

RADIX Sort

und COUNTING Sort

Robert Hilbrich

hilbrich@informatik.hu-berlin.de

Scenario



- Nach dem Studium
- Programmierer bei DATEV (Buchhaltung)
- ca. 80 000 Kunden in Deutschland
- pro Kunde: zw. 1000 – 100000 Datensätze / Jahr
- eindeutig indiziert (N, 10 stellig, ≥ 0000000001)

- ZIEL:
 - Sortierung aller Datensätze (Index)
 - maximal effizient und schnell (Zeit = Geld!)

Scenario – Lösung?



- Wissen: Quicksort
 - Aufwand $O(n \log n)$

- Geht es scheller?

Szenario – Lösung?



- Wissen: Quicksort
 - Aufwand $O(n \log n)$

- Für Sortierverfahren die Schlüssel "vergleichen":
 - Aufwand = $O(n \log n)$

Szenario – Lösung?



- Wissen: Quicksort
 - min. Aufwand $O(n \log n)$
- Für Sortierverfahren die Schlüssel "vergleichen":
 - min. Aufwand = $O(n \log n)$
- Suchen: "schnelleren Algorithmus"
 - also: Sortieren ohne Vergleichen?

Schwerpunkte des Vortrages

- Teil 1: "Counting Sort"
 - Sortieren ohne Vergleichen
 - Voraussetzung für Teil 2
- Teil 2: "Radix Sort"
 - Verallgemeinerung von Teil 1
 - Anwendung für DATEV Beispiel

Teil 1 - Vorbereitungen

Counting Sort

Counting Sort

- Preis:
 - Wertebereich Eingabe: Integer, ≥ 1
- Gewinn (Laufzeit):
 - n Elemente
 - Eingabebereich von $1 \dots k$
 - Wenn $k = O(n)$, dann Komplexität $O(n)$.

Counting Sort - Grundidee

- **Eingabe:** Feld **A**

Counting Sort - Grundidee

- **Eingabe:** Feld A
- Für **jedes** Element $A[i]$:
 - $n \leftarrow$ Anzahl der Elemente $A[j]$ mit $A[j] < A[i]$

Counting Sort - Grundidee

- **Eingabe:** Feld A
- Für **jedes** Element $A[i]$:
 - $n \leftarrow$ Anzahl der Elemente $A[j]$ mit $A[j] < A[i]$
- **Ausgabe:** Feld B
 - Position von $A[i]$: $B[n+1]$

Counting Sort - Grundidee

- **Eingabe:** Feld A
- Für **jedes** Element $A[i]$:
 - $n \leftarrow$ Anzahl der Elemente $A[j]$ mit $A[j] < A[i]$
- **Ausgabe:** Feld B
 - Position von $A[i]$: $B[n+1]$
- Modifikation nötig:
 - falls $A[n] = A[m]$ mit $n \neq m$ existiert

Implementation

// Eingabe A, Ausgabe B, MaxWert k

Counting-Sort(A, B, k)

// Initialisierung vom Hilfsfeld C

for i = 1 to k **do** C[i] = 0;

Implementation

// Eingabe A, Ausgabe B, MaxWert k

Counting-Sort(A, B, k)

// Initialisierung vom Hilfsfeld C

for i = 1 to k **do** C[i] = 0;

// C[i] erhält die Anzahl der Elemente von A mit i = A[j]

for j = 1 to length[A] **do** C[A[j]] = C[A[j]] + 1;

Implementation

// Eingabe A, Ausgabe B, MaxWert k

Counting-Sort(A, B, k)

// Initialisierung vom Hilfsfeld C

for i = 1 to k **do** C[i] = 0;

// C[i] erhält die Anzahl der Elemente von A mit i = A[j]

for j = 1 to length[A] **do** C[A[j]] = C[A[j]] + 1;

// C[i] erhält die Anzahl der Elemente von A mit i <= A[j]

for i = 2 to k **do** C[i] = C[i] + C[i-1];

Implementation

// Eingabe A, Ausgabe B, MaxWert k

Counting-Sort(A, B, k)

// Initialisierung vom Hilfsfeld C

for i = 1 to k **do** C[i] = 0;

// C[i] erhält die Anzahl der Elemente von A mit i = A[j]

for j = 1 to length[A] **do** C[A[j]] = C[A[j]] + 1;

// C[i] erhält die Anzahl der Elemente von A mit i <= A[j]

for i = 2 to k **do** C[i] = C[i] + C[i-1];

// Erstellen des Ausgabefeldes B

for j = length[A] **downto** 1 **do**

 B[C[A[j]]] = A[j];

 C[A[j]] = C[A[j]] - 1;

Ablauf - Eingabe

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

1	2	3	4	5	6 (= k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf - Initialisierung

Counting-Sort(A, B, k)

for i = 1 to k **do** C[i] = 0;

for j = 1 to length[A] **do** C[A[j]] = C[A[j]] + 1;

for i = 2 to k **do** C[i] = C[i] + C[i-1];

for j = length[A] **downto** 1 **do**

 B[C[A[j]]] = A[j];

 C[A[j]] = C[A[j]] - 1;

Eingabefeld A

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

k = 6

Hilfsfeld C

0	0	0	0	0	0
1	2	3	4	5	6 (= k)

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 1$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

0	0	1	0	0	0		
1	2	3	4	5	6 (= k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 2$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

0	0	1	0	0	1		
1	2	3	4	5	6 (= k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 3$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

0	0	1	1	0	1		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 4$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

1	0	1	1	0	1		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 5$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

1	0	2	1	0	1		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 6$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

 |

$k = 6$

Hilfsfeld C

1	0	2	2	0	1		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 7$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	0	2	2	0	1		
1	2	3	4	5	6 (= k)		

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl pro Wert ($j = 8$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	0	2	3	0	1
1	2	3	4	5	6 (= k)

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl mit \leq Wert ($i = 2$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	0	2	3	0	1		
1	2	3	4	5	6 (=k)		

|

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl mit \leq Wert ($i = 2$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	2	3	0	1		
1	2	3	4	5	6 (=k)		

|

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl mit \leq Wert ($i = 3$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	4	3	0	1		
1	2	3	4	5	6 (=k)		

|

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl mit \leq Wert ($i = 4$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	4	7	0	1
1	2	3	4	5	6 (=k)

|

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl mit \leq Wert ($i = 5$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	4	7	7	1		
1	2	3	4	5	6 (=k)		

|

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Anzahl mit \leq Wert ($i = 6$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	4	7	7	8		
1	2	3	4	5	6 (= k)		

|

Ausgabefeld B

1	2	3	4	5	6	7	8

Ablauf – Ausgabe \rightarrow B ($j = 8$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	4	7	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

						4	
1	2	3	4	5	6	7	8

Ablauf – Ausgabe \rightarrow B ($j = 8$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	4	6	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

						4	
1	2	3	4	5	6	7	8

Ablauf – Ausgabe \rightarrow B ($j = 7$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

2	2	4	6	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

	1						4
1	2	3	4	5	6	7	8

Ablauf – Ausgabe \rightarrow B ($j = 7$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

1	2	4	6	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

	1						4
1	2	3	4	5	6	7	8

Ablauf – Ausgabe \rightarrow B ($j = 6$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

1	2	4	5	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

	1				4	4	
1	2	3	4	5	6	7	8

Ablauf – Ausgabe \rightarrow B ($j = 5$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

1	2	3	5	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

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

Ablauf – Ausgabe \rightarrow B ($j = 4$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

0	2	3	5	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

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

Ablauf – Ausgabe \rightarrow B ($j = 3$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

0	2	3	4	7	8		
1	2	3	4	5	6 (=k)		

Ausgabefeld B

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

Ablauf – Ausgabe \rightarrow B ($j = 2$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

$k = 6$

Hilfsfeld C

0	2	3	4	7	7
1	2	3	4	5	6 (=k)

Ausgabefeld B

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

Ablauf – Ausgabe \rightarrow B ($j = 1$)

Counting-Sort(A, B, k)

for $i = 1$ to k do $C[i] = 0$;

for $j = 1$ to $\text{length}[A]$ do $C[A[j]] = C[A[j]] + 1$;

for $i = 2$ to k do $C[i] = C[i] + C[i-1]$;

for $j = \text{length}[A]$ downto 1 do

$B[C[A[j]]] = A[j]$;

$C[A[j]] = C[A[j]] - 1$;

Eingabefeld A

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

|

$k = 6$

Hilfsfeld C

0	2	2	4	7	7		
1	2	3	4	5	6 (=k)		

|

Ausgabefeld B

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

Eigenschaften - Zeit

Counting-Sort(A, B, k)

n = length[A] Anzahl Elemente

for i = 1 to k do C[i] = 0;

O(k)

for j = 1 to length[A] do
 C[A[j]] = C[A[j]] + 1;

O(n)

for i = 2 to k do C[i] = C[i] + C[i-1];

O(k)

for j = length[A] downto 1 do
 B[C[A[j]]] = A[j];
 C[A[j]] = C[A[j]] - 1;

O(n)

→ O(k+n)

Eigenschaften - Zeit

- Fall: $k = O(n)$ (Normalfall)
- Laufzeit: **$O(n)$ linear**

Eigenschaften - Zeit

- Fall: $k = O(n)$ (Normalfall)
- Laufzeit: $O(n)$ **linear**

- Counting Sort schlägt $O(n \log n)$!
- Zählen statt Vergleichen

- Preis:
 - Einschränkung beim Wertebereich
 - speicherintensiv

Weitere Eigenschaften

Eingabefeld A

3	6	4	1	3	4	1	4
---	---	---	---	---	---	---	---

1 2 3 4 5 6 7 8

Ausgabefeld B

1	1	3	3	4	4	4	6
---	---	---	---	---	---	---	---

1 2 3 4 5 6 7 8

Weitere Eigenschaften

Eingabefeld A

3	6	4	1	3	4	1	4
---	---	---	---	---	---	---	---

1 2 3 4 5 6 7 8

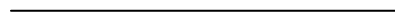
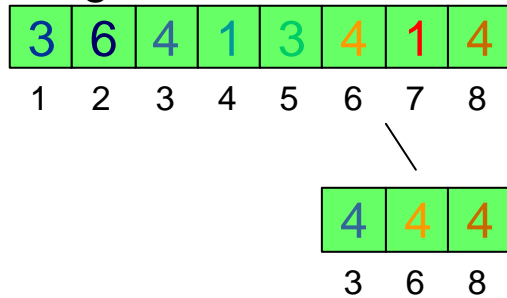
Ausgabefeld B

1	1	3	3	4	4	4	6
---	---	---	---	---	---	---	---

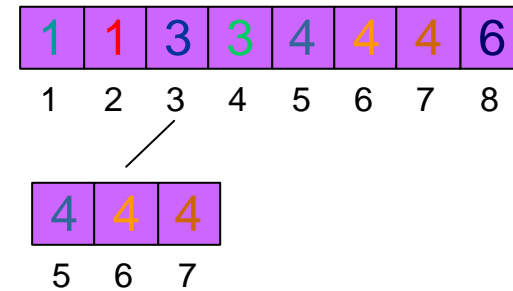
1 2 3 4 5 6 7 8

Weitere Eigenschaften

Eingabefeld A

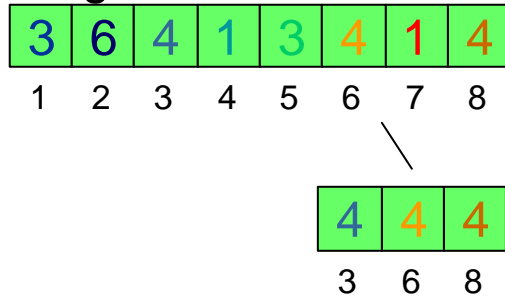


Ausgabefeld B

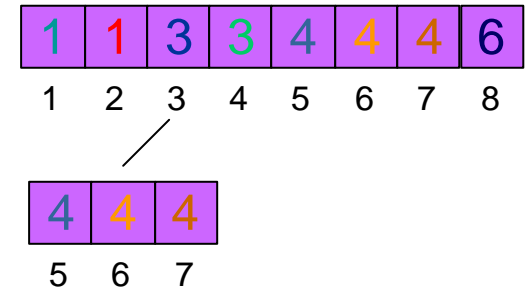


Weitere Eigenschaften

Eingabefeld A



Ausgabefeld B



- Counting Sort sortiert "stabil"
- Zahlen mit gleichem Wert:
 - Reihenfolge in A = Reihenfolge in B

Weitere Eigenschaften

Eingabefeld A

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

Ausgabefeld B

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

- Counting Sort sortiert "stabil"
- Zahlen mit gleichem Wert:
 - Reihenfolge in A = Reihenfolge in B
- Wichtig:
 - Nur, wenn weitere Daten mit $A[i]$ verbunden sind
 - Grundlage für Radix Sort

Beweis: Counting Sort - "stabil"

Indirekter Beweis:

- Algorithmus nicht stabil

Beweis: Counting Sort - "stabil"

Indirekter Beweis:

- Algorithmus nicht stabil
- O.B.d.A.: Wert $x = a = b$ komme an Pos. i und j vor

Beweis: Counting Sort - "stabil"

Indirekter Beweis:

- Algorithmus nicht stabil
- O.B.d.A.: Wert $x = a = b$ komme an Pos. i und j vor
- $a = A[i] \wedge b = A[j] \wedge i < j \rightarrow a = B[n] \wedge b = B[m] \wedge m > n$

Beweis: Counting Sort - "stabil"

Indirekter Beweis:

- Algorithmus nicht stabil
- O.B.d.A.: Wert $x = a = b$ komme an Pos. i und j vor
- $a = A[i] \wedge b = A[j] \wedge i < j \rightarrow a = B[n] \wedge b = B[m] \wedge m > n$
- Abarbeitung von hinten: $A[j]$ vor $A[i]$

Beweis: Counting Sort - "stabil"

Indirekter Beweis:

- Algorithmus nicht stabil
- O.B.d.A.: Wert $x = a = b$ komme an Pos. i und j vor
- $a = A[i] \wedge b = A[j] \wedge i < j \rightarrow a = B[n] \wedge b = B[m] \wedge m > n$

- Abarbeitung von hinten: $A[j]$ vor $A[i]$
- $P_j := C[A[j]] \rightarrow B[P_j] := b$
 $C[P_j] := C[P_j] - 1$

Beweis: Counting Sort - "stabil"

Indirekter Beweis:

- Algorithmus nicht stabil
- O.B.d.A.: Wert $x = a = b$ komme an Pos. i und j vor
- $a = A[i] \wedge b = A[j] \wedge i < j \rightarrow a = B[n] \wedge b = B[m] \wedge m > n$

- Abarbeitung von hinten: $A[j]$ vor $A[i]$
- $P_j := C[A[j]] \rightarrow B[P_j] := b$
 $C[P_j] := C[P_j] - 1$

- $P_i := C[A[i]] = P_j - 1$
- $P_j - 1 < P_j \rightarrow a$ vor $b \rightarrow n > m$

Beweis: Counting Sort - "stabil"

Indirekter Beweis:

- Algorithmus nicht stabil
- O.B.d.A.: Wert $x = a = b$ komme an Pos. i und j vor
- $a = A[i] \wedge b = A[j] \wedge i < j \rightarrow a = B[n] \wedge b = B[m] \wedge m > n$
- Abarbeitung von hinten: $A[j]$ vor $A[i]$
- $P_j := C[A[j]] \rightarrow B[P_j] := b$
 $C[P_j] := C[P_j] - 1$
- $P_i := C[A[j]] = P_j - 1$
- $P_j - 1 < P_j \rightarrow a$ vor $b \rightarrow n > m$

Widerspruch

Teil 2 –
Sortieren, Sortieren, Sortieren
Radix Sort

Was ist Radix - Sort?

- Grundidee:
 - Betrachten lexikografische Ordnungen
 - Analog für Dezimalzahlen (führende Nullen)

Was ist Radix - Sort?

- Grundidee:
 - Betrachten lexikografische Ordnungen
 - Analog für Dezimalzahlen (führende Nullen)
- Verfahren:
 - Zerlegen Gesamtschlüssel in **priorisierte Teilschlüssel**
 - intuitiv: Start mit **höchster** Priorität

Was ist Radix - Sort?

- Grundidee:
 - Betrachten lexikografische Ordnungen
 - Analog für Dezimalzahlen (führende Nullen)
- Verfahren:
 - Zerlegen Gesamtschlüssel in **priorisierte Teilschlüssel**
 - intuitiv: Start mit **höchster** Priorität
- Beispiel:
 - Sortieren von Datumsangaben **TT.MM.JJJJ**
 - Priorität: **JJJJ**, dann **MM** als letztes **TT**

Was ist Radix - Sort?

- Grundidee:
 - Betrachten lexikografische Ordnungen
 - Analog für Dezimalzahlen (führende Nullen)
- Verfahren:
 - Zerlegen Gesamtschlüssel in **priorisierte Teilschlüssel**
 - intuitiv: Start mit **höchster** Priorität
- Beispiel:
 - Sortieren von Datumsangaben **TT.MM.JJJJ**
 - Priorität: **JJJJ**, dann **MM** als letztes **TT**
- Radix Sort:
 - Zerlegt Gesamtschlüssel in Teilschlüssel → **stabil** sortieren
 - ABER: Start mit **niedrigster** Priorität

Beispiel: Zahlen sortieren



- **Eingabe:** Feld A mit Integer Zahlen ≥ 1
- **Ziel:** 10 stellige Zahlen sortieren

Beispiel: Zahlen sortieren



- **Eingabe:** Feld A mit Integer Zahlen ≥ 1
- **Ziel:** 10 stellige Zahlen sortieren
- **Verfahren:**
 - Zerlegung in Dezimalstellen = 10 Teilschlüssel (d = 10)
 - Zahlen nach Teilschlüsseln stabil sortieren

Beispiel: Zahlen sortieren



- **Eingabe:** Feld A mit Integer Zahlen ≥ 1
- **Ziel:** 10 stellige Zahlen sortieren
- **Verfahren:**
 - Zerlegung in 10 Teilschlüssel ($d = 10$)
 - Zahlen nach Teilschlüsseln sortieren

Radix-Sort (A, d)

for $i = 1$ **to** d **do** *stable sort* A für Dezimalstelle d

Beispiel: Zahlen sortieren



- **Eingabe:** Feld A mit Integer Zahlen ≥ 1
- **Ziel:** 10 stellige Zahlen sortieren
- **Verfahren:**
 - Zerlegung in 10 Teilschlüssel ($d = 10$)
 - Zahlen nach Teilschlüsseln sortieren

Radix-Sort (A, d)

for $i = 1$ **to** d **do** *stable sort* A für Dezimalstelle d

- Laufzeit:
 - $O(d n) \rightarrow d = \text{konst.} \rightarrow O(n)$ linear

Beispiel: Zahlen sortieren



- **Eingabe: 5 000 000 000** 10 stellige Zahlen
- **Aufwandsvergleich:**
 - Quicksort:
 - $5\,000\,000\,000 * \log 5\,000\,000\,000 \gg 5\,000\,000\,000 * 32$
 - Radixsort:
 - $5\,000\,000\,000 * 10$
 - Differenz:
 - $5\,000\,000\,000 * 32 - 5\,000\,000\,000 * 10 = 110000\,000\,000$

Beispiel: Zahlen sortieren



- **Eingabe:** 5 000 000 000 10 stellige Zahlen
- **Aufwandsvergleich:**
 - Quicksort:
 - $5\,000\,000\,000 * \log 5\,000\,000\,000 \gg 5\,000\,000\,000 * 32$
 - Radixsort:
 - $5\,000\,000\,000 * 10$
 - Differenz:
 - $5\,000\,000\,000 * 32 - 5\,000\,000\,000 * 10 = 110\,000\,000\,000$
- **Realität:**
 - 1 GHz Prozessor: pro Operation 10^{-9} s
 - Quicksort: 160s Radixsort: 50s
 - Zeitersparnis: 110s

Radix Sort - Beispiel

329

457

657

839

436

720

355

Radix Sort - Beispiel

329

457

657

839

436

720

355

720

355

436

457

657

329

839

Radix Sort - Beispiel

329

457

657

839

436

720

355

720

355

436

457

657

329

839

720

329

436

839

355

457

657

Radix Sort - Beispiel

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Korrektheit

- Nachweis: Induktion über die Zifferanzahl (= d)
- Induktionsanfang ($d = 1$):
 - Ziffernsortierung = Zahlensortierung

Korrektheit

- Nachweis: Induktion über die Zifferanzahl (= d)
- Induktionsanfang ($d = 1$):
 - Ziffernsortierung = Zahlensortierung
- Induktionsschritt:
 - Induktionsvoraussetzung:
 - Sortierung von Zahlen mit d Ziffern =
Radix-Sort für $d-1$ Ziffern + Sortierung nach Ziffer d

Korrektheit

- Nachweis: Induktion über die Zifferanzahl (= d)
- Induktionsanfang ($d = 1$):
 - Ziffernsortierung = Zahlensortierung
- Induktionsschritt:
 - Induktionsvoraussetzung:
 - Sortierung von Zahlen mit d Ziffern =
Radix-Sort für $d-1$ Ziffern + Sortierung nach Ziffer d
 - Induktionsbehauptung:
 - Radix-Sort für d Ziffern \rightarrow Sortierung der Zahlen

Korrektheit

- Nachweis: Induktion über die Zifferanzahl (= d)
- Induktionsanfang ($d = 1$):
 - Ziffernsortierung = Zahlensortierung
- Induktionsschritt:
 - Induktionsvoraussetzung:
 - Sortierung von Zahlen mit d Ziffern =
Radix-Sort für $d-1$ Ziffern + Sortierung nach Ziffer d
 - Induktionsbehauptung:
 - Radix-Sort für d Ziffern \rightarrow Sortierung der Zahlen
 - O.B.d.A.: Zahlen a und b mit Ziffern a_d und b_d

Korrektheit

- Nachweis: Induktion über die Zifferanzahl (= d)
- Induktionsanfang ($d = 1$):
 - Ziffernsortierung = Zahlensortierung
- Induktionsschritt:
 - Induktionsvoraussetzung:
 - Sortierung von Zahlen mit d Ziffern = Radix-Sort für $d-1$ Ziffern + Sortierung nach Ziffer d
 - Induktionsbehauptung:
 - Radix-Sort für d Ziffern \rightarrow Sortierung der Zahlen
 - O.B.d.A.: Zahlen a und b mit Ziffern a_d und b_d
 - **Fall 1:** $a_d > b_d \rightarrow a > b$ unabhängig von a_i und b_i ($i = 1, \dots, d-1$)

Korrektheit

- Nachweis: Induktion über die Zifferanzahl (= d)
- Induktionsanfang ($d = 1$):
 - Ziffernsortierung = Zahlensortierung
- Induktionsschritt:
 - Induktionsvoraussetzung:
 - Sortierung von Zahlen mit d Ziffern =
Radix-Sort für $d-1$ Ziffern + Sortierung nach Ziffer d
 - Induktionsbehauptung:
 - Radix-Sort für d Ziffern \rightarrow Sortierung der Zahlen
 - O.B.d.A.: Zahlen a und b mit Ziffern a_d und b_d
 - **Fall 1:** $a_d > b_d \rightarrow a > b$ unabhängig von a_i und b_i ($i = 1, \dots, d-1$)
 - **Fall 2:** $a_d < b_d \rightarrow a < b$ unabhängig von a_i und b_i ($i = 1, \dots, d-1$)

Korrektheit

- Nachweis: Induktion über die Zifferanzahl (= d)
- Induktionsanfang ($d = 1$):
 - Ziffernsortierung = Zahlensortierung
- Induktionsschritt:
 - Induktionsvoraussetzung:
 - Sortierung von Zahlen mit d Ziffern = Radix-Sort für $d-1$ Ziffern + Sortierung nach Ziffer d
 - Induktionsbehauptung:
 - Radix-Sort für d Ziffern \rightarrow Sortierung der Zahlen
 - O.B.d.A.: Zahlen a und b mit Ziffern a_d und b_d
 - **Fall 1:** $a_d > b_d \rightarrow a > b$ unabhängig von a_i und b_i ($i = 1, \dots, d-1$)
 - **Fall 2:** $a_d < b_d \rightarrow a < b$ unabhängig von a_i und b_i ($i = 1, \dots, d-1$)
 - **Fall 3:** $a_d = b_d$
 - Sortierung für Ziffern a_i und b_i ($i = 1, \dots, d-1$) lt. IV erfolgt
 - Sortierung bleibt erhalten, da a_d und b_d stabil sortiert werden

Invariante

Radix-Sort (A , d)

for $i = 1$ **to** d **do** *stable sort* A für Dezimalstelle d

- Feld A für alle Teilschlüssel $k < i$ stabil sortiert

Terminierung

Radix-Sort (A, d)

for $i = 1$ **to** d **do** *stable sort* A für Dezimalstelle d

- **Notwendig:** *stable sort* terminiert
 - Counting Sort terminiert, da "nur" For-Schleifen
- **Hinreichend:** Terminierungsfunktion
 - $f(i, d) = d - n \rightarrow f(i, d) = 0 \bullet i = d$

Zusammenfassung

- Sortieren mit Vergleichen → untere Schranke
- Wertebereich beschränken → Sortieren ohne Vergleichen
- Counting Sort → linearer Aufwand
- "stabile" Sortierverfahren

- Radix Sort nutzt stabile Sortierverfahren
- sortiert Teilschlüssel einzeln
- niedrigste Priorität zuerst