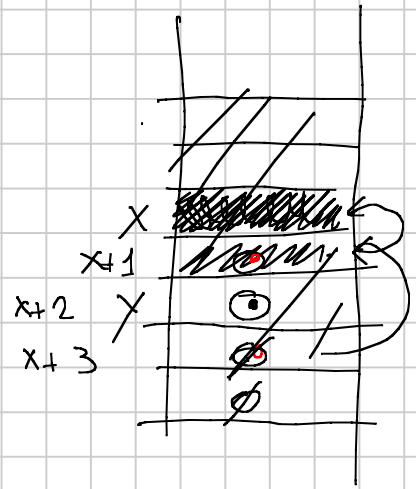
Titolo nota 17/03/2016

Tabelle hash : Cancellazione di K

scansione a passo 1

T

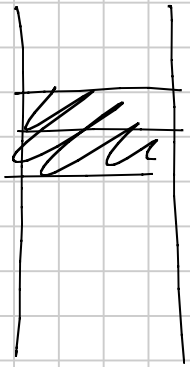


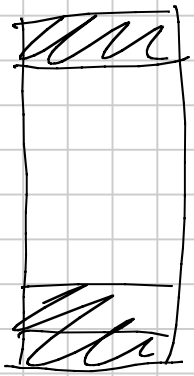
K da cancellare

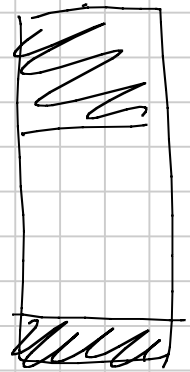
$h(k) = x$

esamino tutte le chiavi che seguono x nell'agglomerato

$n = n - 1$;







$T[0,\dots,m-1]$ n elementi in tabella $h(k)$

HASH-DELETE (T, k): ← è la chiave dell'el. da cancellare

$p = h(k)$;

{ while $(T[p] != k$ && $T[p] != NULL)$ $p = (p+1) \% m$;

if $T[p] == NULL$ "k non c'è";

ricerca di k

$i = (p+1) \% m$ \\ p la posizione liberata, i posizione corrente

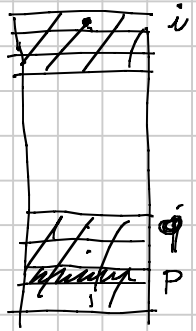
while $T[i] != NULL$ {

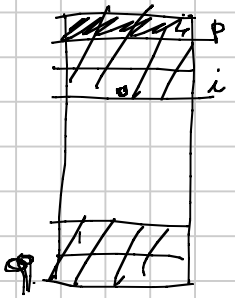
$q = h(T[i])$; if $((q \le p < i) \,||\, (i < q \le p) \,||\, (p < i < q))$

$T[p] = T[q]$; $p = i$;

$i = (i+1) \% m$

} $T[p] = NULL$; $n = n-1$;





HEAP

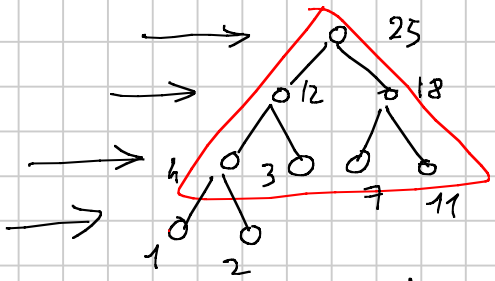
MAX-HEAP

MIN-HEAP

albero binario bilanciato addossato a sin.

MAX : ogni nodo è maggiore dei figli

MIN : " è minore dei figli



H

25	12	18	4	3	7	11	1	2
1	2	3	4	5	6	7	8	9

$1 \dots \lfloor \frac{n}{2} \rfloor$ nodi interni

$\lceil \frac{n}{2} \rceil \dots$ n' foglie

INSERZIONE $O(\log n)$

ESTRAZIONE $O(\log n)$

DEL MAX o DEL MIN

BUILD-HEAP $= \Theta(n)$

i

$parent(i) = \lfloor \frac{i}{2} \rfloor$ $\Theta(1)$

$figliosin(i) = 2i$ $\Theta(1)$

$figliodes(i) = 2i+1$ $\Theta(1)$

altezza $= \lceil \log n \rceil$

ES 1 Dato un MAX-HEAP trovare il minimo.

```
min = H[⌊n/2⌋]
for i = ⌈n/2⌉+1 to n
```

Esamina da $\lceil \frac{n}{2} \rceil$ a n per trovare il min.

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

ES. 2 Progettare un algoritmo che verifichi se gli el. in un array soddisfano la proprietà di heap di minimo.

IS MIN HEAP (H);

```
Is MinHeap(H):
for i = 1 to ⌊n/2⌋ {
  if (H[i] > H[2i]) return false;
  if ((2i+1) ≤ n && H[i] > H[2i+1]) return false;
}
return true;
```

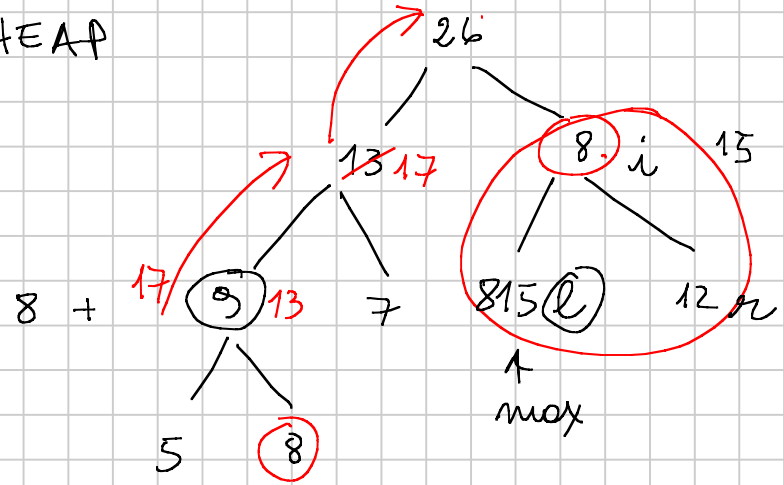
$O(\frac{n}{2}) \equiv O(n)$

Es. 3 H = max Heap size n, Δ un numero intero positivo o negativo

Progettare un algoritmo Cambia(H, i, Δ) che modifica

$H[i] = H[i] + \Delta$

MAX-HEAP




```
MAX-HEAPIFY (A,i):
ℓ = LEFT(i) ; r = RIGHT(i);
if (ℓ ≤ A.heapsize && A[ℓ] > A[i]) max = ℓ;
   else max = i;
   if (r ≤ A.heapsize && A[r] > A[max])
       max = r;
   scambio e continuo con i discendenti
```

$\Delta > 0$

$i = 3$ $\quad \Delta < 0 \quad$ $\Delta = -10$

```
Cambio (H, i, Δ);
H[i] = H[i] + Δ;
if Δ > 0 {
   while (i > 1 && H[parent(i)] < H[i])
       { scambio H[i] ⇔ H[parent(i)];
         i = parent(i); }
   else MAX-HEAPIFY (H, i);
```

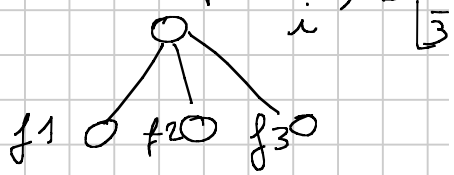
Titolo nota 17/03/2016

Cambio è $O(\log_3 n)$;

Max heap ternario

30	20	15	27	17	12	10	8	13	11
1	2	3	4	5	6	7	8	9	10

parent$(i) = \lfloor \frac{i}{3} \rfloor$

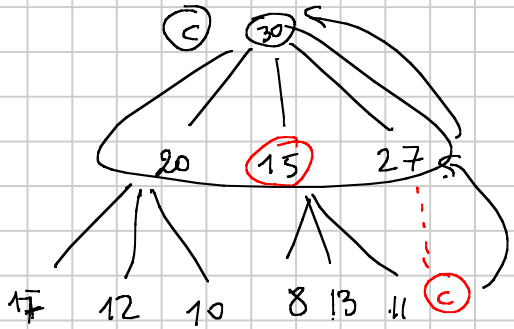


ES. 4

3^0

3^1

$\leq 3^2$



n nodi e altezza h

$n = 10 \quad h = 2$

$f_1(i) = 3i$

$f_2(2) = 3i+1$

$f_3(3) = 3i+2$

1) $0 \leq i < h$ 3^i a livello i;
2) tutte le foglie addossate a sinistra
3) Ogni nodo è maggiore dei figli

$f_1 = 3i - 1$

$f_2 = 3i$

$f_3 = 3i + 1$

parent$(i) = \lfloor \frac{i}{3} \rfloor$

Indicare le formule per passare

Si dimostra per induzione

Titolo nota 17/03/2016

A e un intero K di n elementi distinti restituisca il K-esimo elemento più piccolo di A

1) . Alg. $O(n \log n)$

2) . Alg $O(n + k \log n)$

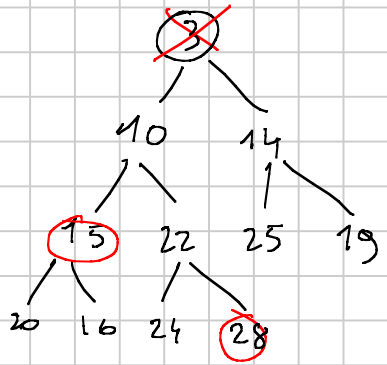
3) . Alg $O(n \log K)$

HeapSort $O(n \log n) + \Theta(1) \Rightarrow A[K]$

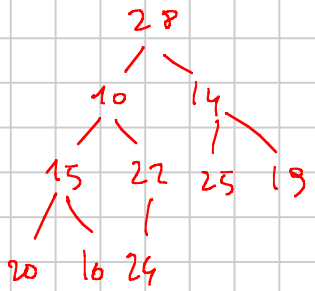
MIN-HEAP con BUILDHEAP $O(n)$ poi
K-1. HEAP-EXTRACT MIN $O(K \log n)$

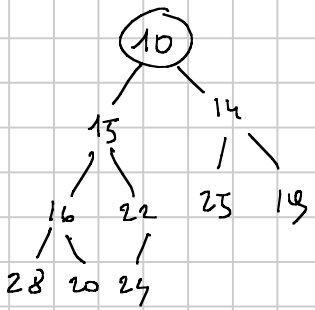
$K = 4$

MIN-HEAP

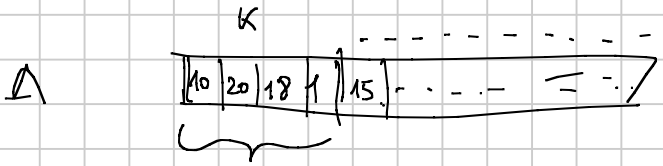


HEAPEXTRACT MIN

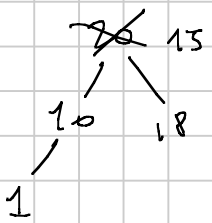




Si costruisce un heap di massimo di K elementi



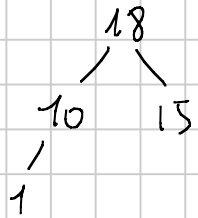
$\Theta(n \log k)$ e risultato in H[1]



se A[i] > H[1] non lo considero

else elimino 20 e inserisci A[i]

alla fine ho un heap di max con K elementi



~~radice~~

soluzione nella radice

```
K-esimo (A);
"H è nuovo array di K elementi"                          O(K log K)
costruisci Heap di K elementi   for i = 1 to K   MAX-HEAP-INSERT(H, A[i])
analizzo i restanti di A        for i = K+1 to n {       O(n log K)
                                    if A[i] < H[1] {
                                        EXTRACTMAX(H);
                                        MAX-HEAP-INSERT(H, A[i]
                                        }
K-esimo nella radice del max heap   } return H[1];
```