TABELLE HASH

indirizzamento aperto.

scansione quadratica

$S = \{10, 22, 31, 4, 15, 28, 17, 88, 59, 26\}$

$T =$

0	1	2	3	4	5	6	7	8	9	10
22	88		59	4	15	28	17	26	31	10

$$h(k,i) = \left(k \bmod 11 + \frac{1}{2}i + \frac{1}{2}i^2\right) \bmod 11$$

$i = 1 \rightarrow 1$
$i = 2 \rightarrow 3$
$i = 3 \rightarrow$
$i = 4$
$i = 5$

chiave	sequenza di scansione
10	10
22	0
31	9
4	4
15	4, 5
28	6,
17	6, 7
88	0, 1
59	4, 5, 7, 10, 3
26	4, 5, 7, 10, 3, 8

Hashing doppio

$U = \{10, 22, 31, 4, 15, 28, 17, 88, 59, 26\}$

$h(k,i) = \left(\underbrace{k \bmod 11}_{h} + i\left(\underbrace{1 + k \bmod 10}_{h'}\right)\right) \bmod 11$

chiave	$h(k)$	$h'(k)$	sequenza
10	10		10
22	0		0
31	9		9
4	4		4
15	4	6	4, 10, 5
28	6		6
17	6	8	6, 3
88	0	9	0, 9, 7
59	4	10	4, 3, 2
26	4	7	4, 0, 7, 3, 10, 6, 2, 9, 5, (1)

$T =$

22	0
26	1
59	2
17	3
4	4
15	5
28	6
88	7
	8
31	9
10	10

ESERCITAZIONE: HEAP

ESERCIZIO

A: array di n interi distinti
k: intero $1 \leq k \leq n$

Progettare un algoritmo che restituisca l'elemento di rango k di A
(k-esimo elemento più piccolo di A).

① Costo $T(n,k) = O(n \log n)$

Rango1(A, k)

```
HeapSort(A);
return A[k];
```

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

② $T(n,k) = O(n + k \log n)$

```
Rango2(A, k)
Build_Min_Heap(A);
for j = 1 to k-1 {
        Heap_Extract_Min(A);
}
return Heap_Minimum(A)
```

$\Theta(n)$

$\{ O((k-1)\log n) = O(k \log n)$

$\Theta(1)$

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

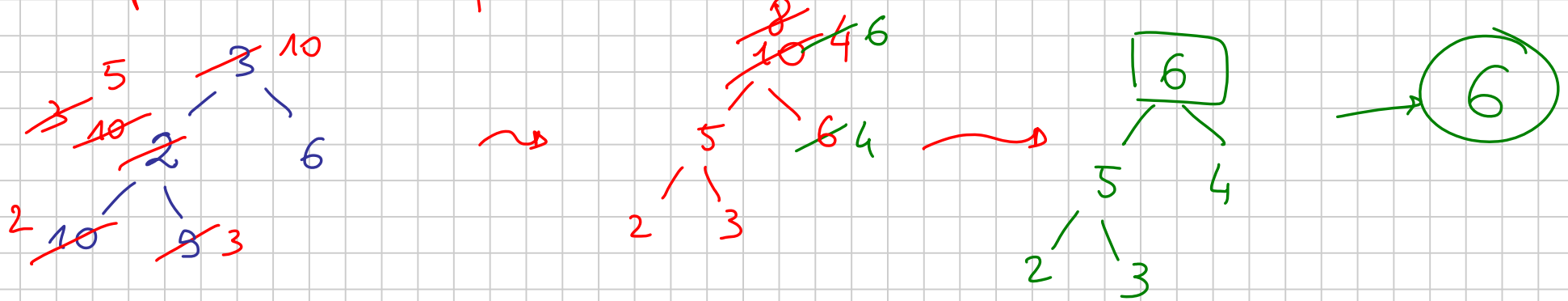
③ $T(n,k) = O(n \log k)$

$A =$ | 3 | 2 | 6 | 10 | 5 | 8 | 4 | 7 |

$k = 5$

(k)

- Heap di massimo sui primi 5 elementi di A



RangoK(A, k)

H = nuovo array di dim k, per la rappresentazione implicita di un heap di massimo

H.heapsize = k

$(k \log k)$ {
for i = 1 to k {
    Max-Heap-Insert(H, A[i]);
}
}

// H contiene un heap di massimo formato dai primi k elementi di A

$O((n-k)\log k)$ {
for i = k+1 to n {
    if (A[i] < H[1]) {
        H[1] = A[i];
        Max-Heapify(H, 1);
    }
}
}

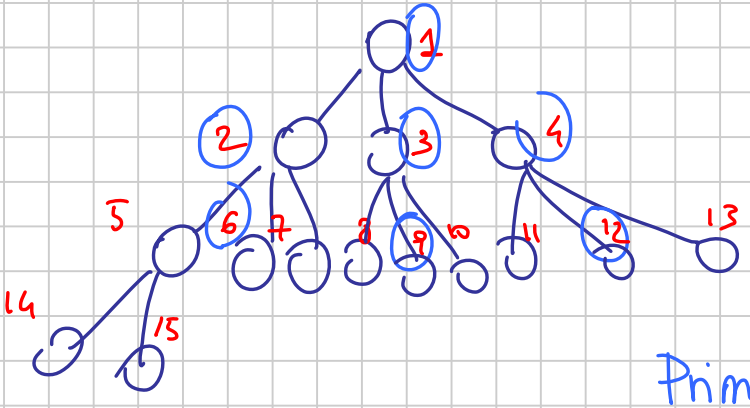
// H è un heap di massimo che contiene i k elementi più piccoli di A

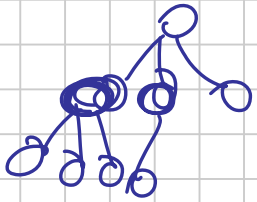
```
return H[1];
```

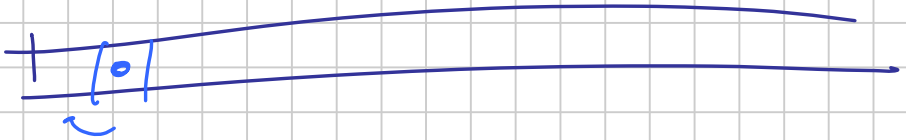
$$T(n,k) = O(k \log k) + O((n-k)\log k) = O(n \log k)$$

Titolo nota 26/02/2016

MAX-HEAP ternario







Primo(i)

return $3*i - 1$

Secondo(i)

return $3*i$

Terzo(i)

return $3*i + 1$

Padre(i)

$\left\lfloor \frac{i+1}{3} \right\rfloor$

$\forall i$ $\quad Padre(Primo(i)) = i$

$Padre(Secondo(i)) = i$

$Padre(Terzo(i)) = i$

$\forall i,\quad Padre(Primo(i)) = Padre(3i-1) = \left\lfloor \frac{(3i-1)+1}{3} \right\rfloor = \left\lfloor \frac{3i}{3} \right\rfloor = i$ ✓

$Padre(Secondo(i)) = Padre(3i) = \left\lfloor \frac{3i+1}{3} \right\rfloor = \left\lfloor i + \frac{1}{3} \right\rfloor = i$ ✓

$Padre(Terzo(i)) = Padre(3i+1) = \left\lfloor \frac{3i+1+1}{3} \right\rfloor = \left\lfloor i + \frac{2}{3} \right\rfloor = i$ ✓

Max-Heapify(A, i)

```
p = Primo(i)
s = secondo(i)
t = terzo(i)          → max = i
if (p ≤ A.heapsize && A[p] > A[max])  max = p;
if (s ≤ A.heapsize && A[max] < A[s])  max = s;
if (t ≤ A.heapsize && A[max] < A[t])  max = t;
if (max ≠ i) {
        Scambia A[i] e A[max];
        Max-Heapify(A, max);
}
```

$\Theta(1)$

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

Max-Heap-Insert(A, key)

```
A.heapsize++;
A[A.heapsize] = key
i = A.heapsize;
while ( i > 1 && A[Padre(i)] < A(i)) {
        scambia A(i) e A[Padre(i)];
        i = Padre(i);
}
```

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

ESERCIZIO

Sono dati due heap di massimo: heap1 e heap2, di dimensione m e n, con $n > m$.
Progettare un algoritmo che restituisca un nuovo heap composto dall'intersezione degli elementi contenuti in heap1 e heap2.

$m \log n$
$+ n \log n$
$(m+n) \log n$

$n \log m$
$+ m \log m$
$(n+m) \log m$

```
HeapIntersect (heap1, heap2)
H = nuovo heap inizialmente vuoto
H.heapsize = 0;
while (heap1.heapsize > 0 && heap2.heapsize > 0) {
    if (Heap_Maximum(heap1) == HeapMaximum(heap2)) {
        e = Heap_Extract-Max (heap1);
        Max-Heap-Insert (H, e);
        Heap_Extract_Max (heap2);
    }
    else if (Heap_Maximum (heap1) > Heap_Maximum (heap2)) {
        HeapExtract Max (heap1);
    else HeapExtract Max (heap2);
}
return H;
```