Algoritmi e Strutture Dati

Lezione 15

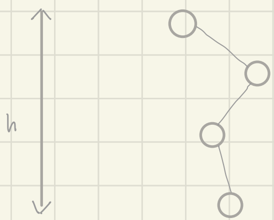
27 ottobre 2025

Heapsort

Alberi binari: $\underbrace{\text{n° nodi}}_{n}$ vs $\underbrace{\text{altezza}}_{h}$

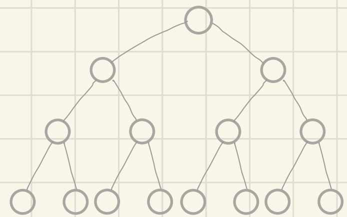
Che relazioni ci sono tra numero di nodi e altezza in un albero binario?

Alberi binari: $\underbrace{\text{n° nodi}}_{n}$ vs $\underbrace{\text{altezza}}_{h}$



$h+1$

albero degenere



$2^{h+1}-1$

albero completo

$$h+1 \le n \le 2^{h+1}-1$$

$$h \le n-1$$

$$h < n$$

$$n < 2^{h+1}$$

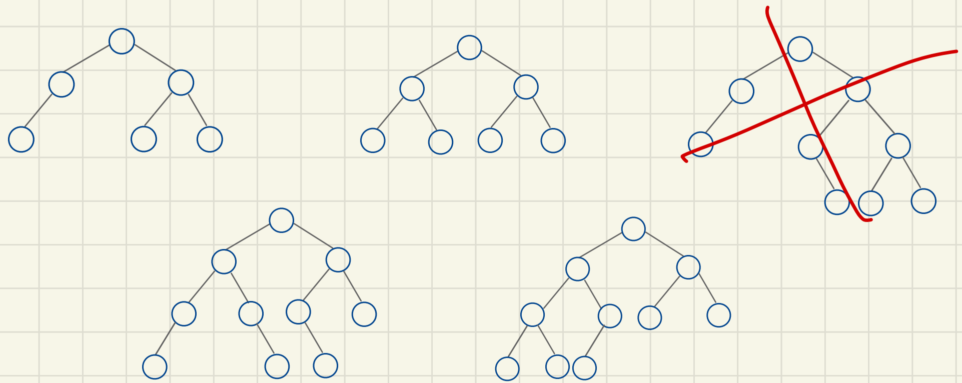
$$\log_2 n < h+1$$

$$\lfloor \log_2 n \rfloor \le h$$

$$\lfloor \log_2 n \rfloor \le h < n$$

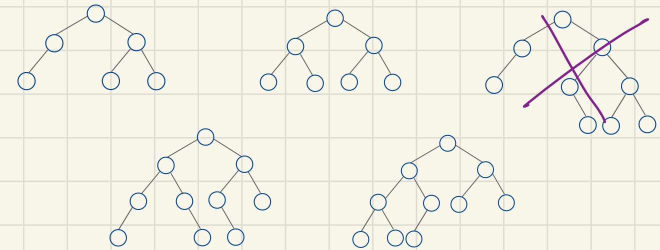
Alberi binari "quasi completi"

Un albero binario è QUASI COMPLETO quando è completo almeno fino al penultimo livello | Definizione



Alberi binari "quasi completi"

Un albero binario è QUASI COMPLETO quando è completo almeno fino al penultimo livello



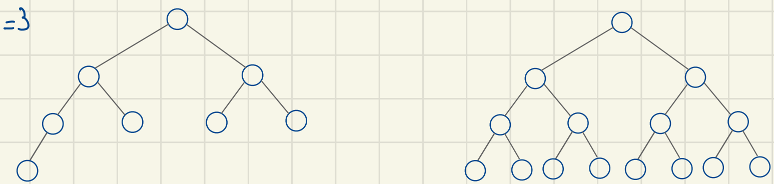
Proprietà:

Un albero binario di altezza h è quasi completo se e solo se ogni nodo di profondità $< h-1$ possiede ENTRAMBI i figli

Alberi binari quasi completi: $\underbrace{\text{n° nodi}}_{n}$ vs $\underbrace{\text{altezza}}_{h}$

fissiamo h

es. $h=3$



\# minimo di nodi — Albero completo fino a prof. $h-1$ + almeno 1 nodo di prof. h

$$\begin{array}{r} 2^h-1 \\ + \\ 1 \\ \hline 2^h \end{array}$$

\# max di nodi — Albero completo di prof. h

$$2^{h+1}-1$$

$$2^h \le n \le 2^{h+1}-1$$

$$2^h \le n < 2^{h+1}$$

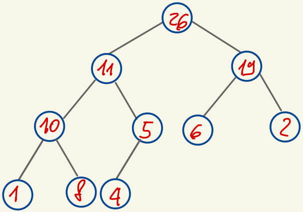
$$h \le \log_2 n \qquad \log_2 n < h+1$$

$$h = \lfloor \log_2 n \rfloor$$

Heap

Definizione

Uno *heap* (o *max-heap*) è un albero binario quasi completo in cui la chiave contenuta in ciascun nodo è maggiore o uguale delle chiavi contenute nei figli



Per comodità, consideriamo heap in cui le foglie dell'ultimo livello si trovano più a sinistra possibile

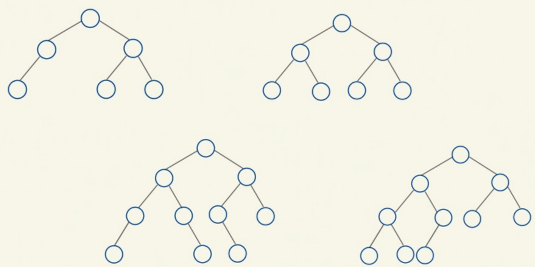
Alberi binari "quasi completi"

Definizione

Un *albero binario* è *quasi completo* quando è completo almeno fino al penultimo livello

in modo equivalente:

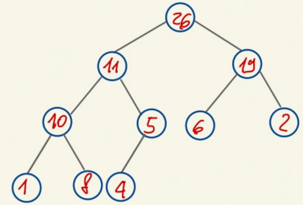
ogni nodo di profondità *minore* di $h - 1$ possiede *entrambi* i figli, dove h è l'altezza dell'albero



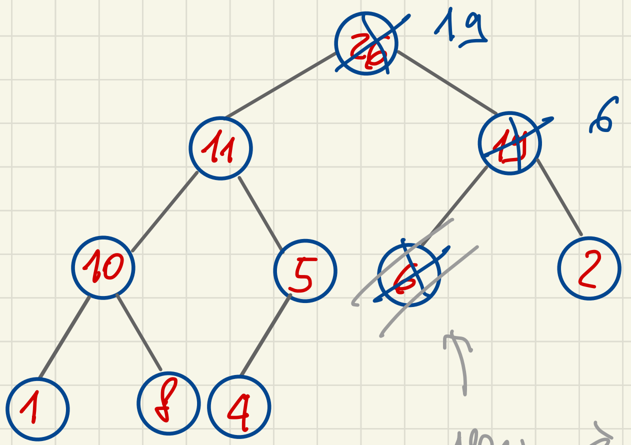
Heap

Definizione

Uno *heap* (o *max-heap*) è un albero binario quasi completo in cui la chiave contenuta in ciascun nodo è maggiore o uguale delle chiavi contenute nei figli

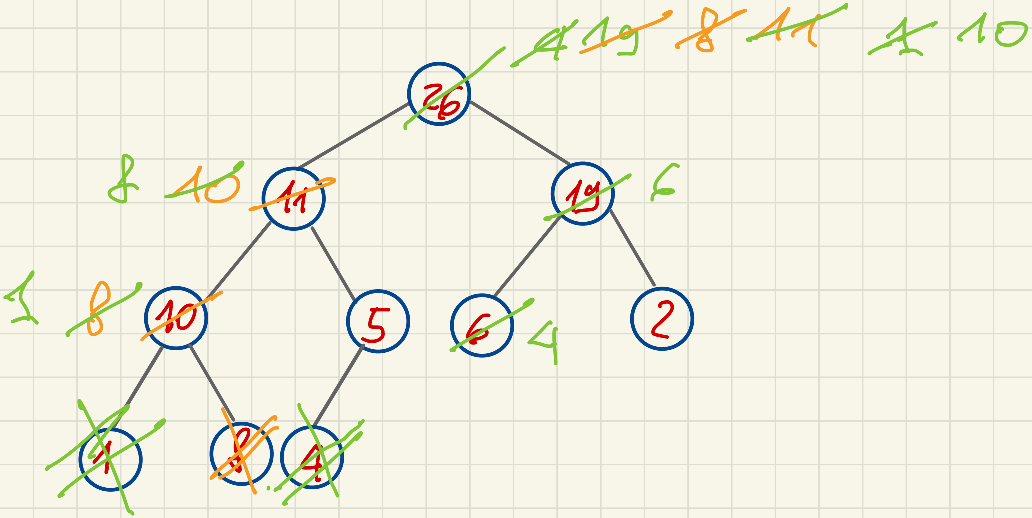


Per comodità, consideriamo heap in cui le foglie dell'ultimo livello si trovano più a sinistra possibile

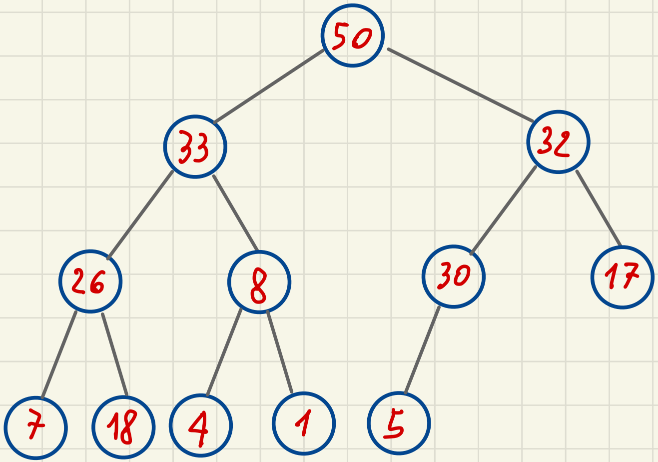


26

non è più un albero binario
quasi completo

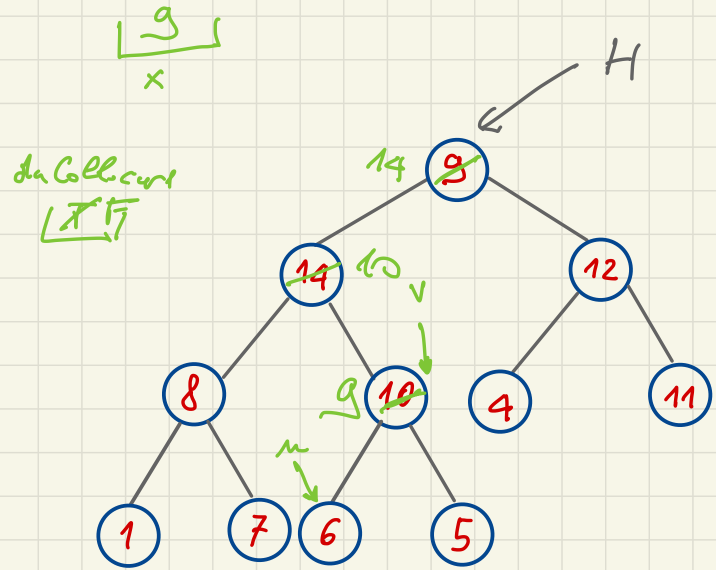


11 19 26


50
33
32
26
8
30
17
7
18
4
1
5

"risistema" (fixHeap) uno heap con radice "sbagliata"

```
PROCEDURA risistema (Heap H)
  v ← H                          v: nodo in esame
  x ← v.chiave                   x: chiave da sistemare
  y ← v.altri campi              y: campi associati a x
  daCollocare ← true
  DO
  | IF v è una foglia THEN
  | | daCollocare ← false
  | ELSE
  | | u ← figlio di v di valore max
  | | IF u.chiave > x THEN
  | | | v.chiave ← u.chiave
  | | | v.altricampi ← u.altricampi
  | | | v ← u
  | | ELSE
  | | | daCollocare ← false
  WHILE daCollocare
  v.chiave ← x
  v.altricampi ← y
```



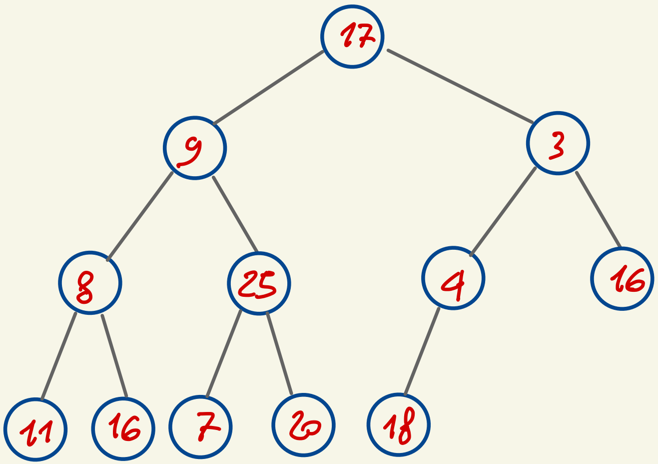
$\Theta(h)$ confronti (h = altezza)

Costruzione di heap

Dato un albero binario quasi completo contenente gli elementi da ordinare, come posso trasformarlo in uno heap?

Soluzione 1: Tecnica divide-et-impera
(strategia top-down)

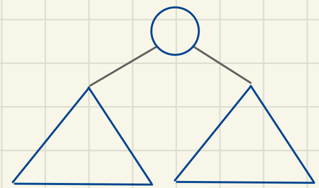
Soluzione 2: Dai sottoalberi più piccoli a quelli più grandi
(strategia bottom-up)



Soluzione 1: divide-et-impera

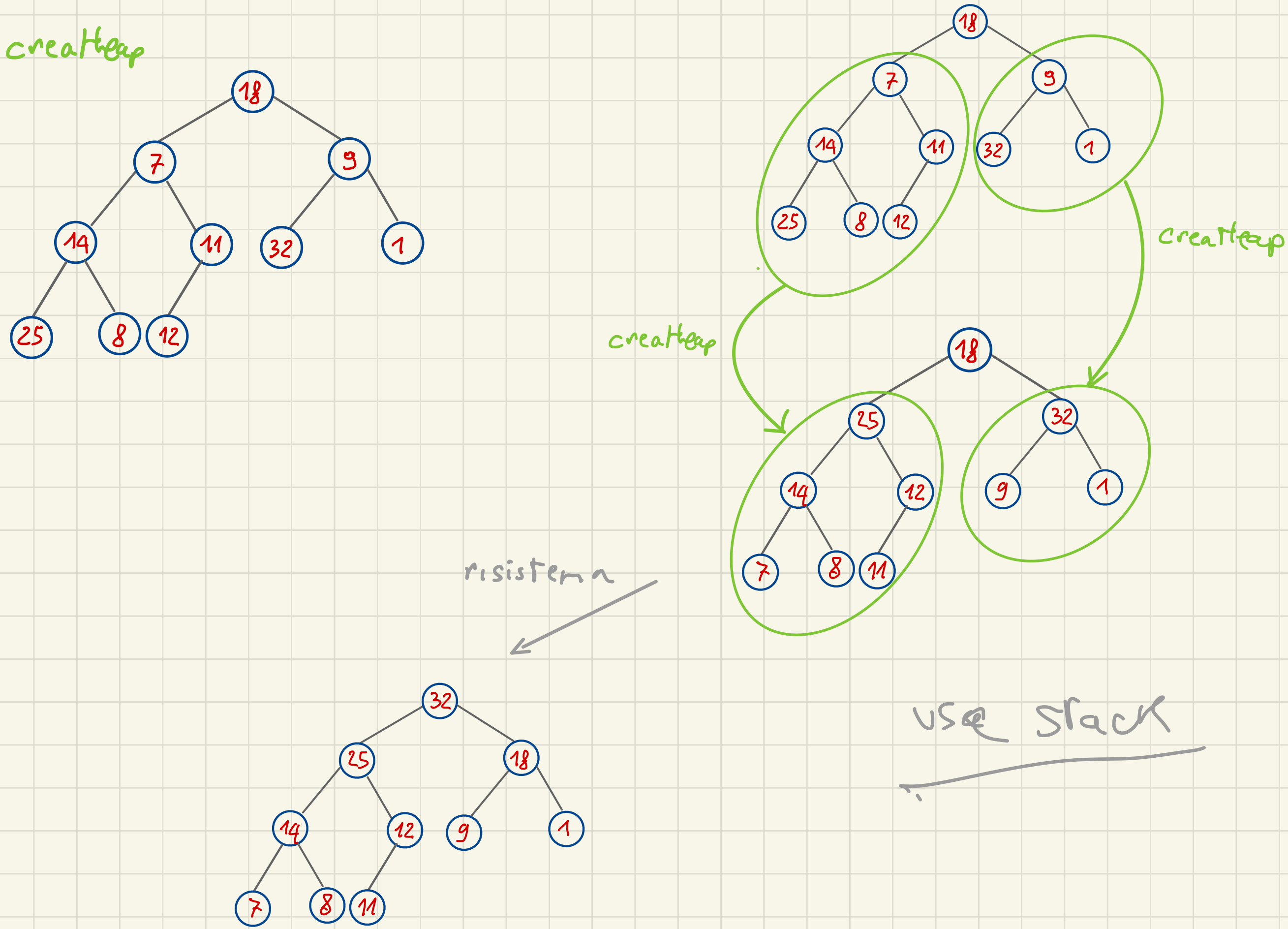
Albero vuoto $\rightarrow$ è già uno heap

Albero non vuoto

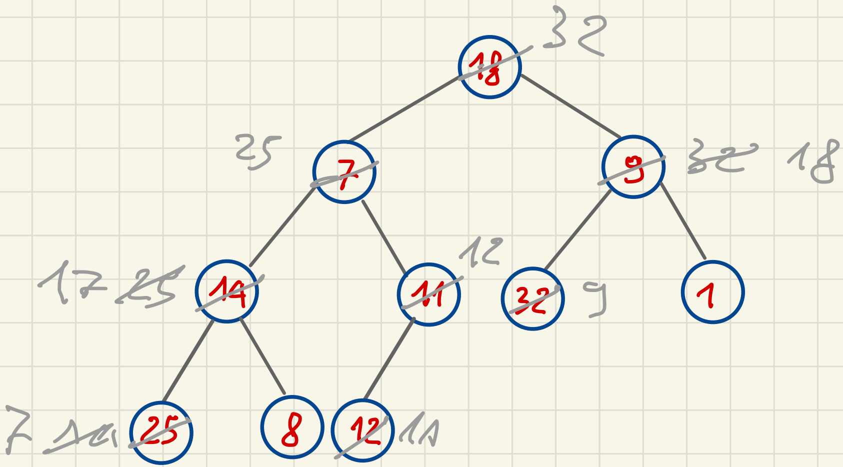


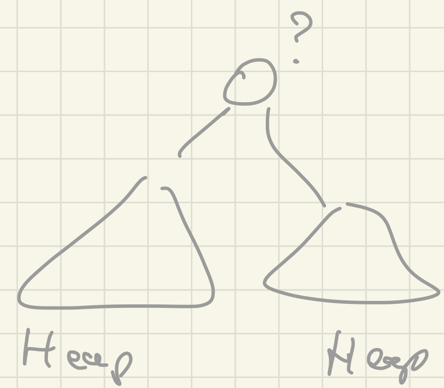
```
PROCEDURA creaHeap(albero T)
    IF T ≠ albero vuoto THEN
        creaHeap(T.sx)
        creaHeap(T.dx)
        risistema(T)
```

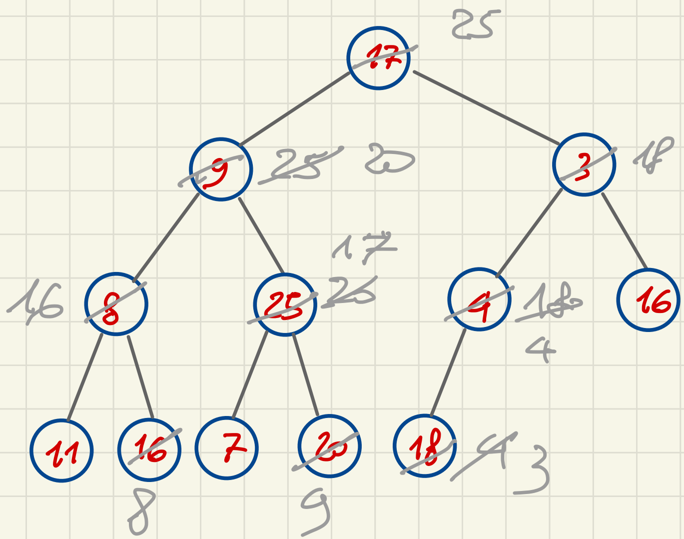
Esempi:



Soluzione 2:   partiamo dalle foglie







25
~~17~~
~~9~~
~~25~~
20
~~3~~
18
17
16
~~8~~
~~25~~
~~26~~
~~4~~
~~18~~
4
16
11
~~16~~
7
~~20~~
~~18~~
~~4~~
3
8
9

PROCEDURA createHeap (Albero T)

h ← altezza di T

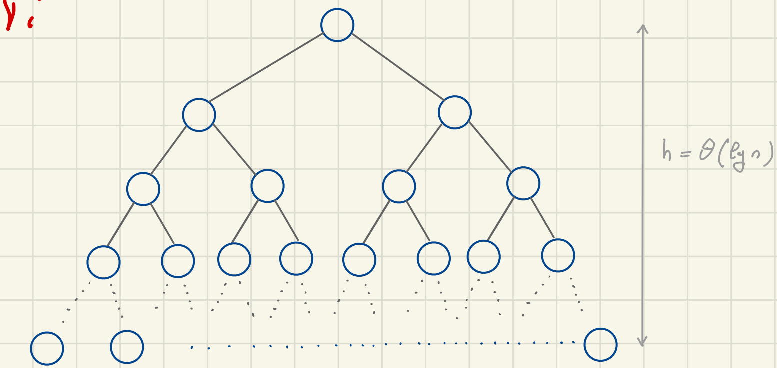
FOR p ← h DOWNTO 0 DO

FOREACH nodo x di profondità p DO

risistema (sottoalbero di radice x)

Analisi di CreaHeap: n° di confronti

- Analisi "immediata":



PROCEDURA createHeap (Albero T)
  h ← altezza di T
  FOR p ← h DOWNTO 0 DO
    FOREACH nodo x di profondità p DO
      risistema (sottoalbero di radice x)

n chiamate di risistema

risistema esegue $O(h')$ confronti
dove h' è l'altezza dell'albero da risistemare

$h' \leq h$ →
ogni chiamata di risistema fa al più $O(h)$ confronti

cfr è al più $n \cdot O(h) \leq O(n \log n)$

$h = \lfloor \log_2 n \rfloor$

- $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

- $\sum_{i=0}^{n} i\,2^i = (n-1)\,2^{n+1} + 2$

verifica ind.

$i=0$ $\qquad 0 \qquad -1 \cdot 2 + 2 = 0$

$i>0$ $\qquad \sum_{i=0}^{n} i\,2^i = \sum_{i=0}^{n-1} i\,2^i + n2^n = (n-2)\,2^n + 2 + n2^n$

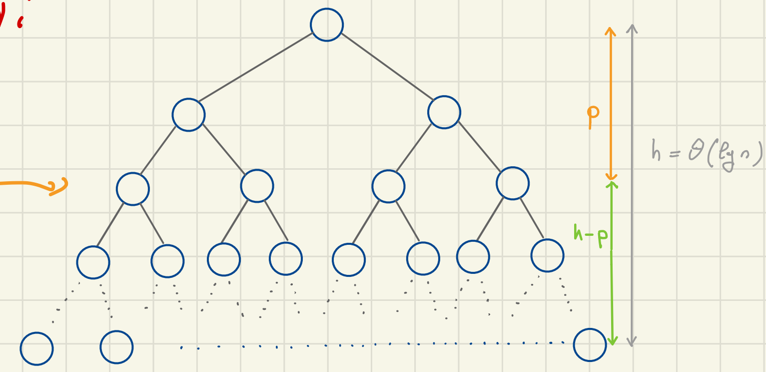
ip. ind.

$$= n2^n - 2^{n+1} + 2 + n2^n = n2^{n+1} - 2^{n+1} + 2$$

$$= (n-1)\,2^{n+1} + 2$$

Analisi di CreaHeap: n° di confronti

- Analisi migliorata



profondità p → alberi di altezza $\underline{h-p}$

$\Theta(h-p)$ confr. per ogni albero

profondità p

2^p alberi

```
PROCEDURA creaHeap (Albero T)
    h ← altezza di T
    FOR p ← h DOWNTO 0 DO
        FOREACH nodo x di profondità p DO
            risistema (sottoalbero di radice x)
```

\# cfr per alberi a prof. p : $2^p \cdot \Theta(h-p)$

\# cfr totali

$$\boxed{\sum_{p=0}^{h} 2^p \cdot \Theta(h-p)}$$

$$\sum_{p=0}^{h} 2^p (h-p) = h\sum_{p=0}^{h} 2^p - \sum_{p=0}^{h} p2^p =$$

$$= h2^{h+1} - h - (h-1)2^{h+1} - 2 = \cancel{h2^{h+1}} - h - \cancel{h2^{h+1}} + 2^{h+1} - 2$$

$$= 2^{h+1} - h - 2 \qquad h \approx \log_2 n \qquad \boxed{\# cfr = \Theta(n)}$$

ALGORITMO heapSort (Array A) $\rightarrow$ lista

crea uno heap H a partire da A

crea un albero binario quasi completo

crea Heap

X $\leftarrow$ lista vuota

WHILE $H \neq \emptyset$ DO

- rimuovi da H il valore alla radice e aggiungilo all'inizio di X
- rimuovi la foglia più a dx dell'ultimo livello e collocane il valore alla radice
- risistema (H)

RETURN X

Analisi di heapSort: n° di confronti

ALGORITMO heapSort (Array A) → lista

crea uno heap H a partire da A — crea un albero binario quasi completo / crea Heap → $\Theta(n)$ confronti

X ← lista vuota

WHILE $H \neq \emptyset$ DO

- rimuovi da H il valore alla radice e aggiungilo all'inizio di X
- rimuovi la foglia più a dx dell'ultimo livello e collocane il valore alla radice
- risistema (H) → $O(\log n)$

n iterazioni → $O(n \log n)$ confronti

RETURN X

Tot. confr. $\Theta(n) + O(n \log n) = O(n \log n)$ confronti