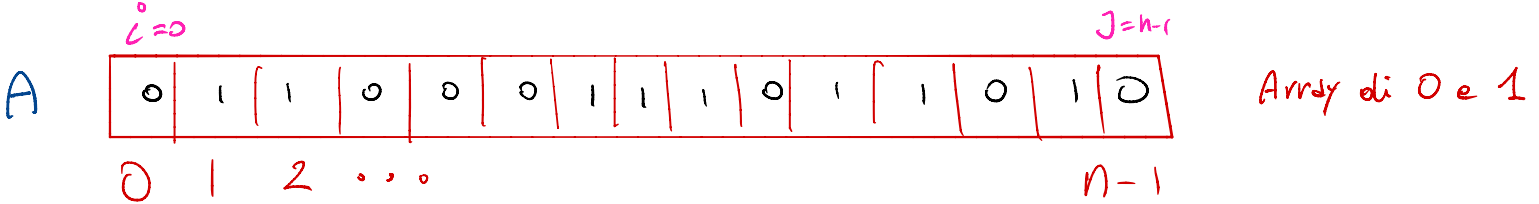
Quicksort & Randomization

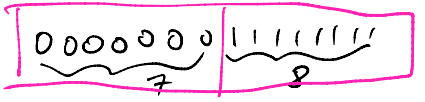


Array di 0 e 1

Scopo: ordinare A mediante scambi del contenuto di due posizioni

scambia(i,j): tmp = A[i]; A[i] = A[j]; A[j] = tmp

oss Senza scambia( ): conta il #0 o #1 e poi sovrascrivi A

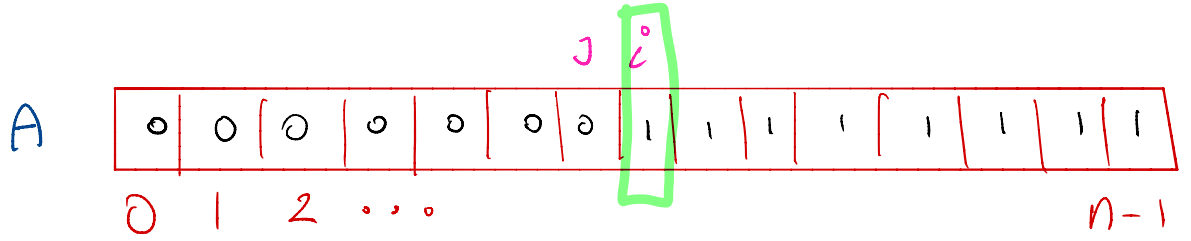
es. 7 zeri e 8 uni → 

OSS usando invece scambia(), e il motivo sarà chiaro tra poco, uso due puntatori i e j

invariante: A[i] = 0 ok {altrimenti} i++
A[j] = 1 ok {altrimenti} j--

Esempio:

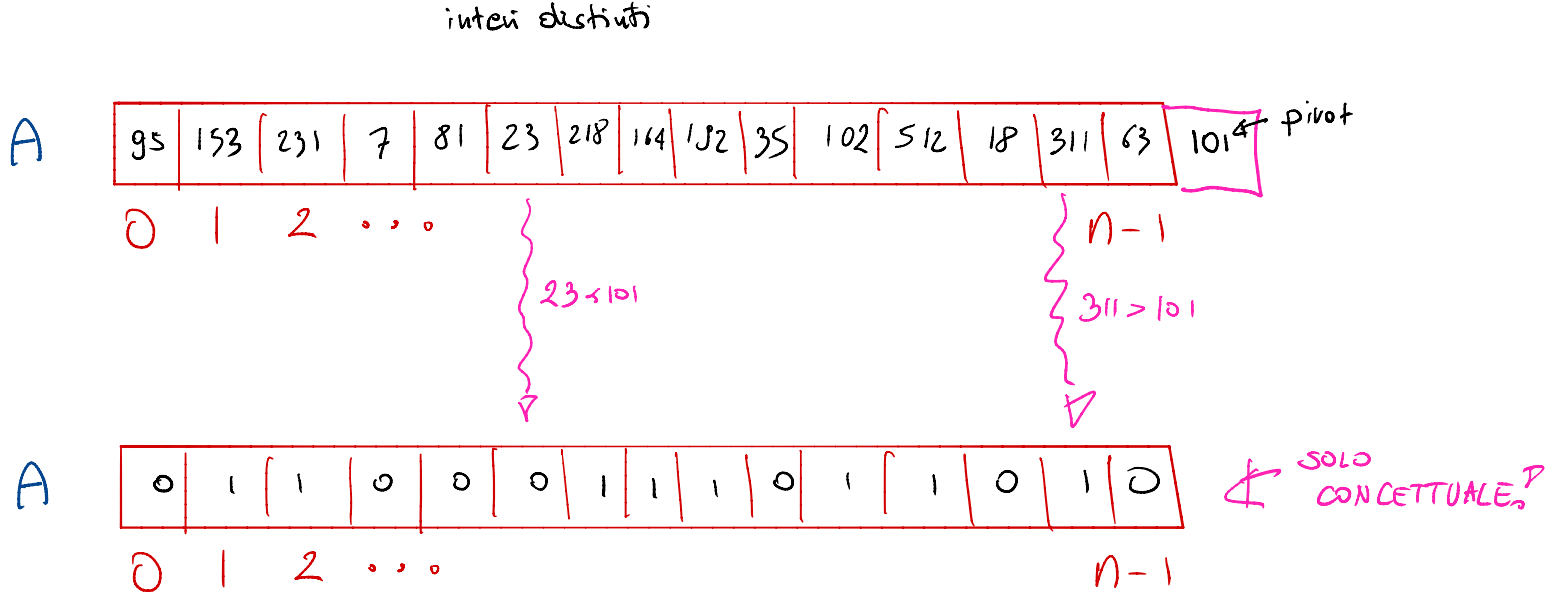
INV: zeri a SX e uni a DX



usiamo scambia(i,j) per ripristinare INV

ci fermiamo quando $j \leq i$

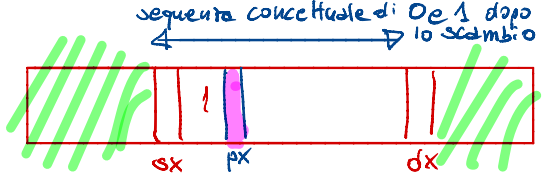
Perché scambiare? Adesso usiamo un array di interi e operiamo una DISTRIBUZIONE o PARTIZIONE



elemento A[i] < pivot → è come se A[i] fosse zero

" " > " → " " " " " uno

IL RISULTATO DELL'APPROCCIO PRECEDENTE È una PARTIZIONE / DISTRIBUZIONE
metto prima tutti gli elementi A[i] < pivot e poi quelli > pivot



ci concentriamo su questo segmento

```
1   Distribuzione( a, sx, px, dx ):          ⟨pre: 0 ≤ sx ≤ px ≤ dx ≤ n − 1⟩
2     IF (px != dx) Scambia( px, dx );
3     i = sx;
4     j = dx-1;
5     WHILE (i <= j) {
6       WHILE ((i <= j) && (A[i] <= A[dx]))
7         i = i+1;
8       WHILE ((i <= j) && (A[j] >= A[dx]))
9         j = j-1;
10      IF (i < j) Scambia( i, j );
11    }
12    IF (i != dx) Scambia( i, dx );
13    RETURN i;
```

perché adesso in dx c'è il pivot

è "zero"

è "uno"

Complessità in tempo:

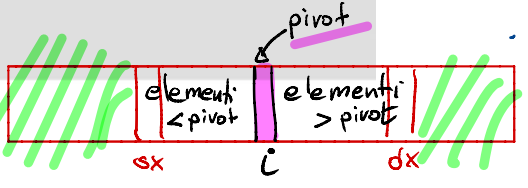
$O(n)$ tempo

$O(1)$ spazio aggiuntivo

$n = dx - sx + 1$ = numero di elementi

ogni volta che passo per (7) e (9) incremento i e/o decremento j
⇒ al massimo succede n volte

OUTPUT:



Questo suppersice uno schema di tipo DIVIDE et IMPERA:

```
1  QuickSort( a, sinistra, destra ):
2       ⟨pre: 0 ≤ sinistra, destra ≤ n - 1⟩
3    IF (sinistra < destra) { ← almeno due elementi in A[sinistra .. destra]
4      scegli pivot nell'intervallo [sinistra...destra];
5      rango = Distribuzione( a, sinistra, pivot, destra );
6      QuickSort( a, sinistra, rango-1 );
7      QuickSort( a, rango+1, destra );
8    }
```

Lo schema di ricorsione è simile al Mergesort, tuttavia la fase di ricombinazione dei risultati delle chiamate ricorsive è qui l'IDENTITÀ (era la fusione nel mergesort).

Viceversa, la fase di divisione in sottoproblemi (era semplice nel mergesort) è qui più complicata (PARTIZIONE/DISTRIBUZIONE)

Relazione di ricorrenza per il costo $T(n)$ del QuickSort su n elementi

$T(n)$

```
1  QuickSort( a, sinistra, destra ):
2       ⟨pre: 0 ≤ sinistra, destra ≤ n - 1⟩
3    IF (sinistra < destra) {
4      scegli pivot nell'intervallo [sinistra...destra];
5      rango = Distribuzione( a, sinistra, pivot, destra );
6      QuickSort( a, sinistra, rango-1 );
7      QuickSort( a, rango+1, destra );
8    }
```

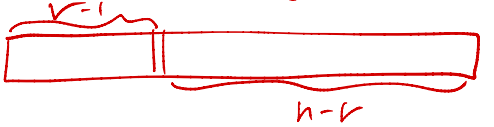
O(1) tempo → riga 3
O(n) tempo → riga 5
$T(r-1)$ tempo → riga 6
$T(n-r)$ tempo → riga 7

r = rango

$$T(n) \leq T(r-1) + T(n-r) + cn \quad , \quad T(n_0) = \text{costante} \qquad n \leq n_0$$

(idealmente: $r \sim \frac{n}{2}$, in modo da avere $T(n) \sim 2T(\frac{n}{2}) + cn$ come nel mergesort)

NON C'È GARANZIA DI CIÒ
CASO PESSIMO: $r=0$ o $r=n$)



Caso pessimo: dopo aver speso $\Theta(n)$ tempo, ho ancora $n-1$ elementi da ordinare
perché $r=0$

Non dipende da come si sceglie il pivot DETERMINISTICAMENTE

se ogni volta il pivot è il minimo della partizione, mi rimangono ancora gli altri da ordinare tutti insieme: la fase di divisione non porta vantaggi.

costo	# elementi ancora nella stessa partizione
cn	$n-1$
$c(n-1)$	$n-2$
$c(n-2)$	$n-3$
$\vdots$	$\vdots$
c	1

$\Theta(n^2)$

MORALE: QS richiede $O(n^2)$ tempo al caso pessimo

In pratica, QS è veloce perché il suo costo medio è $O(n \lg n)$

↳ concettualmente, prendiamo gli infiniti array di n numeri, eseguiamo il QS per ciascuno di essi, e ne facciamo la media.

Purtroppo il caso "sfavorevole" rimane tale nel suddetto costo medio: l'assunzione implicita è che l'array sia scelto casualmente tra i possibili

---

RANDOMIZATION

IDEA PIÙ POTENTE: usiamo la scelta di un numero casuale per sfuggire ("probabilmente") al caso sfavorevole: la "media" qui non viene più fatta su tutti i possibili input ma su tutte le possibili scelte del numero casuale.

```
1  QuickSort( a, sinistra, destra ):
2                              ⟨pre: 0 ≤ sinistra, destra ≤ n − 1⟩
3    IF (sinistra < destra) {
4      pivot = sinistra + (destra − sinistra) × random();
5      rango = Distribuzione( a, sinistra, pivot, destra );
6      QuickSort( a, sinistra, rango-1 );
7      QuickSort( a, rango+1, destra );
8    }
```

← sceglie pivot ∈ [sinistra .. destra] in maniera uniforme e casuale

Il costo medio è su tutte le scelte suddette in tutte le chiamate ricorsive

Vediamo perché il costo medio è $O(n \lg n)$ indipendentemente dall'input (con alta probabilità)

Variabile indicatrice: $X = \begin{cases} 1 & \text{probabilità } p \\ 0 & \text{altrimenti} \end{cases}$

$E[X] = 1 \cdot p + 0 \cdot (1-p) = p$ (non richiede che le variabili siano indipendenti)

Per analizzare QS randomizzato, prendiamo A' = array A ordinato

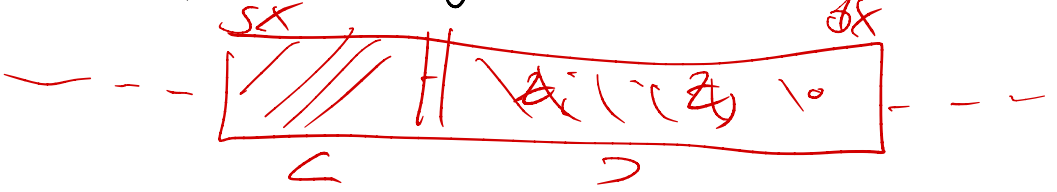
$A' = z_1 < z_2 < \cdots < z_n$ (oss. A contiene tali chiave in modo disordinato)

osservazioni

① z_i e z_j sono confrontate, una di loro è un pivot

② z_i e z_j sono confrontate al massimo una sola volta

③ se z_i e z_j ($i<j$) sono nella stessa porzione da ordinare, allora tale porzione contiene almeno $j-i+1$ elementi, ossia tutte gli elementi da z_i a z_j: $z_i, z_{i+1}, z_{i+2}, \ldots, z_j$



$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ e } z_j \text{ sono confrontati} \\ 0 & \text{altrimenti} \end{cases}$$

$i < j$

① + ② $\Rightarrow X = \sum_{i<j} X_{ij}$ = numero di coppie di chiavi confrontate

① z_i o z_j sono pivot

$$Pr[X_{ij}] \leq \frac{2}{j-i+1}$$

③ pr chiave è pivot $\frac{1}{\text{portata}} \leq \frac{1}{j-i+1}$

costo = $O(n+X)$ tempo ma X è una variabile aleatoria

costo medio = $O(n + E[X])$ tempo

serie armonica = $O(\ln n)$

$$E[X] = \sum_{i<j} E[X_{ij}] \leq \sum_{i<j} \frac{2}{j-i+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = O(n \lg n).$$