Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa 06

Algoritmi di ordinamento

C. Limongelli

Febbraio 2008

Contenuti

- Ordinamento di un array
- Ordinamento di array e API di Java
- Ordinamento per selezione
  - complessità dell'ordinamento per selezione
- Ordinamento a bolle
  - complessità dell'ordinamento a bolle
- Ordinamento per inserzione
  - complessità dell'ordinamento per inserzione
- Ordinamento per fusione
  - complessità dell'ordinamento per fusione
- Ordinamento veloce
  - complessità dell'ordinamento veloce
- Risultati sperimentali

Proprietà di ordinamento di una sequenza

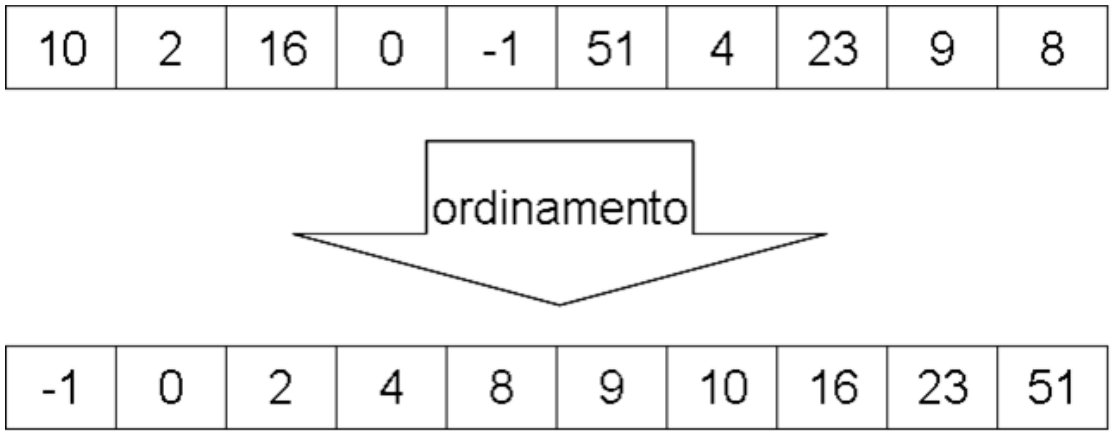
- Sia data una sequenza $<a_1, a_2, \ldots, a_n>$ di n elementi appartenenti ad un insieme dotato di una relazione d'ordine (ad esempio < oppure <=)
- La sequenza si dice totalmente ordinata rispetto alla relazione d'ordine (ad esempio in modo non decrescente) se per ogni i = 1,…,n $a_i <= a_j$ per ogni j >= i
- Altrimenti la sequenza si dice parzialmente ordinata
  - La proprietà $a_i <= a_j$ vale solo per qualche coppia di valori

Problema di ordinamento

- **Ordinare una sequenza in modo non decrescente**
  - Data una sequenza di n numeri $\langle a_1, a_2, \ldots, a_n \rangle$, generare una permutazione $\langle a'_1, a'_2, \ldots, a'_n \rangle$ di $\langle a_1, a_2, \ldots, a_n \rangle$, di in modo che $a'_1 <= a'_2 <= \ldots <= a'_n$
- **Uno dei vantaggi nel mantenere una collezione di dati ordinata è nella possibilità di eseguire operazioni di ricerca in modo efficiente**
- **Per semplicità faremo riferimento al problema dell'ordinamento di array di interi**

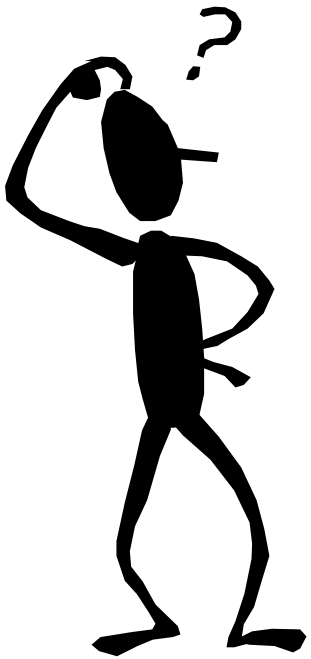
Ordinamento di un array

- **Problema dell'ordinamento non decrescente di un array**
  - sia a un array di interi
  - trasformare l'array a in modo tale che gli elementi vi compaiano in ordine non decrescente
    - modificare la posizione degli elementi di a in modo tale che, per ogni indice i, l'elemento a[i] sia minore o uguale a tutti gli elementi di a di indice j>i



Ordinamento di array

- Si possono utilizzare diverse strategie per ordinare un array (o più in generale una sequenza)



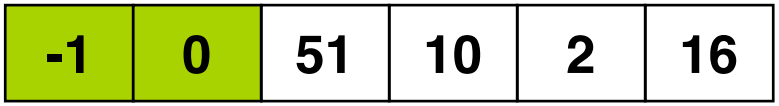
16	2	51	10	0	-1

Ordinamento per selezione

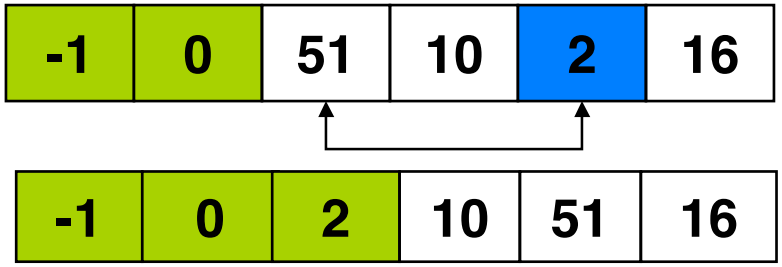
- L'algoritmo di ordinamento per selezione (selection sort) è basato sulla seguente strategia
- finché l'array non è ordinato
  - seleziona l'elemento di valore minimo dell'array tra quelli che non sono stati ancora ordinati
  - disponi questo elemento nella sua posizione definitiva

Strategia dell'ordinamento per selezione

- Supponiamo che i primi due elementi dell'array siano già ordinati:
  - L'array viene partizionato in due sottoarray: il primo che contiene gli elementi già ordinati, il secondo contiene gli elementi ancora da ordinare.



- Si seleziona il minimo sugli elementi ancora da orinare e si scambia con il primo elemento della porzione dell'array ancora da ordinare



Scambio di una coppia di elementi dell'array

- Metodo per scambiare una coppia di elementi in un array

```
/* Scambia gli elementi di indice i e j di dati. */
private static void scambia(int[] dati, int i, int j) {
// pre: dati!=null && 0 <= i,j < dati.length

    int temp;  // variabile di supporto per lo scambio

    /* scambia dati[i] con dati[j] */

    temp = dati[i];
    dati[i] = dati[j];
    dati[j] = temp;
}
```

Strategia dell'ordinamento per selezione...

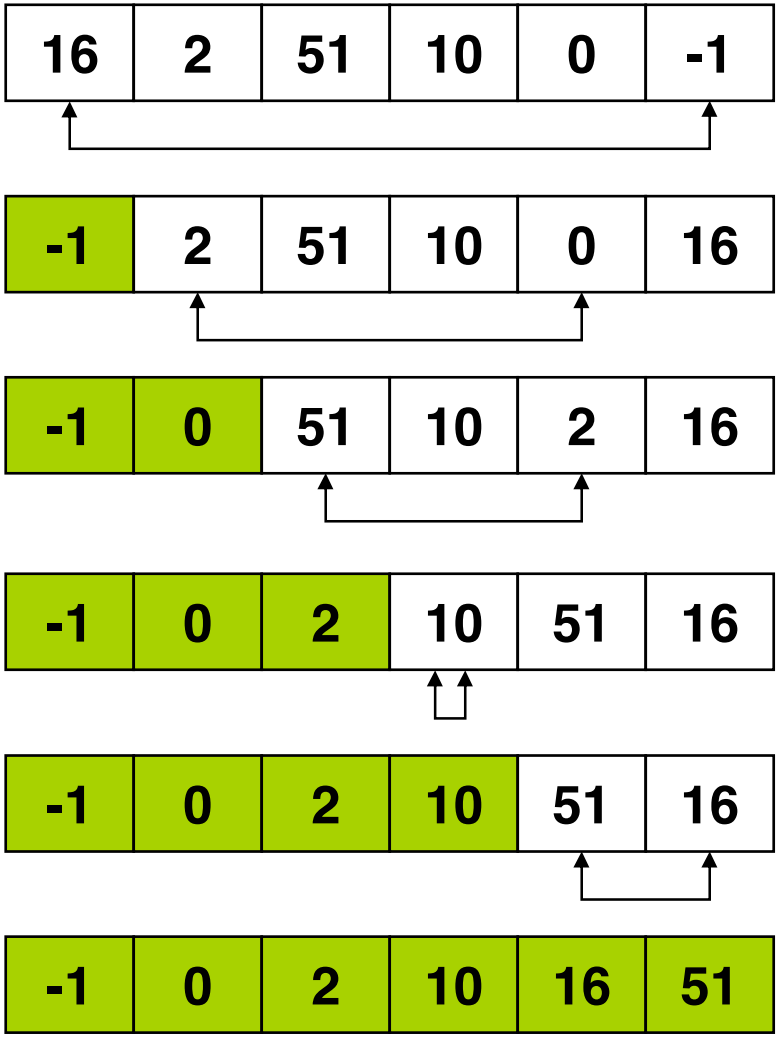
L'algoritmo di ordinamento per selezione

- procede per fasi, chiamate passate
  - ciascuna passata garantisce che almeno un elemento venga "ordinato" – venga posto nella posizione che gli compete nell'ordinamento definitivo
- gestisce una partizione degli elementi dell'array in due insiemi
  - gli elementi ordinati – che sono stati sicuramente collocati nella loro posizione definitiva – l'algoritmo di ordinamento non deve più confrontare né scambiare questi elementi
  - gli elementi non ordinati – che non sono stati ancora collocati nella loro posizione definitiva

...Strategia dell'ordinamento per selezione

- **Nell'ordinamento per selezione**
  - inizialmente, tutti gli elementi sono considerati non ordinati
  - dopo la prima passata, il primo elemento viene ordinato
  - per ordinare l'array si devono eseguire tante passate fino a quando tutti gli elementi dell'array non risultano ordinati

Esempio di applicazione dell'ordinamento per selezione



Implementazione dell'ordinamento per selezione...

```
public static void selectionSort(int[] v) {
int n = v.length;
for (int i = 0; i < n-1; i++) {
   // trova l'elem. minimo con indice tra i e n-1
   int jmin = i;
   for (int j = i+1; j < n; j++) {
       if (v[j] < v[jmin])
       jmin = j;
   }
   // scambia gli elementi con indice i e jmin
   if (i!=jmin)
       scambia(v,i,jmin);
 }
}
```

- Si può scrivere meglio?

...Implementazione dell'ordinamento per selezione...

- Si può definire un metodo per il calcolo dell'indice corrispondente all'elemento di valore minimo

```
/* calcola la posizione in cui si trova il minimo in
 * una porzione dell'array a compresa tra inf e sup */
    public static int min(int[] a, int inf, int sup){
        int i,indmin;
        indmin = inf;
        for (i = inf+1; i <= sup; i++) {
            if (a[i] < a[indmin])
            indmin = i;
        }
        return indmin;
    }
```

...Implementazione dell'ordinamento per selezione...

- Il metodo selectionSort diventa:

```
public static void selectionSort(int[] v) {
      int n;
      n = v.length;
      for (int i = 0; i < n-1; i++)
            scambia(v,i,min(v,i,n-1));
}
```

- Osservazioni?
- Esercizio: scrivere il selectionSort ricorsivo

Complessità dell'ordinamento per selezione

- **Dimensione dell'input:**
  - **lunghezza n dell'array da ordinare**
- **Operazione dominante:**
  - il confronto **v[j] < v[jmin]** durante il calcolo del minimo
- **Caso peggiore:**
  - qualunque

1. **Il confronto v[j] < v[jmin] viene eseguito n-i volte ad ogni passata**
2. **Lo scambio ha complessità costante**
3. **1. e 2. vengono ripetuti n volte (i=0..n-1)**
4. **Complessivamente**

$$\sum_{i=1}^{n-1} n - i = n^2 - \sum_{i=1}^{n-1} i = n^2 - \frac{n(n-1)}{2} = \frac{2n^2 - n^2 + n}{2} = \frac{n^2 + n}{2} = \frac{n(n+1)}{2}$$

- **Quindi il selection sort ha complessità quadratica: $O(n^2)$**

Commento

- **Avendo trovato un algoritmo di complessità quadratica per il problema dell'ordinamento, è possibile concludere che il problema dell'ordinamento ha complessità al più quadratica**
  - **esistono algoritmi di ordinamento con complessità (asintotica, nel caso peggiore) meno che quadratica (ad esempio, lineare) ?**
  - **esistono algoritmi di ordinamento che, pur avendo complessità quadratica nel caso peggiore, hanno una complessità migliore nel caso migliore e nel "caso medio" ?**

Esercizi

- Mostrare l'applicazione dell'ordinamento per selezione al seguente array

10	2	16	0	-1	51	4	23	9	8

  - mostrare lo stato dell'array dopo ciascuna passata, indicando anche quali sono gli elementi ordinati e quali i non ordinati
- Definire un metodo di ordinamento non crescente (anziché non decrescente) basato sull'algoritmo di ordinamento per selezione
- Definisci un metodo basato sull'algoritmo di ordinamento per selezione che ordina un array di stringhe (rispetto all'ordinamento lessicografico)

Ordinamento a bolle

- **Algoritmo di ordinamento a bolle (bubble sort)**
  - la strategia implementata dall'algoritmo è ancora basata su
    - passate
    - confronti e scambi
  - in particolare, l'ordinamento a bolle confronta, durante ciascuna passata, tutte le coppie di elementi adiacenti tra gli elementi non ordinati dell'array
    - ogni volta che una coppia di elementi adiacenti non è ordinata correttamente, gli elementi vengono scambiati

Proprietà di ordinamento: a[0]<a[1]<a[2]<…<a[n-1]

- per ogni coppia a[j], a[j+1] se a[j]>a[j+1], scambia a[j],a[j+1]

passata

```
for (j=0; j<n-1; j++){
  if (a[j]>a[j+1])
    scambia(a[j], a[j+1])
}
```

int a[6]

30 16 31 9 18 4	j=0
16 30 31 9 18 4	j=1
16 30 31 9 18 4	j=2
16 30 9 31 18 4	j=3
16 30 9 18 31 4	j=4
16 30 9 18 4 31	j=5

Prima passata! (n-1 confronti) il più grande a posto

Ordinamento a bolle = sequenza di passate

Seconda passata (n-2 confronti) il penultimo a posto

16 30 9 18 4 **31**	j=0
16 30 9 18 4 **31**	j=1
16 9 30 18 4 **31**	j=2
16 9 18 30 4 **31**	j=3
16 9 18 4 **30 31**	~~j=4~~
	~~j=5~~

passata

```
for (j=0; j<n-2; j++){
   if (a[j]>a[j+1])
      scambia(a[j], a[j+1])
}
```

Ordinamento a bolle = sequenza di passate

Terza passata (n-3 confronti) il terz'ultimo a posto

16 9 18 4 **30 31** j=0

9 16 18 4 **30 31** j=1

9 16 18 4 **30 31** j=2

16 9 4 **18 30 31**

passata

```
for (j=0; j<n-3; j++){
  if (a[j]>a[j+1])
    scambia(a[j], a[j+1])
}
```

Quarta passata (n-4 confronti)

9 4 **16 18 30 31**

```
for (j=0; j<n-4; j++){...
```

Quinta passata cioè (n-1)-esima passata (n-5 = n-n+1 = 1 confronto)

4 **9 16 18 30 31**

```
for (j=0; j<n-5; j++){...
```

---

dopo n-1 passate l'array è ordinato 4 9 16 18 30 31

In generale

i-esima passata

```
for (j=0; j<n-i; j++){
  if (a[j]>a[j+1])
    scambia(a[j], a[j+1])
}
```

algoritmo

per i che conta da 1 a n-1

esegui la i-esima passata

```
public static void bubbleSort(int[] v) {
int n = v.length;
int i, j;
boolean ordinato = false;
 for (i=1; i<n-1; i++)
   for (j=0; j<n-i; j++)
     if (v[j]>v[j+1])
       scambia(v[j], v[j+1]);
}
```

Applicazione sull'array 16 8 2 15 4 9

16 8 15 2 4 9 j=0

8 16 15 2 4 9 j=1

…

8 15 2 4 9 **16** j=5

8 2 4 9 **15 16**

2 4 8 9 **15 16**

2 4 8 9 15 16

2 4 8 9 15 16

2 4 8 9 15 16 !!

Prima passata
5 scambi

Seconda passata
n-2 confronti 3 scambi

Terza passata
n-3 confronti e 2 scambi

Quarta passata
0 scambi

Quinta passata
0 scambi

Applicazione sull'array 16 18 20 25 34 39

prima passata	n-1 confronti e	0 scambi
seconda	n-2	0
terza	n-3	0
quarta	n-4	0
quinta	n-5	0

- L'array potrebbe essere già ordinato prima del termine delle n-1 passate: dipende dalla configurazione della sequenza da ordinare
- L'algoritmo però esegue sempre n-1 passate
- l'i-esima passata esegue sempre n-i confronti (a[i] > a[i+1]) con eventuale scambio

Miglioramento...

Mentre non abbiamo finito {

esegui una passata;

se l'array risulta ordinato

abbiamo finito

}

se durante la passata non abbiamo eseguito scambi

```
while (!finito)   {/* mentre non ... finito */
  fattoscambio = false;
  for (j=0; ... )
    if (a[j]>a[j+1]) {
      scambia(a[j], a[j+1]);
      fattoscambio = true;
    }
  if (fattoscambio==false)
      finito = true;
```

... Miglioramento

```
finito = false;
i=0;       /* fatte 0 passate */
while (!finito) {
  i=i+1;
  fattoscambio = false;
  for (j=0; j<n-i; j++ )
    if (a[j]>a[j+1]) {
      scambia (a[j],a[j+1]);
      fattoscambio = true;
    }
  if ((!fattoscambio)||(i==n-1))
     finito = true;
}
```

Complessità del bubble sort...

- **Istruzione dominante**
  - Confronto tra due elementi adiacenti
    ```
    if (a[j]>a[j+1])
    ```
- **Caso migliore** 16 18 20 25 34 39
  - Finiamo dopo la prima passata (n-1 confronti e 0 scambi)
- **Caso medio** 9 4 31 30 18 16
  - Finiamo dopo alcune passate: $k < n$ confronti e $s < n^2$ scambi
- **Caso peggiore** 30 16 31 9 18 4
  - numero di confronti massimo
- **Caso PEGGIORE** 39 34 25 20 18 16
  - massimo numero di confronti, ognuno con scambio e assegnazione

...Complessità del bubble sort...

- Il miglioramento permette di mettere a frutto eventuali configurazioni favorevoli (ad es. array parzialmente ordinato)
- Nei casi sfavorevoli si comporta come la versione iniziale
- Nei casi favorevoli guadagna.

...Complessità del bubble sort

- prima passata — *n*-1 confronti
- ...
- (*i*-esima passata) — *n-i* confronti
- ...
- (*n*-1)-esima passata — 1 confronto

- Caso peggiore: n-1 passate

$$\sum_{i=1}^{n-1}(n-i) = (n-1)n - \sum_{i=1}^{n-1} i = (n-1)n - \frac{(n-1)n}{2} = \frac{n(n-1)}{2} \qquad O(n^2)$$

- Caso migliore: 1 passata

unica passata: *n*-1 confronti — $O(n)$

si presenta solo per il bubble sort migliorato

Esercizi

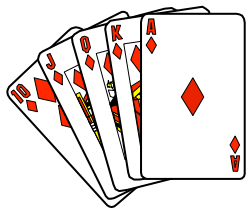
- **Mostrare l'applicazione dell'ordinamento a bolle (entrambe le versioni) al seguente array**

10	2	16	0	-1	51	4	23	9	8

  - **mostra lo stato dell'array dopo ciascuna passata, indicando anche quali sono gli elementi ordinati e quali i non ordinati**

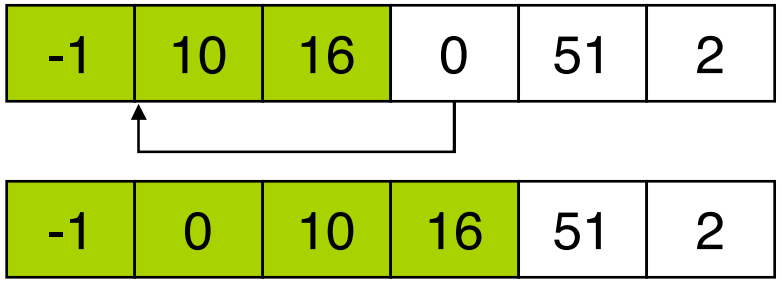
Ordinamento per inserzione

- **Algoritmo di ordinamento per inserzione (insertion sort)**
  - gli elementi sono partizionati in due insiemi
    - elementi relativamente ordinati – elementi ordinati tra di loro, ma che non sono stati necessariamente collocati nelle loro posizioni definitive
    - elementi non relativamente ordinati
  - inizialmente viene considerato relativamente ordinato il primo elemento dell'array
  - ad ogni passata
    - colloca il primo tra gli elementi non relativamente ordinati tra quelli relativamente ordinati

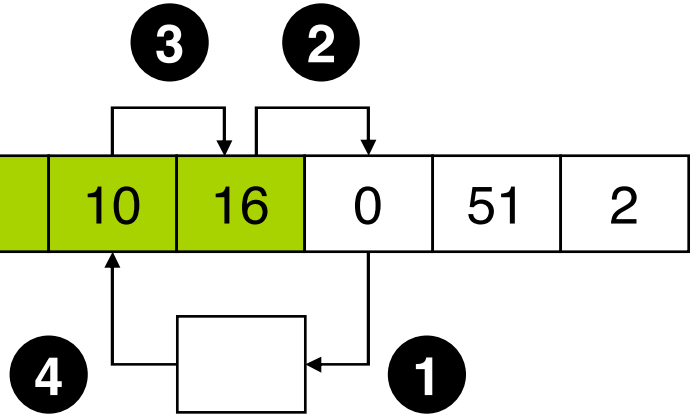


Una passata dell'ordinamento per inserzione

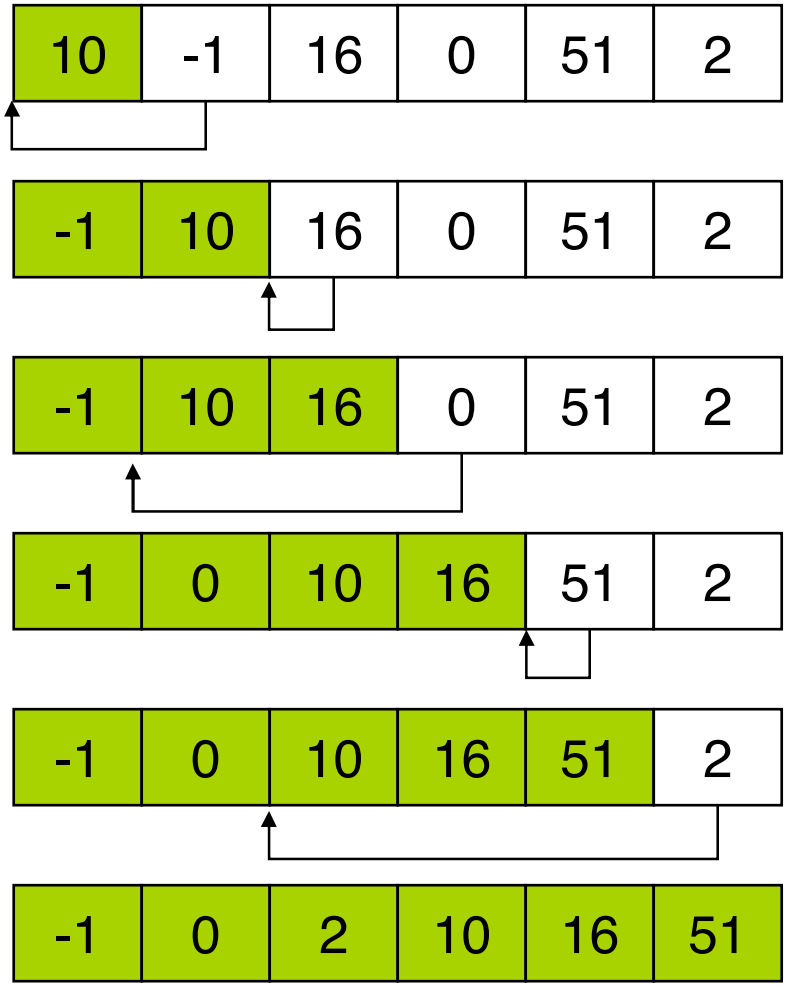
- Effetto di una passata dell'ordinamento per inserzione



- Dinamica dell'inserimento



Applicazione completa dell'ordinamento per inserzione



Implementazione dell'ordinamento per inserzione

```
/* Ordina l'array dati in modo non decrescente.
 * Ordinamento per inserzione. */
public static void insertionSort(int[] dati) {
// pre: dati!=null
int n;              // lunghezza di dati
int i;              // indice per la scansione di dati
int ordinati;       // numero di elementi
                    // "relativamente ordinati"
int corrente;       // elemento da "ordinare"
boolean ins;        // è possibile inserire corrente
                    // tra gli "relativamente ordinati"
```

... segue ...

Implementazione dell'ordinamento per inserzione

```
n = dati.length;
/* esegue n-1 passate */
for (ordinati=1; ordinati<n; ordinati++) {
  /* gli elementi "relativamente ordinati" sono
   * quelli di indice tra 0 e ordinati */
  /* viene "ordinato" il primo elemento
   * tra i "non relativamente ordinati" */
   corrente = dati[ordinati];
   ins = false;
   i = ordinati;
   while (!ins && i>0)
    if (corrente<dati[i-1]) { // sposta verso dx
          dati[i] = dati[i-1];
          i--;
     } else
         ins = true;
  /* inserisce corrente tra i "rel. ordinati" */
  dati[i] = corrente;
  /* ora gli elementi "rel. ordinati" sono quelli
  * di indice compreso tra 0 e ordinati */
  } }
```

Complessità dell'ordinamento per inserzione

Complessità asintotica dell'ordinamento per inserzione

- rispetto alla lunghezza *n* dell'array da ordinare
- operazione dominante
  - il confronto corrente<dati[i-1]
- caso peggiore
  - l'array è ordinato in modo decrescente
- il numero complessivo di confronti è quadratico

Insertion sort ha complessità $O(n^2)$

Esercizio

- cosa succede se la ricerca della posizione corrente in cui inserire viene effettuata mediante la ricerca binaria?

Esercizi

- **Mostrare l'applicazione dell'ordinamento per inserzione al seguente array**

10	2	16	0	-1	51	4	23	9	8

  - **mostra lo stato dell'array dopo ciascuna passata, indicando anche quali sono gli elementi relativamente ordinati e quali i non relativamente ordinati**

- **Definire un metodo di ordinamento non crescente (anziché non decrescente) basato sull'algoritmo di ordinamento per inserzione**

Considerazioni...

- Per ordinare 1.000 elementi usando un algoritmo di complessità $O(n^2)$ (ordinamento per selezione, inserzione o a bolle) sono necessarie 1.000.000 di operazioni. Esistono algoritmi di ordinamento più efficienti?
- SI
- *Idea:* Divido i 1000 elementi in due gruppi da 500 elementi ciascuno:
  - ordino il primo gruppo in $O(n^2)$: 250.000 operazioni
  - ordino il secondo gruppo in $O(n^2)$: 250.000 operazioni
  - combino (fondo) i due gruppi ordinati: si può fare in $O(n)$, cioè 1.000 operazioni
  - Totale: 501.000 operazioni (contro 1.000.000)

...Considerazioni

- **Il procedimento può essere iterato:**
  - **per ordinare le due metà non uso un algoritmo di complessità $O(n^2)$, ma applico lo stesso procedimento di divisione, ordinamento separato e fusione.**
- **La suddivisione in due metà si ferma quando si arriva ad un gruppo costituito da un solo elemento (che è già ordinato).**

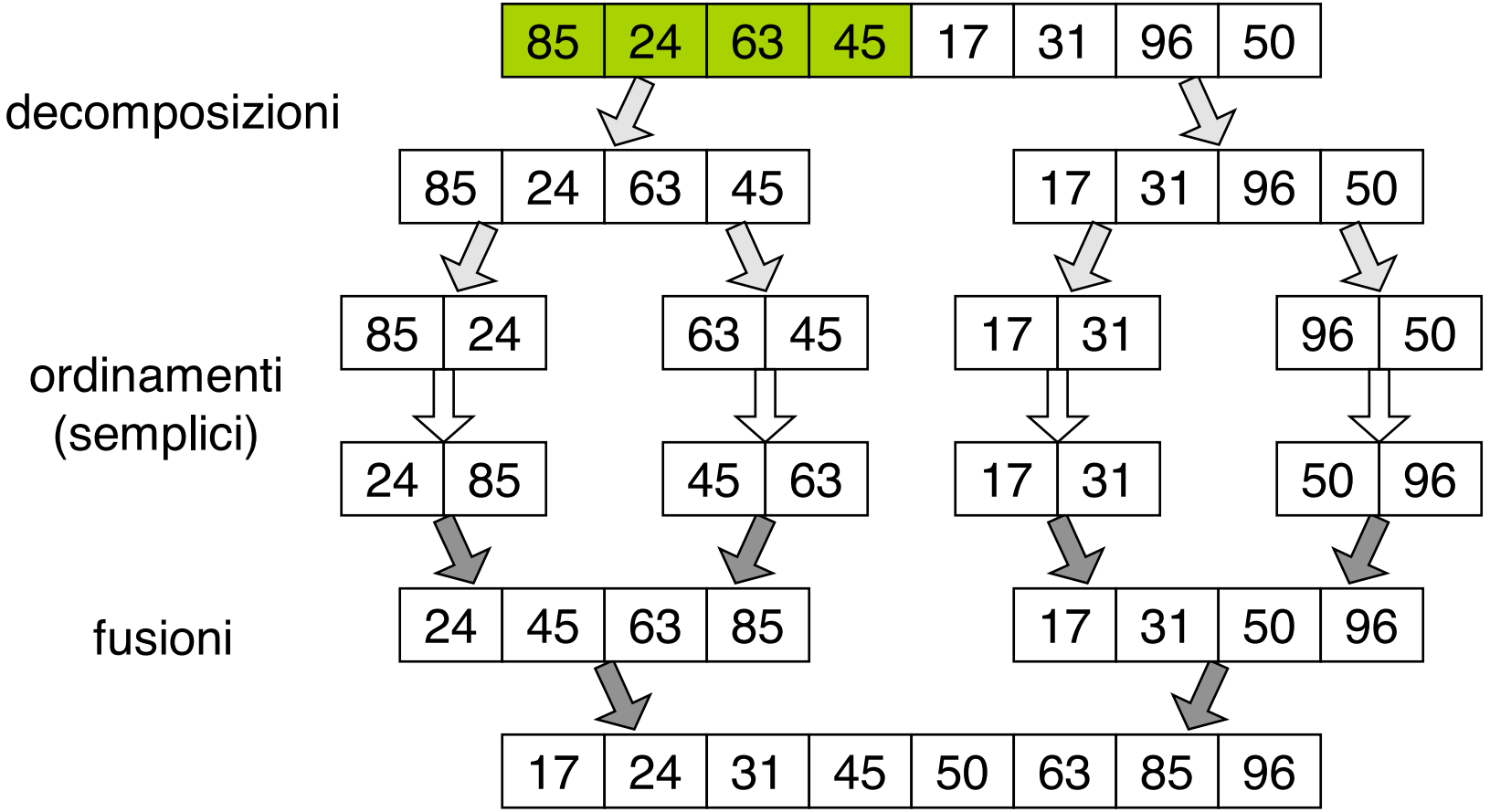
Ordinamento per fusione

- Algoritmo di ordinamento per fusione (merge sort)
- se la sequenza da ordinare contiene un elemento, allora la sequenza viene ordinata direttamente
- se invece la sequenza da ordinare contiene due o più elementi, allora viene ordinata come segue
  - gli elementi della sequenza vengono partizionati in due sotto-sequenze
  - le due sotto-sequenze vengono ordinate separatamente
  - le due sotto-sequenze ordinate vengono fuse in un'unica sequenza ordinata
- Strategia alla base dell'ordinamento per fusione: divide et impera
  - l'ordinamento di una sequenza lunga può essere ricondotto all'ordinamento di due sequenze più corte – seguito da una ricostruzione del risultato

Algoritmo di ordinamento per fusione

- Ordina per fusione gli elementi del vettore a da inf a sup:
- if (inf < sup) se c'è più di un elemento tra inf e sup, estremi inclusi
  - med = (inf + sup)/2;
  - *ordina per fusione gli elementi di v da inf a med*
  - *ordina per fusione gli elementi di v da med+1 a sup*
  - *fondi gli elementi di v da inf a med con gli elementi di v da med+1 a sup* (*restituendo il risultato nel sottovettore di v da inf a sup*)

Applicazione completa dell'ordinamento per fusione



Implementazione del merge sort

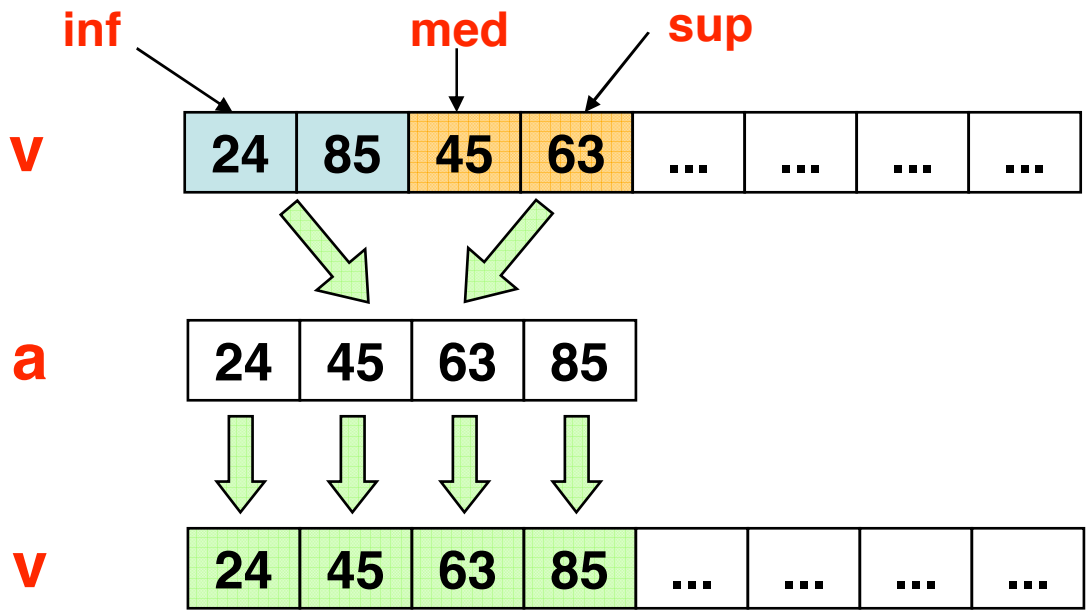
```
public static void mergesort(int v[]) {
  msort(v,0,v.length-1); }


private static void msort(int[] v,
                          int inf, int sup) {
  if (inf < sup) {
     int med = (inf+sup)/2;
     msort(v,inf,med);
     msort(v,med+1,sup);
     merge(v,inf,med,sup);
  }
}
```

Algortimo per la fusione di due array ordinati...

- Per effettuare la fusione di due sottovettori ordinati e contigui ottenendo un unico sottovettore ordinato, si utilizzano un vettore di appoggio e due indici per scandire i due sottovettori, e si procede al seguente modo:
- Il più piccolo tra i due elementi indicati dai due indici viene copiato nella prossima posizione del vettore di appoggio, e viene fatto avanzare l'indice corrispondente; si continua così fino a quando uno dei due sottovettori non è terminato;
- Quando uno dei due sottovettori è terminato si copiano gli elementi rimanenti dell'altro nel vettore di appoggio;
- Alla fine si ricopia il vettore di appoggio nelle posizioni occupate dai due sottovettori.

...Algortimo per la fusione di due array ordinati

- le due sequenze ordinate sono sottosequenze contigue dell'array a, delimitate da indici passati come parametri
- la sequenza ordinata risultato viene temporaneamente memorizzata nell'array di appoggio a e poi copiata in v



Implementazione dell'algortimo di fusione di due array ordinati...

```
private static void merge(int[] v, int inf, int med,
                                                int sup) {
  int[] a = new int[sup-inf+1];
  int i1 = inf;
  int i2 = med+1;
  int i = 0;
  while ((i1 <= med) && (i2 <= sup)) {
    // entrambi i vettori contengono elementi
    if (v[i1] <= v[i2]) {
       a[i] = v[i1];
       i1++;
       i++;
    }
    else {
       a[i] = v[i2];
       i2++;
       i++;
    }
  }
```

... segue ...

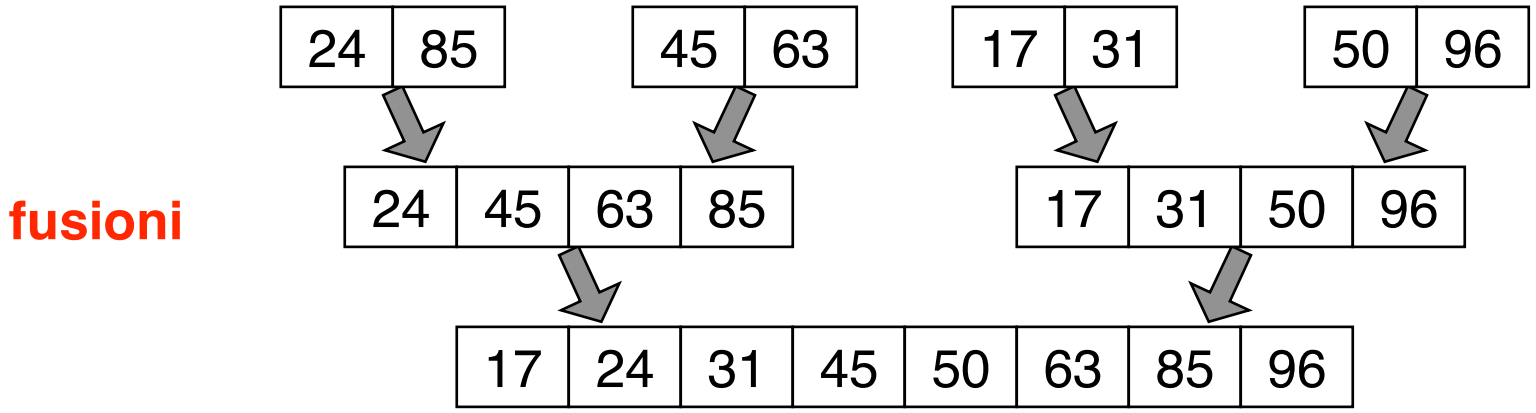
...Implementazione dell'algortimo di fusione di due array ordinati

```
 if (i2 > sup) // e' finito prima il secondo pezzo del vettore
    for (int k = i1; k <= med; k++) {
      a[i] = v[k];
      i++;
    }
  else // e' finito prima il primo pezzo del vettore
    for (int k = i2; k <= sup; k++) {
      a[i] = v[k];
      i++;
  }
  // copiamo il vettore ausiliario nel vettore originario
  for(int k = 0; k < a.length; k++)
    v[inf+k] = a[k];
}
```

Complessità asintotica dell'ordinamento per fusione...

- discussa in modo informale
  - Le equazioni di ricorrenza sono strumenti metodologici per studiare la complessità di algoritmi ricorsivi: non verrranno trattate in questo corso
- rispetto alla lunghezza *n* dell'array da ordinare
- attività dominante
  - fusione di sottosequenze ordinate
- caso peggiore
  - qualunque

...Complessità dell'ordinamento per fusione



- un certo numero di livelli di decomposizioni e fusioni
- in ciascun livello vengono fusi tutti gli elementi
- il costo asintotico di ciascun livello di fusioni, complessivamente, è n
- il numero di livelli è $log_2\ n$

$$T_{mergeSort}(n) = n \log n$$

Valutazioni quantitative

- **Ma il merge sort è veramente più efficiente dei precedenti algoritmi?**

ORDINAMENTO DI UN ARRAY DI 1.000.000 DI NUMERI

hardware:	supercalcolatore	personal computer
linguaggio:	macchina	di alto livello
compilatore:		non efficiente
programmatore:	esperto	medio
algoritmo:	insertion sort	merge sort
tempo:	5.56 ore	16.67 minuti

Esercizio

- **Mostrare l'applicazione dell'ordinamento per fusione al seguente array**

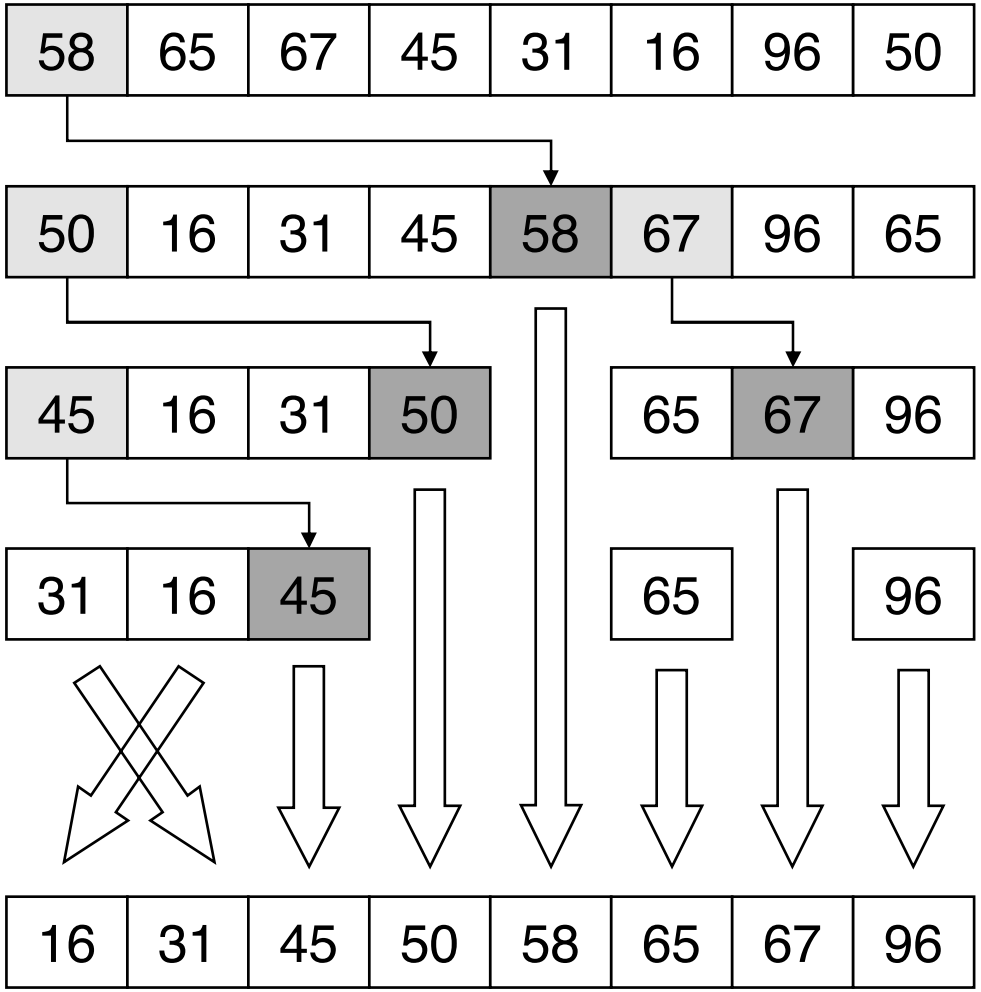
10	2	16	0	-1	51	4	23	9	8

  - **mostrare le varie sottosequenze che vengono identificate e fuse**
  - **mostrare anche l'ordine in cui vengono svolte le varie operazioni**

Ordinamento veloce: quick sort

- L'algoritmo di ordinamento di array più usato in pratica è l'ordinamento veloce (quick sort)
  - se la sequenza contiene un elemento o è vuota, è già ordinata (non bisogna fare nulla)
  - se invece S contiene due o più elementi
  - Si determina un elemento della sequenza che fungerà da discriminante degli elementi della sequenza: il pivot.
  - L'array verrà partizionato in due sottosequenze (nonnecessariamente della stessa lunghezza)
    - Quella più a sinistra conterrà tutti gli elementi minori o uguali del pivot
    - Quella più a destra conterrà tutti gli elementi maggiori o uguali del pivot
    - Alle due sottosequenze così ottenute verrà nuovamente applicato il quick sort

Applicazione dell'ordinamento veloce



Complessità dell'ordinamento veloce

- È stato dimostrato che l'ordinamento veloce ha complessità asintotica:
- $O(n^2)$ nel caso peggiore
- $O(n \log n)$ nel caso medio
- $O(n \log n)$ nel caso migliore
- L'ordinamento veloce viene solitamente preferito all'ordinamento per fusione perché
  - è molto probabile che si verifichi il caso medio
  - il fattore moltiplicativo del termine $n \log n$ per l'ordinamento veloce nel caso medio è minore del fattore moltiplicativo del termine $n \log n$ nell'ordinamento per fusione

Risultati sperimentali

- Tempi medi di esecuzione, in secondi

Algoritmo \ *n*	1 000	10 000	100 000	1 000 000
selection sort	0.02	0.27	24.05	circa 40 minuti
bubble sort elementare	0.01	0.66	62.35	circa 100 minuti
bubble sort	0.02	0.67	62.20	circa 100 minuti
bubble sort bidirezionale	0.01	0.47	43.25	circa 70 minuti
insertion sort	0.01	0.23	18.80	circa 30 minuti
merge sort	0.01	0.01	0.05	0.51
quick sort (API di Java)	0.01	0.02	0.05	0.35

Generazione di un array casuale di dati

- Metodo usato per generare in modo casuale gli array da ordinare

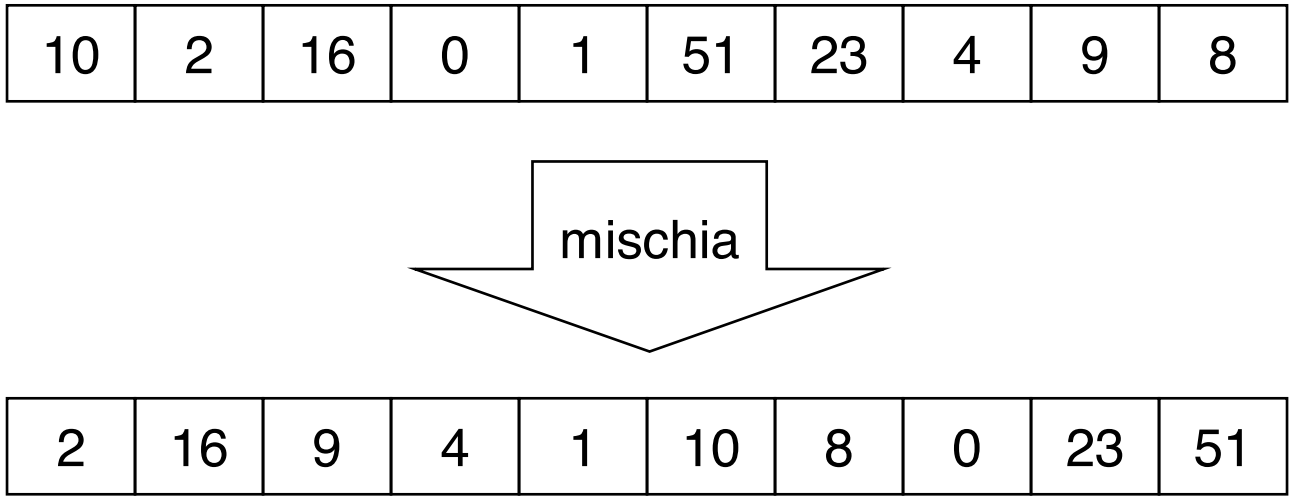
```
/* Crea un array di n numeri casuali. */
public static int[] randomIntArray(int n) {
   // pre: n>=0
   int[] a;    // array di numeri casuali
   int i;      // indice per la scansione di a

   /* crea l'array a */
   a = new int[n];
   /* assegna valori casuali agli elementi di a */
   for (i=0; i<n; i++)
       /* numeri causali compresi tra 0 e 10*n-1 */
       a[i] = (int) (10*n*Math.random());
   return a; }
```

Esercizi

- Scrivere un metodo che, ricevendo come parametro un array di oggetti Rettangolo, ordina l'array in modo non decrescente
  - rispetto all'area dei rettangoli
- Scrivere un metodo che, ricevendo come parametro un array di oggetti Rettangolo, ordina l'array in modo non decrescente
  - rispetto alla base dei rettangoli
  - a parità di base, i rettangoli vanno ordinati per altezza non decrescente

Esercizi

- Scrivere un metodo **void mischia(int[] a)** che modifica, in modo casuale, l'ordine degli elementi all'interno di **a**



Esercizio

SORT ? QUIZ

- Indovina quale metodo di ordinamento è stato usato per ordinare in modo crescente un array di interi di cui si conoscono le diverse successive configurazioni dopo ogni passata

Quale metodo di ordinamento è stato usato ?

10	2	0	16	51	-1

2	0	10	16	-1	51

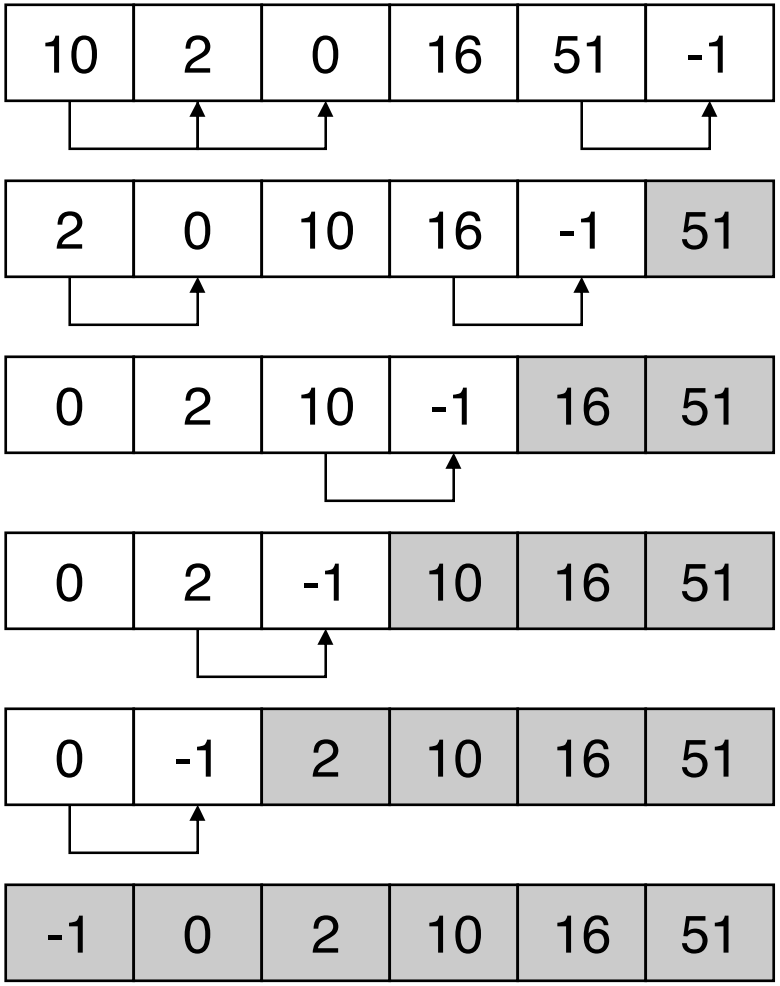
0	2	10	-1	16	51

0	2	-1	10	16	51

0	-1	2	10	16	51

-1	0	2	10	16	51

Applicazione completa dell'ordinamento a bolle



Quale metodo di ordinamento è stato usato ?

85	24	63	45	17	31	96	50

85	24	63	45	17	31	96	50

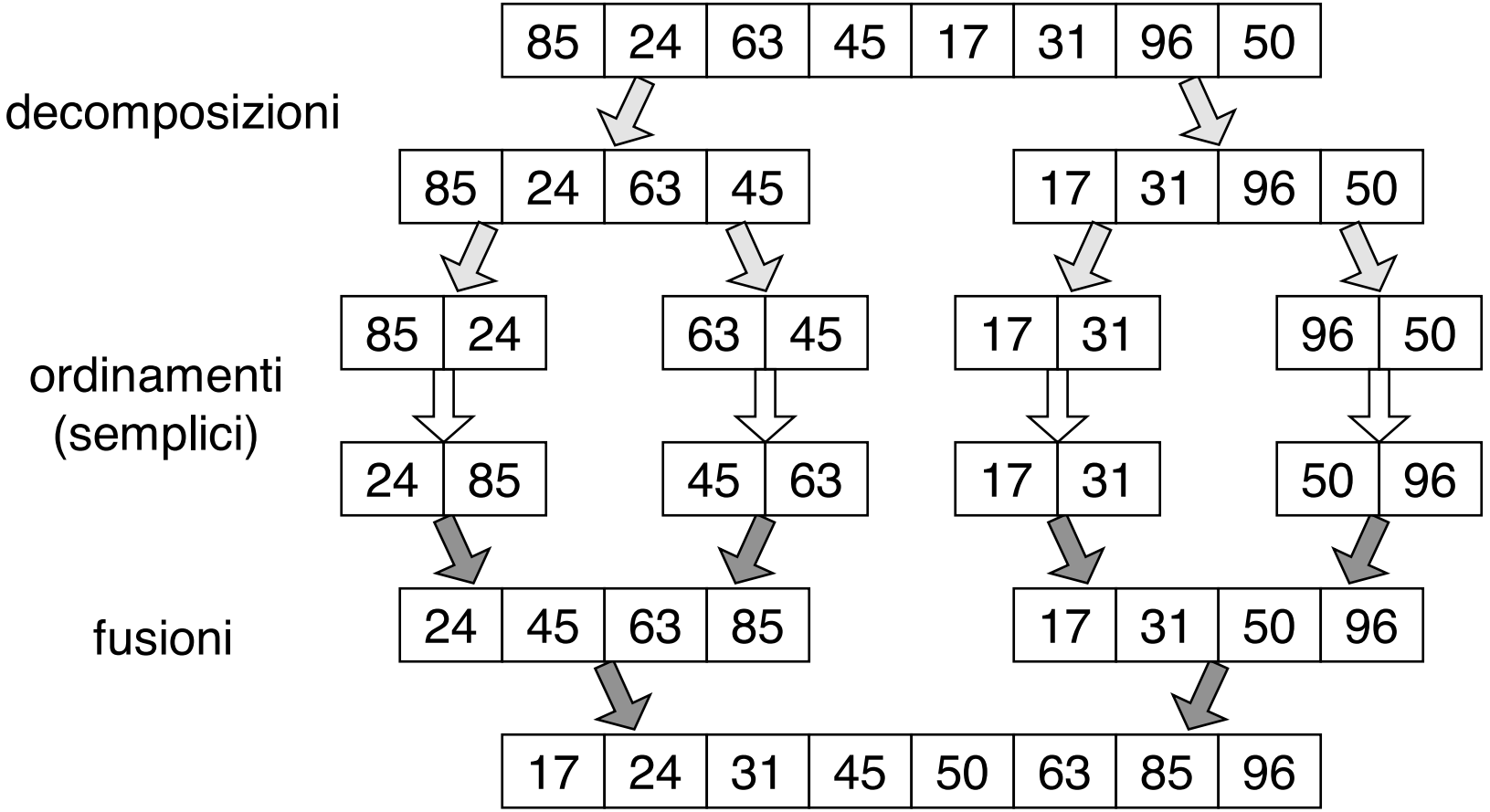
24	85	45	63	17	31	50	96

24	45	63	85	17	31	50	96

24	45	63	85	17	31	50	96

17	24	31	45	50	63	85	96

Applicazione completa dell'ordinamento per fusione



Quale metodo di ordinamento è stato usato ?

16	2	51	10	0	-1

-1	2	51	10	0	16

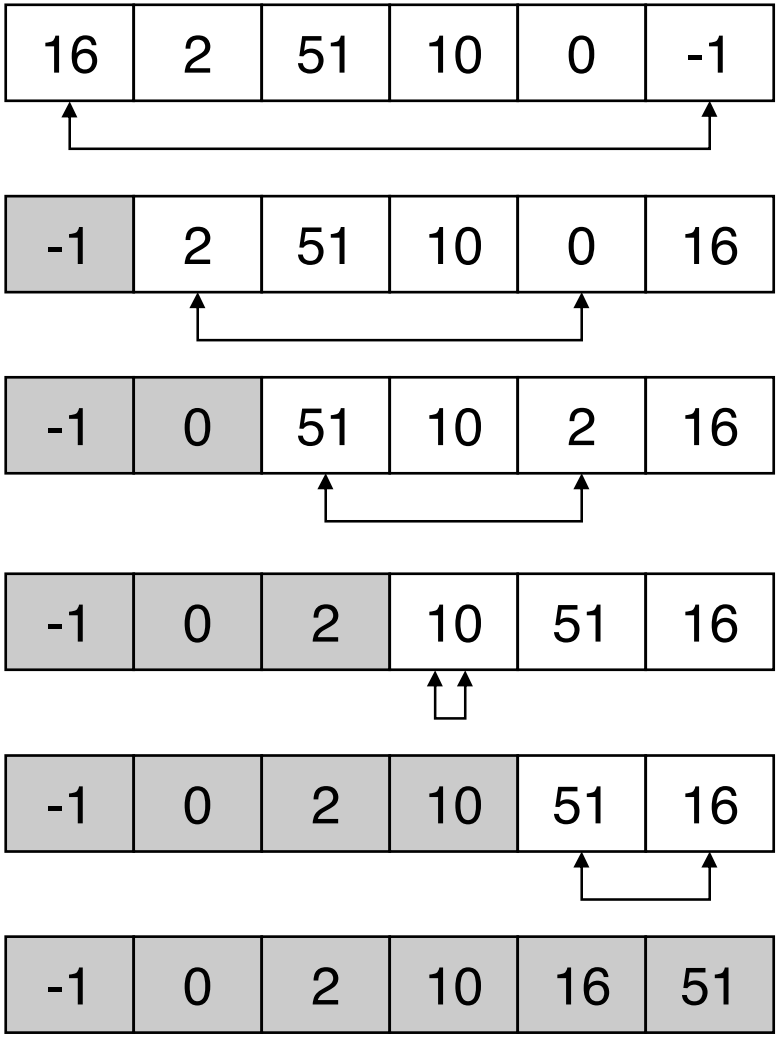
-1	0	51	10	2	16

-1	0	2	10	51	16

-1	0	2	10	51	16

-1	0	2	10	16	51

Applicazione dell'ordinamento per selezione



Quale metodo di ordinamento è stato usato ?

10	-1	16	0	51	2

-1	10	16	0	51	2

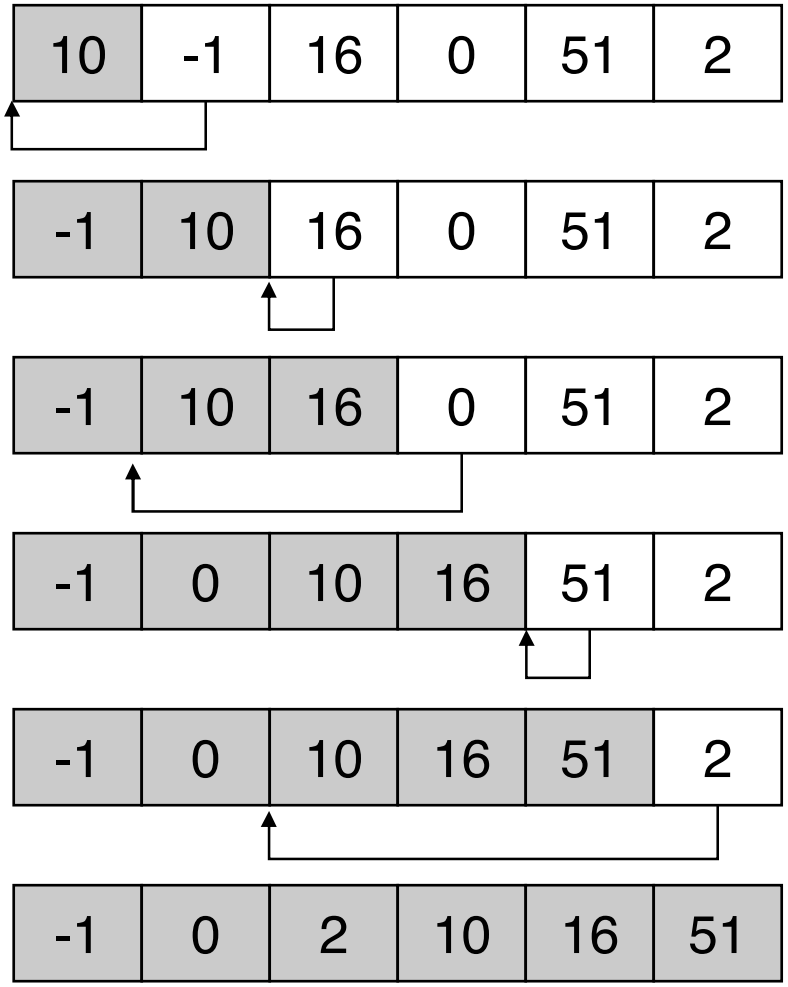
-1	10	16	0	51	2

-1	0	10	16	51	2

-1	0	10	16	51	2

-1	0	2	10	16	51

Applicazione completa dell'ordinamento per inserzione



Riferimenti al libro di testo

- Per lo studio di questi argomenti si fa riferimento al libro di testo, e in particolare al Capitolo 24