QUICKSORT

Basato sul paradigma divide-et-impera (come MERGE-SORT)

- Divide: stabilisce un valore di q tale da dividere l'array A[p .. r] in due sottoarray non vuoti A[p .. q] e A[q+1 .. r], dove ogni elemento del primo assume un valore ≤ rispetto a ogni elemento del secondo
- Impera: ciascuno del due sottoarry è ordinato in loco invocando ricorsivamente QUICKSORT
- (Combina: non è necessaria alcuna operazione)

QUICKSORT (cont.)

```
QUICKSORT(A,p,r)
1 if p < r
2  ▷ A[p .. r] è il sottoarray da ordinare;
   se p ≥ r il sottoarry ha al più un elemento ed è perciò già ordinato
3   then    q ← PARTITION(A,p,r)
4           QUICKSORT(A,p,q)
5           QUICKSORT (A,q+1,r)
```

Per ordinare l'intero array A si invoca QUICKSORT (A, 1, length[A])

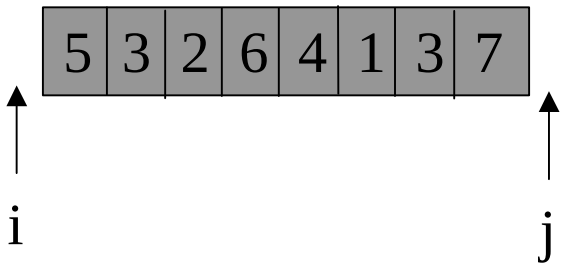
Se A contiene un singolo elemento, $T(1) = \Theta(1)$

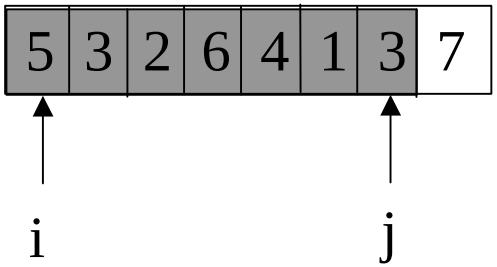
PARTITION

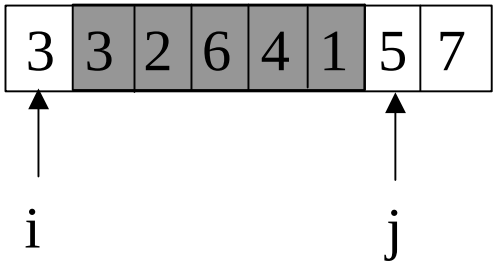
```
PARTITION(A,p,r)
1 x ← A[p]
2 ▷ x è l'elemento perno
3 i ← p – 1
4 j ← r + 1
5 ▷ A[p .. i] e A[j .. r]: regioni bassa e alta di A[p .. r], inizialmente vuote, dove
    vengono inseriti rispettivamente elementi ≤ x e ≥ x
6 while TRUE
7    do  repeat  j ← j – 1
8          until  A[j] ≤ x
9          repeat  i ← i + 1
10         until  A[i] ≥ x
11         if i < j
12           then  scambia A[i] ↔ A[j]
13           else  return j
```

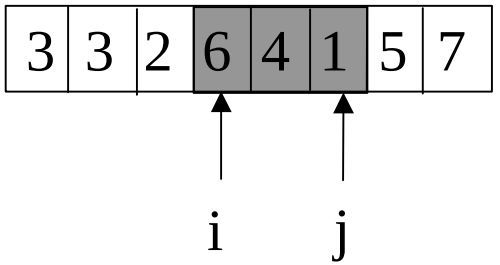
Θ(n)

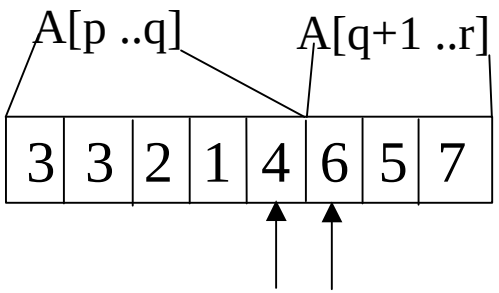
Esempio: PARTITION











j = valore ritornato j i

QUICKSORT: caso pessimo

Il caso pessimo (del tempo di esecuzione) di QUICKSORT si verifica quando PARTITION, a ogni invocazione, produce una regione con n – 1 elementi e una con un solo elemento

$$T^p(n) = \Theta(n) + T(n-1) + \Theta(1) = \sum_{k=1}^{n} \Theta(k) = \Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$

(come per INSERTION-SORT)

QUICKSORT

PARTITION

invocazione di QUICKSORT su n – 1 elementi

invocazione di QUICKSORT su un singolo elemento

Il caso pessimo di QUICKSORT si presenta quando A è già ordinato, una situazione in cui INSERTION-SORT è eseguito in tempo O(n)

QUICKSORT: il caso migliore

Il caso migliore di QUICKSORT si verifica quando PARTITION, a ogni invocazione, produce due regioni con n/2 elementi ciascuna (partizionamento bilanciato)

$T^{best}(n) = \Theta(n) + 2T(n/2) = \Theta(n lg n)$

QUICKSORT: caso medio

Il caso medio di QUICKSORT è vicino a quello migliore

Ad es. si supponga che PARTITION, a ogni invocazione, produca due regioni, una contenente 9n/10 elementi, l'altra n/10 elementi

$T^m(n) = n + T(9n/10) + T(n/10) = \Theta(n\lg n)$

QUICKSORT ha un tempo di esecuzione $\Theta(n\lg n)$ per qualunque suddivisione di proporzionalità costante, per quanto sbilanciata, e così pure quando si alternano partizioni buone e cattive $\rightarrow$ è l'algoritmo ideale per input sufficientemente grandi se vale l'ipotesi che tutte le permutazioni di numeri in input siano ugualmente probabili; per superare questa ipotesi si può ricorrere a versioni randomizzate di QUICKSORT

Algoritmi randomizzati

Sono algoritmi il cui comportamento è determinato non solo dall'input ma anche da valori prodotti da un generatore di numeri casuali RANDOM

RANDOM(a,b) restituisce, con uguale probabilità e in modo indipendente a ogni chiamata, un qualsiasi intero compreso fra a e b (inclusi)

Proprietà di molti algoritmi randomizzati (QUICKSORT incluso): il caso peggiore dipende dal generatore di numeri casuali, non dalla particolare configurazione dell'input

Nel caso di QUICKSORT, l'uscita di RANDOM viene usata per produrre una permutazione casuale dei valori da ordinare → l'algoritmo di ordinamento si comporta male solo se RANDOM "genera" una permutazione sfortunata
Poiché vi sono pochissime permutazioni che causano un comportamento simile a quello del caso peggiore, scegliere una alternativa in modo casuale può portare a un algoritmo efficiente

Versione randomizzata di QUICKSORT

```
RANDOMIZED-PARTITION(A,p,r)
1 i ← RANDOM(p,r)
2 scambia A[p] ↔ A[i]
3 return PARTITION(A,p,r)
```

```
RANDOMIZED-QUICKSORT(A,p,r)
1 if p < r
2    then    q ← RANDOMIZED-PARTITION(A,p,r)
3            RANDOMIZED-QUICKSORT(A,p,q)
4            RANDOMIZED-QUICKSORT (A,q+1,r)
```

Analisi del partizionamento

Assunzione: tutti i numeri in input sono diversi (l'analisi, pur portando agli stessi risultati, è altrimenti più complessa)

rango(x), dove x $\in$ un insieme = n° di elementi dell'insieme $\leq$ x

n = r – p +1 = n° di elementi di A[p..r]
x = A[p]
probabilità(rango(x) = i, $\forall$i = 1, 2 , .. , n) = probabilità(rango(x) = 1) = 1/n

Il valore q restituito da PARTITION dipende solo da rango(x)

Analisi del partizionamento (cont.)

CASO 1

Se rango(x) = 1 (cioè A[p] è l'elemento minimo), la prima esecuzione del ciclo **while** di PARTITION si ferma a

i = p
j = p

restituendo q = j (cioè il lato basso delle partizione ha dimensione 1 perché contiene solo A[p])

probabilità (CASO 1) = probabilità(rango(x) = 1) = 1/n

Analisi del partizionamento (cont.)

CASO 2

Se rango(x) ≥ 2, alla prima iterazione del ciclo **while** di PARTITION si calcola

i = p
j > p

e si effettua lo scambio A[p] ↔ A[j] (per mettere A[p] nel lato alto della partizione).
Al termine, restituisce una partizione il cui lato basso contiene
i = rango(x) – 1 (→ i = 1, 2, .. , n – 1)
elementi strettamente < x

probabilità (CASO 2) = probabilità(rango(x) = k, per ciascun k = 2, .., n) = 1/n

Analisi del partizionamento (cont.)

Combinando i due casi

N° elementi del lato basso della partizione	Probabilità
1	probabilità (CASO 1) + probabilità (CASO 2) = 2/n
i (per ciascun i = 2, 3, .. , n – 1)	probabilità (CASO 2) = 1/n

Ordinamento per confronti

Sono algoritmi basati sul confronto fra gli elementi di input

	T(n)	Metodo
INSERTION-SORT	$O(n^2)$ (limite raggiunto nel caso pessimo, cioè quando l'array è ordinato in ordine opposto a quello desiderato)	"in loco"
MERGE-SORT	O(n*lg*n) (non esiste il caso pessimo)	non "in loco"
HEAPSORT	O(n*lg*n) (limite raggiunto nel caso pessimo di HEAPIFY)	"in loco"
QUICKSORT	O(n*lg*n) (limite raggiunto in media) $O(n^2)$ (limite raggiunto nel caso pessimo, cioè quando l'array è già ordinato)	"in loco"