9. Sortieren in linearer Zeit

- Es gibt Algorithmen, die die $\Omega(n\log(n))$ untere Schranke für Vergleichssortierer schlagen.
- Diese Algorithmen benutzen neben Vergleichen und Kontrolloperationen noch andere Operationen wie z.B. arithmetische Operationen auf den zu sortierenden Zahlen.
- Allerdings benötigen diese Algorithmen gewisse Einschränkungen, um eine Laufzeit von $o(n \log n)$ zu erzielen.
- Zu diesen Algorithmen gehören Counting-Sort, Bucket-Sort und Radix-Sort.

Sortieren durch Abzählen (1)

Annahme: Es gibt ein dem Algorithmus Counting-Sort bekannter Parameter k, so dass für die Eingabefolge $(a_1,\ldots,a_n)$ gilt:

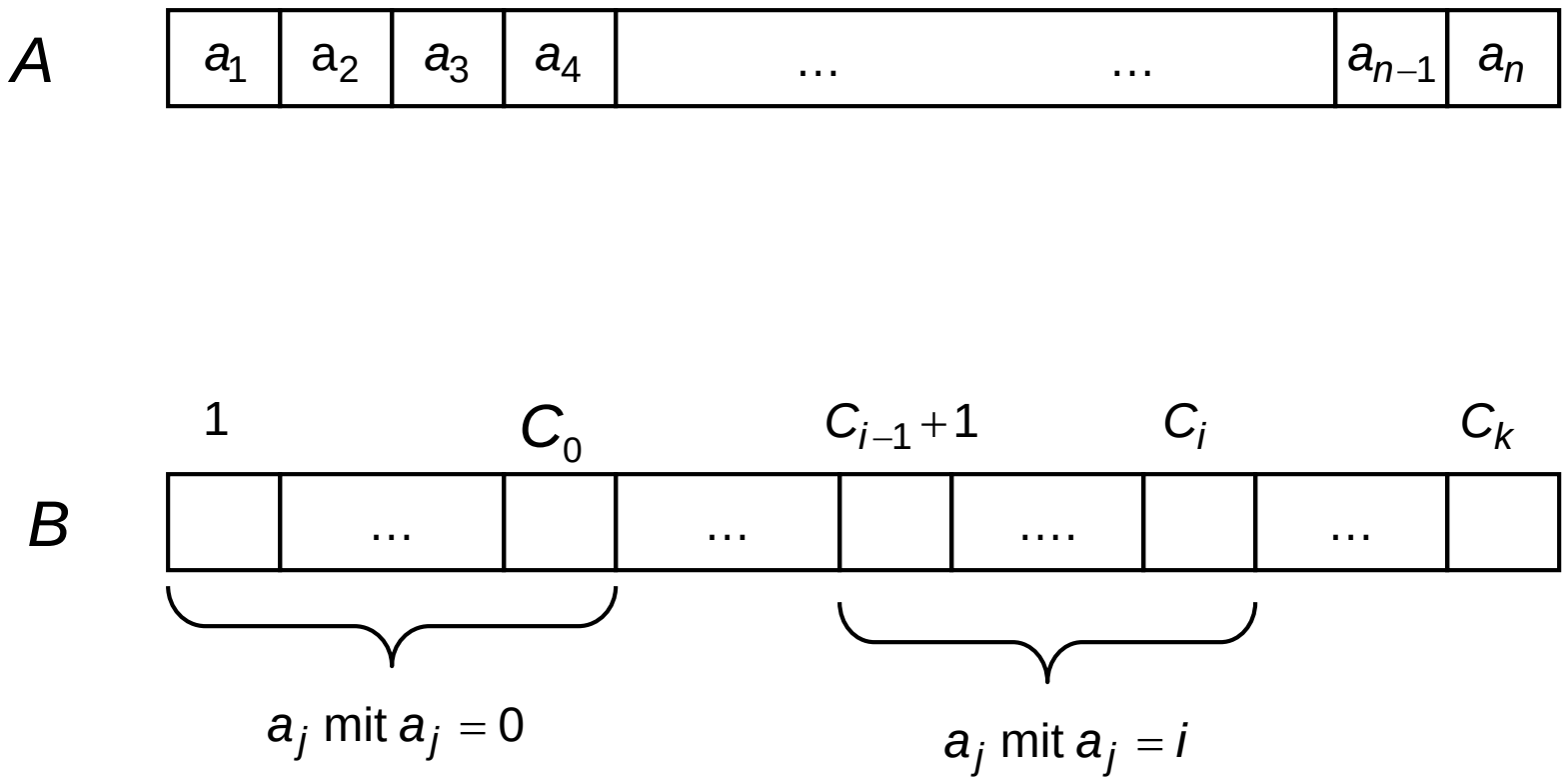
$$0 \le a_i \le k \text{ für alle } 1 \le i \le n.$$

Algorithmusidee:

1. Für alle $i, 0 \le i \le k$ bestimme Anzahl C_i der a_j mit $a_j \le i$.
2. Kopiere a_j mit $a_j = i$ in Felder $B[C_{i-1}+1],\ldots,B[C_i]$ eines Arrays B mit *length*[B]=n. Dabei gilt $C_{-1} = 0$.

Es gilt: $C_i - C_{i-1}$ ist die Anzahl der a_j mit $a_j = i$.

Sortieren durch Abzählen (2)



Counting-Sort

Counting-Sort(A,B,k)

```
1.  for i←0 to k do
2.      C[i]←0
3.  for j←1 to length(A) do
4.      C[A[j]]←C[A[j]]+1
5.   ▷ C[i] enthält Anzahl der Elemente in A mit Wert i
6.  for i←1 to k do
7.      C[i]←C[i]+C[i-1]
8.   ▷ C[i] enthält Anzahl der Elemente in A mit Wert ≤i
9.  for j←length(A) downto 1 do
10.     B[C[A[j]]]←A[j]
11.     C[A[j]]←C[A[j]]-1
```

Illustration für Counting-Sort (1)

A

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

C
(nach Zeilen 3-4)

0	1	2	3	4	5
0	0	0	0	0	0

C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

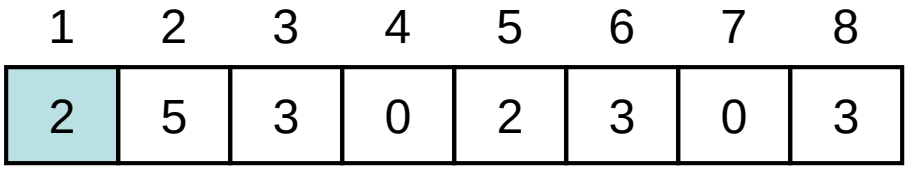
1	2	3	4	5	6	7	8
2	5	3	0	2	3	0	3

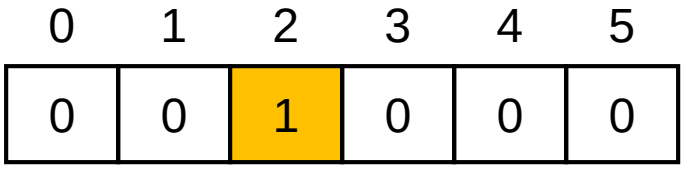
C
(nach Zeilen 3-4)

0	1	2	3	4	5
0	0	0	0	0	0

C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)





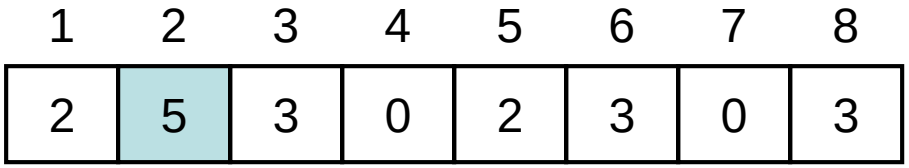
(nach Zeilen 3-4)

C

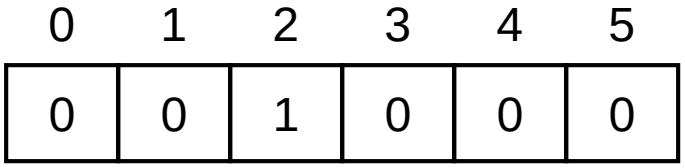
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

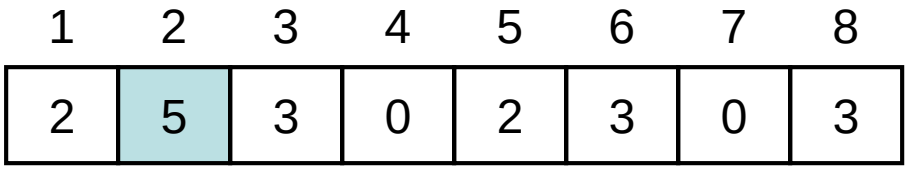


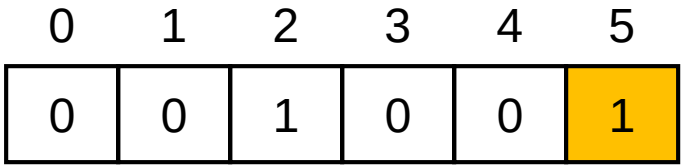
C
(nach Zeilen 3-4)



C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

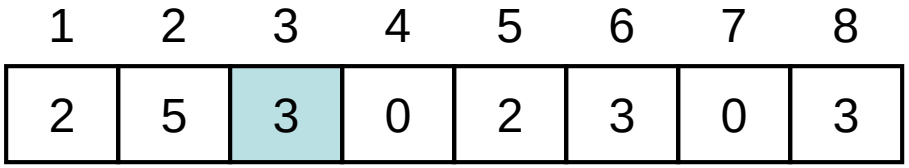




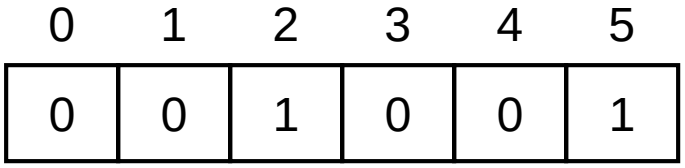
C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A



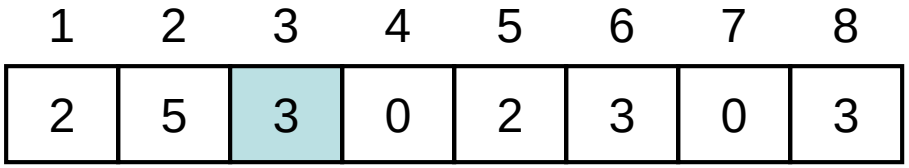
C
(nach Zeilen 3-4)



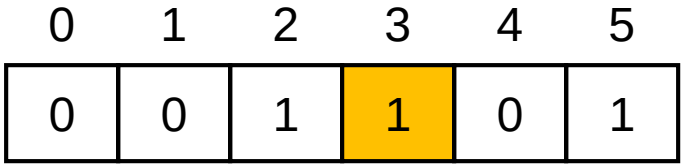
C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A



C
(nach Zeilen 3-4)



C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

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

C
(nach Zeilen 3-4)

0	1	2	3	4	5
0	0	1	1	0	1

C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

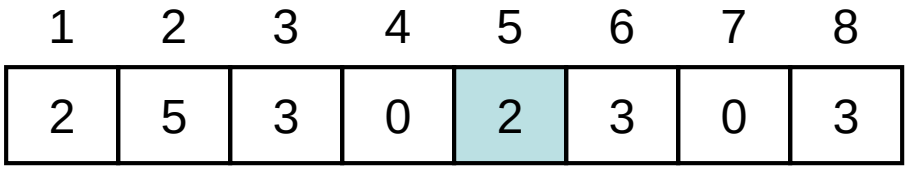
1	2	3	4	5	6	7	8
2	5	3	0	2	3	0	3

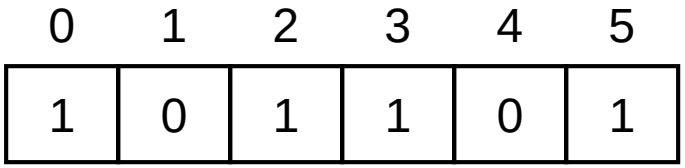
C
(nach Zeilen 3-4)

0	1	2	3	4	5
1	0	1	1	0	1

C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)





C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

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

C
(nach Zeilen 3-4)

0	1	2	3	4	5
1	0	2	1	0	1

C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

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

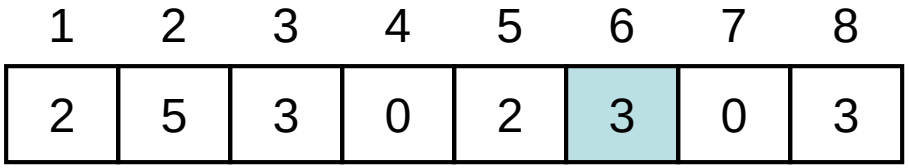
C
(nach Zeilen 3-4)

0	1	2	3	4	5
1	0	2	1	0	1

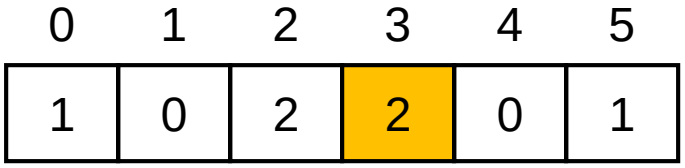
C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A



C
(nach Zeilen 3-4)



C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

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

C
(nach Zeilen 3-4)

0	1	2	3	4	5
1	0	2	2	0	1

C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

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

C
(nach Zeilen 3-4)

0	1	2	3	4	5
2	0	2	2	0	1

C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A

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

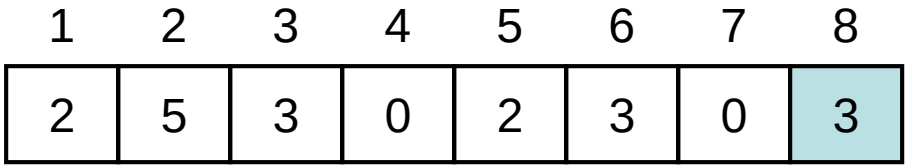
C
(nach Zeilen 3-4)

0	1	2	3	4	5
2	0	2	2	0	1

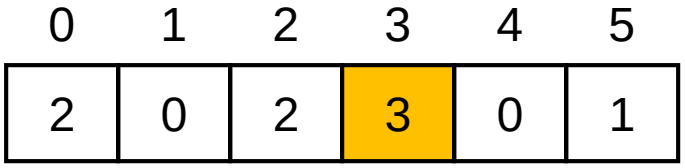
C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

A



C
(nach Zeilen 3-4)



C
(nach Zeilen 6-7)

Illustration für Counting-Sort (1)

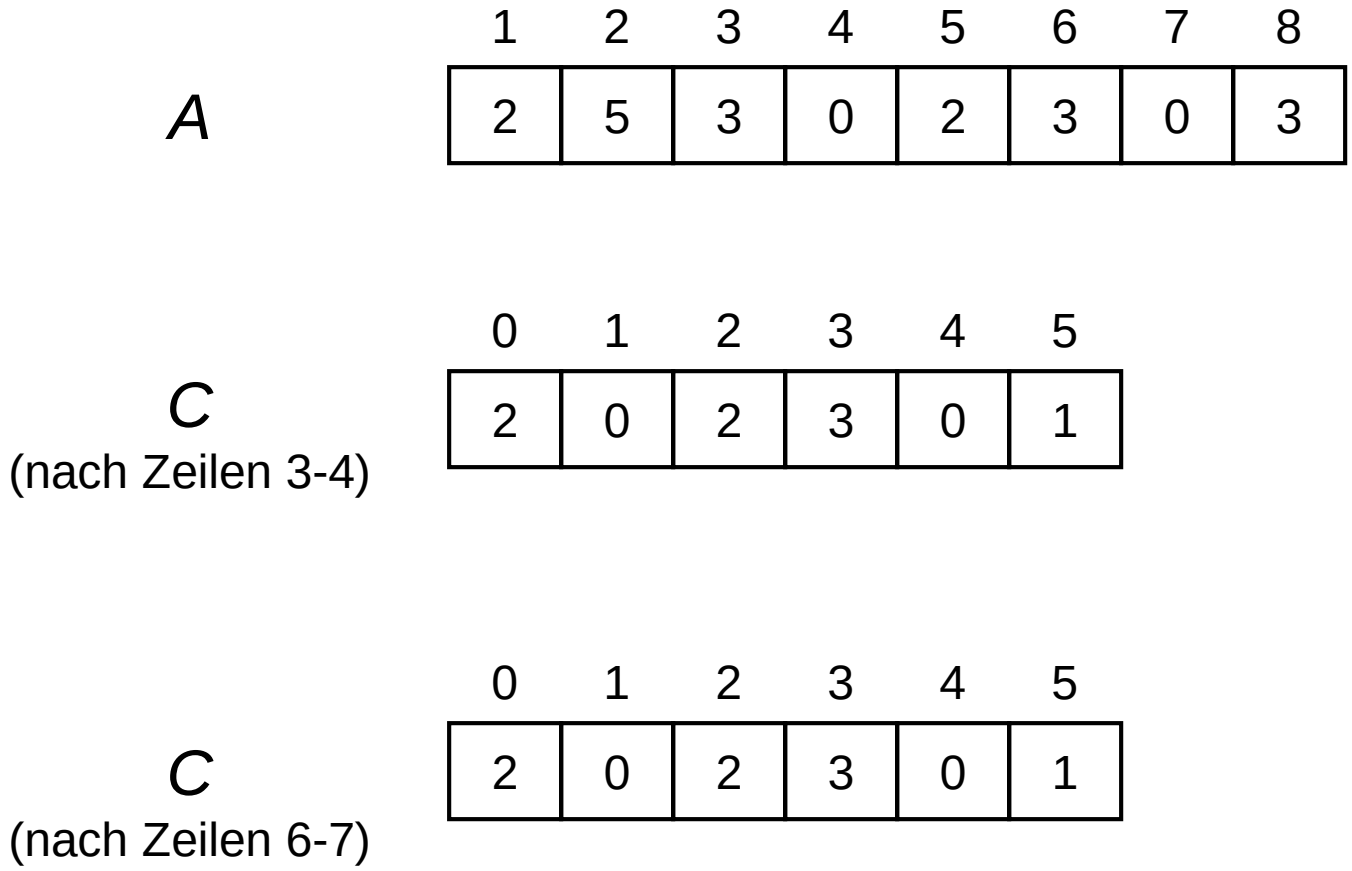


Illustration für Counting-Sort (1)

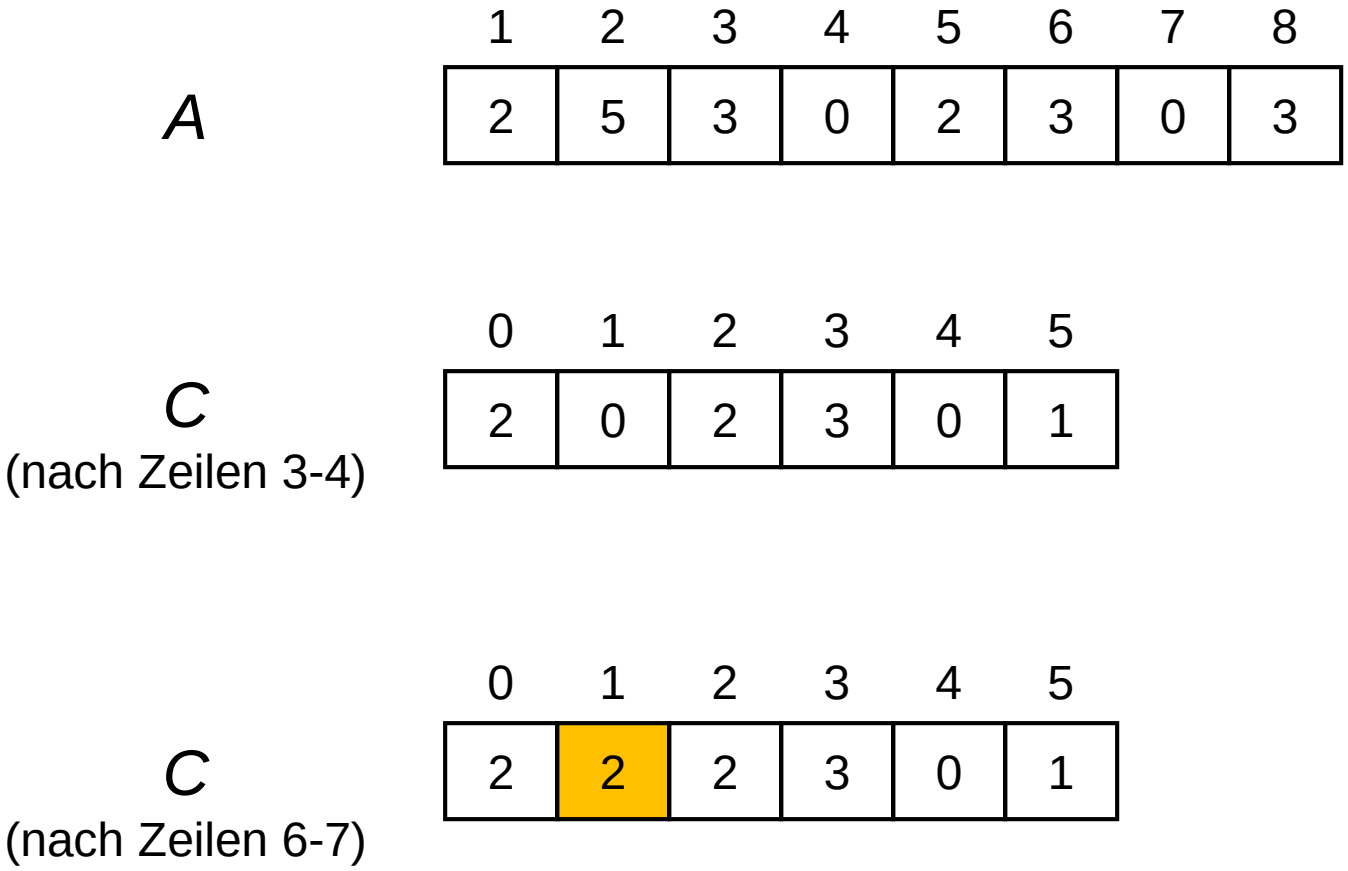


Illustration für Counting-Sort (1)

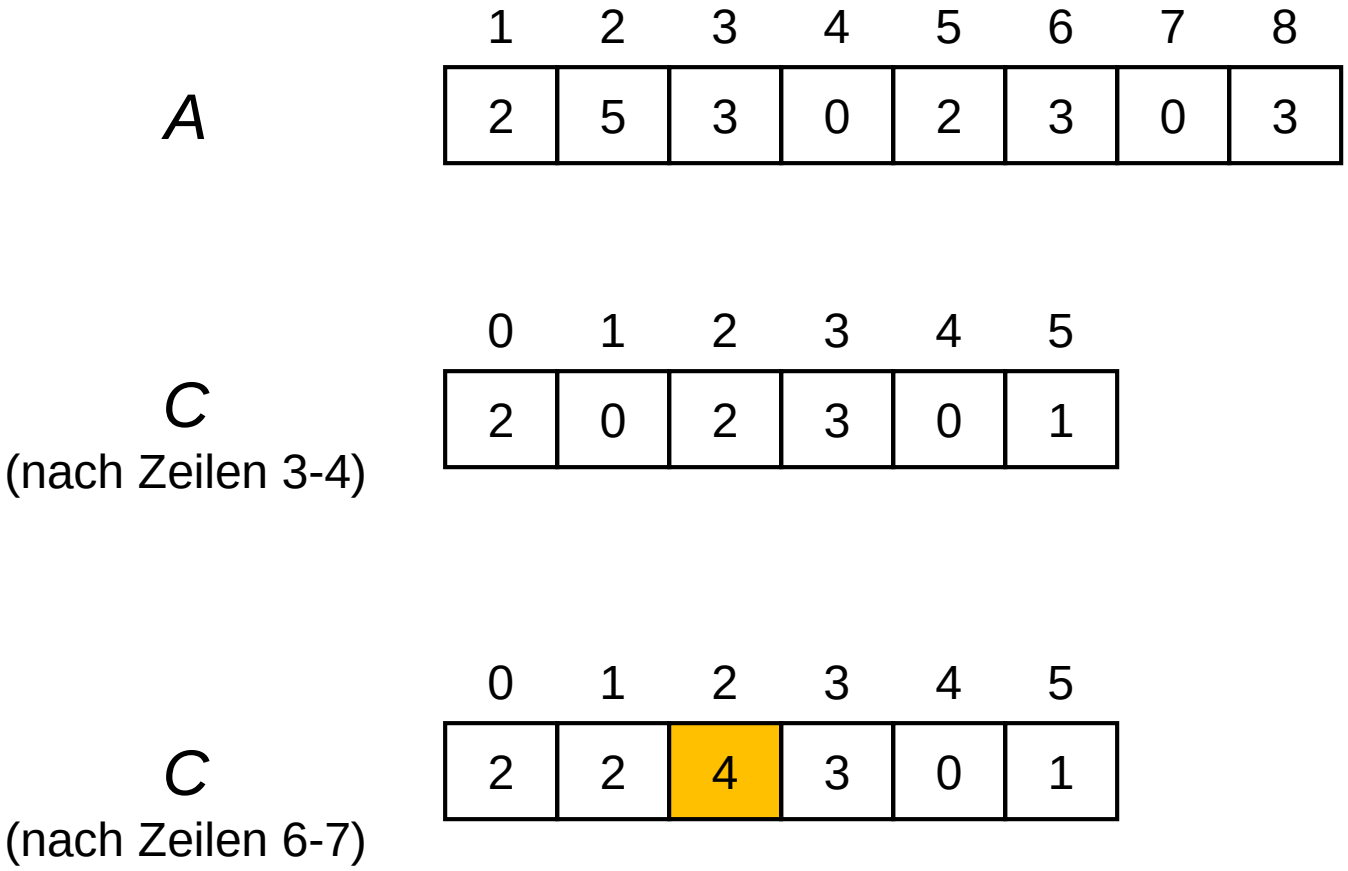


Illustration für Counting-Sort (1)

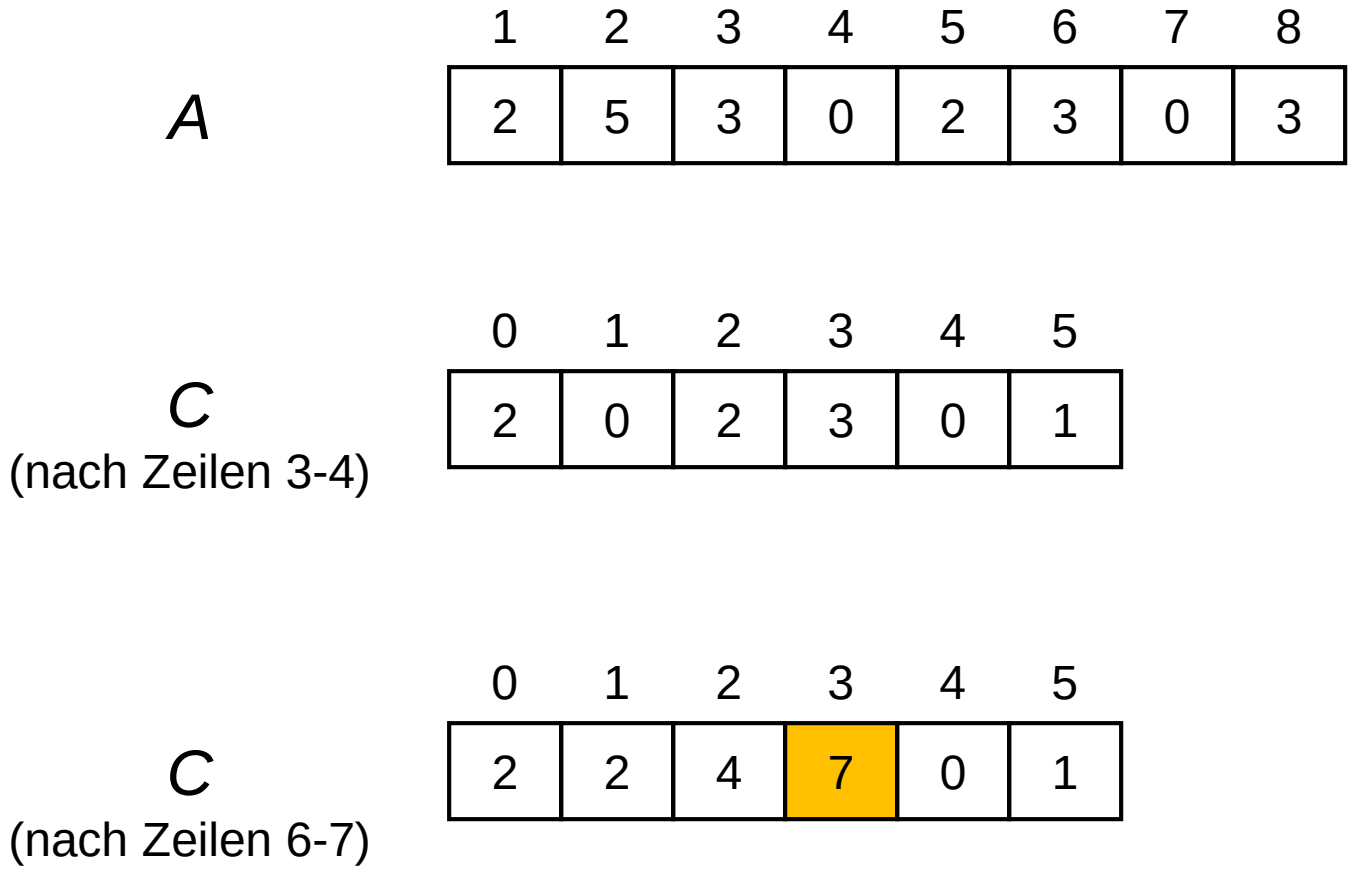


Illustration für Counting-Sort (1)

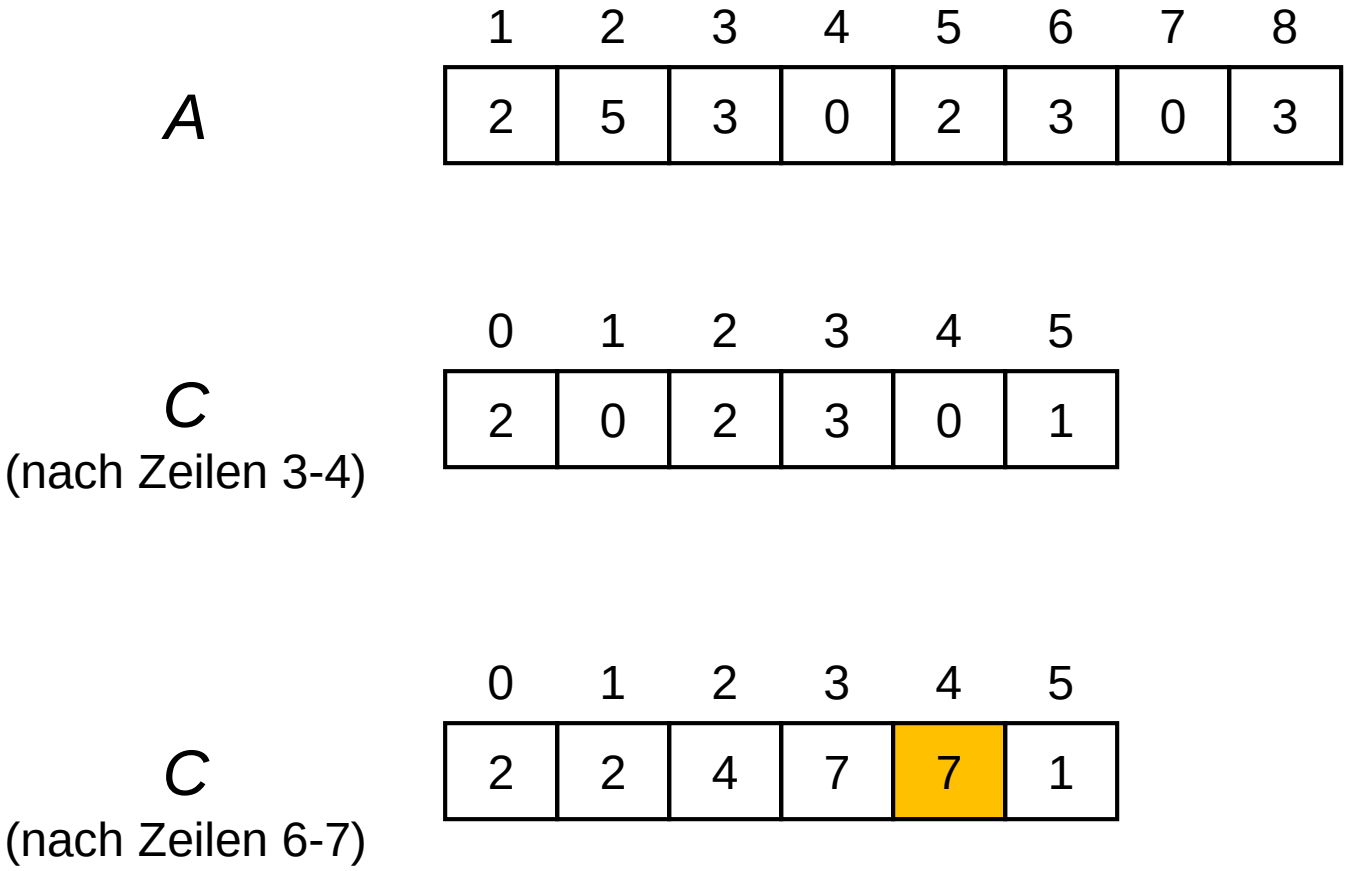


Illustration für Counting-Sort (1)

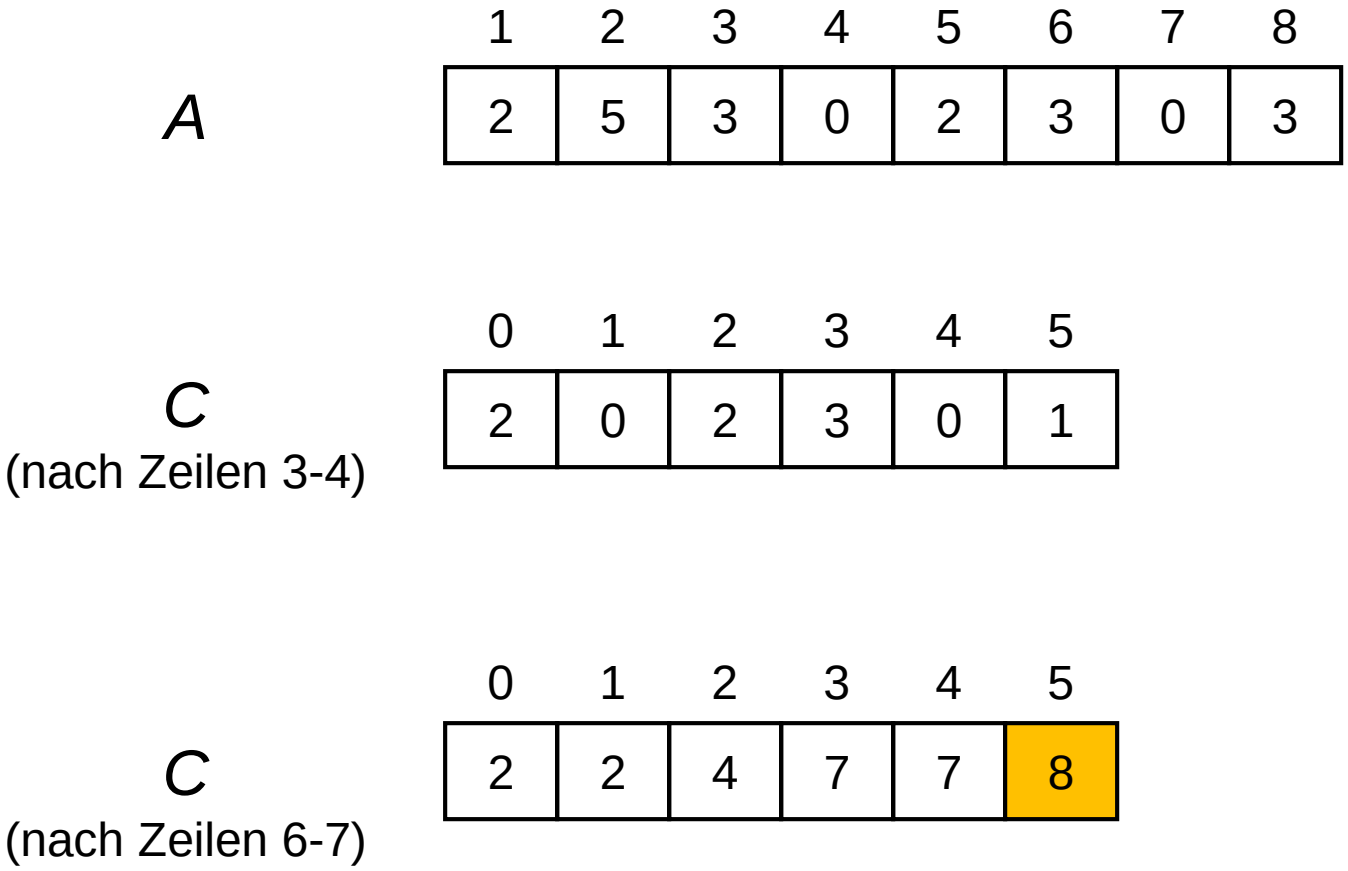


Illustration für Counting-Sort (1)

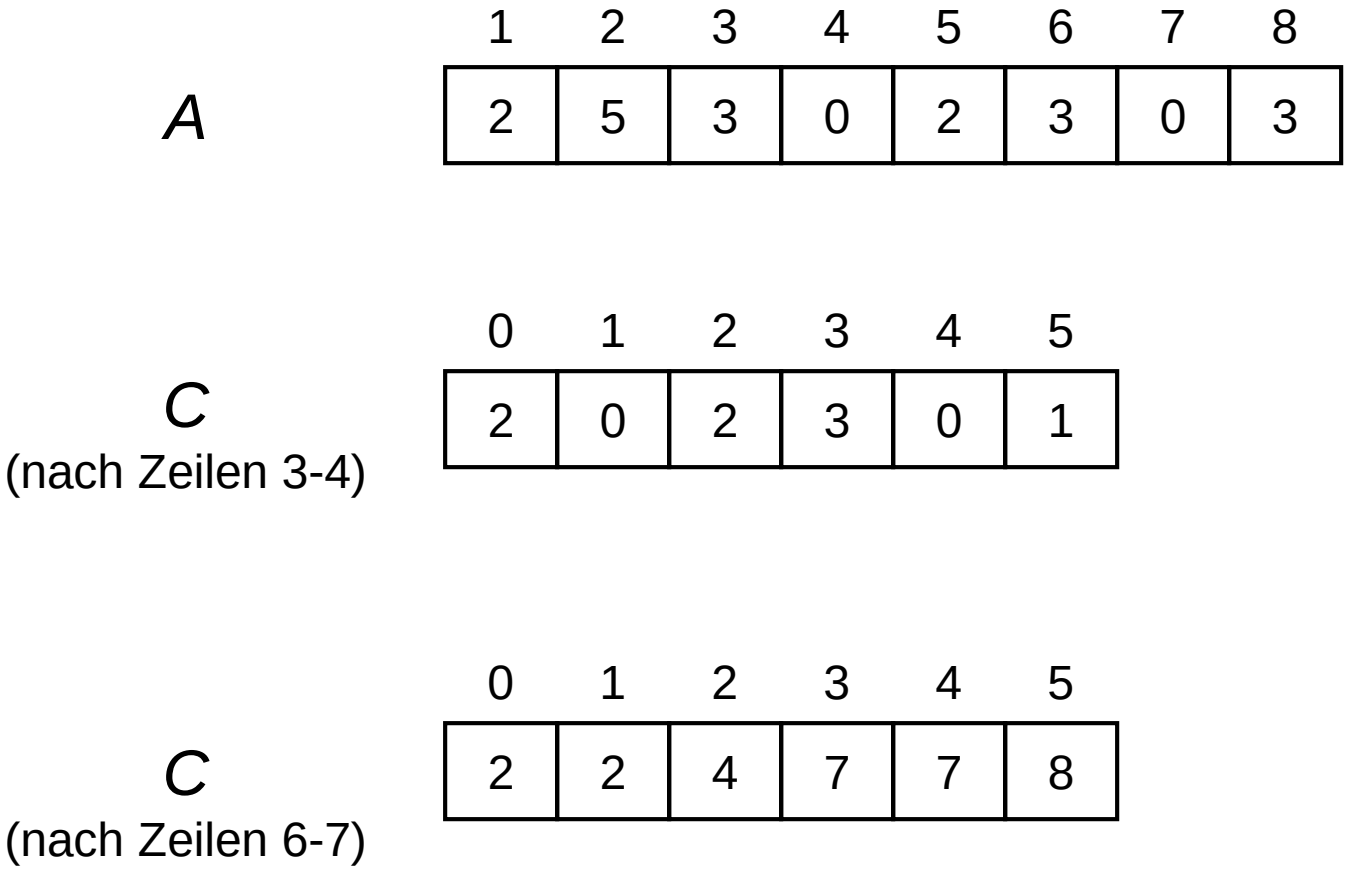


Illustration für Counting-Sort (2)

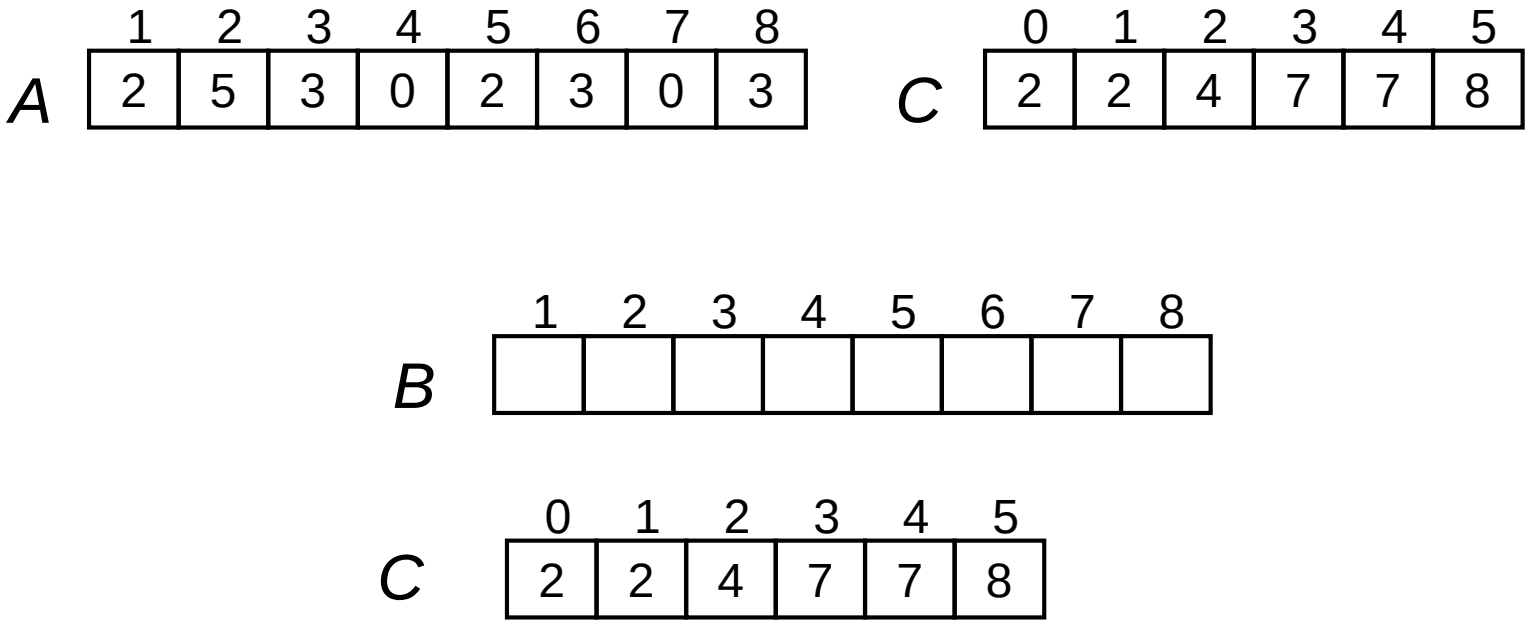


Illustration für Counting-Sort (2)

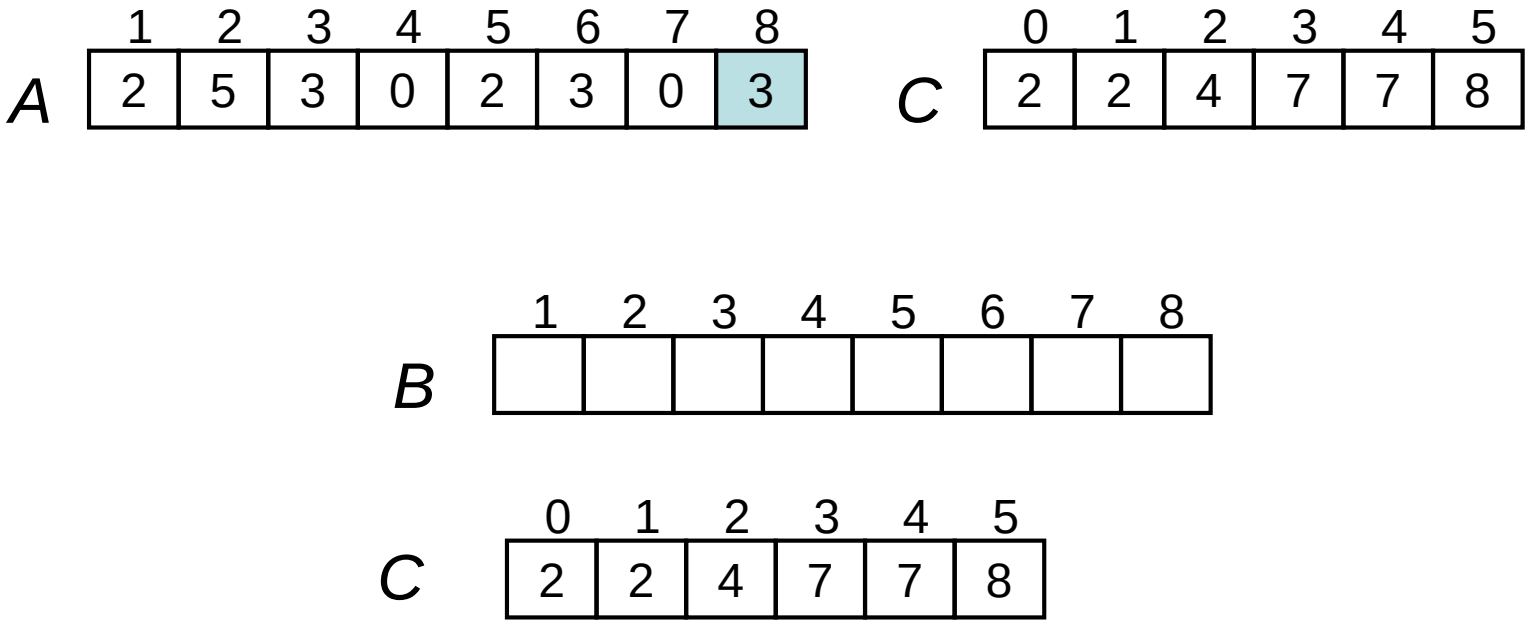


Illustration für Counting-Sort (2)

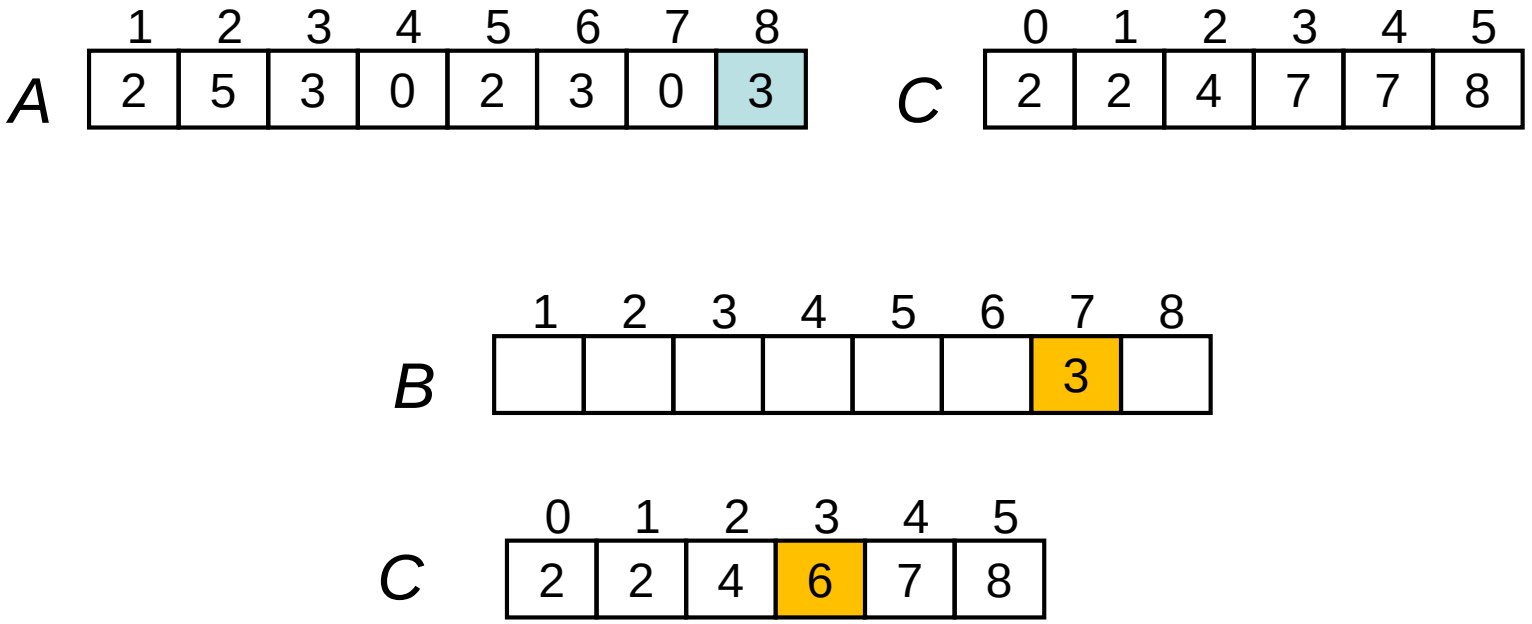


Illustration für Counting-Sort (2)

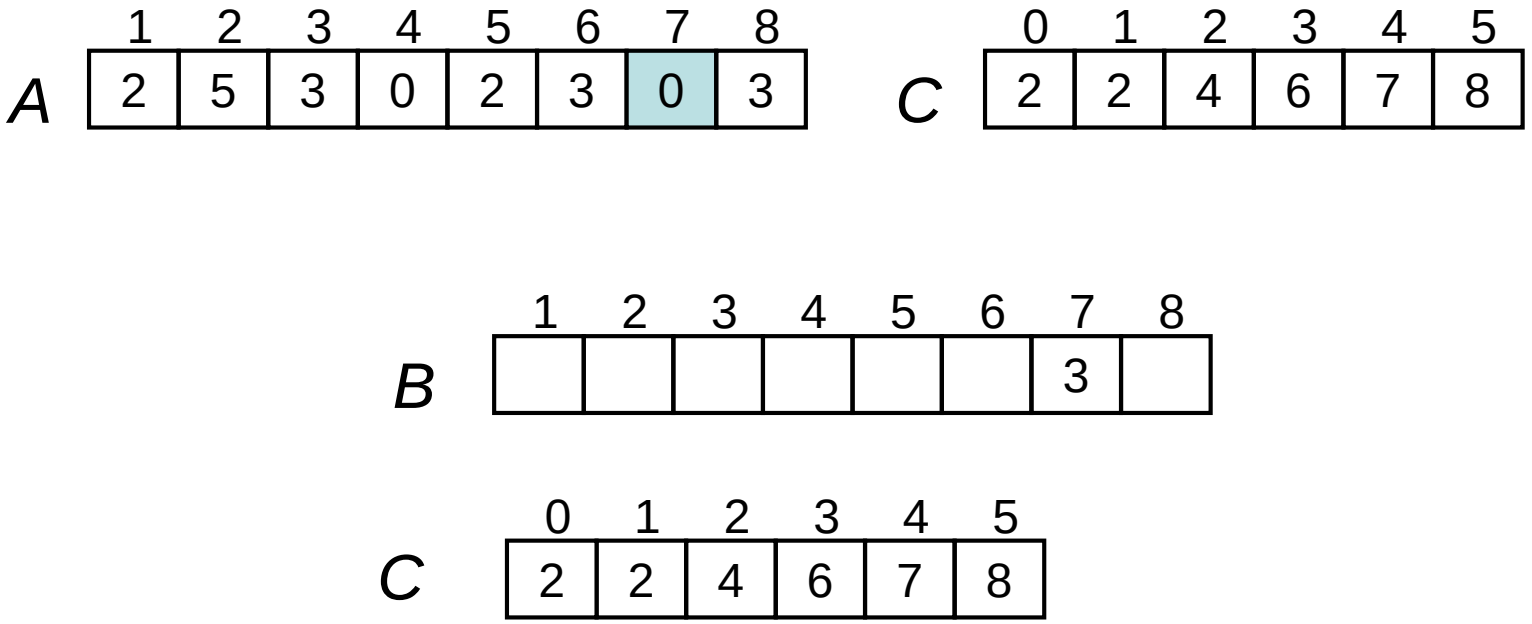


Illustration für Counting-Sort (2)

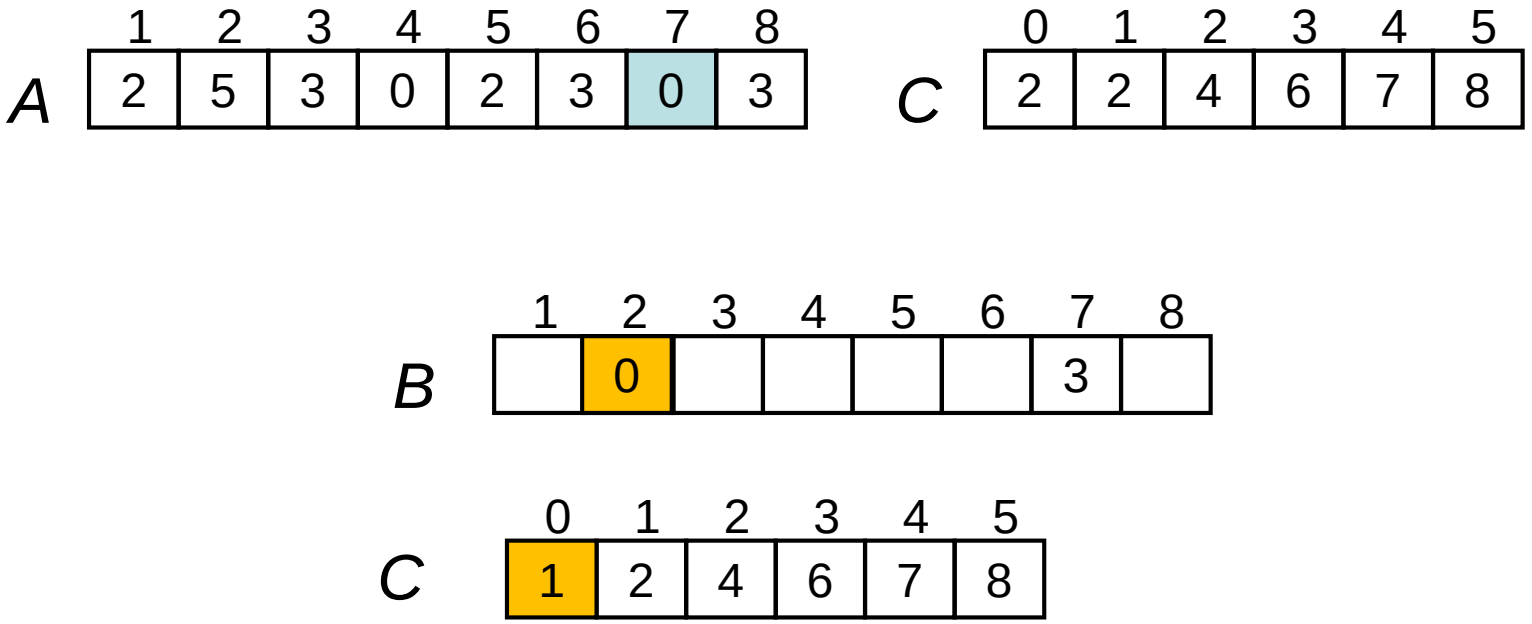


Illustration für Counting-Sort (2)

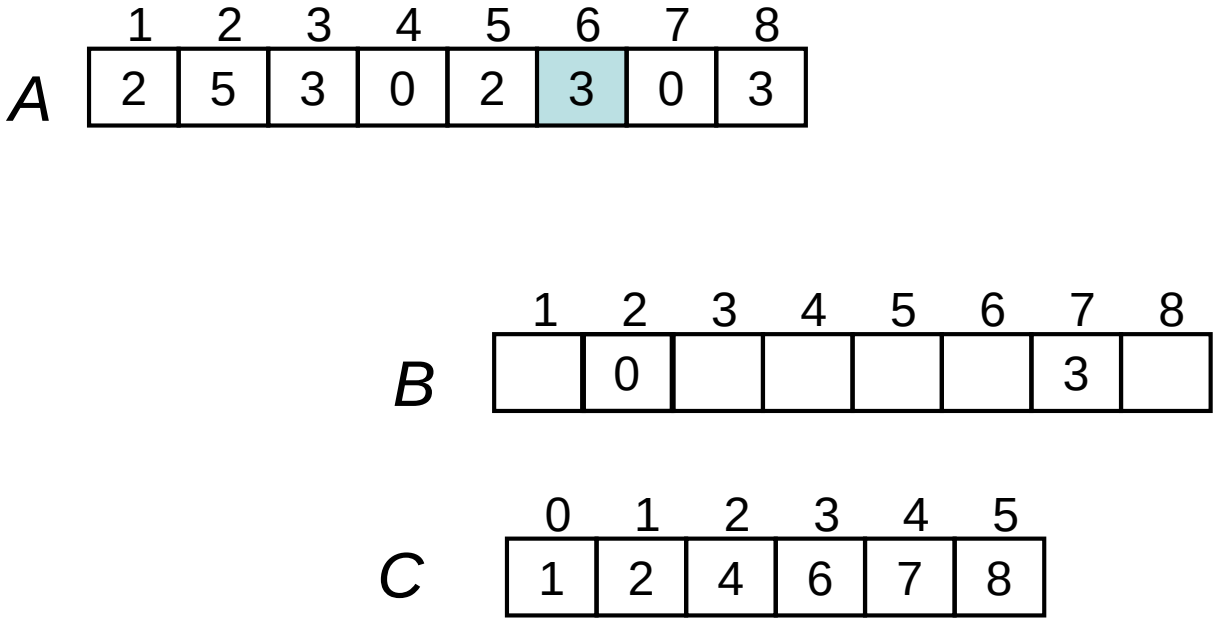
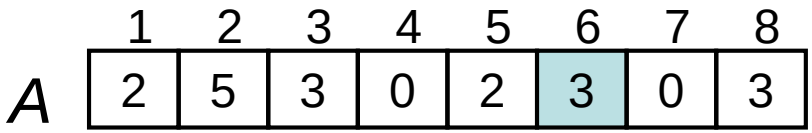
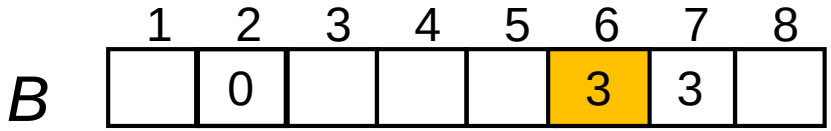


Illustration für Counting-Sort (2)





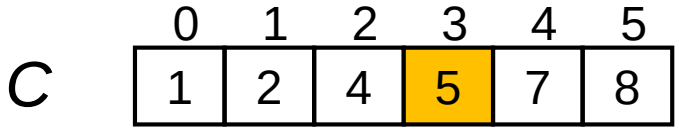


Illustration für Counting-Sort (2)

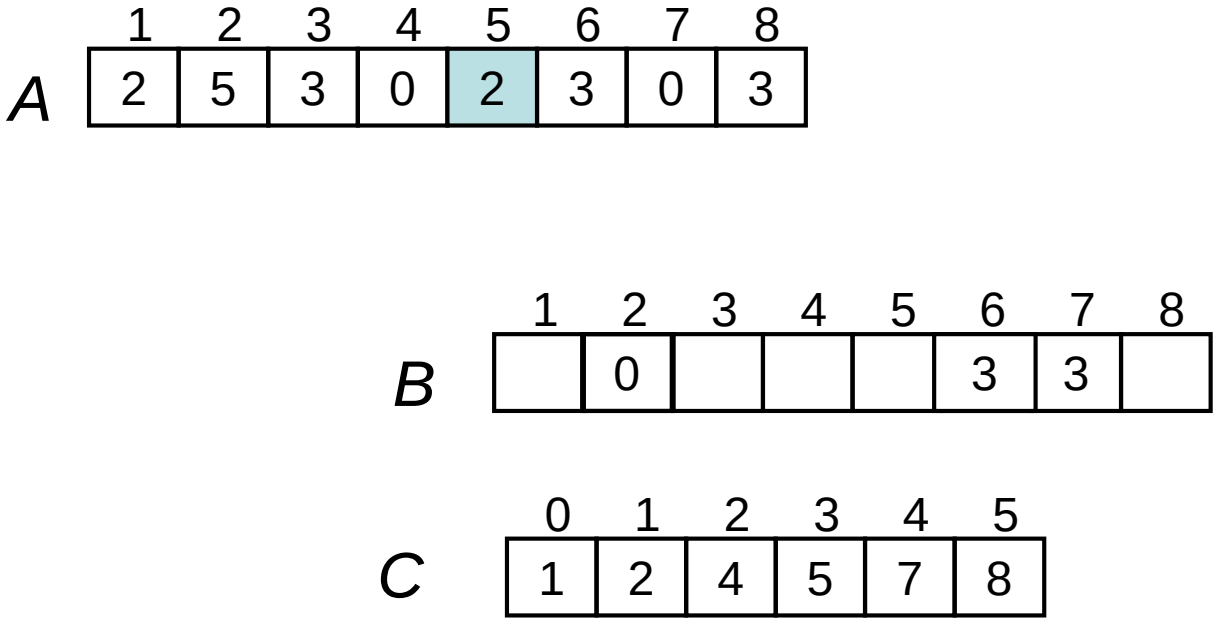


Illustration für Counting-Sort (2)

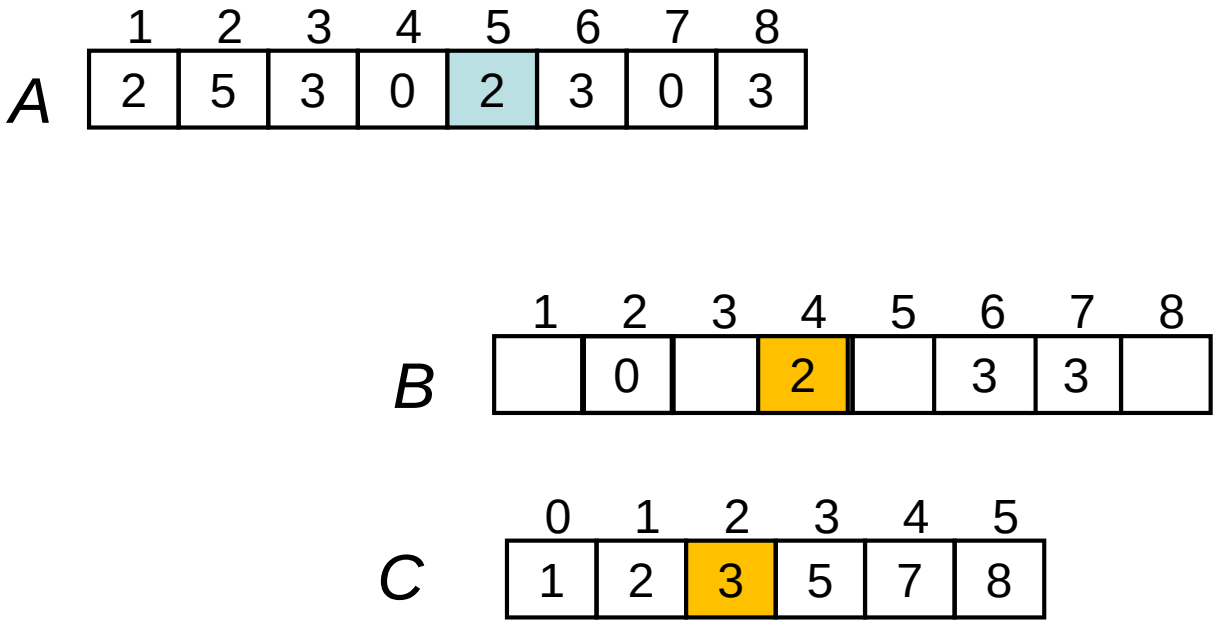
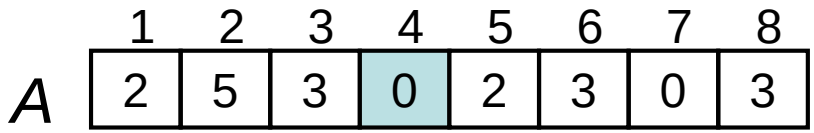
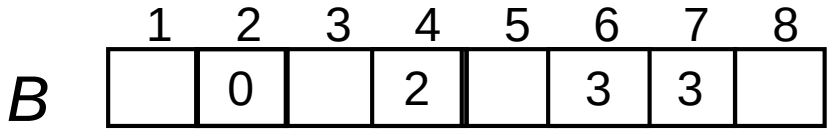


Illustration für Counting-Sort (2)





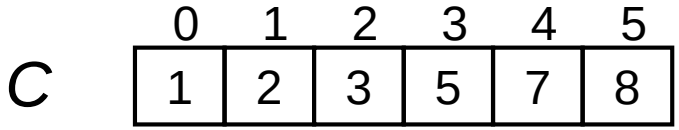
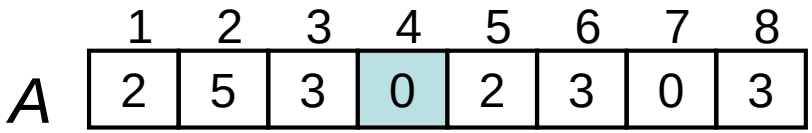
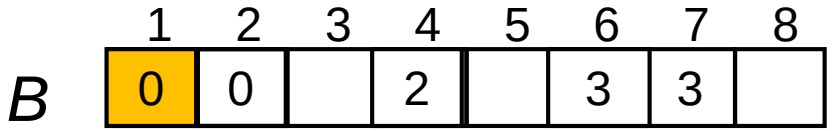


Illustration für Counting-Sort (2)





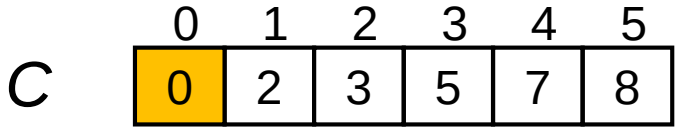


Illustration für Counting-Sort (2)

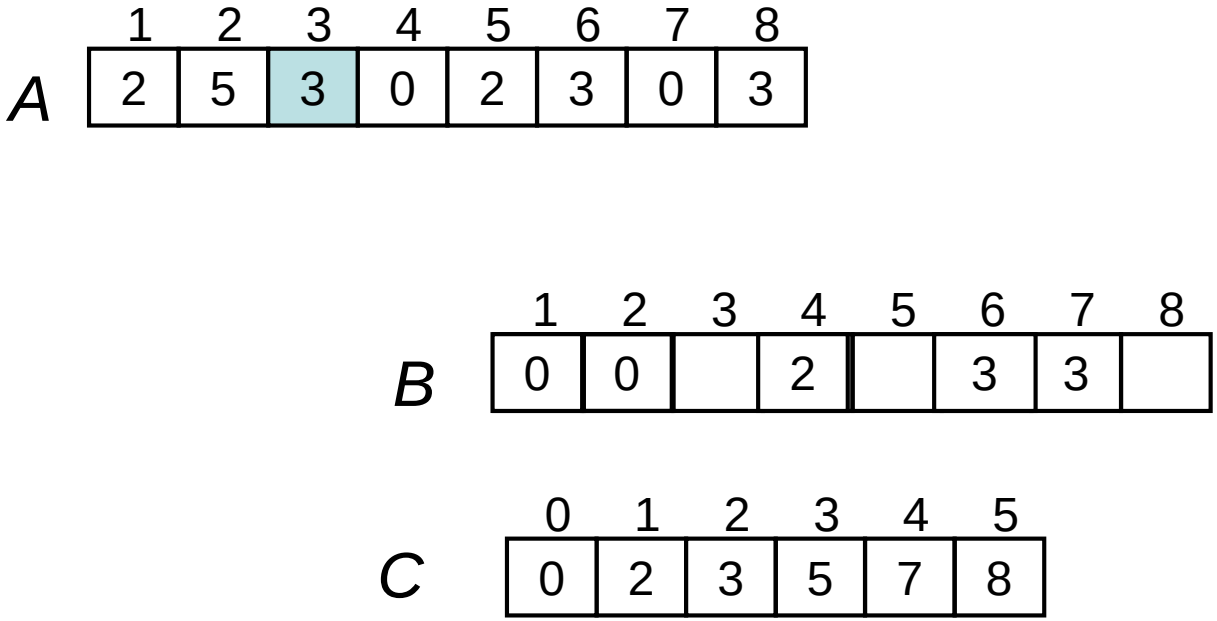
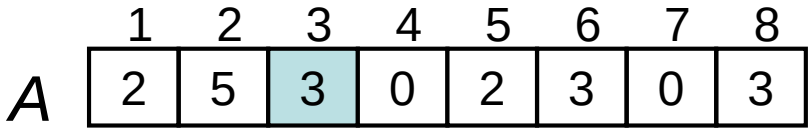
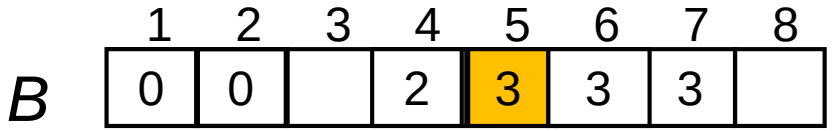


Illustration für Counting-Sort (2)





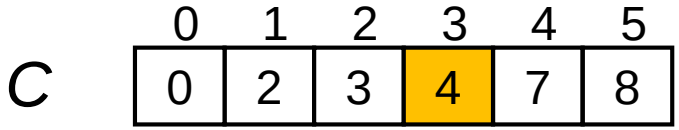


Illustration für Counting-Sort (2)

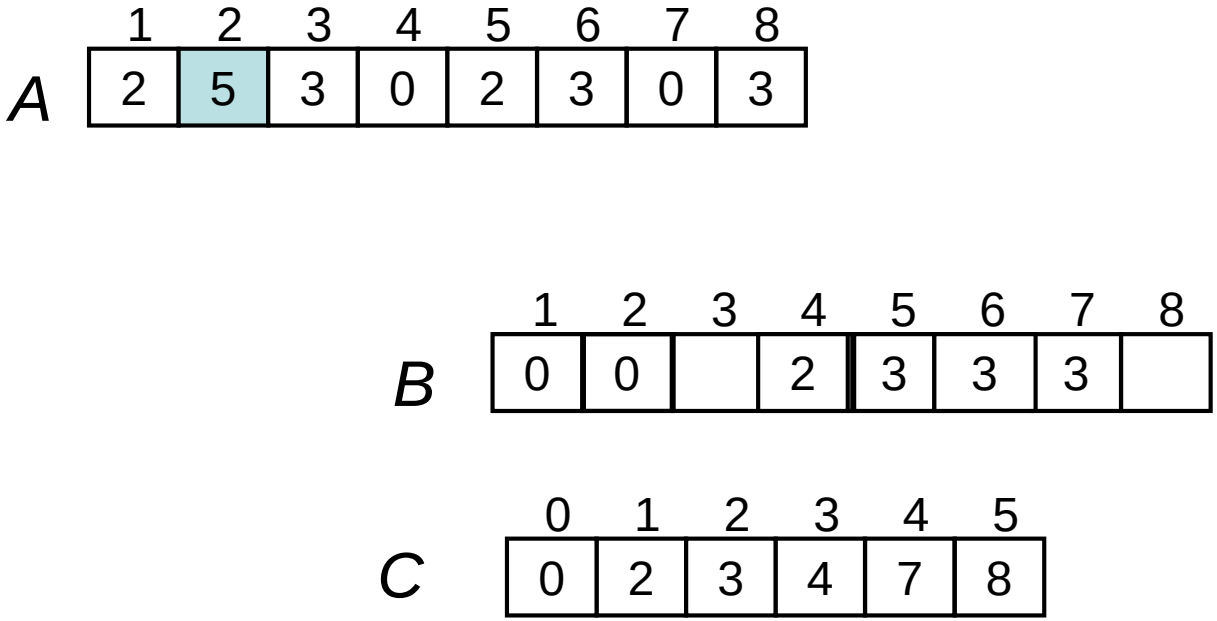
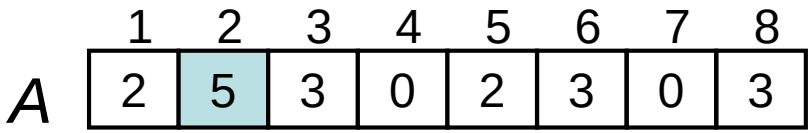
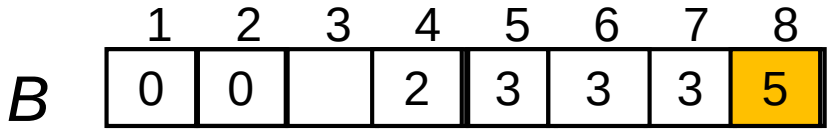


Illustration für Counting-Sort (2)





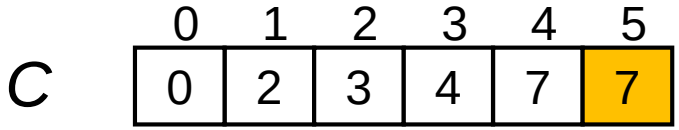
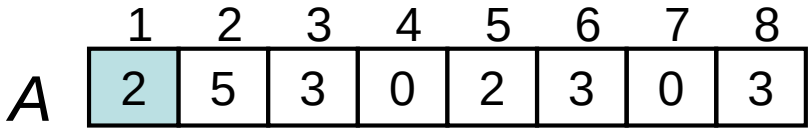
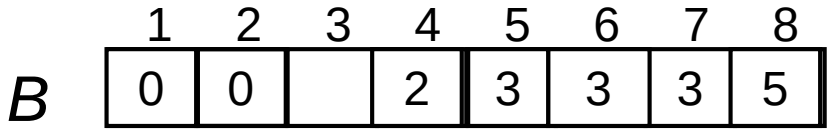


Illustration für Counting-Sort (2)





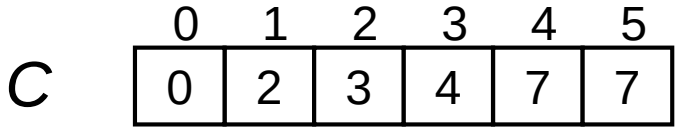


Illustration für Counting-Sort (2)

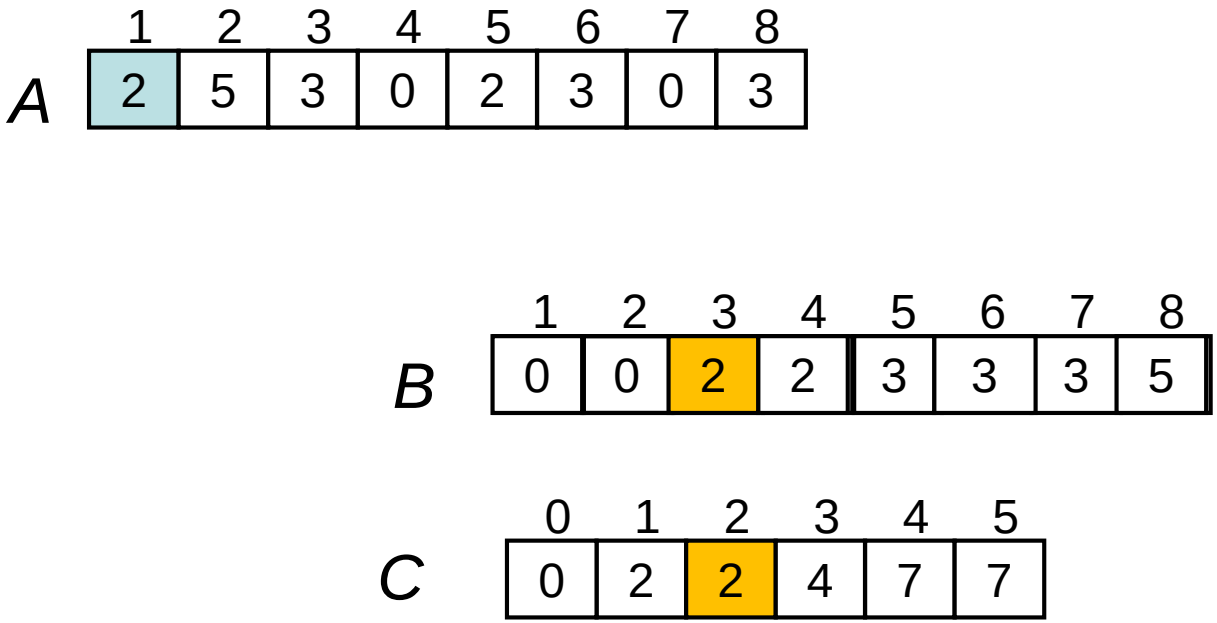
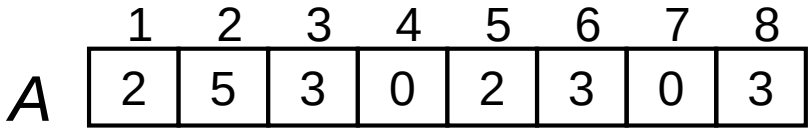
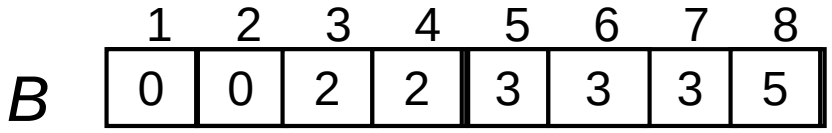
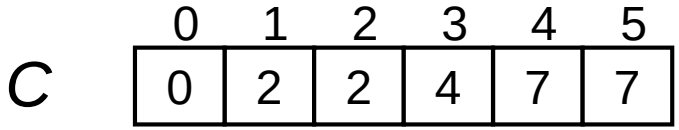


Illustration für Counting-Sort (2)







Laufzeit von Counting-Sort

Counting-Sort(A,B,k)

```
1.   for i←0 to k do
2.      C[i]←0
3.   for j←1 to length(A) do
4.      C[A[j]]←C[A[j]]+1
5.    ▷ C[i] enthält Anzahl der Elemente in A mit Wert i
6.   for i←1 to k do
7.      C[i]←C[i]+C[i-1]
8.    ▷ C[i] enthält Anzahl der Elemente in A mit Wert ≤i
9.   for j←length(A) downto 1 do
10.     B[C[A[j]]]←A[j]
11.     C[A[j]]←C[A[j]]-1
```

- Zeilen 1-2, 6-7: jeweils $\mathbf{O}(k)$
- Zeilen 3-4, 9-11: jeweils $\mathbf{O}(n)$

Laufzeit von Counting-Sort

Satz 9.1: Counting-Sort besitzt Laufzeit $\mathbf{O}(n+k)$.

Korollar 9.2: Gilt $k = \mathbf{O}(n)$, so besitzt Counting-Sort Laufzeit $\mathbf{O}(n)$.

Korrektheit von Counting-Sort

Counting-Sort(A,B,k)

1. for i$\leftarrow$0 to k do
2. C[i]$\leftarrow$0

   $I_1(j)$: $\forall i \in \{0,\ldots,length(A)\}$: C[i]= #i in A[1,…,j-1]
3. for j$\leftarrow$1 to length(A) do
4. C[A[j]]$\leftarrow$C[A[j]]+1
5. ▷ C[i] enthält Anzahl der Elemente in A mit Wert i

   $I_2(j)$: $\forall j \in \{0,\ldots,i-1\}$: C[j]= #$\leq$j in A und $\forall j \in \{i,\ldots,k\}$: C[j]= #j in A
6. for i$\leftarrow$1 to k do
7. C[i]$\leftarrow$C[i]+C[i-1]
8. ▷ C[i] enthält Anzahl der Elemente in A mit Wert $\leq$i

   $I_3(j)$: $\forall i \in \{j+1,\ldots,length(A)\}$: Pos. von A[i] in B ist gleich
   $|\{ 1 \leq l \leq length(A) \mid A[l] < A[i] \}| + |\{1 \leq l < i \mid A[l] = A[i] \}| + 1$ und
   $\forall i \in \{0,\ldots,k\}$: $C[i] = |\{ 1 \leq l \leq length(A) \mid A[l] < i \}| + |\{1 \leq l \leq j \mid A[l] = i\}|$
9. for j$\leftarrow$length(A) downto 1 do
10. B[C[A[j]]]$\leftarrow$A[j]
11. C[A[j]]$\leftarrow$C[A[j]]-1

Korrektheit von Counting-Sort

Counting-Sort(A,B,k)

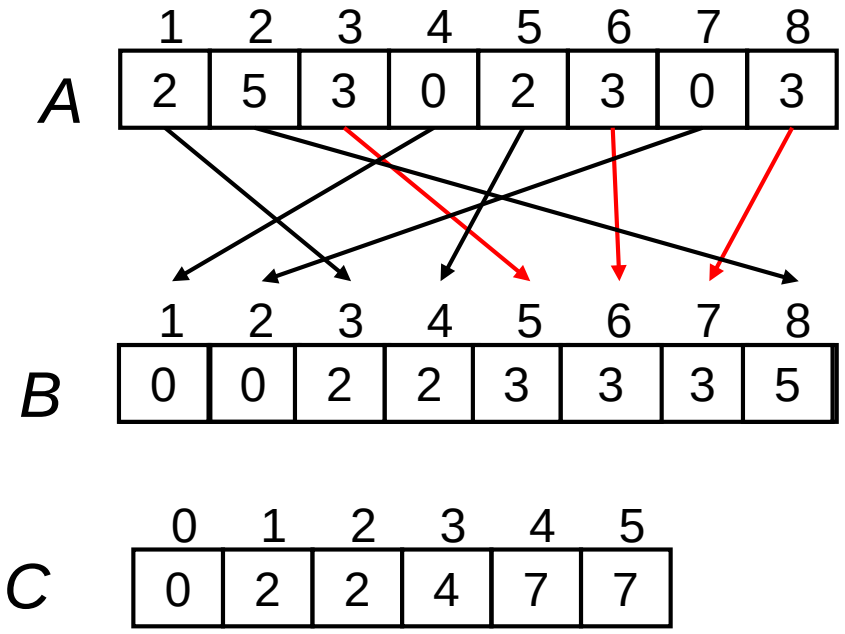
```
1.  for i←0 to k do
2.      C[i]←0
3.  for j←1 to length(A) do
4.      C[A[j]]←C[A[j]]+1
5.   ▷ C[i] enthält Anzahl der Elemente in A mit Wert i
6.  for i←1 to k do
7.      C[i]←C[i]+C[i-1]
8.   ▷ C[i] enthält Anzahl der Elemente in A mit Wert ≤i
9.  for j←length(A) downto 1 do
10.     B[C[A[j]]]←A[j]
11.     C[A[j]]←C[A[j]]-1
```

Am Ende gilt: $\forall i \in \{1,\ldots,length(A)\}$: Pos. von A[i] in B ist gleich
$|\{ 1 \le l \le length(A) \mid A[l] < A[i] \}| + |\{ 1 \le l < j \mid A[l] = A[i] \}| + 1$
d.h. die Folge ist sortiert (bei Beibehaltung der Reihenfolge gleicher El.)

Korrektheit von Counting-Sort

Am Ende gilt: $\forall i \in \{1,\ldots,length(A)\}$: Pos. von A[i] in B ist gleich
$|\{ 1 \le l \le length(A) \mid A[l] < A[i] \}| + |\{ 1 \le l < j \mid A[l] = A[i] \}| + 1$
d.h. die Folge ist sortiert (bei Beibehaltung der Reihenfolge gleicher El.)

Betrachte unser Beispiel:



Radixsort

Ideen:

- verwende k-adische Darstellung der Schlüssel
- Sortiere Ziffer für Ziffer gemäß Counting-Sort
- Behalte bei Betrachtung der i-ten Ziffer die Reihenfolge der Elemente mit gleicher i-ter Ziffer bei (s. Countingsort)

Annahme: für alle Zahlen z: $0 \leq z < k^d$

Radixsort

Radixsort(A,k)

1. for i$\leftarrow$0 to d-1 do
2. Counting-Sort(A,B,i,k-1)
3. ▷ sortiere gemäß der $A_i[j]$-Werte, wobei $A_i[j]$ die i-te Ziffer in A[j] angibt

Laufzeit: O(d(n+k))

k=n und d konstant: Laufzeit O(n).

Radixsort

Counting-Sort(A,B,i,k)

1. for i←0 to k do
2. C[i]←0
3. for j←0 to length(A) do
4. C[A_i[j]]←C[A_i[j]]+1
5. ▷ C[i] enthält Anzahl der Elemente in A mit Wert i
6. for i←1 to k do
7. C[i]←C[i]+C[i-1]
8. ▷ C[i] enthält Anzahl der Elemente in A mit Wert ≤i
9. for j←length(A) downto 1 do
10. B[C[A_i[j]]]←A[j]
11. C[A_i[j]]←C[A_i[j]]-1

Radixsort

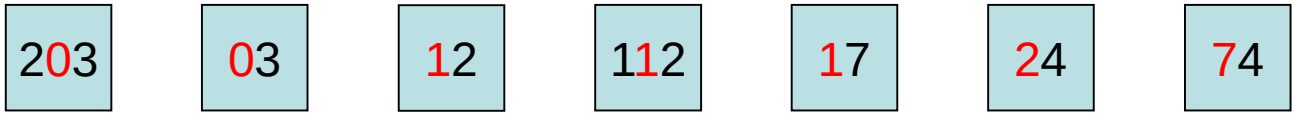
Beispiel:

Nach Counting-Sort(A,B,0,9):

Radixsort

Beispiel:

Nach Counting-Sort(A,B,1,9):



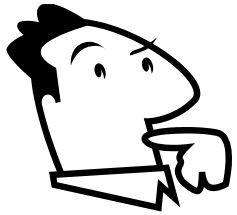
Radixsort

Beispiel:

203	3	12	112	17	24	74

Nach Counting-Sort(A,B,1,9):

003	012	017	024	074	112	203



Zauberei???

Radixsort

Korrektheit:

- Für jedes Paar $a<b$ in A gilt:
  es existiert i mit $a_i<b_i$ und $a_j=b_j$ für alle $j>i$
- Schleifendurchlauf für i: $pos_B(a)<pos_B(b)$
  ($pos_B(a)$: Position von a in Feld B)
- Schleifendurchlauf für $j>i$: Ordnung wird beibehalten wegen Counting-Sort, d.h. auch danach gilt $pos_B(a)<pos_B(b)$
- Da das für alle Zahlenpaare gilt, ist die Folge am Ende sortiert.

Changelog

09.05.16: Folien 4, 47, 49 (neu), 50 (neu), 51 (neu), 52

10.05.16: Folien 49, 50, 51, 58