

EINI

LogWing/WiMa/MP

**Einführung in die Informatik für
Naturwissenschaftler und Ingenieure**

Vorlesung 2 SWS WS 18/19

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@tu-dortmund.de
<http://ls14-www.cs.tu-dortmund.de>

▶ Kapitel 5

Algorithmen und Datenstrukturen

- ▶ Konstruktion von Datentypen: Arrays
- ▶ Algorithmen: Sortieren

▶ Unterlagen

- ▶ Dißmann, Stefan und Ernst-Erich Doberkat: *Einführung in die objektorientierte Programmierung mit Java*, 2. Auflage. München [u.a.]: Oldenbourg, 2002, Kapitel 3.4 & 4.1. (→ ZB oder Volltext aus Uninetz)
- ▶ Echtele, Klaus und Michael Goedicke: *Lehrbuch der Programmierung mit Java*. Heidelberg: dpunkt-Verl, 2000, Kapitel 4. (→ ZB)
- ▶ Gumm, Heinz-Peter und Manfred Sommer: *Einführung in die Informatik*, 10. Auflage. München: De Gruyter, 2012, Kapitel 2.7 – 2.8. (→ Volltext aus Uninetz)

- Prolog
- Arrays
- Sortieren
- Rekursive
Datenstrukturen

Übersicht

Begriffe

- ✓ Spezifikationen, Algorithmen, formale Sprachen
- ✓ Programmiersprachenkonzepte
- ✓ Grundlagen der imperativen Programmierung

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

- Algorithmen und Datenstrukturen
 - Felder
 - Sortieren
 - Rekursive Datenstrukturen (Baum, binärer Baum, Heap)
 - ▶ Heapsort
- ▶ Objektorientierung
 - ▶ Einführung
 - ▶ Vererbung
 - ▶ Anwendung

In diesem Kapitel:

- **Prolog**
- Arrays
- Sortieren
- Rekursive
Datenstrukturen

- Prolog
- **Arrays**
- Sortieren
- Rekursive
Datenstrukturen

▶ Array:

- ▶ Datenstruktur zur Abbildung gleichartiger Daten
- ▶ Deklaration
- ▶ Dimensionierung und Zuordnung von Speicher zur Laufzeit
- ▶ Zuweisung: ganzes Array, Werte einzelner Elemente

▶ Algorithmen auf Arrays: Beispiel Sortieren

- ▶ **naives Verfahren:** Minimum bestimmen, entfernen, Restmenge sortieren
- ▶ **Heapsort:** ähnlich, nur mit Binärbaum über Indexstruktur
- ▶ **Quicksort:** divide & conquer, zerlegen in zwei Teilmengen anhand eines Pivotelementes

Arrays I

Motivation:

Schleifen erlauben die Verarbeitung mehrerer Daten auf „einen Schlag“.

- ▶ Entsprechend auf der Variablenseite ist die Zusammenfassung mehrerer Variablen gleichen Typs: **Arrays** oder **Felder**.
- ▶ Beispiele:
 - ▶ Zeichenketten/Strings: Arrays aus Character/Zeichen
 - ▶ Vektoren, Matrizen: Arrays aus Integer/Float-Variablen
 - ▶ Abbildung eines Lagerbestandes durch Angabe der Menge für einen Artikel und einen Lagerort bei n unterschiedlichen Artikeln und m Orten:
 - Bestand [1] [5]: der Bestand des Artikels 1 am Ort 5
 - In Java: Bestand [i] [j]: Artikel mit Nummer i und Ort j

Arrays II

► Fragen:

- Wie werden Arrays deklariert?
- Wie werden Daten in Arrays abgelegt, verändert und ausgegeben?
- Wie wird die Größe eines Arrays festgelegt?
- Warum müssen wir die Größe überhaupt festlegen?

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- **Arrays**
- Sortieren
- Rekursive
Datenstrukturen

Arrays III

- ▶ **Arrays sind Variablen,**
 - ▶ sie müssen daher auch deklariert werden, bevor sie benutzt werden.
 - ▶ die viele Variablen enthalten, die vom gleichen Typ sind.
- ▶ Die Anzahl der Dimensionen entspricht der Anzahl der Indexausdrücke.

- ▶ **Deklarationen:**

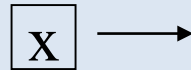
```
int[] x;           // ein-dimensional int
double[][] y;     // zwei-dimensional double
```

- ▶ Sie legen allerdings (anders als bei `int z;`) noch keinen Speicherplatz fest.

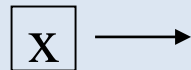
Arrays IV

Die Deklarationen einer Array-Variablen legt nur einen Verweis auf ein Array fest, die Dimensionierung muss extra erfolgen!

```
int[] x;
```

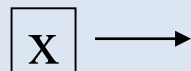


```
x = new int [15];
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

```
int[] x = {1,2,4,3};
```



0	1	2	3
1	2	4	3

- Prolog
- **Arrays**
- Sortieren
- Rekursive
Datenstrukturen

Arrays V

Die Deklarationen einer Array-Variablen legt nur einen Verweis auf ein Array fest, die Dimensionierung muss extra erfolgen!

- ▶ Dimensionierung mittels **new** (Schlüsselwort)
- ▶ [**anzahl**] gibt die Anzahl der Elemente an.
- ▶ Ist kein Inhalt angegeben, wird jedes Element mit 0 initialisiert.
- ❖ Beachte: Indizes **immer** 0 ... Anzahl -1

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- **Arrays**
- Sortieren
- Rekursive
Datenstrukturen

Zuweisungen

- ▶ Bei der Deklaration einer Array-Variablen werden die Standardwerte zugewiesen:
 - ▶ z.B.: `int` alles 0 ...

- ▶ Andere Variante:

- ▶ Belegung mit direkt angegebenen Konstanten

```
int [] m =  
{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

- ▶ Größe und Belegung sind direkt festgelegt:
 - keine Erzeugung mittels **new** notwendig

In diesem Kapitel:

- Prolog
- **Arrays**
- Sortieren
- Rekursive
Datenstrukturen

Zuweisung an Array-Variable

Array-Größe ist nach Ausführung des new-Operators fest!

▶ Veränderung der Größe nur durch Programm möglich:

```
int [] a,b;  
a = new int [10];  
....          // Zuweisung an die Elemente 0 ..9  
  
b = new int [20];  
  
for (int i=0; i < 10; i++) b[i] = a[i];  
  
a = b;      // nun verweisen a und b auf das  
            // gleiche Array!
```

Zuweisung an Array-Variablen: Beispiel 1

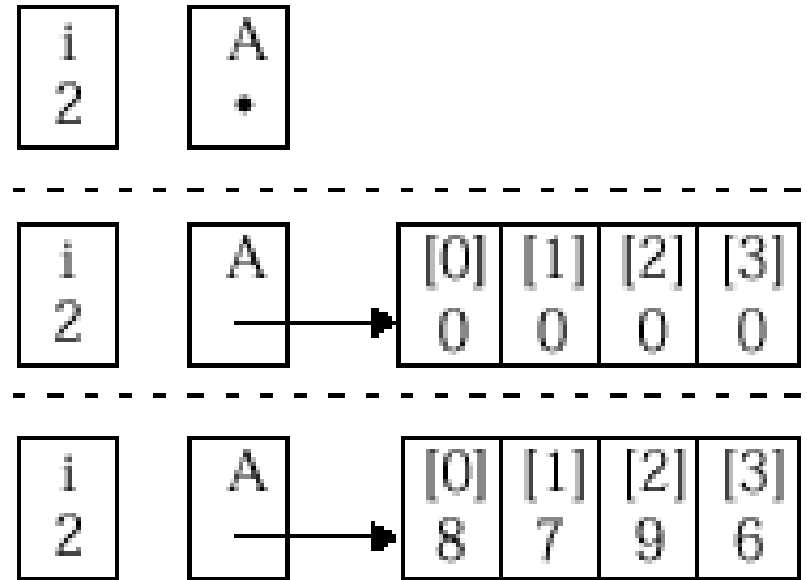
```
int i = 2;
```

```
int [] A;
```

```
A = new int [4];
```

```
A [0] = 8;    A [1] = 7;
```

```
A [i] = 9;    A [3] = 6;
```



Echtle/Goedicke, Heidelberg: *Abb. 2–19* (Ausschnitt), S. 73 © dpunkt 2000.

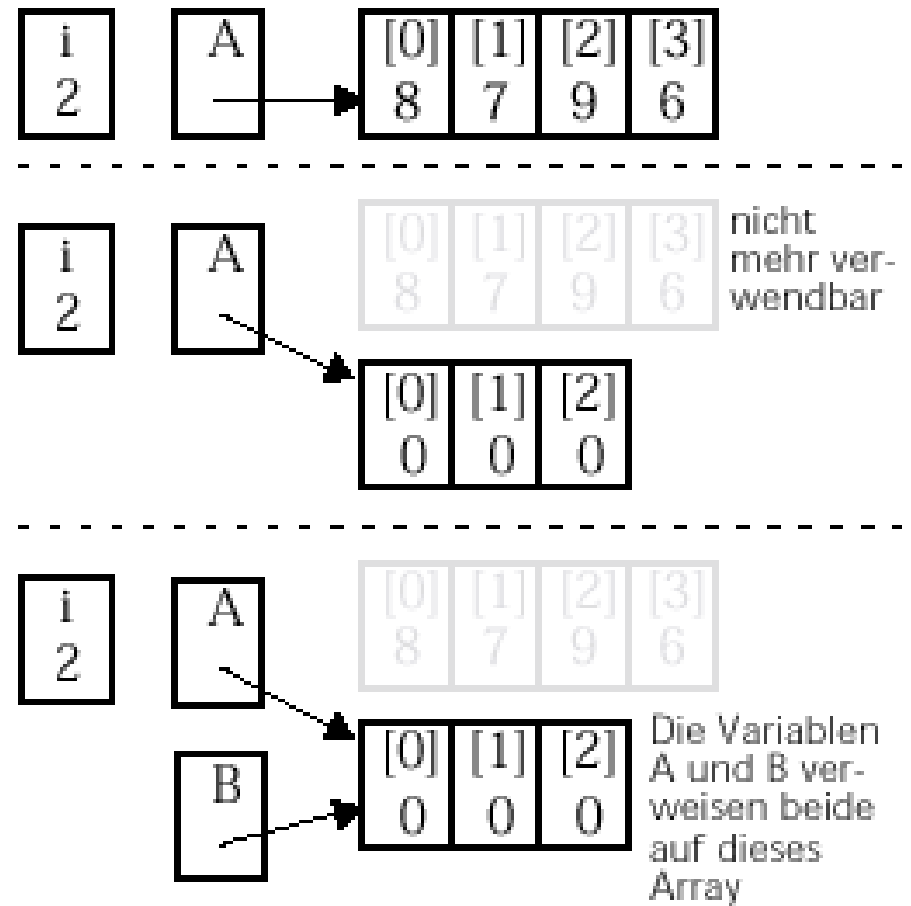
Zuweisung an Array-Variablen: Beispiel 2

...
A [0] = 8; A [1] = 7;
A [i] = 9; A [3] = 6;

A = new int [3];

int [] B;

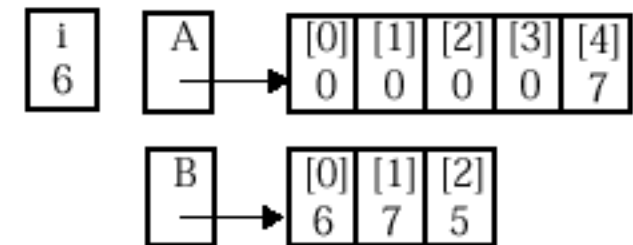
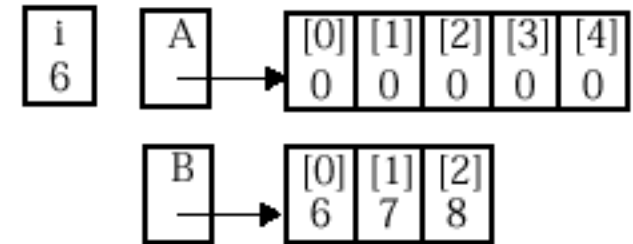
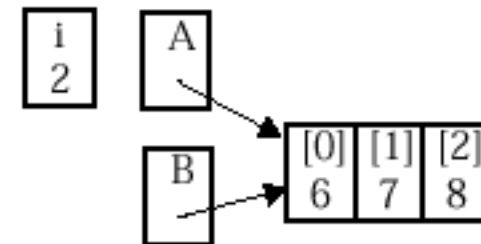
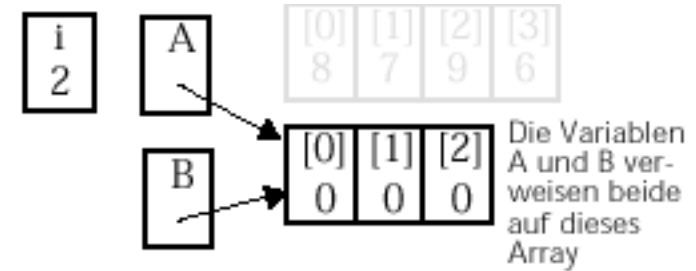
B = A;



Echtle/Goedicke, Heidelberg: *Abb. 2-19* (Ausschnitt), S. 73 © dpunkt 2000.

Zuweisung an Array-Variablen: Beispiel 3

```
.....  
B = A;  
A [0] = 6;  
B [1] = 7;  
B [2] = B [0] + 2;  
  
i = B [0];  
A = new int [5];  
  
A [i - 2] = B [1];  
B [i - 4] = A.length;
```



Echtle/Goedicke, Heidelberg: Abb. 2–19 (Ausschnitt), S. 73 © dpunkt 2000.

Dynamische Größe von Arrays

Falls die Größe eines Arrays zum Zeitpunkt der Erstellung nicht bekannt ist:

- ▶ Die Größe könnte z.B. auch eingelesen werden.
- ▶ In Java kann durch `x.length` die Anzahl der Elemente dynamisch bestimmt werden:

```
int anzahl = scanner.nextInt();
int anfangswert = 0;

int[] vektor = new int[anzahl];

for (int i = 0; i < vektor.length; i++) {
    vektor[i] = anfangswert ;
}
```

- ❖ Beachte: Index läuft 0 ... vektor.length -1.

- Prolog
- **Arrays**
- Sortieren
- Rekursive
Datenstrukturen

Berücksichtigung von Typen

Strenges Typsystem:

- ▶ Für einzelne Elemente eines Arrays, die selbst keine Arrays sind, ist dies klar:

```
int[] a = new int[3];  
a[1] = 3;
```

- ▶ Für Arrays gilt bei Zuweisungen:
 - ▶ Typ der Grundelemente und die Anzahl der Dimensionen müssen übereinstimmen:

```
int[] a;  
...  
a = b; // klappt nur, wenn b ebenfalls  
       // 1-dimensionales int Array ist
```

- Prolog
- **Arrays**
 - Zuweisung
- Sortieren
- Rekursive
Datenstrukturen



Arrays

Artikel im EINI-Wiki:

→ **Array**

→ **Zeichenkette**

Kapitel 5

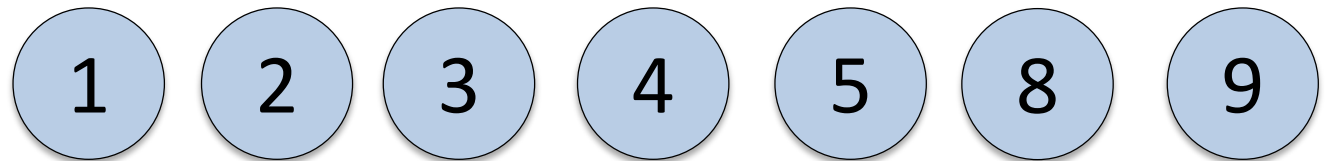
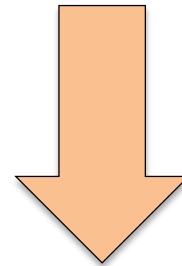
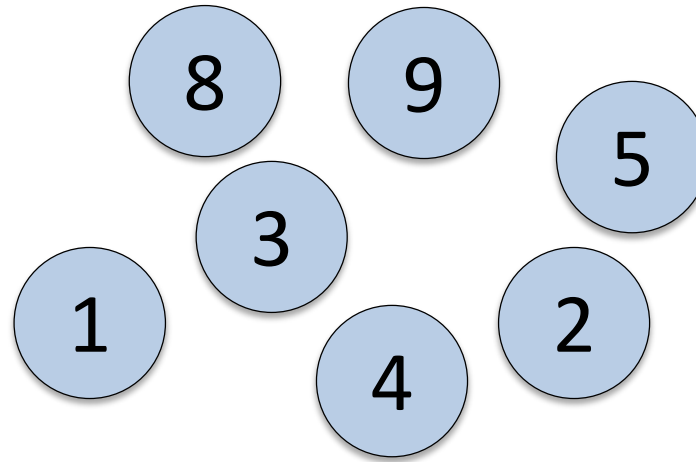
Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- **Arrays**
- Sortieren
- Rekursive
Datenstrukturen

Arrays: Internes Sortieren I

- ▶ Sortieren ist ein Standardproblem in der Informatik.



In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Arrays: Internes Sortieren II

- ▶ Internes Sortieren bringt die Elemente einer Folge in die richtige Ordnung.
- ▶ Viele Alternativen bzgl. Sortieren sind entwickelt worden.
- ▶ Das einfache interne Sortieren (wie hier vorgestellt) hat zwar **geringen** Speicherplatzbedarf, aber eine **hohe** Laufzeit.
- ▶ Verfahren:
 - ▶ Vertausche Elemente der Folge solange, bis sie in der richtigen Reihenfolge sind.
- ▶ Hier wird als Speicherungsstruktur ein Array benutzt.

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

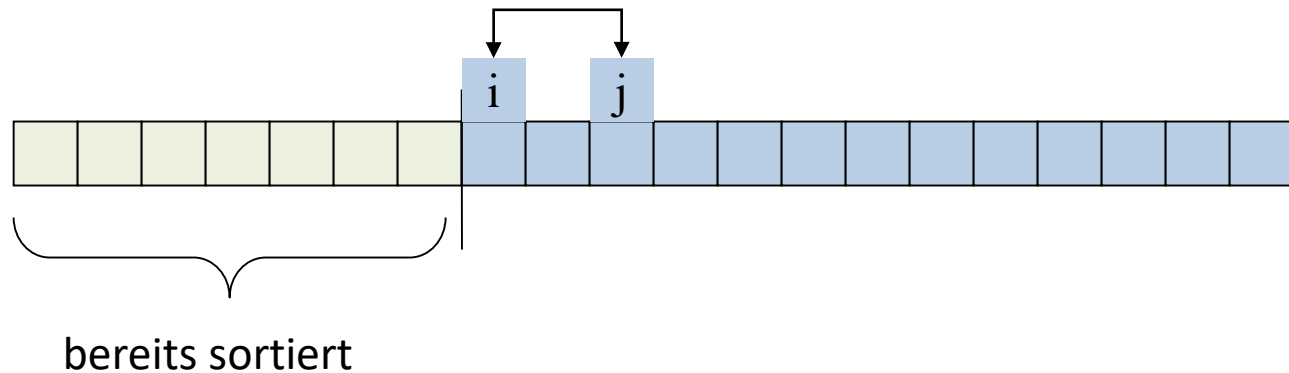
In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung

Die Idee:

- ▶ Ausgangspunkt: Alles vor der Stelle i ist bereits sortiert.
- ▶ Man vergleicht das Element an der Stelle i mit allen weiteren Elementen (im Beispiel: j).
- ▶ Falls das Element an der Stelle i größer ist als an der Stelle j : Vertausche die Elemente an den Stellen i und j .

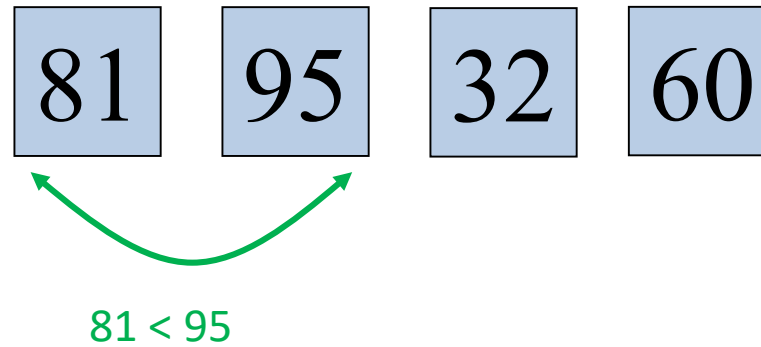


In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel I

Ausgangspunkt:

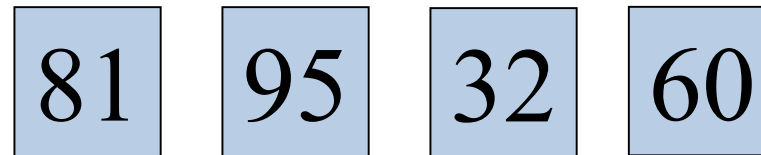


bereits sortiert

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel II



$81 < 32$



EINI LogWing /
WiMa

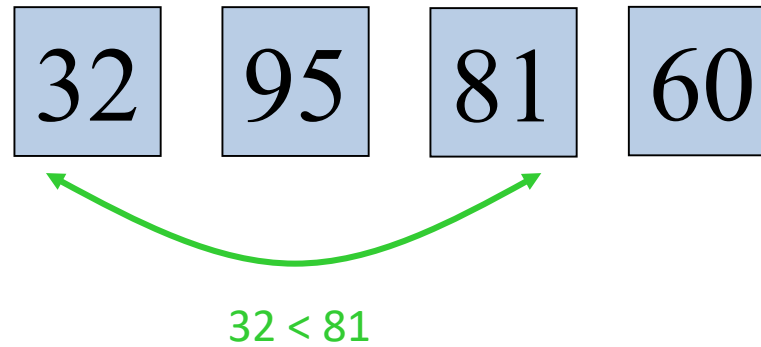
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel III



EINI LogWing /
WiMa

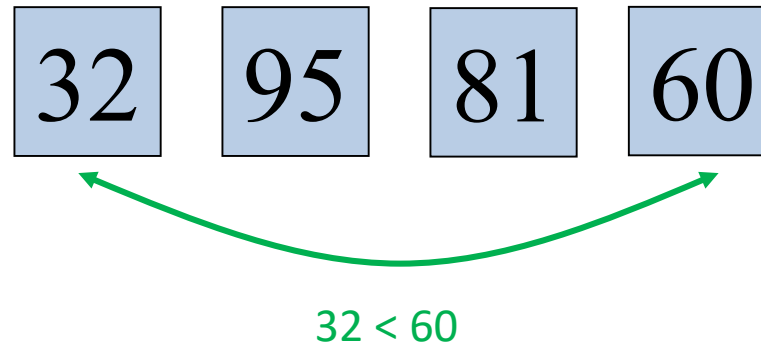
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel IV



EINI LogWing /
WiMa

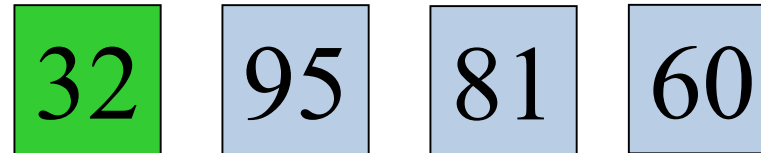
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel V



EINI LogWing /
WiMa

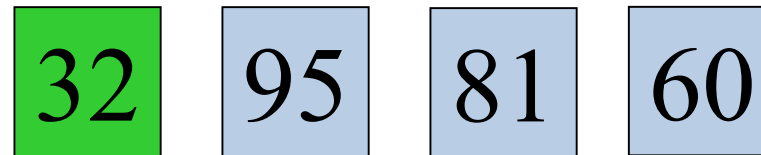
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel VI



95 < 81



EINI LogWing /
WiMa

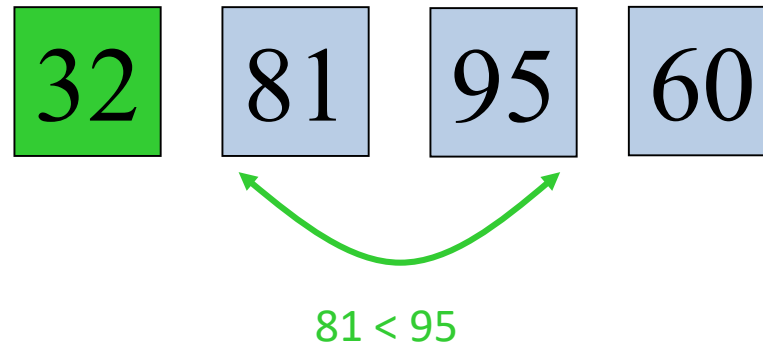
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel VII



EINI LogWing /
WiMa

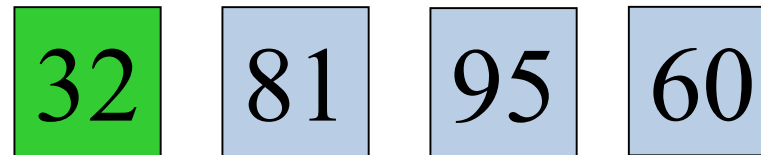
Kapitel 5

Algorithmen und
Datenstrukturen

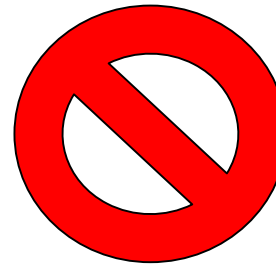
In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel VIII



81 < 60



EINI LogWing /
WiMa

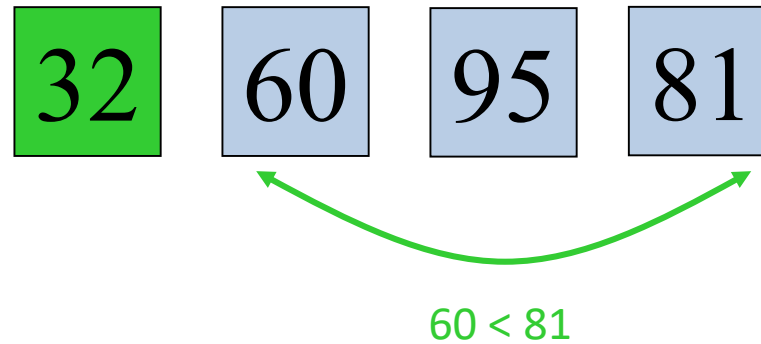
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel IX



EINI LogWing /
WiMa

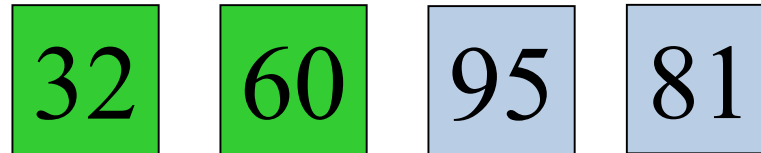
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel X



EINI LogWing /
WiMa

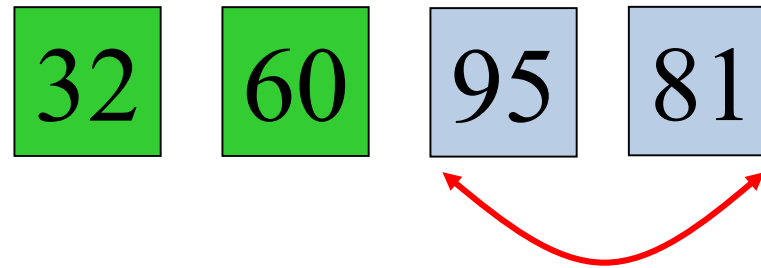
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel XI



95 < 81



EINI LogWing /
WiMa

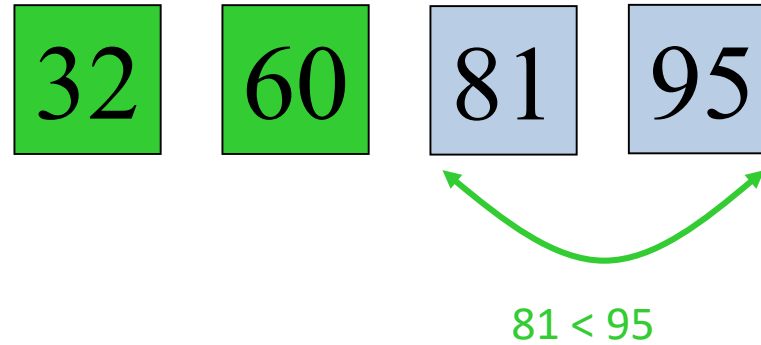
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel XII



EINI LogWing /
WiMa

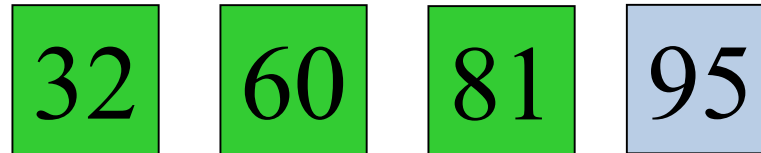
Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel XIII



EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortierung: Beispiel XIV

32 60 81 95

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Einlesen der Werte

```
int n = scanner.nextInt();
int[] a = new int[n];

// Lies Elemente ein

for (int i = 0; i < n; i++) {
    a[i] = scanner.nextInt();
}
```

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Eigentliche Sortierung ...

Diese Schritte müssen für alle Elemente im Array erledigt werden:

```
for (int i = 0; i < n - 1; i++) {  
  
    // Prüfe, ob a[i] Nachfolger hat,  
    // die kleiner als a[i] sind:  
    for (int j = i + 1; j < n; j++) {  
  
        if (a [i] > a [j]) { // Ist ein Nachfolger kleiner?  
  
            // Vertausche a[i] mit a[j]:  
            // Ringtausch mit Hilfsvariable z  
            int z = a [i];  
            a [i] = a [j];  
            a [j] = z;  
  
        }  
  
    }  
  
}
```

Ausgabe

Zum Schluss wird noch alles ausgegeben:

```
// Gib sortierte Elemente aus

System.out.println ("Sortierte Elemente:");
for (int i = 0; i < n; i++) {
    System.out.print (a [i] + ", ");
}
```

EINI LogWing /
WiMa

Kapitel 5
Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Der Ablauf noch einmal tabellarisch:

i	j	a[0]	a[1]	a[2]	a[3]
0	1	81	95	32	60
0	2	81	95	32	60
		32	95	81	60
0	3	32	95	81	60
1	2	32	95	81	60
		32	81	95	60
1	3	32	81	95	60
		32	60	95	81
2	3	32	60	95	81
		32	60	81	95

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Gesamtes Programm

```
01 import java.util.Scanner;
02
03 public class A533 {
04     public static void main(String[] args) {
05         Scanner scanner = new Scanner(System.in);
06
07         int n = scanner.nextInt();
08         int[] a = new int[n];
09
10         for (int i = 0; i < n; i++) {
11             a[i] = scanner.nextInt();
12         }
13
14         for (int i = 0; i < n - 1; i++) {
15             for (int j = i + 1; j < n; j++) {
16                 if (a [i] > a [j]) {
17                     int z = a [i];
18                     a [i] = a [j];
19                     a [j] = z;
20                 }
21             }
22         }
23
24         System.out.println ("Sortierte Elemente:");
25         for (int i = 0; i < n; i++) {
26             System.out.print (a [i] + ", ");
27         }
28     }
29 }
30 }
```

Alternativen?

- ▶ Könnte man die Algorithmusidee auch anders formulieren?
 - ▶ Finde Minimum x der aktuellen Menge.
 - ▶ Positioniere x an den Anfang.
 - ▶ Sortiere Restmenge nach Entfernen von x .
- ▶ Rekursive Formulierung ?
- ▶ Weitere Fragen:
 - ▶ Terminierung?
 - ▶ Korrektheit?
 - ▶ Aufwand, Effizienz?



- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Bemerkungen zum Aufwand I

Der Aufwand wird nach **Anzahl der Ausführungen von Elementaroperationen** betrachtet.

- ▶ Im Wesentlichen sind das beim Sortieren Vergleiche und Zuweisungen.
- ▶ Meist begnügt man sich mit einer vergrößernden Abschätzung, der sogenannten O-Notation.
- ▶ Diese Abschätzung wird in der Regel nach der Größe des Problems bestimmt, hier die Anzahl der zu sortierenden Elemente.

EINI LogWing /
WiMa

Kapitel 5
Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Bemerkungen zum Aufwand II

- ▶ Obiges Sortierverfahren:
 - ▶ **zwei** geschachtelte **for**-Schleifen,
 - ▶ die beide über das gesamte (Rest)Array der Größe n laufen.
 - ▶ Daher ist der Aufwand in der Größenordnung von n^2 .



In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Bemerkungen zum Aufwand III

- ▶ Es stellt sich die folgende Frage:
 - ▶ Ist es möglich, schnellere Algorithmen zu entwerfen, indem man die **Ermittlung des Maximums** beschleunigt?
- ▶ Antwort:
 - ▶ **nein!**
 - ▶ Jeder Algorithmus, der mit Vergleichen zwischen Werten arbeitet, benötigt mindestens $n - 1$ Vergleiche um das Maximum von n Werten zu finden.
- ▶ Beschleunigung also nicht beim Auffinden des Maximums möglich ...

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortieren: Standardproblem der Informatik I

- ▶ Einfach zu verstehende Aufgabenstellung
- ▶ Tritt regelmäßig auf.
- ▶ Grundproblem: **internes Sortieren**
 - ▶ Zu sortierende Menge liegt unsortiert im Speicher vor. Abhängig von der Datenstruktur zur Mengendarstellung kann (im Prinzip) auf jedes Element zugegriffen werden.
 - ▶ Es existieren viele Algorithmen, die nach Algorithmusidee, nach Speicherplatz und Laufzeit (Berechnungsaufwand) unterschieden werden.
 - ❖ Wir brauchen noch ein formales Gerüst, um Speicherplatz und Berechnungsaufwand zu charakterisieren!

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Sortieren: Standardproblem der Informatik II

- ▶ Varianten:
 - ▶ **Externes Sortieren:** Daten liegen auf externem Speichermedium mit (sequentiell)em Zugriff.
 - ▶ **Einfügen** in sortierte Menge
 - ▶ **Verschmelzen** von sortierten Mengen
 - ▶ ...
- ▶ Im Folgenden: Effiziente Alternative zum letzten (naiven) Algorithmus: **Heapsort**
- ▶ Verwendung rekursiver Datenstrukturen für rekursive Algorithmen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive Datenstrukturen



Sortieren

Artikel im EINI-Wiki:

→ **Sortieren**

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- **Sortieren**
- Rekursive
Datenstrukturen

Rekursive Datenstrukturen

Rekursion ist nicht nur ein wichtiges Hilfsmittel für die Formulierung von Algorithmen, sondern auch für die Formulierung von **Datenstrukturen**.

▶ Beispiele

- ▶ Eine **Liste** ist ein Einzelelement, gefolgt von einer Liste, oder der leeren Liste.
- ▶ Eine **Menge** ist leer oder eine 1-elementige Menge vereinigt mit einer Menge.
- ▶ Oder **Bäume** (dazu im folgenden mehr)

EINI LogWing /
WiMa

Kapitel 5

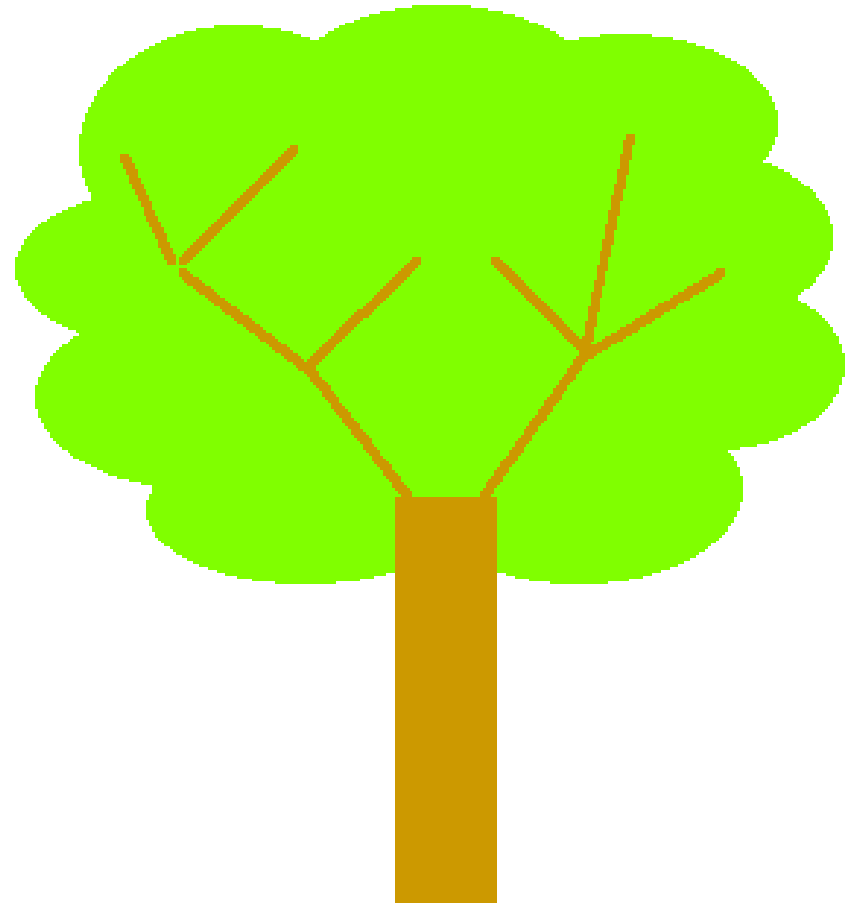
Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**

Idee vom Baum I

Künstlerisch



EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**

Idee vom Baum II

Abstrahiert 1

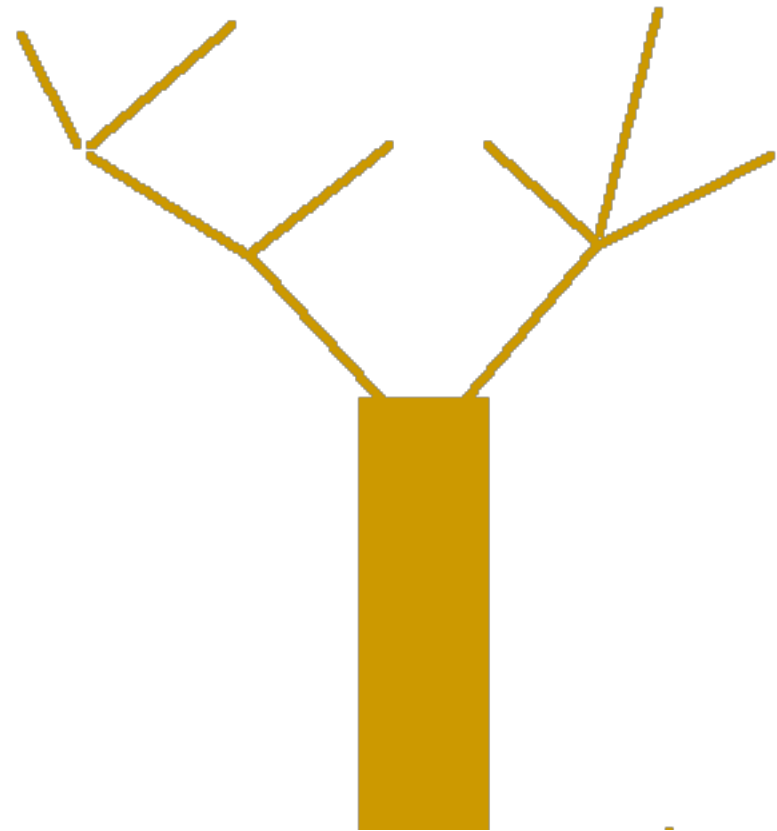
EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

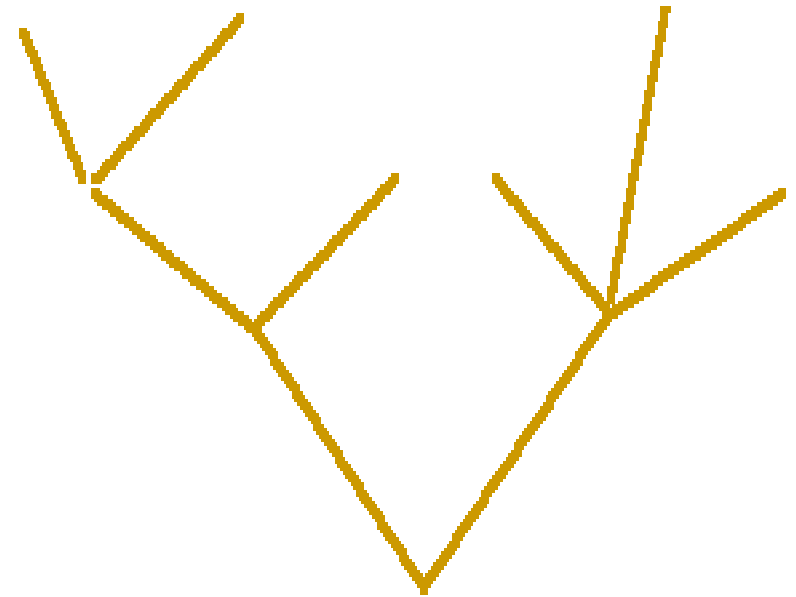
In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**



Idee vom Baum III

Abstrahiert 2



EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**

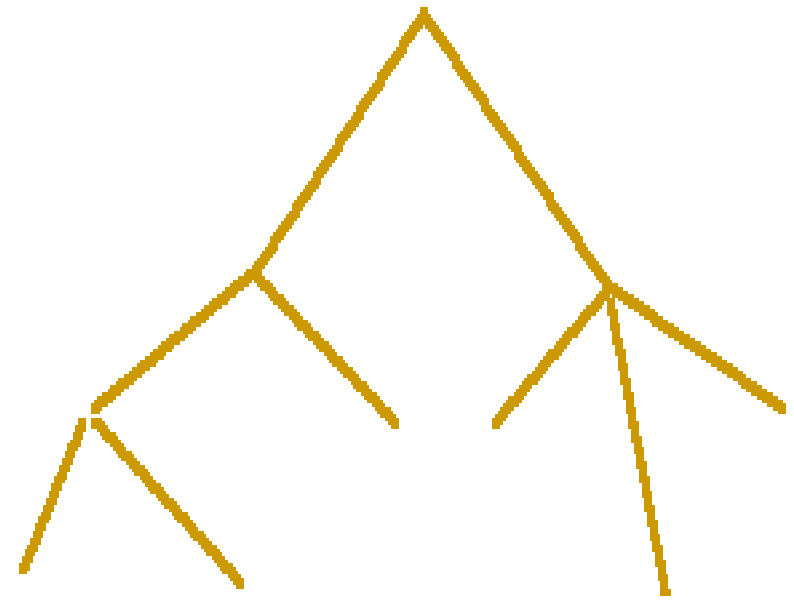
Idee vom Baum IV

Die Informatiksicht

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen



In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive Datenstrukturen**

Binärer Baum I

Definition: Binärer Baum

1. Der "leere" Baum \emptyset ist ein binärer Baum mit der Knotenmenge \emptyset .

2. Seien B_i binäre Bäume mit den Knotenmengen K_i , $i = 1, 2$. Dann ist auch $B = (w, B_1, B_2)$ ein binärer Baum mit der Knotenmenge

$$K = \{w\} \cup^* K_1 \cup^* K_2.$$

(\cup^* bezeichnet disjunkte Vereinigung.)

3. Jeder binäre Baum B lässt sich durch endlich häufige Anwendung von 1.) oder 2.) erhalten.

In diesem Kapitel:

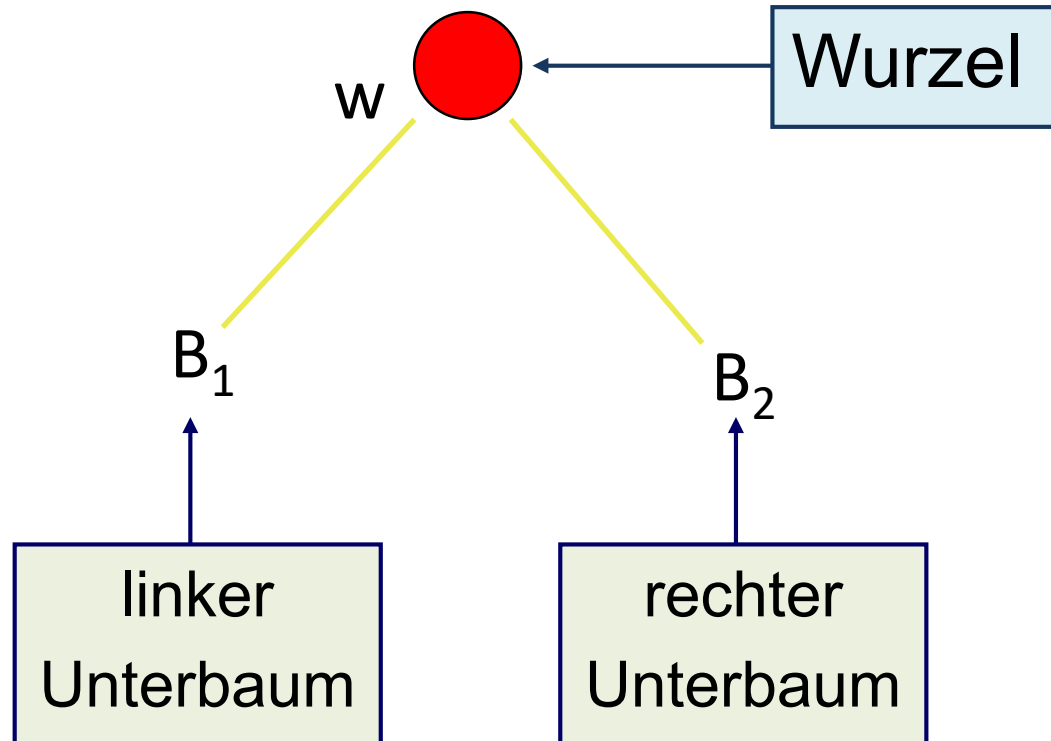
- Prolog
- Arrays
- Sortieren

Binärer Baum II

Sprech- und Darstellungsweisen (siehe Punkt 2):

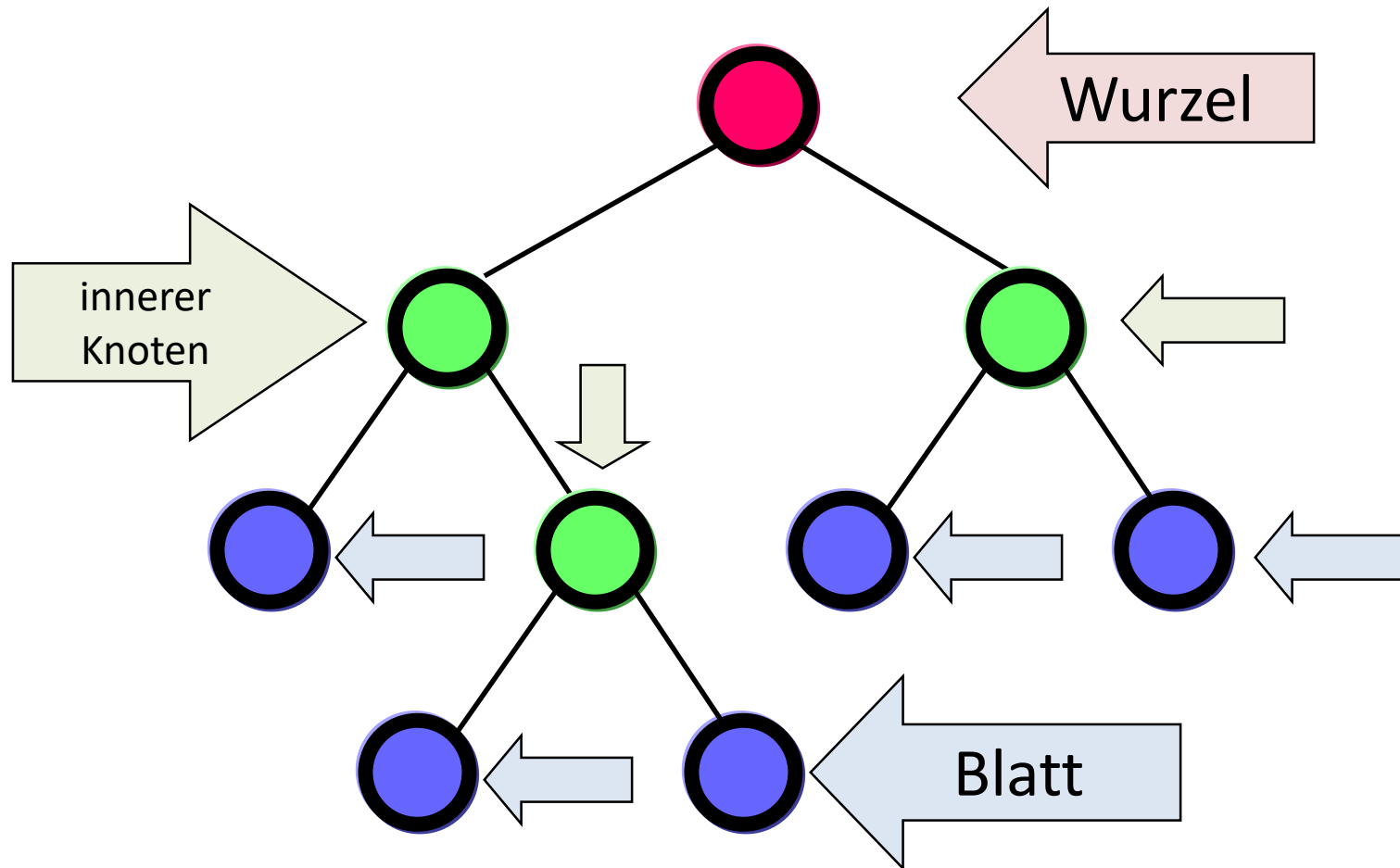
Sei $B = (w, B_1, B_2)$ binärer Baum:

w heißt **Wurzel**, B_1 linker und B_2 rechter **Unterbaum**.



- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**

Terminologie *Binäre Bäume*



EINI LogWing /
WiMa

Kapitel 5
Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**

Knotenmarkierter binärer Baum

► **Definition:** Sei M eine Menge.

(B, km) ist ein **knotenmarkierter binärer Baum**
(mit Markierungen aus M)

: \Leftrightarrow

1. B ist binärer Baum (mit Knotenmenge $K = K(B)$).

2. $km: K \rightarrow M$ Abbildung.

(Markierung/Beschriftung der Knoten $k \in K$ mit Elementen $m \in M$)

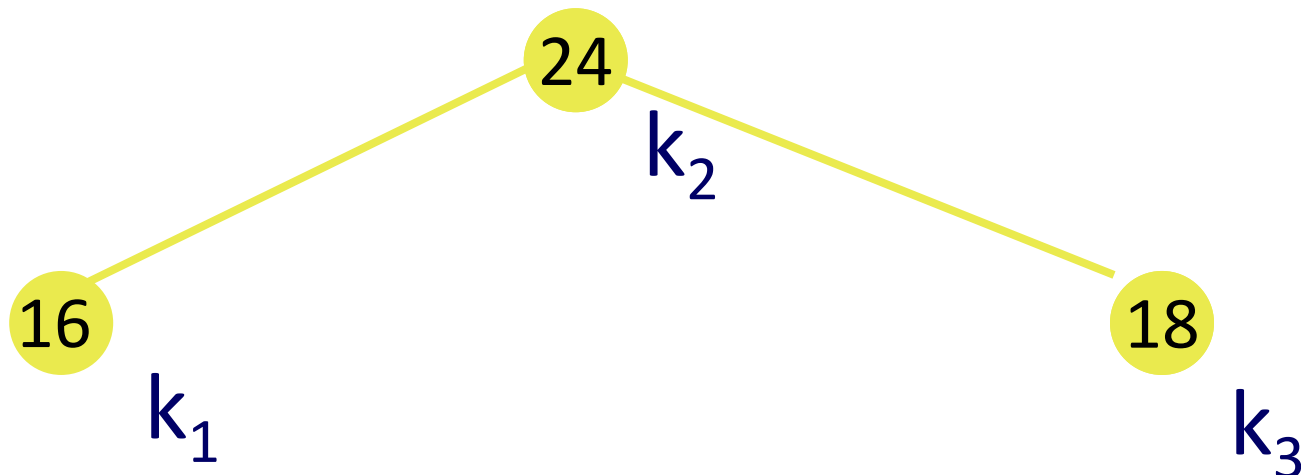
Jedem Knoten wird ein Element aus der Menge M zugeordnet.

Alternative: Markierung von Kanten.

Knotenmarkierter binärer Baum

Beispiel

- ▶ $M := \mathbb{Z}$, $\mathbb{Z} :=$ Menge der ganzen Zahlen
- ▶ Damit existiert auf M eine Ordnung!
- ▶ "Übliche" Darstellung der Knotenbeschriftung km durch "Anschreiben" der Beschriftung an/in die Knoten.



Definition: Heap

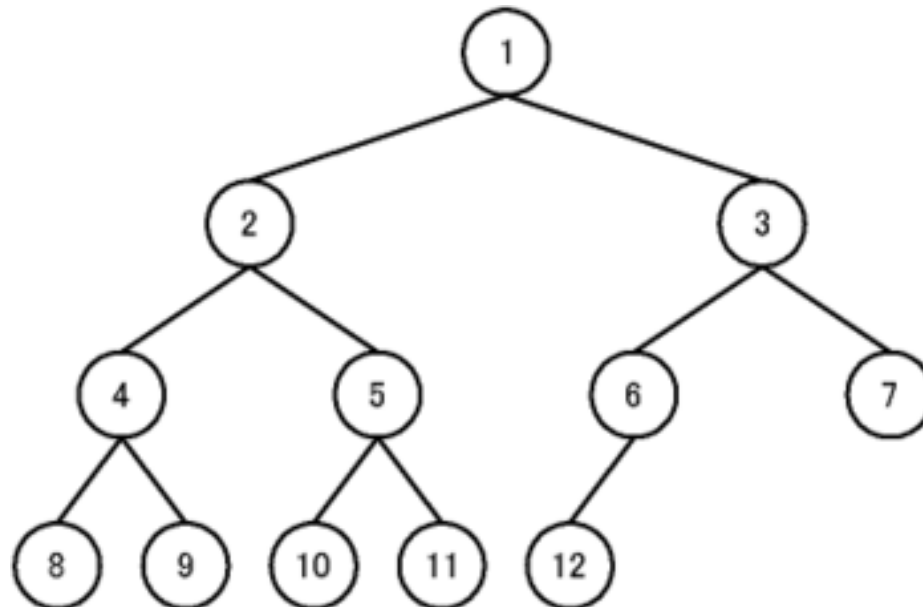
- ▶ Ein **Heap** (Haufen) ist ein knotenmarkierter binärer Baum, für den gilt:
 - ▶ Die Markierungsmenge ist geordnet.
 - ▶ Der binäre Baum ist links-vollständig.
 - ▶ Die Knotenmarkierung der Wurzel ist kleiner oder gleich der Markierung des linken bzw. rechten Sohnes (sofern vorhanden).
 - ▶ Die Unterbäume der Wurzel sind Heaps.
- ▶ An der Wurzel steht das kleinste (eines der kleinsten) Element(e).

In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**

Beispiel: Heap

- ▶ Binärbaum
- ▶ Alle Ebenen, bis auf letzte, vollständig gefüllt
- ▶ Links-vollständig gefüllt
- ▶ Knotenmarkierung der Wurzel kleiner als die der Kinder



In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**



Artikel im EINI-Wiki:

→ Baum

→ Heap

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**

Übersicht

Begriffe

- ✓ Spezifikationen, Algorithmen, formale Sprachen
- ✓ Programmiersprachenkonzepte
- ✓ Grundlagen der imperativen Programmierung

➤ Algorithmen und Datenstrukturen

- ✓ Felder
- ✓ Sortieren
- ✓ Rekursive Datenstrukturen (Baum, binärer Baum, Heap)
- Heapsort

▶ Objektorientierung

- ▶ Einführung
- ▶ Vererbung
- ▶ Anwendung

EINI LogWing /
WiMa

Kapitel 5

Algorithmen und
Datenstrukturen

In diesem Kapitel:

- Prolog
- Arrays
- Sortieren
- **Rekursive
Datenstrukturen**



Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- ▶ Nächste Vorlesung – WiMa 13.12.2018, 08:15
- ▶ Nächste Vorlesung – LogWing 14.12.2018, 08:15