EINI LW

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 11/12

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@udo.edu
http://ls1-www.cs.uni-dortmund.de

technische universität dortmund

Dr. Lars Hildebrand

- **Kapitel 5**
  Algorithmen und Datenstrukturen
  - Konstruktion von Datentypen: Arrays
  - Algorithmen: Sortieren

- **Unterlagen**
  - Gumm/Sommer, Kapitel 2.7 & 2.8
  - Echtle/Goedicke, Einführung in die Programmierung mit Java, dpunkt Verlag, Kapitel 4
  - Doberkat/Dißmann, Einführung in die objektorientierte Programmierung mit Java, Oldenbourg, Kapitel 3.4 & 4.1

Übersicht

Begriffe

- Spezifikationen, Algorithmen, formale Sprachen, Grammatik
- Programmiersprachenkonzepte
- Grundlagen der Programmierung

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

- Objektorientierung
  - Einführung
  - Vererbung
  - Anwendung

Gliederung

- **Arrays**
  - 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 2 Teilmengen anhand eines Pivotelementes

technische universität dortmund

Dr. Lars Hildebrand

Motivation:

Schleifen erlauben die Verarbeitung mehrerer Daten auf einen „Schlag“

- Eine Entsprechung 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, Felder

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

Arrays sind Variablen ...

Dr. Lars Hildebrand

- ... 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`

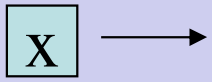
- Legen allerdings anders als bei `int z;` noch keinen Speicherplatz fest

technische universität dortmund

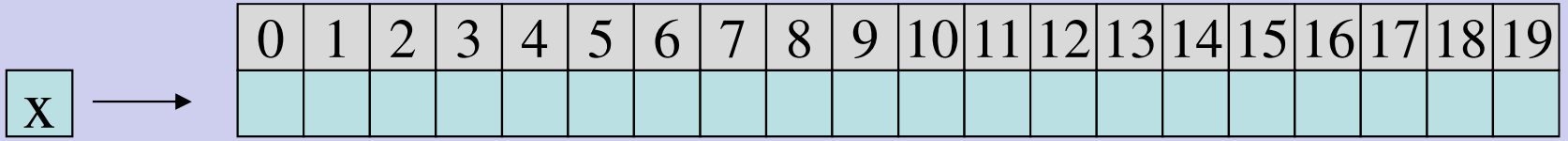
Dr. Lars Hildebrand

Deklarationen einer Array-Variablen legt nur einen Verweis auf ein Array fest, Dimensionierung notwendig!

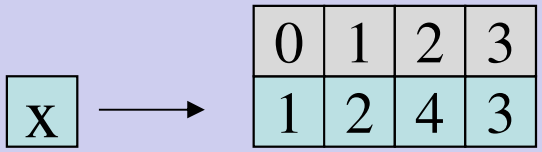
```
int [] x;
```




```
x = new int [20];
```




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



technische universität dortmund

Dr. Lars Hildebrand

Deklarationen einer Array-Variablen legt nur einen Verweis auf ein Array fest, Dimensionierung notwendig!

- 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

Zuweisungen

Dr. Lars Hildebrand

- 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

technische universität dortmund

Dr. Lars Hildebrand

Array-Größe ist fest nach Ausführung des new-Operators

- 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!
```

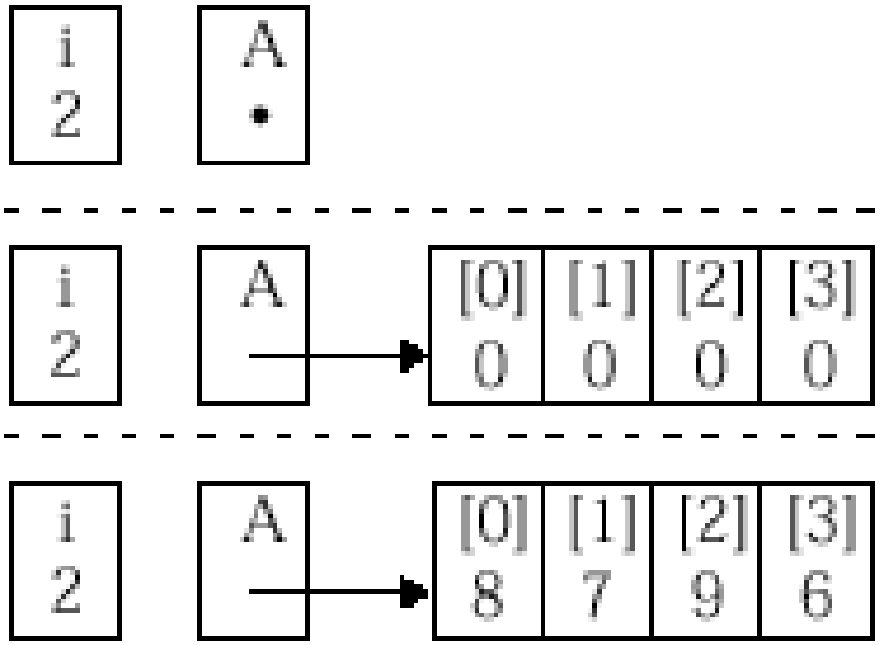
Zuweisung von Array-Variablen ... Beispiel (1)

Dr. Lars Hildebrand

```
int i = 2;
int [] A;

A = new int [4];

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



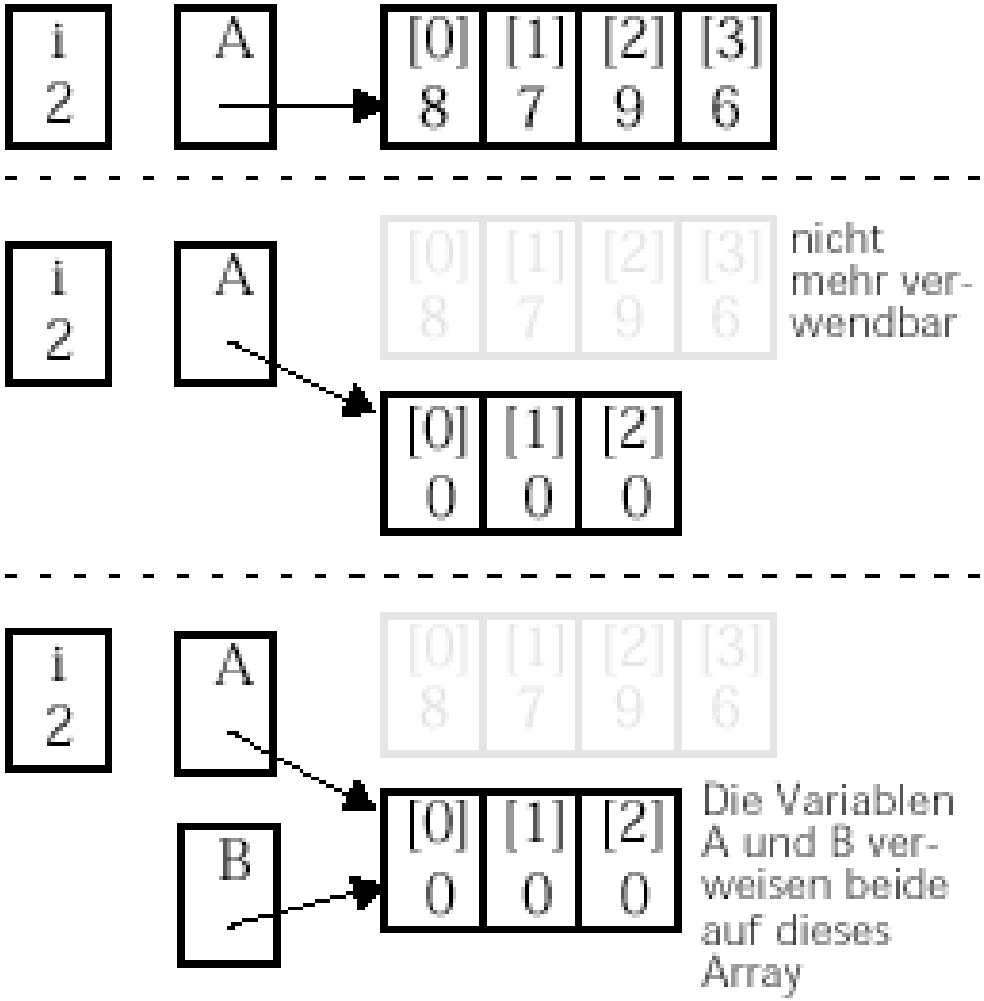
Zuweisung von Array-Variablen ... Beispiel (2)

Dr. Lars Hildebrand

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

A = new int [3];

int [] B;
B = A;
```



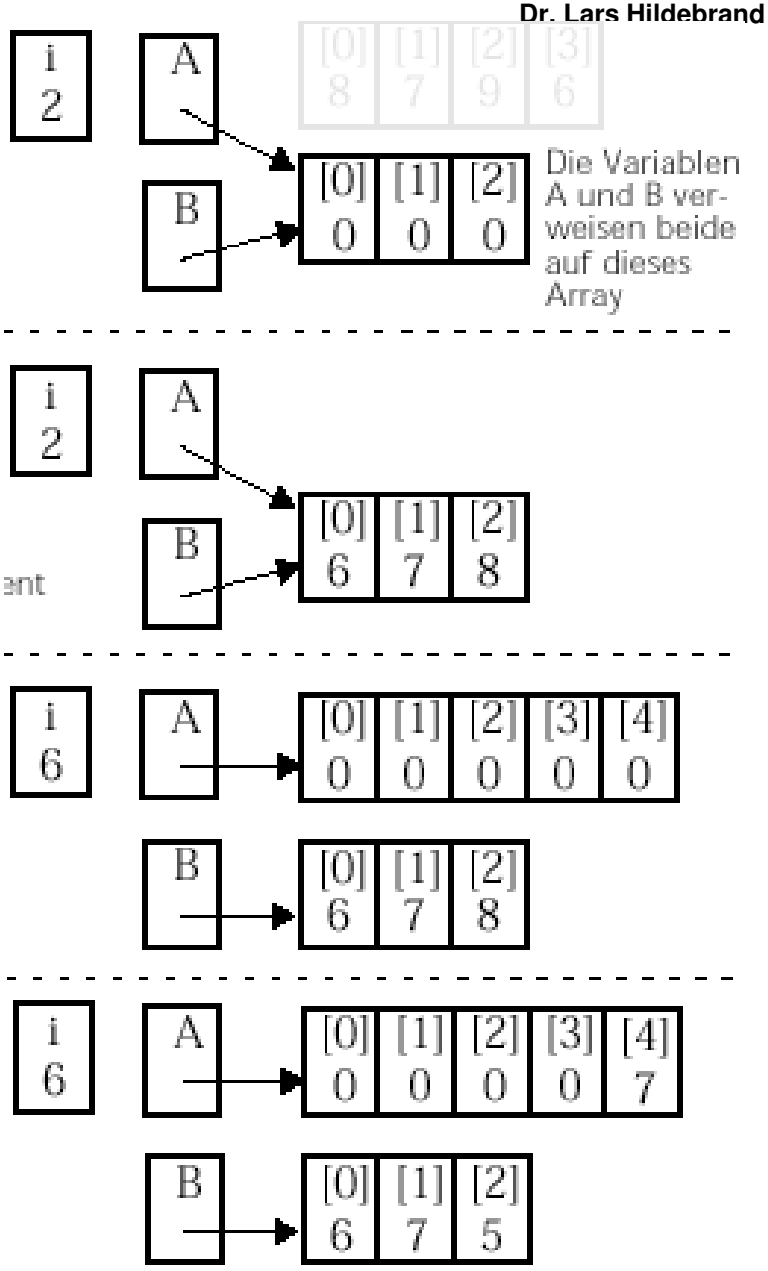
Zuweisung von Array-Variablen ... Beispiel (3)

Dr. Lars Hildebrand

```
....
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;
```



Dynamische Größe von Arrays

Dr. Lars Hildebrand

Falls Größe eines Arrays zum Zeitpunkt der Erstellung nicht bekannt:

- 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

technische universität dortmund
Dr. Lars Hildebrand

strenges Typssystem

- 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 muss übereinstimmen

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

Arrays: Internes Sortieren

Dr. Lars Hildebrand

- 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 **geringen** Speicherplatzbedarf aber **hohe** Laufzeit
- Verfahren:
  - Vertausche Elemente der Folge solange, bis sie in der richtigen Reihenfolge sind
- Hier wird als Speicherungsstruktur ein Array benutzt

Nun die eigentliche Sortierung ...

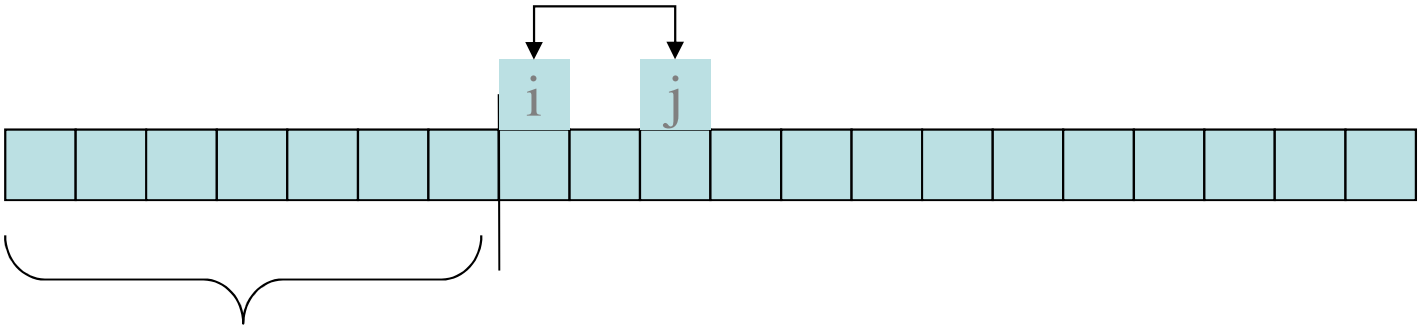
Dr. Lars Hildebrand

Die Idee:

- Ausgangspunkt: Element an der Stelle i hat den richtigen Platz
  - alles vor der Stelle i ist bereits sortiert

- dann sollte eigentlich für alle Elemente an den Stellen ab i gelten, das die Elemente größer sind

- Wenn diese Bedingung nicht gilt: vertausche die Elemente an den Stellen i und der Fehlerstelle j



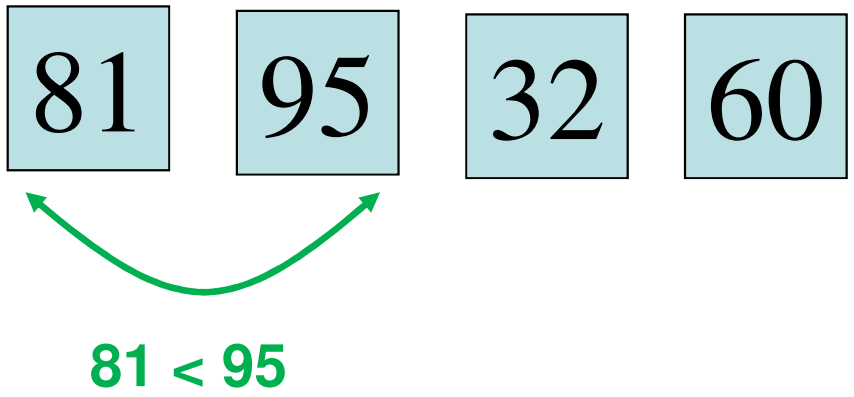
bereits sortiert

Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

Beispiel

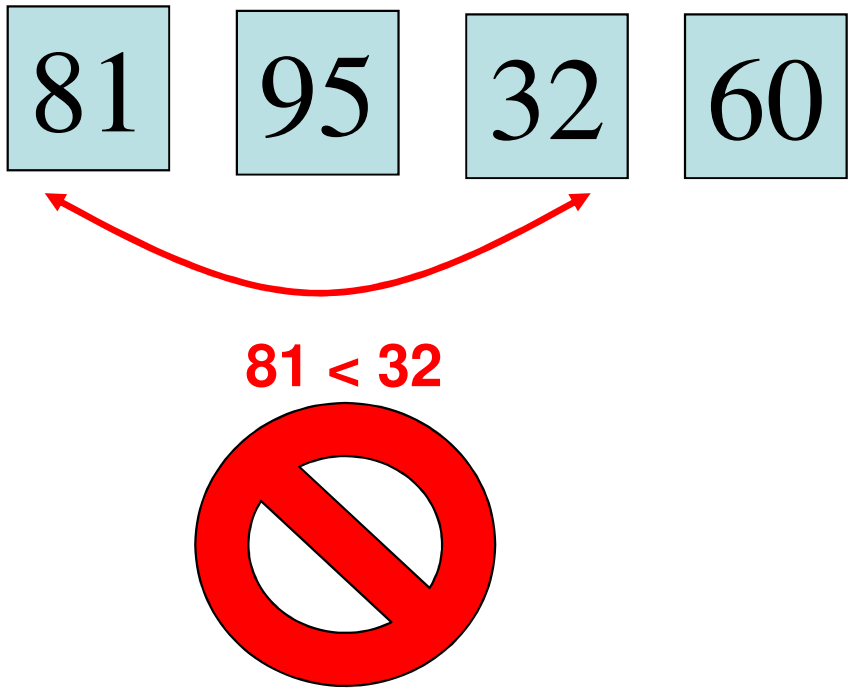
- Ausgangspunkt



Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

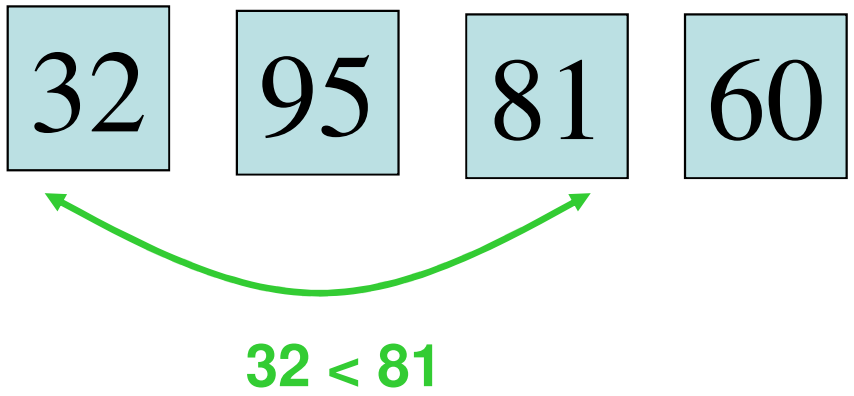
Beispiel



Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

Beispiel

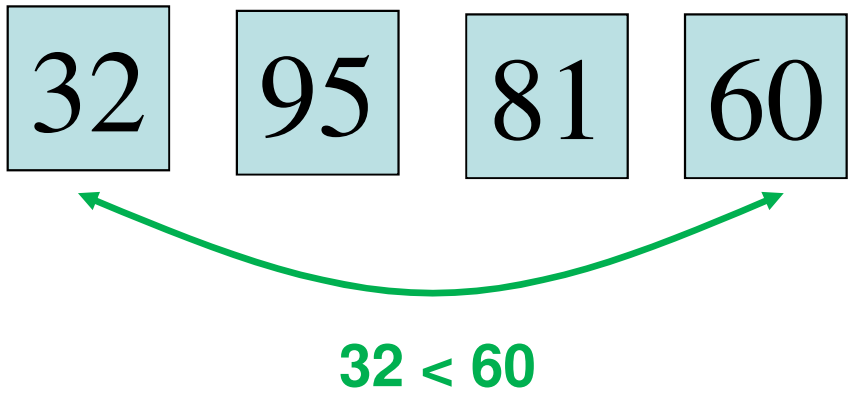


Nun die eigentliche Sortierung ...

technische universität dortmund

Dr. Lars Hildebrand

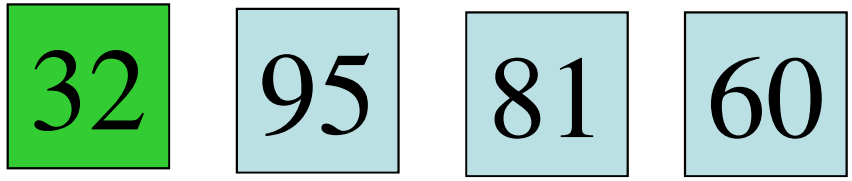
Beispiel



Nun die eigentliche Sortierung ...

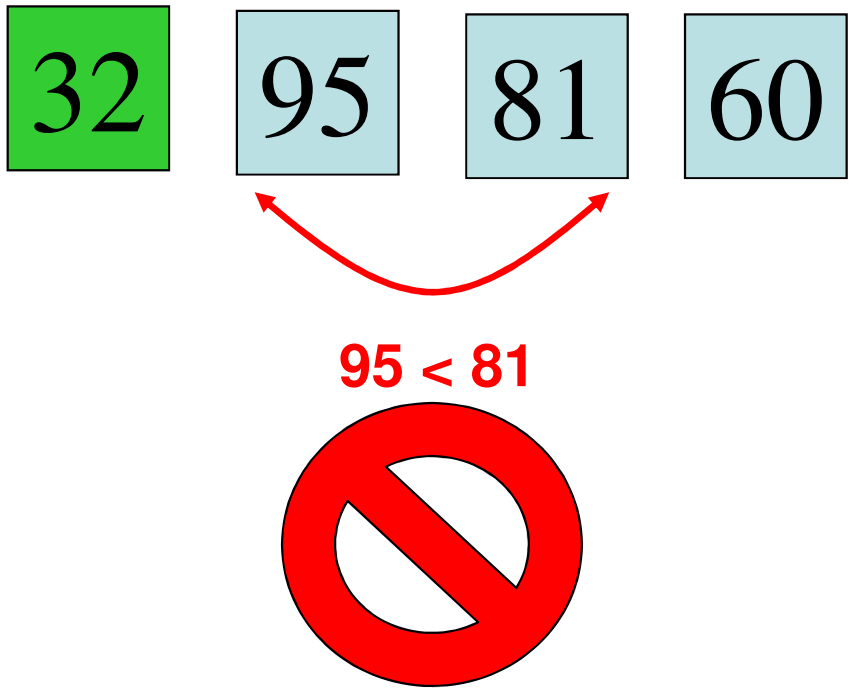
Dr. Lars Hildebrand

Beispiel



Nun die eigentliche Sortierung ...

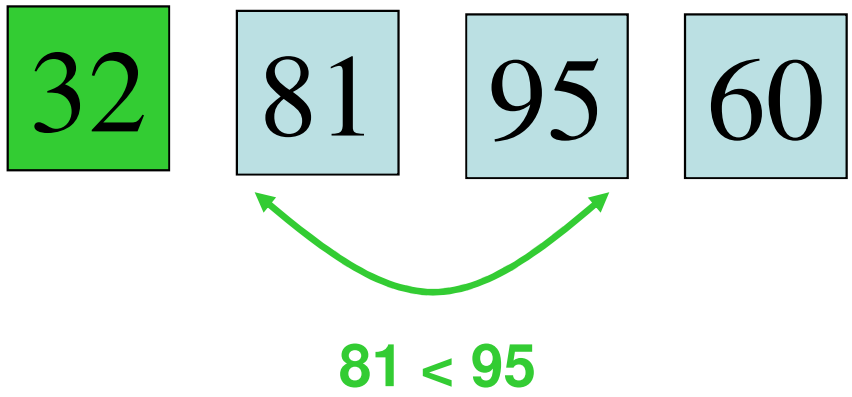
Beispiel



Nun die eigentliche Sortierung ...

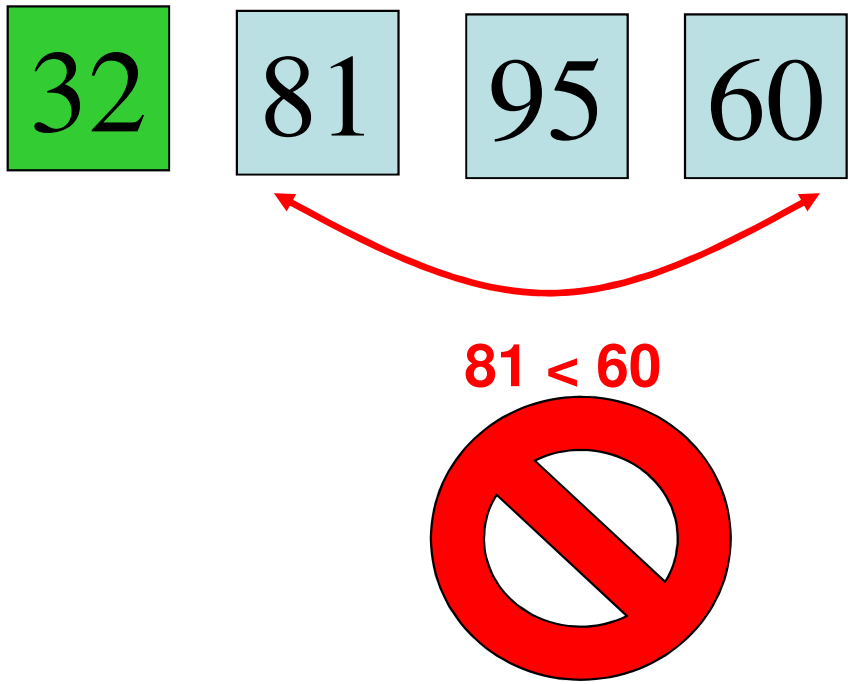
Dr. Lars Hildebrand

Beispiel



Nun die eigentliche Sortierung ...

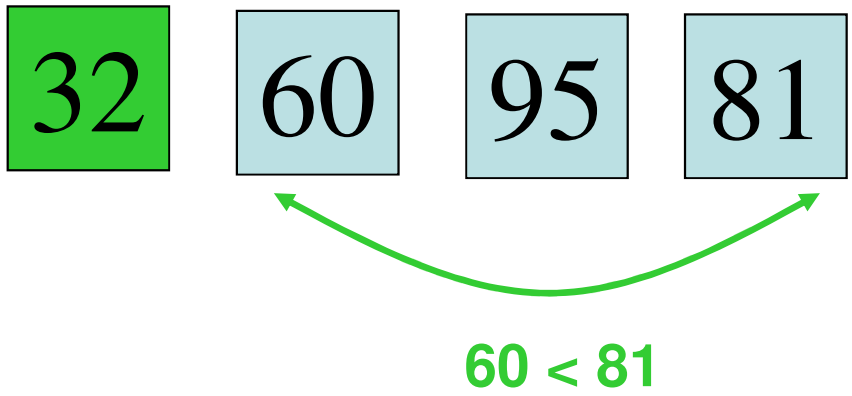
Beispiel



Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

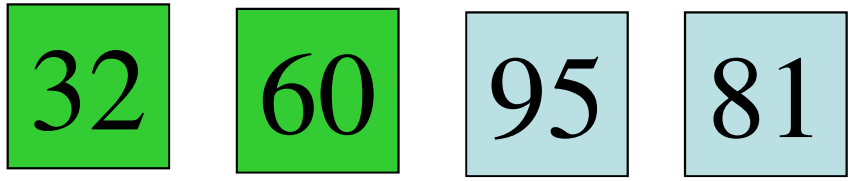
Beispiel



Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

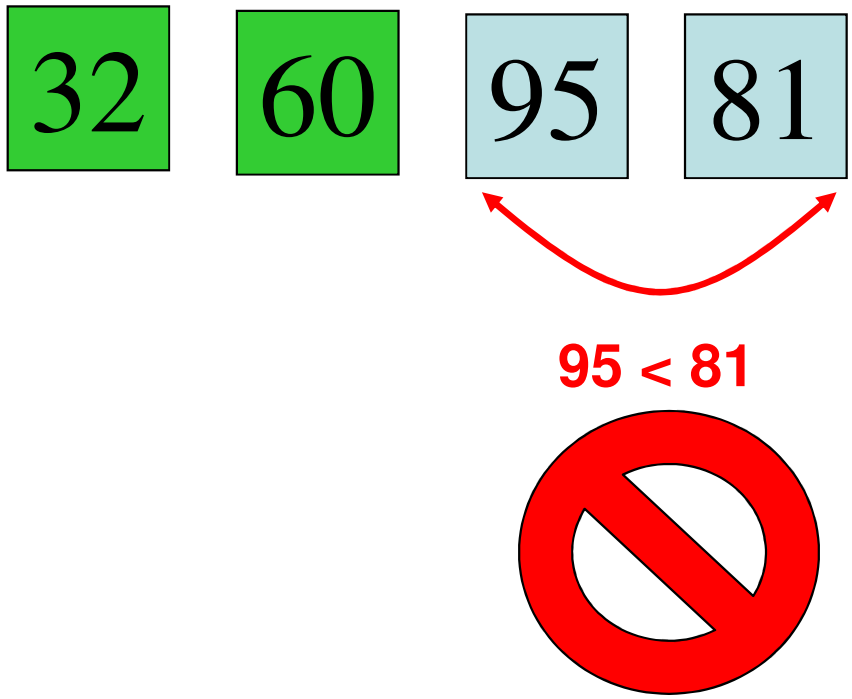
Beispiel



Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

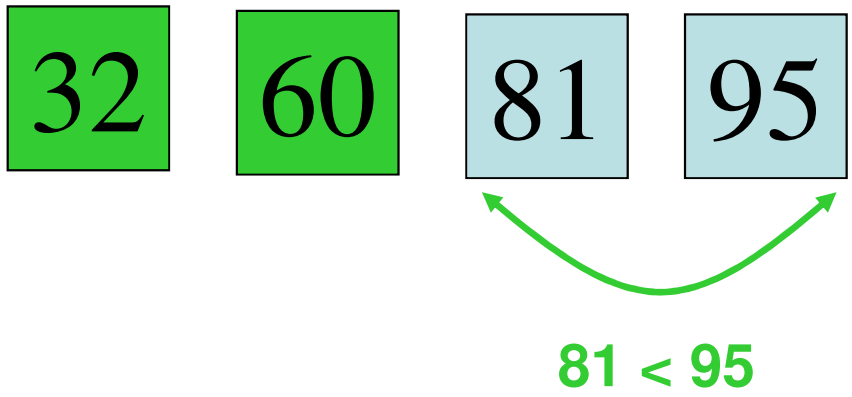
Beispiel



Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

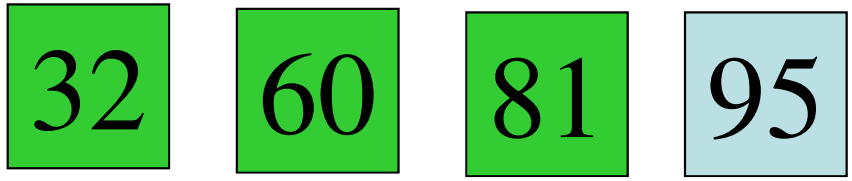
Beispiel



Nun die eigentliche Sortierung ...

Dr. Lars Hildebrand

Beispiel

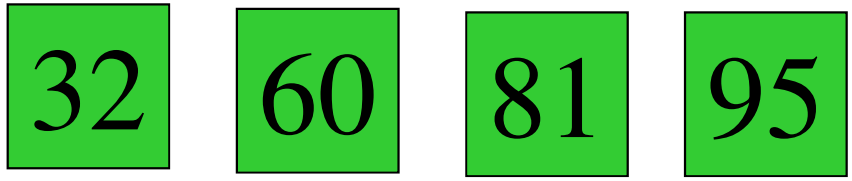


Nun die eigentliche Sortierung ...

technische universität dortmund

Dr. Lars Hildebrand

Beispiel



Zunächst das 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();

}
```

Nun die 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 alles noch ausgegeben

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

Der Ablauf noch einmal tabellarisch

Dr. Lars Hildebrand

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	

Gesamtes Programm

Dr. Lars Hildebrand

```
import java.util.Scanner;

public class A533
{
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

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

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

    for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
        if (a [i] > a [j]) {
          int z = a [i];
          a [i] = a [j];
          a [j] = z;
        }
      }
    }

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

Alternativen ?

- Könnte man die Algorithmus Idee 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

Bemerkungen zum Aufwand

Dr. Lars Hildebrand

- 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 vergröbernden Abschätzung
  - sogenannte O-Notation
- Diese Abschätzung wird in der Regel von der Größe des Problems bestimmt: hier die Anzahl der zu sortierenden Elemente

Bemerkungen zum Aufwand

Dr. Lars Hildebrand

- Obiges Sortierverfahren:
  - zwei geschachtelte FOR-Schleifen,
  - die im schlimmsten Fall über das gesamte Array der Größe n laufen
  - Daher ist der Aufwand in der Größenordnung von n^2

Bemerkungen zum Aufwand

Dr. Lars Hildebrand

- 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 im Auffinden des Maximums möglich …

technische universität dortmund
Dr. Lars Hildebrand

Sortieren: Standardproblem der Informatik

- 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 !

technische universität dortmund
Dr. Lars Hildebrand

Sortieren: Standardproblem der Informatik

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

technische universität dortmund

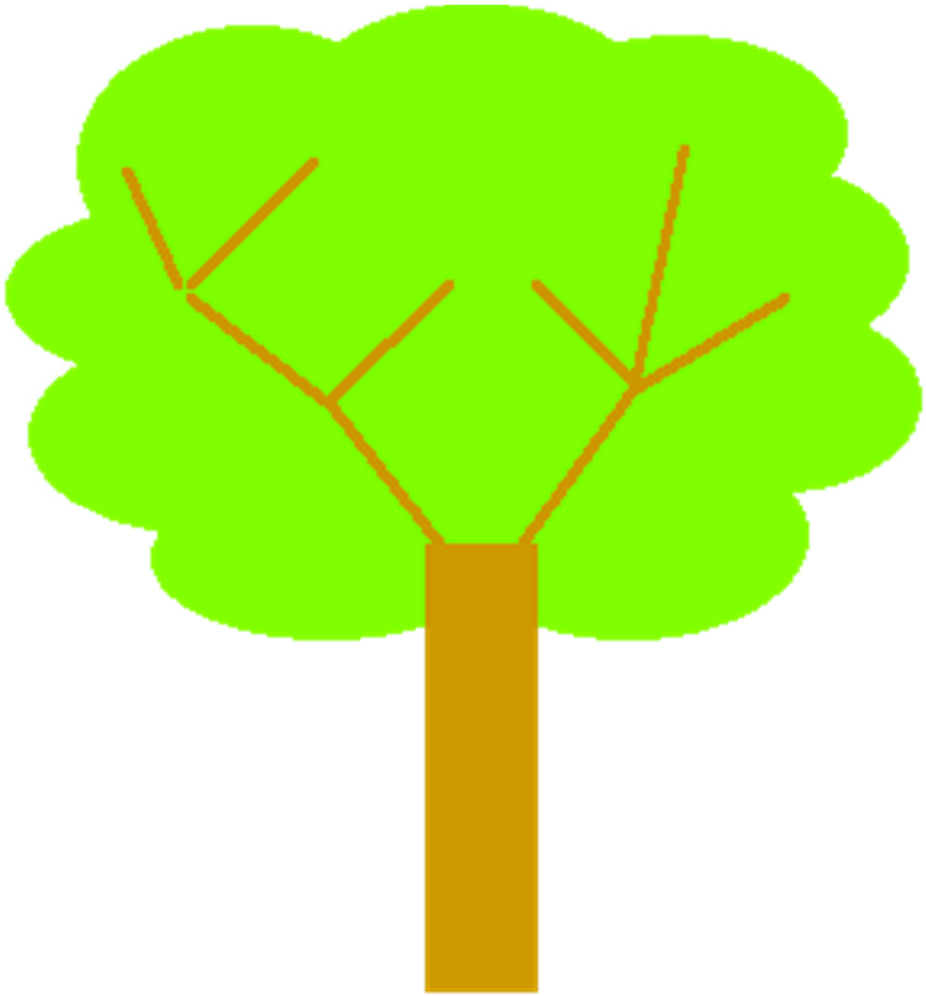
Dr. Lars Hildebrand

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 die leere Liste.
  - Eine **Menge** ist leer oder eine 1-elementige Menge vereinigt mit einer Menge.
  - Oder **Bäume** (dazu im folgenden mehr).

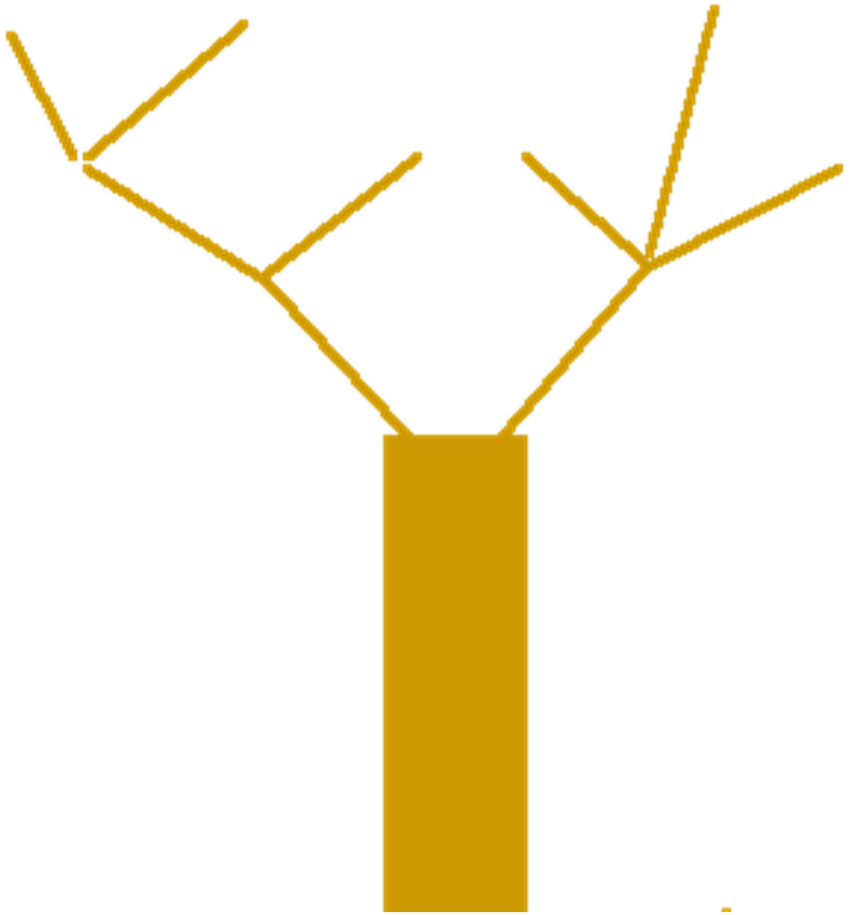
Idee vom Baum

Künstlerisch



Idee vom Baum

Abstrahiert 1



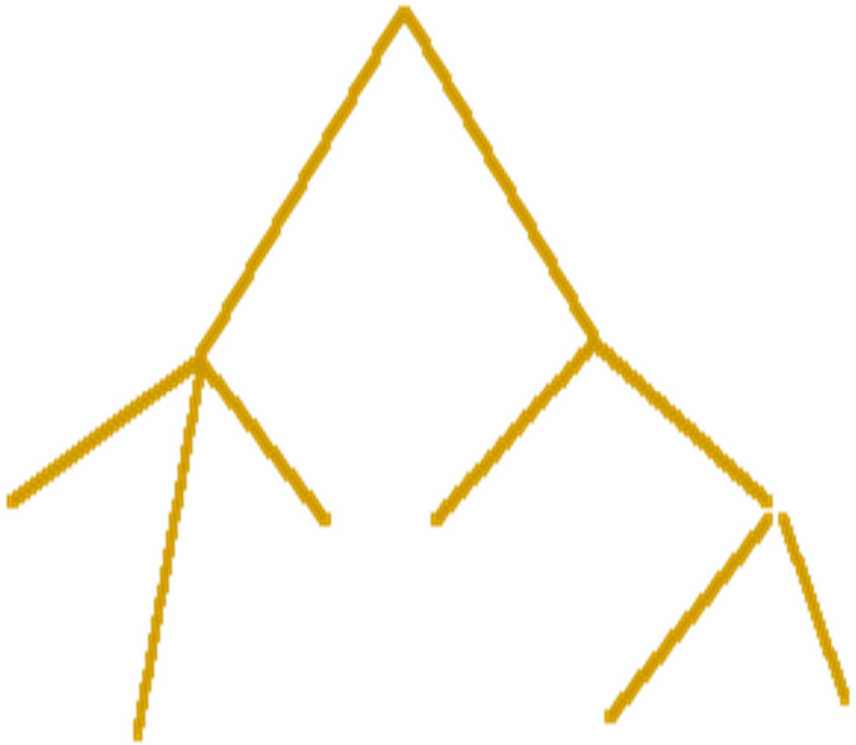
technische universität dortmund

Dr. Lars Hildebrand

Abstrahiert 2

Idee vom Baum

Die Informatiksicht



technische universität dortmund

Dr. Lars Hildebrand

Definition: Binärer Baum

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

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.)

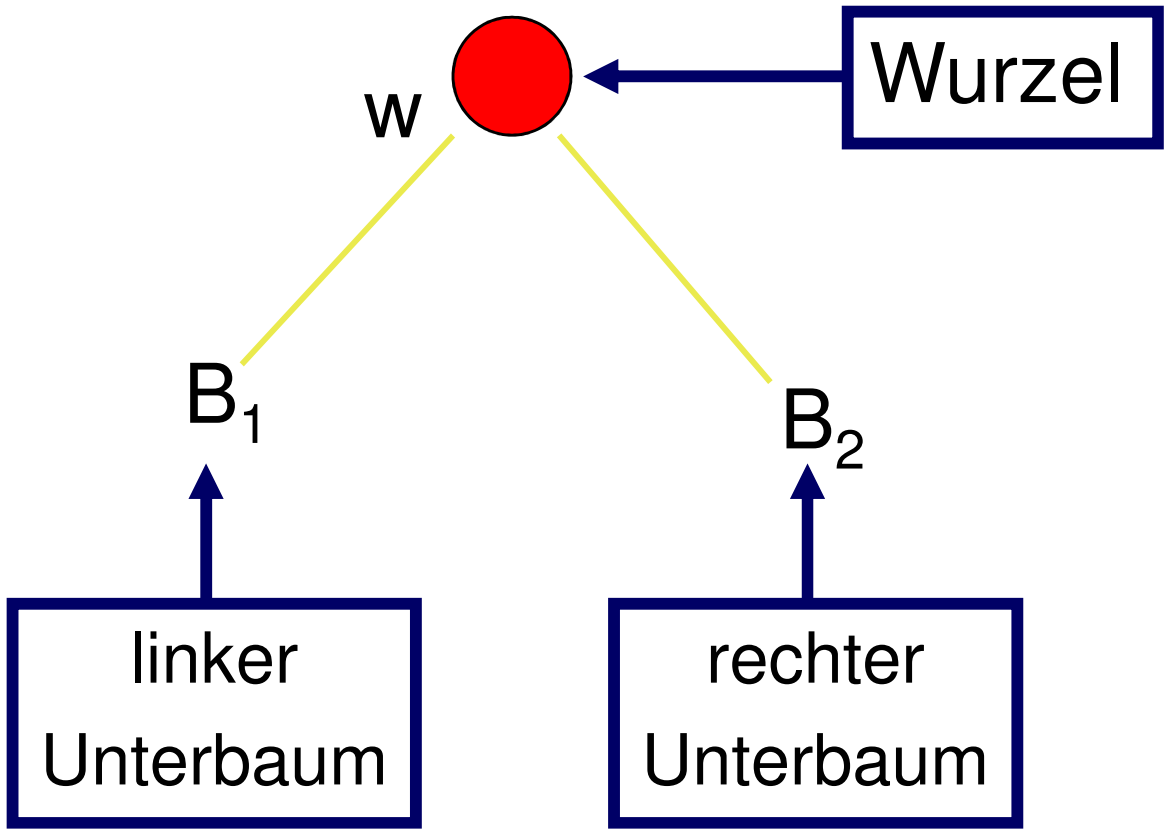
3. Jeder binäre Baum B lässt sich durch endlich häufige Anwendung von 1.) oder 2.) erhalten.

technische universität dortmund
Dr. Lars Hildebrand

- **Sprech-/Darstellungsweisen** (im Falle 2.)):

  Sei B = (w, B_1, B_2) binärer Baum

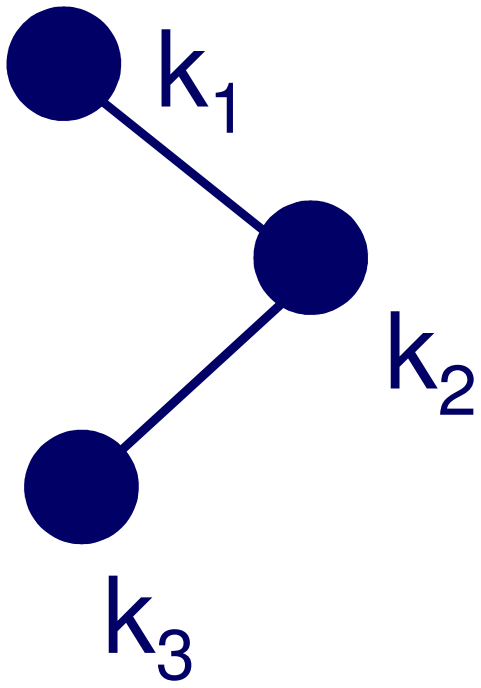
  w heißt **Wurzel**, B_1 linker und B_2 rechter **Unterbaum**.



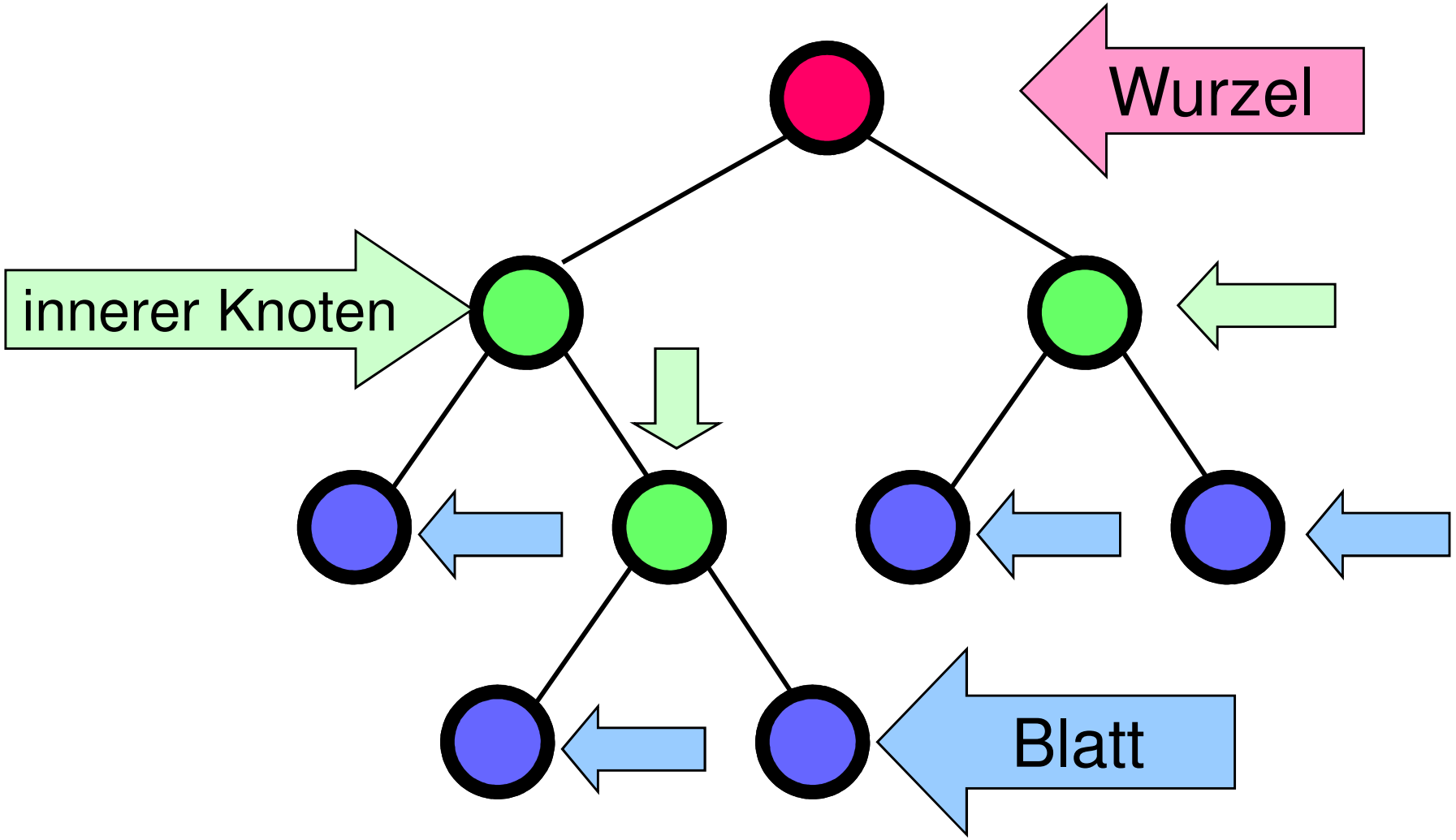
technische universität dortmund
Dr. Lars Hildebrand

- Darstellung eines Beispiels nach Definition:

$$B_1 = (k_1, \varnothing, (k_2, (k_3, \varnothing, \varnothing), \varnothing)).$$



Terminologie *Binäre Bäume*



Knotenmarkierter binärer Baum

- **Definition**: Sei M eine Menge.

  (B, *km*) ist ein **knotenmarkierter** binärer **Baum**

  (mit Markierungen aus M)

  :⇔

  1. B ist binärer Baum (mit Knotenmenge K = K(B))
  2. *km*: K --> M Abbildung.

     (Markierung/Beschriftung der Knoten k ∈ K mit Elementen m ∈ M)

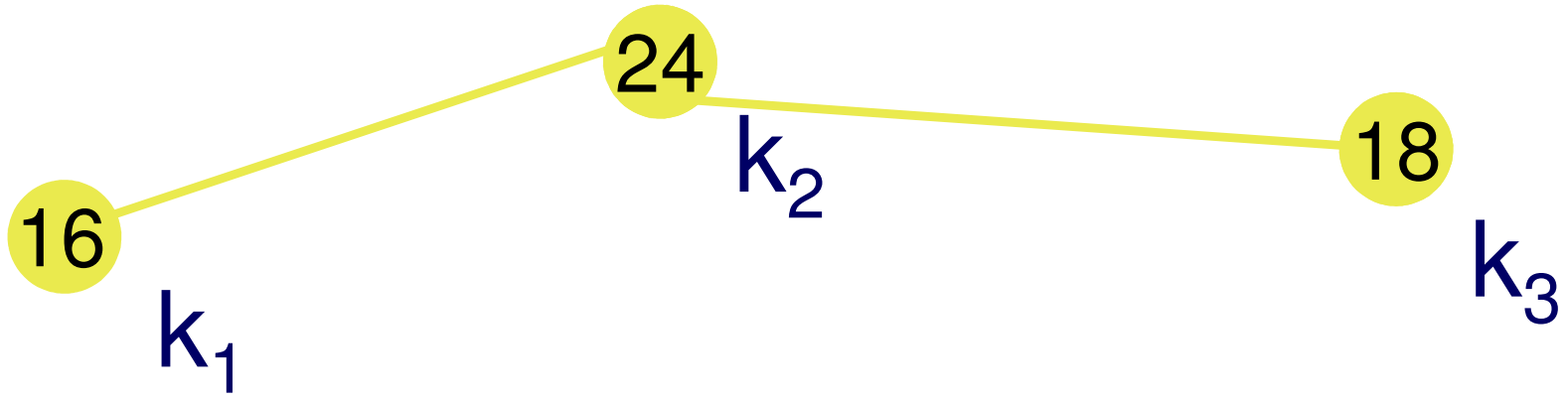
Jedem Knoten wird ein Element aus der Menge M zugeordnet.

Alternative: Markierung von Kanten.

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

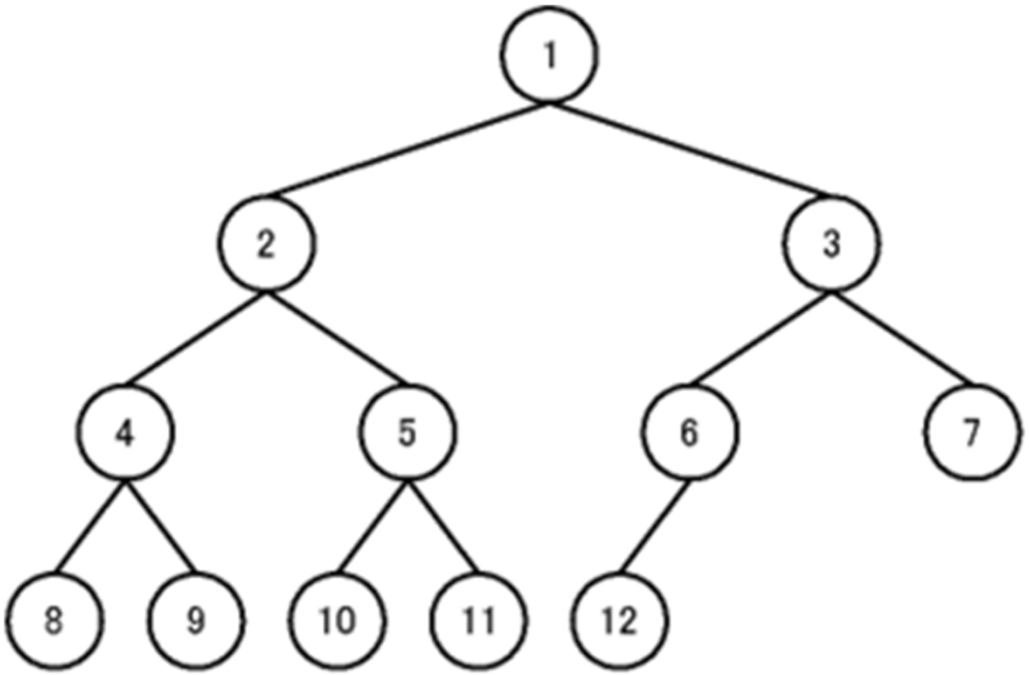
- Ein **Heap** (Haufen) ist ein knotenmarkierter binärer Baum, für den gilt:
  - Die Markierungsmenge ist geordnet.
  - Der binäre Baum ist links-vollständig.
  - Die Knotenmarkierung der Wurzel ist kleiner oder gleich der Markierung des linken resp. rechten Sohnes (, sofern vorhanden).
  - Die Unterbäume der Wurzel sind Heaps.

- An der Wurzel steht das kleinste (eines der kleinsten) Element(e).

Beispiel: Heap

Dr. Lars Hildebrand

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



Übersicht

Begriffe

- Spezifikationen, Algorithmen, formale Sprachen, Grammatik
- Programmiersprachenkonzepte
- Grundlagen der Programmierung

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

- Objektorientierung
  - Einführung
  - Vererbung
  - Anwendung

Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- Nächste Vorlesung

16.12.2011, 08:30
Campus Nord
EF-50, HS 1