Esercitazione 3

Heapsort

Heapsort

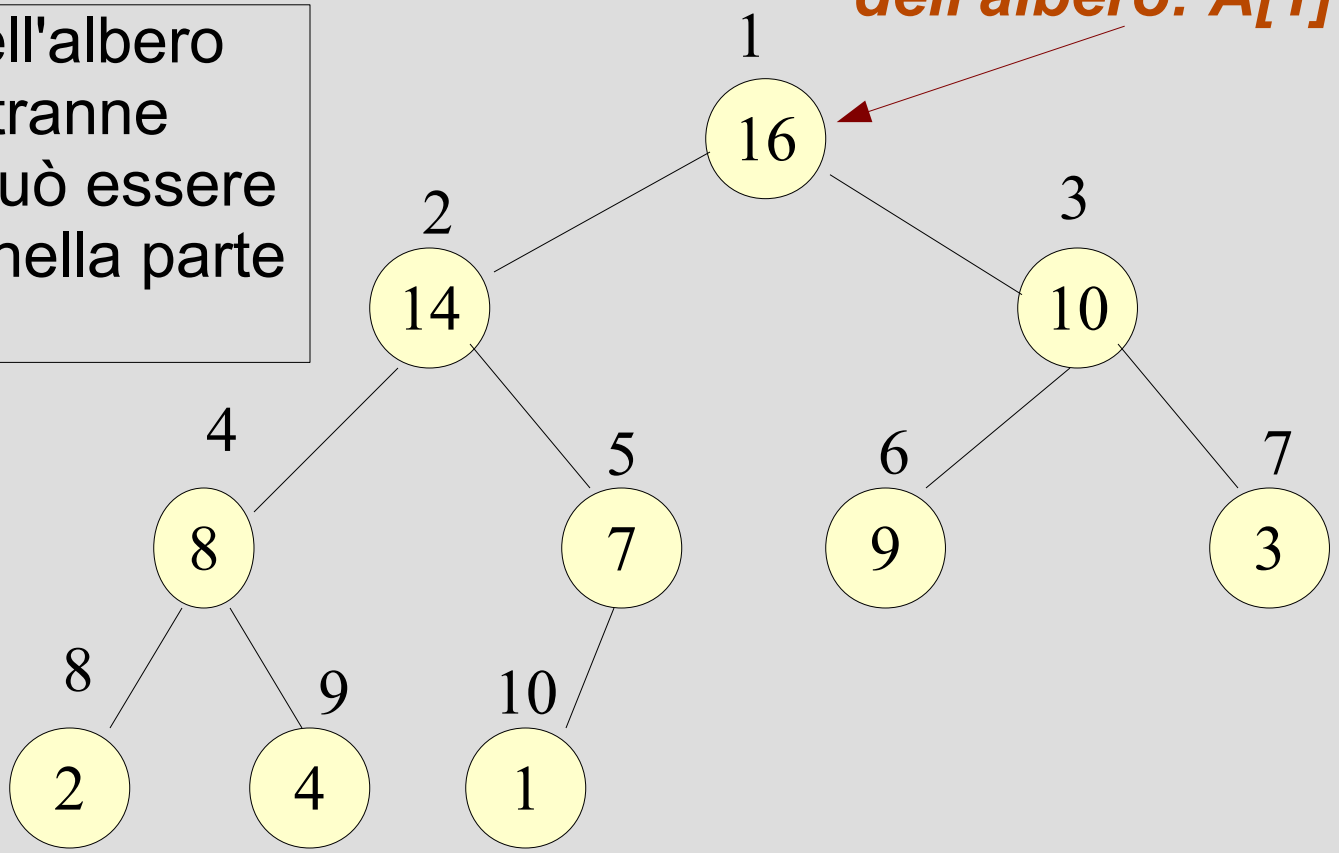
- Algoritmo di ordinamento che utilizza una struttura dati detta ***heap*** per la gestione delle informazioni
- Tempo di esecuzione $O(n \lg n)$
- ***Heap (binario)*** = struttura dati composta da un array che possiamo considerare un **albero binario quasi completo**
- Ogni nodo dell'albero corrisponde ad un elemento dell'array che memorizza il valore del nodo

Heap

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Array A

Tutti i livelli dell'albero sono riempiti tranne l'ultimo, che può essere riempito solo nella parte sinistra



Proprietà dell'heap

Se i è l'indice di un nodo, gli indici di padre, figlio destro e sinistro possono essere facilmente calcolati:

- Parent(i) { return $\lfloor i/2 \rfloor$; }
- Left(i) { return 2i; }
- Right(i) { return 2i+1; }

Funzioni da usare per muoversi all'interno della struttura ad albero!

Due tipi di heap binari: ***max-heap*** e ***min-heap***:

- ***Proprietà max-heap: A[Parent(i)] >= A[i]***
- ***Proprietà min-heap: A[Parent(i)] <= A[i]***

per ogni nodo diverso dalla radice A[1]

Algoritmo di ordinamento

- Ordiniamo un array di interi di lunghezza n con l'***algoritmo heapsort***
- Ordinamento in ***senso non decrescente*** → utilizziamo i ***max-heap***

Subroutine utilizzate:

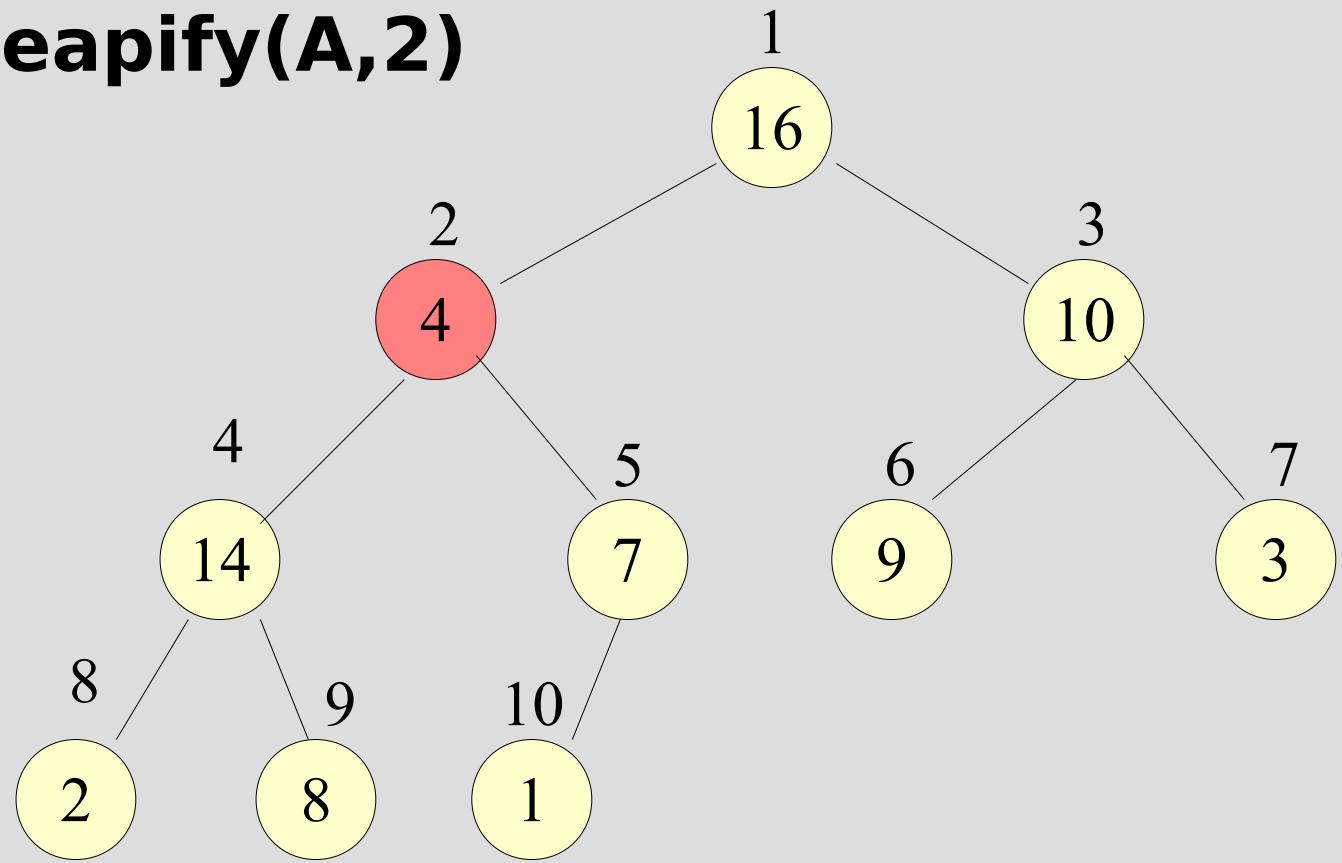
- ***Max-heapify*** per la conservazione (ripristino) delle proprietà dei max-heap
- ***Build-max-heap*** per generare un max-heap da un array di input non ordinato

Max-Heapify

- ***Input***
  - **Array A** e **indice i** nell'array
- ***Ipotesi***
  - Gli alberi binari con radici in **Left(i)** e **Right(i)** siano max-heap
  - **A[i]** può essere più piccolo dei suoi figli, violando quindi la proprietà del max-heap
- ***Obiettivo***
  - **Ripristinare** la proprietà del max-heap

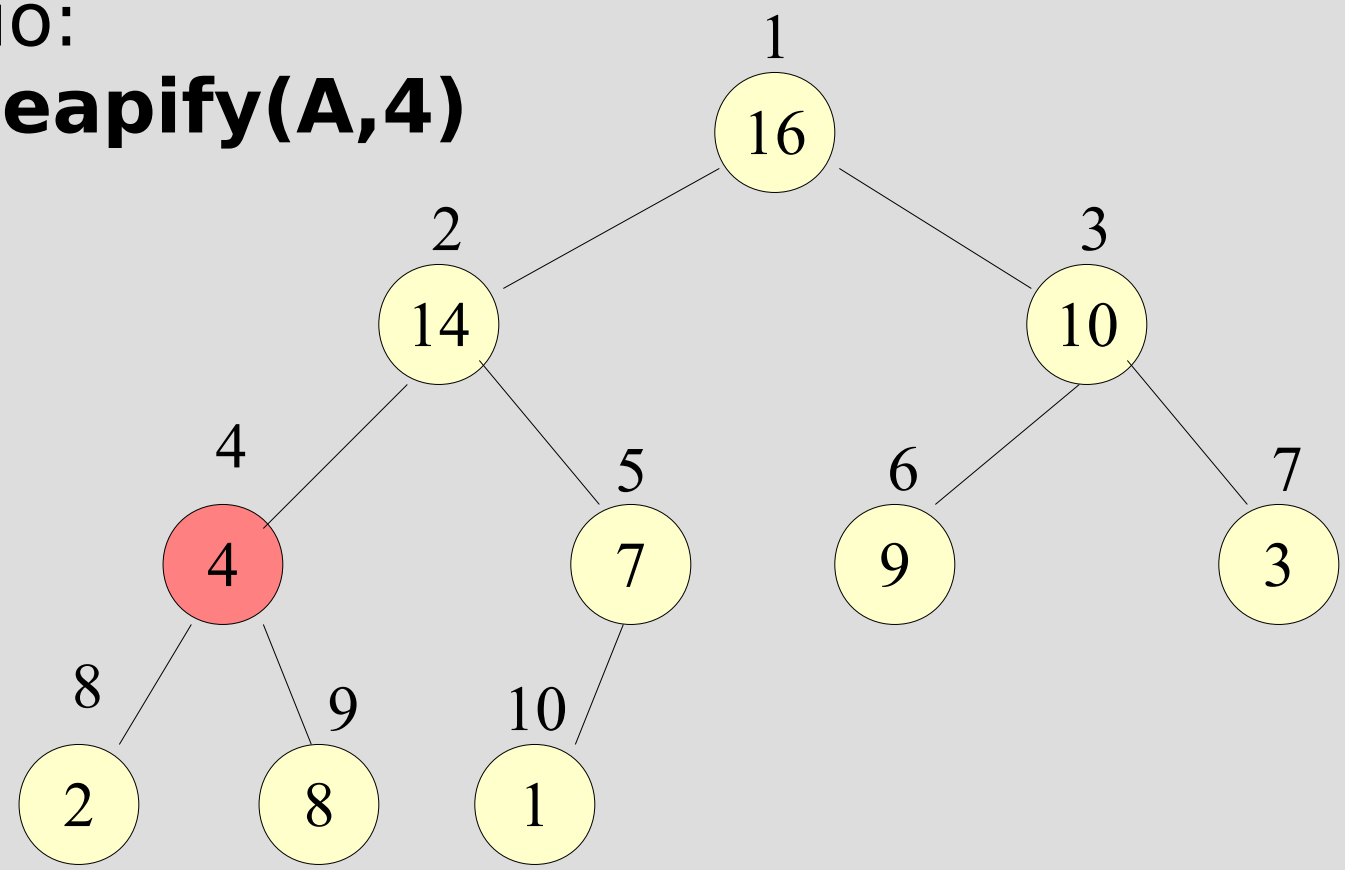
Max-Heapify

Esempio:
Max-Heapify(A,2)



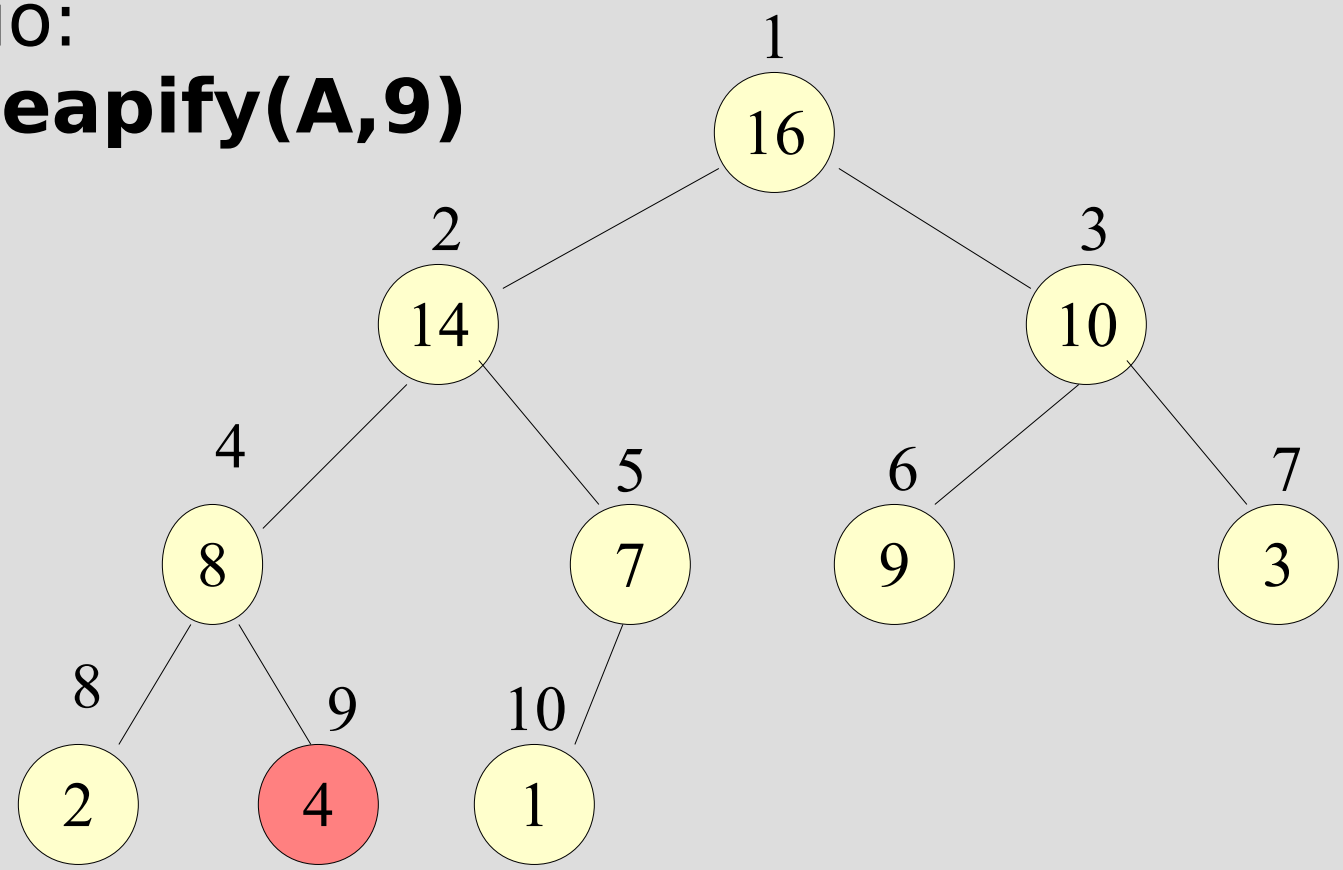
Max-Heapify

Esempio:
Max-Heapify(A,4)



Max-Heapify

Esempio:
Max-Heapify(A,9)



Algoritmo Max-Heapify

Max-Heapify(A,i)

- Controllo sul valore di A[i], Left[i] e Right[i]
- Se A[i] è il maggiore la proprietà è già soddisfatta → termine dell'algoritmo
- Altrimenti, A[i] viene scambiato col massimo tra Left[i] e Right[i]
- A[i], Left[i] e Right[i] soddisfano la proprietà, ma ciò non è assicurato per il sottoalbero che ha per radice il nodo con cui ho effettuato lo scambio

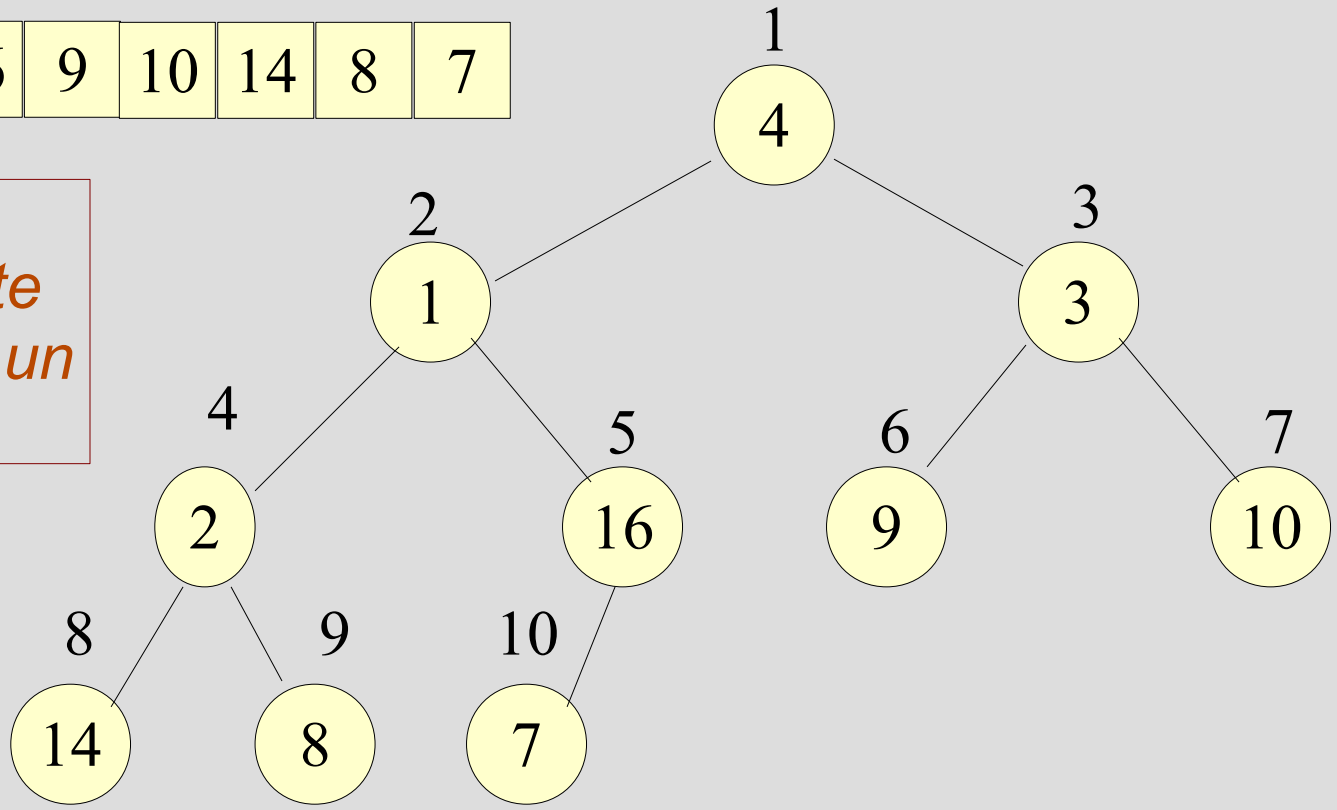
  → ***chiamata ricorsiva a Max-Heapify***

Costruire un heap

Come costruire un heap partendo da un array **A** di input?

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

L'albero corrispondente all'array non è un max-heap



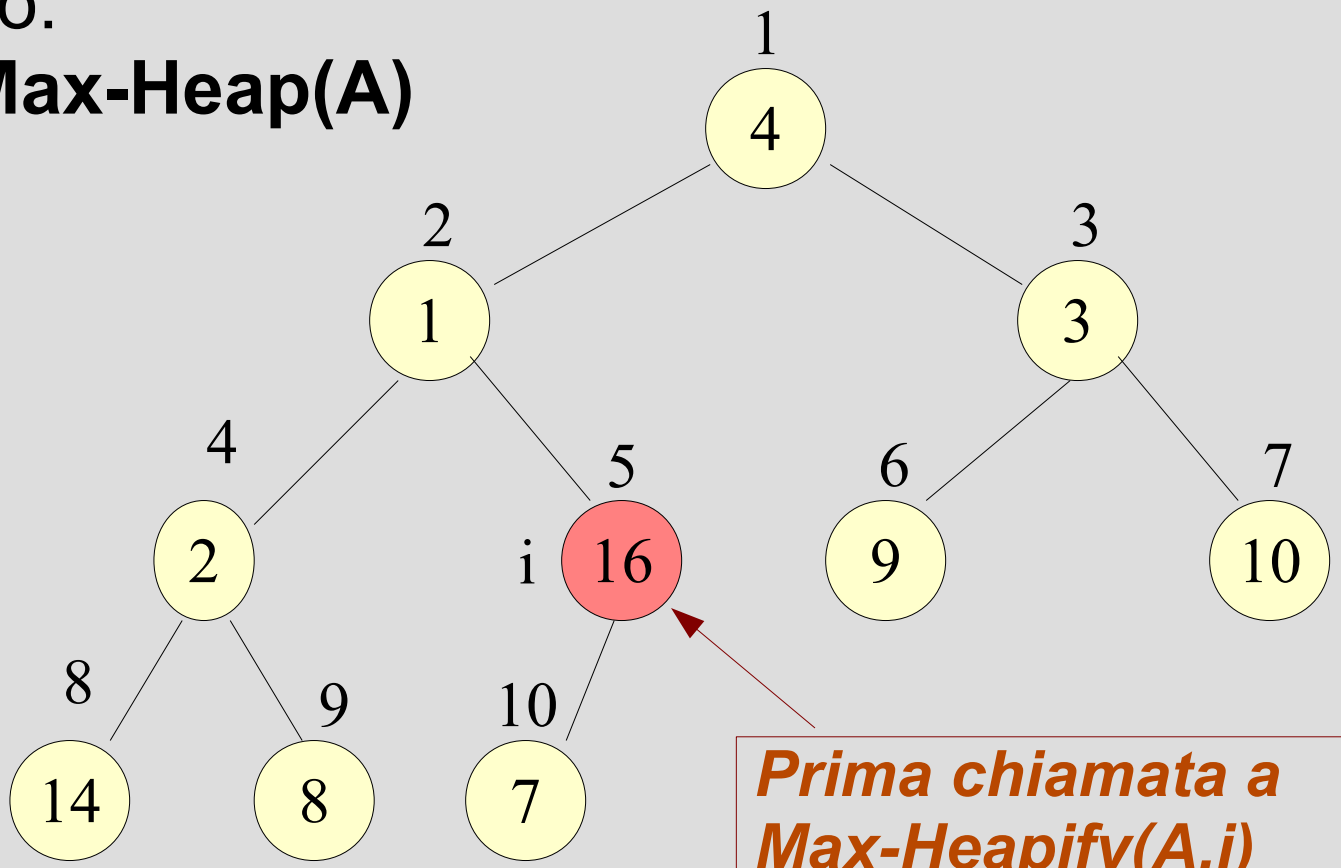
Build-Max-Heap

Algoritmo Build-Max-Heap

- Utilizzo della procedura **Max-Heapify** dal basso verso l'alto per costruire un max-heap
- I nodi con indice $[(\lfloor n/2 \rfloor + 1), ..., n]$ sono foglie dell'albero
- Tutti i restanti nodi sono **radici di potenziali max-heap** su cui chiamare la procedura **Max-Heapify**

Build-Max-Heap

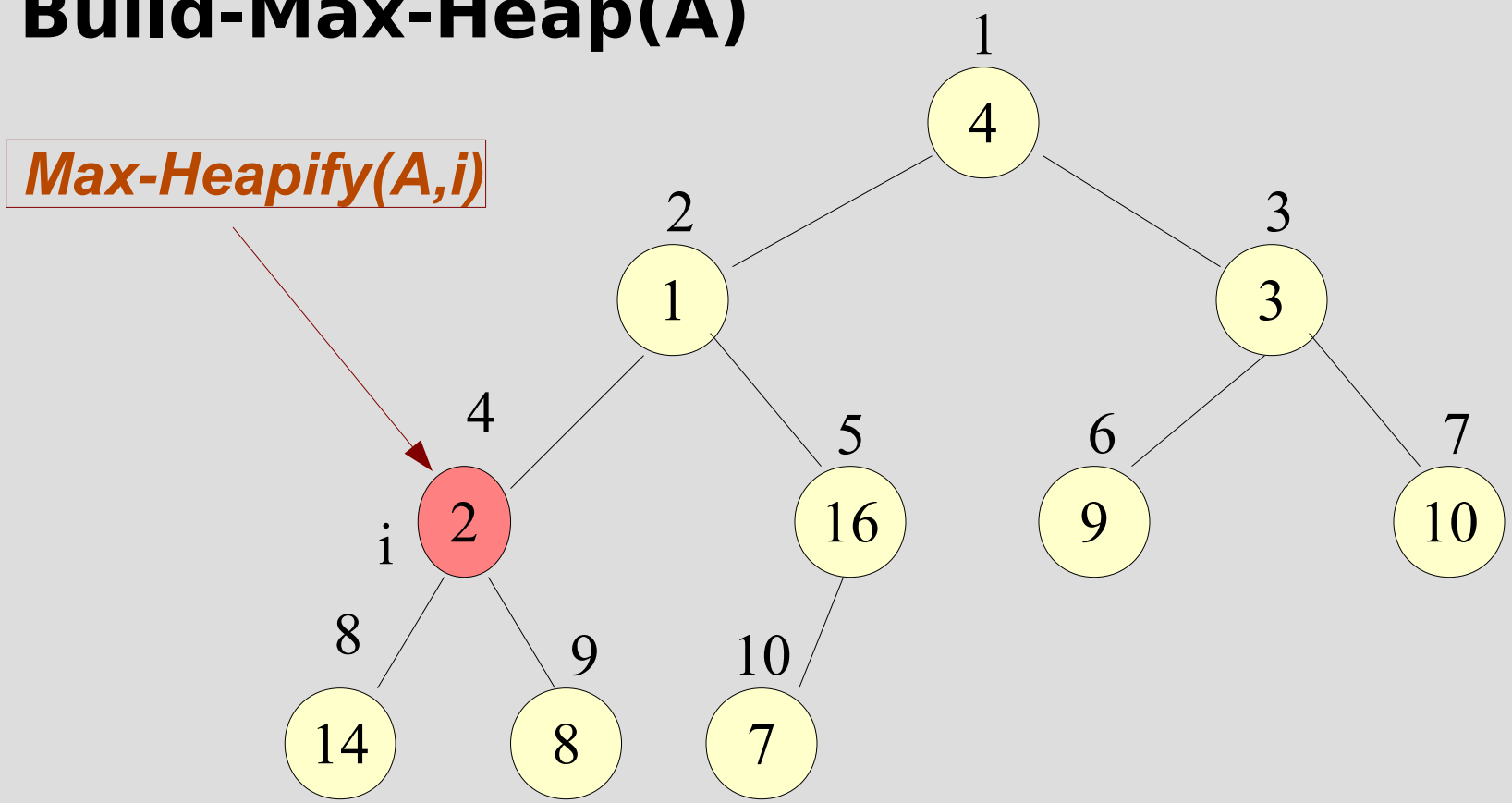
Esempio:
Build-Max-Heap(A)



Prima chiamata a Max-Heapify(A,i) sull'elemento i= $\lfloor n/2 \rfloor$

Build-Max-Heap

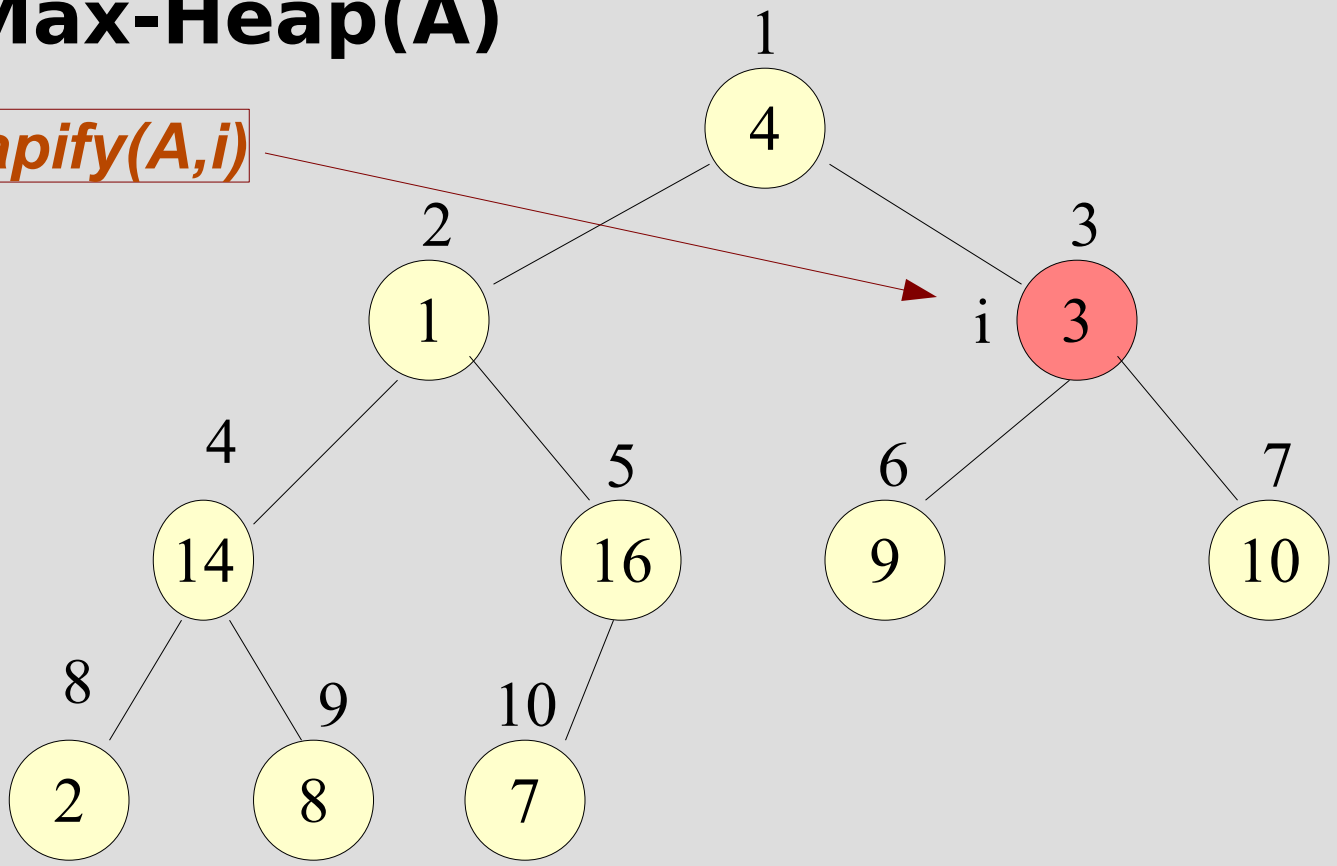
Build-Max-Heap(A)



Build-Max-Heap

Build-Max-Heap(A)

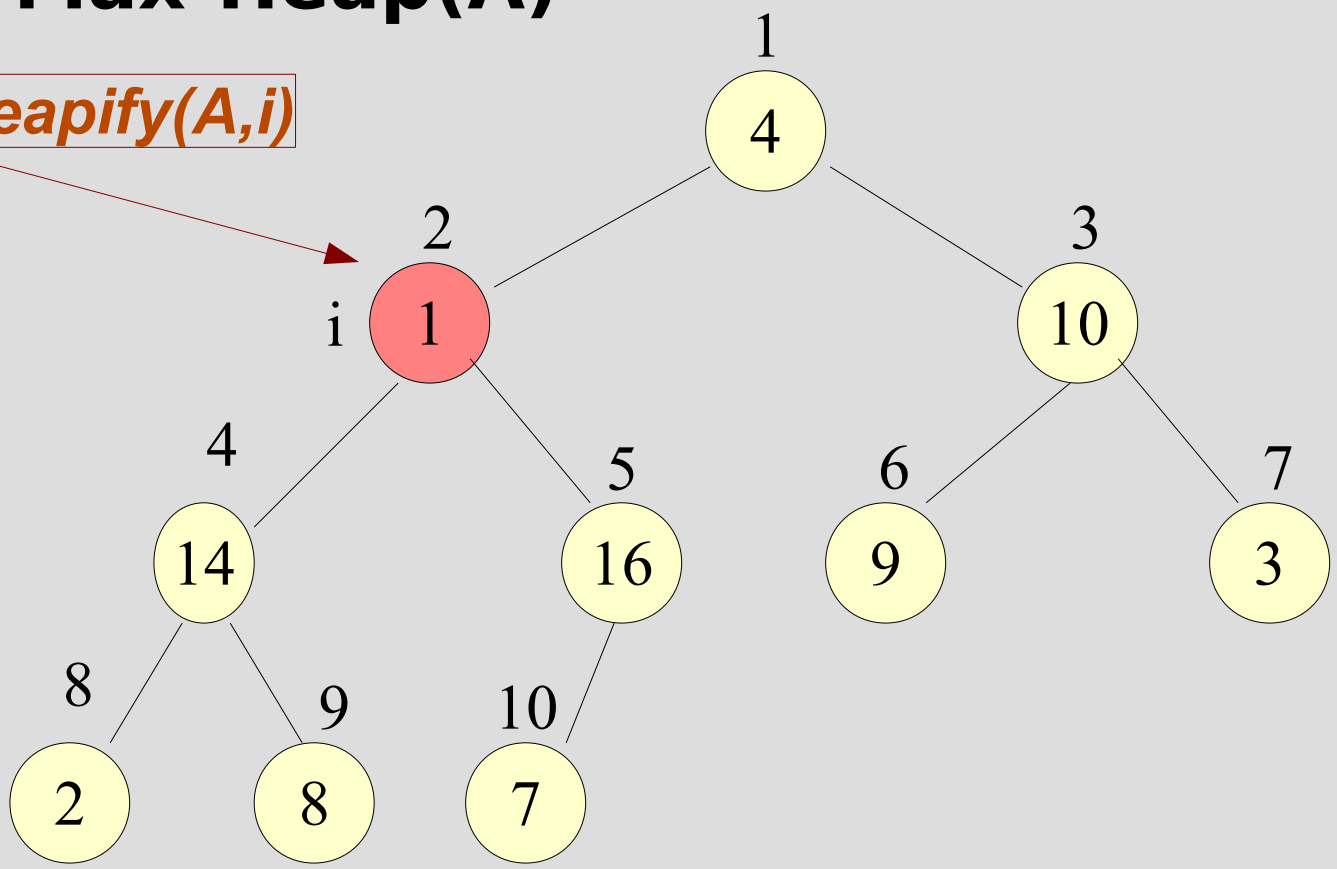
Max-Heapify(A,i)



Build-Max-Heap

Build-Max-Heap(A)

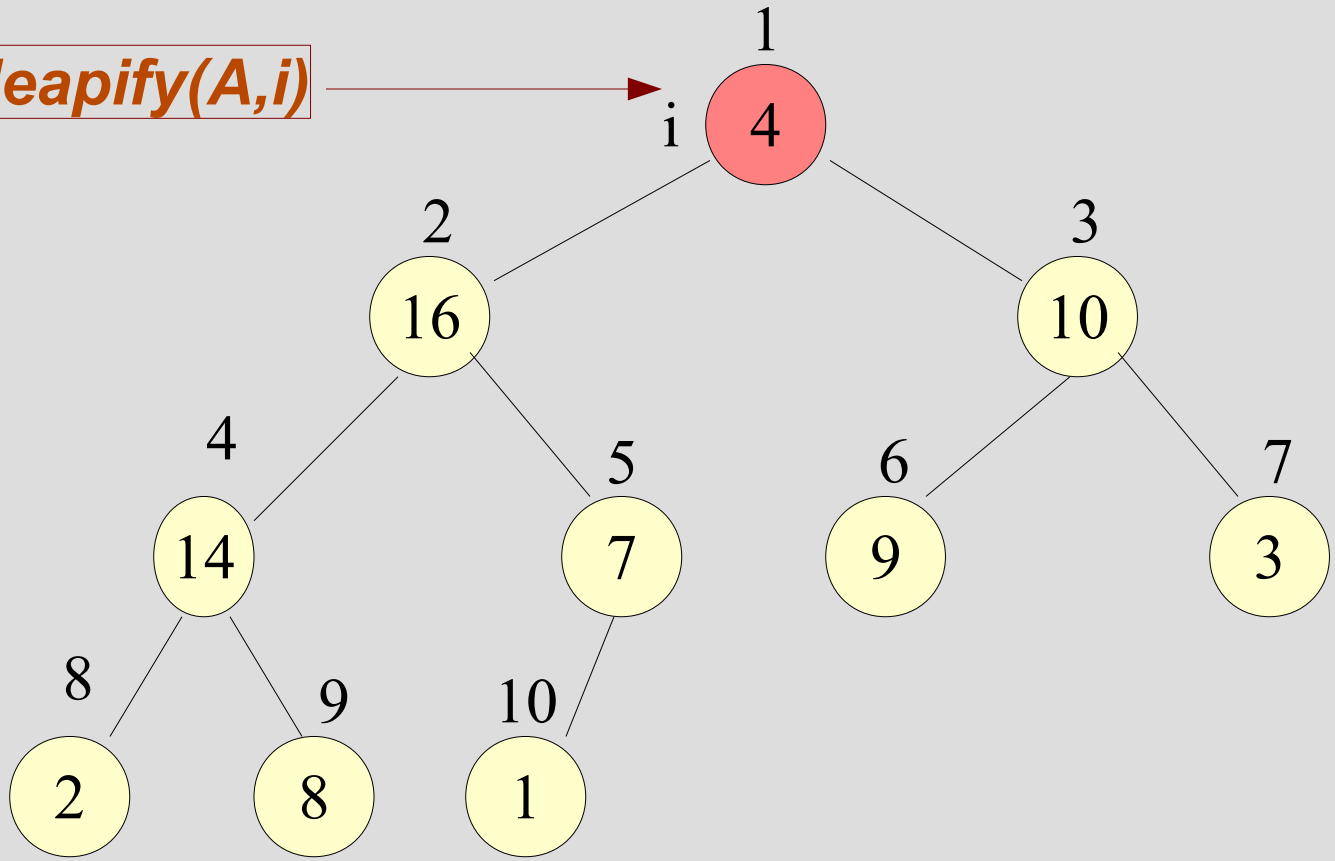
Max-Heapify(A,i)



Build-Max-Heap

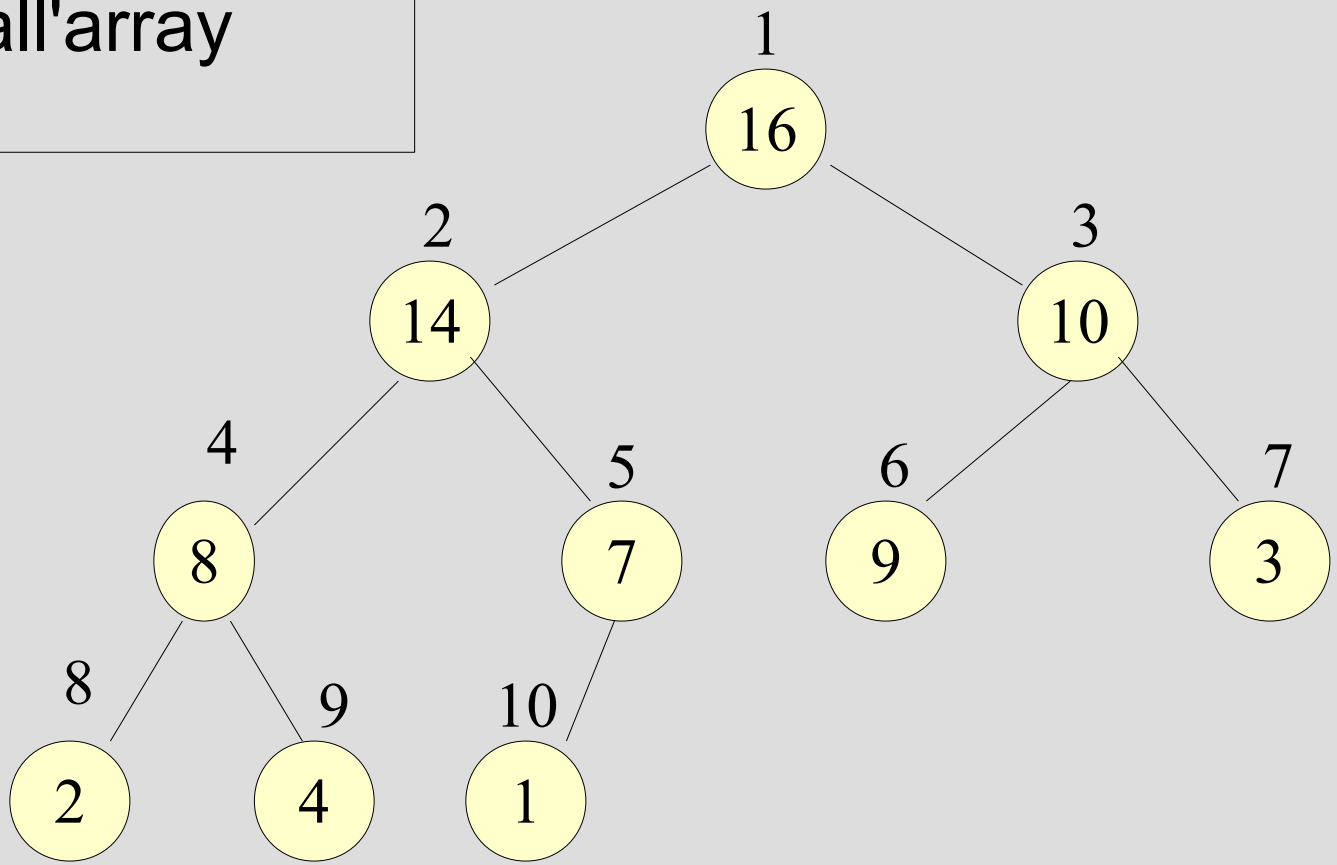
Build-Max-Heap(A)

Max-Heapify(A,i)



Build-Max-Heap

Risultato finale:
costruito un ***max-heap***
a partire dall'array
di input **A**



Programma

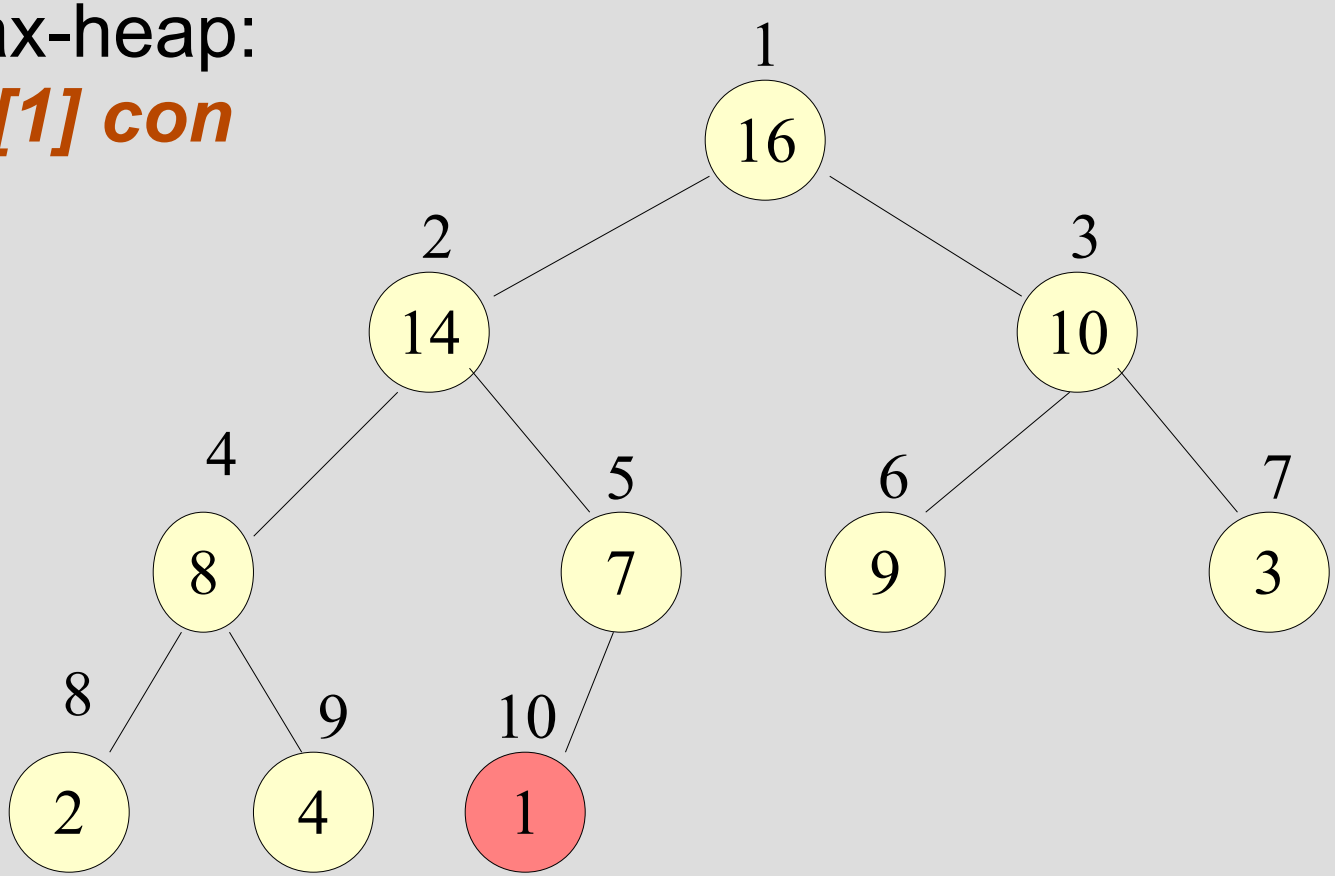
- *build-max-heap.cc*
- ***Procedura Build-Max-Heap:*** Costruisce un max-heap da un array di input di n elementi
- Si noti la necessità di usare correttamente la ***selezione con indice*** per muoversi all'interno del heap (la struttura dati è sempre un array!)
- ***Attenzione all'uso degli indici***
- **Suggerimento**: mantenere separatamente l'informazione sulla lunghezza dell'heap
  - **Usare la posizione 0 dell'array**

Ordinamento

- Algoritmo di ordinamento ***Heapsort***: ***ordinamento non decrescente***
- ***Chiamata iniziale a Build-Max-Heap*** per costruire un max-heap dall'array di input di n elementi da ordinare
- Alla fine l'elemento più grande è memorizzato nella radice A[1] → può essere inserito nella ***posizione finale corretta*** scambiandolo con A[n]

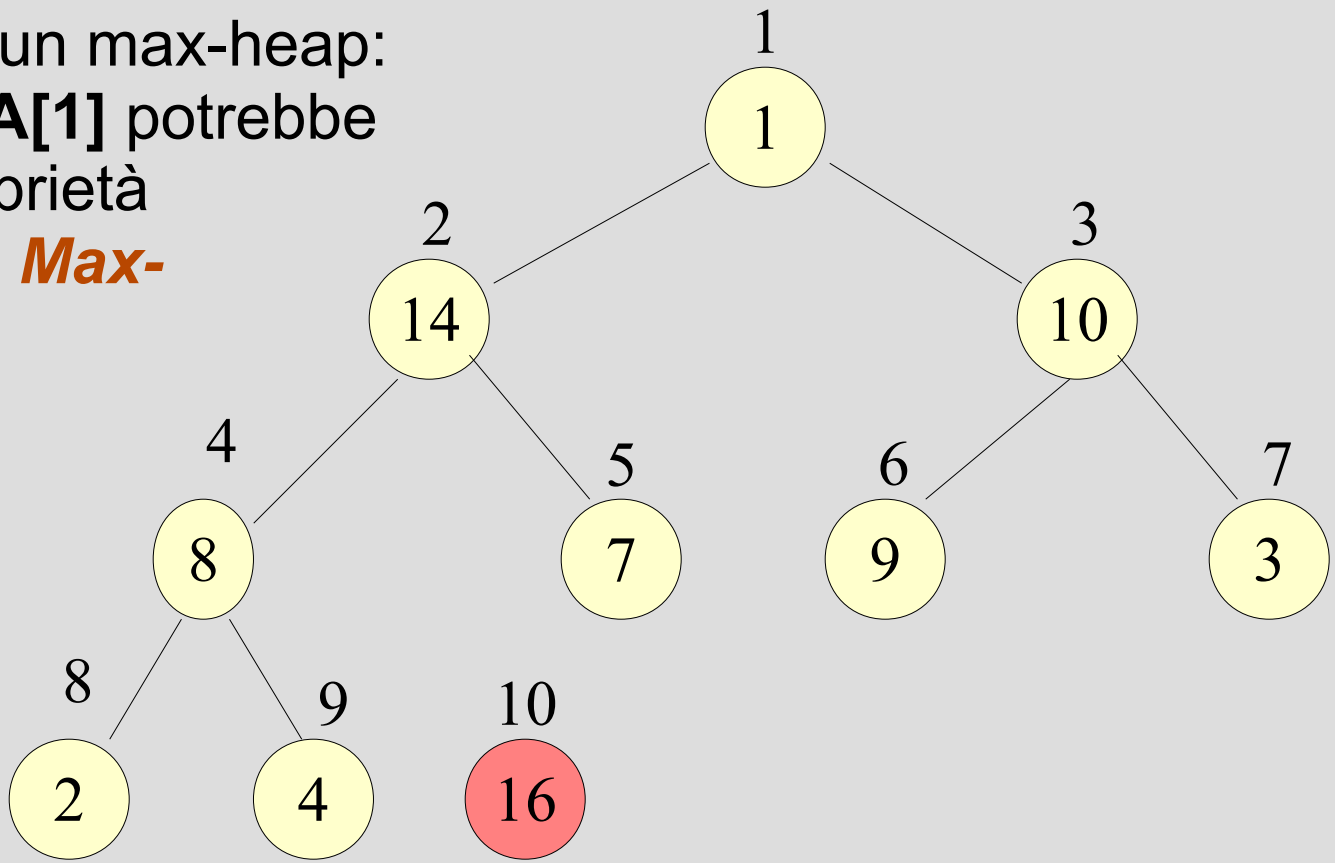
Heapsort

Azione di ***Heapsort*** sulla struttura max-heap: ***scambio A[1] con A[10]...***



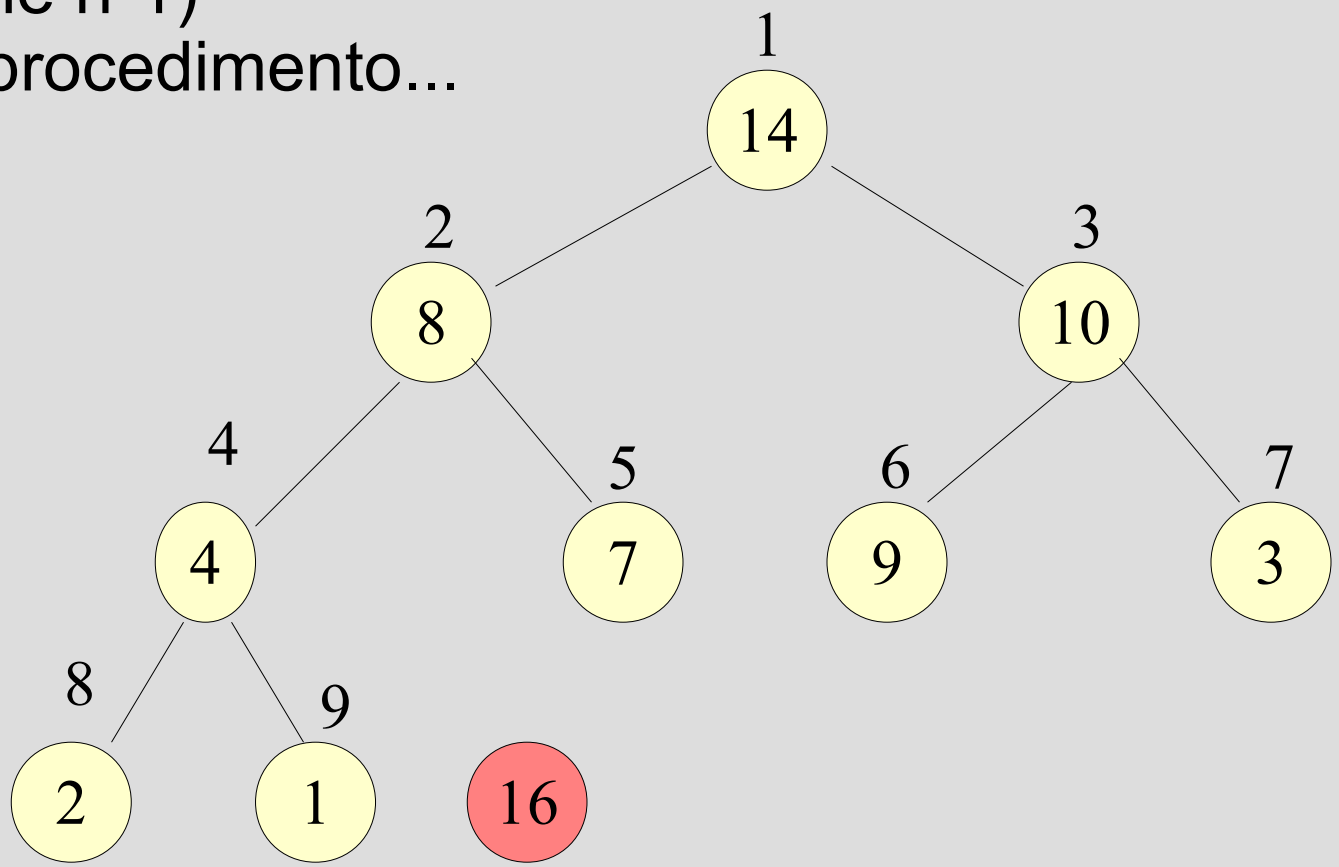
Heapsort

… e **stacco** l'elemento **A[10]**
Ora **A** può essere facilmente trasformato in un max-heap: solo la radice **A[1]** potrebbe violarne le proprietà
→ chiamata a Max-heapify(A,1)



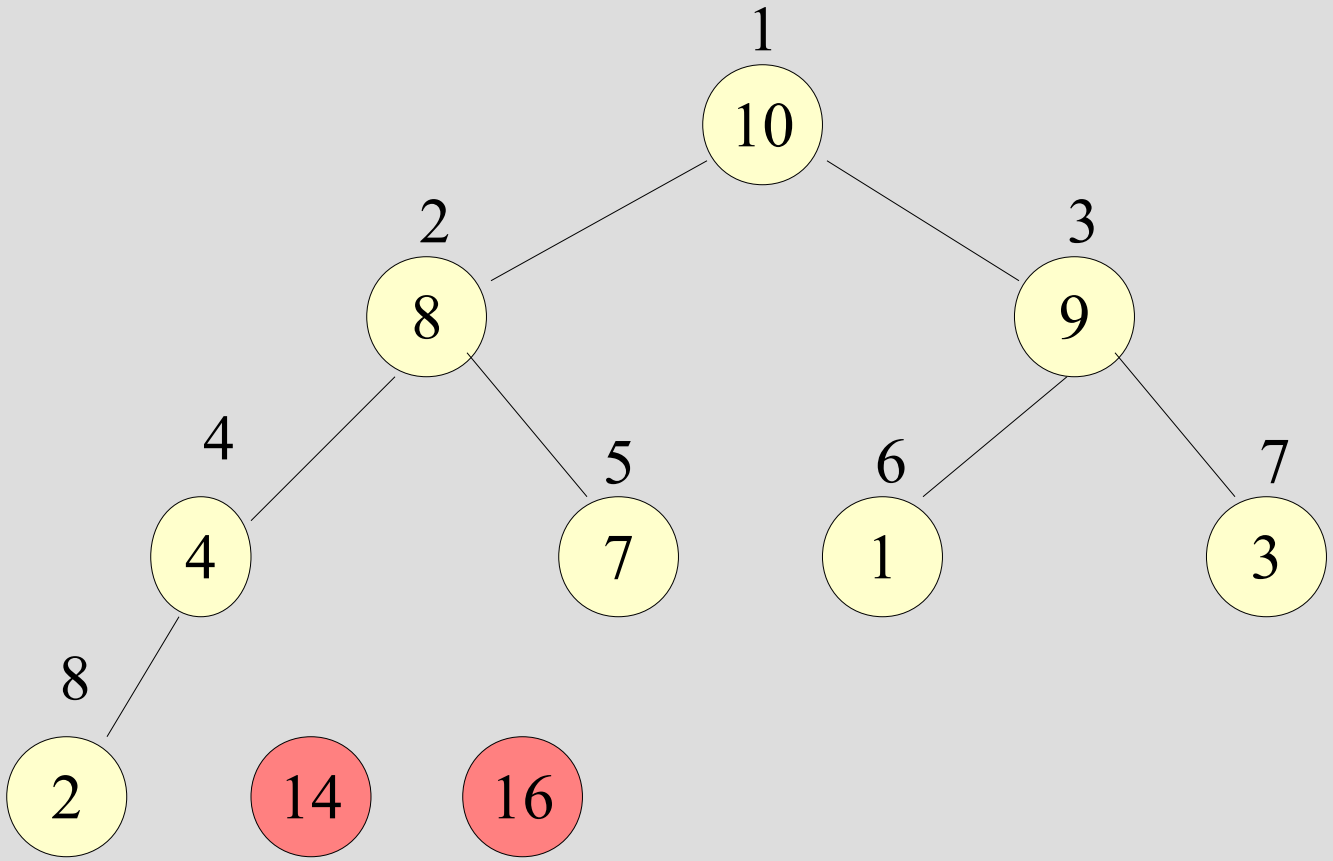
Heapsort

A è di nuovo un max-heap
(di dimensione n-1)
Ora ripeto il procedimento...



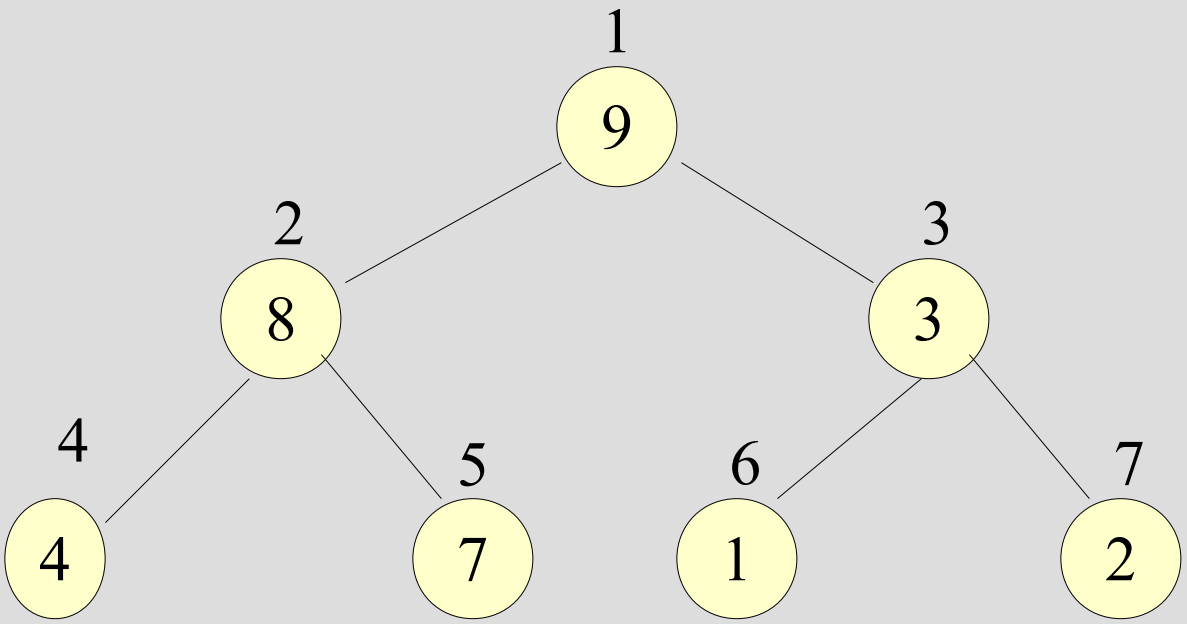
Heapsort

Max-heap di dimensione n-2...



Heapsort

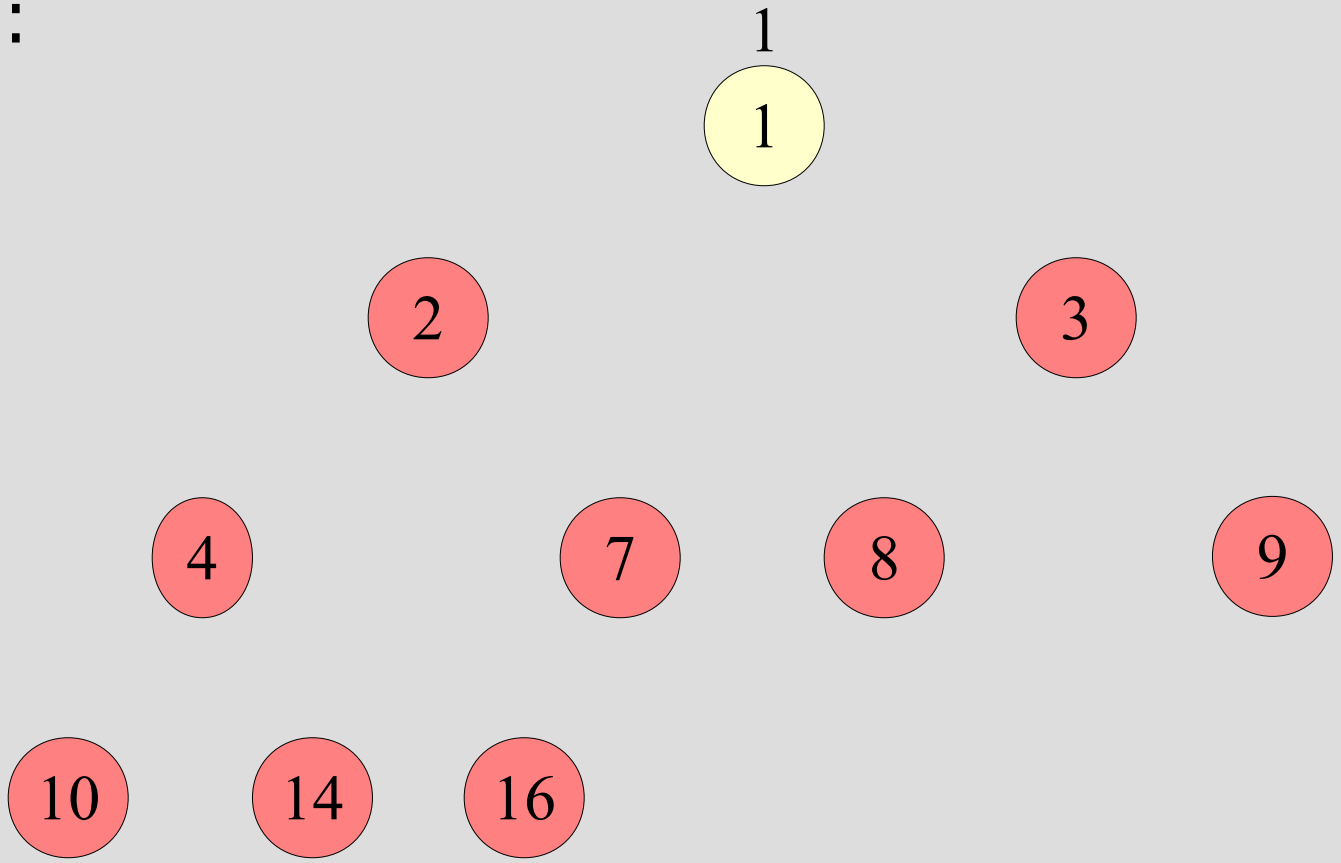
Max-heap di dimensione n-3...



10 14 16

Heapsort

... infine:



Programma

- *hs.cc*
- Programma di ordinamento che utilizza l'algoritmo **heapsort**
- NOTA: Utilizza la posizione 0 dell'array per memorizzare la lunghezza dell'heap

Estensione opzionale

- *hs_optio.cc*
- La parte opzionale viene realizzata attraverso le direttive al preprocessore

  #define, #ifdef, #else, #endif
- Effettua la lettura degli elementi da ordinare da file (nome del file passato al programma da riga di comando) .
- Scrive gli elementi già ordinati su un file "outfile"