Semplici Algoritmi di Ordinamento

Ordinamento di una sequenza di elementi

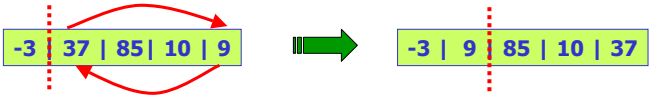
- Esistono molti **algoritmi di ordinamento**. Tutti ricevono in input una sequenza non ordinata di elementi e restituiscono la sequenza ordinata.
- Algoritmi di ordinamento:
  - selection sort,
  - quick sort,
  - bubble sort,
  - merge sort.
- Ognuno di questi algoritmi usa un metodo diverso per ordinare una sequenza di elementi.
- Tutti generano lo stesso risultato (sequenza ordinata), ma alcuni sono più efficienti di altri.

Selection sort

Ordinamento per selezione (selection sort**)**

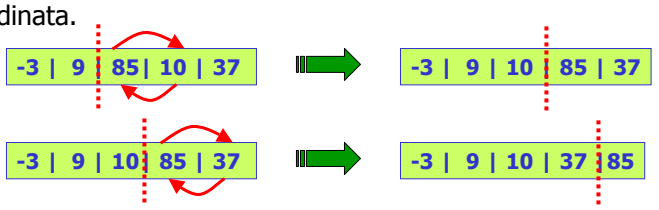
- Questo algoritmo ordina una sequenza di elementi
  - andando a trovare l'elemento minore e portandolo nella posizione iniziale della sequenza, e l'elemento in posizione iniziale nella posizione occupata del valore minore.

  - Quindi sulla sotto-sequenza non ordinata effettua la stessa operazione fino a che rimane un solo elemento (che è ordinato).



Selection sort

- L'algoritmo opera su una sequenza non ordinata come se fosse composta di due sotto-sequenze:
  - la prima **ordinata** e la seconda **non-ordinata**,
  - andando a cercare il valore minimo nella **sequenza non-ordinata** e portandolo nella ultima posizione della sequenza ordinata.



  - Quando la sotto-sequenza non ordinata è composta da un solo elemento l'ordinamento è terminato (l'ultimo elemento è il maggiore).

Selection Sort in Java

```
public void ordinaSel(int[] vet)
{
 for (int j = 0; j < vet.length-1; j++)
 {
    int temp;
    int pos_min = j;
    for (int i = j+1; i < vet.length; i++)
       if (vet[pos_min] > vet[i])
          pos_min = i;
    if (pos_min != j)
    {
      temp = vet[j];
      vet[j] = vet[pos_min];
      vet[pos_min] = temp;
    }
 }   // chiude il for
}
```


Selection Sort in Java

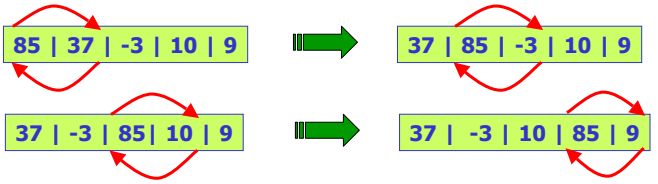
- Si usano due indici **i** e **j**: **j** scorre su tutto l'array, mentre **i** scorre sulla parte dell'array non ordinata.
- All'inizio si assume che **la posizione dell'elemento minore è 0** e dalla posizione 1 fino alla fine si cerca il valore minimo. Se questo è più piccolo dell'elemento nella posizione 0 viene scambiato.
- Quindi si incrementa l'indice **j** (che identifica la prima posizione della parte non ordinata) e si esegue nuovamente la ricerca del minimo nella sotto-sequenza rimanente.
- Alla fine l'elemento che rimarrà nell'ultima posizione dell'array (`v[v.length-1]`) è il valore maggiore.

Bubble sort

Ordinamento per scambio (bubble sort**)**

- Questo algoritmo ordina una sequenza di elementi
  - ☆ andando a confrontare gli elementi a coppie e scambiandoli di posto se il secondo è minore del primo.

1ª scansione

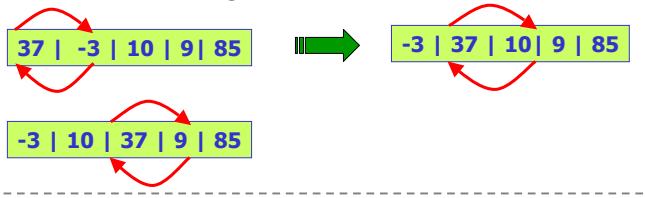


- 🕐 L'algoritmo termina quando dopo aver scandito tutta la sequenza senza che non sia stato effettuato alcuno scambio. In questo caso la sequenza risulta già ordinata.

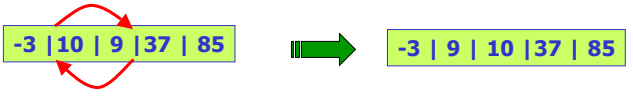
Bubble sort

- Dopo la prima scanzione abbiamo effettuato 4 scambi ma non abbiamo ottenuto la sequenza ordinata. Quindi si riparte dall'inizio a scambiare gli elementi:

2ª scansione



3ª scansione



- Se dopo una scansione (in questo caso la **4ª scanzione**) non sono stati effettuati scambi, gli elementi sono già ordinati e l'ordinamento è completato.

Bubble Sort in Java

```
public void ordinaBub(int[] v)
{ boolean scambio ; int j= v.length-1;
 do
  {
    scambio = false;
    for (int i=0; i < j ; i++)
    {
      int temp;
      if (v[i] > v[i+1])
      {
         temp = v[i];
         v[i] = v[i+1];
         v[i+1] = temp;
         scambio = true;
      }
    }   // chiude il for
    j = j-1;
  }
 while (scambio == true);
}
```


Bubble Sort e Selection Sort in Java

- Gli algoritmi di ordinamento selection sort e bubble sort sono algoritmi abbastanza semplici, ma non sono i più efficienti.
- Il quick sort ed il merge sort sono più complessi ma più efficienti perché effettuano un numero minore di scansioni degli elementi di una sequenza.
- Altri algoritmi di ordinamento :
  - insertion sort
  - heap sort
  - shaker Sort
  - . . . .