

Ordinamenti e grafi

Vittorio Maniezzo - Università di Bologna

1

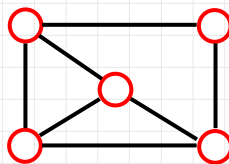
Grafo: definizione

Un **grafo** $G = (V, E)$ è composto da:

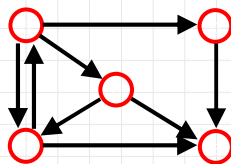
- V : insieme di **vertici**
- $E \subset V' \times V$: insieme di **archi** (*edge*) che connettono i **vertici**

Un **arco** $a = \{u, v\}$ è una coppia di vertici

Se (u, v) è ordinato allora G è un **grafo diretto (orientato)**



$V = \{a, b, c, d, e\}$
 $E = \{\{a, b\}, \{a, c\}, \{a, d\},$
 $\{b, e\}, \{c, d\},$
 $\{c, e\}, \{d, e\}\}$



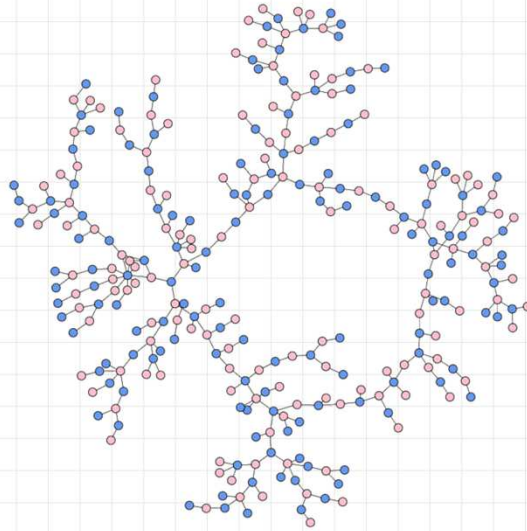
$V = \{a, b, c, d, e\}$
 $A = \{(a, b), (a, c), (a, d),$
 $(d, a), (b, e), (c, d),$
 $(c, e), (d, e)\}$

Vittorio Maniezzo - Università di Bologna

2

2

Flessibilità rappresentativa ...



Vittorio Maniezzo - Università di Bologna

3

3

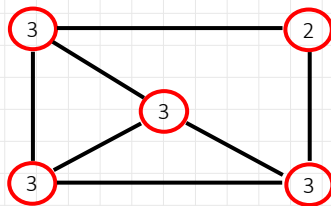
Terminologia

Vertici adiacenti: connessi da un arco

Grafo non orientato: u, v adiacenti se $\{u, v\} \in E$

Grafo orientato: se $(u, v) \in A \rightarrow v$ adiacente a u e (u, v) è incidente in v

Grado (di un **vertice**): num. di vertici adiacenti



$$\sum_{v \in V} \deg(v) = 2(\text{num di archi})$$

Cammino: sequenza di vertici v_1, v_2, \dots, v_k tale per cui ogni coppia di vertici consecutivi v_i e v_{i+1} è adiacente (in grafo orientato: *catena*)

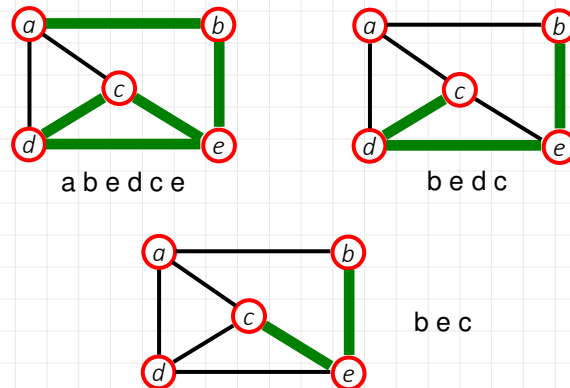
Vittorio Maniezzo - Università di Bologna

4

4

Terminologia (2)

Cammino elementare: non ci sono vertici ripetuti



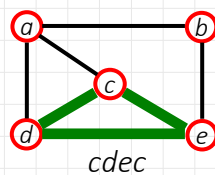
Vittorio Maniezzo - Università di Bologna

5

5

Terminologia (3)

Ciclo: cammino elementare, tranne che per il primo vertice che coincide con l'ultimo (in grafo orientato: *circuito*)



Grafo connesso: qualsiasi coppia di vertici è unita da almeno un cammino



6

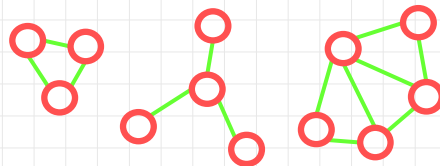
5

Terminologia (4)

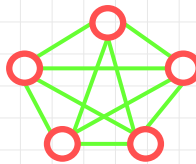
Sottografo: sottinsieme di vertici e archi di un grafo dato

Componente connessa: sottografo connesso massimale.

Ad es., il grafo sotto ha 3 componenti connesse



Grafo completo: ha un arco fra ogni coppia di nodi



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

Vittorio Maniezzo - Università di Bologna

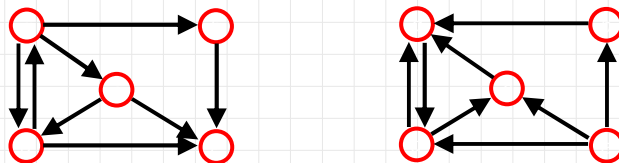
7

7

Grafo trasposto

Il grafo $G^T=(V,E^T)$ è il **trasposto** di $G=(V,E)$ se $E^T = \{(u,v): (v,u) \in E\}$ (rovescia il senso di percorrenza degli archi di G).

G e G^T hanno le stesse componenti fortemente connesse.



Vittorio Maniezzo - Università di Bologna

8

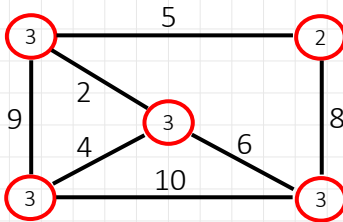
Terminologia (5)

Grafi pesati:

Gli archi possono avere un peso (costo, guadagno) associato.

Il peso può essere determinato da una funzione $p: V \times V \rightarrow \mathbb{R}$

Se non è specificato un peso su un arco, lo si assume infinito



Vittorio Maniezzo - Università di Bologna

9

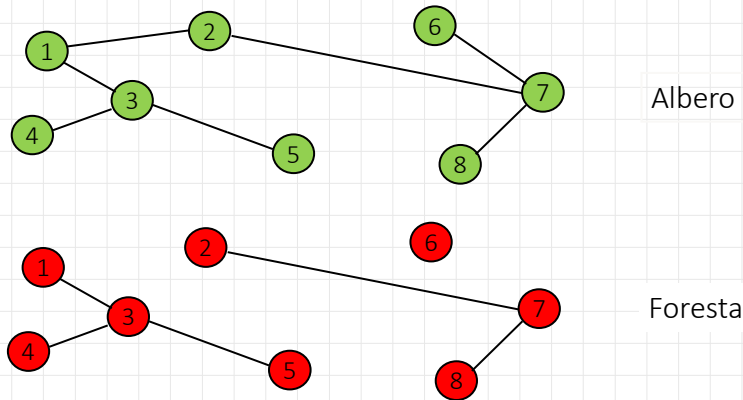
9

Alberi

Un **albero** è un grafo aciclico con un numero di nodi uguale al numero di archi più uno ($|E| = |V| - 1$).

Equiv., un albero è un grafo aciclico totalmente connesso.

Può esistere un nodo particolare chiamato **radice**.



Vittorio Maniezzo - Università di Bologna

10

10

Alberi, definizioni

Se c'è un nodo radice viene implicitamente definita una **relazione di ordinamento fra nodi** basata sulla distanza dalla radice in cui:

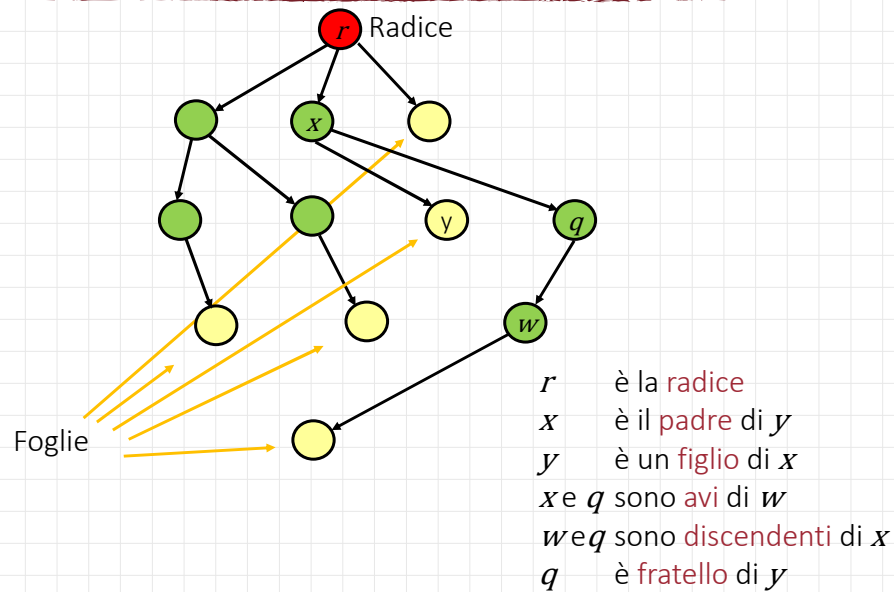
- Ogni nodo non radice ha un unico **predecessore** / **padre**
- Un nodo può avere più **successori** / **discendenti** / **figli**
- Un nodo senza successori è una **foglia**
- Un nodo con successori è un **nodo interno**
- La **profondità di un nodo** è la lunghezza di un cammino dalla radice al nodo
- L'**altezza di un nodo** è la massima lunghezza di un cammino dal nodo a una foglia discendente del nodo
- L'**altezza di un albero** è l'altezza della radice
- Un **livello** / **strato** di un albero consiste di tutti i nodi con la stessa profondità
- Il **numero di livelli** di un albero è pari alla sua **altezza + 1**

Vittorio Maniezzo - Università di Bologna

11

11

Arborescenza



Vittorio Maniezzo - Università di Bologna

12

12

Problema: Ordinamento

Input: una sequenza di n numeri $\langle a_1, a_2, \dots, a_n \rangle$

Output: i numeri ricevuti in input ordinati dal più piccolo al più grande ovvero:

$$\langle a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)} \rangle \text{ dove } a_{\pi(i)} \leq a_{\pi(i+1)}$$

π è una opportuna permutazione degli indici $1, \dots, n$

Esempio: $a = [7, 32, 88, 21, 92, -4]$

$\pi = [6, 1, 4, 2, 3, 5]$

$a[\pi[]] = [-4, 7, 21, 32, 88, 92]$

Vittorio Maniezzo - Università di Bologna

13

13

Ordinamento di chiavi

Più in generale, spesso è dato un array di n elementi, dove ciascun elemento è composto da:

- una **chiave**, con le chiavi confrontabili tra loro
- un **contenuto** (valore), arbitrario

Vogliamo permutare l'array delle chiavi in modo che esse compaiano in ordine non decrescente (oppure non crescente)

Esempio:

Chiave	Valore
4	quattro
2	due
5	cinque
1	uno
3	tre



Chiave	Valore
1	uno
2	due
3	tre
4	quattro
5	cinque

Sarà una semplice estensione degli algoritmi affrontati.

Vittorio Maniezzo - Università di Bologna

14

14

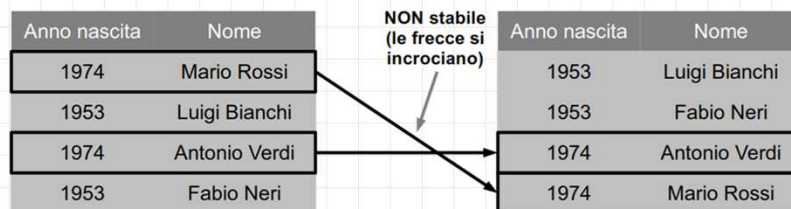
Definizioni

Ordinamento in place (*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



Vittorio Maniezzo - Università di Bologna

15

15

Limite inferiore di complessità

Molti algoritmi di ordinamento, noi vedremo:

Insertion-sort
Merge-sort
Heap-sort
Quick-sort

Tutti “*comparison-sort*”:
algoritmi basati su confronti
di coppie di elementi

Questi metodi calcolano una soluzione che dipende esclusivamente dall'esito di confronti fra numeri

TEOREMA (Lower Bound per algoritmi *comparison-sort*):

Qualsiasi algoritmo “*comparison-sort*” deve effettuare nel caso pessimo $\Omega(n \log(n))$ confronti per ordinare una sequenza di n numeri.

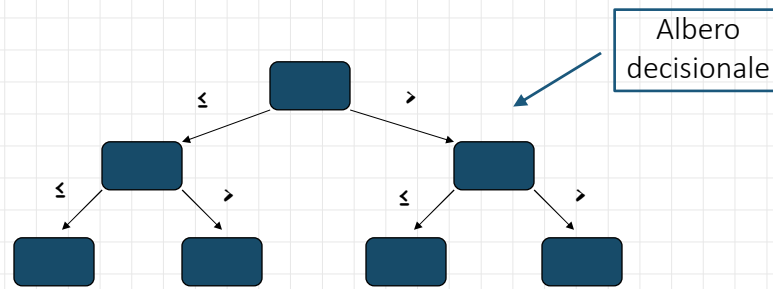
Vittorio Maniezzo - Università di Bologna

16

16

lower bound per comparison sort

Dimostrazione (idea): con n numeri ho $n!$ possibili ordinamenti. Possiamo scegliere quello giusto tramite una sequenza di confronti fra coppie di elementi.



Ogni nodo rappresenta un confronto
Ogni arco porta alla decisione successiva da prendere

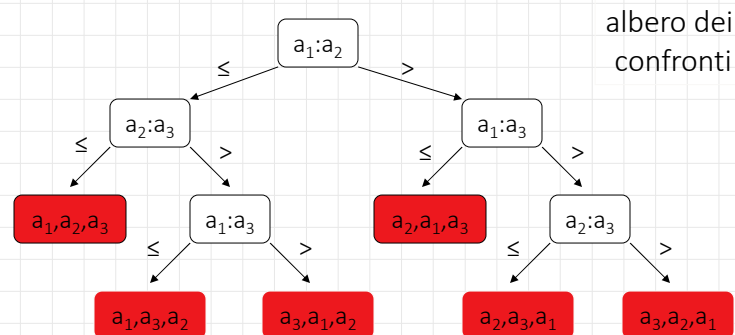
Vittorio Maniezzo - Università di Bologna

17

17

Esempio

$n=3 \quad \{a_1, a_2, a_3\}$



Ogni nodo bianco rappresenta un confronto.
Ogni nodo rosso rappresenta una possibile soluzione.

Vittorio Maniezzo - Università di Bologna

18

18

Limite inf al num dei confronti

Nell'esempio, 3 elementi.

$3! = 6$: numero di foglie dell'albero dei confronti.

- ogni (cammino dalla radice ad una) foglia rappresenta un ordinamento
- ci sono $n!$ ordinamenti.

quanto deve essere alto un albero per avere $n!$ foglie ?

- un albero binario alto h ha al massimo 2^h foglie
- dobbiamo avere $2^h \geq n!$
- Formula di Stirling: $n! > (n/e)^n$ $e=2.718...$

$$h \geq \log[(n/e)^n] = n \log(n) - n \log(e) = \Omega(n \log(n))$$

Limite inf al num dei confronti

Il caso pessimo di un qualsiasi algoritmo comparison-sort eseguito su una sequenza di n numeri è dato dall'altezza dell'albero di decisione associato a quell'algoritmo.

MA

Un albero binario con $n!$ foglie (ordinamenti) ha un'altezza $\Omega(n \log(n))$

QUINDI

qualsiasi algoritmo comparison-sort, nel caso pessimo, esegue $\Omega(n \log(n))$ confronti.



Dimostrazione alternativa

$$2^h \geq n!$$

$$h \geq \log(n!)$$

$$= \log(1 \cdot 2 \cdot \dots \cdot n/2 \cdot \dots \cdot n)$$

$$\geq \log(1 \cdot 1 \cdot \dots \cdot 1 \cdot n/2 \cdot \dots \cdot n/2)$$

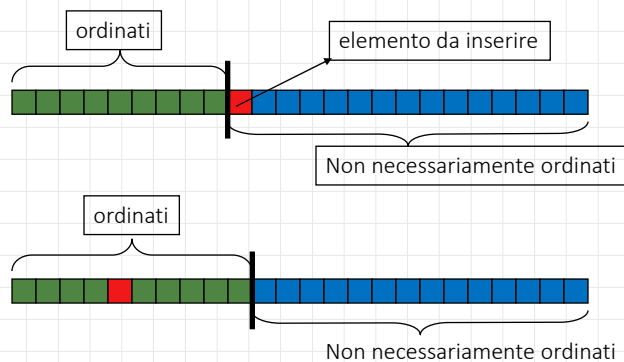
$$= \log((n/2)^{n/2})$$

$$= n/2 \log(n/2)$$

quindi $h \in \Omega(n \log n)$

Idea per ordinare...

Ad ogni passo ho una sottosequenza ordinata in cui inserisco un nuovo elemento dell'input:



Insertion Sort

Insertion-sort (A)

```
1. for j=2 to size(A)
2.   do key = A[j]
3.     // insert A[j] in A[1,...,j-1]
4.     i = j-1
5.     while i>0 and A[i]>key
6.       do A[i+1] = A[i]
7.       i=i-1
8.     A[i+1] = key
```

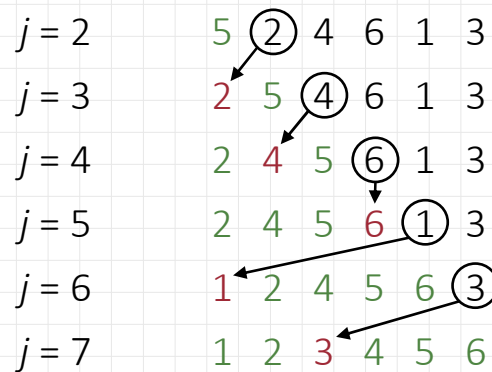
Vittorio Maniezzo - Università di Bologna

23

23

Esempio:

$A = \{5, 2, 4, 6, 1, 3\}$

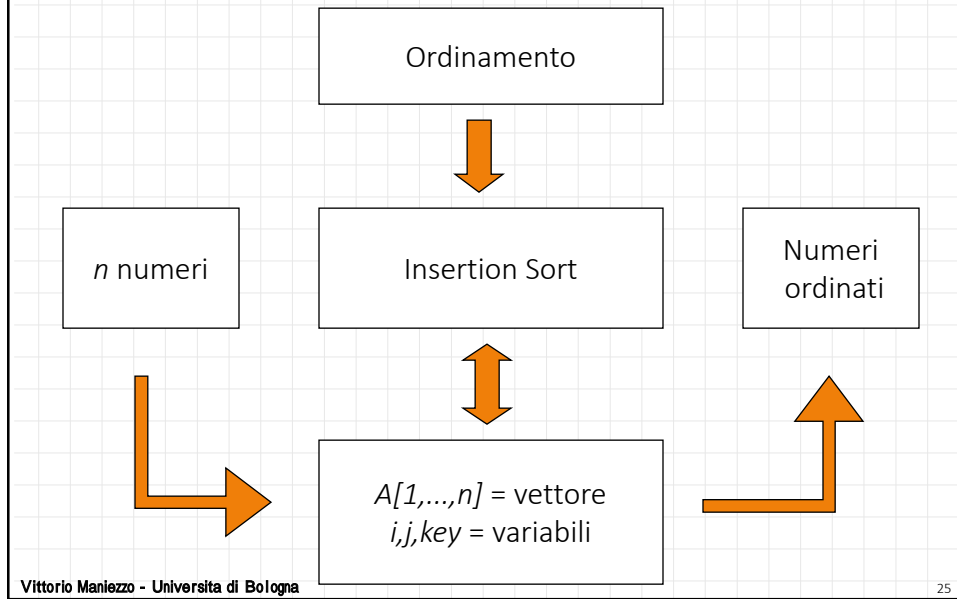


Vittorio Maniezzo - Università di Bologna

24

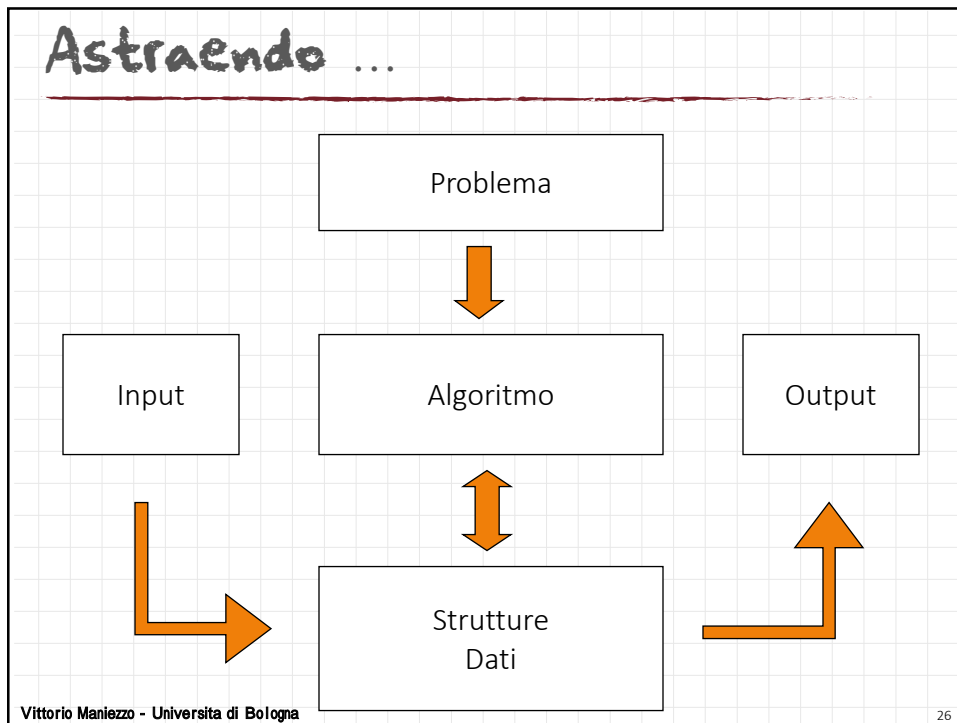
24

Astraendo ...



25

Astraendo ...



26

Insertion Sort

Insertion-sort (A)

```
1. for j=2 to size(A)
2.   do key = A[j]
3.     // insert A[j] in A[1,...,j-1]
4.     i = j-1
5.     while i>0 and A[i]>key
6.       do A[i+1] = A[i]
7.         i=i-1
8.     A[i+1] = key
```

Vittorio Maniezzo - Università di Bologna

27

27

Insertion Sort

Insertion-sort (A)

```
1. for j=2 to size(A)
2.   do key = read(j)
3.     {insert A[j] in A[1,...,j-1]}
4.     i = j-1
5.     while i>0 and read(i)>key
6.       do modify(i+1, read(i))
7.         i=i-1
8.     modify(i+1, key)
```

Vittorio Maniezzo - Università di Bologna

28

28

Strutture dati di Insertion-Sort

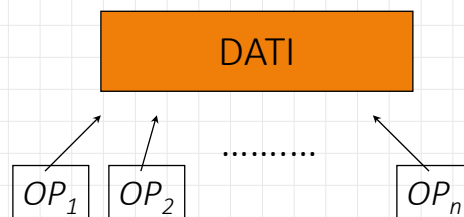
DATI: Insieme di numeri: A

OPERAZIONI: *read(i)*
size(S)
modify(i,x)

Strutture Dati Astratte

DATI
+
OPERAZIONI

} “Che cosa”



Esempio di ADS

Dati = insieme S di numeri

OP_1 = estrai il minimo

OP_2 = estrai il massimo

OP_3 = restituisci la taglia di S

OP_4 = inserisci un nuovo numero in S

ADS

ADS =

Insieme S di numeri
+
Read, Size, Modify

DS =

S="A[1,...,n]" (vettore)
Read(i)="A[i]"
Modify(i,x)="A[i]=x"

ADS

ADS = che cosa vogliamo ?

DS = come lo implementiamo ?

Insertion Sort: pseudocodice

Pseudo codice dell'algoritmo InsertionSort.

```
1: procedure InsertionSort(A)
2:   for j ← 2 to Length(A) do
3:     key ← A[j]
4:     i ← j - 1
5:     while i > 0 & A[i] > key do
6:       A[i+1] ← A[i]
7:       i ← i - 1
8:     end while
9:     A[i+1] ← key
10:  end for
11: end procedure
```

Attenzione se
lo si implementa
in C!

Costo computazionale

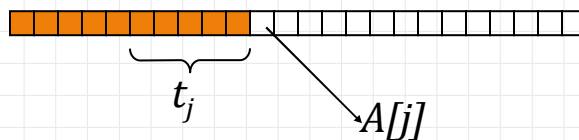
Insertion-sort (A)		costo	Num. ripetizioni
1.	for j=2 to length(A)	C_1	n
2.	do key = A[j]	C_2	$n-1$
3.	i = j-1	C_3	$n-1$
4.	while i>0 and A[i]>key	C_4	$\sum_{j=2}^n t_j$
5.	do A[i+1] = A[i]	C_5	$\sum_{j=2}^n (t_j-1)$
6.	i=i-1	C_6	$\sum_{j=2}^n (t_j-1)$
7.	A[i+1] = key	C_7	$n-1$

Vittorio Maniezzo - Università di Bologna

35

35

costo computazionale



t_j numero di volte in cui viene eseguita l'istruzione while
Dipende dai dati in input

- caso ottimo: $t_j = 1$ $\rightarrow \sum_{i=1}^n t_j = n$
- caso pessimo: $t_j = j$ $\rightarrow \sum_{i=1}^n t_j = \frac{n(n+1)}{2}$
- caso medio: $t_j = j/2$ $\rightarrow \sum_{i=1}^n t_j = \dots$

Vittorio Maniezzo - Università di Bologna

36

36

Costo computazionale

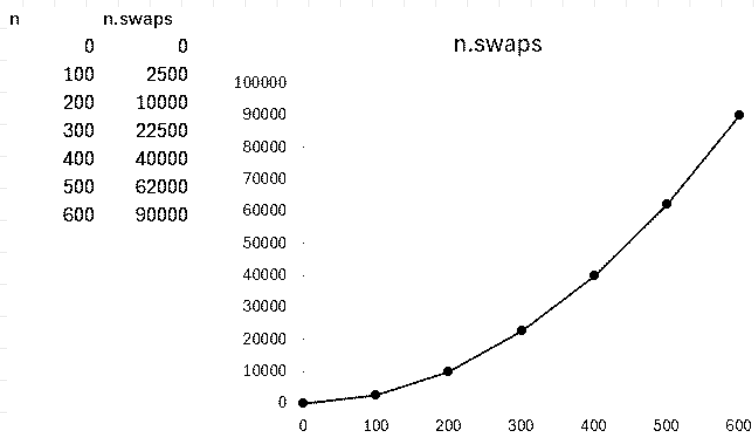
$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum t_j + (c_5 + c_6) \sum (t_j - 1) + c_7 (n-1)$$

t_j numero di esecuzioni istruzione while dipende dai dati in input

- caso ottimo: $t_j = 1$ $T(n) = an + b$; lineare
- caso pessimo: $t_j = j$ $T(n) = an^2 + bn + c$; quadratico
- caso medio: $t_j = j/2$???

37

Insertion sort, costo medio



Sperimentalmente: quadratica!

38