Esercitazione 4

Counting sort

Counting sort

- Algoritmo di ordinamento per valori numerici interi con ***complessità lineare***
- ***Idea di base*** dell'algoritmo
  - Determinare, per ogni elemento x, il numero di elementi minori di x presenti nel vettore di input
  - Tale informazione permette di inserire x direttamente nella sua **posizione finale** nel vettore di output

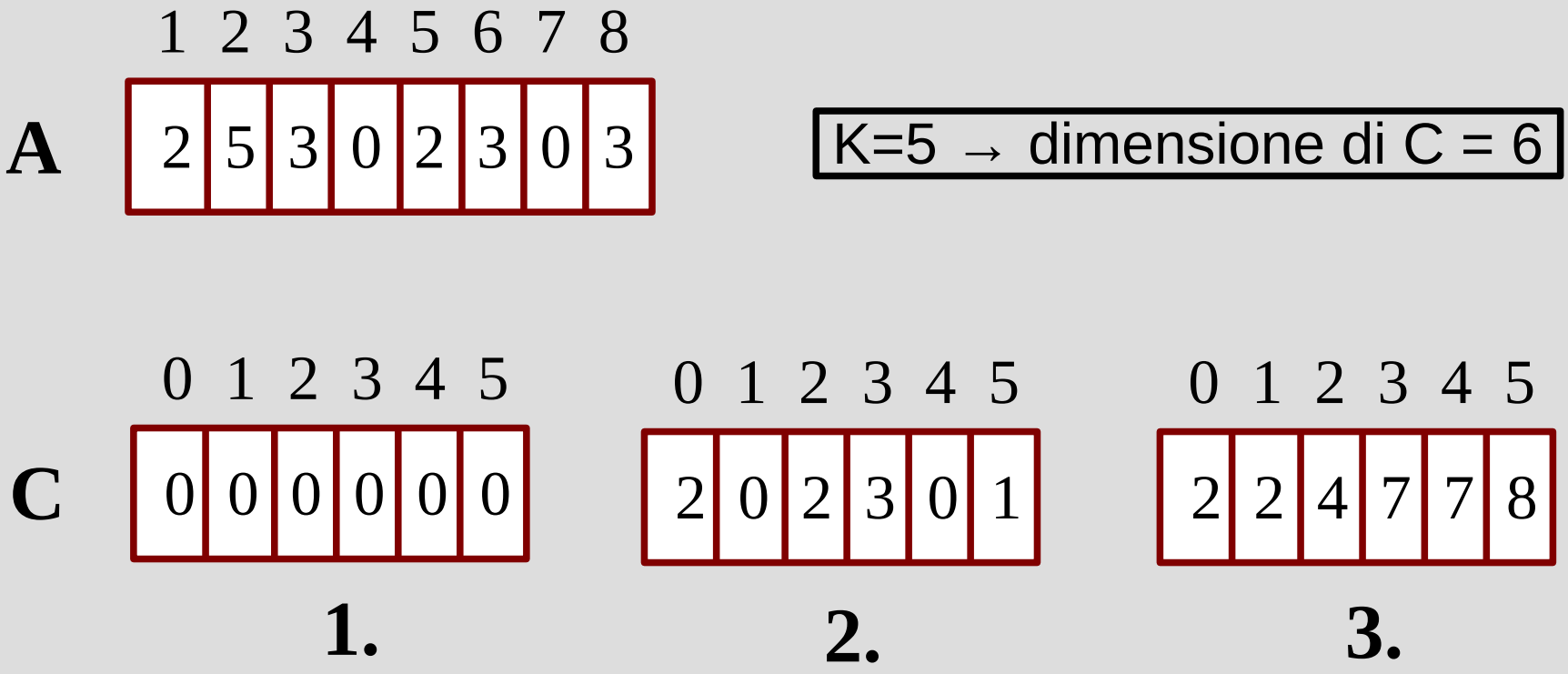
Strutture dati

- **Ipotesi**: l'algoritmo si basa sulla ***conoscenza a priori dell'intervallo*** [0, k] in cui sono compresi i valori da ordinare
- Per ordinare un vettore di **n** elementi counting sort utilizza **3 vettori**:
  - **Vettore di input A (dimensione n)**
  - **Vettore di output B (dimensione n)**
  - **Vettore ausilario C (dimensione k+1)**

Gestione vettore ausiliario C

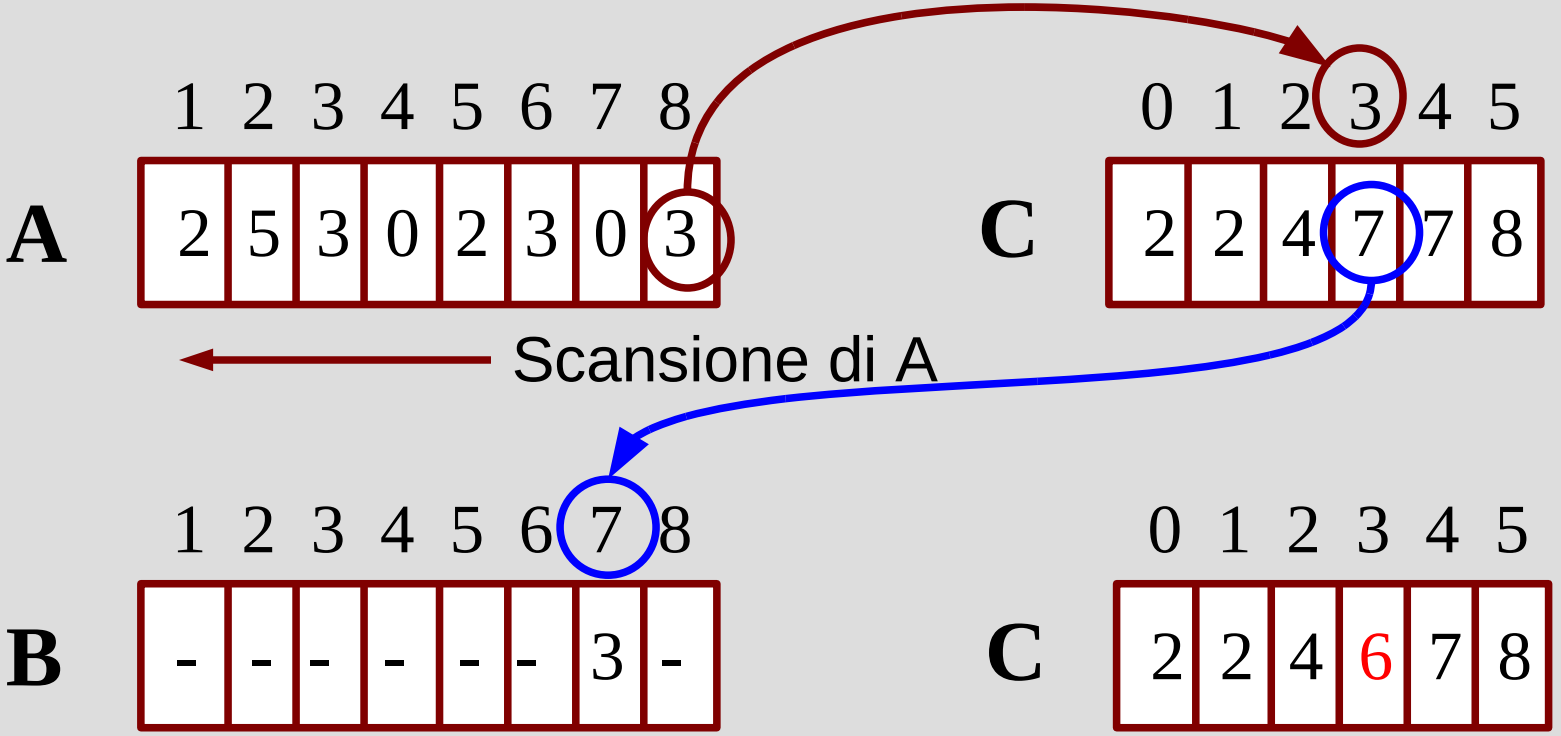
1. **Inizializzazione** del vettore ausiliario C:
   - C[i] = 0 per ogni i=0,..,k
2. **Conteggio** di quanti elementi di A hanno un determinato valore i:
   - C[i] è il numero di elementi di A pari ad i, per ogni i=0,..,k
3. **Somma** dei primi i elementi di C:
   - C[i] è il numero di elementi di A di valore <= i, per ogni i=0,..,k

Ordinamento (1)

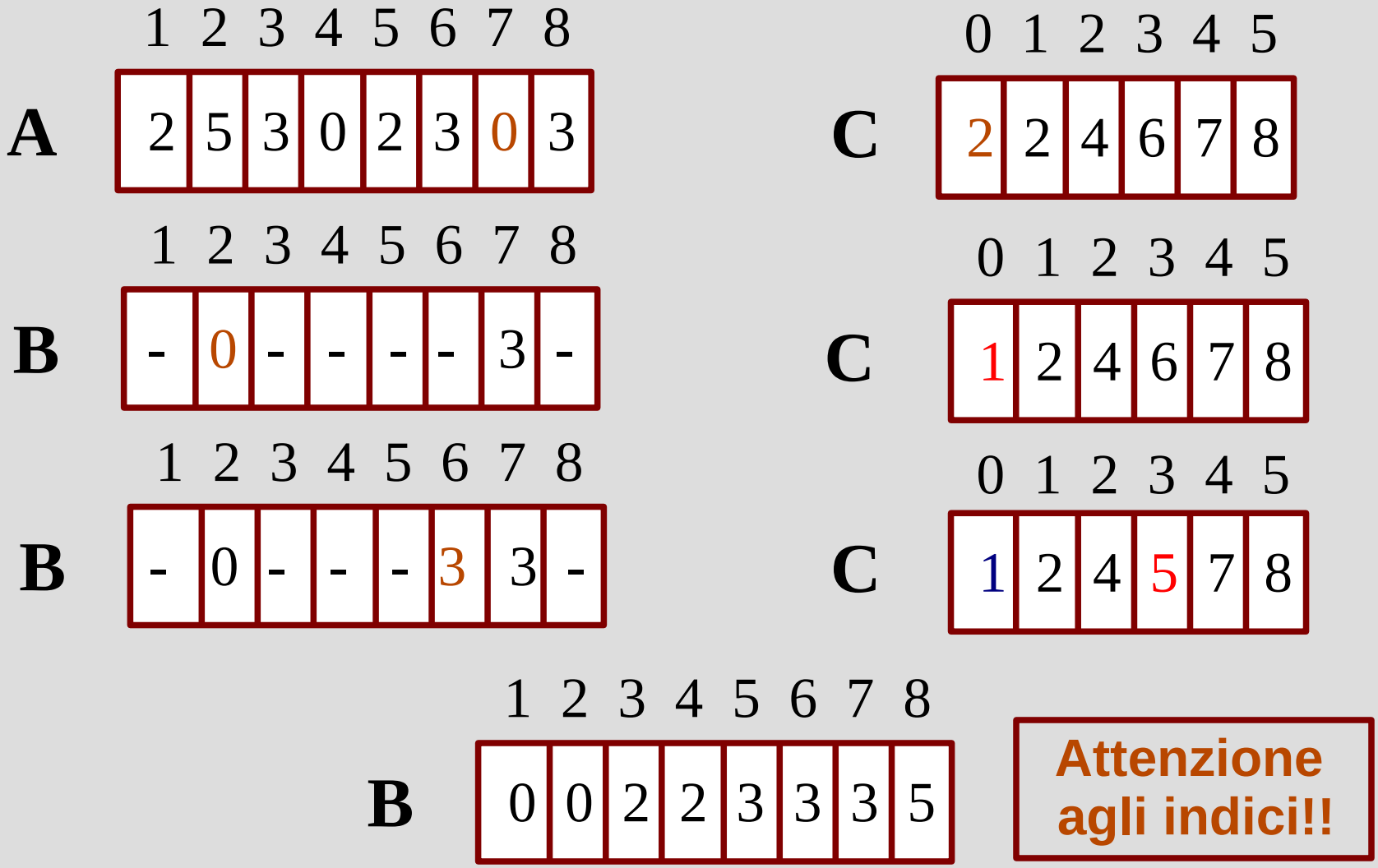


Ordinamento (2)

Riempimento vettore di output **B**: inserisco ogni elemento di **A** nella corretta posizione finale nel vettore **B**



Ordinamento (3)



Programma

- *cs.cc*
- Programma di ordinamento che utilizza l'algoritmo **counting sort** per ordinare un vettore di **n** elementi in senso non decrescente
- Si definisca a tempo di esecuzione del programma il numero **n** di elementi da ordinare e il valore **k** che definisce l'intervallo in cui sono compresi tali elementi
  - Nel vettore di input si inseriscano valori casuali compresi tra **0** e **k**

Ottimizzazione

- Nel vettore di input sono inseriti valori casuali compresi tra **0** e **k**
- *Cosa accade se il valore minimo presente è > 0 e il valore massimo è < k?*

Il funzionamento è assicurato, ma il vettore C è sovradimensionato rispetto al necessario...

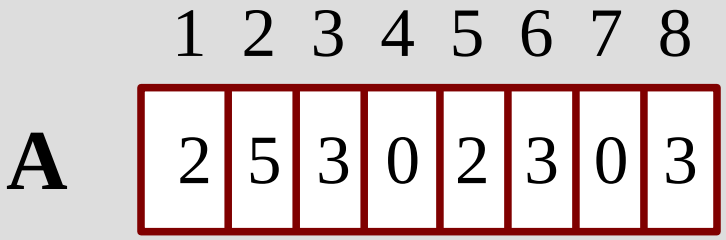
Versione semplificata

- Proviamo ora una **versione semplificata** dell'algoritmo **counting sort**, che utilizza soltanto 2 vettori:
  - **Vettore di input A (dimensione n)**
  - **Vettore ausiliario C (dimensione k+1)**
- **La gestione del vettore ausiliario C si ferma dopo i primi due passi**

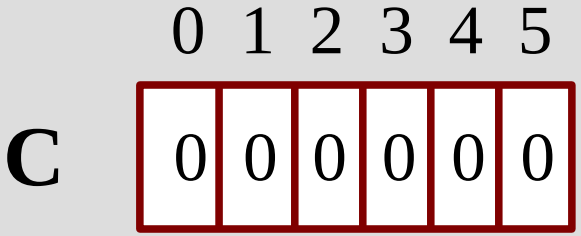
Gestione vettore ausiliario C

1. **Inizializzazione** del vettore ausiliario C:
   - C[i] = 0 per ogni i=0,..,k
2. **Conteggio** di quanti elementi di A hanno un determinato valore i:
   - C[i] è il numero di elementi di A pari ad i, per ogni i=0,..,k
3. **Somma** dei primi i elementi di C:
   - C[i] è il numero di elementi di A di valore <= i, per ogni i=0,..,k

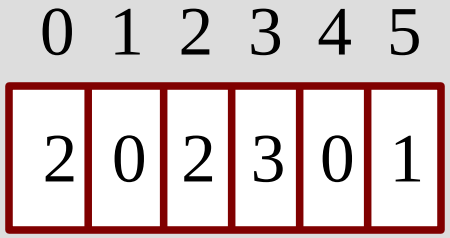
Versione semplificata



K=5 → dimensione di C = 6



1.



2.

- In **C** vi sono le occorrenze dei valori presenti in **A**
- Ora si può scandire l'array **C** in ordine, e inserire nel vettore **A C[i]** copie del valore **i**

Programma

- *Soluzione in sorting-semplificato.cc*
- Verificare quale delle due soluzioni è più efficiente a parità di numero di elementi n (con n grande)