ORDINAMENTI PER CONFRONTI

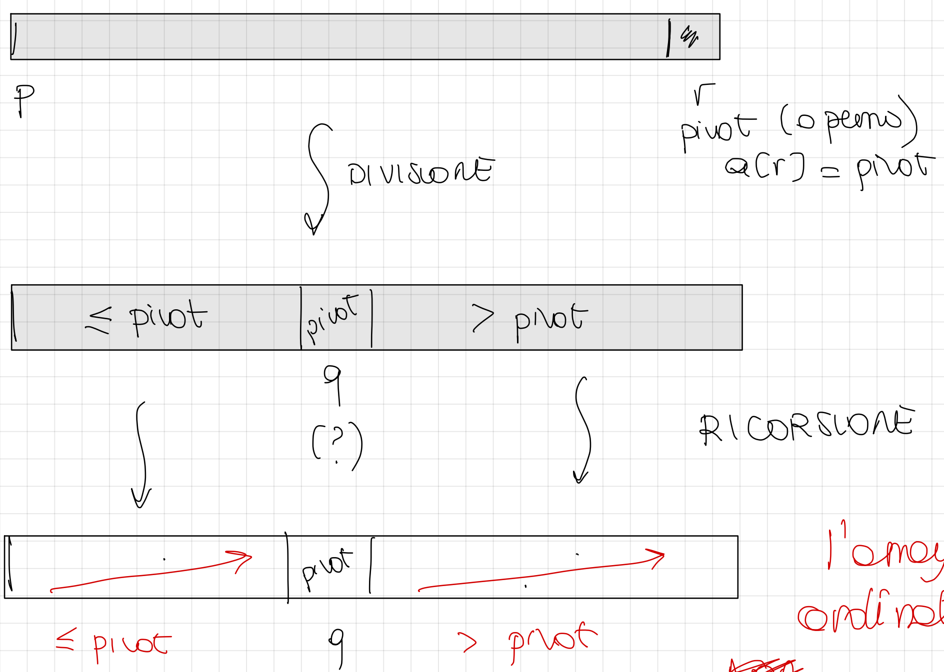
	COSTO IN TEMPO: CASO OTTIMO	COSTO IN TEMPO: CASO MEDIO	COSTO IN TEMPO: CASO PESSIMO	COSTO in SPAZIO	commenti
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	in loco	
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	in loco	
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	OTTIMO in tempo (anche al caso medio)
QuickSort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	in loco	OTTIMO in tempo al caso medio

QuickSort:
→ efficiente al caso medio
→ costanti nascoste dalla notazione asintotica piccole

ma occorre contare lo spazio necessario per gestire le chiamate ricorsive simultaneamente aperte
$\Theta(n)$ caso pessimo
$\Theta(\log n)$ caso medio e ottimo

QuickSort (ordinamento per distribuzione)

Hoare, 1962



l'array è ordinato

La combinazione è vuota

8	2	7	1	3	5	6	4

pivot = 4

r

↓ DIVISIONE

pivot

2	1	3	4	7	5	6	8

$\underbrace{2\ 1\ 3}_{\leq 4}$ q $\underbrace{7\ 5\ 6\ 8}_{\geq 4}$

↓ ↓

1	2	3	4	5	6	7	8

```
QuickSort(A, p, r)
BASE        if (p<r) {                 // se ci sono almeno due elementi
DIVISIONE       q = PARTITION(A, p, r);   // il pivot è già
                                          sistemato in ordine,
                                          in posizione q
RICORSIONE      QuickSort(A, p, q-1);
                QuickSort(A, q+1, r);
            }
```

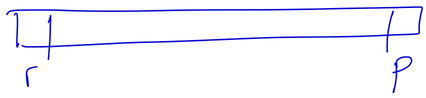
la combinazione è vuota

Prima chiamata:

QuickSort(A, 0, A.length - 1)

Partition: costo

$n = r - p + 1$




```
PARTITION(A, p, r)
 x = A[r]
 i = p-1
 for (j = p, j < r, j++) {
      if (A[j] ≤ x) {
        i++;
        Scambia A[i] con A[j];
      }
 }
 Scambia A[i+1] con A[r];
 return i + 1;
```

$O(1)$

→ si ripete $r-p$ volte

costo $O(1)$

$O(1)$

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

PARTITION

$n = r - p + 1$

(elementi nella porzione) $A[p \dots r]$

Costo in TEMPO

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

NUMERO di CONFRONTI

$C(n) = n - 1$

tutti gli elementi di $A[p \dots r-1]$ si confrontano con il pivot $A[r]$

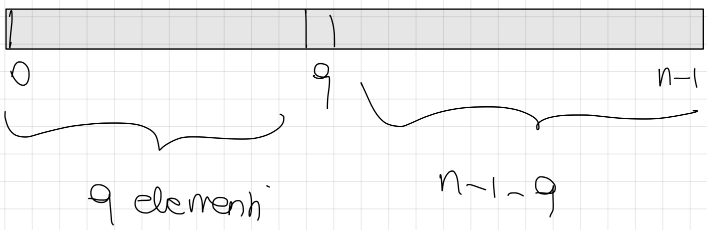
COSTO IN SPAZIO

$S(n) = \Theta(1)$ in loco senza array di appoggio

Analisi di QuickSort

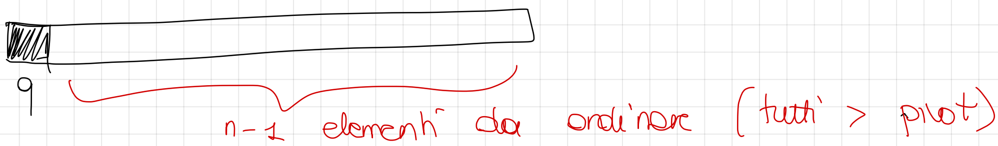
TEMPO di ESECUZIONE: dipende dal bilanciamento della partizione

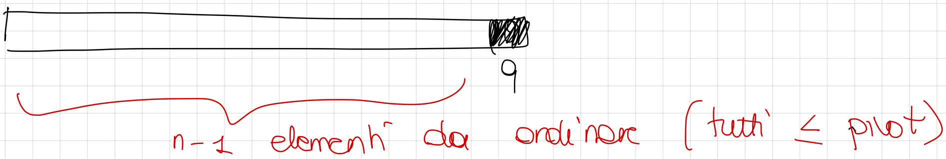
$$T(n) = \begin{cases} O(1) & n \leq 1 \\ T(q) + T(n-1-q) + \underbrace{\Theta(n)}_{\text{costo di Partition}} & n > 1 \end{cases}$$



CASO PESSIMO

quando il partizionamento produce un sottoproblema con n-1 elementi e uno con 0
↳ in ogni chiamata ricorsiva





Se il partizionamento sbilanciato si verifica in ogni chiamata ricorsiva:

$$T(n) = T(n-1) + T(0) + \Theta(n) \qquad n > 1$$
$$T(n) = O(1) \qquad n \leq 1$$

$\Theta(1)$

$$T(n) = T(n-1) + \Theta(n) \qquad n > 1$$

non si può applicare il teorema principale

$$T(n) = T(n-1) + c \cdot n = T(n-2) + c(n-1) + cn =$$
$$= T(n-3) + c(n-2) + c(n-1) + cn = T(n-3) + c\sum_{j=0}^{2}(n-j)$$
$$= \dots = T(n-i) + c\sum_{j=0}^{i-1}(n-j) \underset{i=n-1}{=} T(1) + c\sum_{j=0}^{n-2}(n-j) =$$

$$n-i = 1 \Rightarrow i = n-1$$

$$= O(1) + c\sum_{j=0}^{n-2}(n-j) = O(1) + c\,(n + n-1 + n-2 + + \dots + 2)$$
$$= O(1) + c \cdot \sum_{j=2}^{n} j = O(1) + \left(\frac{n(n+1)}{2} - 1\right) \cdot c = \Theta(n^2)$$

Dim. segmenti array	# confronti
n	$n-1$
$n-1$	$n-2$
$\vdots$	$\vdots$
3	2
2	1

partizioni sempre sbilanciate al massimo

$$C(n) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \binom{n}{2}$$

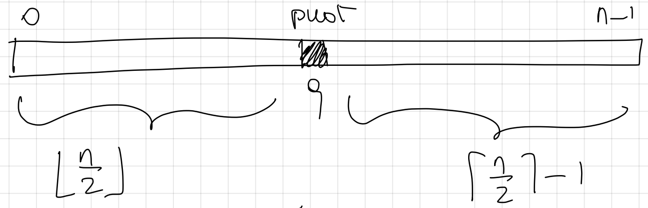
al caso pessimo, QuickSort fa tutti i confronti possibili!

si verifica se il pivot è sempre l'elemento massimo o l'elemento minimo del segmento $A[p \dots r]$ di lavoro

↳ ad esempio se l'array è già ordinato.

Comportamento al caso ottimo

partizioni sempre bilanciate, in ogni chiamata ricorsiva:



$q = \lfloor \frac{n}{2} \rfloor$

$$T(n) = \begin{cases} O(1) & n \leq 1 \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil - 1\right) + \Theta(n) & n > 1 \end{cases}$$

↳ $T(n) = 2T(n/2) + \Theta(n)$

$T(n) = \Theta(n \log n)$ come MergeSort!