ALGORITMI E STRUTTURE DATI

Dr. Manuela Montangero

A.A. 2022/23

ALGORITMI DI ORDINAMENTO: InsertitonSort

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

InsertionSort

Ispiriamoci ai giocatori di carte!

TERZA IDEA:

- Prendiamo le carte una alla volta
- Per ogni carta nuova, confrontiamo il suo numero con quelle già in mano, da destra verso sinistra e la mettiamo al "posto giusto"
- Le carte che abbiamo in mano sono sempre ordinate
- Ci fermiamo quando abbiamo finito le carte

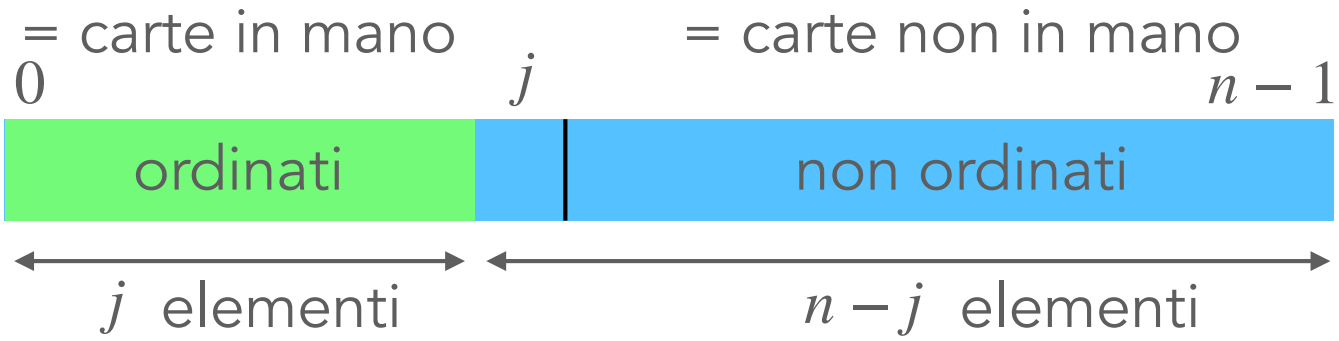
Come trasformiamo questa idea
in un algoritmo?

Abbiamo una serie di iterazioni e, in ogni iterazione,
mettiamo un numero al posto giusto

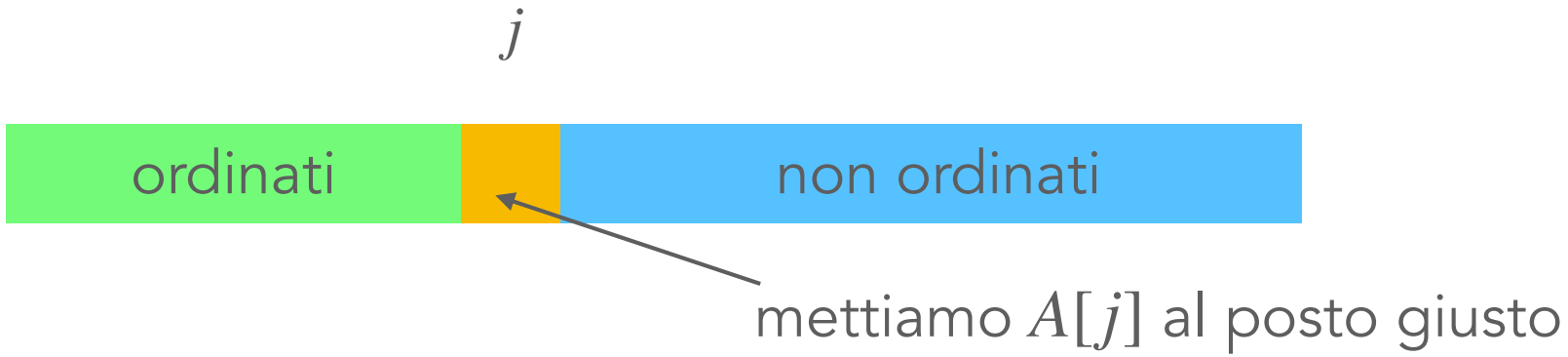
InsertionSort

Iterazione generica j

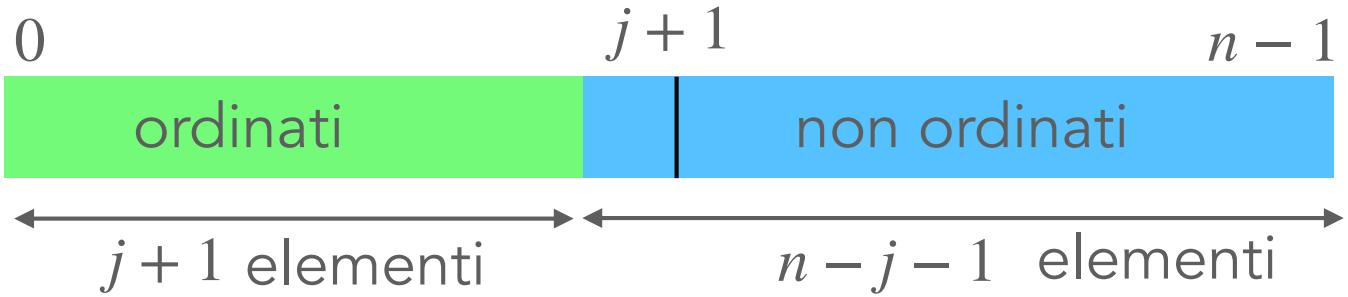
All'inizio della iterazione



Cosa facciamo nella iterazione

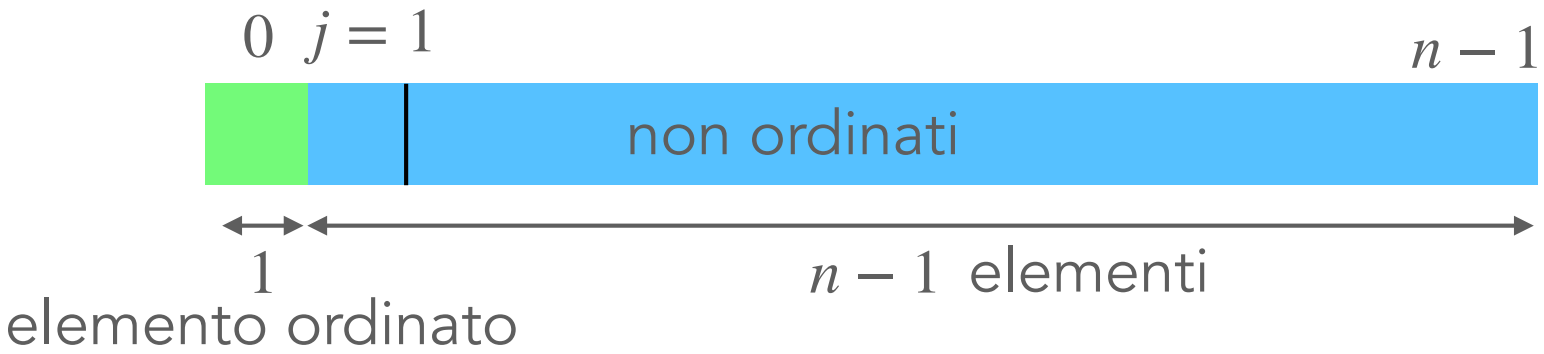


Alla fine della iterazione

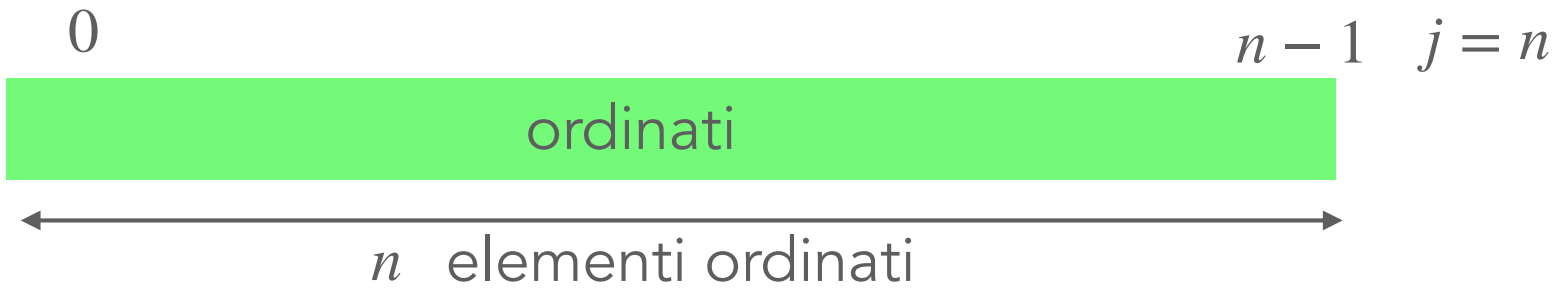


InsertionSort

Prima della prima iterazione



Dopo l'ultima iterazione



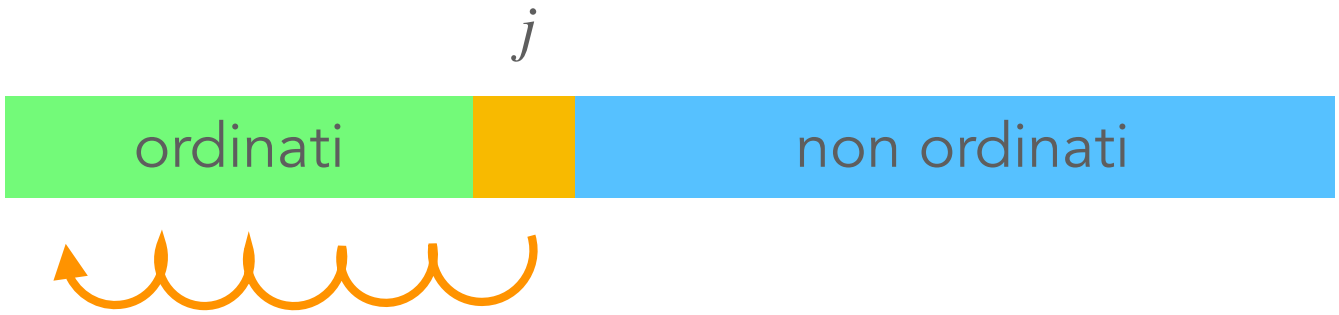
InsertionSort

Iterazione generica j

Cosa facciamo nella iterazione

mettiamo $A[j]$ al posto giusto

COME?



Confrontiamo $A[j]$ con i valori già ordinati, da dx verso sx

Ci fermiamo quando troviamo il suo posto

o

quando arriviamo all'inizio dell'array

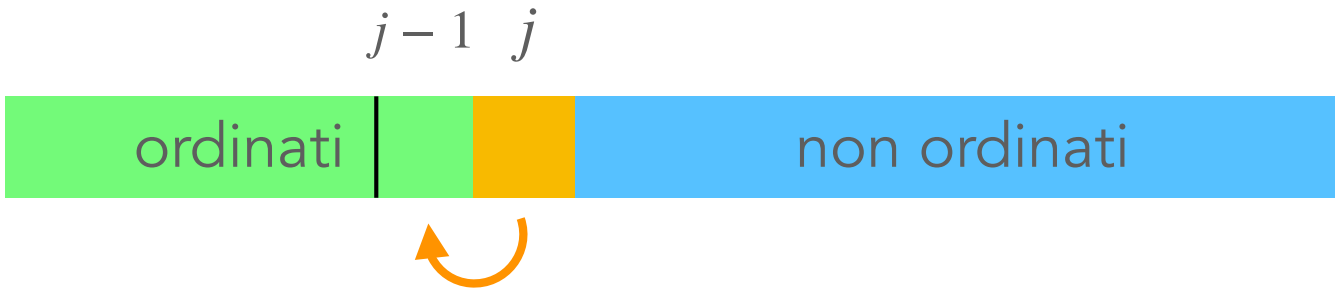
InsertionSort

Iterazione generica j

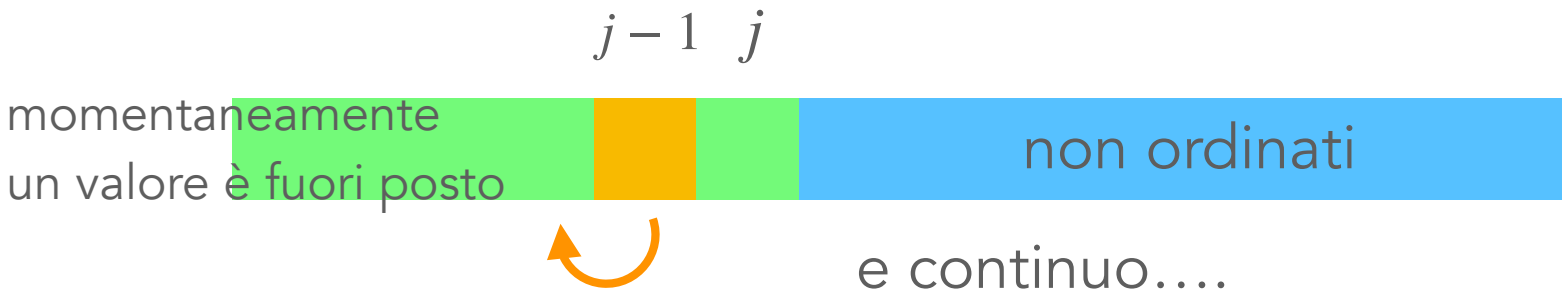
Cosa facciamo nella iterazione

mettiamo $A[j]$ al posto giusto

COME?



Se $A[j-1] > A[j]$, allora inverto i due valori



e continuo….

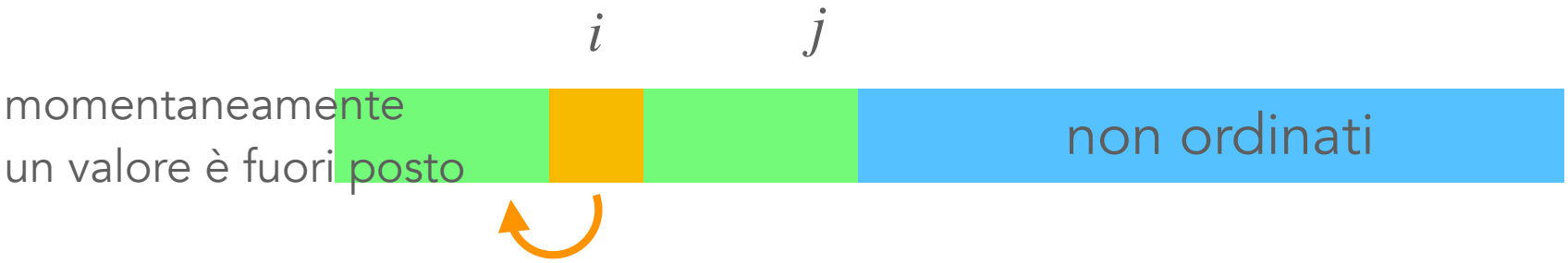
InsertionSort

Iterazione generica j

Cosa facciamo nella iterazione

mettiamo $A[j]$ al posto giusto

COME?



Se $A[i-1] \leq A[i]$, allora mi fermo

perché l'elemento che era "partito" dalla posizione j,
è al posto giusto nella sequenza ordinata di sinistra

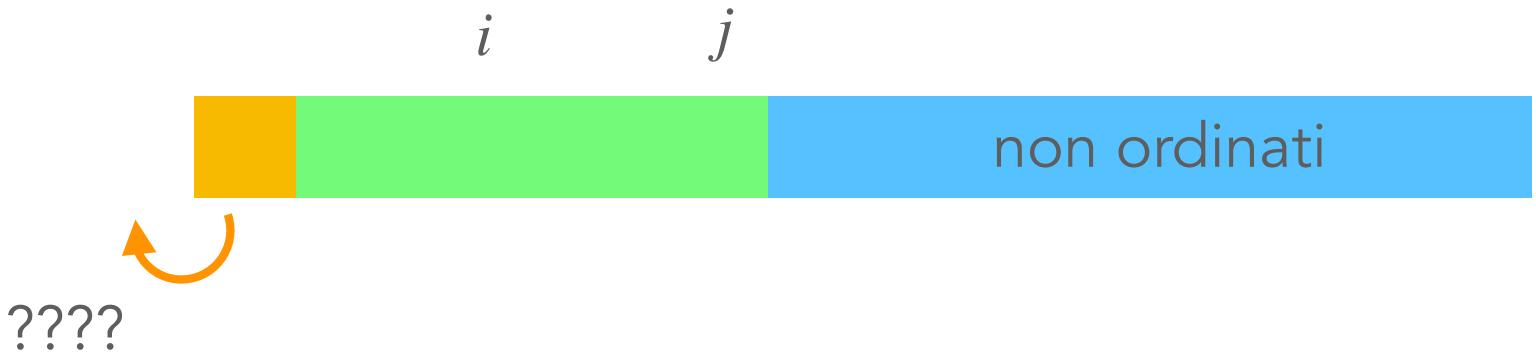
InsertionSort

Iterazione generica j

Cosa facciamo nella iterazione

mettiamo $A[j]$ al posto giusto

COME?



perché l'elemento che era "partito" dalla posizione j,
è arrivato in prima posizione, allora è al posto giusto,
e ci fermiamo

InsertionSort

Scriviamo lo pseudo-codice

```
InsertionSort(A,n)
```

non ci aspettiamo un risultato esplicito,
ma che alla fine l'array `A` sia ordinato

InsertionSort

Scriviamo lo pseudo-codice

OSSERVAZIONE:

- ci saranno due cicli annidati
- il ciclo esterno deve considerare tutti gli elementi dell'array (escluso il primo) per metterli a posto
- il ciclo interno deve mettere a posto l'elemento dell'iterazione corrente, confrontandolo con gli elementi che stanno alla sua sinistra

InsertionSort

Scriviamo lo pseudo-codice

- il ciclo esterno deve considerare tutti gli elementi dell'array (escluso il primo) per metterli a posto

```
InsertionSort(A,n)
for j = 1 to n-1
```

l'elemento in seconda posizione ha indice 1, l'ultimo indice n-1

InsertionSort

Scriviamo lo pseudo-codice

- il ciclo interno deve mettere a posto l'elemento dell'iterazione corrente, confrontandolo con gli elementi che stanno alla sua sinistra

```
InsertionSort(A,n)
for j = 1 to n-1
 i := j-1
 tmp := A[j]
```

il primo confronto deve essere fatto con `A[j-1]`

salviamo il valore da mettere a posto in questa iterazione in `tmp`

InsertionSort

Scriviamo lo pseudo-codice

- il ciclo interno deve mettere a posto l'elemento dell'iterazione corrente, confrontandolo con gli elementi che stanno alla sua sinistra

```
InsertionSort(A,n)
for j = 1 to n-1
 i := j-1
 tmp := A[j]
 while
```

usiamo un ciclo **while** e non un ciclo **for**, perche'?

quando esco dal **while**?

InsertionSort

Scriviamo lo pseudo-codice

- il ciclo interno deve mettere a posto l'elemento dell'iterazione corrente, confrontandolo con gli elementi che stanno alla sua sinistra

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
```

se la condizione del **while** e' **vera**, dobbiamo fare posto per `tmp` in `A[i]`

InsertionSort

Scriviamo lo pseudo-codice

- il ciclo interno deve mettere a posto l'elemento dell'iterazione corrente, confrontandolo con gli elementi che stanno alla sua sinistra

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
```

se la condizione del **while** e' **vera**, dobbiamo fare posto per `tmp` in `A[i]`

non dimentichiamo di decrementare `i`

InsertionSort

Scriviamo lo pseudo-codice

- il ciclo interno deve mettere a posto l'elemento dell'iterazione corrente, confrontandolo con gli elementi che stanno alla sua sinistra

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
```

se la condizione del **while** e' **falsa**,

- o `i = -1`
- o `A[i] <= tmp`

(o entrambe)

In entrambi i casi, la posizione di `tmp` e' in `i+1`

InsertionSort

Scriviamo lo pseudo-codice

- il ciclo interno deve mettere a posto l'elemento dell'iterazione corrente, confrontandolo con gli elementi che stanno alla sua sinistra

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

se la condizione del **while** e' **falsa**,

- o `i = -1`
- o `A[i] <= tmp`

(o entrambe)

In entrambi i casi, la posizione di `tmp` e' in `i+1`

InsertionSort

ESEMPIO

A = <36,14,27,40,31>


```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Prima iterazione ciclo **for** j = 1, i = 0, tmp = A[1] = **14**

Prima iterazione ciclo **while** i = 0 e A[0] = 36 > 14

A[1] := A[0] A = <36,**36**,27,40,31>

i := 0-1 = -1

Seconda iterazione ciclo **while** condizione falsa A[0] := 14

A = <**14**,**36**,27,40,31>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>


```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Prima iterazione ciclo **for** j = 1, i = 0, tmp = A[1] = **14**

1 CONFRONTO

Prima iterazione ciclo **while** i = 0 e A[0] = 36 > 14

A[1] := A[0] A = <36,**36**,27,40,31>

i := 0-1 = -1

Seconda iterazione ciclo **while** condizione falsa A[0] := 14

A = <**14**,**36**,27,40,31>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>

A = <**14**,**36**,27,40,31>

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Seconda iterazione ciclo **for** j = **2**, i = **1**, tmp = A[2] = **27**

Prima iterazione ciclo **while** i = 1 e A[1] = 36 > 27

A[2] := A[1] A = <**14**,**36**,**36**,40,31> i := 1-1 = 0

Seconda iterazione ciclo **while** i = 0 e A[0] = 14 < 27

condizione falsa

A[1] := 27 A = <**14**,**27**,**36**,40,31>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>

A = <**14**,**36**,27,40,31>

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Seconda iterazione ciclo **for** j = **2**, i = **1**, tmp = A[2] = **27**

Prima iterazione ciclo **while** i = 1 e A[1] = 36 > 27 **2 CONFRONTI**

A[2] := A[1] A = <**14**,**36**,**36**,40,31> i := 1-1 = 0

Seconda iterazione ciclo **while** i = 0 e A[0] = 14 < 27

condizione falsa

A[1] := 27 A = <**14**,**27**,**36**,40,31>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>

A = <**14**,**36**,27,40,31>

A = <**14**,**27**,**36**,40,31>

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Terza iterazione ciclo **for** j = **3**, i = **2**, tmp = A[3] = **40**

Prima iterazione ciclo **while** i = 2 e A[2] = 36 < 40

condizione falsa

A[3] := 40 A = <**14**,**27**,**36**,**40**,31>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>

A = <**14**,**36**,27,40,31>

A = <**14**,**27**,**36**,40,31>

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Terza iterazione ciclo **for** j = **3**, i = **2**, tmp = A[3] = **40**

Prima iterazione ciclo **while** i = 2 e A[2] = 36 < 40 **1 CONFRONTO**

condizione falsa

A[3] := 40 A = <**14**,**27**,**36**,**40**,31>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>

A = <**14**,**36**,27,40,31>

A = <**14**,**27**,**36**,40,31>

A = <**14**,**27**,**36**,**40**,31>

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Quarta iterazione ciclo **for** j = **4**, i = **3**, tmp = A[4] = **31**

Prima iterazione ciclo **while** i = 3 e A[3] = 40 > 31

A[4] := A[3] A = <**14**,**27**,**36**,**40**,**40**> i := 3-1 = 2

Seconda iterazione ciclo **while** i = 2 e A[2] = 36 > 31

A = <**14**,**27**,**36**,**36**,**40**>

Terza iterazione ciclo **while** condizione falsa A = <**14**,**27**,**31**,**36**,**40**>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>

A = <**14**,**36**,27,40,31>

A = <**14**,**27**,**36**,40,31>

A = <**14**,**27**,**36**,**40**,31>

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Quarta iterazione ciclo **for** j = **4**, i = **3**, tmp = A[4] = **31**

Prima iterazione ciclo **while** i = 3 e A[3] = 40 > 31 **3 CONFRONTI**

A[4] := A[3] A = <**14**,**27**,**36**,**40**,**40**> i := 3-1 = 2

Seconda iterazione ciclo **while** i = 2 e A[2] = 36 > 31

A = <**14**,**27**,**36**,**36**,**40**>

Terza iterazione ciclo **while** condizione falsa A = <**14**,**27**,**31**,**36**,**40**>

InsertionSort

ESEMPIO

A = <36,14,27,40,31>

A = <**14**,**36**,27,40,31>

A = <**14**,**27**,**36**,40,31>

A = <**14**,**27**,**36**,**40**,31>

A = <**14**,**27**,**31**,**36**,**40**>

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Totale numero di confronti: 1+2+1+3 = 7

E in generale?

InsertionSort

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Quanti confronti tra elementi di A?

- Un confronto per ogni volta che si esegue testa la condizione del ciclo **while**
- Il ciclo esterno viene eseguito per *j* da *1* a *n-1* (estremi inclusi)
- Fissato *j*, il ciclo while viene eseguito **AL PIÙ** *j* volte

tmp viene confrontato **nel caso peggiore** con tutti i valori nelle posizioni da *j-1* a *0* (estremi inclusi) -> *(j-1) - 0 + 1 = j* volte.
Ci possono essere casi in cui ci si ferma prima della posizione 0.

InsertionSort

```
InsertionSort(A,n)
for j = 1 to n-1 do
 i := j-1
 tmp := A[j]
 while (i≥0 AND A[i]>tmp) do
  A[i+1] := A[i]
  i := i-1
 A[i+1] := tmp
```

Quanti confronti tra elementi di A?

- Un confronto per ogni volta che si esegue testa la condizione del ciclo **while**
- Il ciclo esterno viene eseguito per *j* da *1* a *n-1* (estremi inclusi)
- Fissato *j*, il ciclo while viene eseguito **AL PIÙ** *j* volte

$$T(n) \leq \sum_{j=1}^{n-1} 1 = \frac{(n-1)n}{2} \Rightarrow T(n) \in O(n^2)$$

UPPER BOUND
una funzione quadratica (rispetto a n) di confronti è **SUFFICIENTE** per risolvere il problema anche nel caso peggiore

InsertionSort

- Il ciclo esterno viene eseguito per *j* da *1* a *n-1* (estremi inclusi)
- Fissato *j*, il ciclo while viene eseguito **AL PIÙ** *j* volte

Esiste un'istanza che richiede di eseguire il ciclo while **almeno** *j* volte, per ogni *j* da *1* a *n-1*?

Ovvero: esiste un'istanza per cui il valore in posizione j viene SEMPRE spostato in posizione 0 ad ogni iterazione?

SI: quando la sequenza in input ha valori tutti distinti tra di loro ed è ordinata in ordine decrescente (provare con un esempio)

$$T(n) \geq \sum_{j=1}^{n-1} 1 = \frac{(n-1)n}{2} \Rightarrow T(n) \in \Omega(n^2)$$

LOWER BOUND
esiste una istanza per cui una funzione quadratica (rispetto a n) di confronti è **NECESSARIA** per risolvere il problema.

Il caso peggiore non può richiedere meno confronti di questa istanza
(l'istanza potrebbe essere il caso peggiore, ma non è necessario per determinare un lower bound)

InsertionSort

Quanti confronti tra elementi di `A`?

$T(n) \in O(n^2)$

$T(n) \in \Omega(n^2)$

$\Rightarrow T(n) \in \Theta(n^2)$

Nel caso peggiore non abbiamo guadagnato
rispetto al SelectionSort,
ma con InsertionSort ci possono essere istanze
che non hanno bisogno di un numero quadratico di confronti.

ESEMPIO: se la sequenza in input è già ordinata bastano *n-1* confronti