Algorithmen und Datenstrukturen

TECHNISCHE UNIVERSITÄT DARMSTADT

vi visual inference SYSTEMS

Stefan Roth, SS 2025

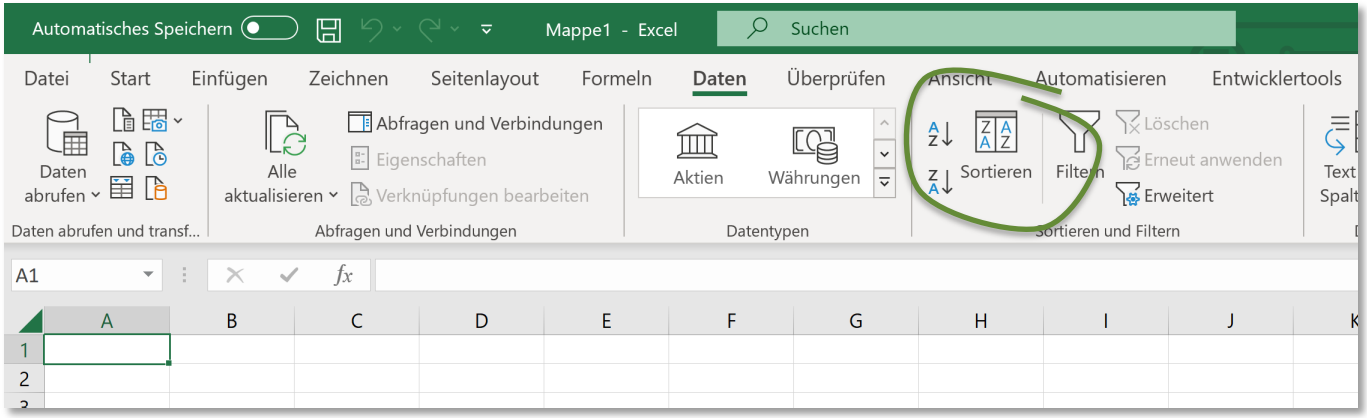
02

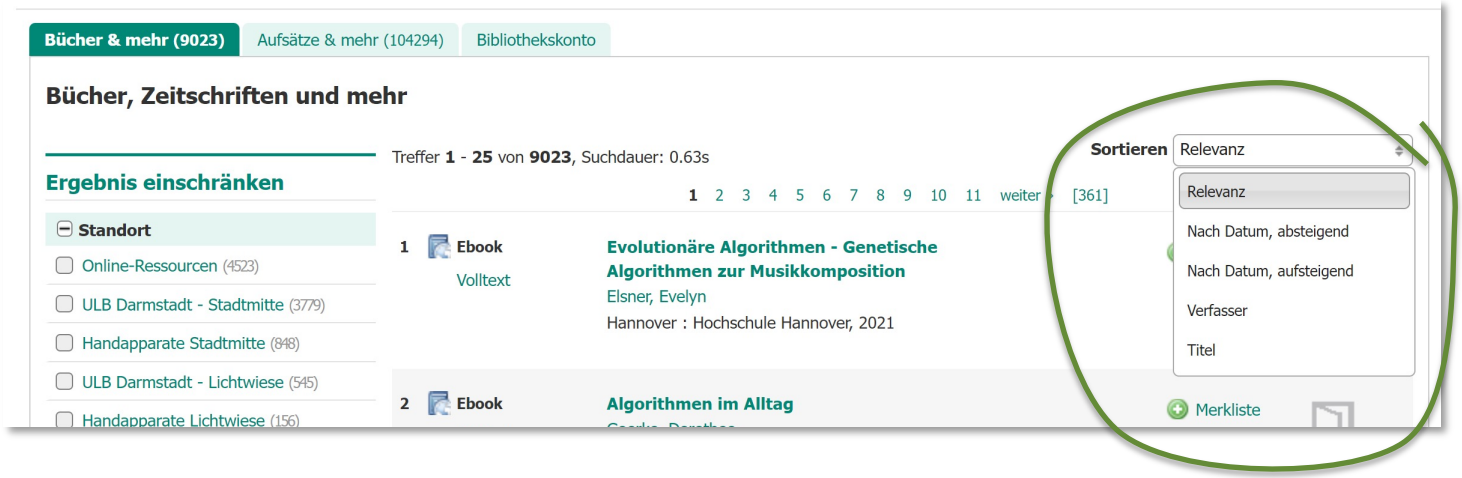
Sortieren

Folien beruhen auf der Veranstaltung von Prof. Marc Fischlin und Christian Janson aus dem SS 2024

Das Sortierproblem

Sortierproblem in der Praxis





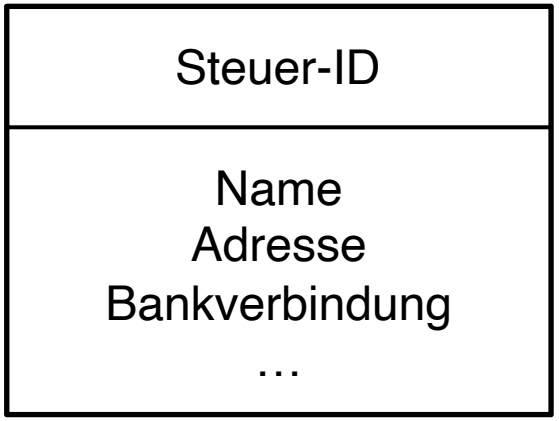
Sortierproblem

Gegeben: Folge von Objekten
Gesucht: Sortierung der Objekte
(gemäß ausgewiesenen Schlüsselwerts)

Wert, nach dem Objekt sortiert wird

Objekt

Beispiel



Schlüsselproblem

Schlüssel(werte) müssen nicht eindeutig sein,

aber müssen „sortierbar“ sein

Wert, nach dem Objekt sortiert wird

Objekt

Annahme im Folgenden:

Es gibt eine totale Ordnung $\leq$ auf der Menge M aller möglichen Schlüsselwerte

Totale Ordnung

Beispiel: lexikographische Ordnung auf Strings

Sei M eine nicht leere Menge und $\leq \subseteq M \times M$ eine binäre Relation auf M

Die Relation $\leq$ auf M ist genau dann eine totale Ordnung, wenn gilt:

Reflexivität: $\forall x \in M: \; x \leq x$

Transitivität: $\forall x, y, z \in M: \; x \leq y \; \wedge \; y \leq z \; \Rightarrow x \leq z$

Antisymmetrie: $\forall x, y \in M: \; x \leq y \; \wedge \; y \leq x \; \Rightarrow x = y$

Totalität: $\forall x, y \in M: \; x \leq y \; \vee \; y \leq x$

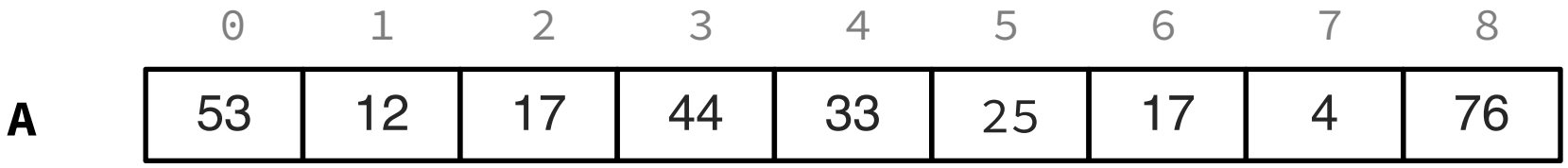
(ohne Totalität: partielle Ordnung)

Bemerkung: Totale Ordnung $\leq$ impliziert Relation $>$ durch $x > y :\Leftrightarrow x \not\leq y$

Darstellung in diesem Abschnitt

Wir betrachten nur Schlüssel(werte) ohne Satellitendaten,
in Beispielen meistens durch Zahlen gegeben

Eingabe der Objekte bzw. Schlüsselwerte in Form eines Arrays **`A`**:



Lese-/Schreibzugriff per Index in konstanter Zeit:

`y=A[2]` weist **`y`** den Wert 17 zu **`A[4]=99`** überschreibt 33 mit 99

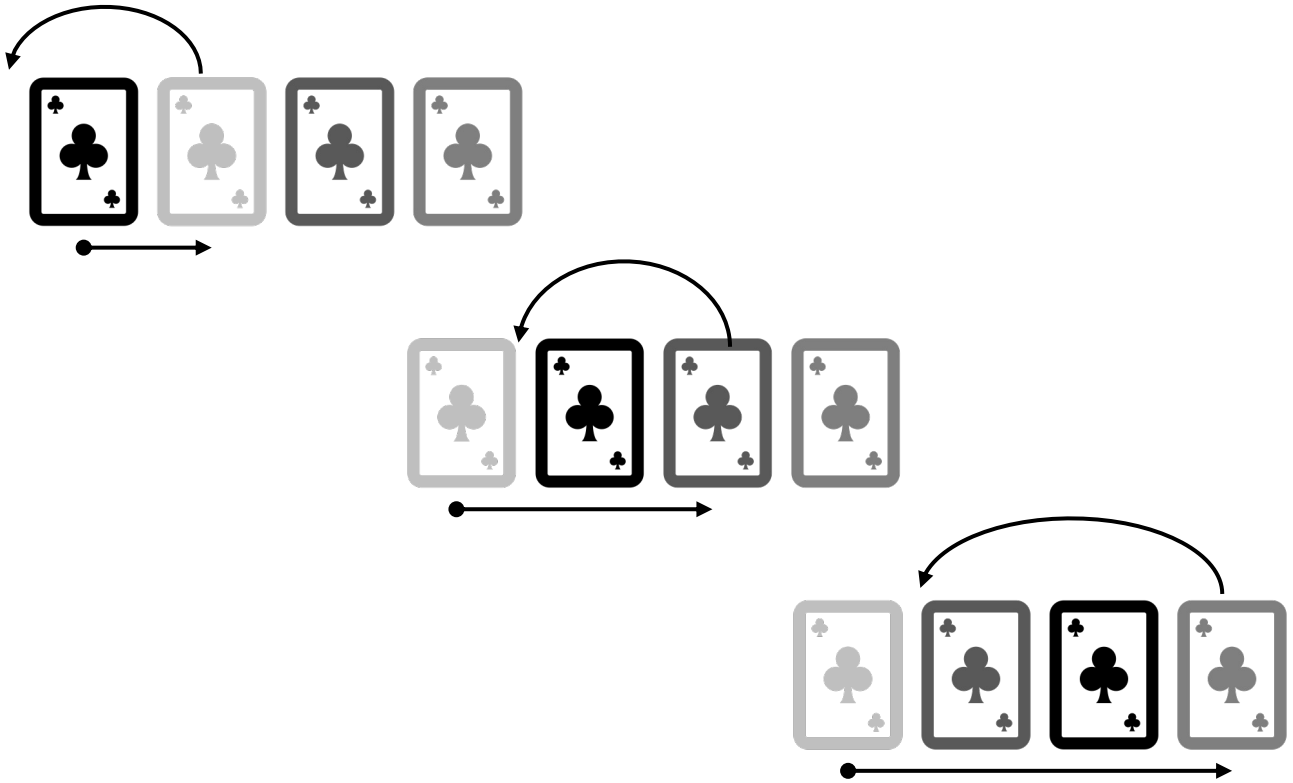
`A.length` beschreibt (fixe) Länge des Arrays, im Beispiel 9

`A[i…j]` beschreibt Teil-Array von Index **`i`** bis **`j`**, **`0<=i<=j<=A.length-1`**

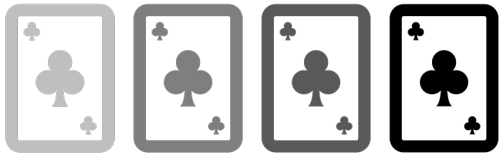
Insertion Sort

Idee: Karten umsortieren

(hell nach dunkel)



Gehe von links nach rechts durch
und sortiere aktuelle Karte richtig nach links ein



Algorithmus: Insertion Sort

Durch `!(A[j]=<key)` wohldefiniert

```
insertionSort(A)
1  FOR i=1 TO A.length-1 DO
     // insert A[i] in pre-sorted sequence A[0...i-1]
2    key=A[i];
3    j=i-1; // search for insertion point backwards
4    WHILE j>=0 AND A[j]>key DO
5        A[j+1]=A[j]; // move elements to right
6        j=j-1;
7    A[j+1]=key;
```

Wir beginnen mit **`i=1`**, aber erstes Element ist **`A[0]`**

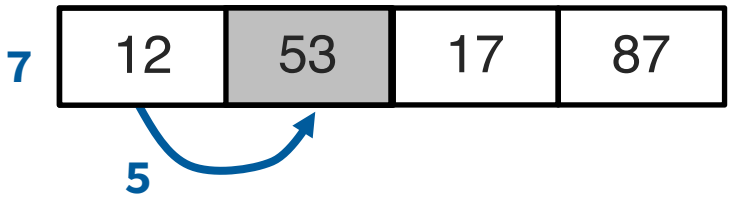
Short Circuit Evaluation (wie in Java):
Wenn erste AND-Bedingung **`false`**, wird zweite Bedingung nicht mehr ausgewertet

Beispiel: Insertion Sort (I)

```
insertionSort(A)
1  FOR i=1 TO A.length-1 DO
     // insert A[i] in pre-sorted sequence A[0...i-1]
2    key=A[i];
3    j=i-1; // search for insertion point backwards
4    WHILE j>=0 AND A[j]>key DO
5        A[j+1]=A[j]; // move elements to right
6        j=j-1;
7    A[j+1]=key;
```

Beispiel: **FOR-Schleife i=1** **2 key=12** **3 j=0**

WHILE-Schleife j=0 **A[j]=53 > key=12**



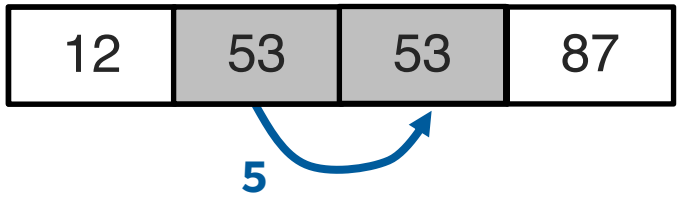
1. Fall:
linker Rand erreicht

Beispiel: Insertion Sort (II)

```
insertionSort(A)
1  FOR i=1 TO A.length-1 DO
     // insert A[i] in pre-sorted sequence A[0...i-1]
2    key=A[i];
3    j=i-1; // search for insertion point backwards
4    WHILE j>=0 AND A[j]>key DO
5        A[j+1]=A[j]; // move elements to right
6        j=j-1;
7    A[j+1]=key;
```

Beispiel: **FOR-Schleife i=2** **2 key=17** **3 j=1**

WHILE-Schleife j=1 **A[j]=53 > key=17**



6 j=0

2. Fall:
Einfügeposition erreicht

Beispiel: Insertion Sort (III)

```
insertionSort(A)
1  FOR i=1 TO A.length-1 DO
     // insert A[i] in pre-sorted sequence A[0...i-1]
2    key=A[i];
3    j=i-1; // search for insertion point backwards
4    WHILE j>=0 AND A[j]>key DO
5        A[j+1]=A[j]; // move elements to right
6        j=j-1;
7    A[j+1]=key;
```

Beispiel: **FOR-Schleife i=2** **2 key=17** **3 j=1**

WHILE-Schleife j=0 **A[j]=12 < key=17**

7

6 j=0

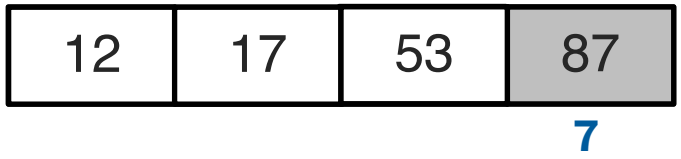
2. Fall:
Einfügeposition erreicht

Beispiel: Insertion Sort (IV)

```
insertionSort(A)
1  FOR i=1 TO A.length-1 DO
     // insert A[i] in pre-sorted sequence A[0...i-1]
2    key=A[i];
3    j=i-1; // search for insertion point backwards
4    WHILE j>=0 AND A[j]>key DO
5        A[j+1]=A[j]; // move elements to right
6        j=j-1;
7    A[j+1]=key;
```

Beispiel: **FOR-Schleife i=3** **2 key=87** **3 j=2**

WHILE-Schleife j=2 **A[j]=53 < key=87**



3. Fall:
Element bleibt

Terminierung

```
insertionSort(A)
1  FOR i=1 TO A.length-1 DO
     // insert A[i] in pre-sorted sequence A[0...i-1]
2    key=A[i];
3    j=i-1; // search for insertion point backwards
4    WHILE j>=0 AND A[j]>key DO
5        A[j+1]=A[j]; // move elements to right
6        j=j-1;
7    A[j+1]=key;
```

Jede Ausführung der **`WHILE`**-Schleife erniedrigt **`j<n`** in jeder Iteration um 1 und bricht ab, wenn **`j<0`** → terminiert also immer

`FOR`-Schleife wird nur endlich oft durchlaufen

Korrektheit (I)

```
insertionSort(A)
1  FOR i=1 TO A.length-1 DO
     // insert A[i] in pre-sorted sequence A[0...i-1]
2    key=A[i];
3    j=i-1; // search for insertion point backwards
4    WHILE j>=0 AND A[j]>key DO
5        A[j+1]=A[j]; // move elements to right
6        j=j-1;
7    A[j+1]=key;
```

Schleifeninvariante (der **FOR**-Schleife):

Bei jedem Eintritt für Zählerwert **i** (und nach letzter Ausführung) entsprechen die aktuellen Einträge in **A[0],…,A[i-1]** den sortierten ursprünglichen Eingabewerten **a[0],…,a[i-1]**. Ferner gilt **A[i]=a[i],…, A[n-1]=a[n-1]**.

Korrektheit (II)

Beweis per Induktion:

Induktionsbasis **`i=1`**: Beim ersten Eintritt ist **`A[0]=a[0]`** und „sortiert“.
Ferner gilt noch **`A[1]=a[1],…,A[n-1]=a[n-1]`**.

Bei jedem Eintritt für Zählerwert **`i`** (und nach letzter Ausführung) entsprechen die aktuellen Einträge in **`A[0],…,A[i-1]`** den sortierten ursprünglichen Eingabewerten **`a[0],…,a[i-1]`**.
Ferner gilt **`A[i]=a[i],…, A[n-1]=a[n-1]`**.

Korrektheit (III)

Beweis per Induktion: Induktionsschritt von **`i-1`** auf **`i`**:

Vor der (**`i-1`**)–ten Ausführung galt Schleifeninvariante nach Voraussetzung. Inbesondere war **`A[0],…,A[i-2]`** sortierte Version von **`a[0],…,a[i-2]`** und **`A[i-1]=a[i-1],…,A[n-1]=a[n-1]`**

Durch die **`WHILE`**-Schleife wurde **`A[i-1]=a[i-1]`** nach links einsortiert und größere Elemente von **`A[0],…,A[i-2]`** um jeweils eine Position nach rechts verschoben. Elemente **`A[i],…,A[n-1]`** wurden nicht geändert

Also gilt Invariante auch für **`i`**

> Bei jedem Eintritt für Zählerwert **`i`** (und nach letzter Ausführung) entsprechen die aktuellen Einträge in **`A[0],…,A[i-1]`** den sortierten ursprünglichen Eingabewerten **`a[0],…,a[i-1]`**.
> Ferner gilt **`A[i]=a[i],…, A[n-1]=a[n-1]`**.

Korrektheit (IV)

Aus Schleifeninvariante folgt für letzte Ausführung
(also quasi vor gedanklichem Eintritt der Schleife für **`i=n`**):

`A[0],…,A[n-1]` ist sortierte Version von **`a[0],…,a[n-1]`**

und somit am Ende das Array sortiert

Bei jedem Eintritt für Zählerwert **`i`** (und nach letzter Ausführung) entsprechen die aktuellen Einträge in **`A[0],…,A[i-1]`** den sortierten ursprünglichen Eingabewerten **`a[0],…,a[i-1]`**.
Ferner gilt **`A[i]=a[i],…, A[n-1]=a[n-1]`**.

Wozu brauchen Sie (intuitiv) beim Sortieren die vier Eigenschaften einer totalen Ordnung?

Überlegen Sie sich, dass Insertion Sort **stabil** ist, d.h. die Reihenfolge von Objekten mit gleichem Schlüssel bleibt erhalten.