

# Algoritmi e Strutture Dati

## Capitolo 4

Lower bound  $\Omega(n \log n)$  per  
il problema dell'ordinamento (\*)

# Approfondimento: una terza variante dell'IS

InsertionSort3 (A)

```

1.  for k=1 to n-1 do
2.      x = A[k+1]
3.      j = ricerca_binaria(A[1,k],x)
   /* j è la posizione in cui andrà inserito x
4.      for i=k downto j do
5.          A[i+1] = A[i]
6.      A[j]=x
  
```

$r_k \leq \log k$  in quanto  
ci si ferma non  
appena si trova un  
elemento pari ad  $x$   
oppure  $x$  non viene  
trovato

$s_k \leq k$   
spostamenti

il tutto  
eseguito  
n-1  
volte

$$T(n) = \sum_{k=1}^{n-1} (r_k + s_k) \leq \sum_{k=1}^{n-1} (\log k + k) \leq \sum_{k=1}^{n-1} k = O(n^2)$$

## Una terza variante dell'IS (2)

- **Caso peggiore:**  $x$  andrà inserito in prima posizione, e quindi in tal caso  $r_k = \log k$  e  $s_k = k$ , e quindi

$$T_{worst}(n) = \sum (r_k + s_k) = \sum (\log k + k) = \Theta(n^2)$$

- **Caso migliore:** si ha quando minimizzo  $r_k + s_k$  e quindi, intuitivamente,  $x$  andrà inserito “vicino” alla posizione  $k$ -esima. Quindi, la ricerca binaria deve spostarsi **sempre verso destra**. Ma se la ricerca binaria termina dopo  $t$  iterazioni, allora  $r_k + s_k = t + n/2^t$ , e questa funzione è **monotona decrescente** per  $1 \leq t \leq \log k$ , e quindi raggiunge il suo minimo per  $t = \log k$  (cioè proprio quando  $x$  è maggiore di tutti gli elementi della sequenza). In tal caso  $r_k = \log k$  e  $s_k = 0$ , e quindi:

$$T_{best}(n) = \sum (r_k + s_k) = \sum \log k = \log n! \leq \log n^n = n \log n,$$

cioè  $T_{best}(n) = O(n \log n)$ , ma come vedremo con l'approssimazione di Stirling,

$\log n! \geq \log (n/e)^n = n \log (n/e) = \Omega(n \log n)$ , cioè  $T_{best}(n) = \Omega(n \log n)$ , e quindi

$$T_{best}(n) = \Theta(n \log n)$$

# Una terza variante dell'IS (3)

- **Caso medio:** la posizione attesa di  $x$  sarà quella mediana della sequenza, e quindi  $s_k = k/2$ , da cui

$$T_{avg}(n) \geq \sum k/2 = \Omega(n^2)$$

da cui, essendo chiaramente  $T_{avg}(n) = O(n^2)$ , ne consegue che  $T_{avg}(n) = \Theta(n^2)$

- Quindi, ricapitolando, InsertionSort3 è meglio di InsertionSort1 ma peggio di InsertionSort2.

## Approfondimento: Algoritmo Bubble Sort

```
Bubble-Sort(A)
limite = n
for k=1 to (n-1)
  do { for i=1 to (limite-1)
        do { if (A[i] > A[i+1])
              then scambia A[i] e A[i+1]
            }
        }
    limite = limite - 1
```

L'algoritmo è corretto? **Sì** (dimostrazione analoga al **Selection Sort**, facendo vedere per induzione che alla fine del passo **k** i **k** elementi più grandi della sequenza compaiono in ordine non decrescente nelle ultime **k** posizioni dell'array)

Qual è la complessità temporale dell'algoritmo?

- linea (o il blocco di linee di codice) eseguito più volte

**if (A[i] > A[i+1])** - operazione di confronto

- Numero volte che viene eseguita

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

**-Notare-** il numero di volte che questa linea viene eseguita non dipende dall'input  $\Rightarrow T(n) = \Theta(n^2)$ .

# Stato dell'arte sull'ordinamento

	Caso migliore	Caso medio	Caso peggiore	T(n)	S(n)
<b>Selection Sort</b>	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
<b>Insertion Sort 1</b>	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
<b>Insertion Sort 2</b>	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(n)$
<b>Insertion Sort 3</b>	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(n)$
<b>Bubble-Sort</b>	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$

**Nota:** il **Bubble-Sort** può essere facilmente modificato in modo tale che il caso migliore costi  $\Theta(n)$ . Come?

Quindi, per il problema dell'ordinamento...

- Lower bound **temporale**:  $\Omega(n)$ 
  - “banale”: dimensione dell'input
- Upper bound **temporale**:  $O(n^2)$ 
  - Insertion Sort 2 e 3

Abbiamo un **gap lineare** tra upper bound e lower bound!

È possibile chiudere tale gap?

# Ordinamento per confronti

Dati due elementi  $a_i$  ed  $a_j$ , per determinarne l'ordinamento relativo effettuiamo una delle seguenti operazioni di confronto:

$$a_i < a_j ; a_i \leq a_j ; a_i = a_j ; a_i \geq a_j ; a_i > a_j$$

Non si possono esaminare i valori degli elementi o ottenere informazioni sul loro ordine in altro modo.

**Notare:** Tutti gli algoritmi di ordinamento considerati fino ad ora sono algoritmi di ordinamento per confronto.

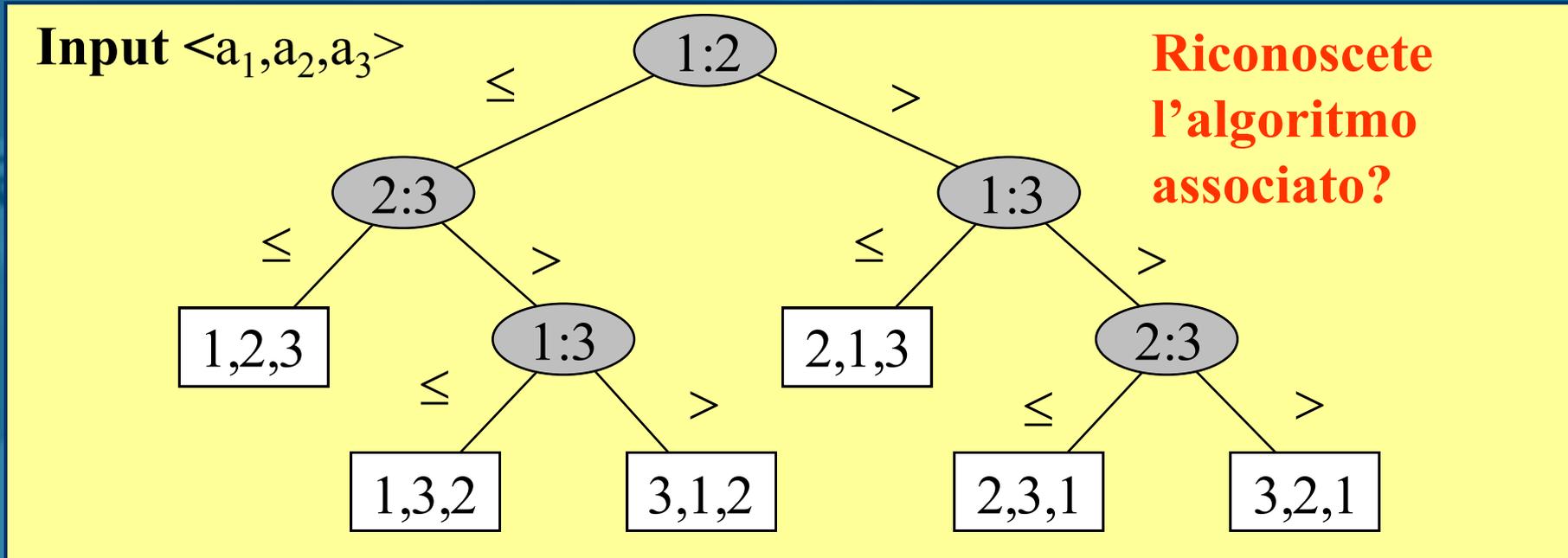
## Lower bound $\Omega(n \log n)$ per l'ordinamento

- Consideriamo un generico algoritmo  $\mathcal{A}$ , che ordini eseguendo **solo confronti**: dimostreremo che  $\mathcal{A}$  esegue (**nel caso peggiore**)  $\Omega(n \log n)$  confronti
  - Un generico algoritmo di ordinamento per confronti lavora nel modo seguente:
    - Confronta due elementi  $a_i$  ed  $a_j$  (ad esempio effettua il test  $a_i \leq a_j$ );
    - A seconda del risultato, riordina e/o decide il confronto successivo da eseguire.
- $\Rightarrow$  Un algoritmo di ordinamento per confronti può essere descritto in modo astratto usando un **albero di decisione**, nel quale i nodi interni rappresentano i confronti, mentre le foglie rappresentano gli output prodotti

# Albero di decisione

- Un albero di decisione è associato ad uno specifico **algoritmo**  $\mathcal{A}$  e ad una specifica **dimensione**  $n$  dell'istanza
- Descrive le diverse sequenze di **confronti** che  $\mathcal{A}$  esegue su un'istanza  $\langle a_1, a_2, \dots, a_n \rangle$  di lunghezza  $n$ ; i movimenti dei dati e tutti gli altri aspetti dell'algoritmo vengono ignorati
- **Nodo interno** (non foglia):  $i:j$  (modella il confronto tra  $a_i$  e  $a_j$ )
- **Nodo foglia**:  $i_1, i_2, \dots, i_n$  (modella una risposta (output) dell'algoritmo, ovvero una permutazione  $\langle a_{i_1}, a_{i_2}, \dots, a_{i_n} \rangle$  degli elementi)

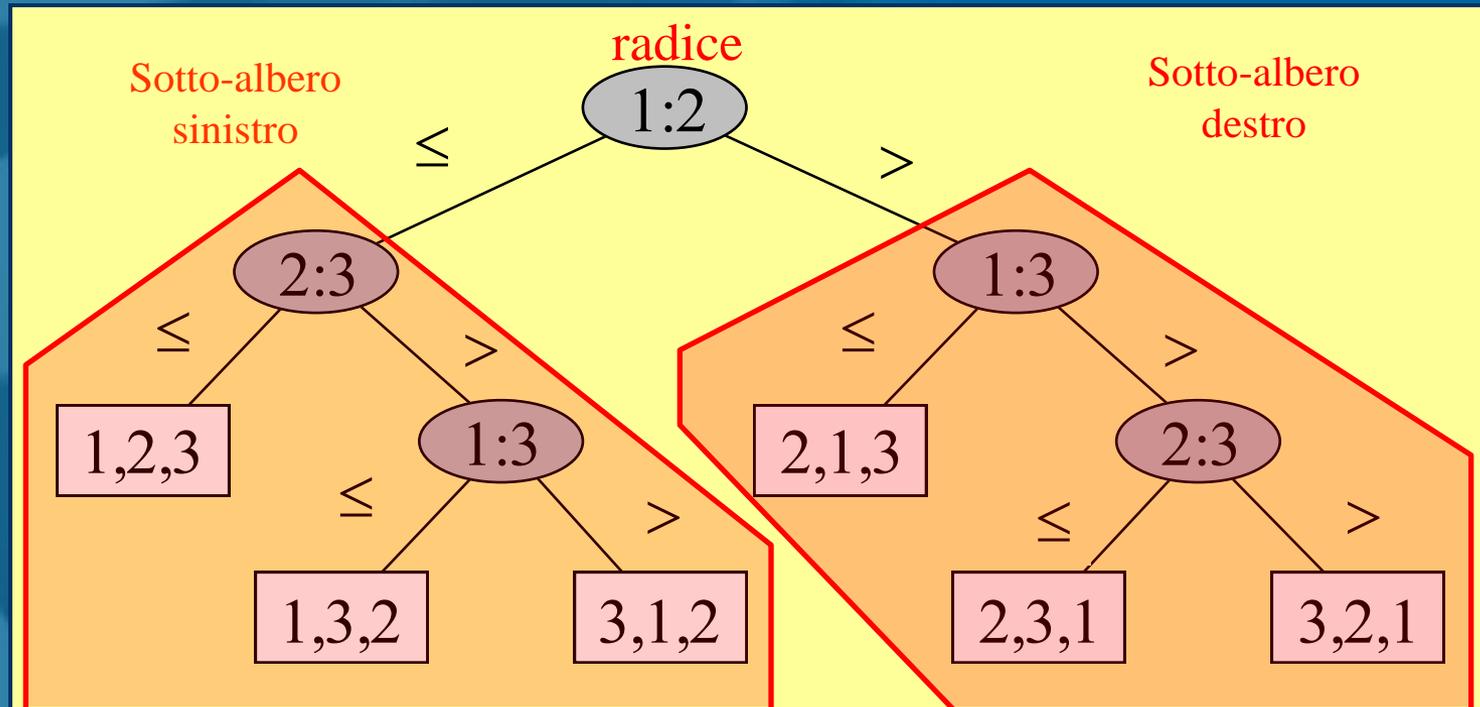
# Esempio



È proprio l'**Insertion Sort 2** su una sequenza di 3 elementi!

**Approfondimento:** costruire l'albero di decisione per il SS (e per il BS) su una sequenza di 3 elementi.

# Richiamo di definizioni



**Profondità di un nodo:** lunghezza del cammino (in termini di **numero di archi**) che lo congiunge alla radice.

**Altezza di un albero:** valore massimo della profondità dei nodi.

# Proprietà

- Per una particolare istanza, i confronti eseguiti da  $\mathcal{A}$  su quella istanza rappresentano un **cammino radice – foglia**
- L'algoritmo segue un cammino diverso a seconda delle caratteristiche dell'input
  - Caso peggiore: cammino più lungo
  - Caso migliore: cammino più breve
- Il numero di confronti nel caso peggiore è pari **all'altezza dell'albero di decisione** (ovvero alla lunghezza, in termini di numero di archi, del più lungo cammino **radice-foglia**)

# Altezza in funzione delle foglie

**Lemma:** Un albero binario (ovvero, in cui ogni nodo interno ha al più due figli) con  $k$  foglie ha altezza  $h(k) \geq \log_2 k$ .

**Dim:** Dimostrazione per induzione sul numero di foglie  $k$ :

- Caso base  $k=1$ : banale, perché ogni albero con una foglia deve avere almeno altezza  $\log_2 1=0$  (anche nel caso limite dell'albero costituito da un unico nodo ●)
- Caso  $k>1$ : supposto vero per  $k-1$  foglie, dimostriamolo per  $k$ ; poiché la radice ha almeno un figlio, uno dei suoi al più due sottoalberi deve contenere almeno la metà (parte intera superiore) delle foglie, e quindi

$$\begin{aligned} h(k) &\geq 1 + h(\lceil k/2 \rceil) \geq (\text{hp induttiva}) 1 + \log_2(k/2) \\ &= 1 + \log_2 k - \log_2 2 = \log_2 k. \end{aligned}$$

QED

# Il lower bound $\Omega(n \log n)$

- Consideriamo l'albero binario di decisione di un qualsiasi algoritmo che risolve il problema dell'ordinamento di  $n$  elementi
- Tale albero deve avere almeno  $n!$  foglie: infatti, se l'algoritmo è corretto, deve contemplare tutti i possibili output, ovvero le  $n!$  permutazioni della sequenza di  $n$  elementi in input
- Dal lemma precedente, avremo che l'altezza  $h(n)$  dell'albero di decisione sarà:

$$h(n) \geq \log_2(\#foglie) \geq \log_2(n!) > \log_2 (n/e)^n =$$

Formula di Stirling:

$$n! \geq (2\pi n)^{1/2} \cdot (n/e)^n > (n/e)^n$$

$$= n \log_2 (n/e) =$$

$$= n \log_2 n - n \log_2 e =$$

$$= \Omega(n \log n)$$

QED