Datenstrukturen & Algorithmen

Peppo Brambilla
Universität Bern
Frühling 2018

Übersicht

- Untere Schranken für Sortieren
- Sortieren mit linearem Aufwand
- Mediane und Ranggrössen

Wie schnell können wir sortieren?

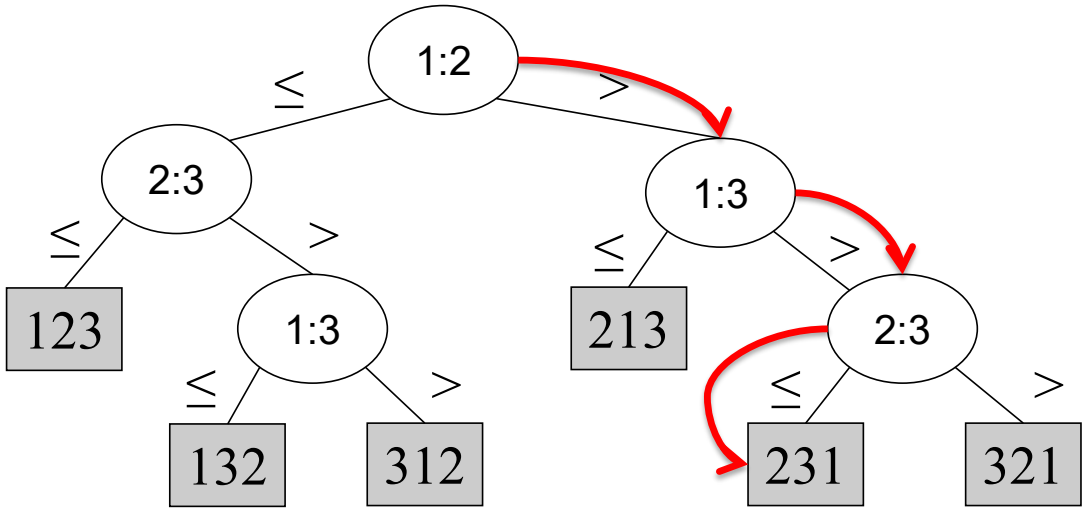
- Bis jetzt: sortieren durch vergleichen
  - Verwenden ausschliesslich paarweise Vergleiche zur Bestimmung der Reihenfolge
  - Beispiele: sortieren durch Einfügen, Mischen, Heapsort, Quicksort
- Kein Algorithmus hat worst-case Laufzeit besser als $O(n \lg n)$
- Gibt es schnellere Algorithmen?
  - Worst-case Laufzeit schneller als $O(n \lg n)$

Entscheidungsbäume

- Beschreibt Ausführung beliebiger Verfahren zum Sortieren durch Vergleiche von Elementen
- Jeder (deterministische) Algorithmus hat einen immer gleichen Baum für jede Eingabelänge n
- Sortieren: traversieren des Baums entlang eines Ausführungspfades von Wurzel zu Blatt
- Ausführung nimmt einen von zwei möglichen Pfaden bei jedem Vergleich
- Baum enthält Vergleiche entlang allen möglichen Ausführungspfaden
- Laufzeit = Länge des Ausführungspfades
- Worst-case Laufzeit = Höhe des Baumes

Beispiel (Insertion Sort)

- Sortiere $\langle a_1, a_2, a_3 \rangle = \langle 9, 4, 6 \rangle$



$\langle a_2, a_3, a_1 \rangle = \langle 4, 6, 9 \rangle$

- $i{:}\,j$ bedeutet Vergleich zwischen a_i und a_j
- Linker Teilbaum falls $a_i \leq a_j$
- Rechter Teilbaum falls $a_i > a_j$

Untere Schranke für Sortieren durch Vergleichen

- Theorem: Jeder Entscheidungsbaum, der n Elemente sortiert, muss mindestens Höhe $\Omega(n \lg n)$ haben (untere Schranke!)
- Beweis: Baum hat $\geq n!$ Blätter, weil es $n!$ Permutationen gibt. Binärbaum der Höhe h hat $\leq 2^h$ Blätter. Also muss $2^h \geq n!$ sein.

$$
\begin{aligned}
h &\geq \lg(n!) \\
&\geq \lg\big((n/e)^n\big) \quad \text{(Stirlingsche Näherung)} \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n)
\end{aligned}
$$

https://de.wikipedia.org/wiki/Stirlingformel

nicht in Prüfung

Untere Schranke für Sortieren durch Vergleichen

- Folgerungen
  - Jeder vergleichende Sortieralgorithmus braucht im schlechtesten Fall mindestens $\Omega(n \lg n)$ Vergleichsoperationen
  - Heapsort und Sortieren durch Mischen sind asymptotisch optimale vergleichende Sortieralgorithmen
  - Quicksort?

Übersicht

- Untere Schranken für Sortieren
- Sortieren mit linearem Aufwand
- Mediane und Ranggrössen

Sortieren mit linearem Aufwand

Countingsort: keine Vergleiche zwischen Elementen!

- Eingabe: $A[1 \ldots n]$, wobei $A[j] \in \{1, 2, \ldots, k\}$
  - Ganze Zahlen zwischen 1 und k
- Ausgabe: $B[1 \ldots n]$, sortiert
- Zwischenspeicher: $C[1 \ldots k]$

Countingsort

$$\textbf{for } i = 1 \textbf{ to } k$$
$$C[i] = 0$$

C initialisieren

$$\textbf{for } j = 1 \textbf{ to } n$$
$$C[A[j]] = C[A[j]] + 1$$

Vorkommnisse der Zahlen $1 \ldots k$ zählen

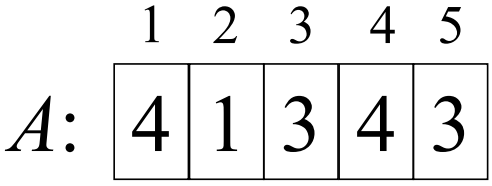
$$\textbf{for } i = 2 \textbf{ to } k$$
$$C[i] = C[i] + C[i-1]$$

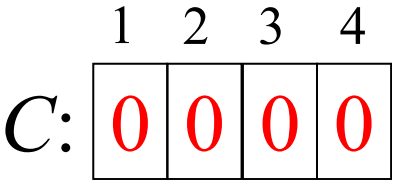
Berechne Anzahl der Zahlen $\leq i$

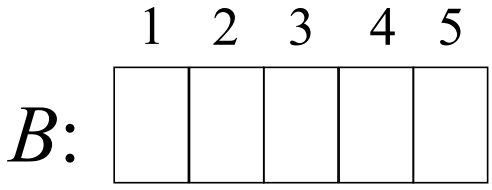
$$\textbf{for } j = n \textbf{ downto } 1$$
$$B[C[A[j]]] = A[j]$$
$$C[A[j]] = C[A[j]] - 1$$

Einordnen der Werte

Beispiel



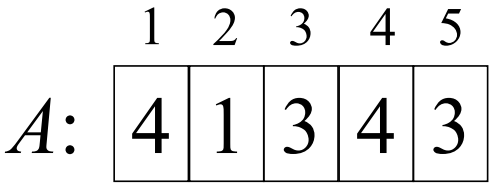


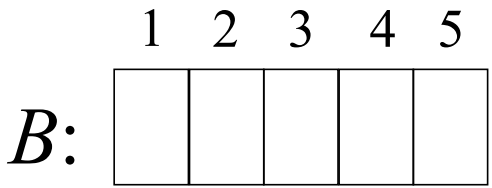


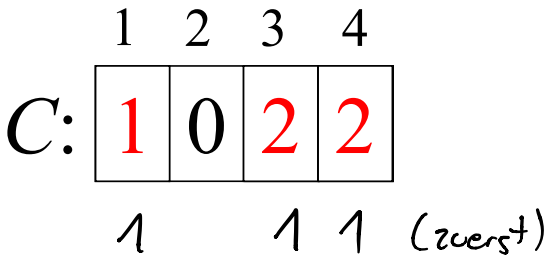
for $i = 1$ **to** k

$C[i] = 0$

Beispiel



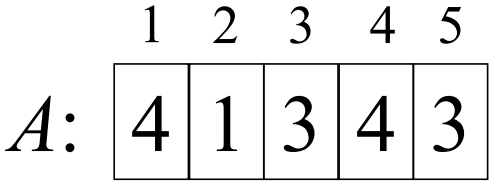


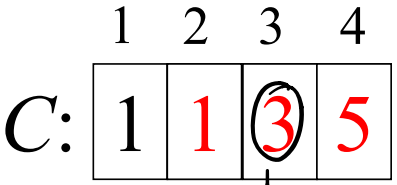


for $j = 1$ **to** n

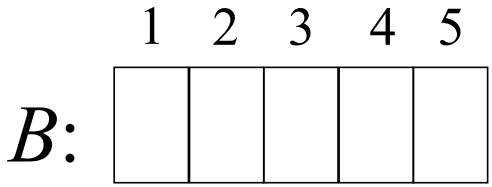
$$C[A[j]] = C[A[j]] + 1 \quad \triangleright\, C[i] = |\{\text{key} = i\}|$$

Beispiel





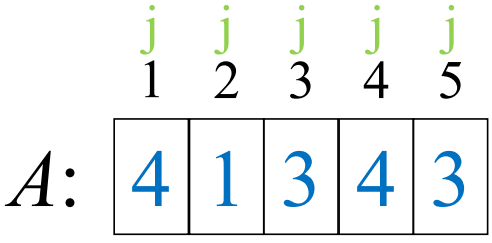
3 Zahlen, welche ≤ 3 sind

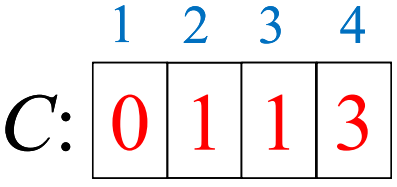


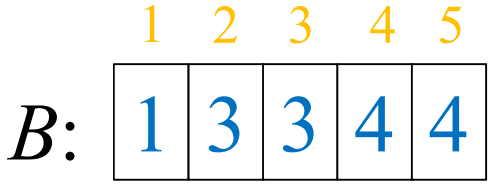
for $i = 2$ **to** k

$$C[i] = C[i] + C[i-1] \quad \triangleright\, C[i] = |\{\text{key} \leq i\}|$$

Beispiel







for $j = n$ **downto** 1

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

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

Analyse

$$\Theta(k) \quad \begin{cases} \textbf{for } i = 1 \textbf{ to } k \\ \quad C[i] = 0 \end{cases}$$

$$\Theta(n) \quad \begin{cases} \textbf{for } j = 1 \textbf{ to } n \\ \quad C[A[j]] = C[A[j]] + 1 \end{cases}$$

$$\Theta(k) \quad \begin{cases} \textbf{for } i = 2 \textbf{ to } k \\ \quad C[i] = C[i] + C[i-1] \end{cases}$$

$$\Theta(n) \quad \begin{cases} \textbf{for } j = n \textbf{ downto } 1 \\ \quad B[C[A[j]]] = A[j] \\ \quad C[A[j]] = C[A[j]] - 1 \end{cases}$$

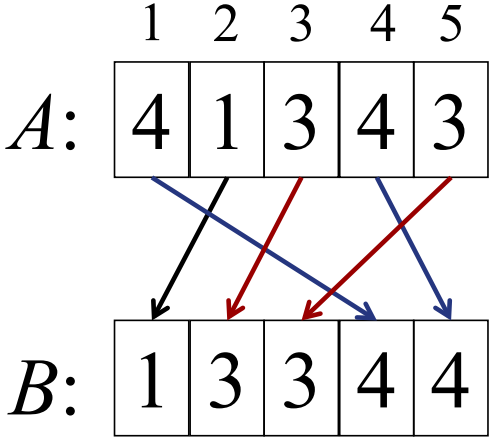
$$\Theta(n+k)$$

Laufzeit

- Laufzeit $\Theta(n + k)$
- Wenn $k = O(n)$, dann sogar $\Theta(n)$
- Besser als untere Schranke $\Omega(n \lg n)$?!
- Countingsort ist kein vergleichender Sortieralgorithmus!
  - Keine Vergleiche im gesamten Algorithmus
  - Nützt aus, dass Werte direkt verwendet werden können, um sortierte Position zu bestimmen

Stabiles Sortieren

- Countingsort ist stabil
  - Reihenfolge gleich grosser Elemente wird nicht verändert

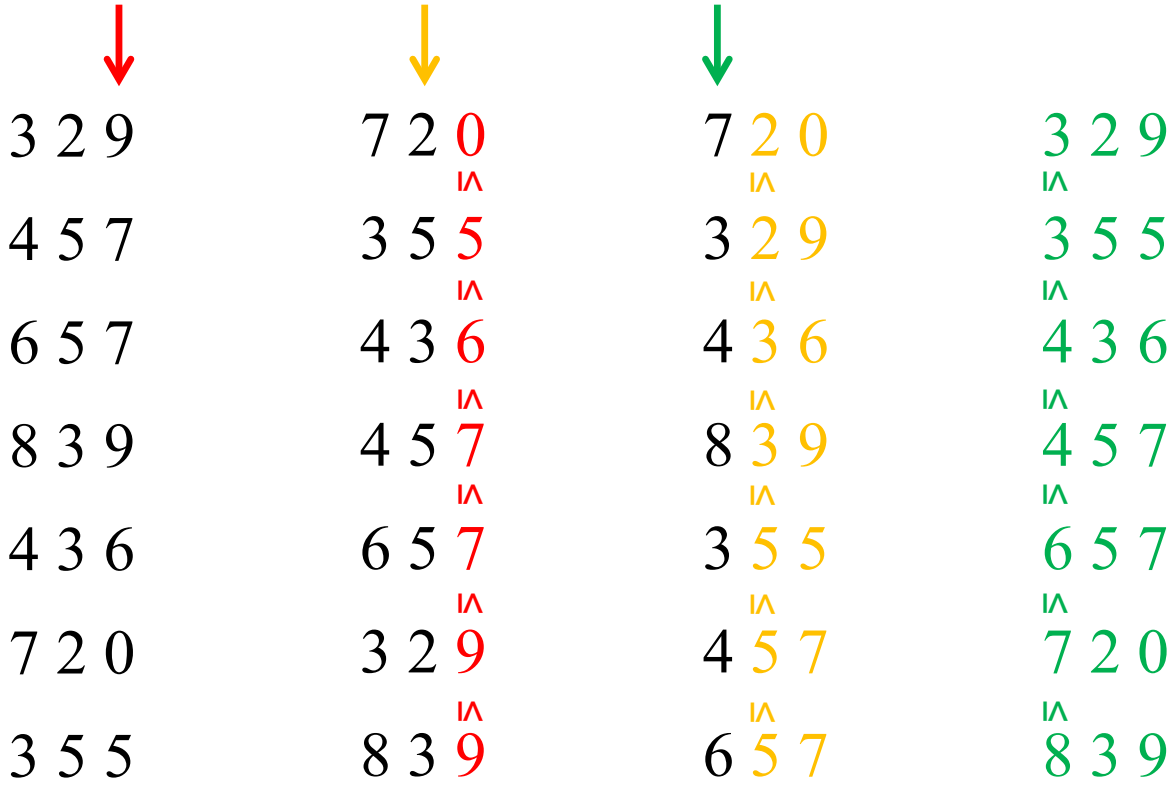


Radixsort

- Zahlen mit mehreren Ziffern: Ziffer um Ziffer sortieren
- Schlechte Idee: bei der höchstwertigen Ziffer anfangen
- Besser: bei der niederwertigsten Ziffer anfangen und eine stabile Sortiermethode verwenden

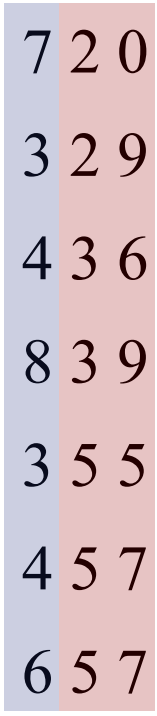
(heisst die relative Position bleibt gleich
Folie vorher)

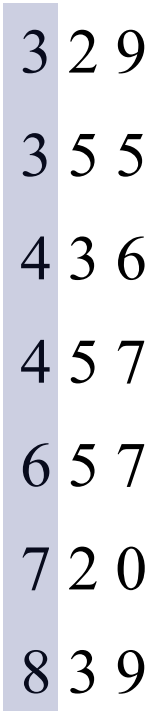
Beispiel


3 2 9
4 5 7
6 5 7
8 3 9
4 3 6
7 2 0
3 5 5
7 2 0
3 5 5
4 3 6
4 5 7
3 2 9
8 3 9
7 2 0
3 2 9
4 3 6
8 3 9
3 5 5
4 5 7
6 5 7
3 2 9
3 5 5
4 3 6
4 5 7
6 5 7
7 2 0
8 3 9

Korrektheit

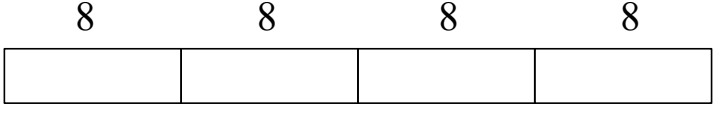
- Induktion über Position der Ziffern
- Annahme: Zahlen sind bis zu Ziffer $t-1$ sortiert
- Induktionsschritt:
  Sortiere Ziffer t
  - Zwei Zahlen, die sich in Ziffer t unterscheiden werden korrekt sortiert
  - Wegen Stabilität: Reihenfolge von Zahlen, die in Ziffer t gleich sind, wird nicht verändert
    → korrekte Reihenfolge





Sortieren von Binärzahlen

- Sortiere n Binärzahlen mit je b Bit
- Interpretiere Zahl als $^{b}/_{r}$ stellige Zahl mit Ziffern zur Basis 2^r
- Countingsort zum stabilen Sortieren
- Beispiel: 32-Bit Zahl



  - $r = 8 \Rightarrow {}^{b}/_{r} = 4$ Countingsort Schritte, vier Ziffern zur Basis 2^8
  - Alternative: $r = 16 \Rightarrow {}^{b}/_{r} = 2$ Countingsort Schritte, zwei Ziffern zur Basis 2^{16}

Analyse

- Zur Erinnerung: Countingsort braucht $\Theta(n+k)$ um n Zahlen im Bereich 0 bis $k-1$ zu sortieren
- b-Bit Zahlen mit r-Bit Ziffern
  - Jeder Countingsort Schritt braucht $\Theta(n+2^r)$
  - $^b/_r$ Schritte (entspricht Anzahl Ziffern)
  - Aufwand $\mathrm{T(n,b)} = \Theta\left(^b/_r\,(n+2^r)\right)$
- Finde bestes r um $T(n,b)$ zu minimieren
  - Grösseres r heisst weniger Countingsort Schritte (Ziffern), aber exponentielles Wachstum in 2^r

Wahl von r

$$T(n,b) = \Theta\left({}^{b}/_{r}\,(n+2^r)\right)$$

- Ableiten nach r, Minimum wo Ableitung 0
- Beste Wahl $r = \lg n \Rightarrow T(n,b) = \Theta(bn/\lg n)$

$$\Theta\left({}^{b}/_{\lg n}\,(n+2^{\lg n})\right) = \Theta\left({}^{b}/_{\lg n}\,(n+n)\right) = \Theta\left({}^{bn}/_{\lg n}\right)$$

- Laufzeit in Anzahl Ziffern, statt Anzahl Bit: Für Zahlen mit d Ziffern à r Bit (Bereich 0 bis $2^{rd}-1$), und optimale Wahl $r = \lg n$

$$b = rd = (\lg n)d \Rightarrow T(n,b) = \Theta(dn)$$

- Falls sogar $b = O(\lg n)$, dann $T(n,b) = \Theta(n)$

Beispiele

- 2000 32-Bit Zahlen, $n = 2000, b = 32$
  - Bit pro Ziffer $r = \lceil \lg n \rceil = \lceil \lg 2000 \rceil = 11$
  - Anzahl Ziffern $d = \lceil {}^{b}/_{r} \rceil = \lceil {}^{32}/_{11} \rceil = 3$
  - Höchstens 3 Schritte
- Sortieren durch Mischen, Quicksort haben mindestens $\lceil \lg 2000 \rceil = 11$ Schritte (d.h. Durchläufe über alle n Zahlen)

Beispiele

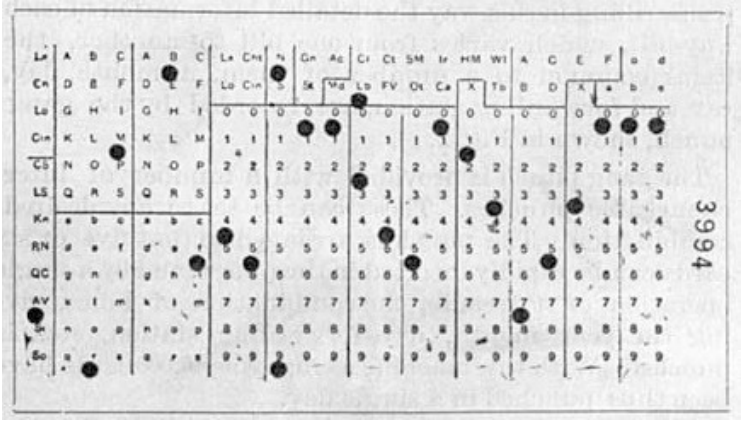
- 2^{20} (1 Mio.) 32-Bit Zahlen, $n = 2^{20}, b = 32$
  - Bit pro Ziffer: $r = \lceil \lg n \rceil = \lceil \lg 2^{20} \rceil = 20$
  - Anzahl Ziffern: $d = \lceil b/r \rceil = \lceil 32/20 \rceil = 2$
  - Höchstens 2 Schritte
- Sortieren durch Mischen, Quicksort haben mindestens $\lceil \lg 2^{20} \rceil = 20$ Schritte
- Aber: „ein Schritt“ in Radixsort beinhaltet zwei Durchläufe über alle n Zahlen
  - Siehe Countingsort Pseudocode

Zusammenfassung

- Radixsort gilt als schnell für grosse Eingaben
  - Falls $b = O(\lg n)$ asymptotische Laufzeit $\Theta(n)$
  - Besser als Quicksort, Heapsort, Mergesort
  - Kein vergleichender Sortieralgorithmus!
  - Nicht in-place
- Nachteil
  - Geringe Lokalität der Speicherzugriffe
  - „versteckte Konstanten“ in asymptotischer Laufzeit relativ gross
  - Optimierter Quicksort ist häufig schneller in Praxis

Ursprung

- Lochkartensortierer
- Erfunden von Herman Hollerith
- Amerikanische Volkszählung 1890



- Seine Firma war Teil der Gründung von IBM

http://en.wikipedia.org/wiki/Herman_Hollerith

Bucketsort

- Annahme: Eingabe sind n reelle Zahlen zufällig und gleichmässig verteilt in $[0,1)$
- Idee
  - Unterteile $[0,1)$ in n gleich grosse Teilintervalle
  - Verteile n Eingabewerte auf Teilintervalle
  - Sortiere jedes Teilintervall
  - Gehe der Reihe nach durch Intervalle und gib Werte sortiert aus

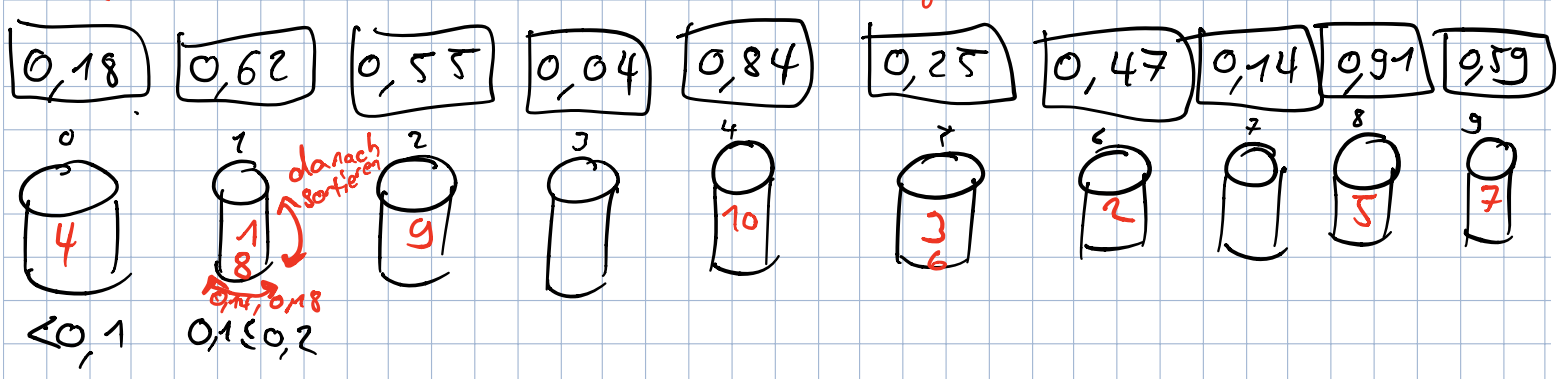
Bucketsort

Input: $A[1 \ldots n]$, where $0 \leq A[i] < 1$ for all i.
Auxiliary array: $B[0 \ldots n-1]$ of linked lists, each initially empty.

BUCKET-SORT(A, n)

1 **for** $i = 1$ **to** n
2 insert $A[i]$ into list $B\left[\lfloor n \cdot A[i] \rfloor\right]$
3 **for** $i = 0$ **to** $n - 1$
4 sort list $B[i]$ with insertion sort
5 cancatenate lists $B[0], B[1], \ldots, B[n-1]$ together in order
6 **return** the concatenated lists

1 2 3 4 5 6 7 8 9 10


0,18
0,62
0,55
0,04
0,84
0,25
0,47
0,14
0,91
0,59
0
1
2
3
4
5
6
7
8
9
4
1
8
9
10
3
6
2
7
5
7
danach sortieren
0,18, 0,62
< 0,1
0,1 ≤ 0,2

Korrektheit

- Zwei Elemente $A[i], A[j]$
- Sei $A[i] \leq A[j]$
- Für Indizes der Teilintervalle gilt
$$\lfloor n \cdot A[i] \rfloor \leq \lfloor n \cdot A[j] \rfloor$$
  - $A[i]$ ist in Intervall mit kleinerem oder gleichem Index wie $A[j]$
- Falls gleicher Index, Reihenfolge korrekt wegen Sortieren durch Einfügen
- Sonst korrekt wegen Auflistung der Intervalle am Schluss

Analyse

- Intuition
  - Funktioniert effizient wenn jedes Teilintervall etwa gleichviele Werte hat
- Z.B. n Intervalle für n Werte
  - Durchschnitt ist ein Wert pro Intervall
  - Im Durchschnitt kostet Sortieren pro Intervall $O(1)$
  - Für alle n Intervalle zusammen $O(n)$
- Erwartete Laufzeit ist $\Theta(n)$
  - Einschränkung: gilt nur, falls Eingabe „genügend gleichmässig verteilt“

Analyse

- Sei n_i Anzahl Elemente in Intervall $B[i]$
- Laufzeit von Bucketsort

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

  - Da quadratische Laufzeit für Sortieren durch Einfügen
- Analysiere erwartete Laufzeit (d.h. Erwartungswert der Laufzeit)

$$E[T(n)] = E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right]$$

Analyse

$$
\begin{aligned}
E[T(n)] &= E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\
&= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] && \text{(Erwartungswert ist linear)} \\
&= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) && (E[aX] = aE[X])
\end{aligned}
$$

Behauptung

$$E[n_i^2] = 2 - (1/n) \text{ für } i = 0, \ldots, n-1$$

Analyse

Beweis

Definiere für $i = 0, 1, \ldots, n-1$ und $j = 1, 2, \ldots, n$ Indikatorfunktionen $X_{ij} = I\{A[j] \text{ wird in Bucket } i \text{ eingefügt}\}$

Also $n_i = \sum_{j=1}^{n} X_{ij}$

$$E[n_i^2] = E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right]$$

$$= E\left[\sum_{j=1}^{n} X_{ij}^2 + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} X_{ij}X_{ik}\right]$$

$$= \sum_{j=1}^{n} E\left[X_{ij}^2\right] + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} E[X_{ij}X_{ik}]$$

Analyse

$$E[X_{ij}^2] = 0^2 \cdot \text{Wkt}\{A[j] \text{ wird nicht in Bucket i eingefügt}\} +$$
$$1^2 \cdot \text{Wkt}\{A[j] \text{ wird in Bucket i eingefügt}\}$$
$$= 0 \cdot \left(1 - \frac{1}{n}\right) + 1 \cdot \frac{1}{n}$$
$$= \frac{1}{n}$$

Für $k \neq j$ sind die Variablen X_{ij} und X_{ik} unabhängig, also

$$E[X_{ij}X_{ik}] = \underbrace{E[X_{ij}]}_{\frac{1}{n}} \cdot \underbrace{E[X_{ik}]}_{\frac{1}{n}}$$
$$= \frac{1}{n^2}$$

Analyse

$$
\begin{aligned}
E[n_i^2] &= \sum_{j=1}^{n} E[X_{ij}^2] + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} E[X_{ij}X_{ik}] \\
&= \sum_{j=1}^{n} \frac{1}{n} + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \frac{1}{n^2} \\
&= n \cdot \frac{1}{n} + 2\binom{n}{2}\frac{1}{n^2} \\
&= 1 + 2 \cdot \frac{n(n-1)}{2} \cdot \frac{1}{n^2} \\
&= 1 + \frac{n-1}{n} \\
&= 2 - \frac{1}{n}
\end{aligned}
$$

Analyse

$$
\begin{aligned}
E[T(n)] &= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \\
&= \Theta(n) + \sum_{i=0}^{n-1} O\left(2 - \frac{1}{n}\right) \\
&= \Theta(n) + O(n) \\
&= \Theta(n)
\end{aligned}
$$

Übersicht

- Untere Schranken für Sortieren
- Sortieren mit linearem Aufwand
- Mediane und Ranggrössen

Mediane und Ranggrössen

- i-te Ranggrösse ist das i-kleinste Element einer Menge mit n Elementen
- Minimum ist die erste Ranggrösse ($i = 1$)
- Maximum ist die n-te Ranggrösse ($i = n$)
- Median ist der „Mittelpunkt“
  - Unterer Median $\lfloor i = (n + 1)/2 \rfloor$
  - Oberer Median $\lceil i = (n + 1)/2 \rceil$
  - Identisch falls n ungerade

Auswahlproblem

- Eingabe: Menge A aus n (paarweise versch.) Zahlen und eine Zahl $1 \leq i \leq n$
- Ausgabe: Das Element x in A, das grösser als genau $i - 1$ andere Elemente von A ist
- Mögliche Lösung: mittels Sortieren
  1. Sortiere A
  2. Gib i-tes Element zurück
- Laufzeit ist $O(n \lg n)$
- Kann aber in linearer Zeit gelöst werden!

Minimum (erste Ranggrösse)

```
MINIMUM(A, n)
1   min = A[1]
2   for i = 2 to n
3       if min > A[i]
4           min = A[i]
5   return min
```

- Maximum ähnlich
- Geht nicht schneller als $O(n)$

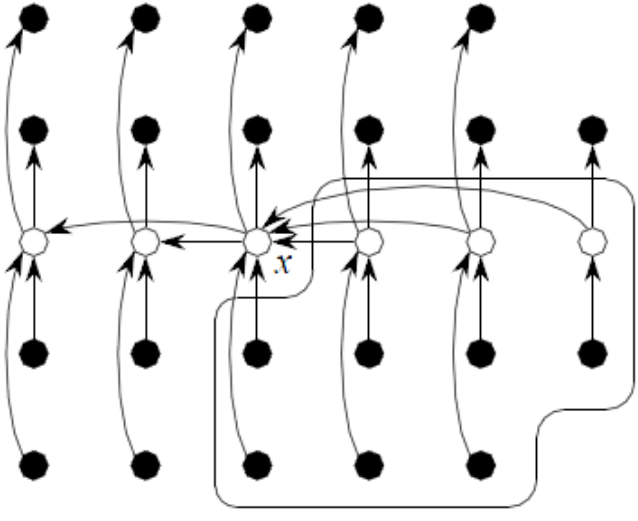
Auswahl in linearer Zeit

Algorithmus „$\mathrm{Select}(A, i)$“

1. Teile n Elemente in $g = n/5$ Gruppen von 5 Elementen (eine Gruppe enthält ev. weniger Elemente)
2. Bestimme Mediane $M = \{m_1, \dots, m_g\}$ aller Gruppen
3. Bestimme rekursiv Median x von M aus Schritt 2 ($\mathrm{Select}\left(M, {}^{g+1}/_{2}\right)$)
4. Zerlege Feld um x (d.h. x ist Pivot).
   Resultat: sei x die k-te Ranggrösse von A.
5. Falls $i = k$: Fertig, gib x zurück
   Sonst: Finde i-te Ranggrösse in richtiger Seite der Zerlegung rekursiv (je nachdem ob $i < k$ oder $i > k$)

Analyse

- Finde untere Schranke für Anzahl Elemente grösser als Pivotelement x
- x ist Median der Mediane: Mindestens Hälfte der Mediane ist $\geq x$
- Jede Gruppe mit Median $\geq x$ hat 3 Elemente $> x$
  - Ausnahmen: Gruppe, die x enthält; Gruppe, die weniger als 5 Elemente hat
- Anzahl Elemente $> x$ ist mindestens



$$3\left(\left\lceil {}^{1}/_{2}\left\lceil {}^{n}/_{5}\right\rceil - 2\right\rceil\right) \geq {}^{3n}/_{10} - 6$$

Analyse

- Symmetrisch: untere Schranke für Anzahl Elemente kleiner als Pivot x auch

$$3\left(\left\lceil {}^{1}/_{2}\left\lceil {}^{n}/_{5}\right\rceil - 2\right\rceil\right) \geq {}^{3n}/_{10} - 6$$

- D.h.: Schritt 5 ruft „Select“ mit höchstens $n - (3n/10 - 6) = 7n/10 + 6$ Elementen auf
- Rekursionsgleichung

$$T(n) \leq \begin{cases} O(1) & \text{wenn } n < 140 \\ T\left(\left\lceil {}^{n}/_{5}\right\rceil\right) + T\left({}^{7n}/_{10} + 6\right) + O(n) & \text{wenn } n \geq 140 \end{cases}$$

- Beweise $T(n) = O(n)$ mit Substitutions-methode (Buch Abschnitt 9.3)

Zusammenfassung

- Vergleichende Sortieralgorithmen
  - Untere Schranke: $\Omega(n \lg n)$
  - Beweis mithilfe Entscheidungsbaum
- Countingsort
  - Zählen der Vorkommnisse
  - Positionen bestimmen
  - Rückwärts einsortieren
  - $\Theta(n + k)$, falls $k \in O(n) \Rightarrow \Theta(n)$
  - Stabil
  - Kein vergleichender Sortieralgorithmus

Zusammenfassung

- Radixsort
  - Ziffer um Ziffer sortieren
  - Bei kleinster Wertigkeit anfangen
  - Stabilen inneren Sortieralgorithmus verwenden
- Bucketsort
  - Hier für Zahlen in $[0,1)$
  - Input auf n Kübel verteilen
  - Kübelweise sortieren
  - Zusammenfügen
  - Wenn Input gleichverteilt ist: $\Theta(n)$

Zusammenfassung

- Auswahlproblem in linearer Zeit $O(n)$ gelöst
- Wie bei vergleichenden Sortieralgorithmen nur Vergleiche von Elementen benutzt
  - Keine zusätzlichen Annahmen über Eingabe wie bei Countingsort, Radixsort, etc.
- Nicht betroffen von der unteren Schranke $\Omega(n \lg n)$ der vergleichenden Sortier-verfahren
  - Sortieren nicht nötig zur Lösung des Problems

Nächstes Mal

Kapitel 10: Elementare Datenstrukturen

- Verkettete Listen, Stacks, Bäume