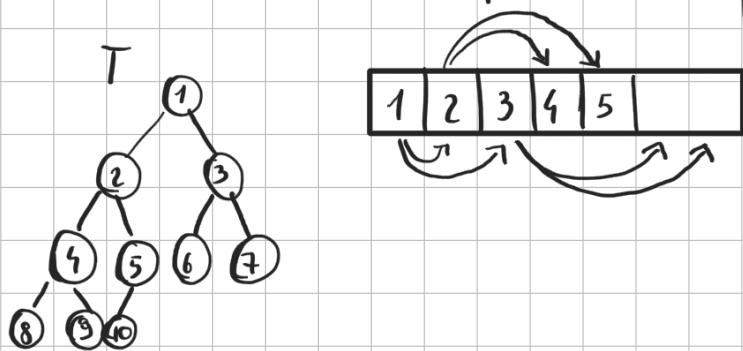


## "CONVERSOINE" DI UN ALBERO COMPLETO IN ARRAY

Vediamo stato un albero completo come implementarlo come array



Secondo quest'ordinamento la radice di  $T$  e' in posizione 1

$$\text{Vic A} \quad \text{figlio-sx}(i) = 2i$$

$$\text{figlio-dx}(i) = 2i+1$$

Abbiamo quindi la corrispondenza tra array e albero

**HEAPIFY** funzione implementata

**HEAPIFY(A, i)**

$$j = 2i$$

$$k = 2i+1$$

IF  $j \leq \text{HEAPSIZE}[A]$  AND  $A[i] < A[j]$  THEN

$$\text{MAX} = j$$

ELSE

$$\text{MAX} = i$$

$i$  e' la pos. della radice

$j$  e' la posizione del figlio sx

"

dx

HEAPSIZE\_A mantiene la fine

dell'heap. Vediamo se  $j$  sta

nell'heap e se e' piu' grande  
della radice

IF  $k \leq \text{HEAPSIZE}[A]$  AND  $A[k] > A[\text{MAX}]$  THEN

$\text{MAX} = k$

IF  $\text{MAX} \neq i$  THEN

$\text{SWAP}(A, i, \text{MAX})$

$\text{HEAPIFY}(A, \text{MAX})$

se  $\text{MAX} = i$  allora la radice e' in posizione giusta  
e, siccome abbiamo supposto i due sott'alberi  
heap, non dobbiamo fare nulla. Altrimenti:  
Aggiustiamo la radice e richiamiamo  $\text{HEAPIFY}$

Si ricordi che nell'HEAP ci sono solo  
elementi non ordinati.

Si noti che cosi' l'algoritmo non ci aiuta molto perch' partendo  
dalla radice non ordina tutto l'albero, bensì solo un percorso oppure  
se il primo modo e' ordinato non controllera' mai gli altri.

(ciò che pero' ci aiuta a fare e' convertire un albero completo  
in heap (sotto l'ipotesi che i sott'alberi già lo siano))

Possiamo vedere le foglie come degli heap minimi; prendendo due  
fratelli, l'albero che compongono con il padre risulta essere un  
albero completo che ha come sott'alberi due heap, quindi per  
definizione possiamo applicare  $\text{HEAPIFY}$  al padre (come forse la

radice) e costruire il nostro piccolo heap. Controllando entrambi i figli per avere il nostro albero ordinato. Analogamente faremo con i fratelli dei primi due nodi (quindi le altre foglie).

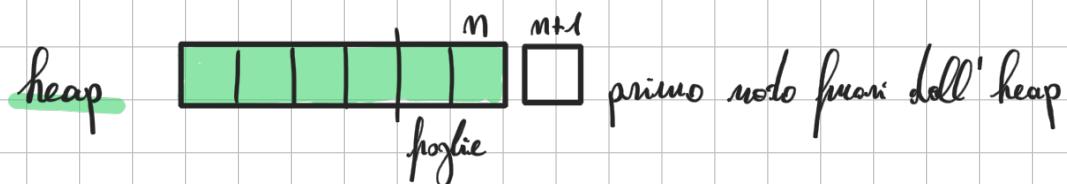
Tenuto il livello delle foglie soliamo di un livello e iteriamo e così via per tutto l'albero scendendo.

Non abbiamo la certezza che tutti i nodi siano ordinati, ma con questo procedimento prendiamo il massimo di ogni livello e lo mandiamo su fino a quando arrivare alla radice.

Quando avremo il massimo alla radice possiamo scambiarlo con l'ultima foglia (che è l'ultimo elemento dell'array).

A questo punto sicuramente l'ultima foglia è in posizione corretta rispetto all'array (massimo alla fine) quindi possiamo ripetere il processo escludendo l'ultima foglia (l'ultimo elemento dell'array) fino ad ottenere l'albero ordinato.

Vediamo come si riflette sull'array:

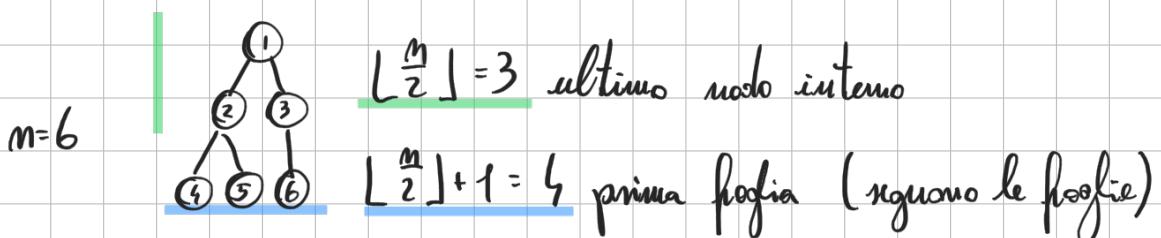


L'elemento in posizione  $m+1$  è il primo fuori dall'heap (e' ordinato)

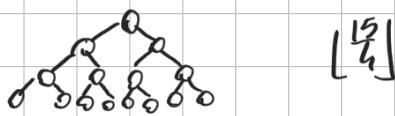
Sai noti che l' heap e' costituito solo da nodi non interni perché non hanno figli (sono fuori dall' heap tecnicamente) e in particolare heapify non verrà mai chiamato sulle foglie (come abbiamo già visto). Come possiamo trovare i nodi che non sono foglie?

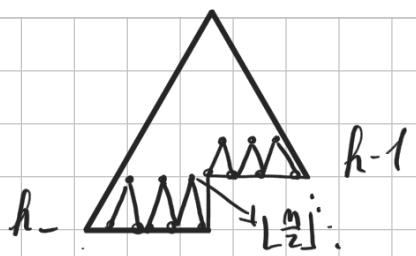
Sappiamo che  $m+1$  e' figlio di un nodo in posizione  $\lfloor \frac{m+1}{2} \rfloor$  (per come vengono sistemati i nodi nell' array) (sarà il floor perché non sappiamo se  $m+1$  è pari o dispari, quindi potrebbe essere figlio sinistro o destro).

Da  $1$  a  $\lfloor \frac{m}{2} \rfloor$  abbiamo tutti i nodi interni (perché presso qualsiasi nodo in quel range arriviamo massimo a  $m$  individuando i suoi figli con  $2m+1$  e  $2m$ ). Quindi da  $\lfloor \frac{m}{2} \rfloor + 1$  in poi sono foglie (usiamo il floor perché potrebbe essere dispari), si può verificare graficamente



Dovremo applicare heapify solo ai nodi da  $1$  a  $\lfloor \frac{m}{2} \rfloor$ .





Sappiamo che non dobbiamo applicare heapify alle foglie, quindi daremo partire dai nodi (padri) in posizione  $\lfloor \frac{m}{2} \rfloor$ .

Applicato a tutti i padri potremo "eliminare" le foglie, a questo punto lo applichiamo ai padri dei nostri ex-padri (attraverso foglie dopo l'eliminazione) ma se prima i padri erano in posizione  $\lfloor \frac{m}{2} \rfloor$ , i nuovi padri sono in posizione  $\left\lfloor \frac{\lfloor \frac{m}{2} \rfloor}{2} \right\rfloor$ . Si noti che HEAPIFY va applicato a tutti i padri quindi bisogna scorrere in orizzontale il livello che dal punto di vista dell'array significa decrementare la posizione di 1 (perché vengono distribuiti da sinistra verso destra in ordine crescente dell'array, quindi a ritroso decrementeremo di 1)



Vediamo in definitiva l'algoritmo che permette ciò e come viene implementato in heapsort:

## ● BUILDHEAP (A, m)

HEAPSIZE[A] = m

FOR  $i=\lfloor \frac{m}{2} \rfloor$  DOWN TO 1 DO

HEAPIFY(A, i)

Costruisce lo heap e metti il massimo in 1

Scans il livello dei padri

Applica heapify ad ogni padre.

## ● HEAPSORT (A, m)

BUILDHEAP(A, m)

FOR  $i=m$  DOWN TO 1 DO Partendo dall' ultimo nodo a sinistro

SWAP(A, i, 1)

Il massimo e' in 1, mettilo nell' ultimo  
nodo (m) ed "escludilo" (heapsize parte da m)

HEAPSIZE[A] --

HEAPIFY (A, 1)

Porta il massimo in 1

## COMPLESSITÀ COMPUTAZIONALE HEAP SORT

La complessità di HEAPIFY dipende direttamente dall' altezza

dell' albero e nell' algoritmo abbiamo tutte operazioni costanti o

lineari quindi  $T_{HF}(h) = O(h)$ . Ricordiamo che  $h = \log_2 m$  con m nodi

quindi  $T_{HF}(m) = O(\lg_2 m)$

Vediamo BUILDHEAP, il ciclo per viene ripetuto  $\frac{m}{2}$  volte chiamando HEAPIFY ogni volta. Ci si presenta un problema, non sappiamo la diretta relazione tra  $m$  e gli alberi che passano a HEAPIFY.

Sicuramente però sarà minore di una stima per eccesso, ovvero se passassimo ogni volta tutto l'albero (che in realtà venire fatto solo una volta solido, ovvero alla fine)

$$T_{BH}(m) \leq \sum_{i=1}^{\frac{m}{2}} T_{HF}(m) = O(m \lg m)$$

Vediamo meglio come si comporta HEAPIFY (all'interno di BUILDHEAP)

Partendo dal basso assumiamo che le foglie sono circa  $\frac{m}{2}$  e i loro padri quindi  $\frac{m}{4}$ . Bisogna vedere l'altezza dell'albero che stiamo trattando perché il tempo come abbiamo visto dipenderà direttamente da questo:

avranno  $\frac{m}{4}$  chiamate su alberi  $h=1$

$\frac{m}{8}$  chiamate su alberi  $h=2$  (perché va sempre fino in fondo)

Il tempo impiegato sarà dato dal numero di chiamate per l'altezza dell'albero, quindi:

$$T_{BH}(m) = \sum_{i=1}^h \left( \frac{m}{2^{i+1}} i \right) = \frac{m}{2} \sum_{i=1}^h \frac{i}{2^i}$$

