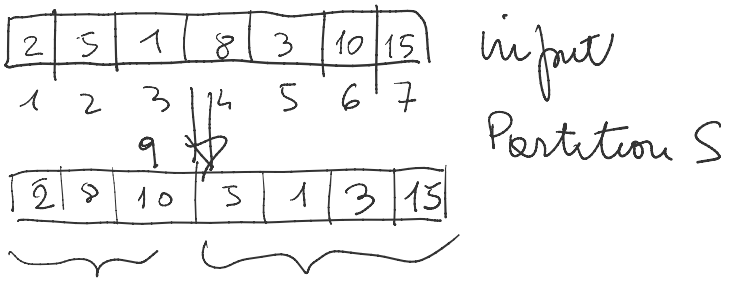
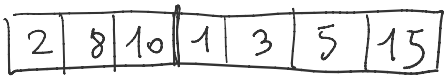
Esercitazione 2 aprile

Thursday, April 2, 2020 3:59 PM



input

Partition S

output 

Partition S divide in pari e dispari

```
QuickSort S (a):
if n > 1 {
    q = PartitionS
    QuickSort (a, 1, q);
    QuickSort (a, q+1, n);
}
```

$O(n \log n)$ in media

Heap

Trovare il minimo in un heap di massimo = max heap

minimo è una foglia

assurdo

Cercare il minimo controllando tutte le foglie.

Proprietà: in un heap ci sono $\lceil \frac{n}{2} \rceil$ foglie ($\lfloor \frac{n}{2} \rfloor$ nodi interni)

Per induzione

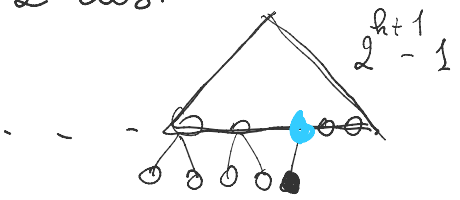
base: $n = 1$

$f = 1$ $\lceil \frac{1}{2} \rceil$ foglie vero

passo induttivo $n \to n+1$

ipotesi induttiva $f = \lceil \frac{n}{2} \rceil$

2 casi



il nuovo nodo è un figlio sinistro

$\Downarrow$

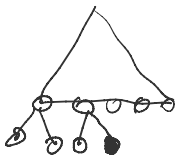
n dispari $\to n+1$ pari

il numero di foglie resta invariato

$$f = \lceil \frac{n}{2} \rceil = \lceil \frac{n+1}{2} \rceil$$ è vero

caso 2 n pari $\to n+1$ dispari

inseriamo il nodo come figlio destro

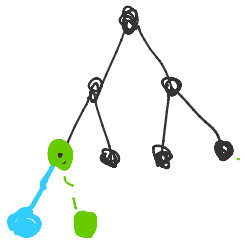


il numero di foglie aumenta di 1.

$$f = \lceil \frac{n}{2} \rceil + 1 = \lceil \frac{n+1}{2} \rceil$$ vero

$n = 10$ $f = 5 + 1 = 6$ $= \lceil \frac{11}{2} \rceil = 6$ vero

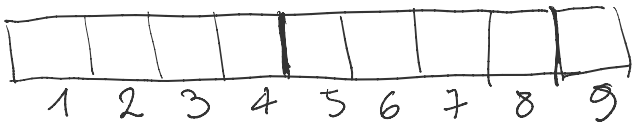
MinMax Heap



$n = 7$
$f = 4$

$n \to n+1$
$8 \to 9$

$f = \lceil \frac{9}{2} \rceil = 5$



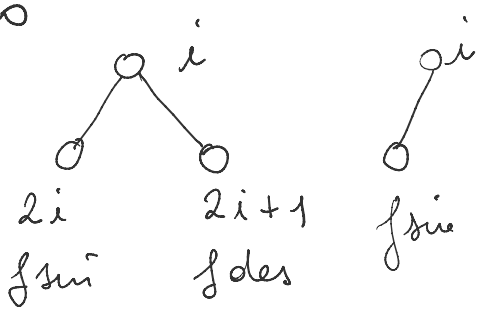

```
MinMaxHeap(a):
min = a[⌈n/2⌉];
for (i = ⌈n/2⌉ + 1; i ≤ n; i++){
    if (a[i] < min)
        min = a[i];
} return min;
```

$\Theta(n) = \frac{n}{2}$

Es 2

Verificare se un array di n elementi è un heap di massimo

i = nodo interno




```
IsHeap?(a)
for (i = 1, i ≤ ⌊n/2⌋, i++){   // per tutti i nodi interni
    if (a[i] < a[2i]) return false;
```

figlio sin è sempre definito

```
if(a[i] < a[2i]) return false;
if((2i+1) ≤ n && a[i] < a[2i+1])
                return false;
}
return true;        Θ(n)
```

Es. 3

Si consideri un max_heap ternario con n nodi e altezza h, definito come segue:

1) $0 \le i < h$ ci sono 3^i nodi a livello i.
2) tutte le foglie a livello h sono addossate a sinistra
3) la chiave memorizzata in un nodo è maggiore delle chiavi memorizzate nei figli

- Descrivere un'implementazione efficiente dell'heap ternario tramite un array a, indicando le formule per calcolare per ogni i

  posizione del padre
  posizione primo figlio
  posizione secondo "
  " terzo figlio

- Descrivere e analizzare le operazioni:

  Heap-Extract-Max (a):
  Max-Heap-Insert (a, k):

Heap-Extract-Max (a):
Max-Heap-Insert (a, k):

Progettare un algoritmo che, dato in input un array a di n elementi distinti e un intero k, restituisca il k-esimo elemento.

1). Progettare una soluzione di costo $O(n \log n)$ tempo

2). Progettare una soluzione di costo $O(n + k \log n)$ tempo

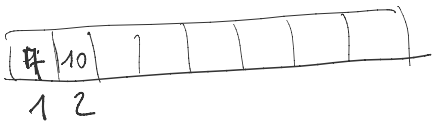
3). Progettare una soluzione di costo $O(n \log k)$

Quick-Select Selezione del k-esimo elemento

se $k = 1$ o $k = n$
min max

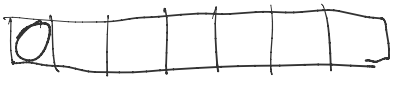
$O(n)$ in media

1) . HeapSort (a)



$O(n \log n)$ tempo

. return a[k] $\Theta(1)$

2) a  Min_BuildHeap $O(n)$

k volte l'estrazione del min

. Heap_Extract_min (a) $O(\log n)$

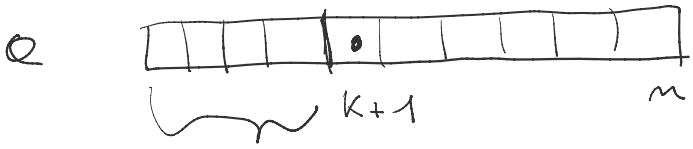
Alla k-esima estrazione

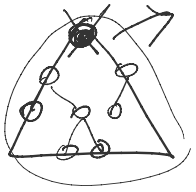
abbiamo come minimo il

abbiamo come minimo il
K-esimo elemento.

$O(n + K \log n)$

(3) Si usa un heap di K elementi
max heap





Max heap di K elementi

$a[i] \quad K+1 \leq i \leq n$

if $a[i] >$ max lo scarto
else Estraggo il max
Max heap Insert (a, i)

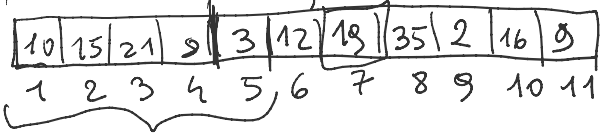
alla fine nell'heap ci stanno
i K elementi più piccoli dell'insieme
il K-esimo è nella radice

```
Select (a, k):
H = nuovo array di k elementi;
for (i = 1; i ≤ k; i++){
    MAX_HEAP_INSERT (H, a[i])          O(k log k)
}
for (i = k+1; i <= n; i++){
    if a[i] < H[1] {
        EXTRACT_MAX (H);               O(n log k)
        MAX_HEAP_INSERT (H, a[i]);
```

}

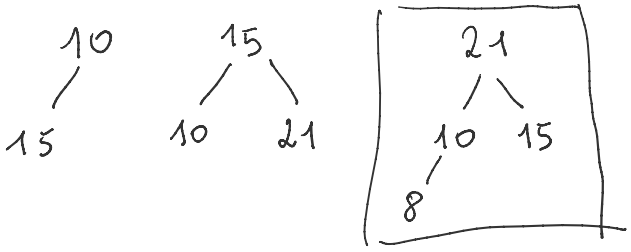
} return H[1];

$O(k \log k + n \log k) = O(n \log k)$



$n = 11$

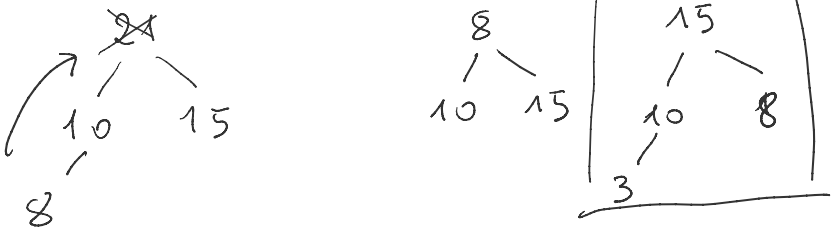
$k = 4$



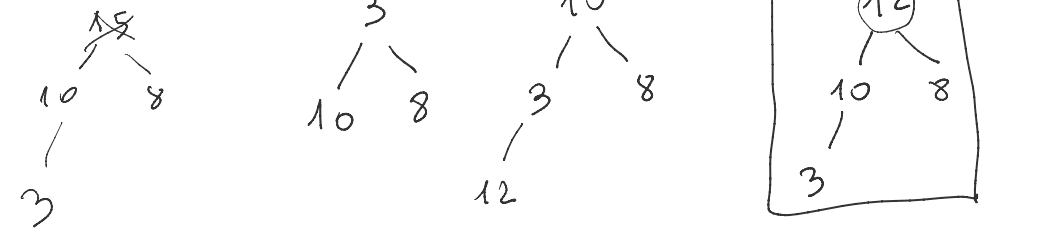
dopo il primo for

max heap di K el.

$Q[i] = 3$



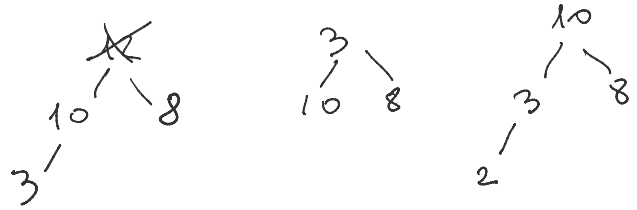
$Q[i] = 12$



$Q[i] = 19$ NIENTE

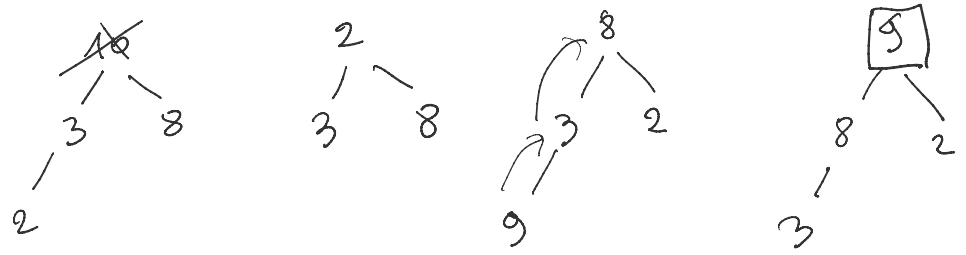
$Q[i] = 35$ 11

$Q[i] = 2$



$Q[i] = 16$ NIENTE

a[i] = 9



Il max heap ad ogni passo i

$k+1 \leq i \leq n$

contiene i k elementi più piccoli visti fino al passo i-esimo.

n sia molto grande

$n \gg k$

memoria limitata a k elementi

Streaming