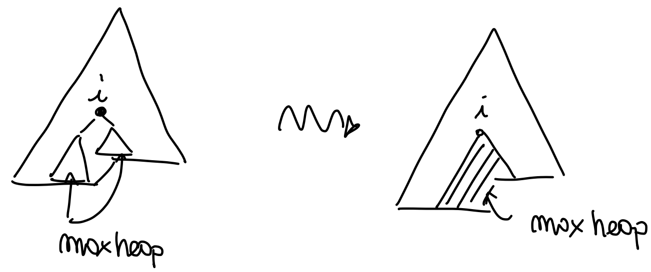
Algoritmi e Strutture Dati (14/10/2021)

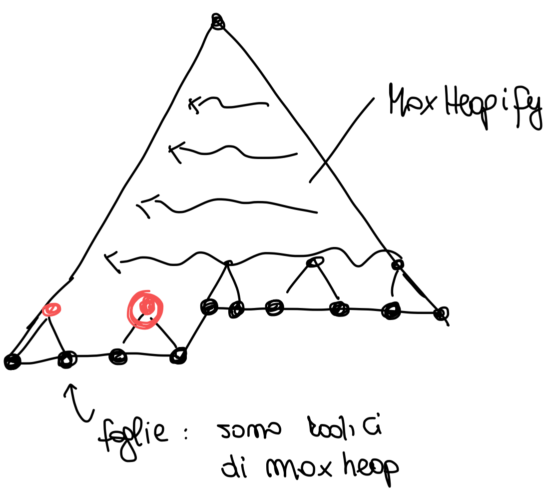
* Heap sort

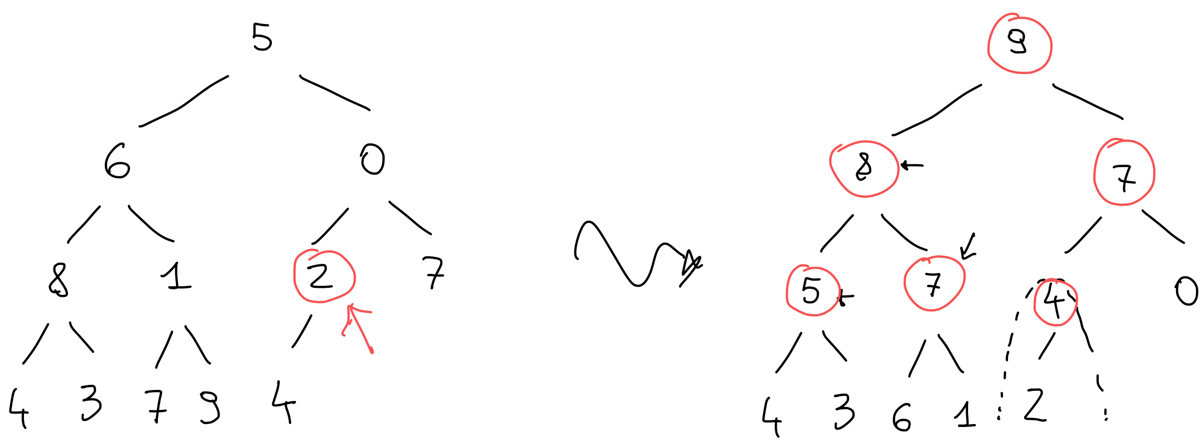
* MaxHeapify (A, i)

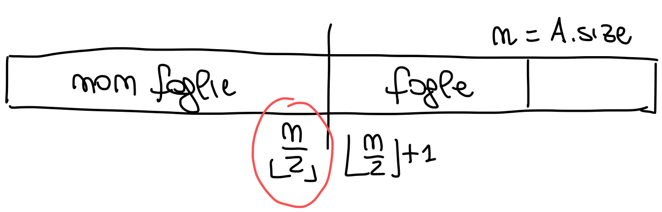


* Build Max Heap

dato A array







* **OSSERVAZIONE**: Dato A array visto come heap $A.size = n$

i nodi foglia sono $A[i]$ $\quad i \geq \lfloor \frac{n}{2} \rfloor + 1$

perché?

- se $i \geq \lfloor \frac{n}{2} \rfloor + 1$ $\quad Left(i) = 2 \cdot i \geq 2\left(\lfloor \frac{n}{2} \rfloor + 1\right) = \underbrace{2\lfloor \frac{n}{2} \rfloor}_{\geq n-1} + 2 \geq n+1$

  $\Rightarrow$ i non ha figlio sinistro $\Rightarrow$ è foglia

- si $i \leq \lfloor \frac{n}{2} \rfloor$ allora $Left(i) = 2 * i \leq 2\lfloor \frac{n}{2} \rfloor \leq n$

  $\Rightarrow$ i ha figlio sinistro $\Rightarrow$ non è foglia

Build Max Heap (A)

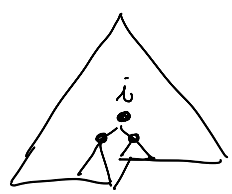
$A.size = A.length$

for $i = \lfloor \frac{A.size}{2} \rfloor$ downto 1 $\quad$ // proprietà: $\forall j > i$ $A[j]$ è radice di max heap

MaxHeapify (A, i) $\leftarrow$

inizio: $i = \lfloor \frac{A.size}{2} \rfloor$ $\quad \forall j > i$ $A[j]$ foglia è radice di max heap

"mantenimento":



$Left(i) = 2 * i$
$Right(i) = 2 * i + 1$ $\quad > i$

sono radici di max heap

$\Rightarrow$ MaxHeapify (A, i) albero in $A[i]$ max heap

conclusione: $i = 0$

$\forall j > 0$ $A[j]$ radice di max heap

$j = 1$ $\quad A[1]$ " $\quad$ " - - " "

Analisi :

Build Max Heap (A) $\qquad A.size = n$

$A.size = A.length$

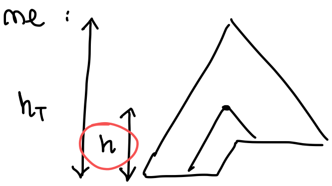
for $i = \lfloor \frac{A.size}{2} \rfloor$ downto 1

MaxHeapify (A, i)

* limite superiore semplificato

\# di iterazioni $\lfloor \frac{n}{2} \rfloor \qquad O(n)$

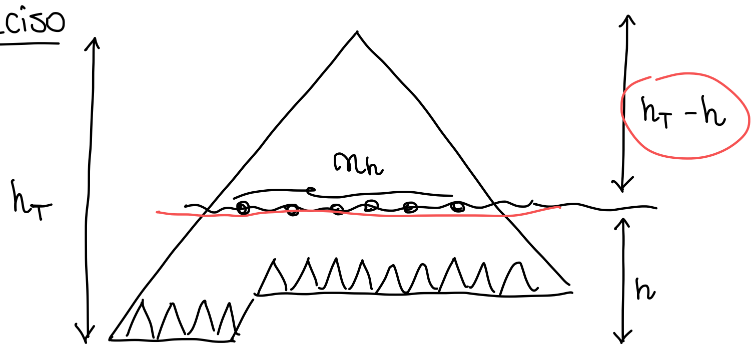
costo singola iterazione : $\qquad O(h) \leq O(h_T) = O(\log n)$



$$h \leq \boxed{h_T \leq \log_2 n}$$

costo totale $\quad O(n \log n)$

* limite preciso



$$n_h \leq 2^{h_T - h} = \frac{2^{h_T}}{2^h} \overset{h_T \leq \log_2 n}{\leq} \frac{n}{2^h}$$

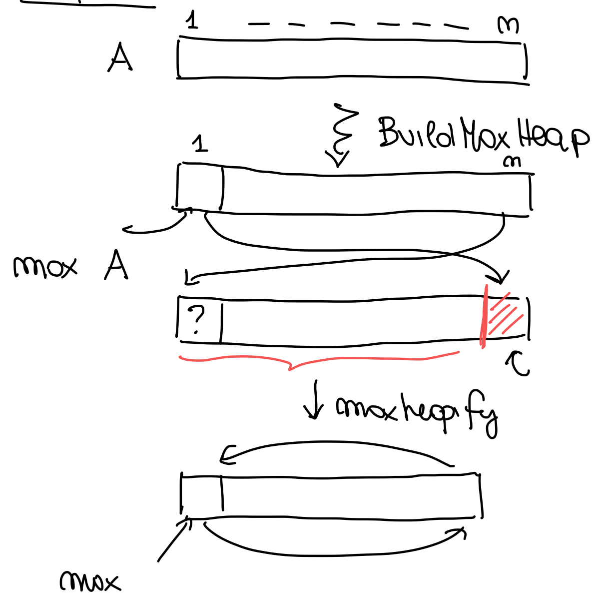
costo Max Heapify su albero di altezza h

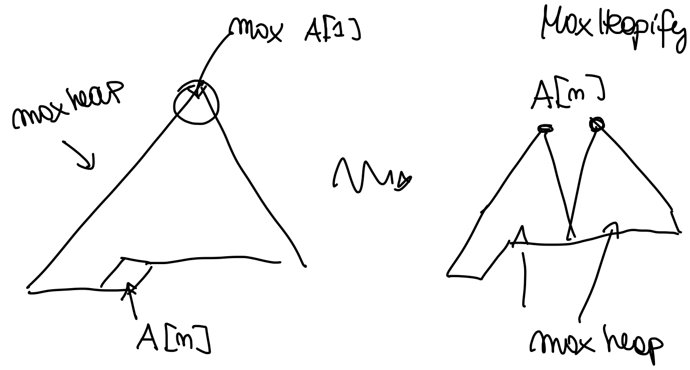
$$T(n) = \sum_{h=1}^{h_T} \underbrace{(\#\text{ sottoalb. di altezza } h)}_{n_h \leq \frac{n}{2^h}} * O(h)$$

$$\leq \sum_{h=1}^{h_T} \frac{n}{2^h} O(h) = O\left(\sum_{h=1}^{h_T} \frac{n}{2^h} h \right)$$

$$= O\Big(n \underbrace{\sum_{h=1}^{h_T} \frac{h}{2^h}}_{\leq \sum_{h=1}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-\frac{1}{2})^2} = 2} \Big) = O(n)$$

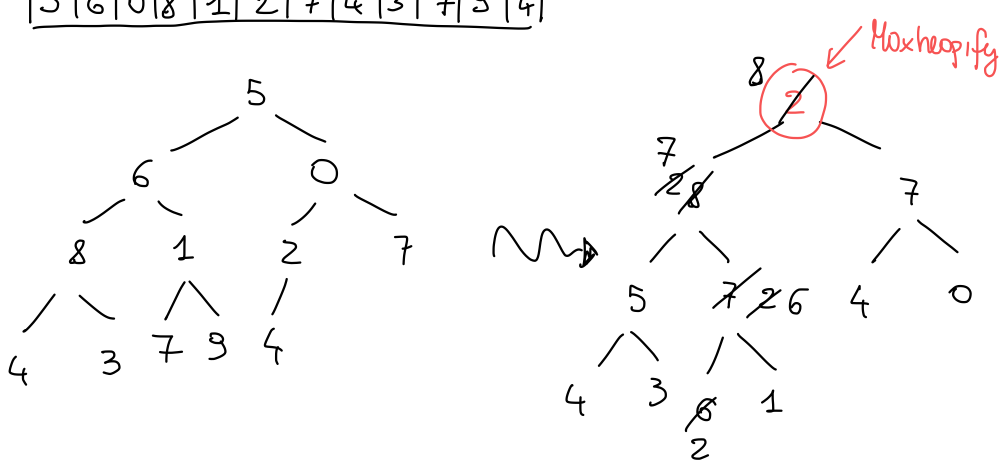
* HeapSort





HeapSort (A)
- BuildMaxHeap (A) ←
- for i = A.length downto 2
  - A[1] ↔ A[i] ←
  - A.size = A.size − 1
  - MaxHeapify (A, 1) ←

5	6	0	8	1	2	7	4	3	7	9	4



9	8	7	5	7	4	0	4	3	6	1	2

2	8	7	5	7	4	0	4	3	6	1	9

8	7	7	5	6	4	0	4	3	2	1	9

1	7	7	5	6	4	0	4	2	8	9

Correttezza :

Invariante : $A[1..i]$ max heap
$A[i+1..n]$ ordinato e $\underbrace{A[1..i] \leq A[i+1..n]}$

Conclusione : $i = 1$

$A[1]$ max heap

$\underbrace{A[2..n] \text{ ordinati} \qquad A[1] \leq A[2..n]}$

$\Rightarrow A[1..n]$ ordinato

Analisi

n = dim. array

```
HeapSort (A)
  BuildMaxHeap (A)          ←  O(n)
  for i = A.length downto 2
      A[1] ↔ A[i]
      A.size = A.size - 1
      MaxHeapify (A, 1)     ←  O(log n)
```

iterazione $O(n)$

costo : $\underbrace{O(n)} + O(n \log n) = O(n \log n)$

* Code con priorità

strutture dinamiche insieme S di elementi

Max (S) x.key

Remove Max (S)

Insert (S, x)

⋮

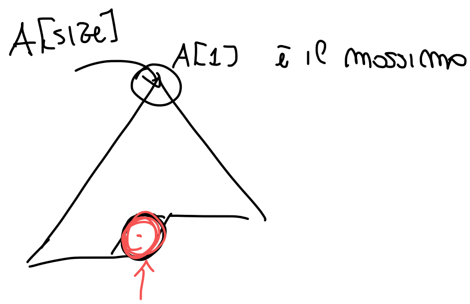
idea : Uso max Heap

$A \qquad \underbrace{A[i] = \text{key}} \qquad A[i] = x$

* **Operazione max**

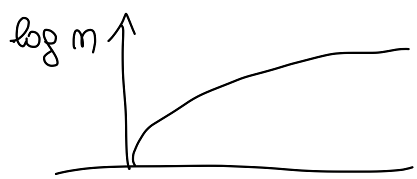
```
Max (A)                    // assumo A.size > 0          Θ(1)
    return A[1]
```

* **Estrazione del massimo**




```
Extract Max (A)
    max = A[1]                 ← Θ(1)
    A[1] = A[A.size]
    A.size = A.size - 1
    Max Heapify(A, 1)          ← O(log n)
    return max
```

$O(\log n)$

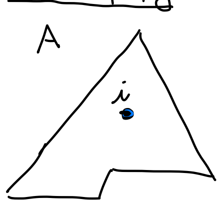


$n = 100 \qquad \log_{10} 100 = 2$

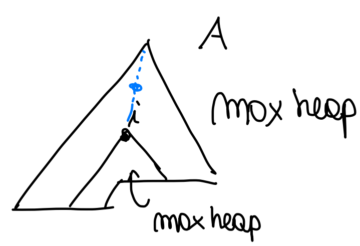
$n = 10 \cdot 10^9 \qquad \log_{10} n = 10$

* **Insert**

→ MaxHeapify (A, i)

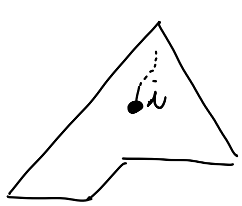


$\forall j \neq i \qquad A[j] \geq$ tutti i discendenti



→ ogni modo $A[i] \leq$ antenati

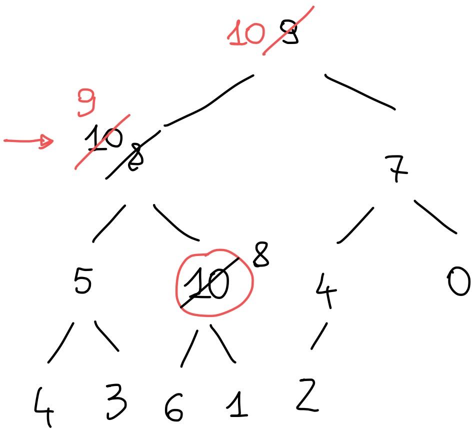
Dato A



$\forall j \neq i \qquad A[j] \leq$ antenati

MaxHeapifyUp (A , i)   trasforma A in un max heap

```
if (i > 1) and (A[i] > A[parent(i)])
    A[i] ↔ A[parent(i)]
    MaxHeapifyUp(A, parent(i))
```

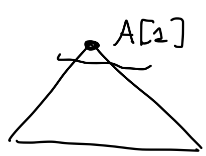


* **Correttezza** :

induzione sul livello di i

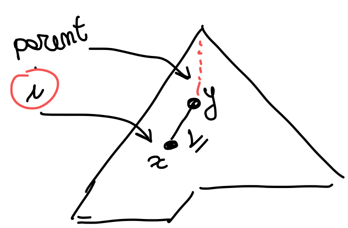
casi base :

- $i = 1$



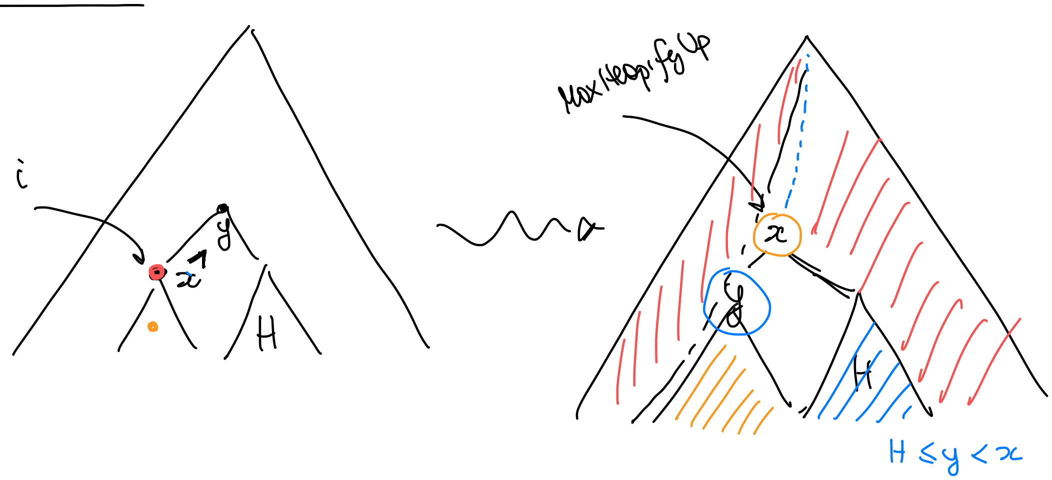
non faccio niente
è già max heap

- $i > 1$, $A[i] \leq A[parent(i)]$



$x \leq y$

caso induttivo $(i > 1)$ e $A[i] > A[parent(i)]$



* Insert (A, k)

ESERCIZIO