

ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

A.A. 2022/23

ALGORITMI DI ORDINAMENTO:
CountingSort

"E' vietata la copia e la riproduzione dei contenuti e
immagini in qualsiasi forma.

E' inoltre vietata la redistribuzione e la pubblicazione dei
contenuti e immagini non autorizzata espressamente
dall'autore o dall'Università di Modena e Reggio Emilia."



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

PROBLEMA dell'ORDINAMENTO

INPUT:

Una sequenza di n **NATURALI** $A = \langle a_0, a_1, \dots, a_{n-1} \rangle$ tale che $0 \leq a_i \leq k, \forall i \in \{0, 1, \dots, n-1\}$.

OUTPUT:

La sequenza A ordinata in ordine non decrescente

NON verranno effettuati confronti tra elementi di A

CountingSort

IDEA:

- Per ogni possibile valore j assunto dagli elementi di A (sono solo quelli che vanno da 0 a k), conto quanti elementi in A sono uguali a j , usando un array di appoggio C
- Genero la sequenza A appropriata, partendo da C

ESEMPIO $A = \langle 5, 8, 3, 0, 8, 10, 7, 7, 3, 2 \rangle$ $k = 10$

CountingSort

- Per ogni possibile valore j assunto dagli elementi di A (sono solo quelli che vanno da 0 a k), conto quanti elementi in A sono uguali a j , usando un array di appoggio C

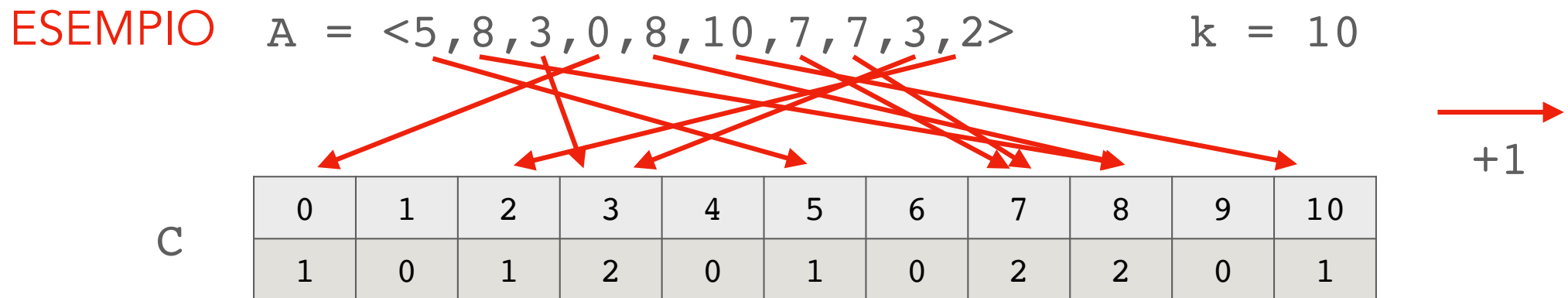
ESEMPIO $A = \langle 5, 8, 3, 0, 8, 10, 7, 7, 3, 2 \rangle$ $k = 10$

C

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0

CountingSort

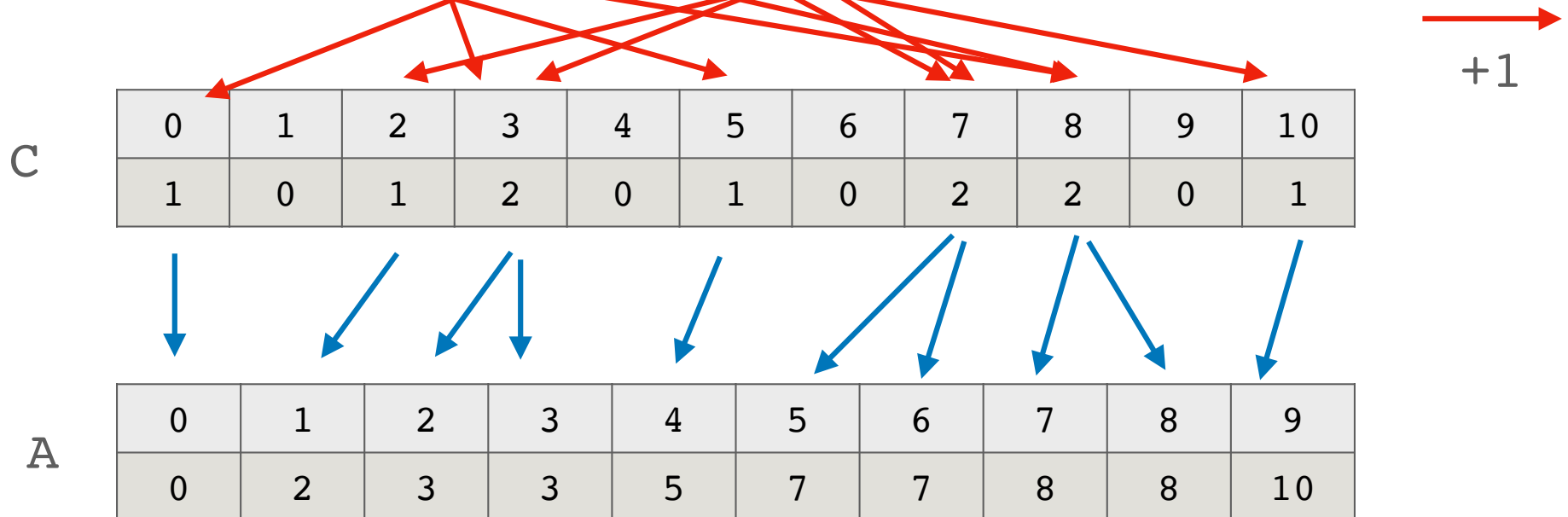
- Per ogni possibile valore j assunto dagli elementi di A (sono solo quelli che vanno da 0 a k), conto quanti elementi in A sono uguali a j , usando un array di appoggio C



CountingSort

- Genero la sequenza A appropriata, partendo da C

ESEMPIO $A = \langle 5, 8, 3, 0, 8, 10, 7, 7, 3, 2 \rangle$ $k = 10$



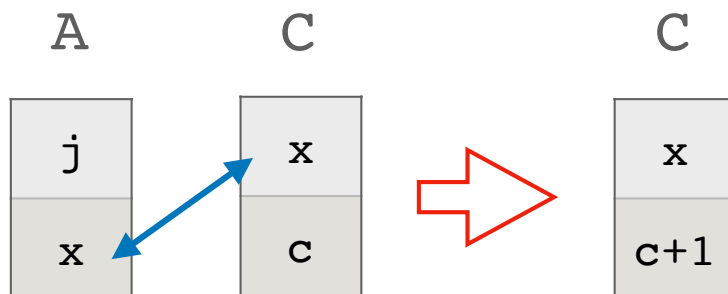
CountingSort

- Usiamo un array C di appoggio, che deve avere tante celle quanti i possibili valori assunti dai valori in $A \rightarrow k+1$
- $C[i] =$ numero di occorrenze in A del valore i

$A = \langle 5, 8, 3, 0, 8, 10, 7, 7, 3, 2 \rangle$ $k = 10$

0	1	2	3	4	5	6	7	8	9	10
1	0	1	2	0	1	0	2	2	0	1

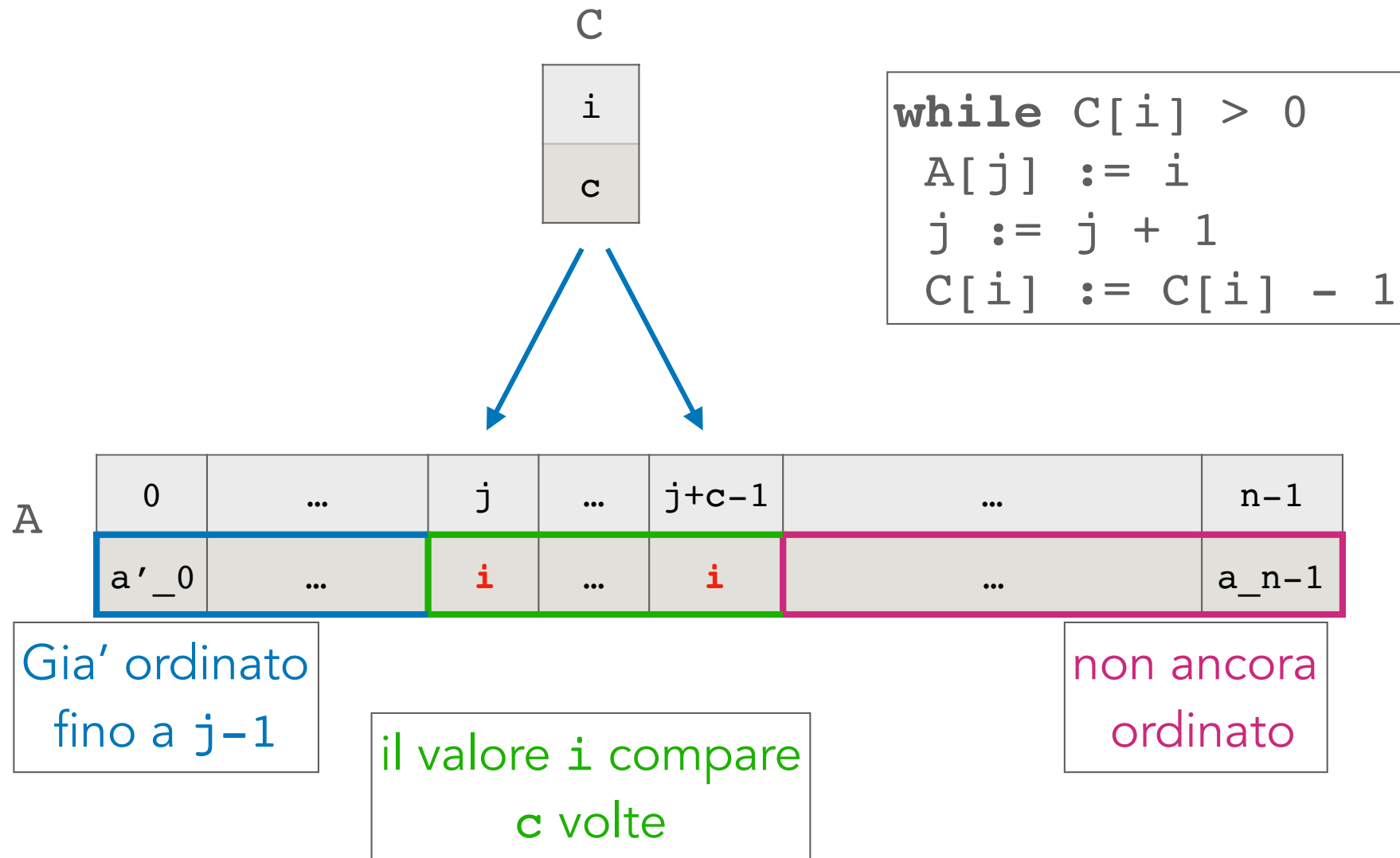
- Per riempire C , scorriamo A e aggiorniamo C opportunamente



```
for  $j = 0$  to  $n-1$   
     $C[A[j]] := C[A[j]] + 1$ 
```

CountingSort

- Generiamo la sequenza appropriata in A, scorrendo C



CountingSort

```
COUNTINGSORT(A,k)
  n := length(A) // cardinalità di A
  crea C[0..k]
  for i = 0 to k // inizializzazione C
    C[i] := 0
  for j = 0 to n-1
    C[A[j]] := C[A[j]] + 1
  //calcolo occorrenze: C[i] = numero di
  // occorrenze di i
  j = 0
  for i = 0 to k // "riempimento" di A
    while C[i] > 0
      A[j] := i
      j := j + 1
      C[i] := C[i] - 1
```

Torniamo a contare
le operazioni logico-
aritmetiche

$\Theta(k)$

$\Theta(n)$

$\Theta(n + k)$

Costo computazionale
tempo $\Theta(n + k)$ spazio $\Theta(k)$

Test eseguito per ogni
indice di C

Istruzioni eseguite per
ogni indice di A



CountingSort

COSTO COMPUTAZIONALE

Tempo $T(n) \in \Theta(k) + \Theta(n) + \Theta(n + k) = \Theta(n + k)$ e $T(n) \in \Omega(k)$

Spazio $\Theta(k)$

Buone notizie:

- Se $k \in O(n \log n) \Rightarrow T(n) \in O(n \log n)$ 
- Se $k \in O(n) \Rightarrow T(n) \in O(n)$ 

Ma come la mettiamo con il lower bound??

Brutte notizie:

- Se $k \in \Omega(n^t) \Rightarrow T(n) \in \Omega(n^t)$ e spazio $\Theta(n^t)$
- Se $k \in \Omega(2^n) \Rightarrow T(n) \in \Omega(2^n)$ e spazio $\Theta(2^n)$



➡ CountingSort si usa quando $k \in O(n)$, meglio se $k \in \Theta(1)$