

ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangelo

A.A. 2022/23

ALGORITMI DI ORDINAMENTO:
CountingSort stabile

"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

CountingSort

VERSIONE STABILE

La versione di CountingSort che abbiamo visto **NON** è **STABILE**

2 2 2 A

2 2 2 stabile

2 2 2 non
stabile

Valori uguali si trovano nella sequenza
ordinata nello stesso ordine in cui erano
nella sequenza di partenza

CountingSort

VERSIONE STABILE

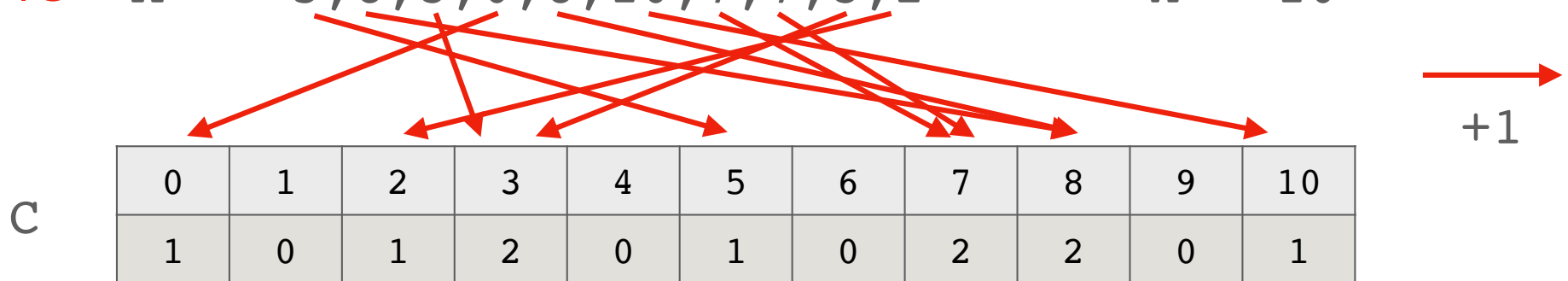
- Abbiamo bisogno di un array di appoggio B , di n elementi, per non sovrascrivere gli elementi di A mentre ordiniamo
- Dobbiamo usare C in modo più intelligente

CountingSort

VERSIONE
STABILE

1. 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 **COME PRIMA**

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



$C[i]$ = numero di occorrenze di i in A

CountingSort

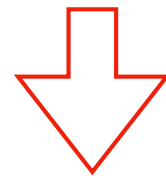
VERSIONE
STABILE

2. Modifichiamo C di modo che $C[i] = \text{numero di valori in } A \leq i$

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

$C[i] = \text{numero di occorrenze di } i \text{ in } A$

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



Come?

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

$C[i] = \text{numero di valori in } A \leq i$

CountingSort

VERSIONE
STABILE

2. Modifichiamo C di modo che $C[i] = \text{numero di valori in } A \leq i$

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

$C[i] = \text{numero di occorrenze di } i \text{ in } A$

C

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



C

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

$C[i] = \text{numero di valori in } A \leq i$ | $C[i] = \text{numero di occorrenze di } i \text{ in } A$

CountingSort

VERSIONE
STABILE

2. Modifichiamo C di modo che $C[i] = \text{numero di valori in } A \leq i$

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

$C[i] = \text{numero di occorrenze di } i \text{ in } A$

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

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

$C[i] = \text{numero di valori in } A \leq i$ | $C[i] = \text{numero di occorrenze di } i \text{ in } A$

CountingSort

VERSIONE
STABILE

2. Modifichiamo C di modo che $C[i] = \text{numero di valori in } A \leq i$

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

$C[i] = \text{numero di occorrenze di } i \text{ in } A$

C

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

C

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

$C[i] = \text{numero di valori in } A \leq i$ | $C[i] = \text{numero di occorrenze di } i \text{ in } A$

CountingSort

VERSIONE
STABILE

2. Modifichiamo C di modo che $C[i] = \text{numero di valori in } A \leq i$

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

$C[i] = \text{numero di occorrenze di } i \text{ in } A$

C

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

C

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

$C[i] = \text{numero di valori in } A \leq i$ | $C[i] = \text{numero di occorrenze di } i \text{ in } A$

CountingSort

VERSIONE
STABILE

2. Modifichiamo C di modo che $C[i] = \text{numero di valori in } A \leq i$

```
for i = 1 to k  
  C[i] := C[i] + C[i-1]  
// C[i] = numero di valori  $\leq i$  in A
```

C

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

CountingSort

VERSIONE
STABILE

3. Copiamo in B i valori di A in modo ordinato, partendo dalla fine fine di A, usando C appropriatamente

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

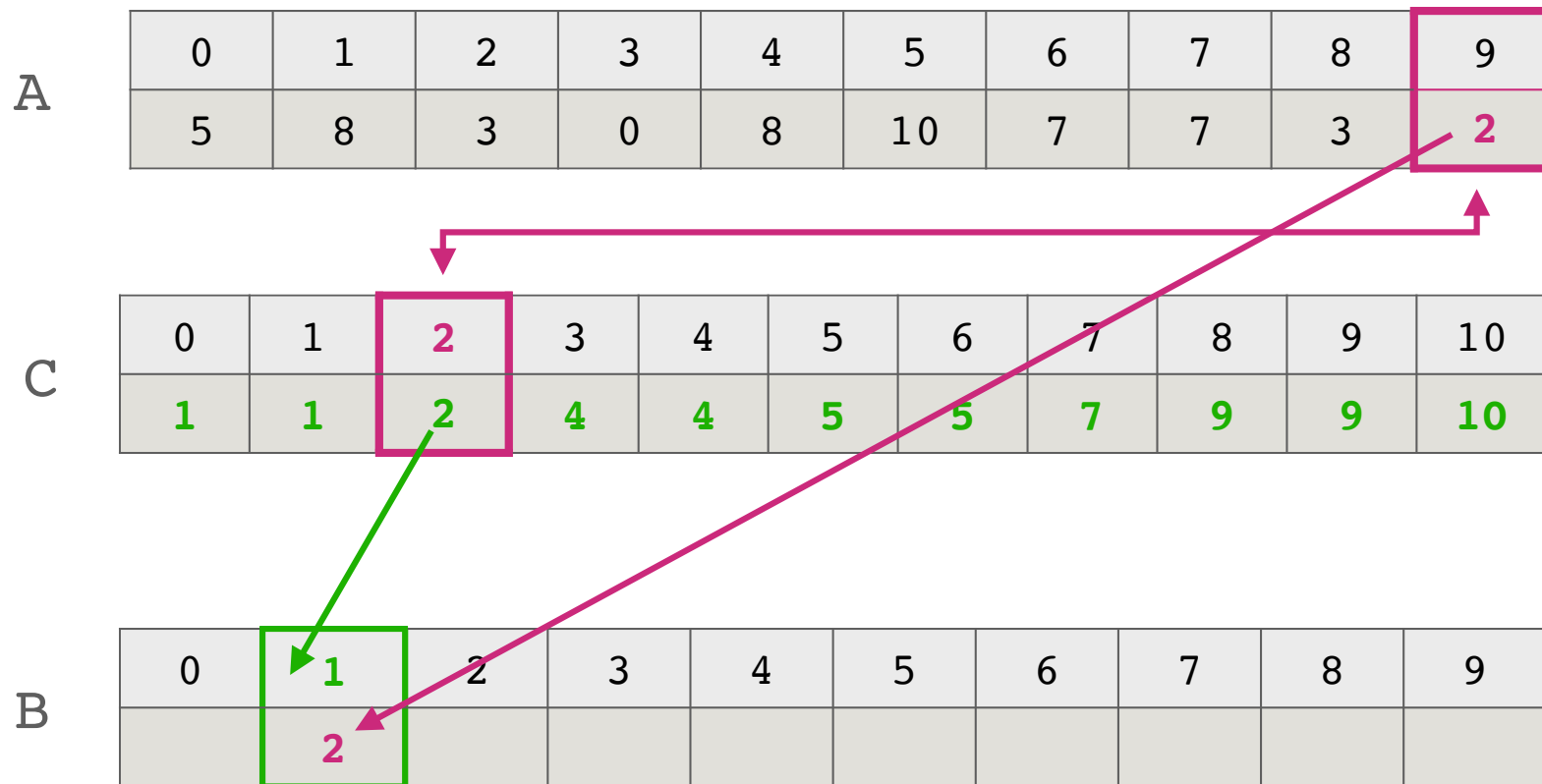
C	0	1	2	3	4	5	6	7	8	9	10
	1	1	2	4	4	5	5	7	9	9	10

B	0	1	2	3	4	5	6	7	8	9

CountingSort

VERSIONE
STABILE

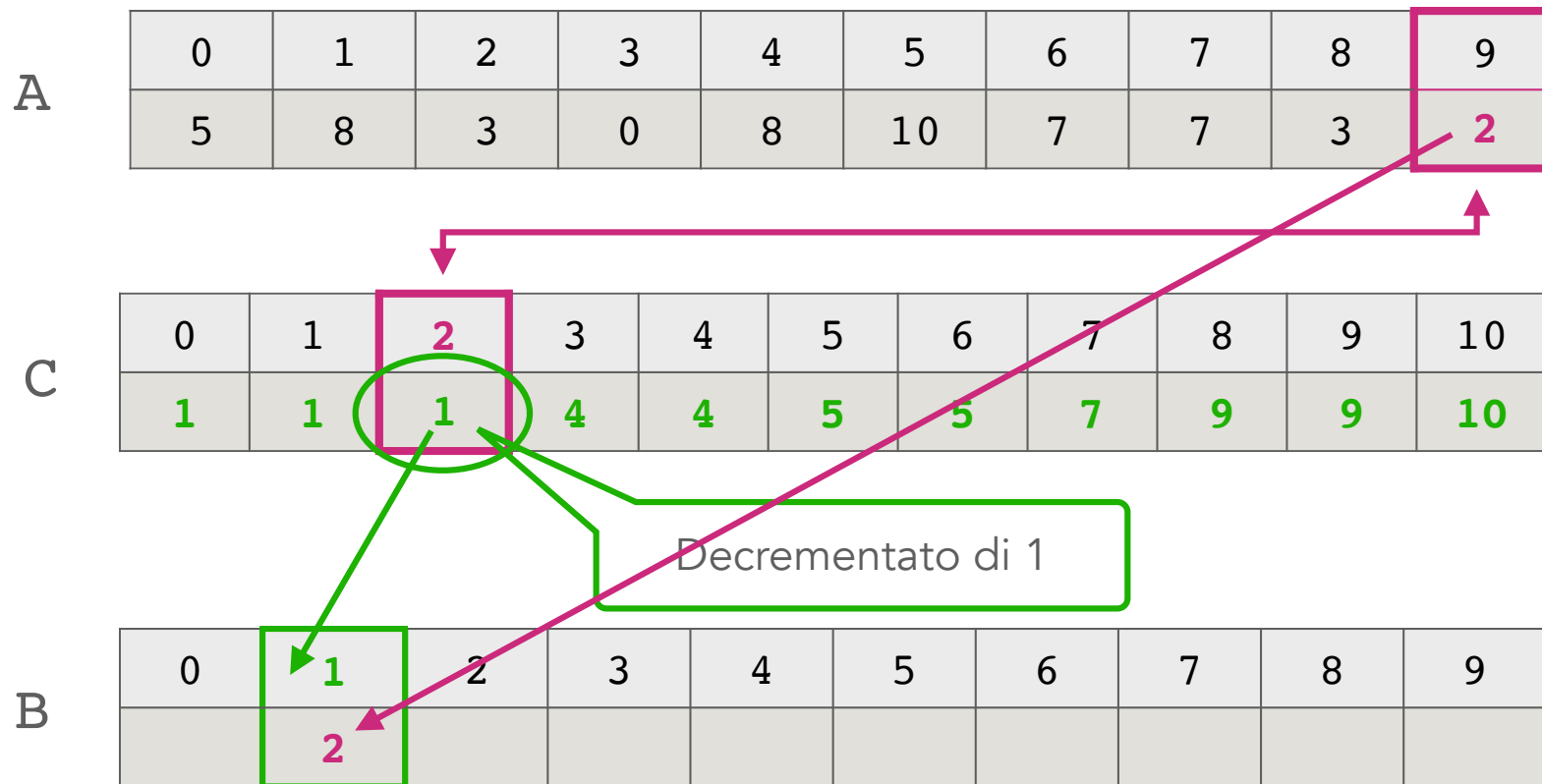
3. Copiamo in B i valori di A in modo ordinato, partendo dalla fine fine di A, usando C appropriatamente



CountingSort

VERSIONE
STABILE

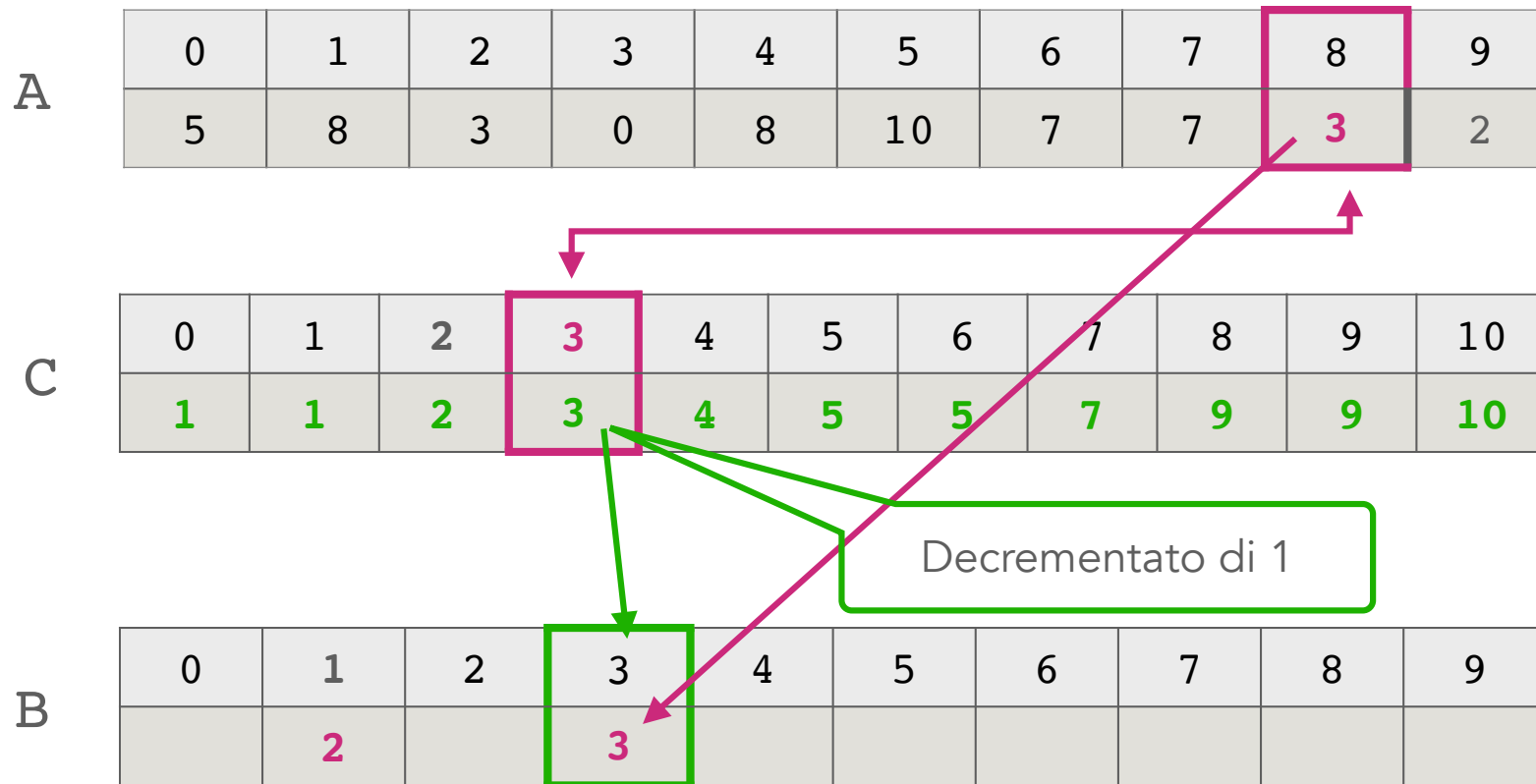
3. Copiamo in B i valori di A in modo ordinato, partendo dalla fine fine di A, usando C appropriatamente



CountingSort

VERSIONE
STABILE

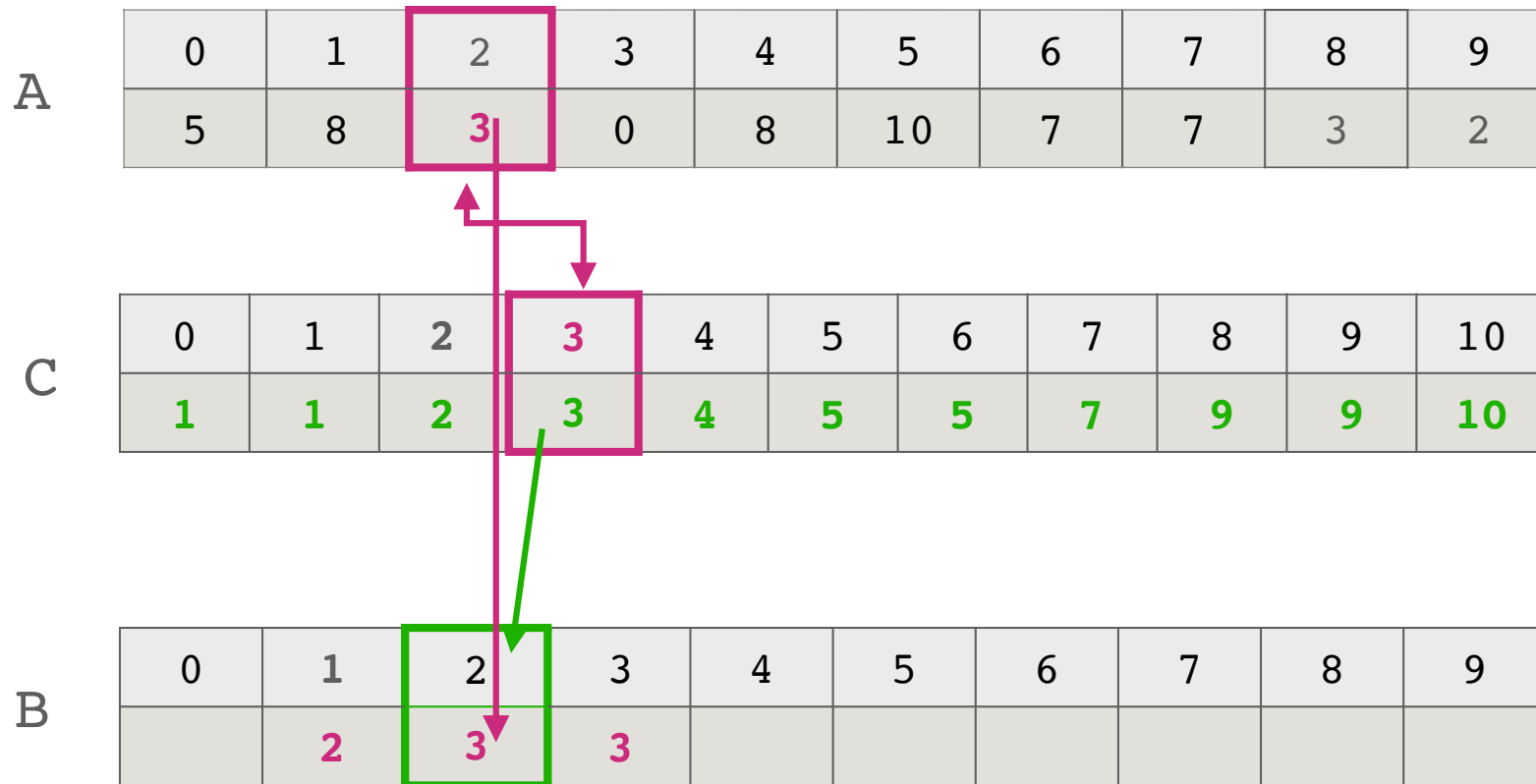
3. Copiamo in B i valori di A in modo ordinato, partendo dalla fine fine di A, usando C appropriatamente



CountingSort

VERSIONE
STABILE

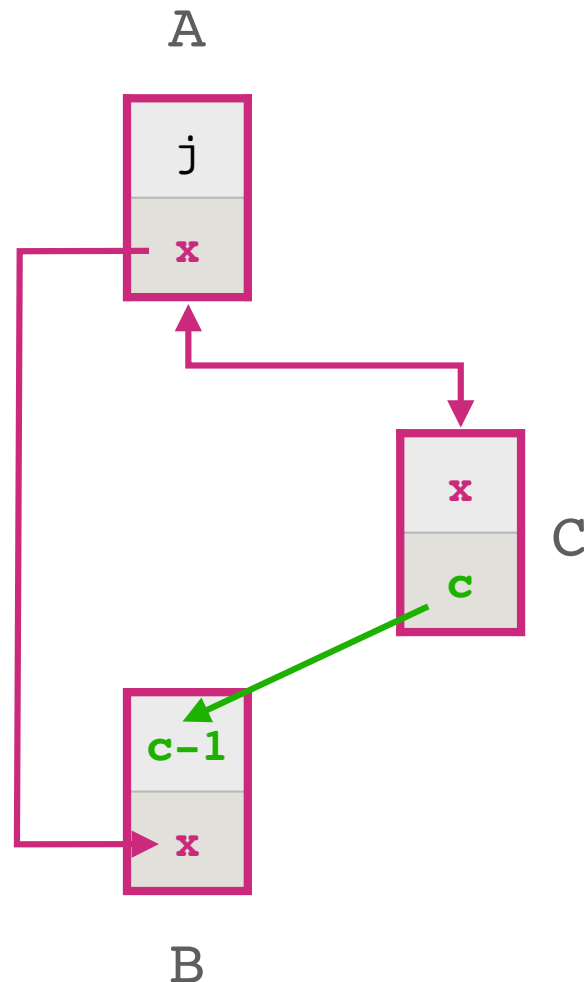
3. Copiamo in B i valori di A in modo ordinato, partendo dalla fine fine di A, usando C appropriatamente



CountingSort

VERSIONE
STABILE

3. Copiamo in B i valori di A in modo ordinato, partendo dalla fine fine di A, usando C appropriatamente



$A[j] = x$ = elemento da copiare in B

$C[x]$ = numero di elementi in A con
valore $\leq x$

(meno quelli = già copiati)

$B[C[x]-1]$ = posizione in cui copiare x

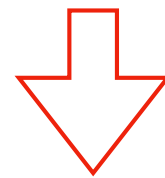
```
for j = n-1 downto 0 // "riempimento" di B
  B[C[A[j]]-1] := A[j]
  C[A[j]] := C[A[j]] - 1
```


CountingSort

VERSIONE
STABILE

4. Copiamo B in A

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



Copiare

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

```
for j = 0 to n-1 // copia B in A  
  A[j] := B[j]
```

CountingSort

VERSIONE
STABILE

```
COUNTINGSORT(A,k)
  n := length(A) // cardinalità di A
  crea C[0..k]
  crea B[0..n-1]
  for i = 0 to k // inizializzazione C →  $\Theta(k)$ 
    C[i] := 0
  for j = 0 to n-1
    C[A[j]] := C[A[j]] + 1
  // calcolo occorrenze: C[i] = numero di →  $\Theta(n)$ 
  // occorrenze di i
  for i = 1 to k →  $\Theta(k)$ 
    C[i] := C[i] + C[i-1]
  // C[i] = numero di valori ≤ i in A
  for j = n-1 downto 0 // "riempimento" di B →  $\Theta(n)$ 
    B[C[A[j]]-1] := A[j]
    C[A[j]] := C[A[j]] - 1
  for j = 0 to n-1 // copia B in A →  $\Theta(n)$ 
    A[j] := B[j]
```

Costo computazionale: TEMPO $\Theta(n + k)$ SPAZIO $\Theta(n + k)$