Algoritmi e Strutture Dati

Lezione 16

4 novembre 2022

Heapsort

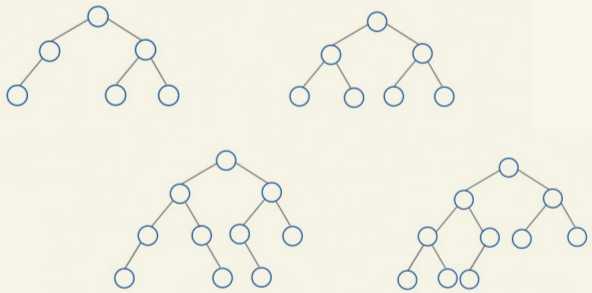
Alberi binari "quasi completi"

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 la profondità dell'albero

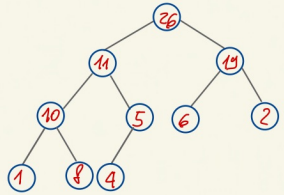


$$h = \lfloor \lg_2 n \rfloor$$

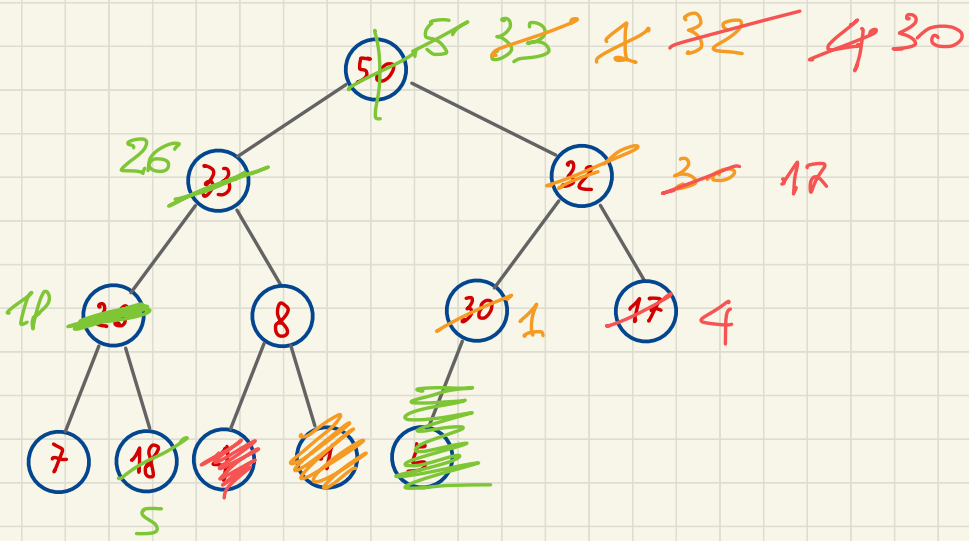
Heap

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



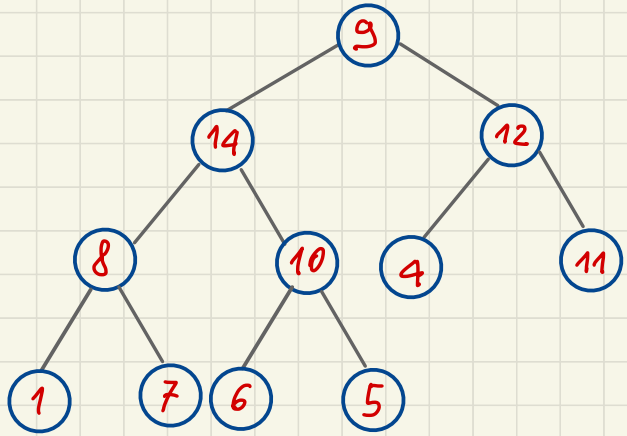
Per comodità, consideriamo heap in cui le foglie dell'ultimo livello si trovano più a sinistra possibile



- 3233 50

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

```
PROCEDURA risistema (Heap H)
    v ← H                    v: nodo in esame
    x ← v.chiave             x: chiave da risistemare
    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.altri campi
                v ← u
            ELSE
                daCollocare ← false
    WHILE daCollocare
    v.chiave ← x
    v.altricampi ← y
```



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

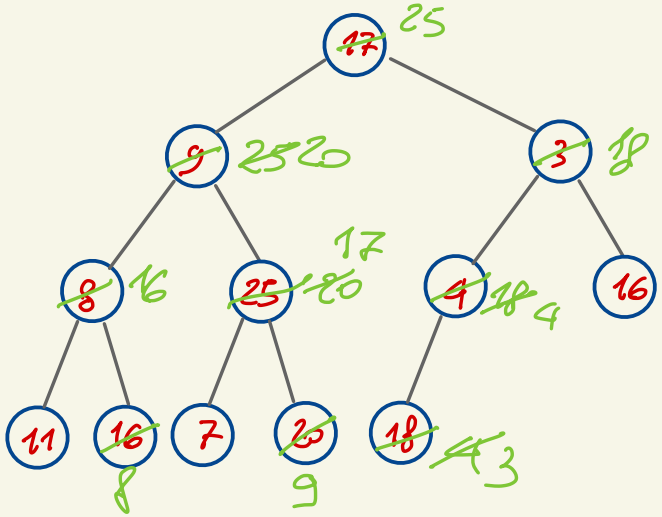
Costruzione di heap

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)

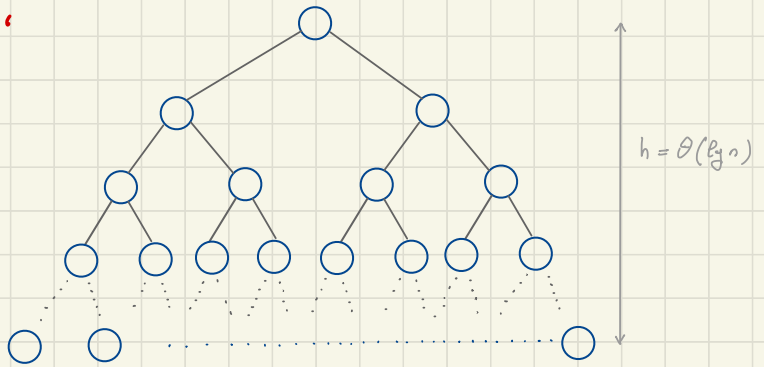
Soluzione 2: Dai sottoalberi più piccoli a quelli più grandi
(strategia bottom-up)

```
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)
```



Analisi di CreaHeap: n° di confronti

- Analisi "immediata":




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

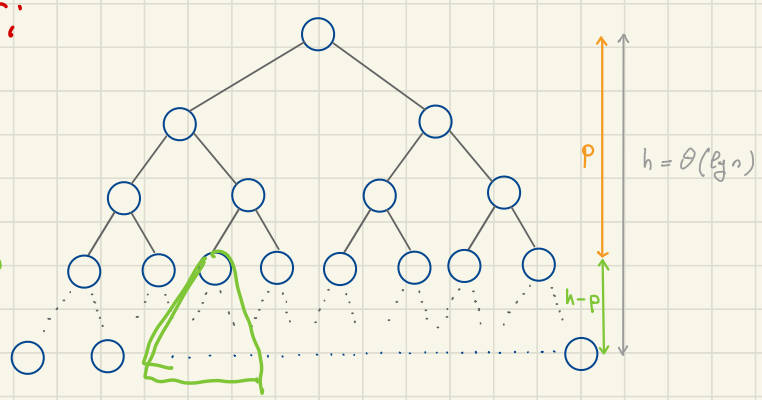
n volte

$O(\log n)$ passi

$O(n \log n)$

Analisi di CreaHeap: n° di confronti

- Analisi migliorata



profondità p → 2^p nodi

- ristrutturare un nodo di profondità p → $\Theta(h-p)$
- # nodi prof. p : 2^p

$\Theta\left(2^p(h-p)\right)$ passi

profondità p

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

$\Theta(h-p)$ passi

$$\sum_{p=0}^{h} 2^p(h-p) = \sum_{p=0}^{h} h2^p - \sum_{p=0}^{h} p2^p = h\left(2^{h+1}-1\right) - (h-1)2^{h+1} - 2 =$$

$$\sum_{i=0}^{n} i2^i = (n-1)2^{n+1} + 2$$

$$= h2^{h+1} - h - h2^{h+1} + 2^{h+1} - 2$$

$$= 2^{h+1} - h - 2 \qquad h \approx \lg_2 n \qquad \approx 2^{\lg_2 n + 1} + \dots = \Theta(n)$$

ALGORITMO heapSort (Array A) → lista

crea uno heap H a partire da A — trasforma A in un albero binario quasi completo → creaHeap(A)

X ← lista vuota

WHILE H ≠ ∅ DO

- rimuovi da H il valore della radice e aggiungilo all'inizio di X
- rimuovi l'ultima foglia di H e colloca il valore alla radice
- risistema (H)

Analisi di heapSort: n° di confronti

ALGORITMO heapSort (Array A) → lista

crea uno heap H a partire da A — trasforma A in un albero binario quasi completo

X ← lista vuota — createHeap(A) → $\Theta(n)$

WHILE $H \neq \emptyset$ Do

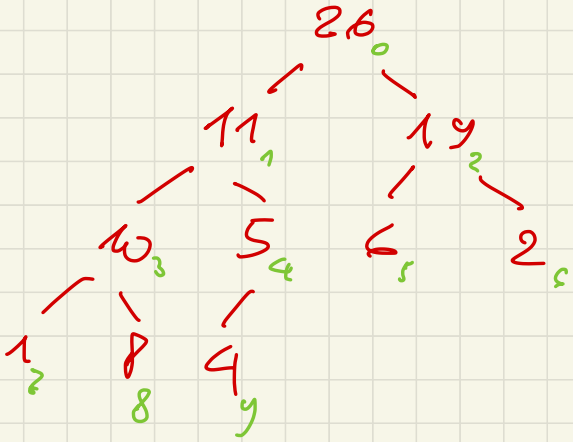
- rimuovi da H il valore della radice e aggiungilo all'inizio di X
- rimuovi l'ultima foglia di H e colloca il valore della radice
- risistema (H) → $O(\lg n)$

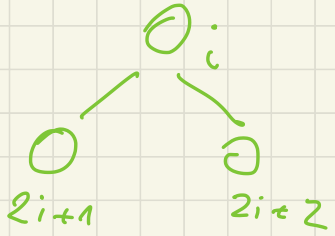
n iterazioni → $O(n \lg n)$

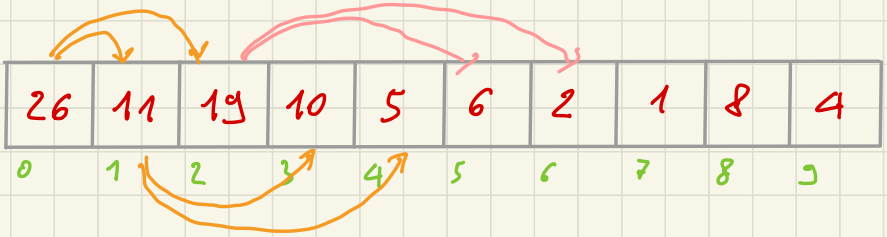
$$\Theta(n) + O(n \lg n) = O(n \lg n)$$

HeapSort: implementazione in Loco

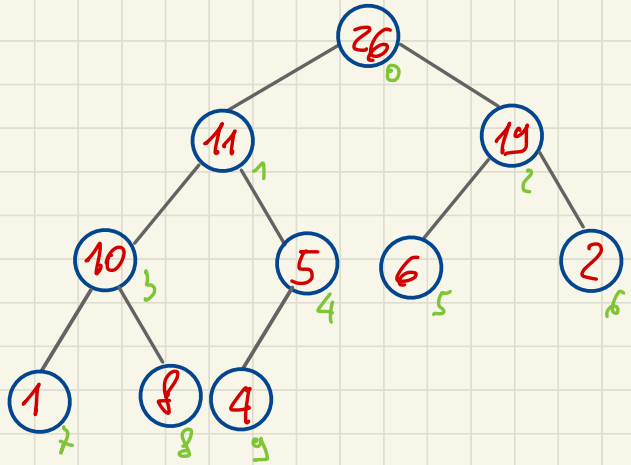
26	11	19	10	5	6	2	1	8	4
0	1	2	3	4	5	6	7	8	9

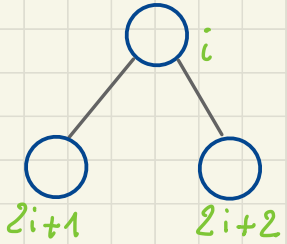


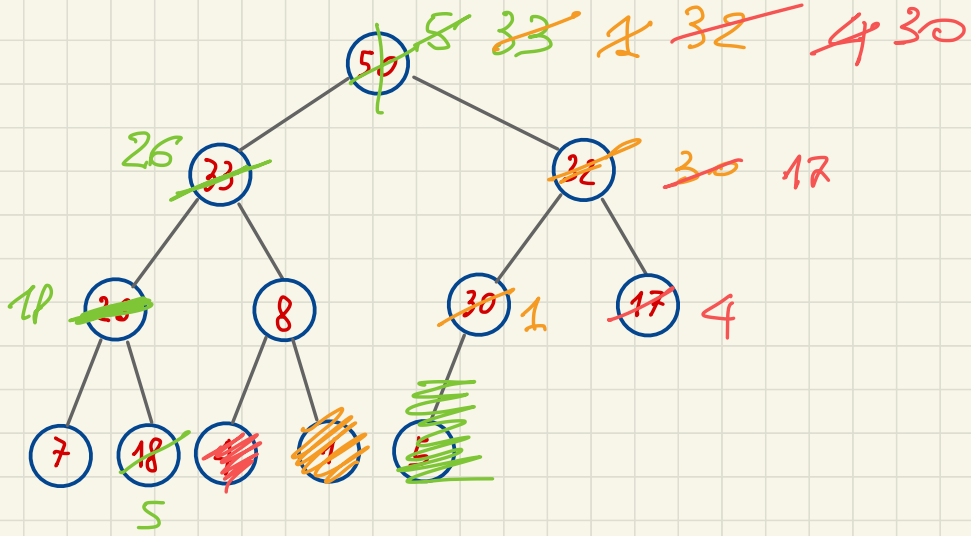




VETTORE POSIZIONALE



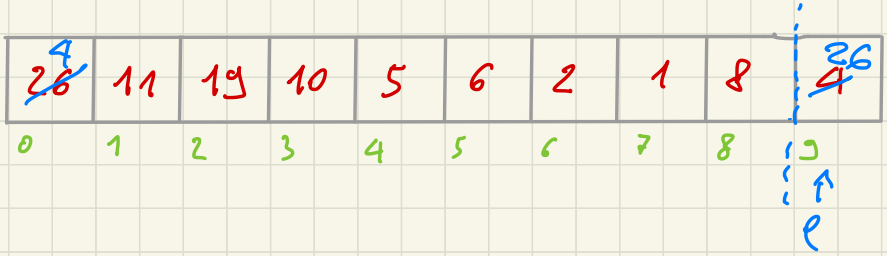


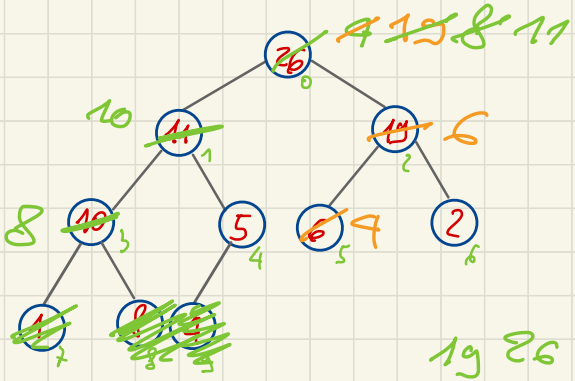


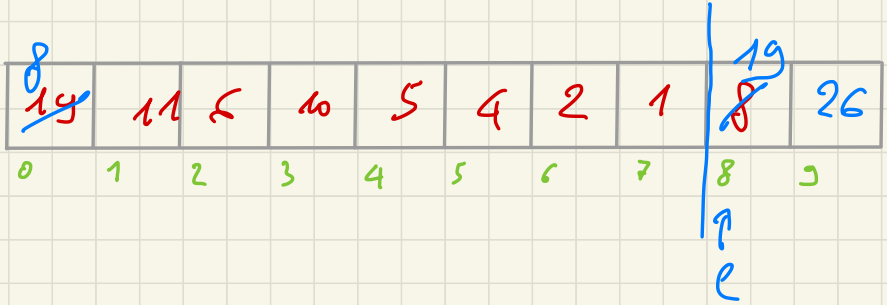
- 32 33 50

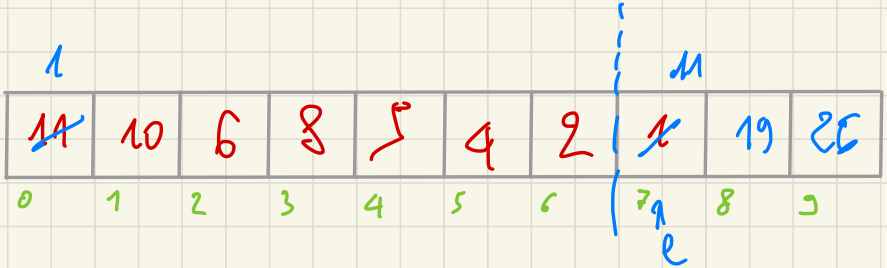
30	26	17	18	8	1	4	7	5	32	33	50
0	1	2	3	4	5	6	7	8	9	10	11

↑
R


26 11 19 10 5 6 2 1 8 4
0 1 2 3 4 5 6 7 8 9
4
26
i

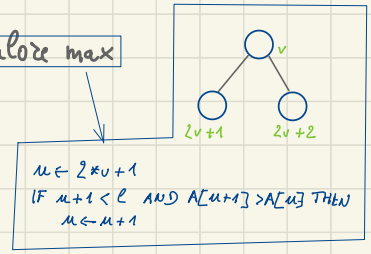

26 11 19 10 5 6 2 1 8 4
0 1 2 3 4 5 6 7 8 9
19 26

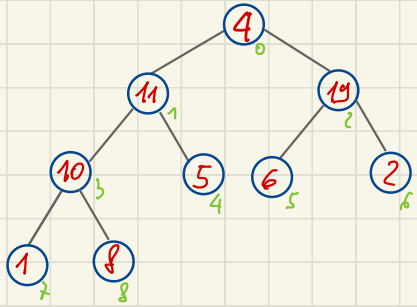

19 11 5 10 5 4 2 1 8 26
0 1 2 3 4 5 6 7 8 9
8
19
i


11 10 6 8 5 4 2 1 19 26
0 1 2 3 4 5 6 7 8 9
1
11
i

PROCEDURA risistema (Array A[0..n-1], intero r, intero e)

```
v ← r          v: posizione in esame
x ← A[v]       x: chiave da risistemare
per semplicità viene indicata solo la chiave
daCollocare ← true
DO
    IF 2*v+1 ≥ e THEN        (v è l'indice di una foglia)
        daCollocare ← false
    ELSE
        u ← indice del figlio di v di valore max
        IF A[u] > x THEN
            A[v] ← A[u]
            v ← u
        ELSE
            daCollocare ← false
WHILE daCollocare
A[v] ← x
```

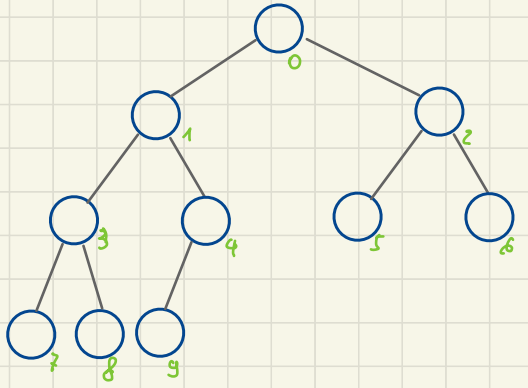




PROCEDURA creaHeap (Array A[0..n-1])

FOR i ← ~~n-1~~ $\lfloor \frac{n}{2} - 1 \rfloor$ DOWNTO 0 DO

risistema (A, i, n)



```
ALGORITMO heapSort(Array A[0..n-1])
    creaHeap(A)
    FOR l ← n-1 DOWNTO 1 DO
    |   scambia A[0] con A[l]
    |   risistema(A, 0, l)
```

HeapSort: riassumendo

- Algoritmo di ordinamento IN LOCO
- No memoria aggiuntiva (con creaHeap bottom-up)
- # CFR $\Theta(n \lg n)$
- Tempo $\Theta(n \lg n)$ se i cfr costano $O(1)$
- Metodo NON STABILE

10_2	10_1	20

10_1 10_2 20