Algoritmi e Strutture Dati

Capitolo 4
Lower bound Ω(n log n) per il problema dell'ordinamento (*)

Approfondimento: una terza variante dell'IS

InsertionSort3 (array 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

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

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

$$T_{worst}(n) = \sum_{k=1}^{n-1} (\log k + k) = \Theta(n^2)$$

- 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 una tale ricerca binaria termina dopo t iterazioni, allora $r_k + s_k = t + k/2^t$, e questa funzione è monotona descrescente 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:

$$T_{best}(n) = \sum_{k=1}^{n-1} \log k = \log (n-1)+\ldots+ \log 1 \leq \log n! \leq \log n^n = n \log n,$$

cioè $T_{best}(n) = O(n \log n)$, ma vale anche

$\log (n-1) +\ldots+ \log 1 \geq$ (prendo solo i primi (n-1)/2 termini)

$\log ((n-1)/2)+\ldots+\log ((n-1)/2) = (n-1)/2 \log ((n-1)/2) = \Omega(n \log n)$,

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, senza nemmeno contare i confronti delle chiamate ricorsive

$$T_{avg}(n) \geq \sum_{k=1}^{n-1} k/2 = \Omega(n^2)$$

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.

Stato dell'arte sull'ordinamento

	Caso migliore	Caso medio	Caso peggiore	T(n)	S(n)
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
Insertion Sort 1	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
Insertion Sort 2	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(n)$
Insertion Sort 3	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(n)$

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 Ω(n log n) per l'ordinamento

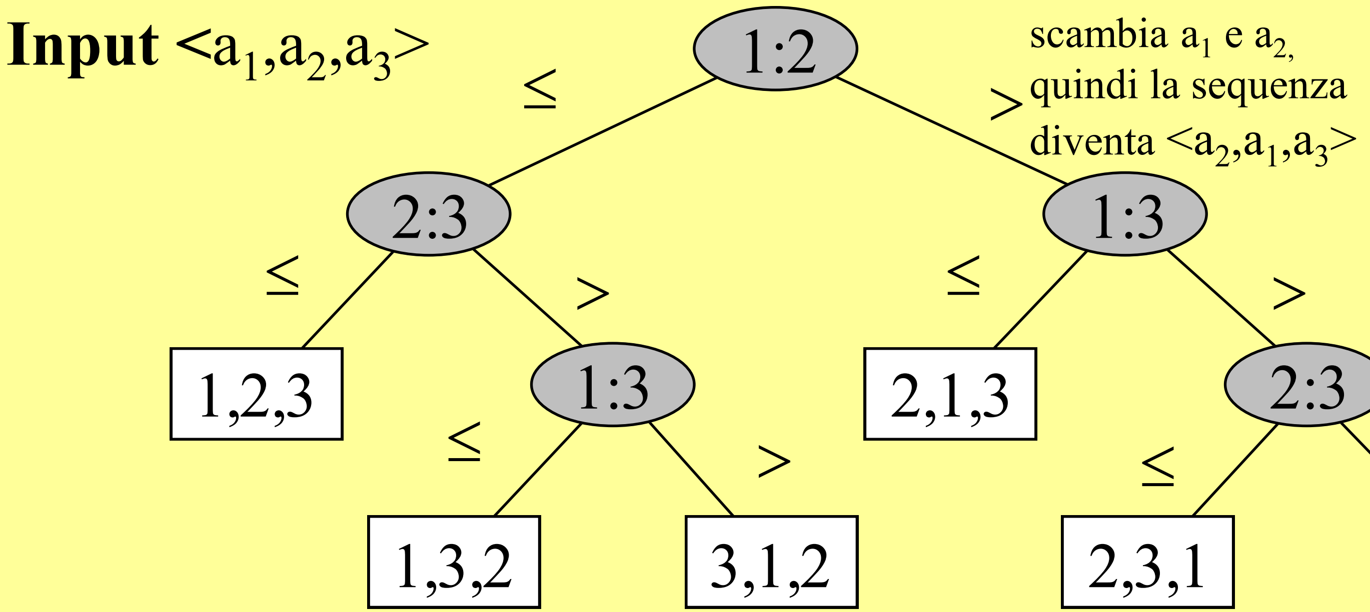
- Consideriamo un generico algoritmo $\mathcal{A}$, che ordini eseguendo solo confronti: dimostreremo che $\mathcal{A}$ esegue (nel caso peggiore) Ω(n log n) confronti
- Un generico algoritmo di ordinamento per confronti lavora nel modo seguente:
  - Confronta due elementi a_i ed a_j (ad esempio effettua 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 $<a_1,a_2,\ldots,a_n>$ 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: $\boxed{i_1,i_2,\ldots,i_n}$ (modella una risposta (output) dell'algoritmo, ovvero una permutazione $<a_{i_1},a_{i_2},\ldots,a_{i_n}>$ degli elementi)

Esempio

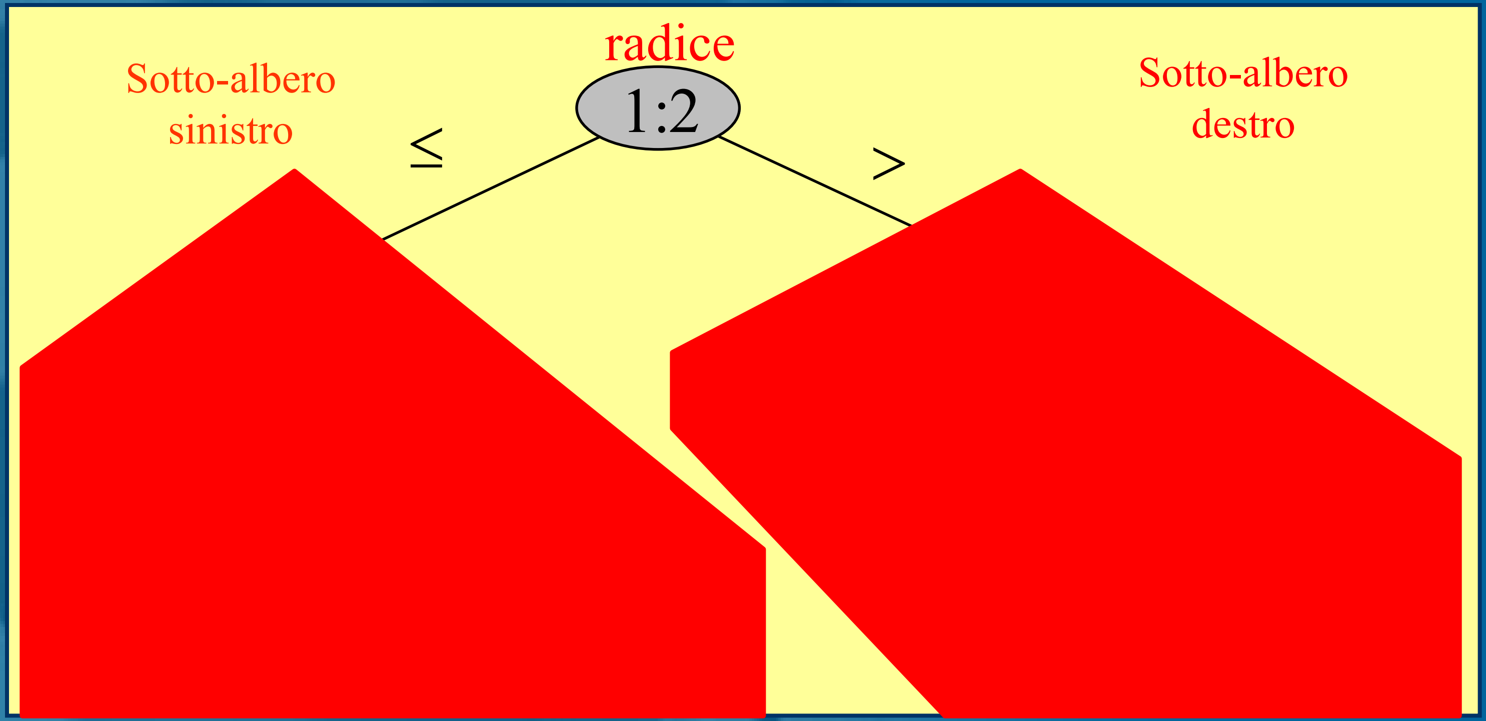


Riconoscete l’algoritmo associato?

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

Approfondimento: costruire l’albero di decisione per il SS e per l’IS-1 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 $h(k) \geq \log k$.

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 delle foglie, e quindi

$$h(k) \geq 1+h(k/2) \geq_{\text{(hp induttiva)}} 1+\log (k/2)$$

$$=1+\log k-\log 2=\log k.$$

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:

$$
\begin{aligned}
h(n) \geq \log(\#\text{foglie}) \geq \log(n!) & \\
&= \log(n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot 2\cdot 1) \\
&= \log n + \log(n-1) + \ldots + \log(n/2) + \ldots + \log 2 + \log 1 \\
&\geq \log n + \log(n-1) + \ldots + \log(n/2) \quad \text{(i termini successivi vengono trascurati)} \\
&\geq \log(n/2) + \log(n/2) + \ldots + \log(n/2) \\
&= n/2 \log(n/2), \text{ ovvero } h(n) = \Omega(n \log n). \qquad \text{QED}
\end{aligned}
$$