

# Algorithmen und Datenstrukturen

## A11. Sortieren: Untere Schranke

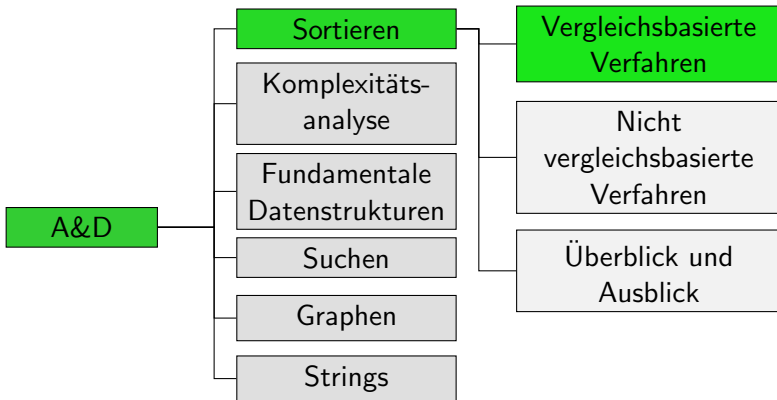
Marcel Lüthi and Gabriele Röger

Universität Basel

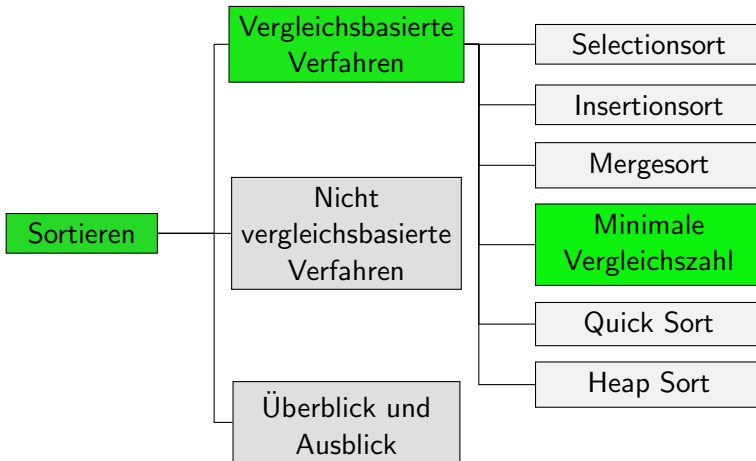
23. März 2022

# Untere Schranke an erforderliche Vergleichsoperationen

# Inhalt dieser Veranstaltung



# Sortierverfahren



# Fragestellung

- Mergesort hatte bisher mit  $O(n \log_2 n)$  die beste (Worstcase-)Laufzeit.
- Geht es noch besser?
- **Wir zeigen:** Nicht mit vergleichsbasierten Verfahren!

# Vorgehen

- **Schwierigkeit:** Wir können nicht einen bestimmten Algorithmus analysieren, sondern müssen eine Aussage über **alle möglichen Verfahren** treffen.

# Vorgehen

- **Schwierigkeit:** Wir können nicht einen bestimmten Algorithmus analysieren, sondern müssen eine Aussage über **alle möglichen Verfahren** treffen.
- Vergleichsbasierte Verfahren können die Eingabe nur anhand von Schlüsselvergleichen analysieren.

# Vorgehen

- **Schwierigkeit:** Wir können nicht einen bestimmten Algorithmus analysieren, sondern müssen eine Aussage über **alle möglichen Verfahren** treffen.
- Vergleichsbasierte Verfahren können die Eingabe nur anhand von Schlüsselvergleichen analysieren.
- Sie müssen jede Eingabe korrekt sortieren.



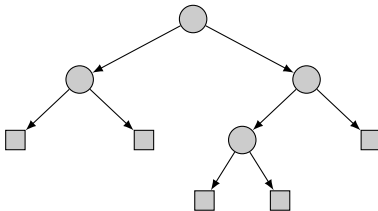
# Vorgehen

- **Schwierigkeit:** Wir können nicht einen bestimmten Algorithmus analysieren, sondern müssen eine Aussage über **alle möglichen Verfahren** treffen.
- Vergleichsbasierte Verfahren können die Eingabe nur anhand von Schlüsselvergleichen analysieren.
- Sie müssen jede Eingabe korrekt sortieren.
- Daraus können wir eine untere Schranke an die Anzahl der Schlüsselvergleiche im worst-case ableiten.

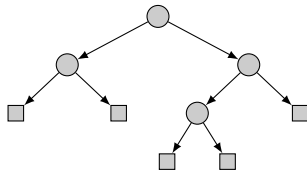
# Abstraktes Verhalten als Baum

Betrachte beliebigen vergleichsbasierten Sortieralgorithmus  $A$ .

- Verhalten hängt nur vom Ergebnis der Schlüsselvergleiche ab.
- Bei jedem Schlüsselvergleich gibt es zwei Möglichkeiten, wie der Algorithmus weiter macht.
- Wir können das graphisch als Baum darstellen.

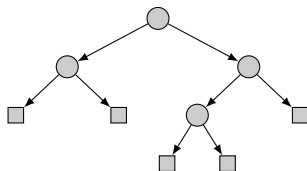


# Crashkurs Binärbäume



- **Binärbaum**: jeder Knoten hat höchstens zwei Nachfolger
- Knoten ohne Nachfolger heissen **Blätter** (Bild: eckige Knoten).
- Der Knoten ganz oben ist die **Wurzel**.
- Die **Tiefe** eines Blattes entspricht der Anzahl von Kanten von der Wurzel zu dem Blatt.

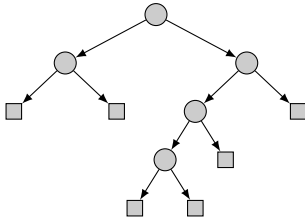
# Crashkurs Binärbäume



- **Binärbaum**: jeder Knoten hat höchstens zwei Nachfolger
- Knoten ohne Nachfolger heissen **Blätter** (Bild: eckige Knoten).
- Der Knoten ganz oben ist die **Wurzel**.
- Die **Tiefe** eines Blattes entspricht der Anzahl von Kanten von der Wurzel zu dem Blatt.

Die maximale Tiefe eines Blattes in einem Binärbaum mit  $k$  Blättern ist mindestens  $\log_2 k$ .

# Aufgabe (Slido)



Was ist die maximale Tiefe eines Blattes in diesem Baum?



# Ergebnis als Permutation

Was muss der Algorithmus können?

- **Annahme:** alle Elemente unterschiedlich
- Muss **alle Eingaben** der Grösse  $n$  **korrekt** sortieren.

# Ergebnis als Permutation

Was muss der Algorithmus können?

- **Annahme:** alle Elemente unterschiedlich
- Muss **alle Eingaben** der Grösse  $n$  **korrekt** sortieren.
- Wir können alle Algorithmen so anpassen, dass sie verfolgen, von welcher Position zu welcher Position die Elemente bewegt werden müssen.
- Das Ergebnis ist dann nicht das sortierte Array, sondern die entsprechende **Permutation**.

Beispiel:  $\text{pos0} \mapsto \text{pos2}$ ,  $\text{pos1} \mapsto \text{pos1}$ ,  $\text{pos2} \mapsto \text{pos0}$

# Ergebnis als Permutation

Was muss der Algorithmus können?

- **Annahme:** alle Elemente unterschiedlich
- Muss **alle Eingaben** der Grösse  $n$  **korrekt** sortieren.
- Wir können alle Algorithmen so anpassen, dass sie verfolgen, von welcher Position zu welcher Position die Elemente bewegt werden müssen.
- Das Ergebnis ist dann nicht das sortierte Array, sondern die entsprechende **Permutation**.  
Beispiel:  $pos_0 \mapsto pos_2$ ,  $pos_1 \mapsto pos_1$ ,  $pos_2 \mapsto pos_0$
- Da alle möglichen Eingaben der Grösse  $n$  korrekt gelöst werden müssen, muss der Algorithmus **alle  $n!$  möglichen Permutationen** erzeugen können.



# Untere Schranke

- Jedes Blatt in der Baumdarstellung entspricht einer Permutation.
- Bei Eingabegrösse  $n$  muss der Baum also mindestens  $n!$  Blätter haben.
- Die maximale Tiefe des entsprechenden Baumes ist demnach  $\geq \log_2(n!)$ .
- Es gibt also eine Eingabe der Grösse  $n$  mit  $\geq \log_2(n!)$  Schlüsselvergleichen.

# Untere Schranke: Abschätzung

Abschätzung von  $\log_2(n!)$

- Es gilt  $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

$$4! = 1 \cdot 2 \cdot 3 \cdot 4 \geq 2^2$$

$\quad \quad \quad \geq 2 \quad \geq 2$

## Untere Schranke: Abschätzung

Abschätzung von  $\log_2(n!)$

- Es gilt  $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

$$4! = 1 \cdot 2 \cdot 3 \cdot 4 \geq 2^2$$

$\geq 2 \quad \geq 2$

- $\log_2(n!) \geq \log_2\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) = \frac{n}{2} \log_2\left(\frac{n}{2}\right)$   
 $= \frac{n}{2}(\log_2 n + \log_2 \frac{1}{2}) = \frac{n}{2}(\log_2 n - \log_2 2)$   
 $= \frac{n}{2}(\log_2 n - 1)$

## Untere Schranke: Abschätzung

Abschätzung von  $\log_2(n!)$

- Es gilt  $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

$$4! = 1 \cdot 2 \cdot 3 \cdot 4 \geq \underset{\geq 2}{2} \cdot \underset{\geq 2}{2} \geq 2^2$$

- $\log_2(n!) \geq \log_2\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) = \frac{n}{2} \log_2\left(\frac{n}{2}\right)$   
 $= \frac{n}{2}(\log_2 n + \log_2 \frac{1}{2}) = \frac{n}{2}(\log_2 n - \log_2 2)$   
 $= \frac{n}{2}(\log_2 n - 1)$

### Theorem

Jeder *vergleichsbasierte Sortieralgorithmus* benötigt  $\Omega(n \log n)$  viele Schlüsselvergleiche. Damit liegt auch die *Laufzeit* in  $\Omega(n \log n)$ .

Mergesort ist asymptotisch optimal.

# Zusammenfassung

# Zusammenfassung

- Jedes **vergleichsbasierte Sortierverfahren** hat **mindestens leicht überlineare Laufzeit**.