

**INPUT:** SEQUENZA  $\langle n \text{ NUMERI} \rangle$   $\langle a_1, a_2, a_3, \dots \rangle$   
**OUTPUT:** PERMUTAZIONE tale che  $\exists i \quad a_1 \leq a_2 \leq a_3 \dots$

### SELECTION SORT

#### TECNICA GREEDY

- SELEZIONA DALLA SEQUENZA IL MINIMO E LO POSIZIONA  
ALTERNATIVA



LOCALMENTE OTTIMA  
GLOBALMENTE NON SEMPRE OTTIMA => ET. COMMESSO \* VIAGGIATORE \*

### SELECTION SORT USA LA TECNICA GREEDY

$38471326 \rightarrow 18475326 \rightarrow 12475386 \dots$

- SELEZIONA L'ELEMENTO PIÙ PICCOLO E LO POSIZIONA

```
SELECTION_SORT(A)
1. for i = 0 to A.length-2
2.   min = i      // indice elemento minimo in A[i..n-1]
3.   for j = i + 1 to A.length-1 // scorri l'array
4.     if A[j] < A[min]        // devo aggiornare min
5.       min = j
6.   temp = A[i]           // scambio A[i] con A[min]
7.   A[i] = A[min]
8.   A[min] = temp
```

COMPLESSITÀ NON VARIÀ NEI  
CAST MEDI, MIGLIORE E PEGGIORE  
I VALORI DI INPUT NON  
MODIFICANO I CICLI

-  $O(n)$  CICLI INTERNI  $\times O(1)$  ESTERNI  $\Rightarrow O(n)$

- PER ESEGUIRSI  $(n-1) + (n-2) + \dots + 1 = \frac{(n-1)n}{2}$  VOLTE  $\Rightarrow O(n^2)$

### OPERARE IN LOCO

- ALGORITMI CHE NON NECESSITANO DI COPIARE L'INPUT IN  
STRUTTURE DATI DIVERSE → UTILE PER GRANDI STRUTTURE

### SELECTION SORT OPERA IN LOCO

- ELEMENTI SOLO SCAMBIAI

- NECESSITA SOLO DI UNA MEMORIA COSTANTE OLTRE A QUELLA PER  
MEMORIZZARE A.  $\Rightarrow$  MEMORIA UTILIZZATA  $O(1)$

### ORDINAMENTI STABILI

- ALGORITMO CHE NON MODIFICA L'ORDINE DI

ELEMENTI CON STESSO VALORE → UTILE SE: ?

1) ELEMENTI COLLEGATI A  
DATI SATELLITE

2) ELEMENTI CON LA STESSA CHIAVE  
HANNO UNA POSIZIONE RECIPROCA  
SIGNIFICATIVA

### SELECTION SORT È STABILE

- Se  $A(i) = A(k)$  con  $i < k$

$A(j)$  viene selezionato prima

### INSERTION SORT

#### ALGORITMI INCREMENTALI

LA SOLUZIONE DI UNA ISTANZA DI  
DIMENSIONE  $(m-1)$  PUÒ RISOLVERNE UNA  $\Rightarrow$  b) di dimensione  $(n)$

a) Istanza  $n=5 \Rightarrow [5|2|4|6|1]$

SOLUZIONE  $\underline{\text{a}}$  AIUTA A RIDURRE b)  
 $12456 + [3]$

### INSERTION SORT

- PARTENDO DA UN SOTTOINSIEME ORDINATO DELL'INTERO  $(n)$ .

VIENE AGGIUNTO A DENI PASTAGGIO UN ELEMENTO  $\leftarrow$   
TRASLATI QUELLI GIÀ PRESENTI

L'ORDINE

```
INSERTION_SORT(A)
1. for i = 1 to A.length-1
2.   key = A[i]
3.   i = i - 1
4.   while i >= 0 and A[i] > key
5.     A[i+1] = A[i]
6.     i = i - 1
7.   A[i+1] = key
```



COMPLESSITÀ CASO: MIGLIORE

- CICLO ESTERNO  $O(n)$  VOLTE  
- ELEMENTO CORRENTE VA IN  
POSIZIONE GIUSTA  
(non è già ordinato)  
-  $\Theta(n)$

### PROPRIETÀ INSERTION SORT

• STABILE

• IN LOCO

• ONLINE → Può essere utilizzato quando i numeri arrivano uno alla volta

• EFFICIENTE SU PICCOLE ISTANZE

- Più efficiente di selection sort grazie a caso migliore con complessità lineare

- Si può calcolare la complessità  $\Theta(n^2)$

• ADATTIVO

- Veloce su istanze già ordinate

MIGLIORE

- CICLO ESTERNO  $O(n)$  VOLTE  
- ELEMENTO CORRENTE VA  
POSIZIONATO NEL MEZZO DI  
 $A[0, \dots, n]$   
-  $\Theta(n^2)$

PEGGIORE

CICLO ESTERNO  $O(n)$  VOLTE  
- ELEMENTO CORRENTE VA  
SEMPRE IN PRIMA POSIZIONE  
- A è DECRESCENTE  
-  $\Theta(n^2)$

Algoritmi di ordinamento visti finora

	SELECTION-SORT	INSERTION-SORT	QUICK-SORT	HEAP-SORT	MERGE-SORT
Stabile	Si	Si	No	No	Si
In loco	Si	Si	Si	Si	Si
Online	Si	Si	Si	Si	Si
Efficiente su piccole istanze	Si	Si	Si	Si	Si
Adattivo	Si	Si	Si	Si	Si
Veloce su istanze già ordinate	Si	Si	Si	Si	Si