HEAPSORT

ORDINAMENTI PER CONFRONTI

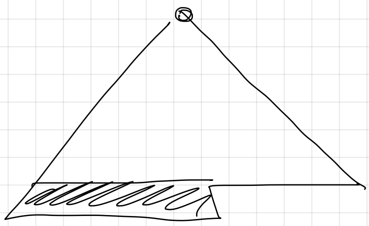
ordine di grandezza del

	COSTO IN TEMPO caso ottimo	caso medio	caso pessimo	COSTO in SPAZIO	commenti
Insertion Sort	n	n^2	n^2	in loco	
Selection Sort	n^2	n^2	n^2	in loco	
Merge Sort	$n \log n$	$n \log n$	$n \log n$	n	ottimo in tempo
QuickSort	$n \log n$	$n \log n$	n^2	in loco + gestione ricorsione	OTTIMO in tempo al caso medio
Heapsort	$n \log n$	$n \log n$	$n \log n$	in loco	ottimo in tempo e spazio

→ usare una struttura dati HEAP di MASSIMO

Heap

albero binario quasi completo



pieno su tutti i livelli tranne l'ultimo, dove le foglie sono accumulate a sinistra

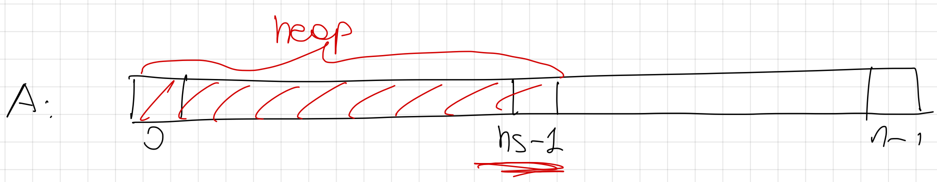
→ rappresentazione su array, senza memorizzare riferimenti al padre e ai nodi figli

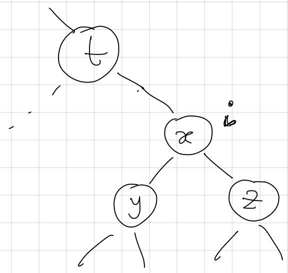
array A

A.length = n

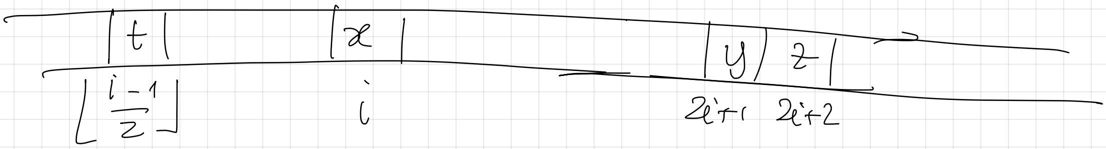
A[0] : radice dell'albero

A.hs = # elementi dell'heap memorizzati in A





x : corrisponde all'elemento di indice i in A

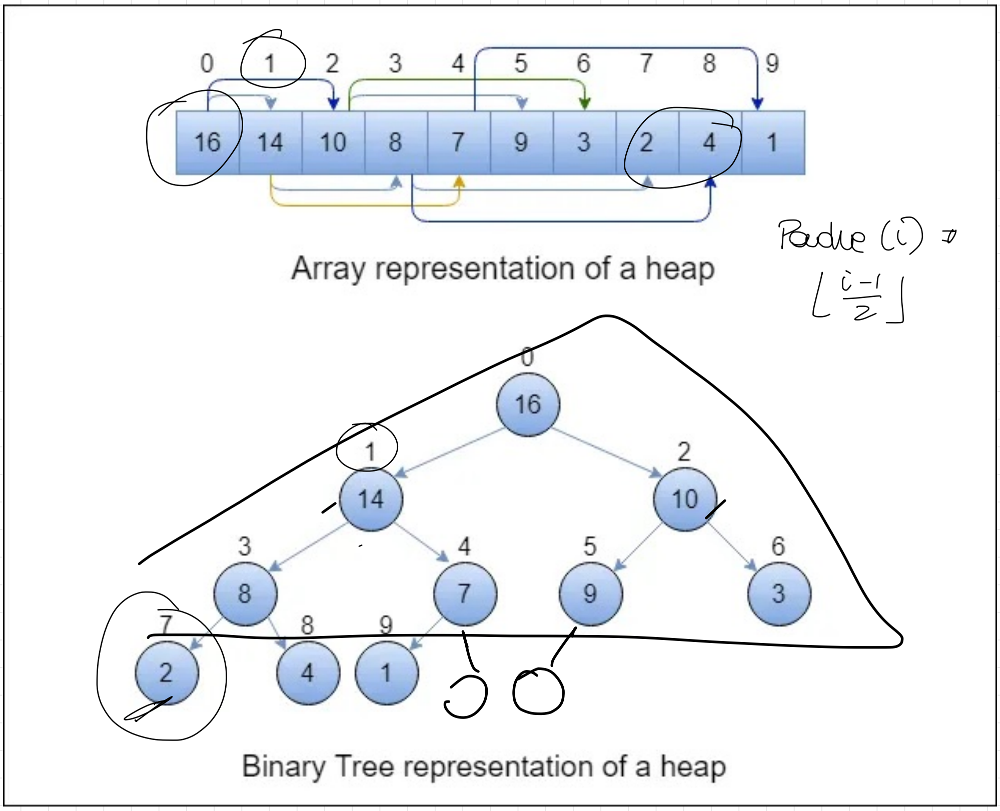


Regole di posizionamento

Parent(i)
return $\lfloor \frac{i-1}{2} \rfloor$;

Left(i)
return $2*i+1$;

Right(i)
return $2*i+2$;



Array representation of a heap

Binary Tree representation of a heap

$i = 3$

1

$2*i + 1$

$2*i + 2$

Correttezza regole di posizionamento

$$Parent(Left(i)) = i$$

$$Parent(Right(i)) = i$$

↳

$$Parent(Left(i)) = Parent(2 * i + 1) =$$

$$= \left\lfloor \frac{2 * i \cancel{+1} \cancel{-1}}{2} \right\rfloor = \left\lfloor \frac{2i}{2} \right\rfloor = \lfloor i \rfloor = i \quad \checkmark$$

$$Parent(Right(i)) = Parent(2i + 2) =$$

$$= \left\lfloor \frac{2i + 2 - 1}{2} \right\rfloor = \left\lfloor \frac{2i + 1}{2} \right\rfloor = \left\lfloor i + \frac{1}{2} \right\rfloor = i \quad \checkmark$$

PROPRIETÀ di HEAP (Heap di massimo)

Un ~~heap~~ max-heap è:

Un albero binario quasi completo, in cui i valori dei nodi soddisfano la proprietà di MAX-heap

$\forall$ nodo i, diverso dalla radice ($\forall i > 0$) $\quad A[Parent(i)] \geq A[i]$

(il valore di un nodo è $\leq$ al valore del nodo padre)

CONSEGUENZE:
- l'elemento massimo è nella radice
- il sottoalbero radicato in un nodo contiene valori $\leq$ a quelli del nodo stesso

Definizioni

Altezza di un nodo

$\#$ archi nel cammino più lungo dal nodo a una foglia

Profondità di un nodo

distanza dalla radice ($\#$ archi)

Altezza di un heap: altezza della radice

→ (massima distanza di una foglia dalla radice)

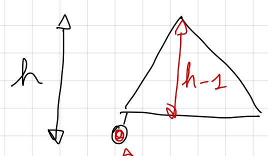
→ massima profondità raggiunta dalle foglie

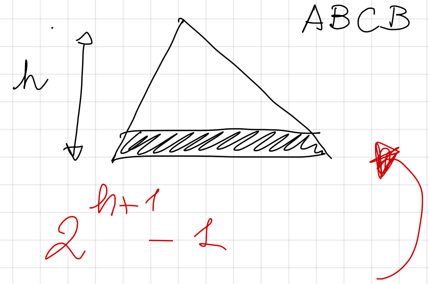
PROPRIETÀ

1. Un heap di n elementi ha altezza $\lfloor \log n \rfloor$

$(h = O(\log n))$

Dim





ABCB

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

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

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

$h \le \log n$ $\qquad$ $\log n < h+1 \Rightarrow h > \log n - 1$

$$\log n - 1 < h \le \log n$$

$$\Rightarrow \quad h = \lfloor \log n \rfloor$$

(2) Un heap di n nodi contiene $\lceil \frac{n}{2} \rceil$ foglie

<u>Dim</u> contiamo i nodi interni

Un nodo i è un nodo interno se ha almeno un figlio:

$$\underbrace{2*i+1}_{\text{indice del figlio sx di } i} \leq \underbrace{n-1}_{\text{indice nell'array A dell'ultimo nodo dell'heap}}$$

$$2i \leq n-2 \qquad i \leq \frac{n}{2} - 1$$

tutti i nodi di indice da $\underbrace{0 \text{ a } \lfloor \frac{n}{2} \rfloor - 1}$ sono nodi interni

$$\# \text{ nodi interni} = \lfloor \frac{n}{2} \rfloor$$

$$\Rightarrow \# \text{ foglie} = n - \lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil \qquad \blacksquare$$

(3) In un heap di n nodi ci sono al più $\lceil \frac{n}{2^{h+1}} \rceil$ nodi di altezza h.

<u>Intuizione</u> heap pieno su tutti i livelli (A B C B) Albero Binario Completamente Bilanciato

$$n = 2^{H+1} - 1$$ dispari ABCB di altezza H

$$\# \text{ foglie: } 2^H \text{ foglie} = \lceil \frac{n}{2} \rceil = \lceil \frac{n}{2^{0+1}} \rceil$$

altezza delle foglie $h = 0$

nodi $h = 0$ (foglie) $\left\lceil \frac{n}{2} \right\rceil = \left\lceil \frac{n}{2^{0+1}} \right\rceil$

nodi $h = 1$ (padri delle foglie) $\left\lceil \frac{n}{4} \right\rceil = \left\lceil \frac{n}{2^{1+1}} \right\rceil$

⋮

nodi $h = 2$ $\longrightarrow$ $\left\lceil \frac{n}{8} \right\rceil = \left\lceil \frac{n}{2^{2+1}} \right\rceil$

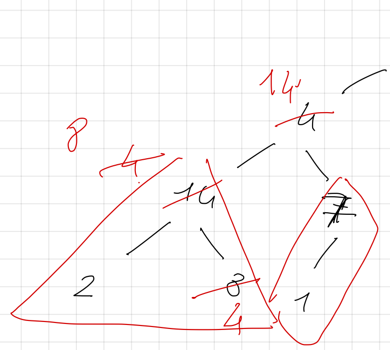
nodi di altezza i $\longrightarrow$ $\left\lceil \frac{n}{2^{i+1}} \right\rceil$

MAX-heapify

procedura che ripristina la proprietà di heap, in un heap di radice i, t.c. gli alberi radicati in Left(i) e in Right(i) sono heap di massimo, ma $A(i)$ viola la proprietà di max-heap

A =

16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9



Max-Heapify (A, i)

// IPOTESI: alberi radicati in Left(i) e Right(i) sono max-heap

```
l = Left(i);
r = Right(i);
max = i;
if ( l < A.hs  && A[l] > A[i]) max = l;
if ( r < A.hs  && A[r] > A[max]) max = r;
if (max ≠ i) {
        scambio A[i] e A[max];
        Max-Heapify (A, max);
}
```

Analisi

- nel caso peggiore A[i] "scende" fino alle foglie
- su ogni livello si spende tempo costante (ricerca del massimo di al più 3 elementi)

$\Rightarrow$ costo proporzionale al numero di livelli (→ all'altezza del nodo i):

$$T(n) = O(h) \overset{①}{=} O(\log n)$$

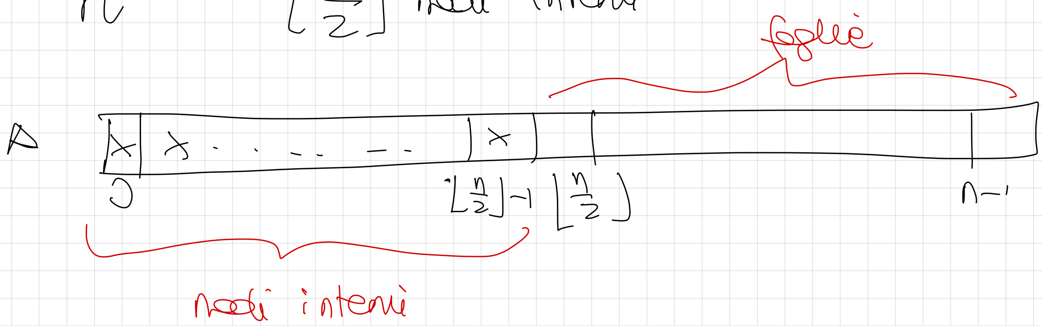
h: altezza del nodo i (A[i])

COSTRUZIONE HEAP

Build-Max-Heap : trasforma un array A nella rappresentazione implicita di un max-heap

BOTTOM-UP chiamiamo Max-Heapify dal basso verso la radice, partendo dal nodo interno + profondo

n $\quad \lfloor \frac{n}{2} \rfloor$ nodi interni



le $\lceil \frac{n}{2} \rceil$ foglie sono heap di un solo elemento

Build-Max-Heap (A, n) // A.length = n

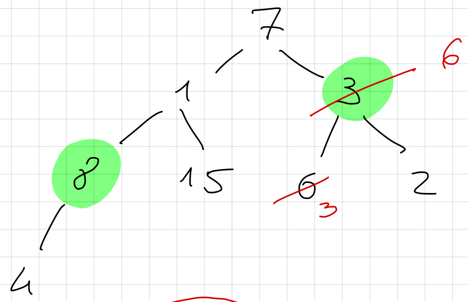
```
A.hs = n;
for (i = ⌊n/2⌋-1; i ≥ 0; i--) {     // ⌊n/2⌋-1
        Max-Heapify (A, i);           // posizione
}                                     // dell'ultimo nodo
                                      // interno
```

DA FARE : correttezza
+
analisi di complessità

ESEMPIO

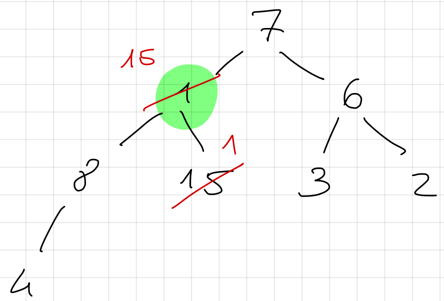
A =

7	1	3	8	15	6	2	4



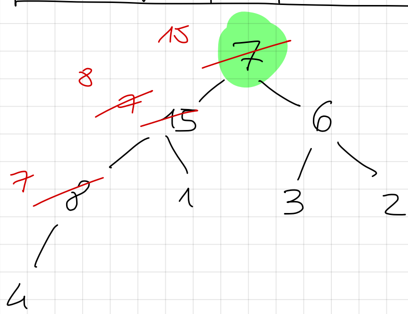
A =

7	1	6	8	15	3	2	4



A =

7	15	6	8	1	3	2	4



A =

15	8	6	7	1	3	2	4