ESERCITAZIONE

① Più 0 che 1. Array NON ordinato

Limite inferiore $\Omega(n)$ (Dimensione Input)

② Più 0 che 1 Array ordinato

Limite inferiore ? $S(n) = 2$ $L(n) = \Omega(\log 2) = \Omega(1)$

Algoritmo ottimo ? quando l'elemento centrale

Contare il # di 0 in un array ordinato di 0 e 1.

LIMITE INFERIORE

$S(n) = n+1$ $\qquad 0 \leq \#0 \leq n$

~~$S(n) = \log$~~

$L(n) = \Omega(\log S(n)) = \Omega(\log n)$

Algoritmo ottimo

Compitino Aprile 2015

A: array di n interi distinti.

Trovare la coppia $A[i], A[j]$, $i \neq j$, che minimizza la differenza in valore assoluto tra le coppie di elementi

se non ordino: $\Theta(n^2)$

se ordino, la coppia va cercata tra le coppie adiacenti

$$T(n) = \underbrace{\Theta(n \log n)}_{\text{x ordinare}} + \underbrace{\Theta(n)}_{\substack{\text{ricerca della coppia} \\ \text{nell'array ordinato}}} = \Theta(n \log n)$$

CercaCoppia(A) // n = dim di A, n ≥ 2

```
HeapSort(A);
diff = A[2] - A[1];
coppia = <A[1], A[2]>
for i = 3 to n {
    if (A[i] - A[i-1] < diff) {
        diff = A[i] - A[i-1];
        coppia = <A[i-1], A[i]>
    }
}
return coppia;
```

Limite inferiore (Albero di decisione)

$$L(n) = \Omega(\log S(n))$$

$S(n)$ = # soluzioni = # coppie $= \binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)$

$$L(n) = \Omega\left(\log \binom{n}{2}\right) = \Omega(\log n^2) = \Omega(\cancel{2}\log n) = \Omega(\log n)$$

NON SIGNIFICATIVO

Dim. INPUT

$$L(n) = \Omega(n)$$

4) trovare la coppia che massimizza la differenza in valore assoluto.

↳ soluzione: coppia $\langle min, max \rangle$

$min \to \Theta(n)$ $max \to \Theta(n)$

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

Limite inferiore (dimensione input)

$L(n) = \Omega(n)$.

COMPITINO APRILE 2014

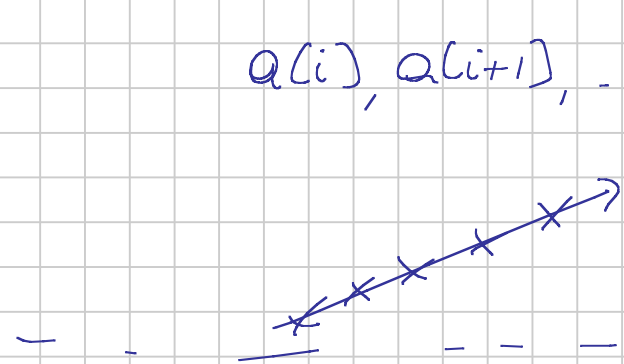
(4) trovare il compagno tra 25 cavalli da corsa.
corse di 5 cavalli
contro l'orologio di amico (non il tempo).
Limite inferiore al numero minimo di corse necessarie

↳ 6 gare sono sufficienti
(ogni gara permette di individuare 4 perdenti)
24 cavalli devono essere dichiarati perdenti.

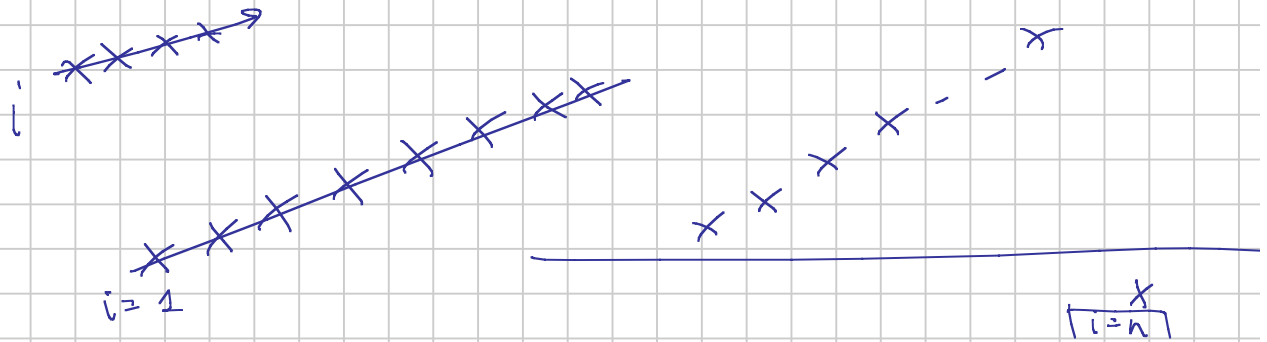
(2) a: array di n interi distinti CICLICAMENTE ORDINATO

$\exists i : 1 \le i \le n$ t.c.

$a[i], a[i+1], \ldots, a[n], a[1], \ldots, a[i-1]$

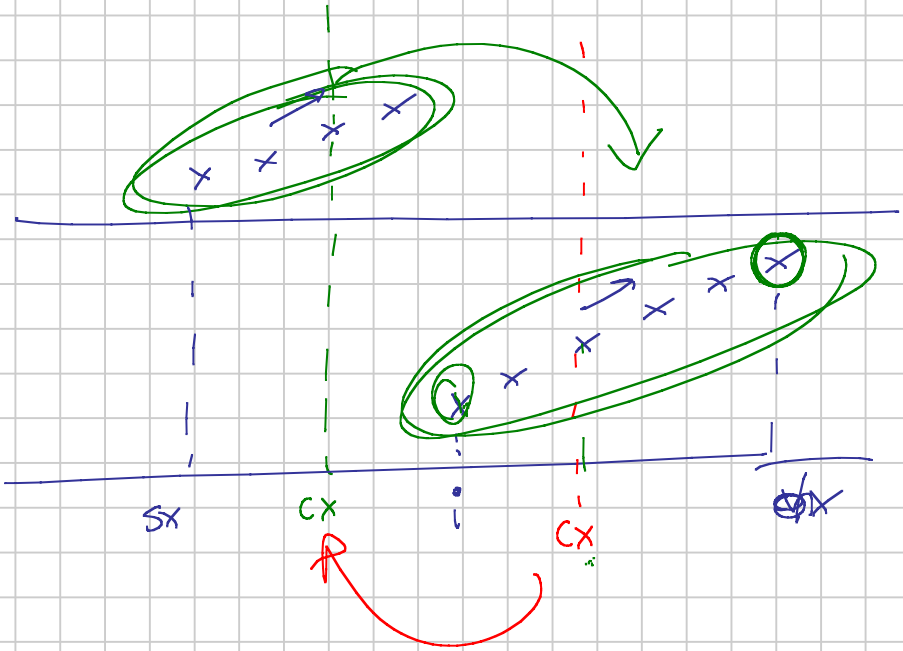


trovare i



① Limite inferiore

$$L(n) = \Omega(\log \text{scelta}) = \Omega(\log n)$$




```
Trova i (a, sx, dx)
if (sx > dx) return -1;
if (sx == dx) return sx;
cx = (sx + dx) / 2
if (a[cx] < a[dx]) return Trova i (a, sx, cx);
else return Trova i (a, cx+1, dx)
```

Titolo nota 26/02/2016

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

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

ESERCIZIO

- array a di n interi non necessariamente distinti
- un elemento è DI MAGGIORANZA se appare in a almeno $\lceil \frac{n}{2} \rceil$ volte

Progettare un algoritmo di complessità

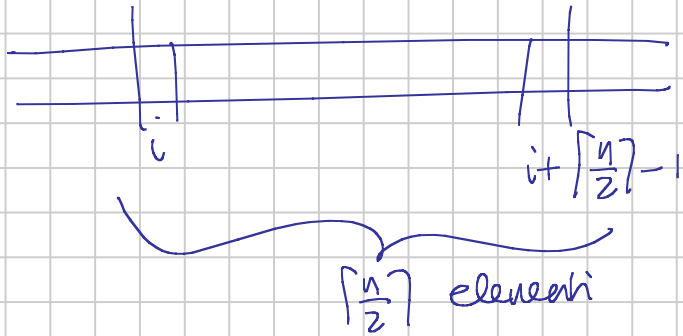
1) $O(n^2)$
2) $\Theta(n \log n)$

che stabilisca se a contiene un elemento di maggioranza, e in caso affermativo lo restituisca

Maggioranza (a)

HeapSort(a)

```
i = 1;
while (i ≤ n/2) {
    if( a[i] == a[ i + ⌈n/2⌉ - 1 ]) return <trovato: a[i]>
    else i++;
}
return <non TROVATO>
```



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

Esercizio (Appello Giugno 2013)

- array A di n chiavi, ORDINATO
- Progettare un algoritmo D&I per la ricerca di una chiave, dove la ripartizione in due sottoarray avviene tramite selezione casuale di una posizione dell'array.
- Studiare la complessità dell'algoritmo al caso ottimo, medio, peggiore.

```
RicercaRandom (a, sx, dx, k)
if (sx > dx) return -1;
if (sx == dx) {
        if (a[sx] == k) return sx;
        else return -1;
}
cx = Random (sx, dx);
if (a[cx] == k) return cx;
else if (a[cx] < k) return RicercaRandom (a, cx+1, dx, k);
     else return RicercaRandom (a, sx, cx-1, k)
```

CASO PESSIMO

$k \notin Q$

partizione sbilanciata

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ T(n-1) + c & n > 1 \end{cases}$$

c costante.

$$T(n) = T(n-1) + c = T(n-2) + c + c = T(n-2) + 2c = T(n-3) + c + 2c =$$

$$= T(n-3) + 3c = \dots = T(n-i) + ic = T(n-(n-1)) + (n-1)\cdot c =$$

$i = n-1$

$$= T(1) + (n-1)c = \Theta(n)$$

$T(1) = \Theta(1)$

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1) = \Theta(\log n)$$

Partizioni di prop. costante.

$\frac{1}{k} \cdot n$ $\qquad$ $\left(1 - \frac{1}{k}\right) \cdot n$

$\frac{n}{k}$ $\qquad$ $\left(\frac{k-1}{k}\right) n$

$$T(n) = T\left(\left(\frac{k-1}{k}\right) n\right) + \Theta(1)$$

$a = 1$

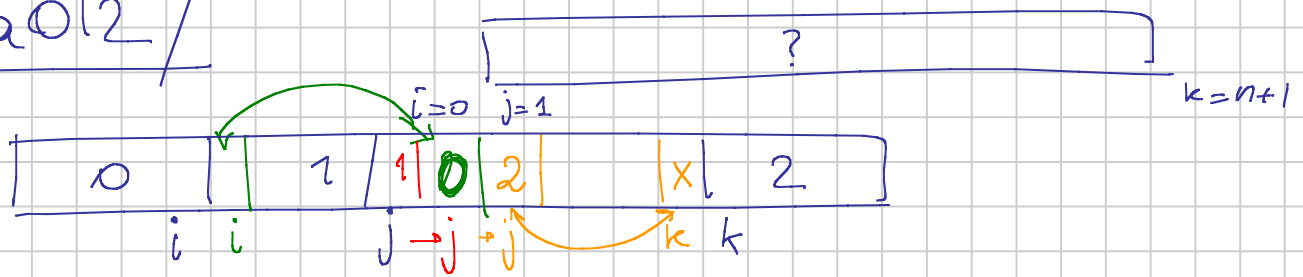
$b = \frac{k}{k-1}$

$n^{\log_b a} = n^{\log_{\frac{k}{k-1}} 1} = n^0 = \Theta(1)$

$f(n) = \Theta(1)$

Th. Esperto, II° caso $T(n) = \Theta(\log n)$

OrdinaO12




```
OrdinaO12(a)
i=0 // ultimo 0
k=n+1 // primo 2
j=1
while (j<k) {
    if (a[j]==0) { i++; scambia(a[i],a[j]); j++ }
    else if (a[j]==2) { k--; scambia(a[k],a[j]); }
    else // a[j]==1
        j++
}
```