Algoritmi e Strutture Dati

Soluzione esercizi di approfondimento

Stefano Leucci

stefano.leucci@univaq.it

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=j to k do
5.             A[i+1] = A[i]
6.        A[j]=x
```

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

$s_k \le k$ spostamenti

il tutto eseguito n-1 volte

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

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 $r_k + s_k = t + k/2^t$, e questa funzione è monotona descrescente, 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:

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

cioè $T_{best}(n) = O(n \log n)$, ma come vedremo con l'approssimazione di Stirling, $\log n! = \Omega(n \log n)$, 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 $r_k = O(\log k)$ e $s_k = k/2$, da cui

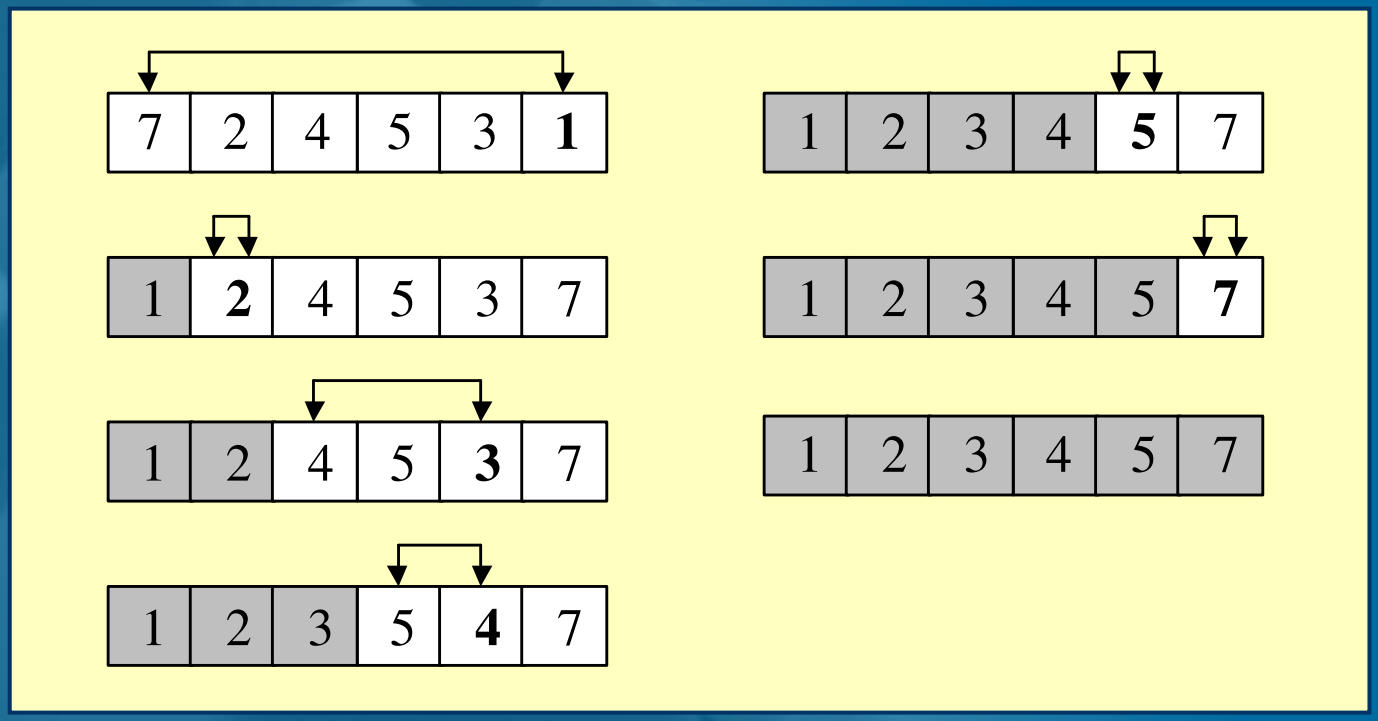
$$T_{avg}(n) = \sum(r_k + s_k) = \sum (\log k + k/2) = \Theta(n^2)$$

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

	Caso migliore	Caso medio	Caso peggiore	T(n)	S(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`	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(n)$

SelectionSort

Approccio incrementale: assumendo che i primi k-1 elementi siano ordinati e siano i k-1 elementi più piccoli della sequenza, estende l'ordinamento ai primi k elementi scegliendo il **minimo** degli elementi non ancora ordinati in posizione k, k+1,…, n, e mettendolo in posizione k

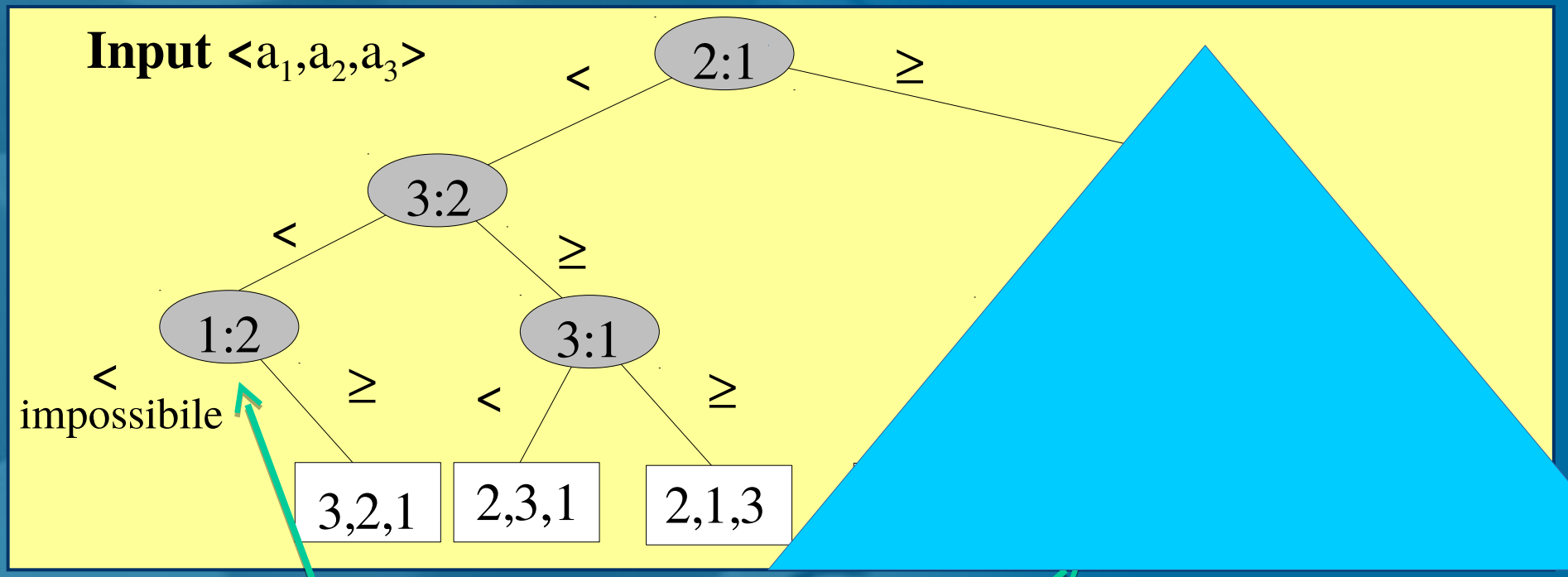


```
SelectionSort (A)
1.    for k=1 to n-1 do
2.        m = k
3.        for j=k+1 to n do
4.            if (A[j] < A[m]) then m=j
5.        scambia A[m] con A[k]
```

NOTA: Assumiamo che il primo elemento dell'array sia in A[1]

- Linea 1: k mantiene l'indice dove andrà spostato il minimo degli elementi in posizione k, k+1, …, n.
- Linea 2: m mantiene l'indice dell'array in cui si trova il minimo corrente
- Linee 3-4: ricerca del minimo fra gli elementi A[k],…,A[n] (m viene aggiornato con l'indice dell'array in cui si trova il minimo corrente)
- Linea 5: il minimo è spostato in posizione k (si noti che questa operazione richiede 3 operazioni elementari di assegnamento)

Albero di decisione del SS



- Osservazione 1: l'albero non è strettamente binario: compare un confronto inutile!
- Osservazione 2: tutte le foglie sono alla stessa altezza (e infatti il SS esegue sempre lo stesso numero di confronti!)
- Osservazione 3: la permutazione <3,2,1> è raggiungibile solo nel caso in cui $a_1=a_2$

Merge alternativo di 2 heap d-ari

Fornire un'implementazione alternativa dell'operazione di **merge(heap d-ario c1, heap d-ario c2)** in cui gli elementi di uno dei due heap vengono aggiunti sequenzialmente all'altro heap. Analizzarne quindi la convenienza asintotica rispetto all'implementazione classica di costo $\Theta(n)$.

Soluzione: Sia $k=\min\{|c_1|,|c_2|\}$. Inseriamo ad uno ad uno tutti gli elementi della coda più piccola nella coda più grande; questo costa $O(k \log n)$, dove $n=|c_1|+|c_2|$.

L'approccio conviene quindi per $k \log n=o(n)$, cioè per $k=o(n/\log n)$.

Code con Priorità: Riepilogo delle operazioni elementari

	FindMin	Insert	Delete	DeleteMin
Array non ord.	$\Theta(n)$	$O(1)$	$O(1)$	$\Theta(n)$
Array ordinato	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Lista non ordinata	$\Theta(n)$	$O(1)$	$O(1)$	$\Theta(n)$
Lista ordinata	$O(1)$	$O(n)$	$O(1)$	$O(1)$

Esercizio di approfondimento #2

Valutare i costi delle operazioni aggiuntive (IncreaseKey, DecreaseKey e Merge) sulle implementazioni elementari (vettori e liste).

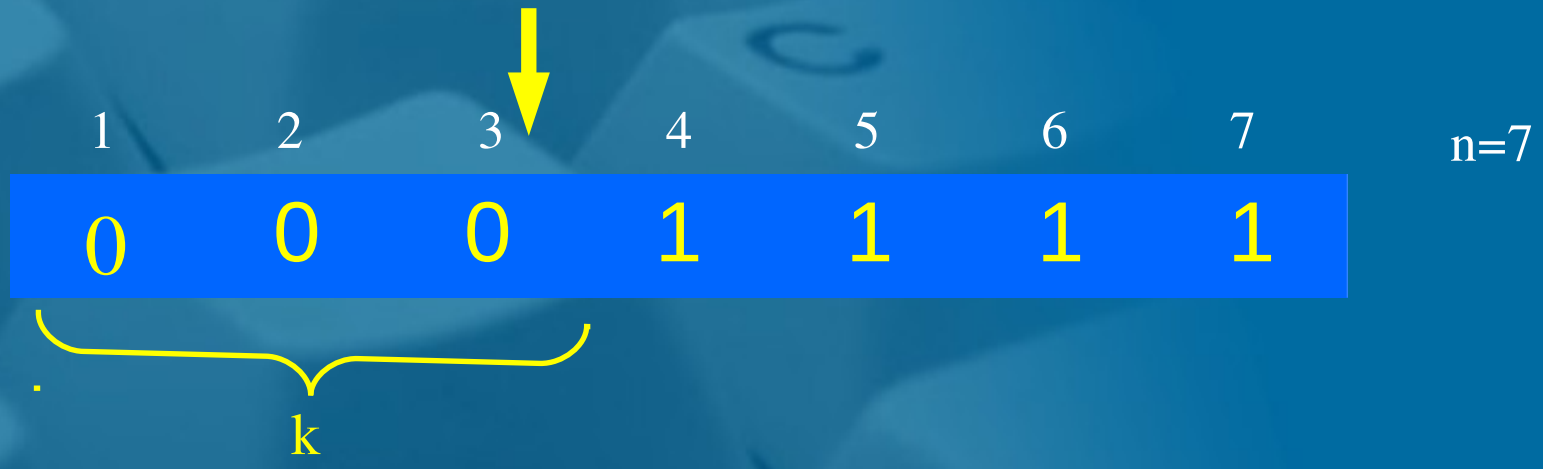
Soluzione esercizio di approfondimento #2

	Increase Key	Decrease Key	Merge
Array non ord.	O(1)	O(1)	$\Theta(k)$ $k=\min\{\|c_1\|,\|c_2\|\}$
Array Ordinato	O(n)	O(n)	$\Theta(n)$ $n=\|c_1\|+\|c_2\|$
Lista non Ordinata	O(1)	O(1)	O(1)
Lista Ordinata	O(n)	O(n)	O(n)

Alcuni esercizi

Dato un vettore ordinato di n elementi binari, trovare:

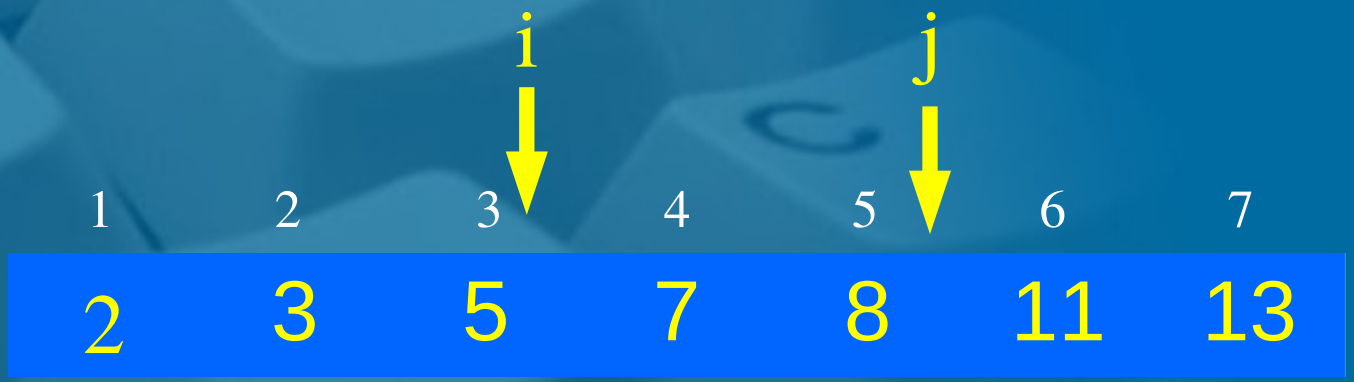
- l'ultimo 0 in tempo o(n)
- l'ultimo 0 in tempo O(log k) dove k è il numero complessivo di 0 nel vettore.



Alcuni esercizi

Dato un vettore A ordinato di n interi positivi ed un intero x, trovare (se esistono) due indici i,j con i<j tali che: A[i]+A[j]=x.

Tempo: $o(n^2)$. Sugg: si può fare in O(n log n) e O(n)



Alcuni esercizi

Si consideri una tavoletta di cioccolata rettangolare composta da n file di m quadratini di cioccolata. Si vuole effettuare una serie di spezzate in modo da avere tutti i quadratini di cioccolata separati.
Ogni spezzata consiste nel prendere un pezzo di cioccolata (di qualsiasi forma) e separarlo in due pezzi di cioccolata (di qualsiasi forma).

Una strategia è quella di separare prima le n file e poi, per ogni fila, separare i quadratini ad uno ad uno. Questa strategia esegue $(n-1)+n(m-1) = nm-1$ spezzate.

Esiste una strategia migliore? (Perché?)

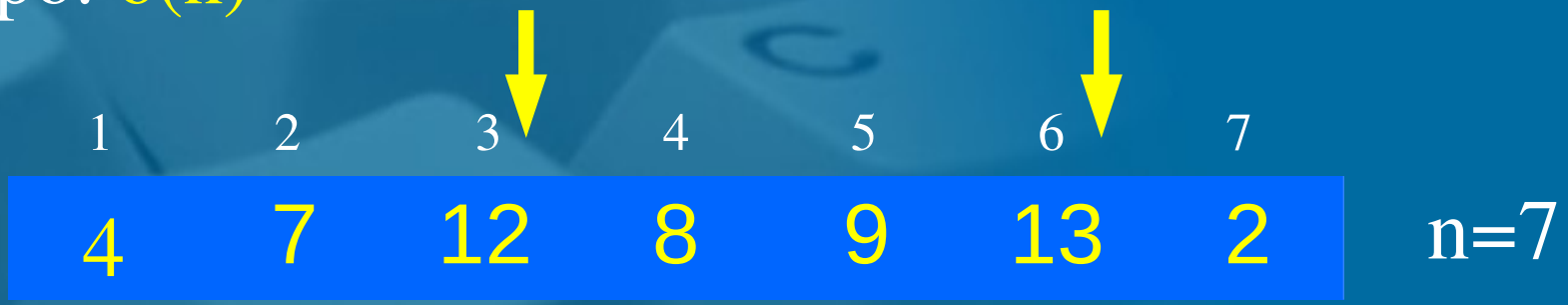
m

n

Alcuni esercizi

Dato un vettore A[1:n] di interi (non necessariamente ordinato), individuare l'indice $1 \le i \le n$ di un "picco", cioè di un elemento A[i] tale che $A[i] \ge [Ai+1]$ ed $A[i] \ge A[i-1]$. (si pensi $A[0]=A[n+1]=-\infty$).

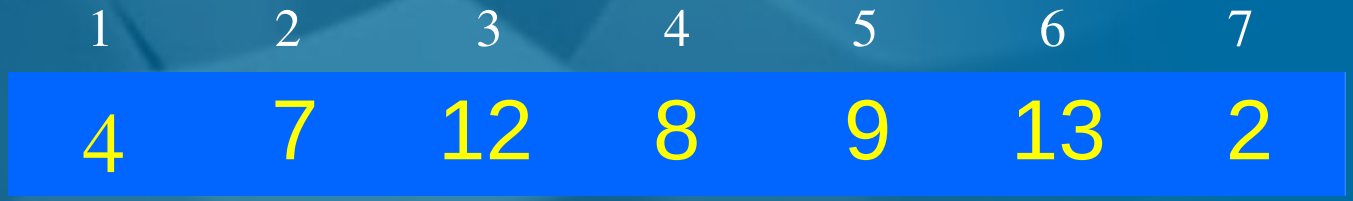
Tempo: o(n)



Alcuni esercizi

Dato un vettore (potenzialmente non ordinato) con elementi interi distinti, si definisce inversione una coppia di elementi A[i], A[j] con i<j tale che A[i]>[j]. Progettare un algoritmo efficiente che calcola il numero di inversioni.

Tempo: o(n^2)



Numero inversioni = 8