Correzione compitini 31/03

ESERCIZIO

$[1, n+1]$

a ordinato.

1. Algoritmo lineare per trovare l'intero mancante

```
Cerca1(a)
i = 1
while( i ≤ n && a[i] == i) i++;
return i;
```

$T(n) = O(n)$

CASO OTTIMO l'intero mancante è 1 $\Rightarrow T(n) = \Theta(1)$

$C(n) = 1$ confronto

CASO PESSIMO l'intero mancante è $n+1$ $\Rightarrow T(n) = \Theta(n)$

$C(n) = n$

CASO MEDIO

osservazione: se l'intero mancante è i $\Rightarrow$ $C(n) = i$

$C(n) = \#$ confronti

$$T_{medio}(n) = \frac{1}{n+1} \cdot \left(\sum_{i=1}^{n} i + n \right)$$

$\frac{1}{n+1}$: casi possibili; $\sum_{i=1}^{n} i$: intero mancante è i; n: intero mancante $i+1$

$$= \frac{1}{n+1} \left(\frac{n(n+1)}{2} + n \right) = \frac{n}{2} + \frac{n}{n+1} \leq \frac{n}{2} + 1 \quad \text{confronti}$$

$$T_{medio}(n) = \Theta(n)$$

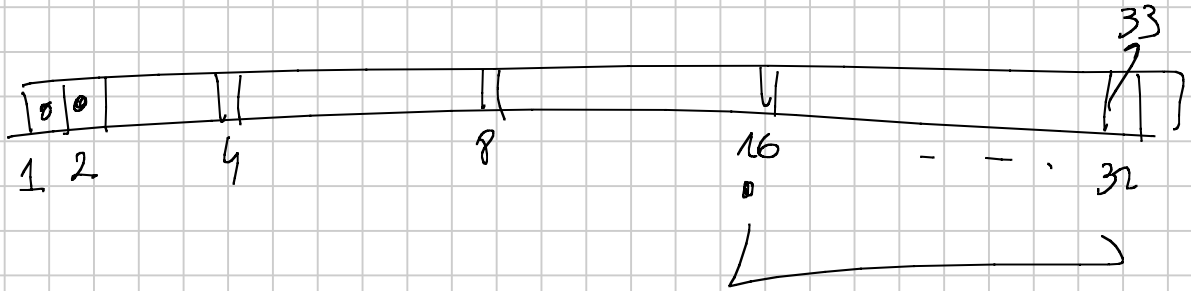
2.

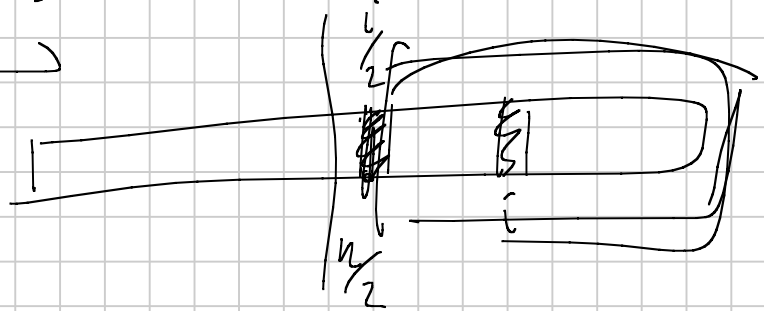
```
Cerca2 (a, sx, dx)
if ( sx > dx) return n+1;
if (sx == dx)
        if ( a[sx] == sx) return n+1;
        else return sx;
else {
        cx = (sx+dx)/2
        if (a[cx] == cx) return Cerca2( a, cx+1, dx);
        else return Cerca2 (a, sx, cx);
}
```

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ T\left(\frac{n}{2}\right) + \Theta(1) & n > 1 \end{cases}$$

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

3. Algoritmo di costo $\Theta(\log k)$ k = intero mancante





Cerca3(a)

$\Theta(\log k)$ [$i = 1$
while ($i \le n$ && $a[i] == i$) $i = 2 * i$;
if ($i > n$) // $\frac{n}{2}+1 < \frac{i}{2}+1 \le k \le n+1$ $\quad k \in [\frac{n}{2}+1, n)$
① return Cerca2(a, $\frac{i}{2}+1$, n); $\quad (k = \Theta(n)$
else
② return Cerca2(a, $\frac{i}{2}+1$, i) // $\frac{i}{2}+1 \le k \le i$

Analisi

① $k \geq \frac{n}{2}$

$\frac{n}{2}+1 \leq k \leq n+1 \Rightarrow k = \Theta(n)$

Cerca2 è chiamata su una porzione di array di dimensione $\frac{n}{2}$

↳ Costo $\Theta(\log n) = \underset{k=\Theta(n)}{\Theta(\log k)}$

② $\frac{i}{2}+1 \leq k \leq i \Rightarrow k = \Theta(i)$

Cerca2 è chiamata su una porzione di dimensione $\frac{i}{2}$

Costo $\Theta(\log i) = \underset{k=\Theta(i)}{\Theta(\log k)}$

Costo del while eseguito $\lceil \log_2 k \rceil$ volte.

ESERCIZIO 2

```
MergeSort2(a, sx, dx)
if (sx < dx) {
    cx = (sx+dx)/2
    MergeSort2(a, sx, cx)
    QuickSort(a, cx+1, dx)
    Merge(a, sx, cx, dx)
}
```

$T(n)$

$\Theta(1)$ } $T(n/2)$ } $O(n^2)$ } $\Theta(n)$

$T(n) = T(n/2) + O(n^2) + \Theta(n) + \Theta(1) = T(\frac{n}{2}) + O(n^2)$

Soluzione della ricorrenza (th. esperto)

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

$a = 1$, $b = 2$ $\quad n^{\log_2 1} = n^0 = \Theta(1)$

$$f(n) = \Theta(n^2) = \Omega\left(n^{\log_b a + \varepsilon}\right) = \Omega(n^{\varepsilon})$$

$0 < \varepsilon \leq 2$

+ verifica cond. di regolarità

$f(n) = c \cdot n^2$ ✓

III° caso

$$\boxed{T(n) = \Theta(n^2)}$$

ESERCIZIO 3

n monete
una falsa (più leggera)

1. algoritmo $O(\log n)$
   - con $\log_3 n$ pesate

1) $n \% 3 = 0$

3 gruppi da $\frac{n}{3}$ monete ciascuno: $G1, G2, G3$

peso 2 gruppi: $G1, G2$

$G1 == G2 \Rightarrow$ ricomo su $G3$

se $G1 < G2 \Rightarrow$ " " $G1$

se $G1 > G2 \Rightarrow$ " " $G2$

BASE $n \leq 3$

- risolvo con una sola pesata

2) $n \% 3 = 1$

$n = 3k+1$

3 gruppi: k k $k+1$

confronto i due gruppi di k monete

se i due gruppi hanno lo stesso peso ricerco nel gruppo di $k+1$ monete, altrimenti nel più leggero dei due

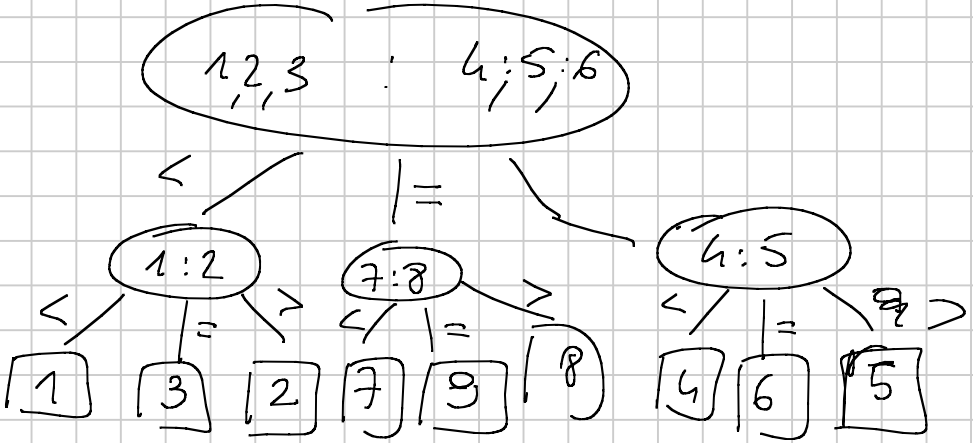
3) $n \% 3 = 2$ $n = 3k+2$

3 gruppi $k+1$ $k+1$ k

confronto i due gruppi di $k+1$ monete
e procedo come al punto precedente

2. il lim. inferiore è $\log_3 9 = 2$

3.



ESERCITAZIONE: heap

1) Progettare un algoritmo che verifichi se gli elementi memorizzati in un array soddisfano la proprietà di heap di massimo.

```
IsHeap(a)
    for i = 1 to ⌊n/2⌋ {                // nodi interni dello heap
        if(a[i] < a[2i]) return false;
        if((2i+1) ≤ n && a[i] < a[2i+1]) return false;
    }
    return true
```

$T(n) = O(n)$

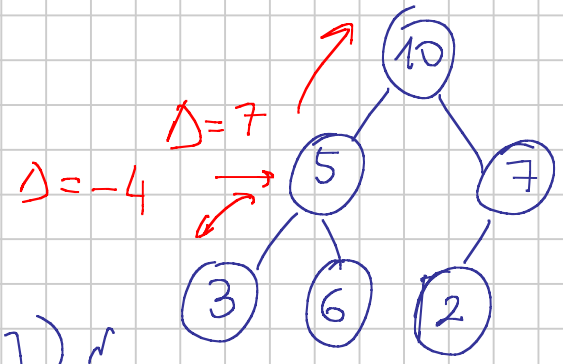
ottimo

(2) Sia H uno heap di dimensione n.
Progettare un algoritmo

CAMBIA(H, i, Δ) $\quad 1 \le i \le n$

che modifica la chiave contenuta in $H[i]$ come

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




```
Cambia(H, i, Δ)
H[i] = H[i] + Δ;
if (Δ > 0) {
    while( i > 1 && H[Parent(i)] < H[i] ) {
        Scambia H[i] e H[Parent(i)];
        i = Parent(i);
    }
}
else Max-Heapify(H, i);
```

Analisi

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

PROGETTARE UN ALGORITMO CHE RICEVUTO IN INPUT UN INTERO k e UN ARRAY A di n ELEMENTI DISTINTI, RESTITUISCA IL k-ESIMO ELEMENTO PIÙ PICCOLO in A.

1) SOLUZIONE DI COSTO IN TEMPO $O(n \log n)$
2) " " " " $O(n + k \log n)$
3) " " " " $O(n \log k)$ che usi uno heap su k elementi.