esempio, anche nel caso in cui l'insieme da ordinare sia già completamente ordinato, l'algoritmo eseguirà tutte le iterazioni dei due cicli nidificati e quindi tutti i confronti, con il solo vantaggio di non scambiare mai tra loro gli elementi presi in esame.

Implementazione dell'algoritmo:

```
void Ord_sostituzione (int  V [ ],  int N)    // ordinamento Crescente
 {int temp;
  for (int i=0; i<N-1;  i++)          //  i = indice elemento perno
     for (int j=i+1;  j<N;  j++)      //  j = indice elementi successivi confrontati con elemento perno
         {if ( V[j] < V[i])           //  se l'elemento confrontato è minore SCAMBIA
             { temp=V[j];
               V[j]=V[i];
               V[i]=temp;
             }
         }
  return;
 }
```

➢ Ordinamento per selezione o Selection sort

Un algoritmo decisamente intuitivo ed estremamente semplice. Nella pratica è utile quando l'insieme da ordinare è composto da pochi elementi e dunque anche un algoritmo non molto efficiente può essere utilizzato con il vantaggio di non rendere troppo sofisticata la codifica del programma che lo implementa. L'idea di fondo su cui si basa l'algoritmo è quella di ripetere per N volte una procedura in grado di selezionare alla i-esima iterazione l'elemento più piccolo nell'insieme e di scambiarlo con l'elemento che in quel momento occupa la posizione i che fa da perno.
In altre parole:

- alla prima iterazione verrà selezionato l'elemento più piccolo dell'intero insieme $\{v_1,v_2,v_3,...,v_N\}$ e sarà scambiato con quello che occupa la prima posizione;
- alla seconda iterazione è selezionato il secondo elemento più piccolo dell'insieme, ossia l'elemento più piccolo dell'insieme "ridotto"costituito dagli elementi $\{v_2,v_3,...,v_N\}$ ed è scambiato con l'elemento che occupa la seconda posizione;
- si ripete fino ad aver collocato nella posizione corretta tutti gli N elementi.

n° di confronti : sempre	**N· (N – 1)/2**	[$(N-1)+(N-2)+\cdots+2+1 = N\cdot(N-1)/2$]
n° di scambi : (caso migliore)	**0**	(caso peggiore)	**N – 1**

Anche in questo caso il numero totale di confronti rimane sempre lo stesso anche se il vettore all'inizio è già ordinato o parzialmente ordinato. Il numero invece di scambi è al massimo N **– 1** .
L'ordinamento per selezione ha un'importante applicazione: poiché ciascun elemento viene spostato al più una volta, questo tipo di ordinamento è il metodo da preferire quando si devono ordinare file costituiti da record estremamente grandi e da chiavi molto piccole. Per queste applicazioni il costo dello spostamento dei dati è prevalente sul costo dei confronti e nessun algoritmo è in grado di ordinare un file con *spostamenti* (scambi) di dati sostanzialmente inferiori a quelli dell'ordinamento per selezione.

Implementazione dell'algoritmo:

```
void SelectionSort (int  V [ ],  int N)      // ordinamento Crescente
 {int temp;
  for (int i=0; i<N-1;  i++)          // i = indice elemento perno
     { // cerca il più piccolo elemento  dalla posizione  i  alla posizione  N-1
       int jmin = i ;                 // jmin  indice dell'elemento minore
       for (int j=i+1;  j<N;  j++)
           {if (V[j] < V[jmin] )     // se  l'elemento minore è in posizione j,  assegna  j a jmin
                 jmin = j ;
           }
       if (i != jmin)                 // SCAMBIA   se  i  è diverso da jmin
          { temp=V[jmin];
            V[jmin]=V[i];
            V[i]=temp;
          }
     }
  Return;
 }
```

➢ Ordinamento a bolle o Bubble Sort

L'algoritmo BUBBLE SORT si basa sull'idea di far emergere pian piano gli elementi più piccoli verso l'inizio dell'insieme da ordinare facendo sprofondare gli elementi maggiori verso il fondo: un po' come le bollicine in un bicchiere di acqua gassata → da qui il nome di ordinamento a bolle.
La strategia adottata è quella di scorrere più volte la sequenza da ordinare, verificando ad ogni passo l'ordinamento reciproco degli elementi contigui, v_i e v_{i+1} ed eventualmente scambiando di posizione di quelle coppie di elementi non ordinati.
Procedendo dall'inizio alla fine della sequenza al termine di ogni scansione si è ottenuto che l'elemento massimo è finito in fondo alla sequenza stessa, mentre gli elementi più piccoli hanno cominciato a spostarsi verso l'inizio della sequenza stessa.
Dunque:

- *alla fine della prima scansione* possiamo essere certi che ***l'elemento massimo*** *ha raggiunto la sua posizione corretta* nell'ultima posizione della sequenza;
- la scansione successiva *potrà fermarsi senza considerare l'ultimo elemento dell'insieme* perché è già stato sistemato nella scansione precedente e *riuscendo così a collocare nella posizione corretta* (la penultima) *il secondo elemento più grande dell'insieme*;
- si ripete fino ad aver completato l'ordinamento dell'intera sequenza.

Nella versione ottimale si controlla se durante una scansione degli elementi del vettore avviene o meno uno scambio; **se non si verificano scambi il vettore risulta ordinato e l'algoritmo termina**.

n° di confronti	: (caso migliore)	**N – 1**	(caso peggiore)	**N· (N – 1)/2**
n° di scambi	: (caso migliore)	**0**	(caso peggiore)	**N· (N – 1)/2**

Il *numero di confronti* nel caso peggiore (vettore ordinato in senso decrescente) è analogo a quello degli algoritmi di sostituzione e di selezione [$(N-1)+(N-2)+\cdots+2+1=N\cdot(N-1)/2$]. Nel caso migliore (vettore già ordinato il senso crescente) l'algorimo esegue nella prima scansione N – 1 confronti *senza eseguire alcuno scambio*, se ne accorge e non prosegue inutilmente.

Video: http://www.youtube.com/watch?v=gWkvvsJHbwY&feature=player_detailpage

Implementazione dell'algoritmo:

```
void  BubbleSort (int  V[ ], int N)      // ordinamento Crescente
{ bool  Scambio=true;          // Indica se durante una scansione si è verificato almeno uno scambio
  int  Temp,  Ultimo=N – 1 ;   // posizione dell'elemento  limite di ogni scansione
  while (Scambio==true)       // L'algoritmo continua fintantoché c'è stato almeno uno scambio durante
      {                       //  l'ultima scansione
       // inizio nuova scansione
        Scambio=false;
        for (int i=0; i<Ultimo; i++)
            { if (V[i]>V[i+1])
                 { // scambia  e imposta la variabile booleana a   true
                   Temp=V[i];
                   V[i]=V[i+1];
                   V[i+1]=Temp;
                   Scambio=true;
                 }
            }
       Ultimo – –;              //  Alla successiva scansione ci si può fermare alla posizione precedente
      }
  return;
}
```