Algoritmi e Strutture Dati

Capitolo 4
Ordinamento

Ordinamento

Dato un insieme S di n oggetti presi da un dominio totalmente ordinato, ordinare S

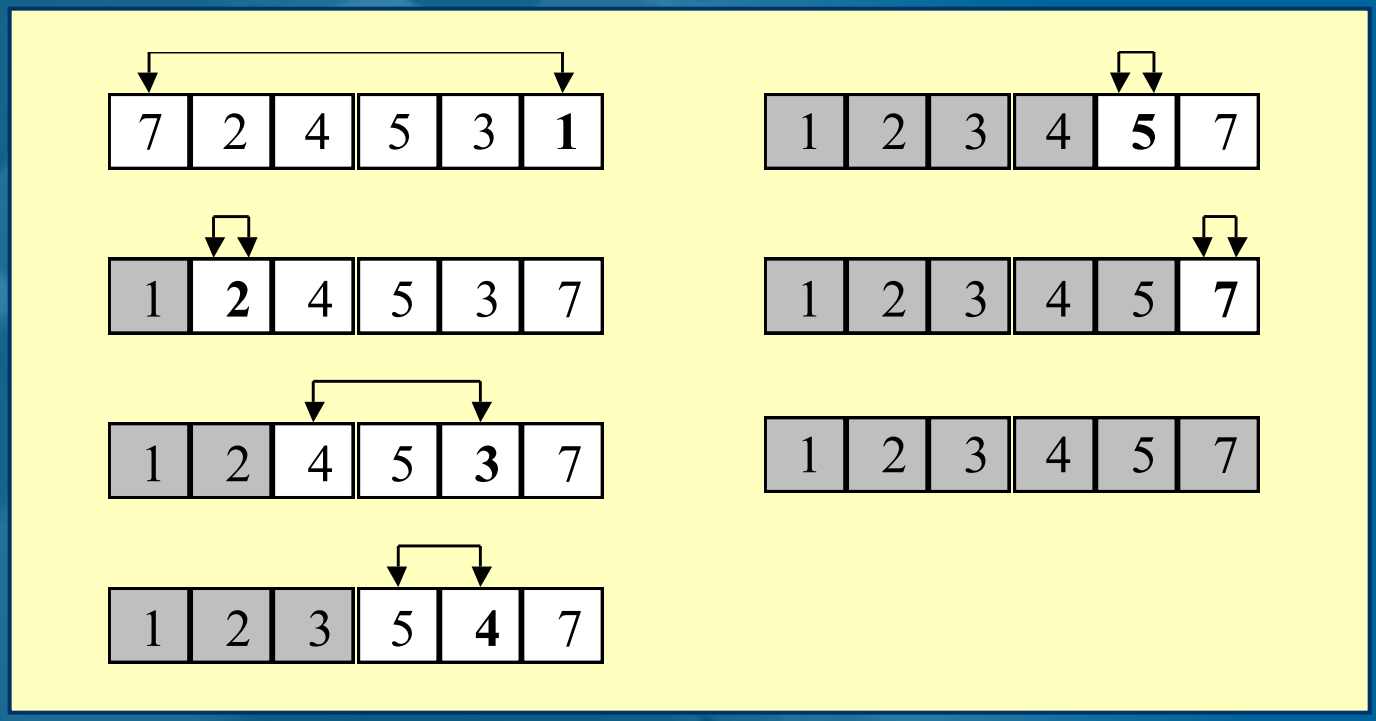
- Esempi: ordinare una lista di nomi alfabeticamente, o un insieme di numeri, o un insieme di compiti d'esame in base al cognome dello studente
- Subroutine in molti problemi
- E' possibile effettuare ricerche in array ordinati in tempo *O(log n)*

Il problema dell'ordinamento

- **Input:** una sequenza di *n* numeri $\langle a_1, a_2, \ldots, a_n \rangle$
- **Output:** una permutazione (riarrangiamento) $\langle a_1', a_2', \ldots, a_n' \rangle$ della sequenza di input tale che $a_1' \leq a_2' \leq \ldots \leq a_n'$

SelectionSort

Approccio incrementale: estende l'ordinamento da k a k+1 elementi, scegliendo il minimo degli n-k elementi non ancora ordinati e mettendolo in posizione k+1



```
SelectionSort (A)
1.    for k=0 to n-2 do
2.        m = k+1
3.        for j=k+2 to n do
4.            if (A[j] < A[m]) then m=j
5.        scambia A[m] con A[k+1]
```

- al generico passo k, A[1],…,A[k] sono già ordinati
- linee 2-4: ricerca del minimo fra gli elementi A[k+1],…,A[n]
- m è l'indice dell'array in cui si trova il minimo
- il minimo è messo in posizione k+1

Corretto?

- E' facile convincersi che l'algoritmo mantiene le seguenti *invarianti*: dopo il generico passo k (k=0,…,*n*-2) abbiamo che:
  - (i) i primi k+1 elementi sono ordinati e
  - (ii) sono i k+1 elementi più piccoli dell'array

Suggerimento: ragionare per invarianti è uno strumento utile per dimostrare la correttezza di un algoritmo, perché permette di isolare proprietà dell'algoritmo, spiegarne il funzionamento, capire a fondo l'idea su cui si basa.

Complessità temporale (analisi)

$T(n) =$ #operazioni elementari sul modello RAM a costi uniformi eseguite dall'algoritmo nel caso peggiore su istanze di dimensione n.

Complessità: un upper bound

```
SelectionSort (A)
1.    for k=0 to n-2 do
2.        m = k+1
3.        for j=k+2 to n do
4.            if (A[j] < A[m]) then m=j
5.        scambia A[m] con A[k+1]
```

eseguite al più n volte per ogni ciclo esterno

ciclo esterno eseguito al più n volte

ogni linea di codice costa tempo O(1)

$T(n) \leq 5\, n^2\, O(1) = \Theta(n^2)$ ⇒ $T(n) = O(n^2)$

L'analisi è stretta? Cioè, T(n) è $\Theta(n^2)$?

Complessità: un lower bound

```
SelectionSort (A)
1.    for k=0 to n-2 do
2.        m = k+1
3.        for j=k+2 to n do
4.            if (A[j] < A[m]) then m=j
5.        scambia A[m] con A[k+1]
```

Idea: conto solo i confronti fra elementi

} n-k-1 confronti

$$T(n) \geq \sum_{k=0}^{n-2} (n-k-1) = \sum_{k=1}^{n-1} k = n(n-1)/2 = \Theta(n^2)$$

$$\Rightarrow \quad T(n) = \Omega(n^2) \quad \Rightarrow \quad \boxed{T(n) = \Theta(n^2)}$$

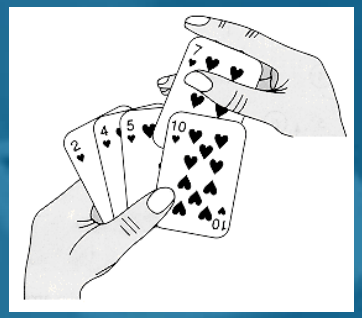
Complessità

```
SelectionSort (A)
1.    for k=0 to n-2 do
2.        m = k+1
3.        for j=k+2 to n do
4.            if (A[j] < A[m]) then m=j
5.        scambia A[m] con A[k+1]
```

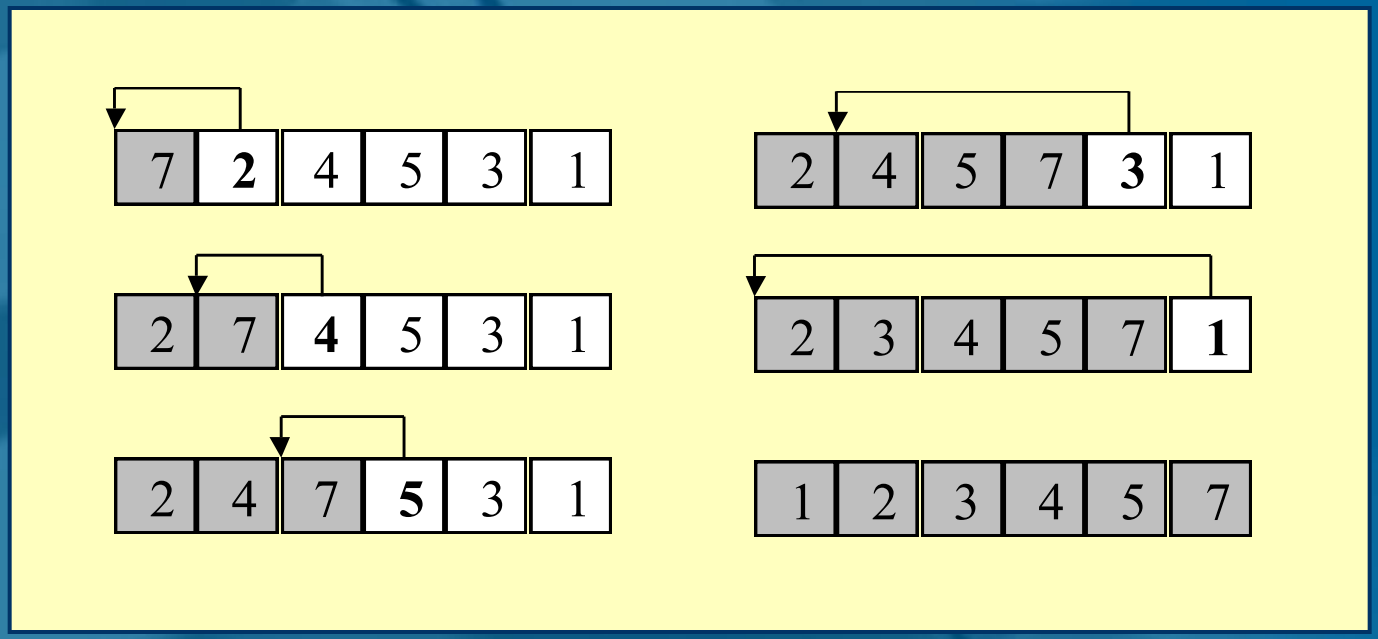
Complessità
nel caso migliore?

$$T_{best}(n) = \Theta(n^2)$$

InsertionSort



Approccio incrementale: estende l'ordinamento da k a k+1 elementi, posizionando l'elemento (k+1)-esimo nella posizione corretta rispetto ai primi k elementi



```
InsertionSort (A)
1.    for k=1 to n-1 do
2.        x = A[k+1]
3.        for j=1 to k+1 do
4.            if (A[j] > x) then break
5.        if (j < k+1) then
6.            for t=k downto j do A[t+1]= A[t]
7.            A[j]=x
```

- al generico passo k, A[1],…,A[k] sono già ordinati
- voglio mettere l'elemento x=A[k+1] nella posizione che gli compete
- righe 3 e 4: individuano la posizione j in cui va messo x
- riga 6: fa spazio per inserire x

Correttezza

- Si dimostra facendo vedere che dopo il generico passo k vale la seguente *invariante*:

  (i) i primi k+1 elementi sono ordinati (fra di loro)

```
InsertionSort (A)
1.    for k=1 to n-1 do
2.        x = A[k+1]
3.        for j=1 to k+1 do
4.            if (A[j] > x) then break
5.        if (j < k+1) then
6.            for t=k downto j do A[t+1]= A[t]
7.            A[j]=x
```

eseguite al più n volte per ogni ciclo esterno

ciclo esterno eseguito al più n volte

ogni linea di codice costa tempo O(1)

$T(n) \leq 7n^2\ O(1) = \Theta(n^2)$ ⇒ $T(n) = O(n^2)$

L'analisi è stretta? Cioè, T(n) è $\Theta(n^2)$?

```
InsertionSort (A)
1.    for k=1 to n-1 do
2.        x = A[k+1]
3.        for j=1 to k+1 do
4.            if (A[j] > x) then break
5.        if (j < k+1) then
6.            for t=k downto j do A[t+1]= A[t]
7.            A[j]=x
```

$$T(n) = \Theta(n^2)$$

se esegue h confronti…

…faccio k+1- h spostamenti

k+1 operazioni

$$T(n) \geq \sum_{k=1}^{n-1} (k+1) = \Theta(n^2)$$

complessità nel caso migliore?

$$T_{best}(n) = \Theta(n^2)$$

```
InsertionSort2 (A)
1.    for k=1 to n-1 do
2.        x = A[k+1]
3.        j = k
4.        while j > 0 e A[j] > x do
5.            A[j+1] = A[j]
6.            j= j-1
7.        A[j+1]=x
```

una variante un po' più furba

- Caso migliore
  - Si verifica quando l'array è già ordinato
  - $T_{best}(n) = \Theta(n)$
- Caso peggiore
  - array ordinato in ordine decrescente
  - $T(n) = \Theta(n^2)$

Esercizio

Dimostrare la correttezza e stimare la complessità temporale nel caso peggiore e nel caso migliore del seguete algoritmo

```
BubbleSort (A)
1.    for i=1 to n-1 do
2.        scambi= false
3.        for j=2 to n-i+1 do
4.            if (A[j-1] > A[j]) then scambia A[j-1] e A[j]; scambi=true
5.        if (scambi=false) then break
```