10. Sortieren III

Untere Schranken für das vergleichsbasierte Sortieren, Radix- und Bucketsort

10.1 Untere Grenzen für Vergleichbasiertes Sortieren

[Ottman/Widmayer, Kap. 2.8, Cormen et al, Kap. 8.1]

Untere Schranke für das Sortieren

Bis hierher: Sortieren im schlechtesten Fall benötigt $\Omega(n \log n)$ Schritte.

Geht es besser? Nein:

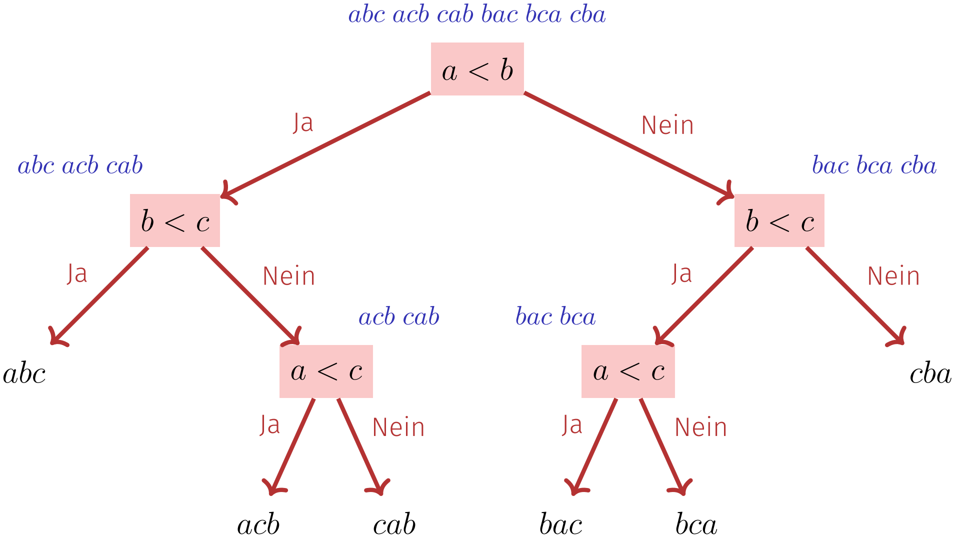
Theorem 15

Vergleichsbasierte Sortierverfahren benötigen im schlechtesten Fall und im Mittel mindestens $\Omega(n \log n)$ Schlüsselvergleiche.

Vergleichsbasiertes Sortieren

- Algorithmus muss unter $n!$ vielen Anordnungsmöglichkeiten einer Folge $(A_i)_{i=1,\dots,n}$ die richtige identifizieren.
- Zu Beginn weiss der Algorithmus nichts.
- Betrachten den "Wissensgewinn" des Algorithmus als Entscheidungsbaum:
  - Knoten enthalten verbleibende Möglichkeiten
  - Kanten enthalten Entscheidungen

Entscheidungsbaum



Entscheidungsbaum

Ein binärer Baum mit L Blättern hat $K = L - 1$ innere Knoten.[11]

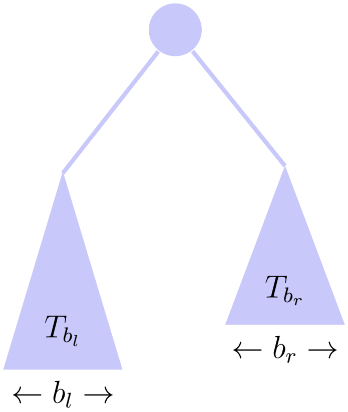
Die Höhe eines binären Baumes mit L Blättern ist mindestens $\log_2 L$. $\Rightarrow$ Höhe des Entscheidungsbaumes $h \geq \log n! \in \Omega(n \log n)$.

Somit auch die Länge des längsten Pfades im Entscheidungsbaum $\in \Omega(n \log n)$.

Bleibt zu zeigen: mittlere Länge $M(n)$ eines Pfades $M(n) \in \Omega(n \log n)$.

[11]Beweis: starte mit leerem Baumm, $K = 0$, $L = 1$. Jeder hinzugefügte Knoten ersetzt ein Blatt durch 2 Blätter. Also.

Untere Schranke im Mittel



- Entscheidungsbaum T_n mit n Blättern, mittlere Tiefe eines Blatts $m(T_n)$
- Annahme: $m(T_n) \geq \log n$ nicht für alle n.
- Wähle kleinstes b mit $m(T_b) < \log b \Rightarrow b \geq 2$
- $b_l + b_r = b$ mit $b_l > 0$ und $b_r > 0 \Rightarrow$ $b_l < b, b_r < b \Rightarrow m(T_{b_l}) \geq \log b_l$ und $m(T_{b_r}) \geq \log b_r$

Untere Schranke im Mittel

Mittlere Tiefe eines Blatts:

$$\begin{aligned} m(T_b) &= \frac{b_l}{b}(m(T_{b_l})+1) + \frac{b_r}{b}(m(T_{b_r})+1) \\ &\geq \frac{1}{b}(b_l(\log b_l + 1) + b_r(\log b_r + 1)) = \frac{1}{b}(b_l \log 2b_l + b_r \log 2b_r) \\ &\geq \frac{1}{b}(b \log b) = \log b. \end{aligned}$$

Widerspruch. ■

Die letzte Ungleichung gilt, da $f(x) = x \log x$ konvex ist $\left(f''(x) = 1/x > 0\right)$ und für eine konvexe Funktion gilt $f((x+y)/2) \leq 1/2 f(x) + 1/2 f(y)$ $\left(x = 2b_l,\ y = 2b_r\right.$ einsetzen).[12] Einsetzen von $x = 2b_l$, $y = 2b_r$, und $b_l + b_r = b$.

---

[12]allgemein $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$ für $0 \leq \lambda \leq 1$.

10.2 Radixsort und Bucketsort

Radixsort, Bucketsort [Ottman/Widmayer, Kap. 2.5, Cormen et al, Kap. 8.3]

Radix Sort

Vergleichsbasierte Sortierverfahren: Schlüssel vergleichbar ($<$ oder $>$, $=$). Ansonsten keine Voraussetzung.

Andere Idee: nutze mehr Information über die Zusammensetzung der Schlüssel.

Annahmen

Annahme: Schlüssel darstellbar als Wörter aus einem Alphabet mit m Elementen.

Beispiele

$m = 10$	Dezimalzahlen	$183 = 183_{10}$
$m = 2$	Dualzahlen	101_2
$m = 16$	Hexadezimalzahlen	$A0_{16}$
$m = 26$	Wörter	`"INFORMATIK"`

m heisst die Wurzel (lateinisch *Radix) der Darstellung.*

Annahmen

- Schlüssel =m-adische Zahlen mit gleicher Länge.
- Verfahren z zur Extraktion der k-ten Ziffer eines Schlüssels in $\mathcal{O}(1)$ Schritten.

Beispiel

$$z_{10}(0, 85) = 5$$
$$z_{10}(1, 85) = 8$$
$$z_{10}(2, 85) = 0$$

Radix-Exchange-Sort

Schlüssel mit Radix 2.
Beobachtung: Wenn für ein $k \geq 0$:

$$z_2(i, x) = z_2(i, y) \text{ für alle } i > k$$

und

$$z_2(k, x) < z_2(k, y),$$

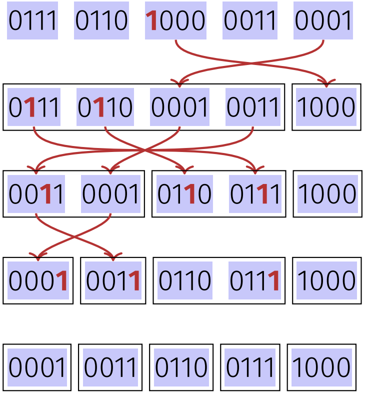
dann ist $x < y$.

Radix-Exchange-Sort

Idee:

- Starte mit maximalem k.
- Binäres Aufteilen der Datensätze mit $z_2(k, \cdot) = 0$ vs. $z_2(k, \cdot) = 1$ wie bei Quicksort.
- $k \leftarrow k - 1$.

Radix-Exchange-Sort



Algorithmus RadixExchangeSort(A, l, r, b)

Input: Array A der Länge n, linke und rechte Grenze $1 \leq l \leq r \leq n$, Bitposition b

Output: Array A, im Bereich $[l, r]$ nach Bits $[0, \ldots, b]$ sortiert.

if $l < r$ **and** $b \geq 0$ **then**

 $i \leftarrow l - 1$

 $j \leftarrow r + 1$

 repeat

  repeat $i \leftarrow i + 1$ **until** $z_2(b, A[i]) = 1$ **or** $i \geq j$

  repeat $j \leftarrow j - 1$ **until** $z_2(b, A[j]) = 0$ **or** $i \geq j$

  if $i < j$ **then** swap($A[i], A[j]$)

 until $i \geq j$

 RadixExchangeSort($A, l, i - 1, b - 1$)

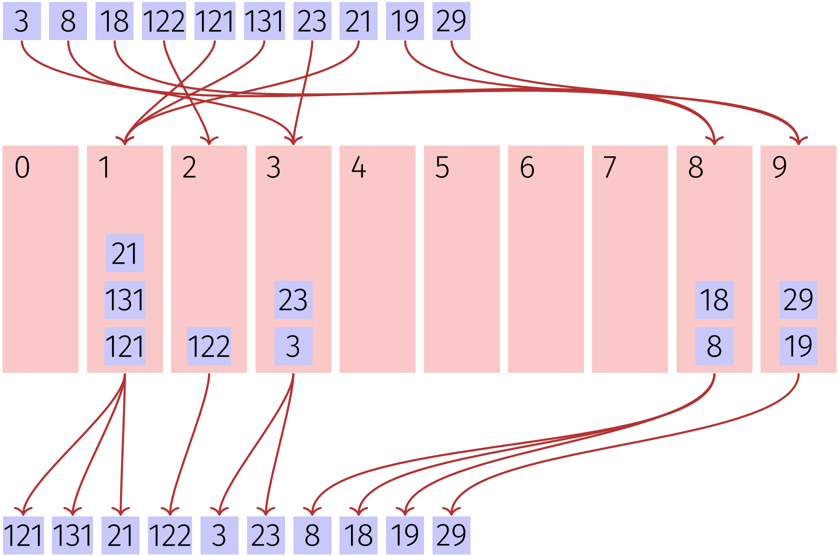
 RadixExchangeSort($A, i, r, b - 1$)

Analyse

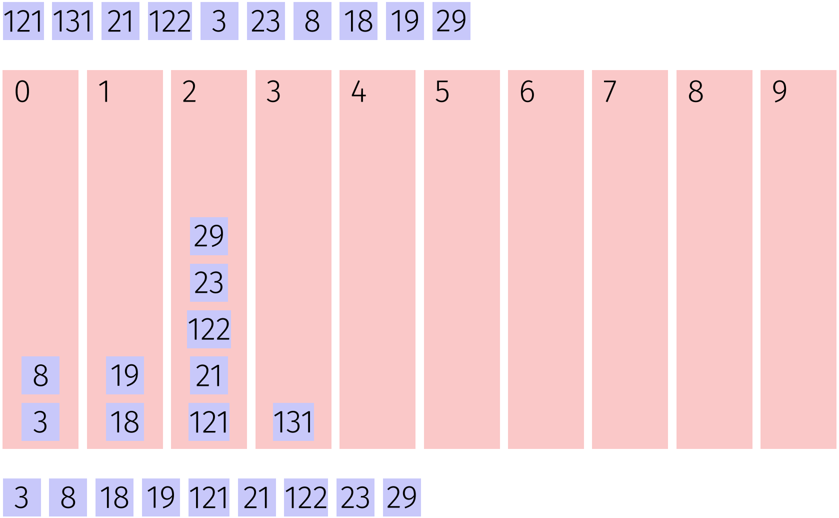
RadixExchangeSort ist rekursiv mit maximaler Rekursionstiefe = maximaler Anzahl Ziffern p.

Laufzeit im schlechtesten Fall $\mathcal{O}(p \cdot n)$.

Bucket Sort (Sortieren durch Fachverteilen)

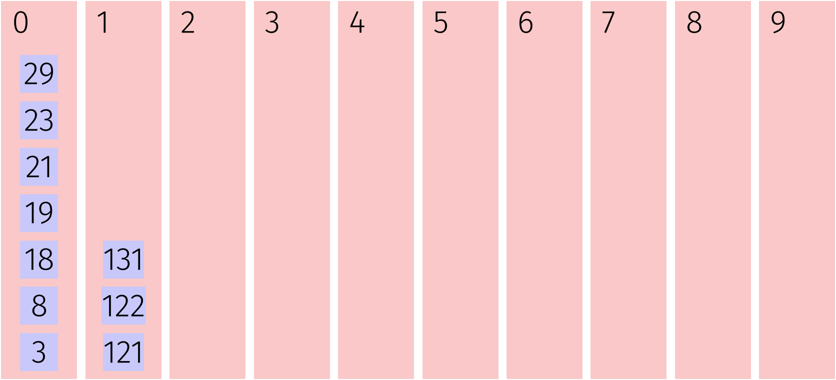


Bucket Sort (Sortieren durch Fachverteilen)



Bucket Sort (Sortieren durch Fachverteilen)

3 8 18 19 121 21 122 23 29



3 8 18 19 21 23 29 121 122 131

Implementationsdetails

Bucketgrösse sehr unterschiedlich. Möglichkeiten

- Verkettete Liste oder dynamisches Array für jede Ziffer.
- Ein Array der Länge n, Offsets für jede Ziffer in erstem Durchlauf bestimmen.

Annahmen: Eingabelänge n , Anzahl Bits / Ganzzahl: k , Anzahl Buckets: 2^b
Asymptotische Laufzeit $\mathcal{O}(\frac{k}{b} \cdot (n + 2^b)$.
Zum Beispiel: $k = 32$, $2^b = 256 : \frac{k}{b} \cdot (n + 2^b) = 4n + 1024$.

Bucket Sort – Andere Voraussetzung

Annahme: gleichmässig verteilte Daten, z.B. aus $[0,1)$

Input: Array A der Länge n, $A_i \in [0,1)$, Konstante $M \in \mathbb{N}^+$
Output: Sortiertes Array

$k \leftarrow \lceil n/M \rceil$
$B \leftarrow$ new array of k empty lists
for $i \leftarrow 1$ **to** n **do**
    $B[\lfloor A_i \cdot k \rfloor]$.append$(A[i])$
for $i \leftarrow 1$ **to** k **do**
    sort $B[i]$ // z.B. insertion sort, mit Laufzeit $\mathcal{O}(M^2)$
return $B[0] \circ B[1] \circ \cdots \circ B[k]$ // konkateniert

Erwartete asymptotische Laufzeit $\mathcal{O}(n)$ (Beweis in Cormen et al, Kap. 8.4)