Ordinamento

Lorenzo Donatiello,Moreno Marzolla
Dip. di Scienze dell'Informazione
Università di Bologna

Ordinamento

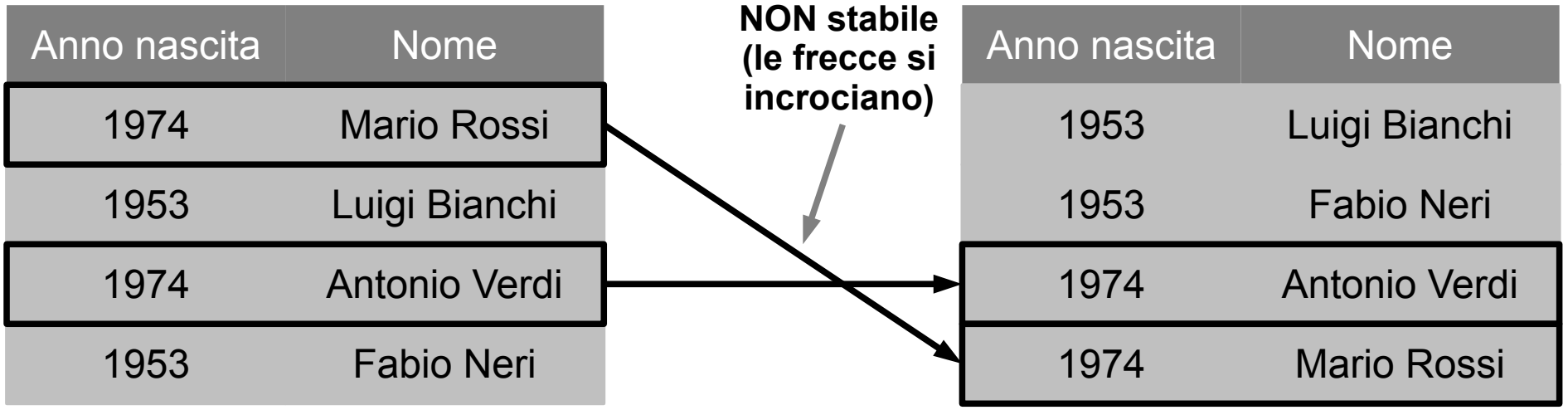
- Consideriamo un array di n numeri v[1], v[2], ... v[n]
- Vogliamo trovare (indirettamente) una permutazione
  p[1], p[2], ... p[n]
  degli interi 1, ..., n tale che
  $v[p[1]] \leq v[p[2]] \leq \ldots \leq v[p[n]]$
- Esempio:
  - v = [7, 32, 88, 21, 92, -4]
  - p = [6, 1, 4, 2, 3, 5]
  - v[p[]] = [-4, 7, 21, 32, 88, 92]

Ordinamento

- Più in generale: è dato un array di n elementi, tali che ciascun elemento sia composto da:
  - una **chiave**, in cui le chiavi sono confrontabili tra loro
  - un **contenuto** arbitrario
- Vogliamo permutare l'array in modo che le chiavi compaiano in ordine non decrescente (oppure non crescente)

Definizioni

- Ordinamento in loco
  - L'algoritmo permuta gli elementi direttamente nell'array originale, senza usare un altro array di appoggio
- Ordinamento stabile
  - L'algoritmo preserva l'ordine con cui elementi con la stessa chiave compaiono nell'array originale



Nota

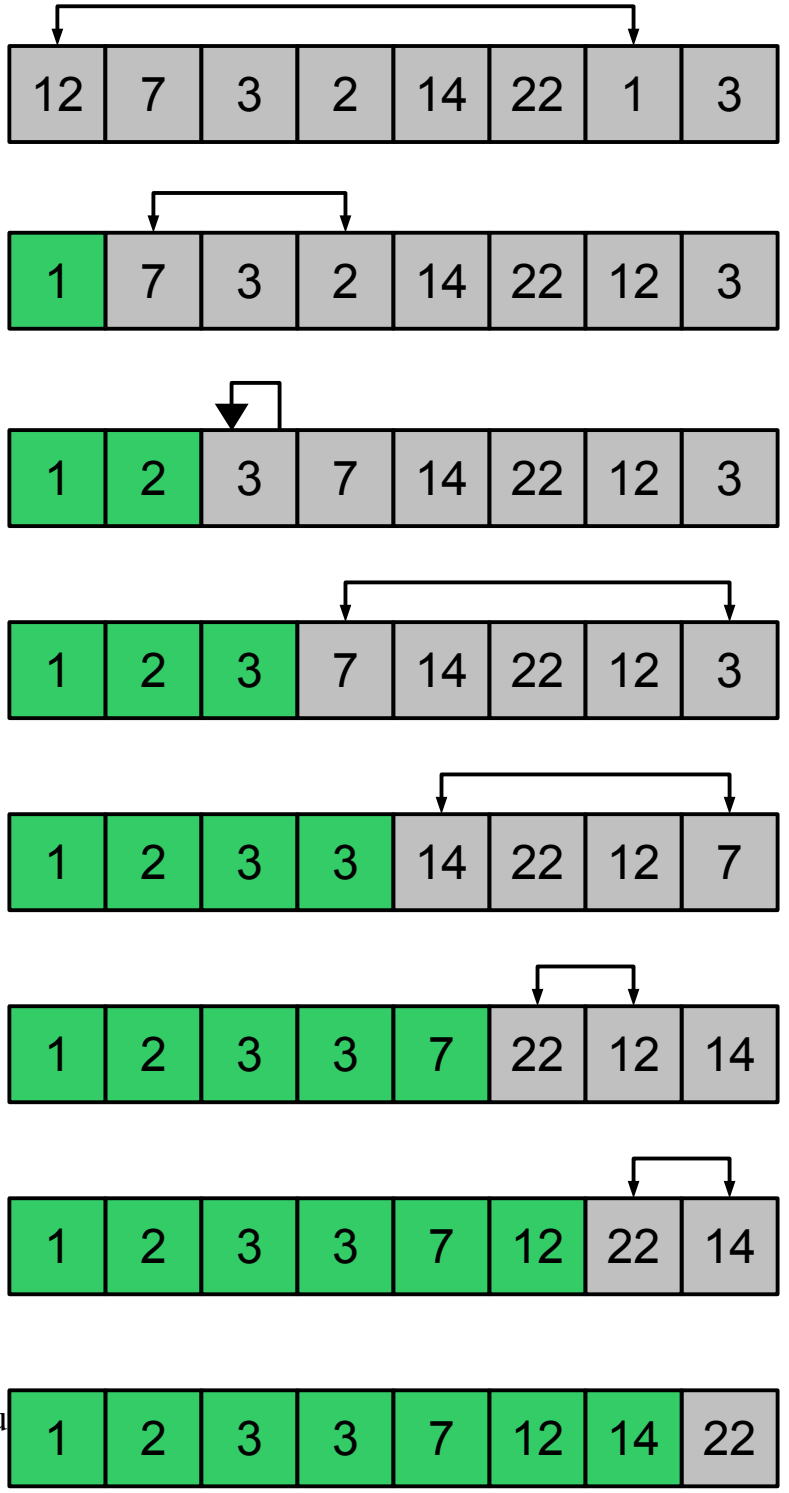
- È possibile rendere ogni algoritmo stabile:
  - Basta usare come chiave di ordinamento la coppia (chiave, posizione nell'array non ordinato)
  - (k1, p1) < (k2, p2) se e solo se:
    - (k1 < k2), oppure
    - (k1 == k2) and (p1 < p2)

Algoritmi di ordinamento "incrementali"

- Partendo da un prefisso A[1..k] ordinato, "estendono" la parte ordinata di un elemento: A[1..k+1]
- Selection sort
  - Cerca il minimo in A[k+1..n] e spostalo in posizione k+1
- Insertion sort
  - Inserisce l'elemento A[k+1] nella posizione corretta all'interno del prefisso già ordinato A[1..k]

Selection Sort

- Cerco il minimo in A[1]...A[n] e lo scambio con A[1]
- Cerco il minimo in A[2]...A[n] e lo scambio con A[2]
- ...
- Cerco il minimo in A[k]...A[n] e lo scambio con A[k]
- ...

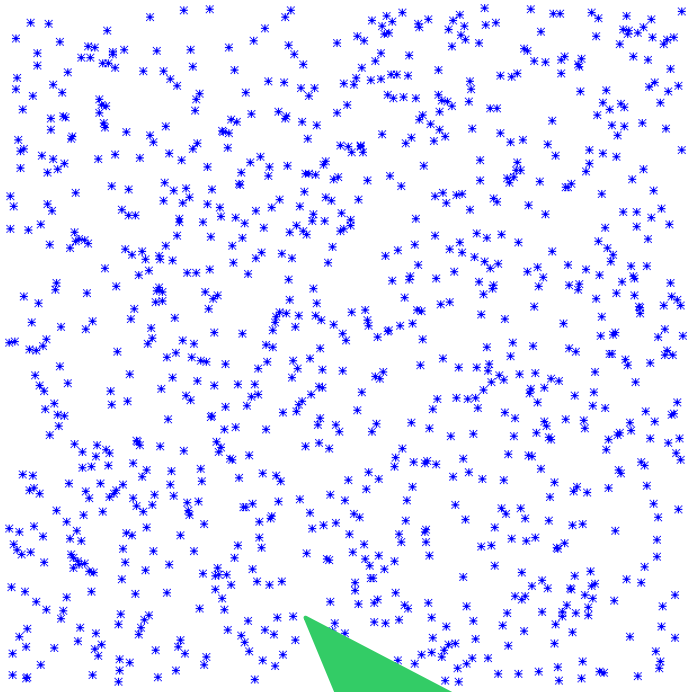


Selection sort

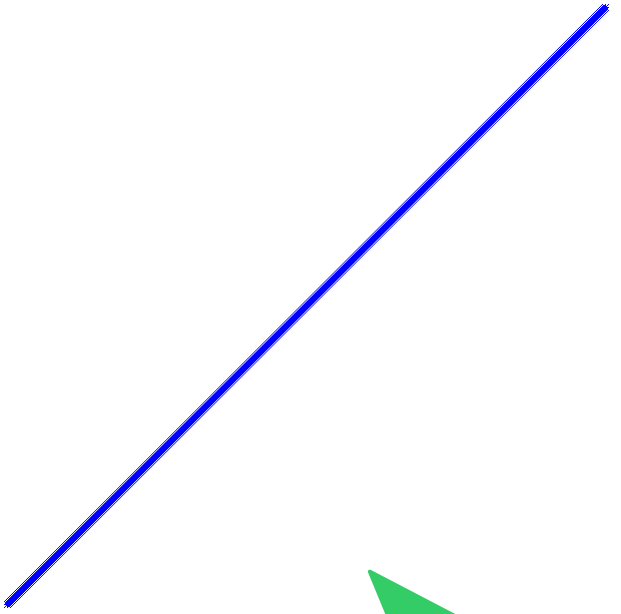
```
selectionSort(ITEM[ ] A, integer n)

for integer i ← 1 to n do
    integer j ← min(A, i, n)
    A[i] ↔ A[j]

integer min(ITEM[ ] A, integer k, integer n)
    integer min ← k                          % Posizione del minimo parziale
    for integer h ← k + 1 to n do
        if A[h] < A[min] then min ← h        % Nuovo minimo parziale
    return min
```

“Visualizzare” il comportamento di un algoritmo di ordinamento

- Consideriamo un vettore A[] contenente tutti e soli gli interi da 1 a N
- Plottiamo i punti di coordinate (i, A[i])

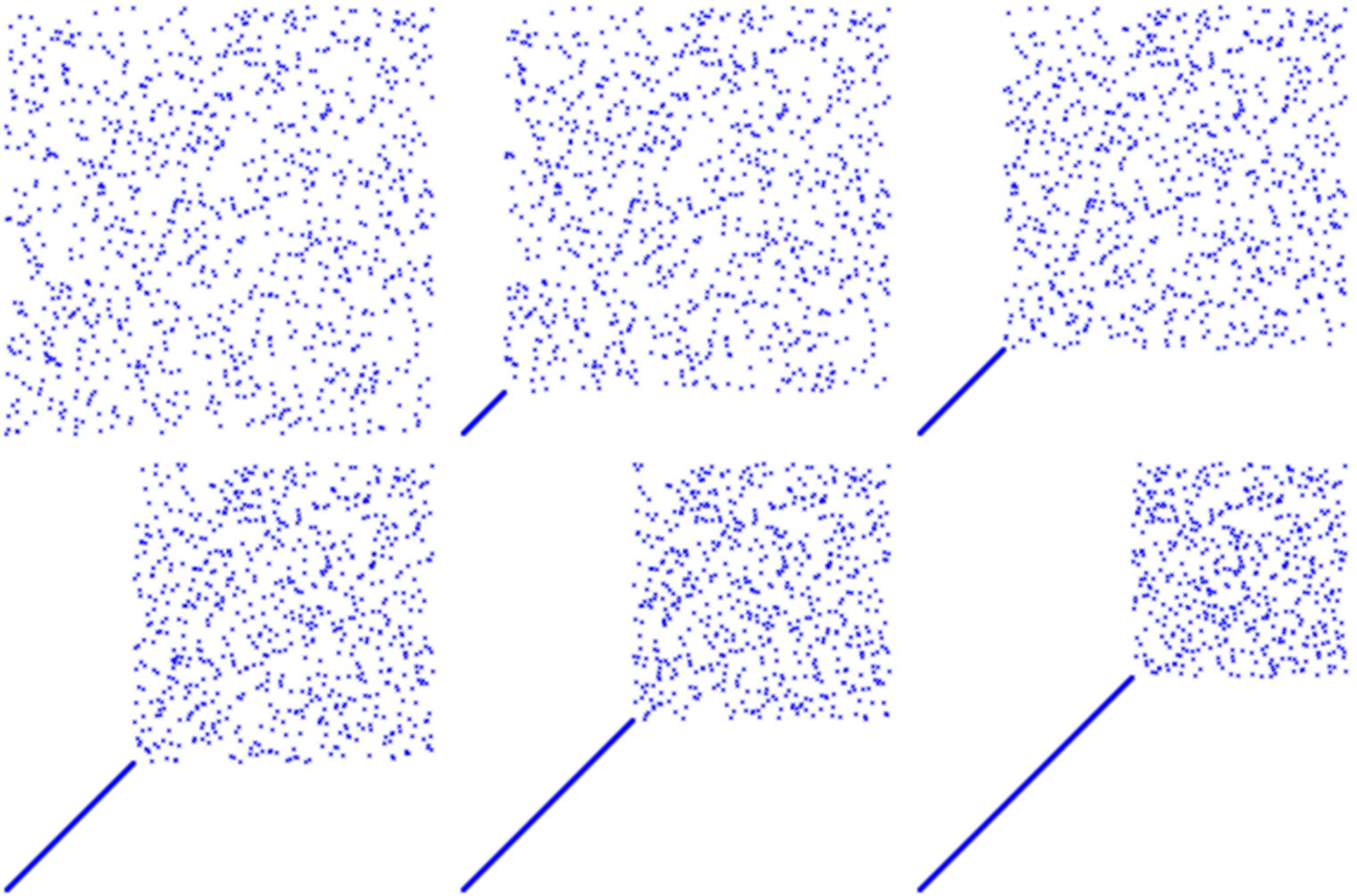


Situazione iniziale (array disordinato)



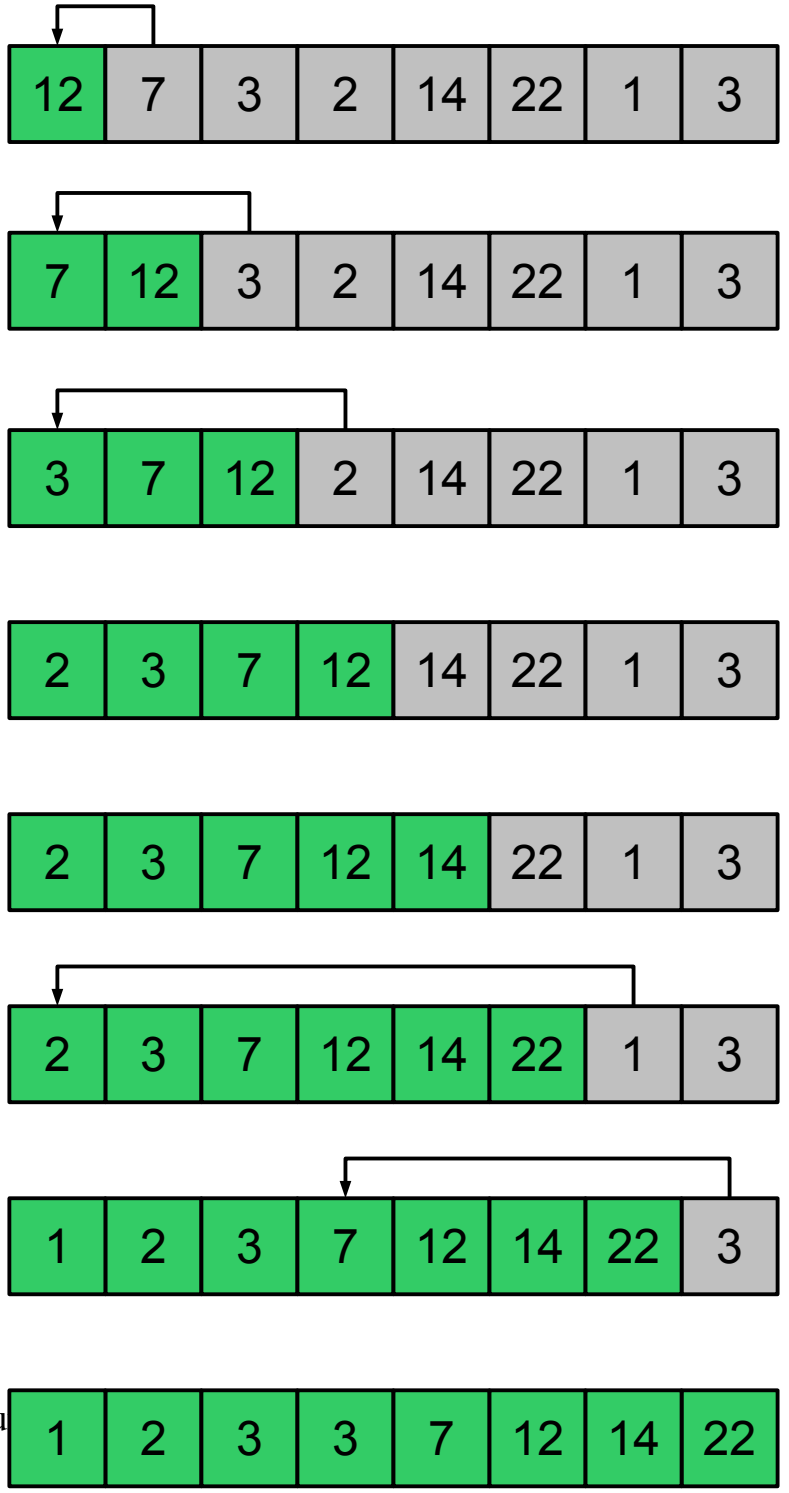
Situazione finale (array ordinato)

Selection Sort per immagini



Insertion Sort

- Idea: al termine del passo k, il vettore ha le prime k componenti ordinate
- Inserisco l'elemento di posizione k+1 nella posizione corretta all'interno dei primi k elementi ordinati



Insertion Sort

- **Algoritmo efficiente per ordinare piccoli insieme di elementi**
- **Come ordinare una sequenza di carte da gioco "a mano"**

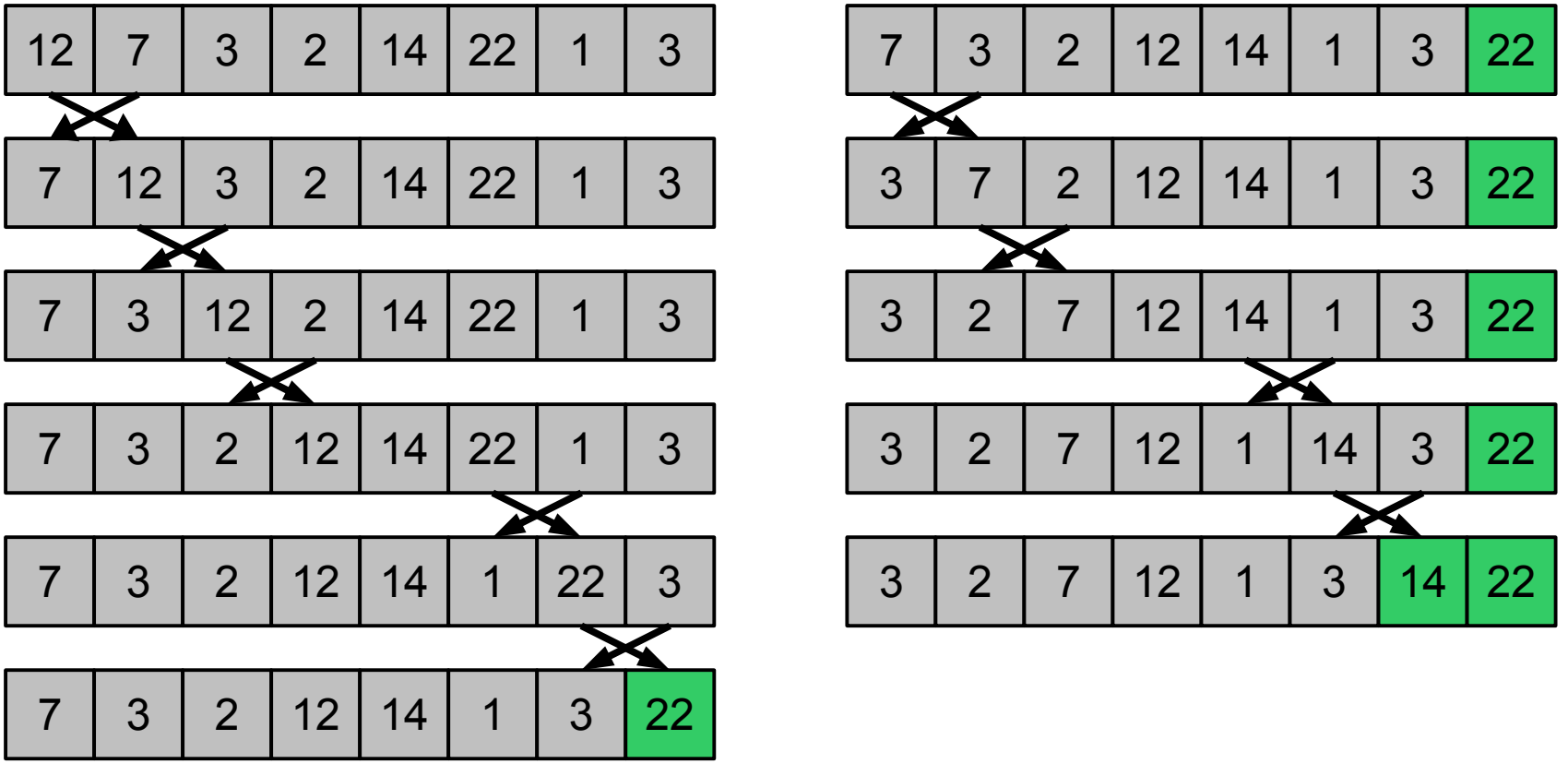
```
insertionSort(ITEM[ ] A, integer n)
for integer i ← 2 to n do
    ITEM temp ← A[i]
    integer j ← i
    while j > 1 and A[j − 1] > temp do
        A[j] ← A[j − 1]
        j ← j − 1
    A[j] ← temp
```

1	2	3	4	5	6	7
7	(4)	2	1	8	3	5
4	7	(2)	1	8	3	5
2	4	7	(1)	8	3	5
1	2	4	7	(8)	3	5
1	2	4	7	8	(3)	5
1	2	3	4	7	8	(5)
1	2	3	4	5	7	8

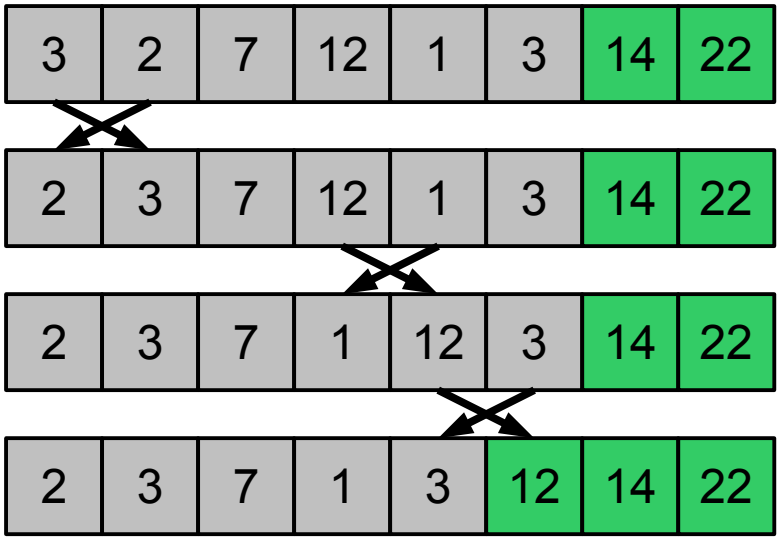
Bubble Sort

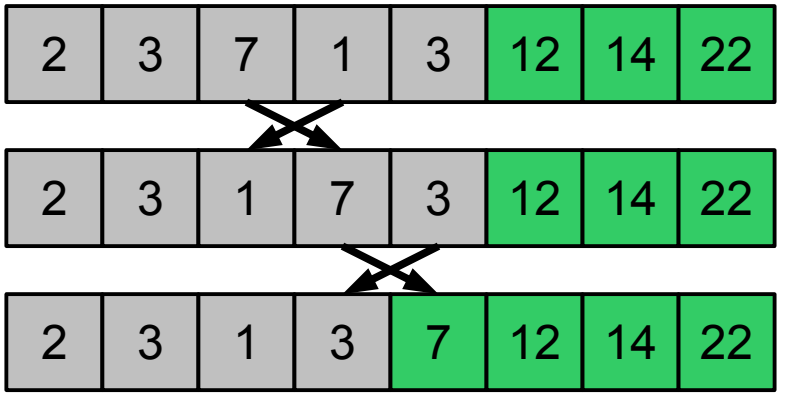
- Esegue una serie di scansioni dell'array
  - Ad ogni scansione scambia le coppie di elementi adiacenti che non sono nell'ordine corretto
  - Se al termine di una scansione non è stato effettuato nessuno scambio, l'array è ordinato
- Dopo la prima scansione, l'elemento massimo occupa l'ultima posizione
- Dopo la seconda scansione, il "secondo massimo" occupa la penultima posizione...
- ...dopo la k-esima scansione, i *k* elementi massimi occupano la posizione corretta in fondo all'array

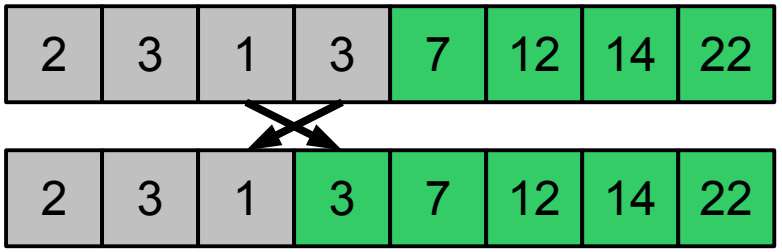
Bubble Sort

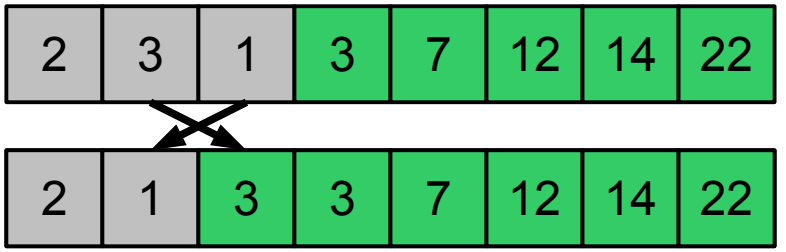

12 7 3 2 14 22 1 3
7 12 3 2 14 22 1 3
7 3 12 2 14 22 1 3
7 3 2 12 14 22 1 3
7 3 2 12 14 1 22 3
7 3 2 12 14 1 3 22
7 3 2 12 14 1 3 22
3 7 2 12 14 1 3 22
3 2 7 12 14 1 3 22
3 2 7 12 1 14 3 22
3 2 7 12 1 3 14 22

Bubble Sort

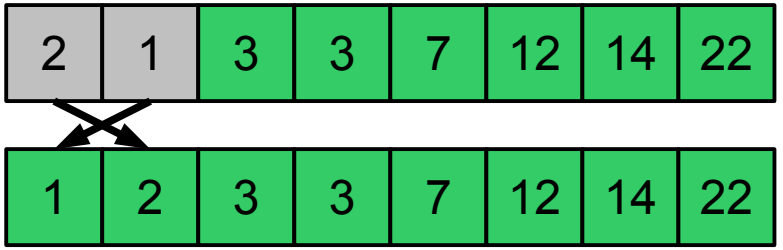

3 2 7 12 1 3 14 22
2 3 7 12 1 3 14 22
2 3 7 1 12 3 14 22
2 3 7 1 3 12 14 22


2 3 7 1 3 12 14 22
2 3 1 7 3 12 14 22
2 3 1 3 7 12 14 22


2 3 1 3 7 12 14 22
2 3 1 3 7 12 14 22


2 3 1 3 7 12 14 22
2 1 3 3 7 12 14 22

Bubble Sort



Bubble sort: prima versione

```
bubbleSort(ITEM[] A, integer n)
  for integer i ← 1 to n do
      for integer j ← 2 to (n-i +1) do
          if A[j-1] > A[j] then
          ITEM temp ← A[j]
          A[j] ← A[j-1]
          A[j-1] ← temp
```

Si può fare di meglio?