

Esercitazione 5

Algoritmi di selezione

Algoritmi di selezione

- Gli algoritmi per *risolvere problemi di selezione e di statistiche d'ordine* permettono di estrarre da grandi quantità di dati un piccolo insieme di numeri che ne rappresentino alcune caratteristiche salienti
- Esempi tipici includono il calcolo della *media*, della *moda*, o del *mediano di n numeri*

Il problema della selezione

- Restituisce la **k-esima statistica d'ordine** dell'array di input **A**
 - Il k-esimo elemento più piccolo in **A**
- **Input**: un insieme di **n** numeri (array **A**) e un numero **k**, con $1 \leq k \leq n$
- **Output**: l'elemento **x** di **A** che è maggiore esattamente di altri **k-1** elementi di **A**

	1	2	3	4	5	6	7	8	9	10	
Esempio	4	3	6	8	7	1	2	5	9	10	k=3 x=3

Quickselect

- ***Algoritmo randomizzato ricorsivo*** che trova il **k-esimo** elemento di un array disordinato di dimensione **n**
- Si basa sull'algoritmo di ordinamento **quicksort**
- **Idea di base:** partizionare ricorsivamente l'array *A*
- Comportamento determinato in parte dall'**output di un generatore di numeri casuali**

Funzionamento

- Partizionamento dell'array attorno ad un elemento *pivot*
 - Stessa funzione *partition* vista in precedenza – *versione randomizzata*
- Diversamente da **quicksort**, **quickselect** opera su un solo **sottoarray** della partizione prodotta
 - Identificazione del **sottoarray** che contiene il k-esimo elemento più piccolo
 - Chiamata **ricorsiva** su quel sottoarray

Algoritmo ricorsivo

- ***Passo ricorsivo***
 - **Partizionamento**
 - **Termine** se il pivot è il k-esimo elemento più piccolo → ritorno del pivot
 - Altrimenti **invocazione ricorsiva** sul sottoarray che contiene il k-esimo elemento più piccolo
- ***Condizione di terminazione***
 - Invocazione su un sottoarray di **dimensione unitaria**

Programma

- *quickselect.cc*
- Programma che seleziona e ritorna la k-esima statistica d'ordine in un array di input di n elementi usando l'algoritmo **quickselect**
- L'algoritmo fa uso della **randomized partition** per partizionare l'array di input

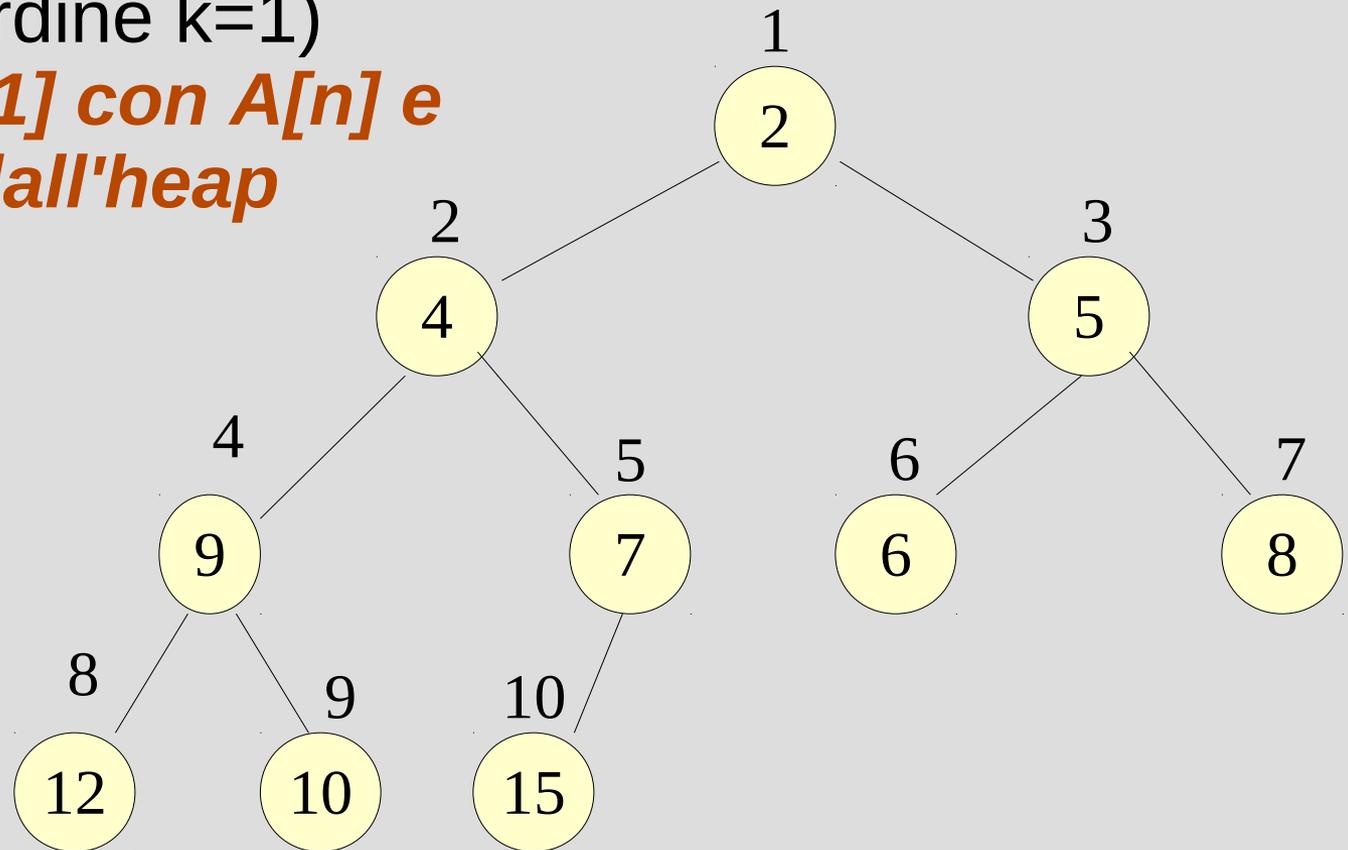
Heapselect

- Utilizzo di una struttura *min-heap* per calcolare la k-esima statistica d'ordine
 - Ispirato all'algoritmo **heapsort**
- Si costruisce un **min-heap** a partire da un array di input **A** di **n** elementi
 - *Proprietà min-heap: $A[\text{Parent}(i)] \leq A[i]$*
 - *BuildMinHeap e MinHeapify*
- Infine si seleziona il **k-esimo elemento più piccolo del min-heap**

Selezione a partire da una struttura min-heap

In $A[1]$ ho l'elemento più piccolo del **min-heap** (1° statistica d'ordine $k=1$)

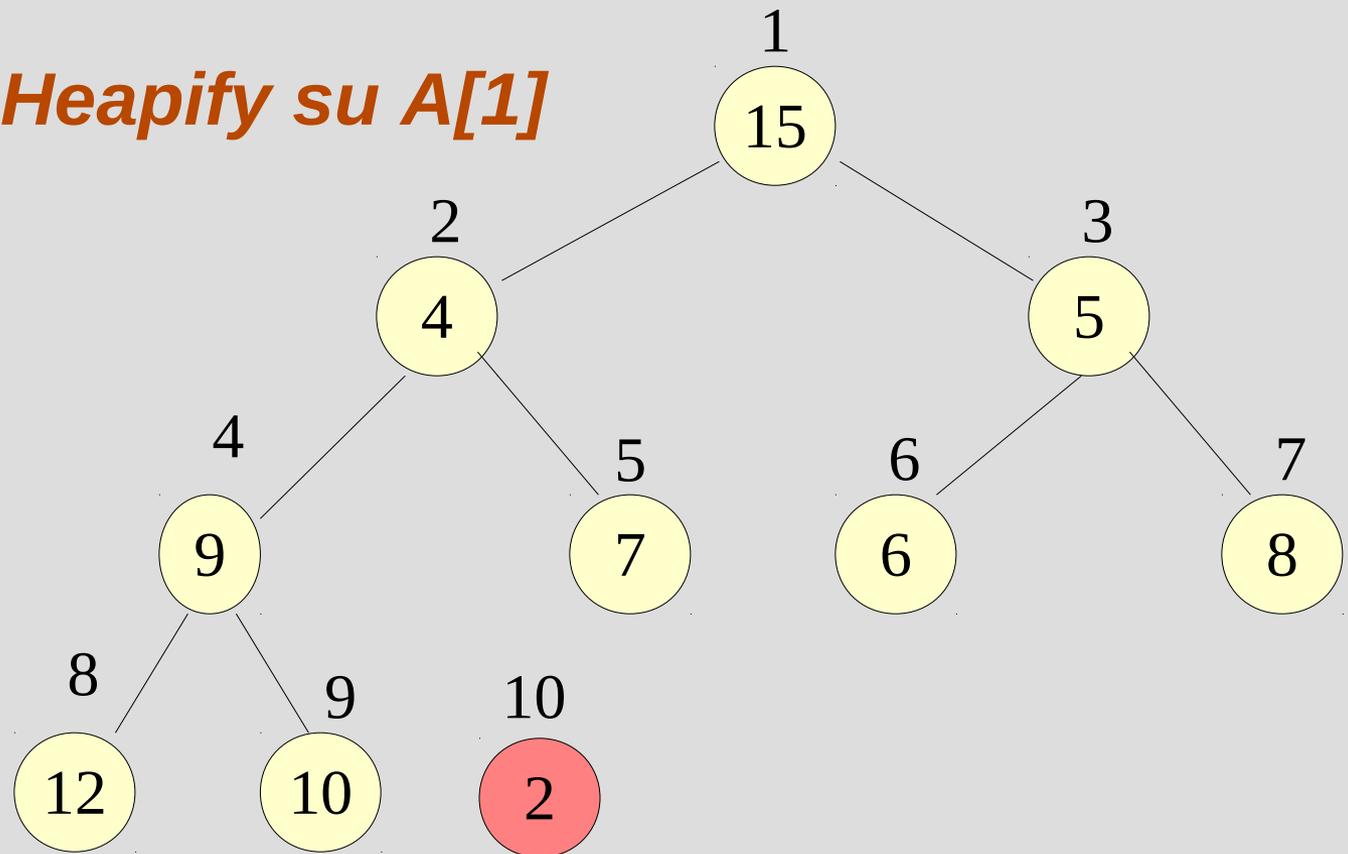
Scambio $A[1]$ con $A[n]$ e lo 'stacco' dall'heap



Selezione a partire dal min-heap

In $A[1]$ ho l'unico elemento che può violare le proprietà di min-heap

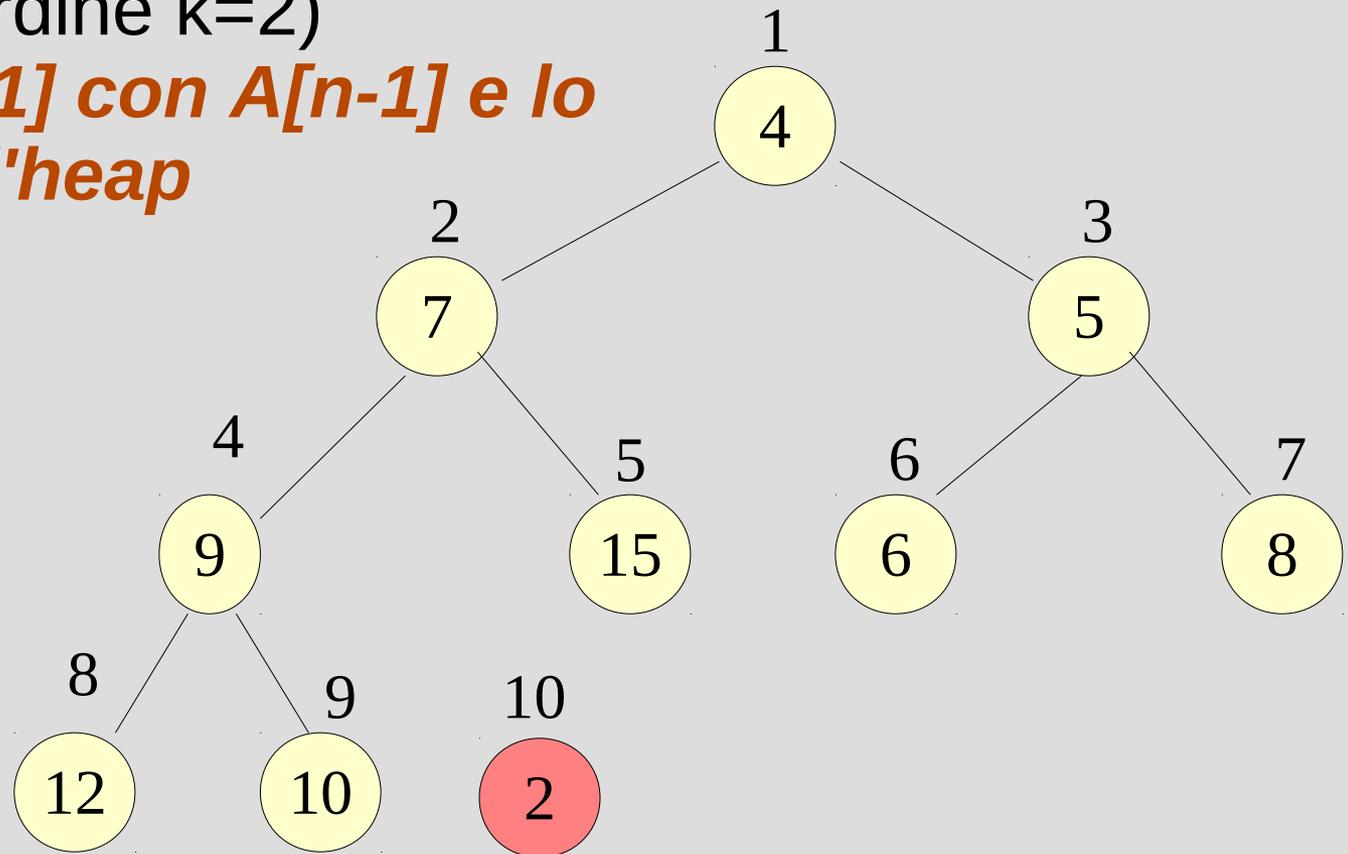
Chiamo MinHeapify su $A[1]$



Selezione a partire dal min-heap

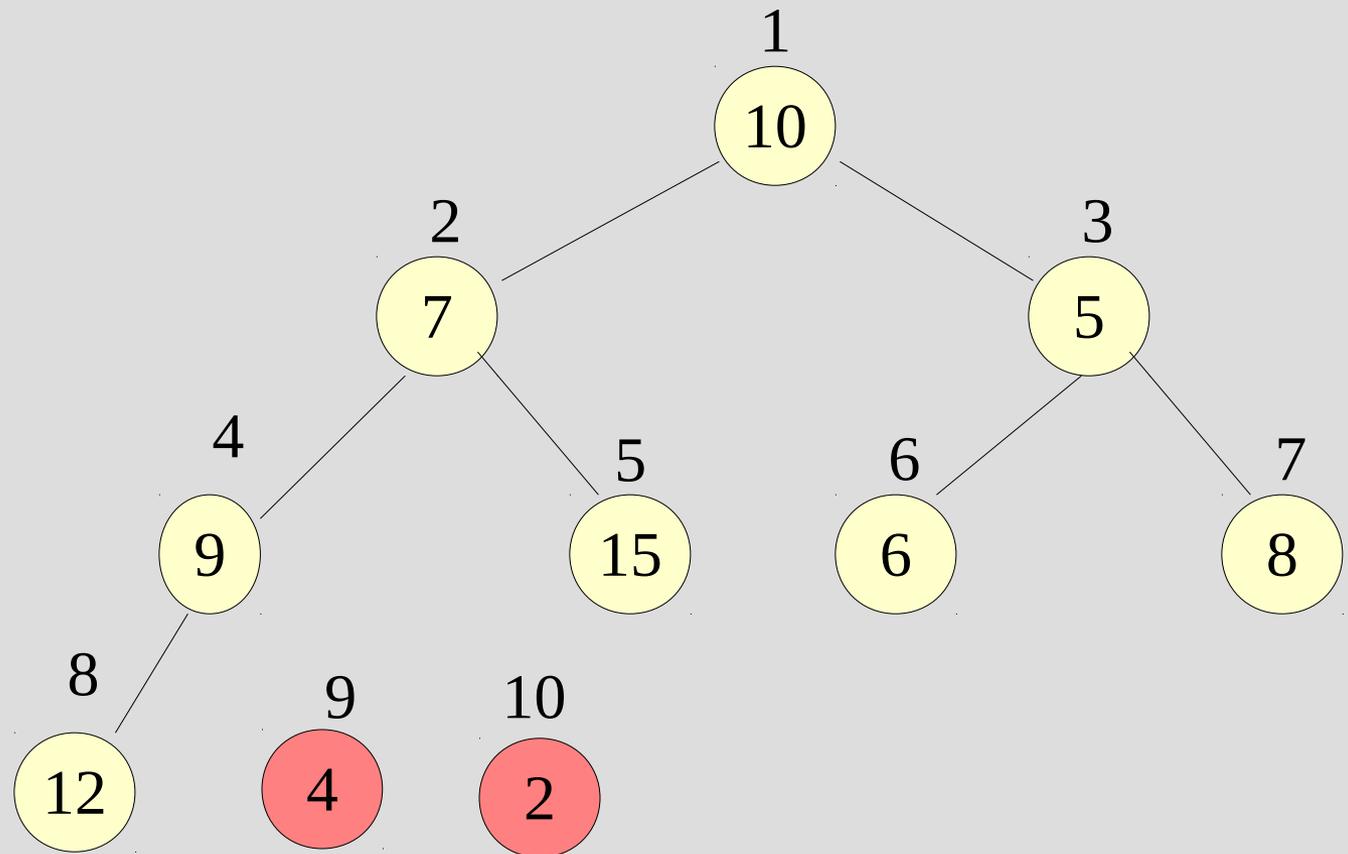
In $A[1]$ ho di nuovo l'elemento più piccolo del **min-heap** (2° statistica d'ordine $k=2$)

Scambio $A[1]$ con $A[n-1]$ e lo 'stacco' dall'heap



Selezione a partire dal min-heap

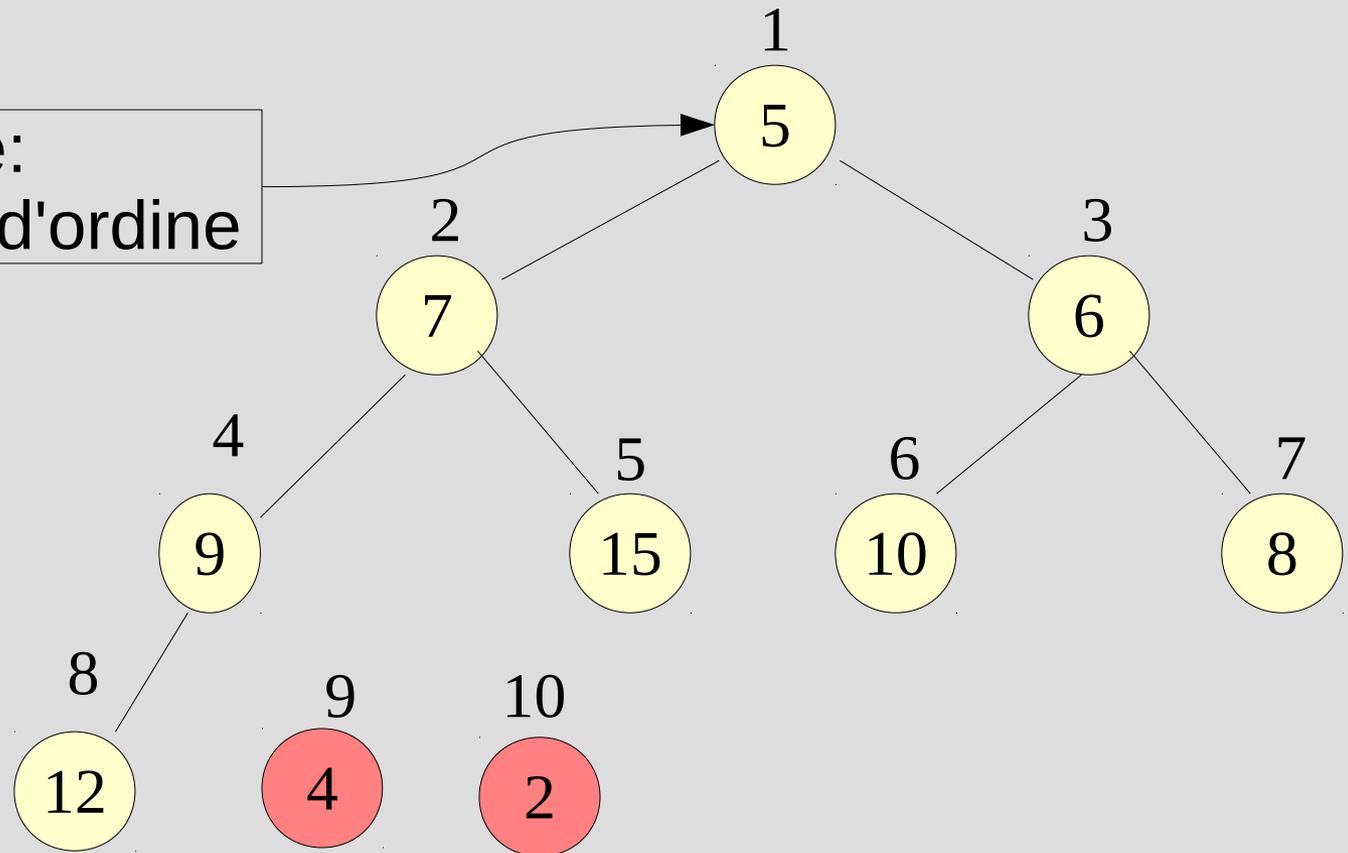
Chiamo MinHeapify su A[1]...



Selezione a partire dal min-heap

Così via per k volte: alla k -esima iterazione ho in $A[1]$ il k -esimo elemento più piccolo dell'array iniziale

3° iterazione:
3° statistica d'ordine



Programma

- *heapselect.cc*
- Programma che seleziona e ritorna la k-esima statistica d'ordine in un array di input di n elementi usando l'algoritmo **heapselect**
- L'algoritmo utilizza una struttura **min-heap** costruita a partire dall'array di input iniziale