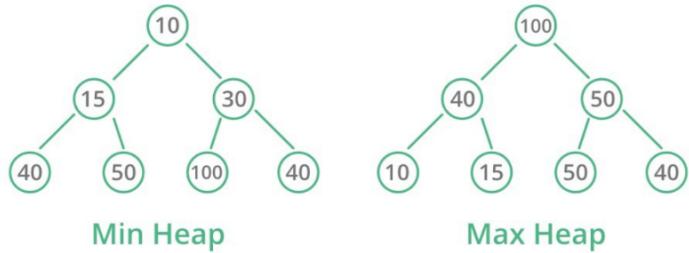


**HEAP** = albero binario completo (salvo al più l'ultimo livello)

↳ spesso utilizzato per implementare priority queues

↳ due tipi

minimo  
massimo



## HEAP MASSIMO

- per ogni nodo, il valore del figlio è  $\leq$  del suo stesso valore
- il valore della radice è il più grande

$$H.\text{key} \geq H.\text{left}.\text{key}$$

$$H.\text{key} \geq H.\text{right}.\text{key}$$

## HEAP MINIMO

- per ogni nodo, il valore del figlio è  $\geq$  del suo stesso valore
- il valore della radice è il più piccolo

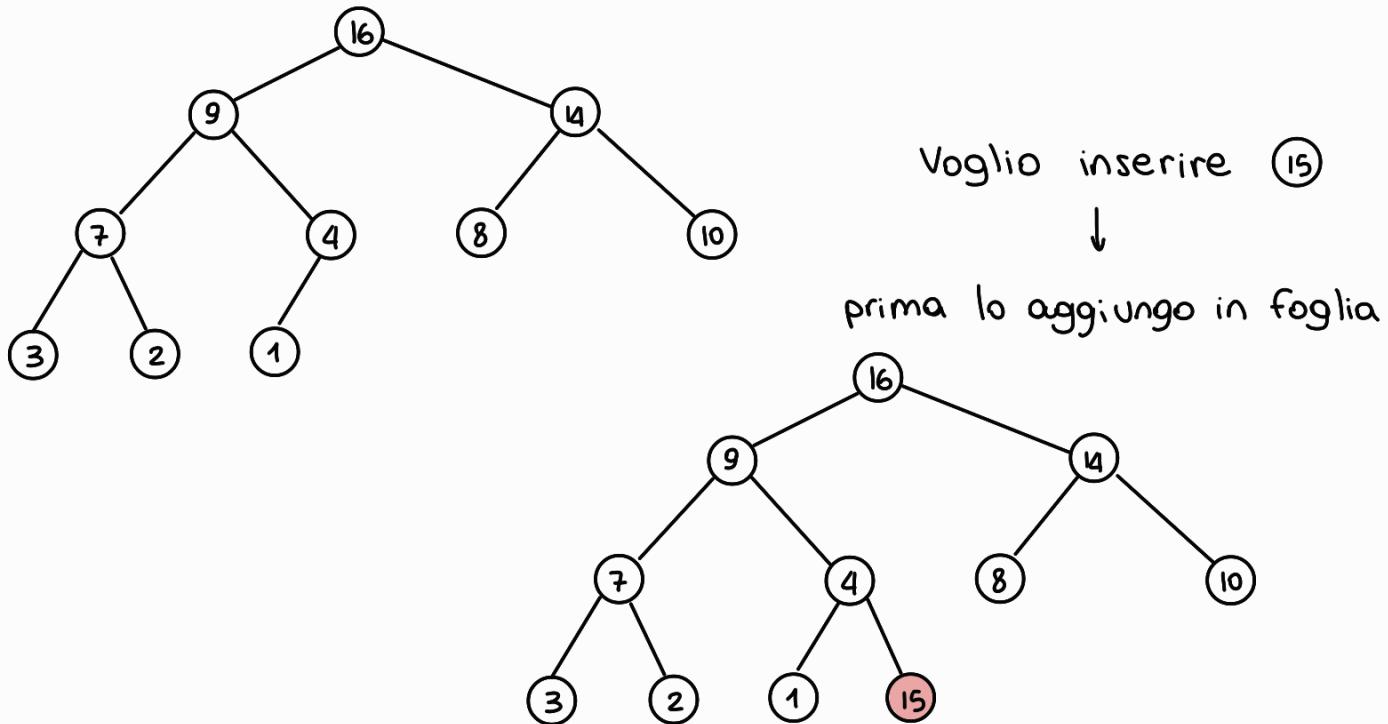
$$H.\text{key} \leq H.\text{left}.\text{key}$$

$$H.\text{key} \leq H.\text{right}.\text{key}$$

E' conveniente rappresentare lo heap con un array in cui :

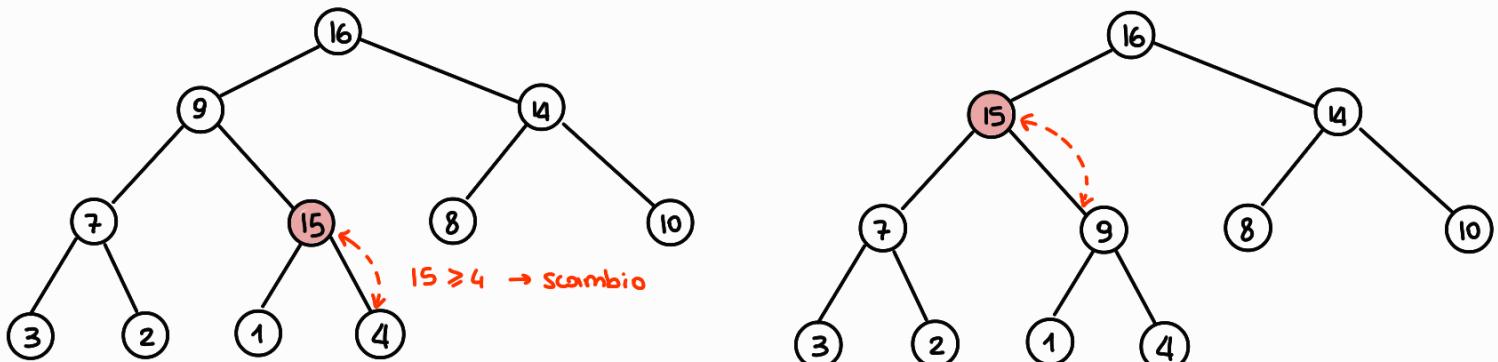
- radice =  $H[0]$
- figlio sx di  $H[i] = H[2i+1]$
- figlio dx di  $H[i] = H[2i+2]$
- genitore di  $H[i] = H[\lfloor i/2 \rfloor]$
- le foglie occupano il sottovettore  $H[\lfloor n/2 \rfloor + 1, \dots, n]$

## INSERIMENTO IN HEAP MAX



poi lo faccio risalire lungo il ramo finche' non ricostruisco lo heap

COME? Scambio  $x$  con  $x.parent$  finche'  $x > x.parent$



```

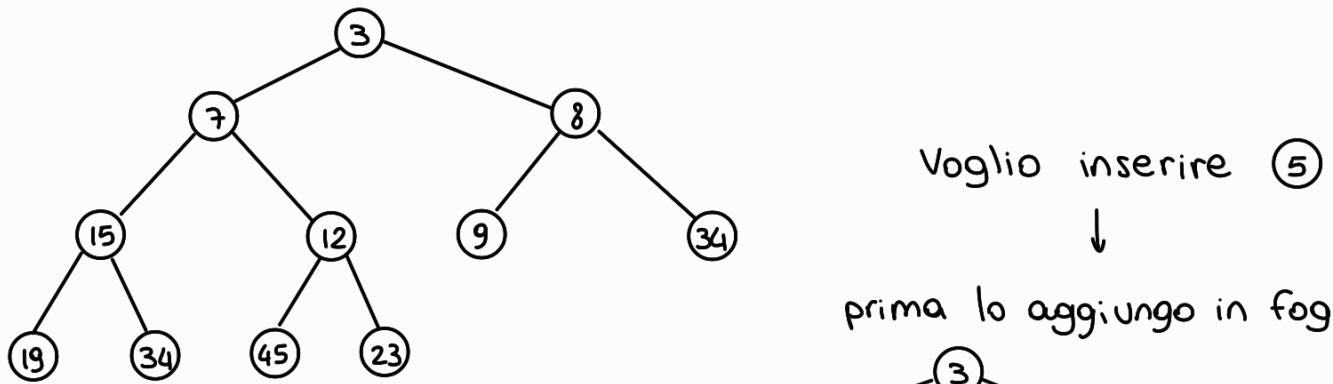
HEAPINSERT( $H, x$ )
  ▷ Pre:  $H$  è un heap
  ▷ Post:  $H$  è un heap con  $x$  inserito
 $H.N \leftarrow H.N + 1$ 
 $p \leftarrow H.N$ 
 $H[p] \leftarrow x$ 
while  $p > 1 \wedge H[p] > H[\text{PARENT}(H, p)]$  do
  scambia  $H[p]$  e  $H[\text{PARENT}(H, p)]$ 
   $p \leftarrow \text{PARENT}(H, p)$ 
end while

```

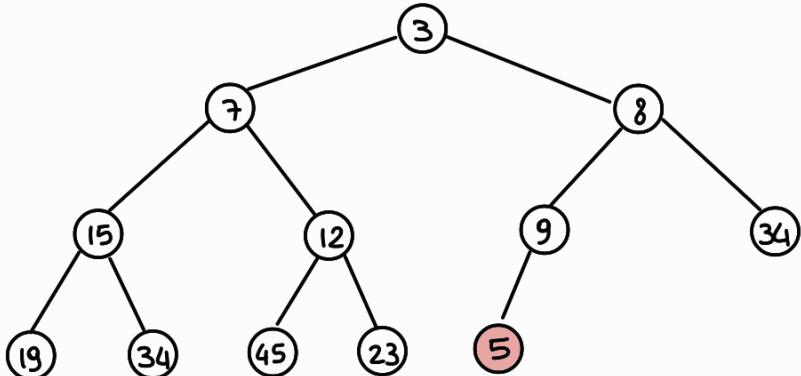
e'  $O(\log n)$

HEAPIFYUP

## INSERIMENTO IN HEAP MIN

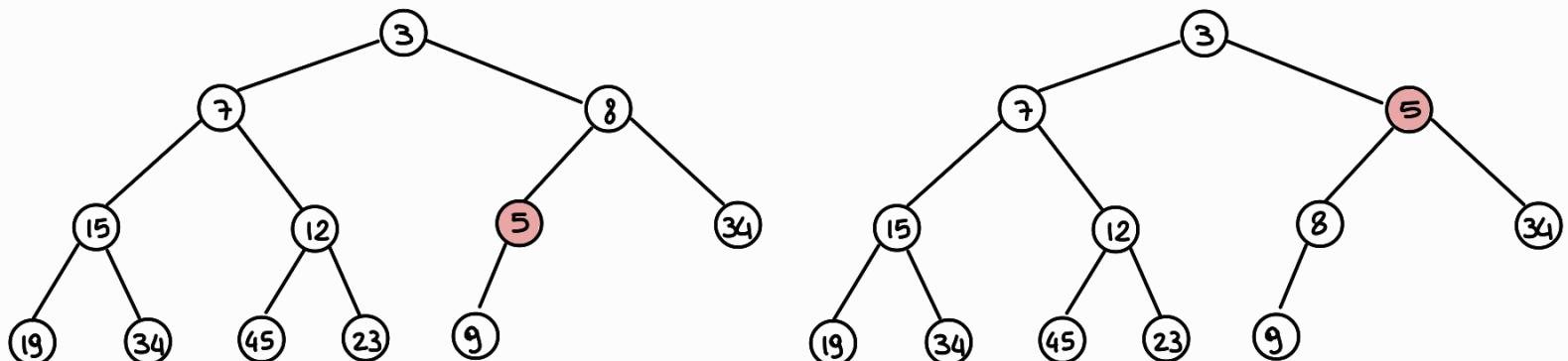


prima lo aggiungo in foglia



poi lo faccio risalire lungo il ramo finche' non ricostruisco lo heap

COME? Scambio x con x.parent finche'  $x < x.parent$



**HEAPINSERT( $H, x$ )**

▷ Pre:  $H$  è un heap

▷ Post:  $H$  è un heap con  $x$  inserito

$H.N \leftarrow H.N + 1$

$p \leftarrow H.N$

$H[p] \leftarrow x$

**while**  $p > 1 \wedge H[p] < H[\text{PARENT}(H, p)]$  **do**

scambia  $H[p]$  e  $H[\text{PARENT}(H, p)]$

$p \leftarrow \text{PARENT}(H, p)$

**end while**

e:  $O(\log n)$

HEAPIFY UP

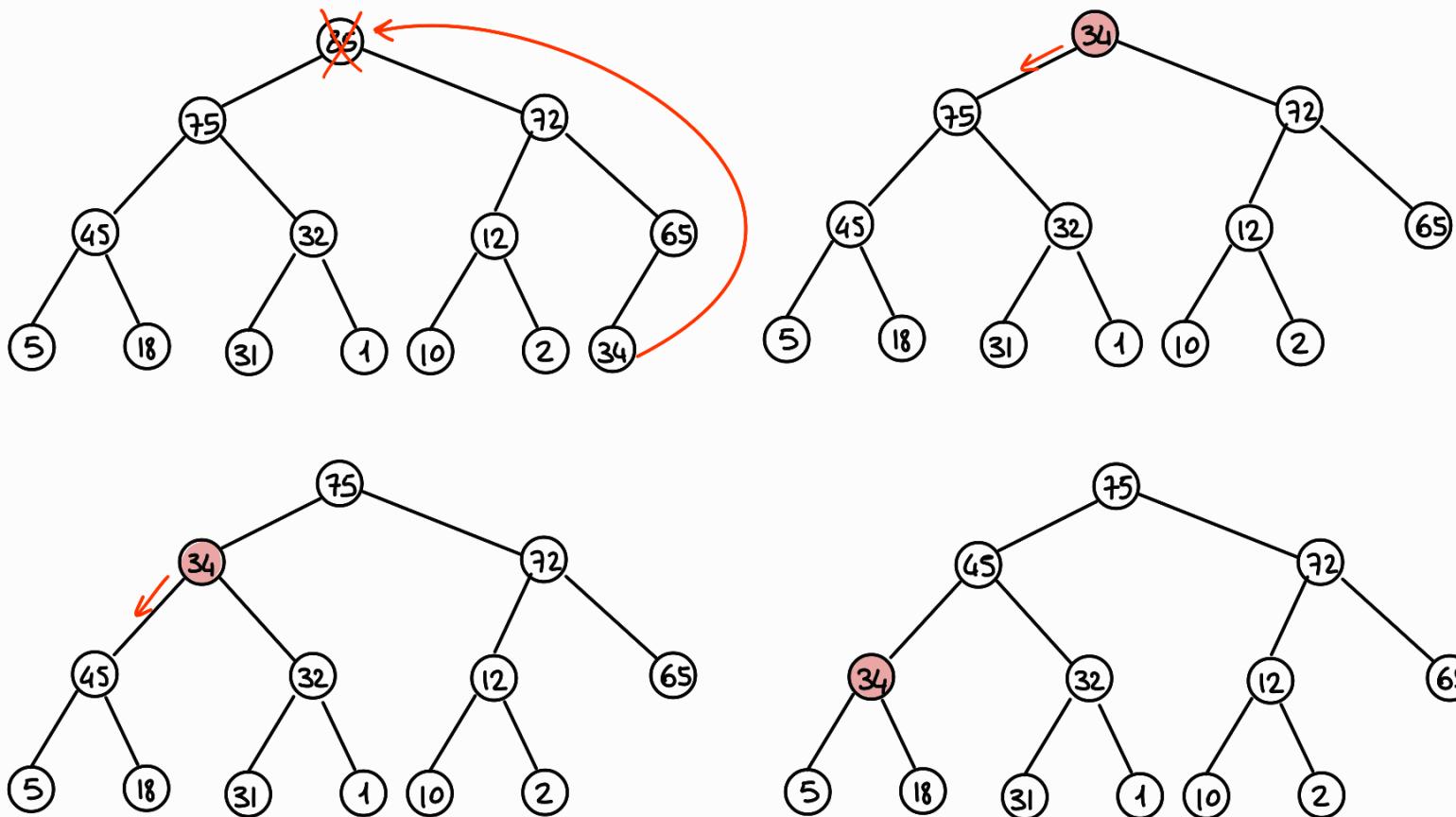
## ESTRAZIONE DA HEAP MAX

Toglie l'elemento dalla radice (quindi il massimo)

Due fasi :

- 1) l'elemento più a dx dell'ultimo livello rimpiazza la radice
- 2) l'elemento nuovo in radice viene fatto discendere lungo l'albero finché non è maggiore di entrambi i figli;

Scendo sempre verso il figlio maggiore.



## PSEUDO CODICE

`HEAPEXTRACT( $H$ )`

▷ Pre:  $H$  è un heap

▷ Post:  $H$  è un heap con etichetta massimo eliminata

$H[1] \leftarrow H[H.N]$

$H.N \leftarrow H.N - 1$

`HEAPIFY( $H, 1$ )`

entrambi  $\mathcal{O}(\log n)$

`HEAPIFY( $H, i$ )`

▷ Pre:  $1 \leq i \leq H.N$ , i sottoalberi con radice in  $\text{LEFT}(H, i)$  e  $\text{RIGHT}(H, i)$  sono heap

▷ Post: l'albero con radice in  $i$  è heap

$m \leftarrow \text{index of } \text{MAX}\{H[i], H[\text{LEFT}(H, i)], H[\text{RIGHT}(H, i)]\}$

`if  $m \neq i$  then`

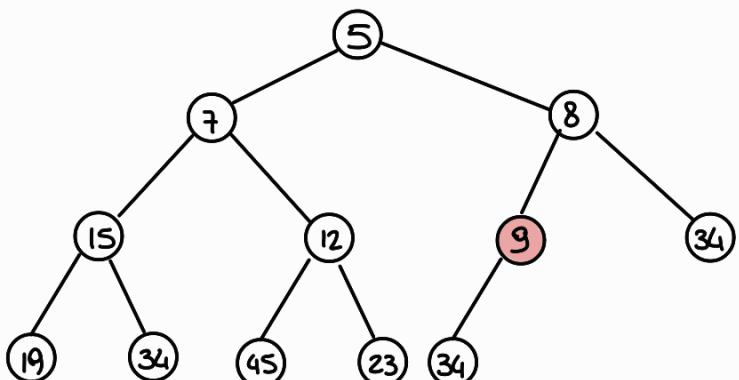
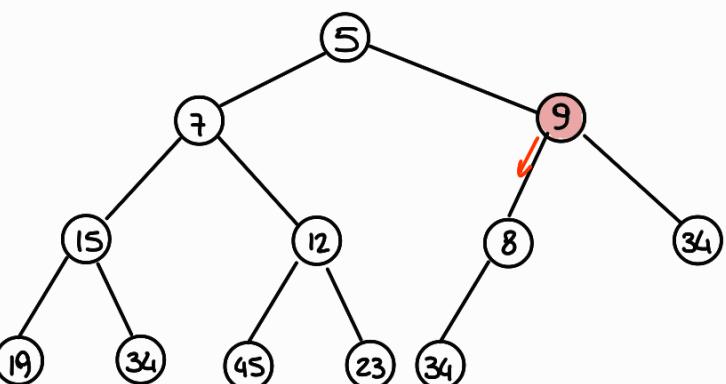
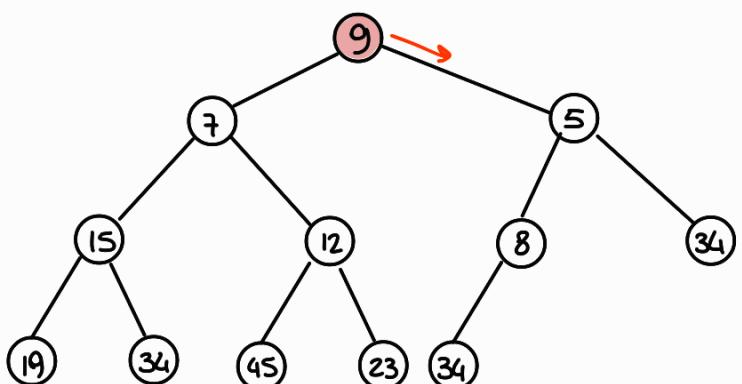
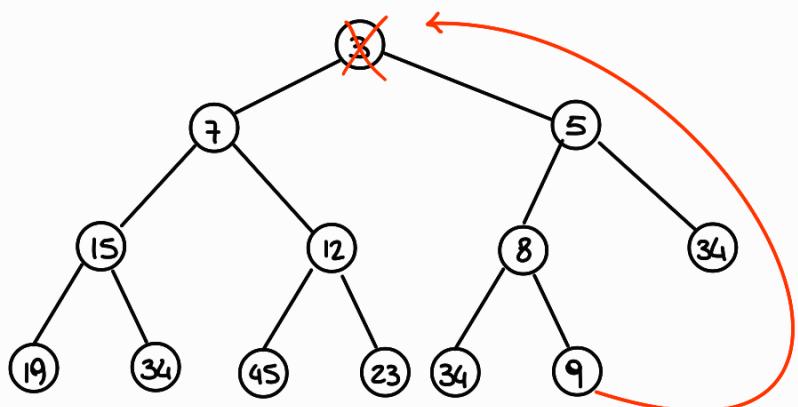
scambia  $H[m]$  e  $H[i]$

`HEAPIFY( $H, m$ )`

`end if`

## ESTRAZIONE DA HEAP MIN

Analogo all'estrazione da heap max, ma ovviamente sto estraendo l'elemento minimo e per lo scambio scelgo il minimo tra i figli.



`HEAPEXTRACT` è identico, cambia `HEAPIFY` :

`HEAPIFY( $H, i$ )`

- ▷ Pre:  $1 \leq i \leq H.N$ , i sottoalberi con radice in  $\text{LEFT}(H, i)$  e  $\text{RIGHT}(H, i)$  sono heap
- ▷ Post: l'albero con radice in  $i$  è heap

$m \leftarrow \text{index of } \text{MIN}\{H[i], H[\text{LEFT}(H, i)], H[\text{RIGHT}(H, i)]\}$

**if**  $m \neq i$  **then**

scambia  $H[m]$  e  $H[i]$

`HEAPIFY( $H, m$ )`

**end if**

Sempre  
 $O(\log n)$

## HEAP SORT

- il vettore  $H[N]$  rappresenta lo heap.
- Sfruttando l'estrazione del max si può ordinare un vettore  $V$  di  $M$  elem.

dove  $V = \{ \underbrace{V[0], V[1], \dots, V[N-1]}_H, V[N+1], \dots, V[M-1] \}$

IDEA : allargo la parte ordinata del vettore estraendo il max di  $H$ .

Riorganizzo  $V$  in modo che sia uno heap:

- gli elementi in posizione foglia sono già heap;
  - $V[i]$  è foglia se  $2i+1 > M$
  - le foglie sono in  $V[\lfloor M/2 \rfloor, \dots, M-1]$
- Itero Heapify partendo dalla posizione  $\lfloor M/2 \rfloor$  fino alla radice.

BUILDHEAP( $V$ )

▷ Pre:  $V$  è vettore di  $M$  elementi

▷ Post:  $V$  è uno heap

$V.N \leftarrow V.M$

for  $i \leftarrow \lfloor V.M/2 \rfloor$  down to 1 do

    HEAPIFY( $V, i$ )

end for

$O(n \log n)$

Ora che  $V$  è uno heap, lo ordino tramite estrazione del max

HEAPSORT( $V$ )

▷ Pre:  $V$  è un vettore di  $M$  elementi

▷ Post:  $V$  è ordinato

BUILDHEAP( $V$ )

for  $i \leftarrow V.N$  down to 2 do

    scambia  $V[i]$  e  $V[1]$

$V.N \leftarrow V.N - 1$

    HEAPIFY( $V, 1$ )

end for

$O(n \log n)$

Con Heap minimo, estraggo il min della radice e lo inserisco nella parte finale dell'array ordinato.

## RIPRISTINARE LO HEAP DOPO RIMOZIONE

HEAPIFY-DOWN (corrisponde a HEAPIFY visto prima)

## RIPRISTINARE LO HEAP DOPO INSERIMENTO      HEAPIFY-UP

Per Heap max:

HEAPIFY UP( $H, i$ )

$p \leftarrow H[i].parent$

if  $p \geq 0 \wedge H[i] > H[p]$  then

scambia  $A[i] \leftrightarrow A[p]$

HEAPIFYUP( $A, p$ )

Per Heap min:

HEAPIFY UP( $H, i$ )

$p \leftarrow H[i].parent$

if  $p \geq 0 \wedge H[i] \leq H[p]$  then

scambia  $A[i] \leftrightarrow A[p]$

HEAPIFYUP( $A, p$ )

## IMPLEMENTAZIONE DI UNA CODA DI PRIORITA'

Funzione PRIORITA' : codifico ogni elemento come una coppia  
<elemento, priorita'>

Sfrutto lo heap in base alla chiave / priorita'.

La funzione e' DISTRUTTIVA  $\rightarrow$  non richiede HEAPIFY  $\rightarrow$  e'  $O(1)$

CHANGEPRIORITY ( $H, element, newPriority$ )

$i \leftarrow \text{FINDINDEX}(H, element)$

if  $i$  non trovato then return

$H[i].priority = newPriority$

$p \leftarrow H[i].parent$

if  $p \geq 0 \wedge H[i].priority < H[p].priority$  then

HEAPIFYUP( $H, i$ )

else

HEAPIFYDOWN( $H, i$ )

$O(\log n)$  se FINDINDEX sfrutta le Hash