Algoritmi di Ricerca e di Ordinamento

Ricerca in una sequenza di elementi

- Data una sequenza di elementi, occorre verificare se un elemento fa parte della sequenza oppure l'elemento non è presente nella sequenza stessa.

- In generale una sequenza di elementi si può realizzare come un **`array`**. E la scansione avviene usando un indice.

- Se la sequenza non è ordinata a priori occorre eseguire una **ricerca lineare** o **sequenziale**.

- Se la sequenza è ordinata è opportuno eseguire una **ricerca binaria**.

Ricerca lineare

- L'algoritmo di ricerca lineare (o sequenziale) in una sequenza (**`array`**) è basato sulla seguente strategia:
  - Gli elementi dell'array vengono analizzati in sequenza, confrontandoli con l'elemento da ricercare (chiave) per determinare se almeno uno degli elementi è uguale alla chiave.
  - Quando si trova un elemento uguale alla chiave la ricerca termina.

- La ricerca è sequenziale, nel senso che gli elementi dell'**`array`** vengono scanditi uno dopo l'altro sequenzialmente.

- L'algoritmo prevede che al più tutti gli elementi dell'array vengano confrontati con la chiave**.** Se l'elemento viene trovato prima di raggiungere la fine della sequenza non sarà necessario proseguire la ricerca.

Ricerca lineare in Java

```
/* Questo metodo implementa la ricerca lineare
restituendo l'indice di un elemento di seq uguale
all'elemento cercato (chiave) oppure -1 che indica
che l'elemento non è presente nella sequenza.*/

public static int ricLineare(int[] seq, int chiave)
{
   int i;          // indice per la scansione di seq
   int ind_elem; // indice di un elemento uguale a
                   // alla chiave
   ind_elem = -1;

   for(i=0; ind_elem==-1 && i<seq.length; i++)
   {
    if (seq[i]==chiave)
     ind_elem = i;
   }
   return ind_elem;
}
```

Uso della ricerca lineare per contare le occorrenze

- L'algoritmo di ricerca lineare può essere usato per verificare quante volte un elemento è presente in una sequenza:
  1. Gli elementi dell'array vengono analizzati in sequenza, confrontandoli con l'elemento da ricercare (chiave) per determinare se uno degli elementi è uguale alla chiave.

  2. Quando si trova un elemento uguale alla chiave **si incrementa un contatore** e il passo 1 viene rieseguito fino alla fine della sequenza.

Calcolo delle occorrenze

```
/* Questo metodo conta quante volte l'elemento
cercato (chiave) è presente in seq e ritorna il
valore delle occorrenze (il valore 0 indica che
l'elemento non è presente nella sequenza).*/

public static int Conta(int[] seq, int chiave)
{
   int i;          // indice per la scansione di seq
   int occorre;    // valore delle occorrenze della
                   // chiave in seq
   occorre = 0;

   for(i=0; i<seq.length; i++)
   {
    if (seq[i] == chiave)
     occorre++;
   }
   return occorre;
}
```

Ricerca binaria

- L'algoritmo di ricerca lineare richiede che al più tutti gli elementi dell'array vengano confrontati con la chiave. Questo è necessario perché la sequenza non è ordinata.

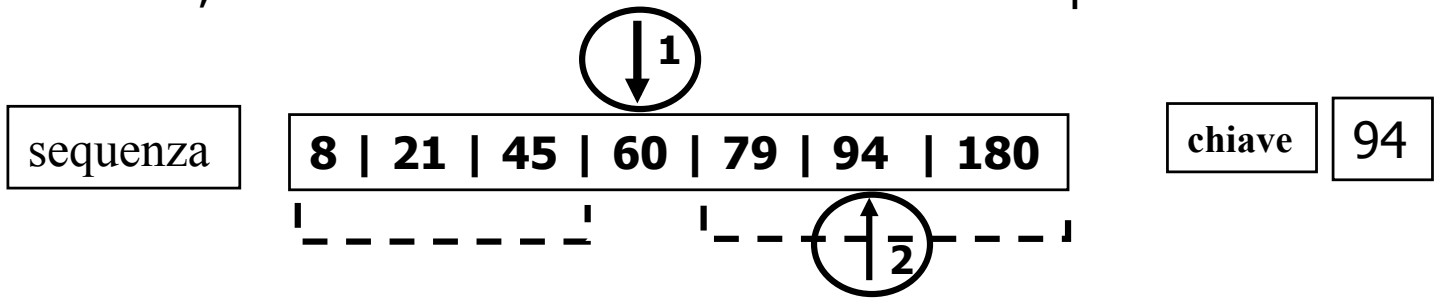
- Se la sequenza su cui occorre effettuare la ricerca è ordinata si può usare un algoritmo di ricerca molto più efficiente che cerca la chiave sfruttando il fatto che gli elementi della sequenza sono già disposti in un dato ordine.

- Esempi di sequenze ordinate: elenco telefonico, agenda, etc.

- In questi casi si usa un algoritmo di **ricerca binaria** che è più efficiente perché riduce lo spazio di ricerca.

Ricerca binaria

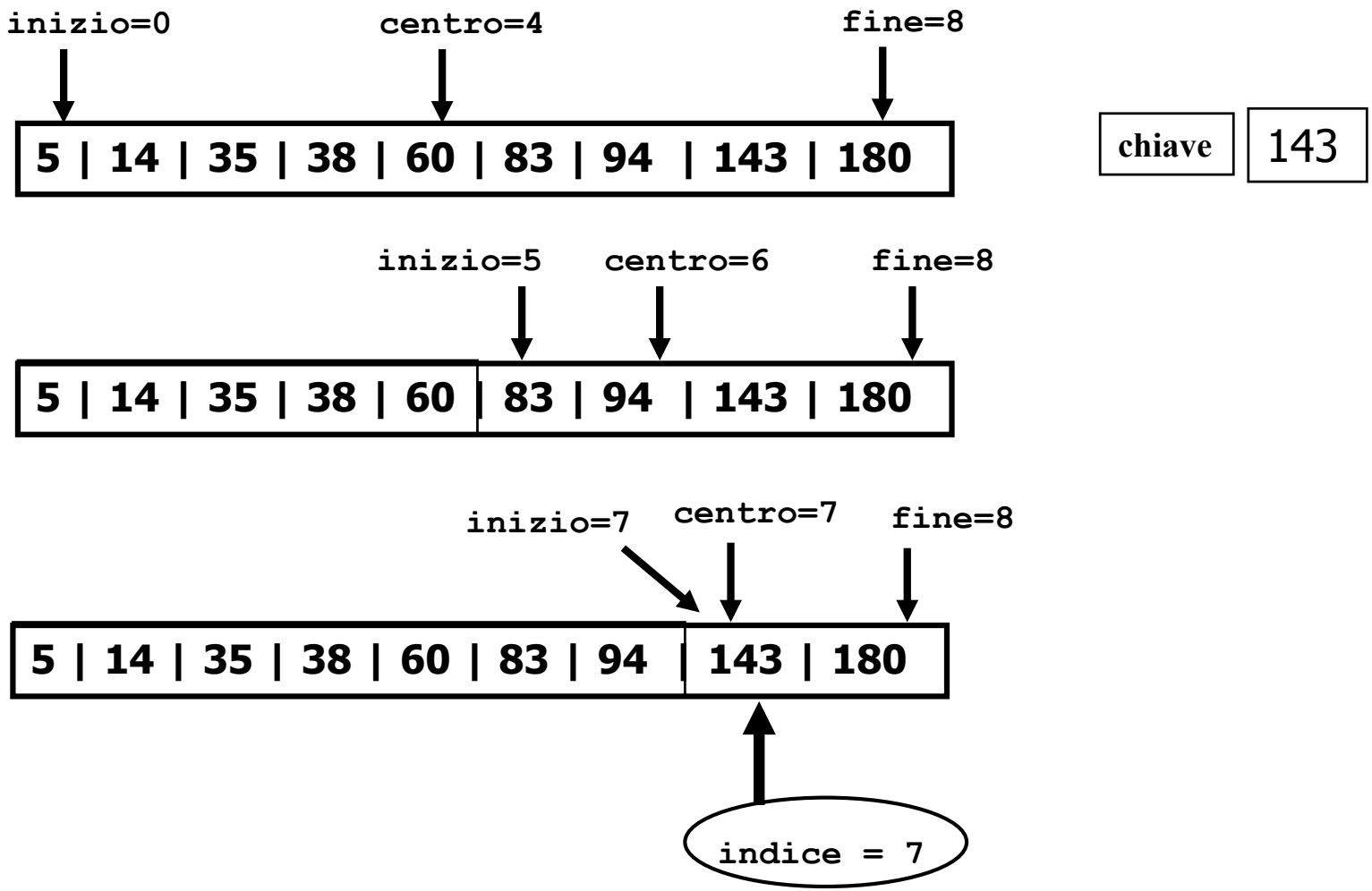
- L'algoritmo di ricerca binaria cerca un elemento in una sequenza ordinata in maniera crescente (o non decrescente) eseguendo i passi seguenti finché l'elemento viene trovato o si è si è completata la ricerca senza trovarlo:
  1. Confronta la chiave con l'elemento centrale della sequenza,
  2. Se la chiave è uguale all'elemento centrale, allora la ricerca termina positivamente,
  3. Se invece la chiave è maggiore dell'elemento centrale si effettua la ricerca solo sulla sottosequenza a destra,
  4. Se invece la chiave è minore dell'elemento centrale dello spazio di ricerca, si effettua la ricerca solo sulla sottosequenza a sinistra.



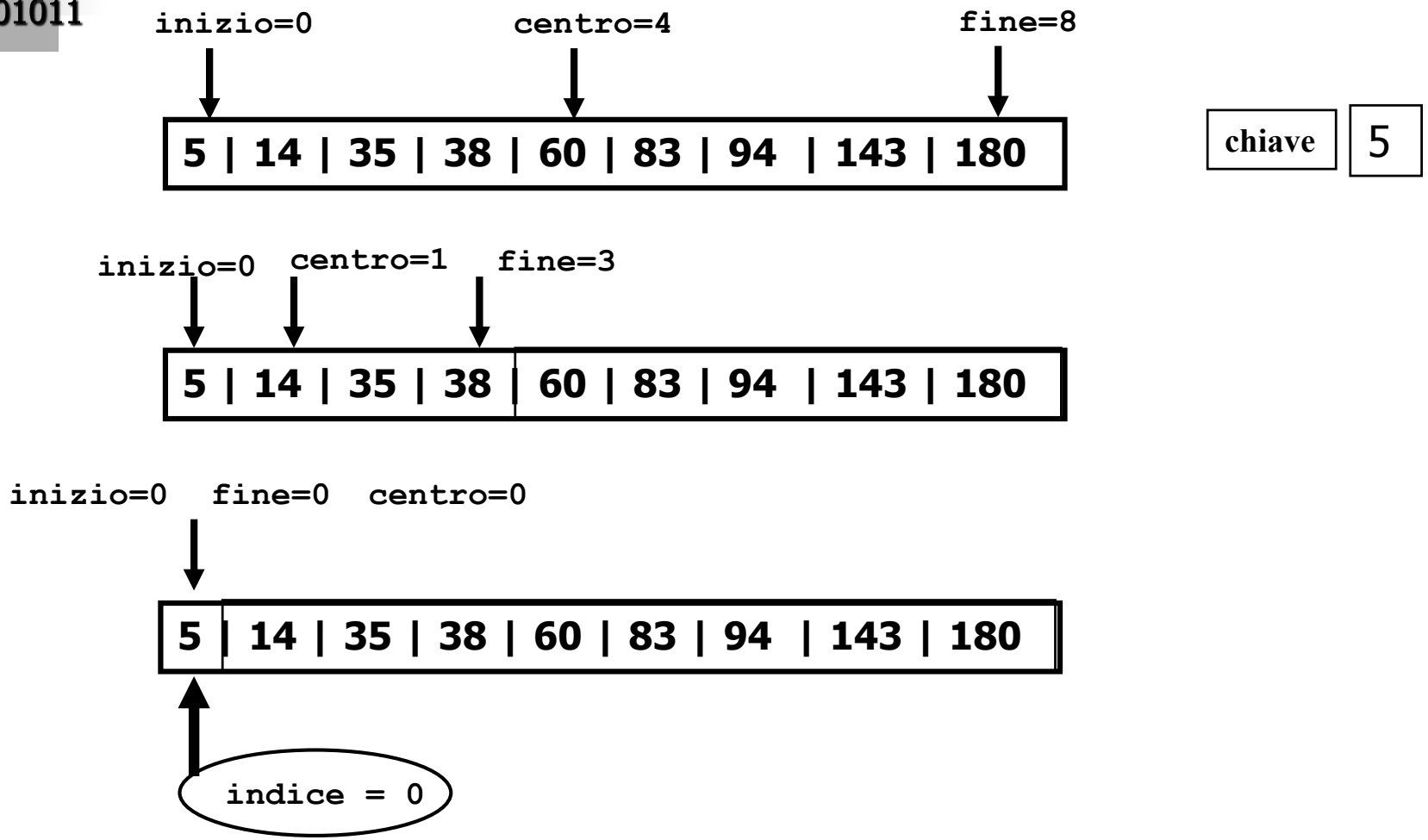
Ricerca binaria in Java

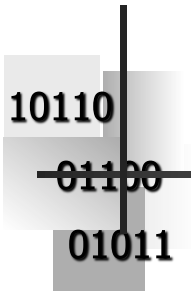
```
/* Questo metodo implementa la ricerca binaria restituendo
l'indice di un elemento di seq uguale all'elemento cercato
(chiave) o -1 che indica che l'elemento non è presente*/
public static int ricBinaria(int[] seq, int chiave)
{  // seq è ordinato in modo non decrescente
 int indice; // indice di un elemento uguale a chiave
 int inizio; int fine; int centro;
 inizio = 0;
 fine = seq.length-1;
 indice = -1;
 while (indice ==-1 && inizio <= fine)
 {
  centro = (inizio+fine)/2;
  if (seq[centro]==chiave)       // trovato
    indice = centro;
  else
   if (chiave > seq[centro])  // continua a destra
      inizio = centro + 1;
   else                       // continua a sinistra
      fine = centro - 1;
 }
 return indice;
}
```

Ricerca binaria



Ricerca binaria





Ricerca binaria

- Inizialmente la ricerca è fatta su **N** elementi dove **N** indica la lunghezza della sequenza (lo spazio di ricerca ha dimensione N).
- Ad ogni iterazione lo spazio della ricerca si riduce di "circa" la metà. Potremmo dire che si passa da N ad N/2 e cosi via.
- Il caso peggiore si ha quando l'elemento cercato non si trova nella sequenza (non esiste un elemento uguale alla chiave).
- Nel caso peggiore, l'iterazione viene eseguita $\mathbf{\log_2 N}$ volte.

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.

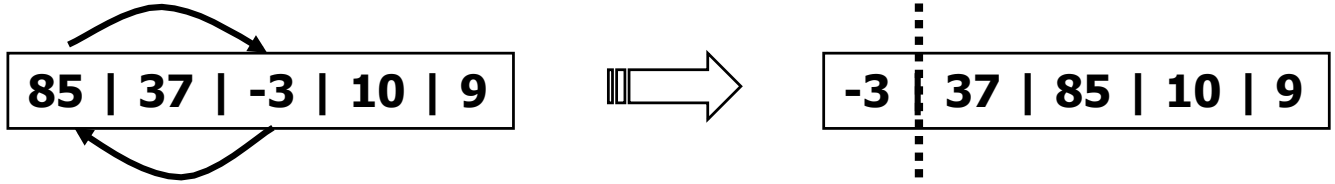
- Ognuno di questi algoritmi usa un metodo diverso per ordinare una sequenza di elementi.

- Tutti generano lo stesso risultato (sequenza ordinata), ma alcuni sono più efficienti di altri.

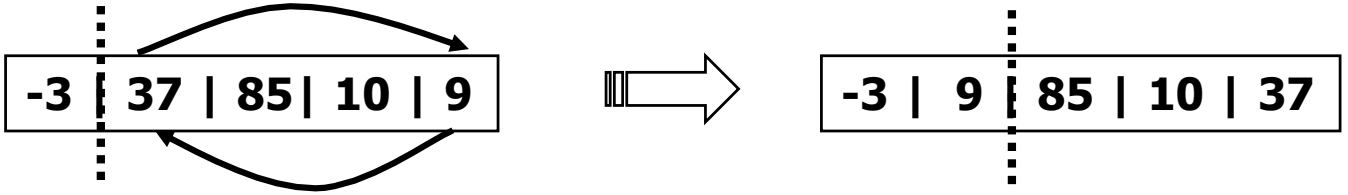
Selection sort

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.

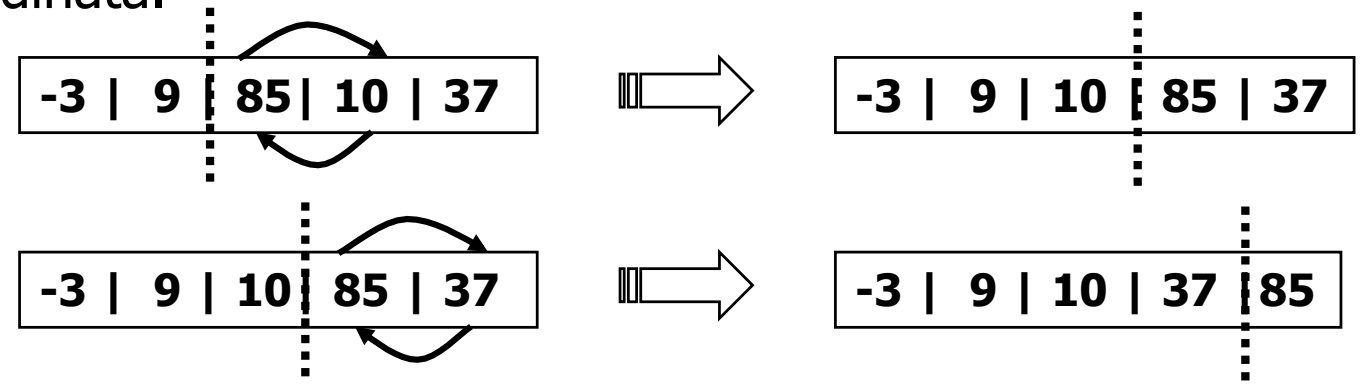


  - ❤ Quindi sulla sotto-sequenza non ordinata effettua la stessa operazione fino a che rimane un solo elemento (che è ordinato).



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

```
// selection sort su un vettore di interi 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.

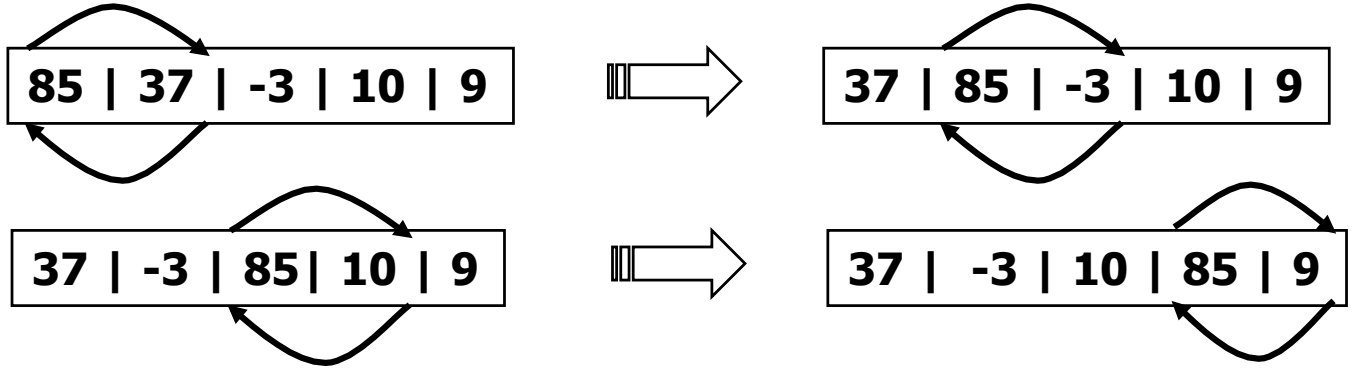
- Alla fine l'elemento che rimarrà nell'ultima posizione dell'array (`v[v.length-1]`) è il valore maggiore.

Bubble sort

Ordinamento per scambio (bubble sort)

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

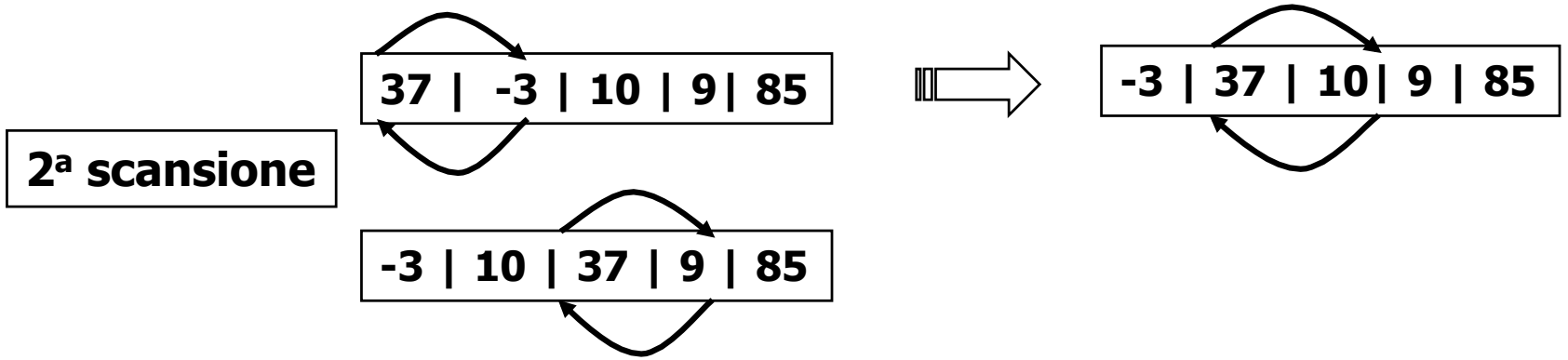
1ª scansione



  - ❷ 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:

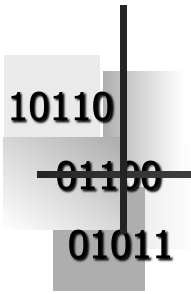


- Se dopo una scansione (in questo caso la **4ª scanzione**) non sono stati effettuati scambi, gli elementi sono già ordinati e l'ordinamento è completato.

Bubble Sort in Java

```
// selection sort su un vettore di interi 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
  - . . . .