

5.7 Sortieren in linearer Zeit

(48)

In Satz 5.4 haben wir gezeigt, dass das Sortieren von n Objekten nicht schneller als in $\Omega(n \log n)$ möglich ist - vorausgesetzt, man kann dafür keine Zusatzinformation einsetzen und darf Objekte nur paarweise vergleichen (und ggf. vertauschen).

In vielen Situationen hat man aber zusätzliche Informationen! Das wollen wir hier analysieren.

5.7.1 Countingsort

Im einfachsten Fall sind die n Objekte Zahlen aus dem Bereich $1, \dots, k$.

Wenn nun k klein ist (insbesondere $k \in O(n)$), dann gibt es einen einfachen Ansatz, der ganz ohne Vergleiche auskommt:

- (1) lege einen "Zählarray" $C[0], \dots, C[k]$ an
- (2) Gehe den zu sortierenden Array $A[1], \dots, A[n]$ durch; für jedes Objekt $A[j]$ erhöhe den entsprechenden Eintrag in C , d.h. $C[A[j]]$ um 1.
- (3)

Für $j = 1, \dots, k$:
Schreibe jeweils j in die nächsten $C[A[j]]$ Einträge von $B[i]$.
Am Ende liefert $B[1], \dots, B[n]$ eine sortierte Form von $A[1], \dots, A[n]$.

Im Detail:

ALGORITHMUS 5.14

INPUT : Array $A[1], \dots, A[n]$ mit Schlüsselwerten $\in \{1, \dots, k\}$
OUTPUT : Sortierte Kopie $B[1], \dots, B[n]$ von $A[1], \dots, A[n]$

COUNTING-SORT (A, B, k)

1. FOR ($i=1$) TO k
1.1 DO $C[i] := 0$ // Zählerarray initialisieren
2. FOR ($j=1$) TO n
2.1 DO $C[A[j]] := C[A[j]] + 1$ // Anzahl Elemente mit Wert $A[j]$ um 1 erhöhen
3. FOR ($i=1$) TO k
3.1 DO $C[i] := C[i] + C[i-1]$ // Anzahl Elemente mit Wert höchstens i zählen
4. FOR ($j=n$) DOWNTO 1
4.1 DO $B[C[A[j]]] := A[j]$ // Element $A[j]$ an richtige
4.2 $C[A[j]] := C[A[j]] - 1$ // Stelle in B schreiben.

SATZ 5.15

Für n Objekte mit Schlüssel $\in \{1, \dots, k\}$ benötigt COUNTING-SORT die Zeit $O(n+k)$. Insbesondere benötigt man für $k \in O(n)$ die Zeit $\Theta(n)$.

5.7.2 Radixsort

Idee: Sortiere nicht gleich nach den Schlüsselwerten, sondern nach den Ziffern der Schlüssel!

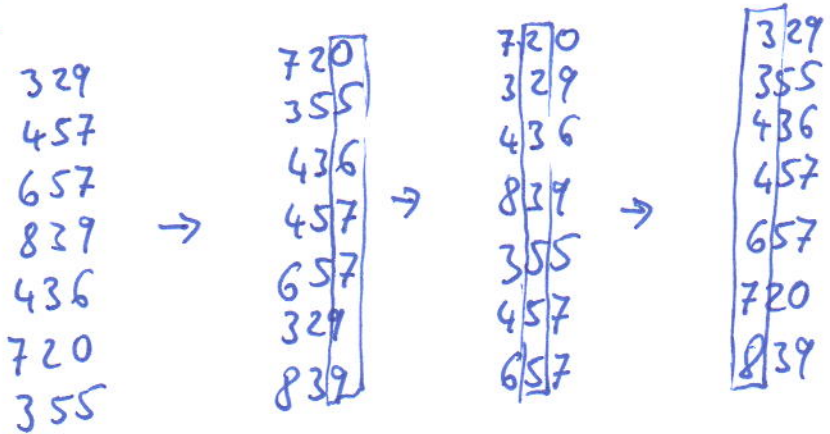
(Früher sehr häufig bei Lochkarten eingesetzt.)

Naiver Ansatz: Sortiere nach größter Ziffer, dann nach zweiter Ziffer, ...

Problem: Entweder benötigt man viele temporäre Zwischenmengen - oder die Vorsortierung geht kaputt!

Besserer Ansatz: Sortiere nach ... letzter Ziffer, dann nach vorletzter Ziffer, ...

Beispiel:



Wichtig: Jeweilige Sortierung darf Reihenfolge gleichwertiger Ziffern nicht verändern!

DEFINITION 5.16 (Stabilität)

Ein Sortierverfahren heißt stabil wenn gleiche Werte nach dem Algorithmsdurchlauf in der gleichen Reihenfolge sind wie vorher.

SATZ 5.17 (Stabilität)

Mergesort und Countingsort sind stabil, Quicksort je nach Umsetzung der Pivotregel.

Beweis: Selbst!

Damit:

ALGORITHMUS 5.18

INPUT: n Zahlen mit je d Ziffern, die k verschiedene Werte annehmen können, $A[1], \dots, A[n]$

OUTPUT: Sortierter Array

RADIX-SORT (A, d)

1 FOR ($i=1$) TO d

1.1 DO Countingsort für A auf Stelle i

SATZ 5.19

Radixsort liefert ein korrektes Ergebnis in Zeit $\Theta(d(n+k))$.

Beweis:

Für je zwei Zahlen muss am Ende die Reihenfolge stimmen; diese entspricht der Ordnung der höchsten Ziffern, die verschieden sind. Die Sortierung nach diesen Ziffern liefert also die richtige Reihenfolge, die sich wegen der Stabilität nicht mehr ändert.

Die Laufzeit folgt aus Satz 5.15.

□