Heap

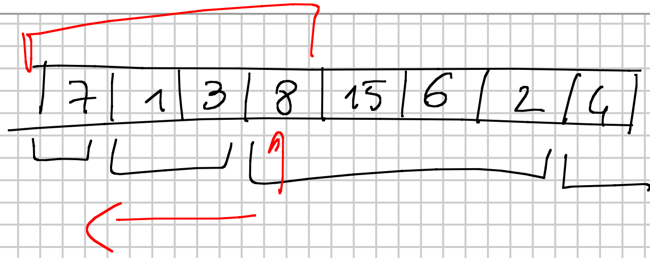
n = # nodi
H = altezza

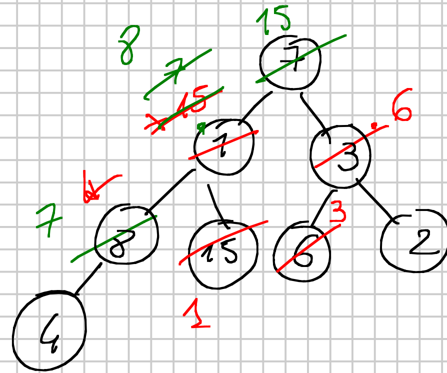
1. $H = \lfloor \log n \rfloor$
   $H = O(\log n)$

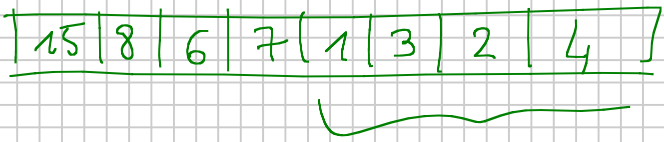
2. Un heap di n nodi contiene $\lceil \frac{n}{2} \rceil$ foglie

3. Un heap di n nodi contiene al più $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ nodi di altezza h.

Titolo nota 26/02/2016







Build-Max-Heap (A) // dim A = n

A.heap-size = n;
for i = $\lfloor \frac{n}{2} \rfloor$ downto 1 {
Max-Heapify (A, i);
}

Titolo nota 26/02/2016

Correttezza (Invariante di ciclo)

All'inizio di ogni iterazione del for ogni nodo $i+1, i+2, \dots, n$ è radice di un max-heap.

Complessità chiamo $\left\lfloor \frac{n}{2} \right\rfloor$ volte $\underbrace{\text{Max-Heapify}}_{O(\log n)}$

$$T(n) = \left\lfloor \frac{n}{2} \right\rfloor \cdot O(\log n) = O(n \log n)$$

Teorema $T(n) = \Theta(n)$

$n_h = \#$ nodi di altezza h

Dim

$$T(n) = \sum_{h=1}^{H} n_h \cdot O(h) \underset{①}{=} \sum_{h=1}^{\lfloor \log n \rfloor} n_h \cdot O(h) \underset{③}{\leq} \sum_{h=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot O(h)$$

Titolo nota 26/02/2016

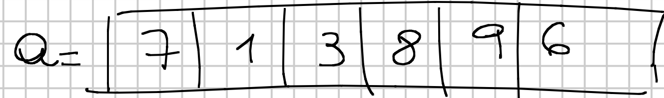
$$T(n) \leq \sum_{h=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot O(h) = O\left(\sum_{h=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2} \cdot \frac{h}{2^h} \right\rceil \right) =$$

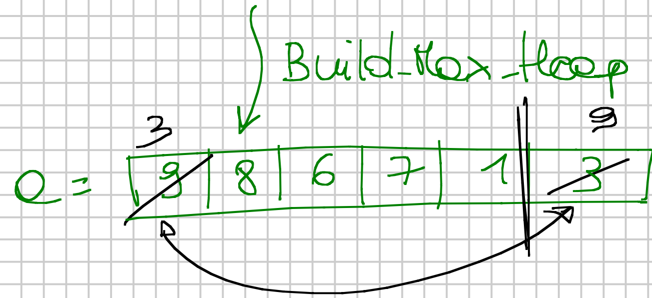
$$= O\left(\frac{n}{2} \sum_{h=1}^{\lfloor \log n \rfloor} \frac{h}{2^h} \right) = O\left(\frac{n}{2} \underbrace{\sum_{h=1}^{+\infty} \frac{h}{2^h}}_{2} \right) = O(n)$$

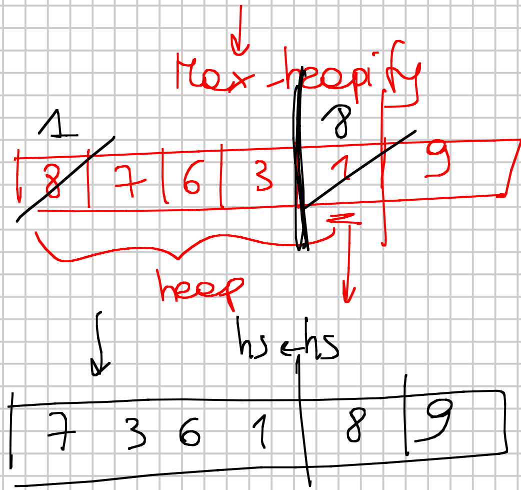
D

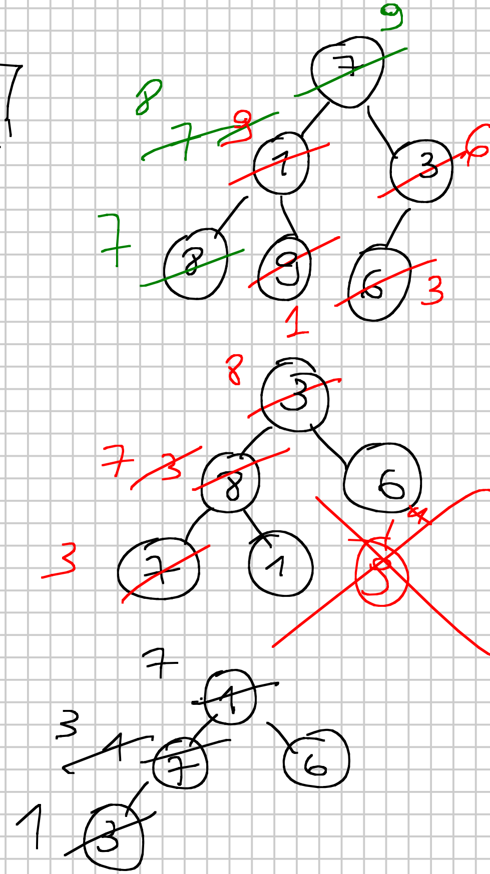
$$S(n) = \Theta(1)$$

Heapsort









Titolo nota 26/02/2016

HeapSort(A)

```
Build-Max-Heap(n);
A.heap-size=n;
for i = n downto 2 {
    scambio(A[1], A[i]);
    A.heapsize --;
    Max-Heapify(A, 1);
}
```

$\Theta(n)$

$\Theta(1)$

ripeto n-1 volte il corpo, di costo

$\Theta(1) + O(\log n)$

$T(n) = \Theta(n) + O(n \log n) = O(n \log n)$

OTTIMO

$S(n) = \Theta(1)$

Correttezza

invariante di ciclo

All'inizio di ogni iterazione del for il segmento di array $A[1..i]$ è un max-heap che contiene gli i elementi più piccoli di A, e il segmento $A(i+1, ..., n)$ contiene gli $(n-i)$ elementi più grandi di A, ordinati.

Titolo nota 26/02/2016

HEAP

implementazione delle code di priorità

CODA FIFO

	array non ordinato	array ordinato	lista non ordinata	lista ordinata	heap
Insert	$\Theta(1)$	$O(n)$	$\Theta(1)$	$O(n)$	$O(\log n)$
Maximum	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Extract-Max	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$O(\log n)$
Increase key	$\Theta(1)$	$O(n)$	$\Theta(1)$	$O(n)$	$O(\log n)$

Titolo nota 26/02/2016

Implementazione delle operazioni della Coda di Priorità su heap

Heap_Maximum(A)

```
return A[1];
```

$T(n) = \Theta(1)$

Heap_Extract_Max(A)

```
if (A.heap_size < 1) error "underflow"
max = A[1];
A[1] = A[A.heap_size];
A.heapsize --;
Max_heapify(A, 1);
return max;
```

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

Titolo nota 26/02/2016

Heap_Increase_key (A, i, k)

```
if (k < A[i]) errore * la nuova chiave è più piccola della chiave corrente A
A[i] = k;
while (i > 1 && A[Parent(i)] < A[i]) {
        scambia (A[i], A(parent(i)));
        i = Parent (i);
}
```

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

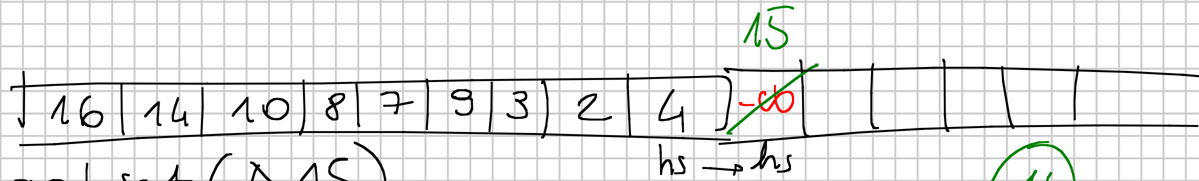
Titolo nota 26/02/2016

Max-Heap-Insert (A, key)

A.heapsize ++ ;
A[A.heapsize] = −∞ ;
Heap-Increase-Key(A, A.heapsize, key)

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

ESEMPIO



Max-Heap-Insert(A, 15)