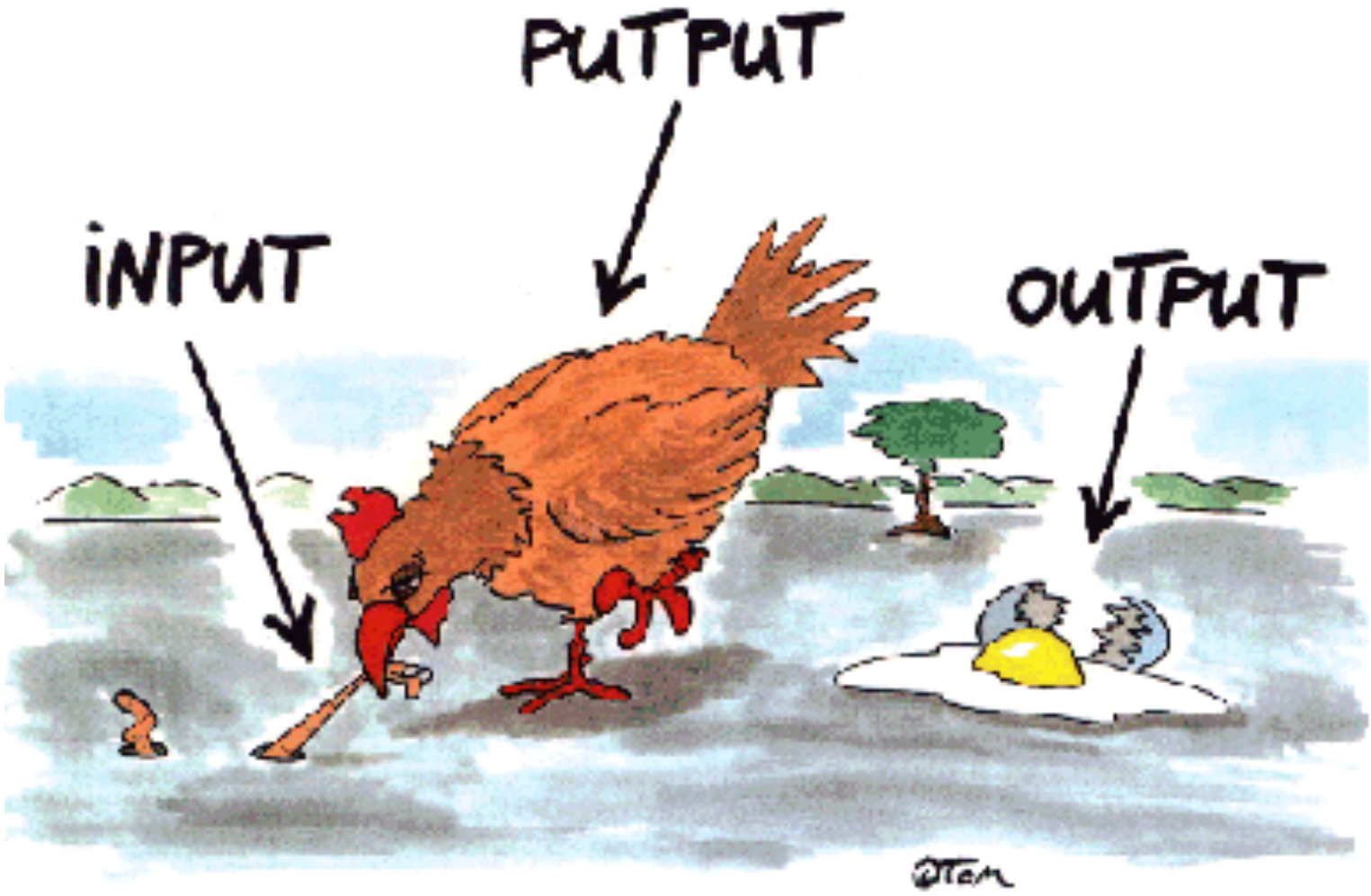
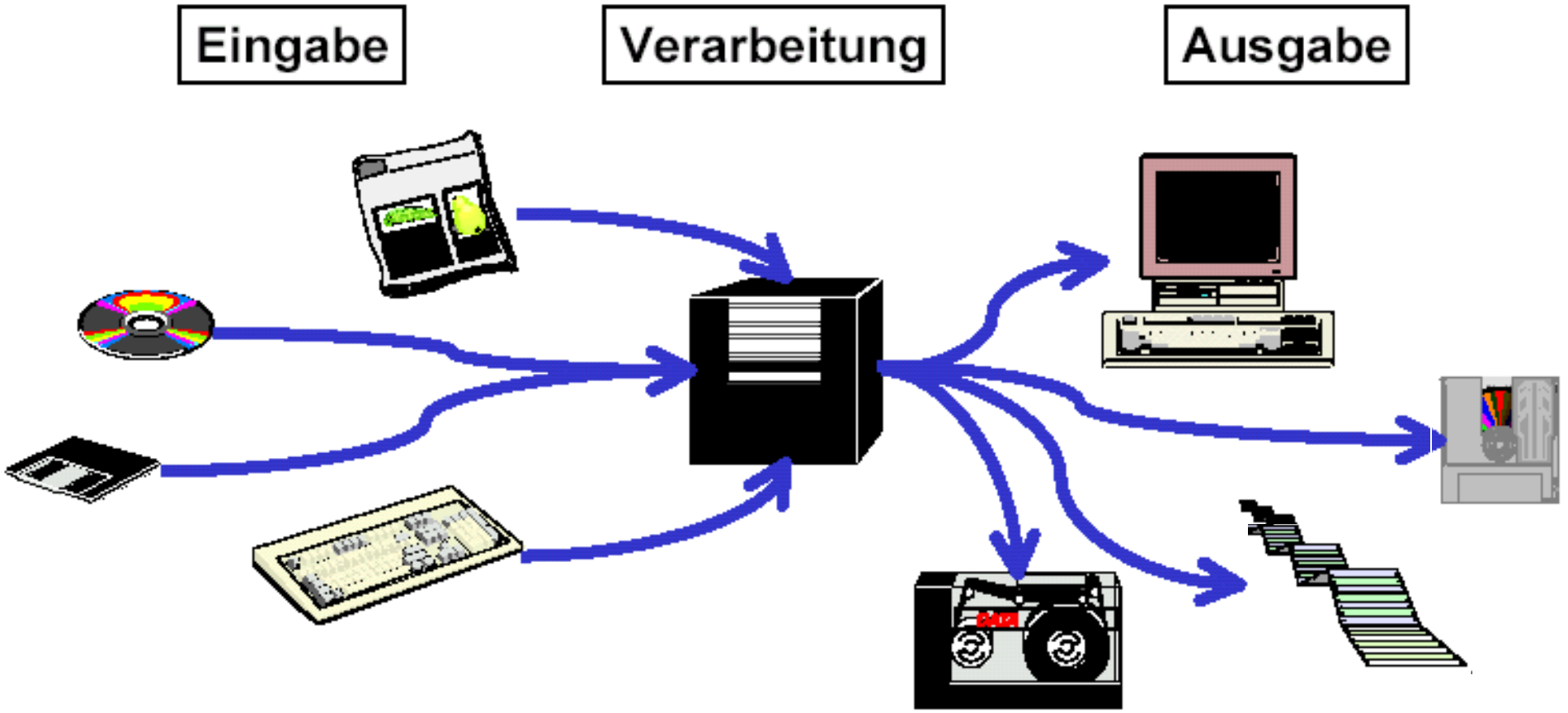
Grundprinzip EVA: Input – Output

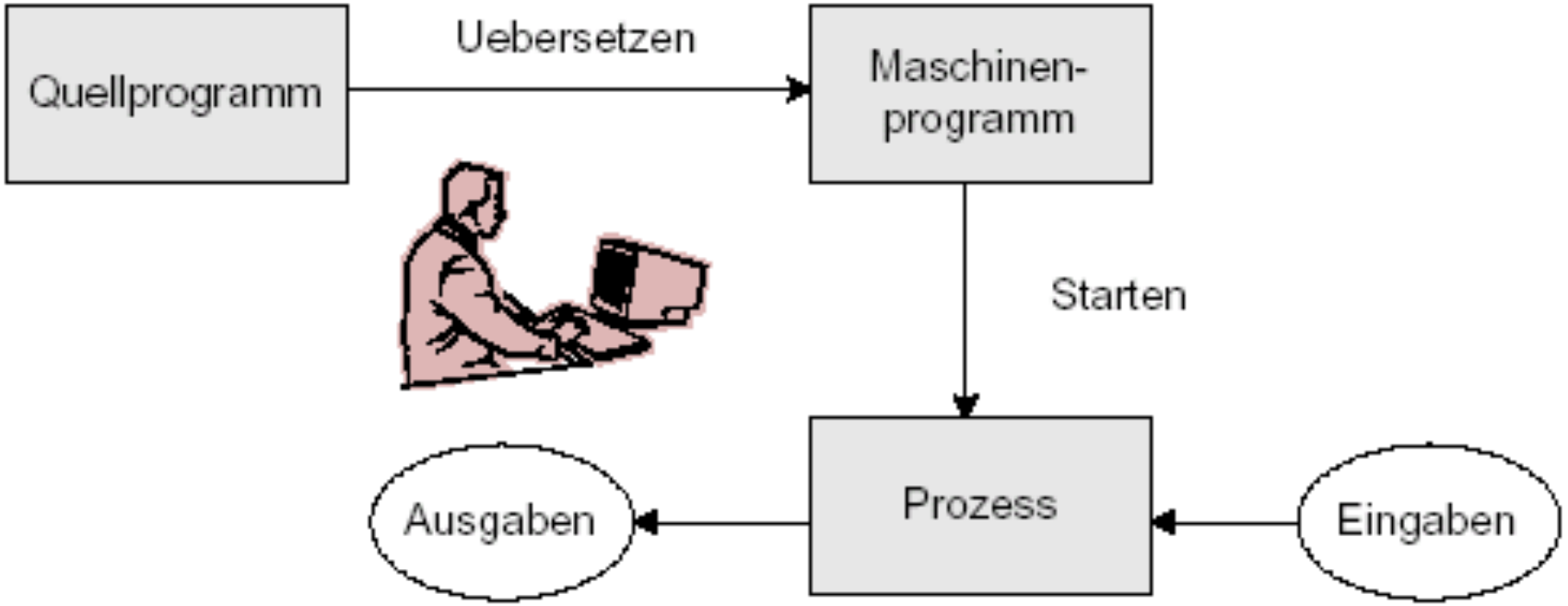
Hochschule Ludwigshafen am Rhein



Das Grundprinzip: EVA



Programmierung: Ablauf



Programm:

- Eine zur Lösung einer Aufgabe vollständige Anweisung an den Computer
- Der Vorgang zur Erstellung einer derartigen Anweisung heißt **Programmieren**.

Quelle: Schlichter, Einführung in die Informatik

Algorithmus

Hochschule Ludwigshafen am Rhein

- Verfahrensvorschrift zur Lösung eines Problems
  - Exakt
  - vollständig formuliert
  - schrittweise ausführbar
  - effektiv ausführbar
  - endlich
- Formulierung kann in natürlicher oder formaler Sprache vorliegen
- Die Ausführung kann durch Menschen oder eine Maschine erfolgen

Algorithmus - Beispiel

Hochschule Ludwigshafen am Rhein

- Bestimme das Alter der ältesten Person im Raum
  - Gehe zur ersten Person;
  - Frage Person nach dem Alter;
  - Merke das Alter;
  - Wiederhole bis alle Personen gefragt sind
    - gehe zur nächsten Person;
    - frage nach dem Alter;
    - wenn das Alter größer als das gemerkte Alter, dann merke Dir das neue Alter;
  - Das Alter der ältesten Person ist "gemerktes Alter";

Algorithmus

Hochschule Ludwigshafen am Rhein

- wesentliche Elemente
  - Sequenz von Anweisungen
  - Variablen zum Speichern von Daten
  - Wertzuweisungen
  - Wiederholung (Schleife)
  - Alternativen
- Damit können alle Algorithmen beschrieben werden
- wesentliche Fragen
  - Darstellung
  - Korrektheit
  - Ressourcenverbrauch

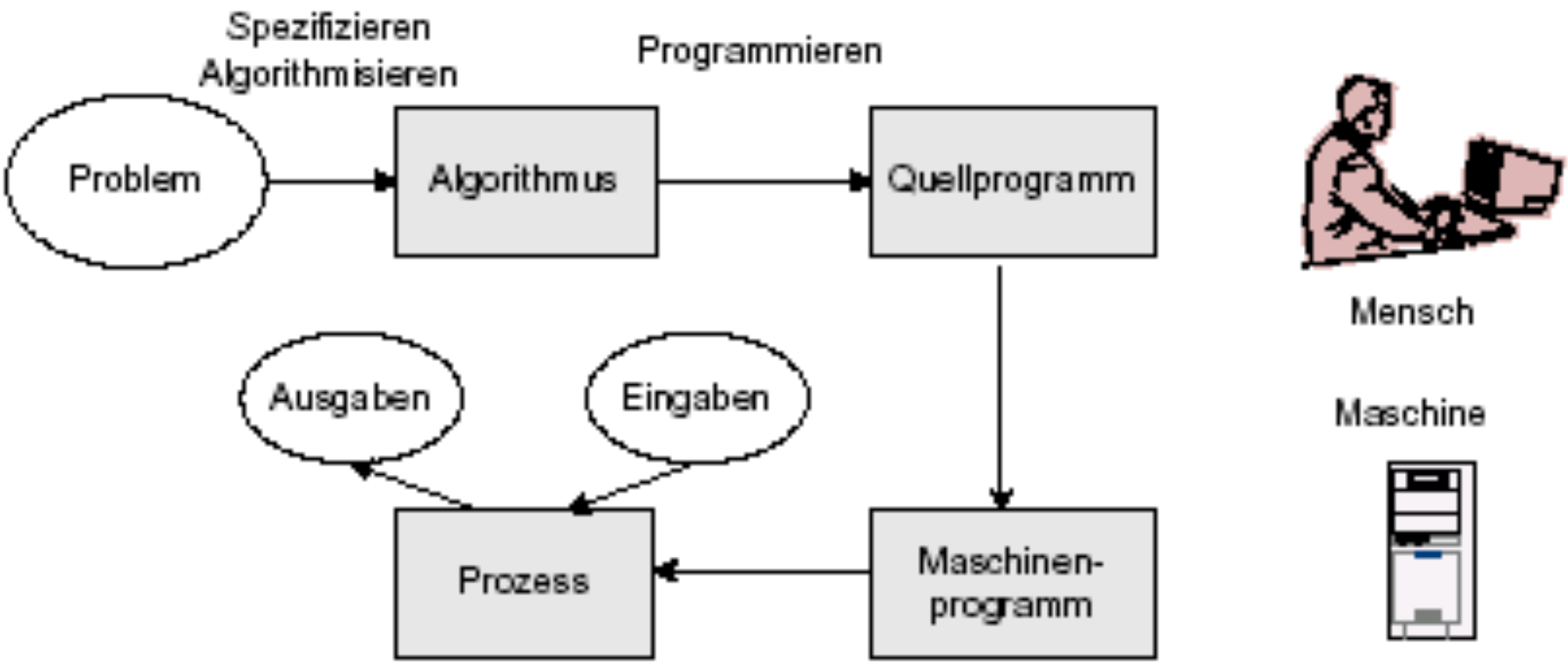
Variable und Wertzuweisungen

Hochschule Ludwigshafen am Rhein

- Eine **Variable** ist ein benannter Behälter für einen Wert
- Änderung des Wertes über Wertzuweisungen
  - x = 5
  - y = x+1 // y enthält nun den Wert 6
  - x = 2*y // x enthält nun den Wert 12
  - Dies ist nicht zu verwechseln mit dem mathematischen „=„ dort wäre so etwas ein Widerspruch!
  - Manchmal schreibt man deswegen auch x ←5 (hat sich aber nicht durchgesetzt)
- In Java muss eine Variable vor der Verwendung erst definiert werden.

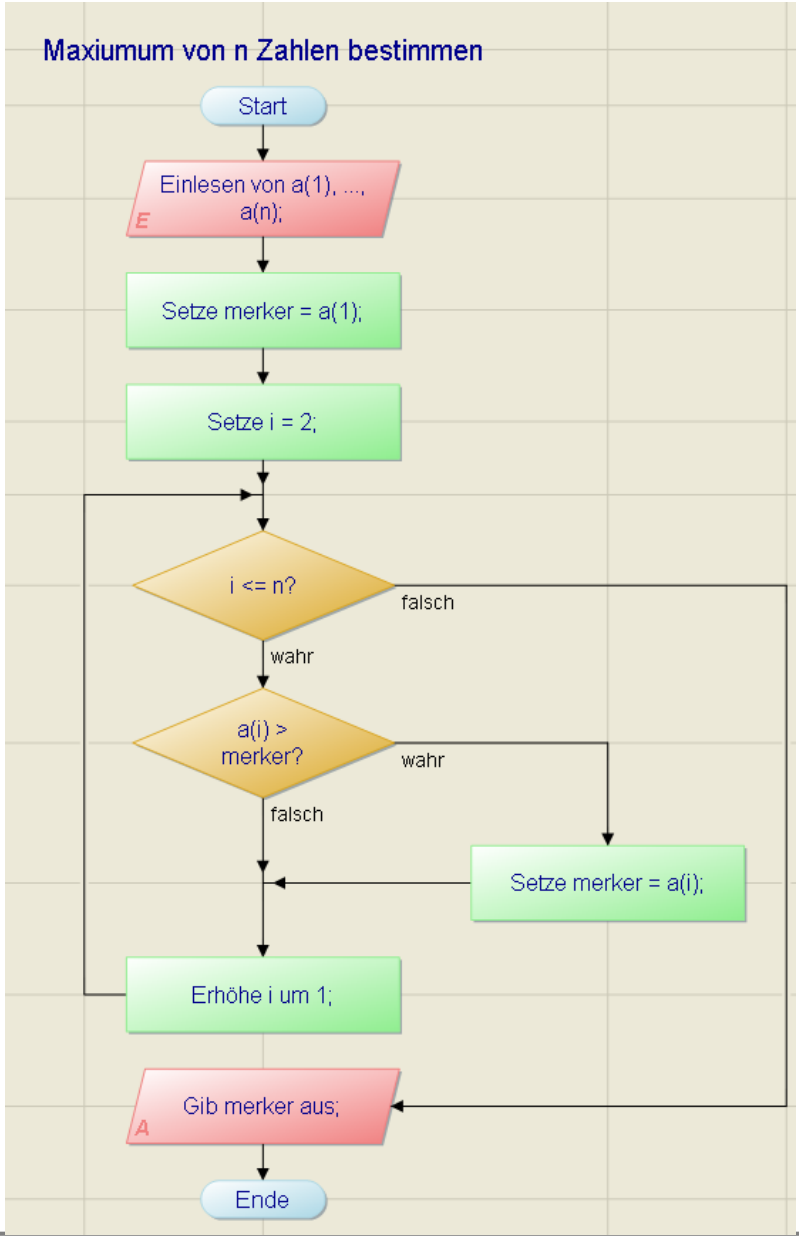
Algorithmus und Programm

Hochschule Ludwigshafen am Rhein



Quelle: Schlichter, Einführung in die Informatik

Darstellung als Programmablaufplan



Problem: Bestimme die größte Zahl der Zahlen $a_1, a_2, \ldots, a_n$

Darstellung als Pseudocode

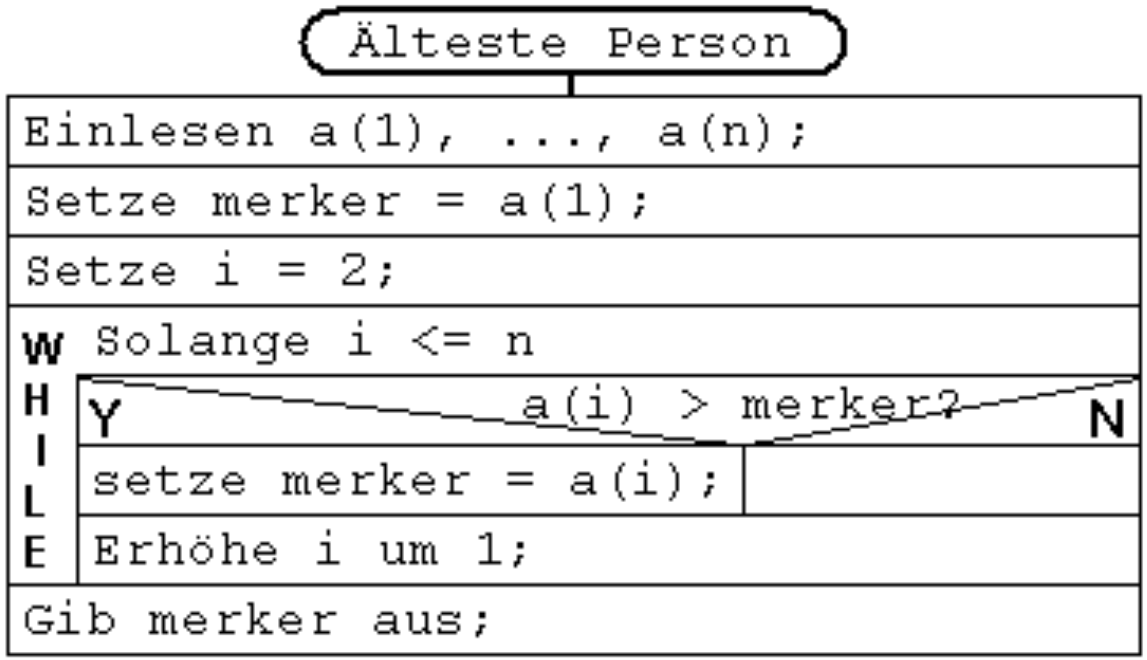
Hochschule Ludwigshafen am Rhein

einlesen $a_1,\ldots, a_n$;
setze merker = a_1;
setze i = 2;
solange (i <= n)
führe aus
    falls a_i > merker
        dann setze merker = a_i ;
    erhöhe i um 1;

gebe merker aus;

Darstellung als Struktogramm

Hochschule Ludwigshafen am Rhein



Darstellung: Fazit

Hochschule Ludwigshafen am Rhein

- In Programmablaufplänen gibt es für Kontrollstrukturen Schleifen und Wiederholungen keine eigenen Konstrukte
- Ist eine graphische Darstellung gewünscht, sind Struktogramme vorzuziehen
- Als Pseudocode lassen sich Algorithmen sehr kompakt beschreiben → daher wird diese Darstellung oft favorisiert

Definition Datenstruktur

Hochschule Ludwigshafen am Rhein

- Unter einer **Datenstruktur** versteht man die rechnerverarbeitbare Darstellung von **Information** (zweckorientiertes oder zielgerichtetes Wissen, vgl. Vorlesung „Einführung in die Wirtschaftsinformatik“)
- Bisher
  - Primitive Datentypen, Arrays, Objekte (+ Prog 2)
- Hier nun Listen und Bäume
    - ohne Verwendung des Java Collections Framework

Kursverlauf der Aktie

Hochschule Ludwigshafen am Rhein

- 1000-Euro-Aktie der Firma „MikroSoftie“
- Tabelle zeigt Veränderung des Kurses an einem Tag

Tag	1	2	3	4	5	6	7	8	9	10
Gewinn/Verlust	+5€	-8€	+3€	+3€	-5€	+7€	-2€	-7€	+3€	+5€

- also: Aktie z.B.
  - an Tag 4 um 3 € gestiegen,
  - an Tag 7 um 2 € gefallen

max. Teilsumme als Pseudocode

Hochschule Ludwigshafen am Rhein

```
/* Durchlaufe alle Teilfolgen, berechne deren Teilsumme  und vergleiche mit der
maximalen Teilsumme; Beginne mit dem Wert 0 (leere Teilfolge) und untersuche
nur die echten Teilfolgen*/

Setze MaxTeilsumme = 0;
wiederhole für alle möglichen Anfangspunkte der Teilfolgen
        wiederhole für alle möglichen Endpunkte
                /* Berechne Teilsumme für Teilfolge von Anfangspunkt bis Endpunkt */
                Setze Teilsumme = 0;
                wiederhole für alle Elemente von Anfangspunkt bis Endpunkt
                        Erhöhe Teilsumme um den Wert des Elementes;
                /* nun hat Teilsumme den Wert der Summe der Teilfolge  */
                falls Teilsumme > MaxTeilsumme
                        dann Setze MaxTeilsumme = Teilsumme;

Gib MaxTeilsumme aus;
```

Teilsumme: Erste Lösung in Java

Hochschule Ludwigshafen am Rhein

```java
public class P2ErsteLoesung {
  public static void main(String[] args) {
    int[] folge = { 5, -8, 3, 3, -5, 7, -2, -7, 3, 5 };

    // maximale Teilsumme ist mindestens 0 (Summe der leeren Teilfolge)
    int maxTeilsumme = 0;

    for (int anfang = 0; anfang < folge.length; anfang++) {
      for (int ende = anfang; ende < folge.length; ende++) {
        int Teilsumme = 0;
        for (int element = anfang; element <= ende; element++)
          Teilsumme = Teilsumme + folge[element];
        if (Teilsumme > maxTeilsumme)
          maxTeilsumme = Teilsumme;
      }
    }
    System.out.println(maxTeilsumme);
  }
}
```

Anzahl der Schritte

Hochschule Ludwigshafen am Rhein

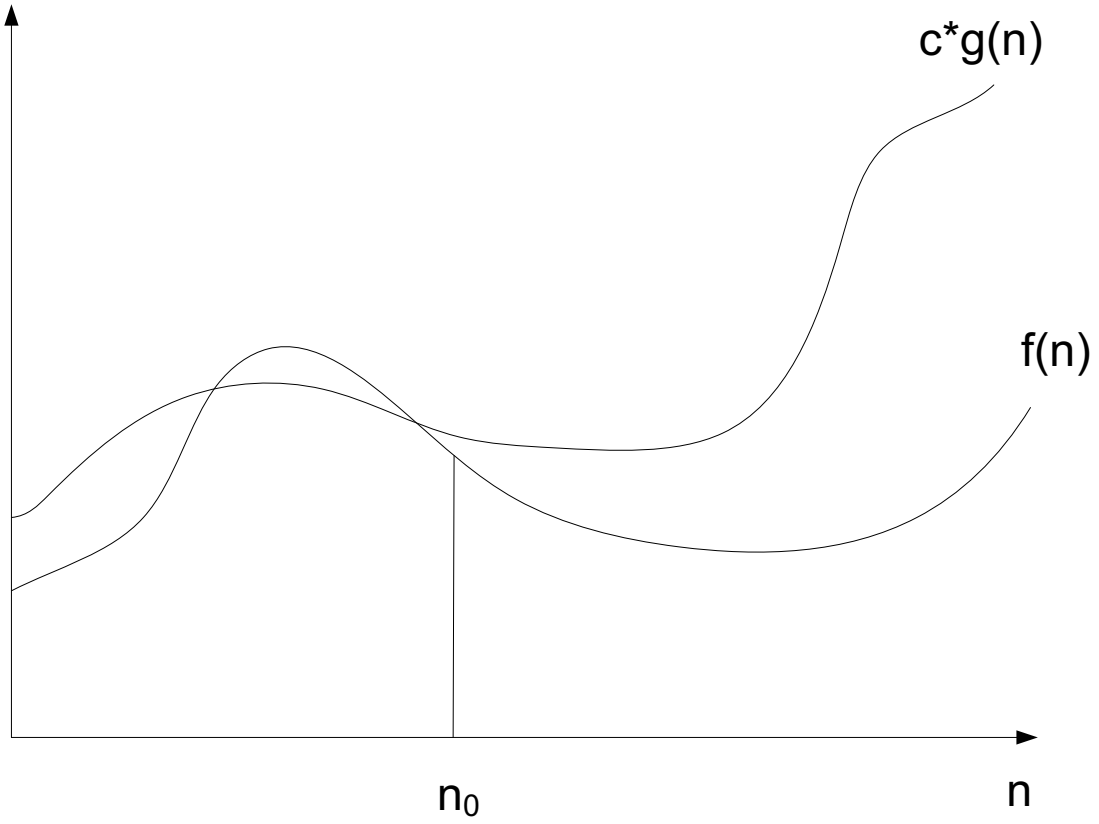
n	10	100	1.000	10.000
1/6*n^3+1/2*n^2+1/3*n	220	171.700	167.167.000	166.716.670.000
1/2(n^2+n)	55	5.050	500.500	50.005.000
Summe	275	176.750	167.667.500	166.766.675.000
1/6*n^3 (zum Vergleich)	167	166.667	166.666.667	166.666.666.667

- Für große n ist die Anzahl der Schritte ungefähr $1/6*n^3$
- Zeitkomplexität $O(n^3)$

O-Notation

Hochschule Ludwigshafen am Rhein

- Eine Funktion f ist O(g), wenn es eine Konstante c gibt, so dass für große n gilt: $f(n) \leq cg(n)$



- Wichtigste Vertreter für g: $1, n, n^2, n^3, 2^n, \log(n)$

Wachstum für Komplexitätsklassen

Hochschule Ludwigshafen am Rhein

	n=2	2^4 = 16	2^8 = 256	2^10 = 1024	2^20 = 1048756
$\log_2(n)$	1	4	8	10	20
n	2	16	256	1024	1048576
n* $\log_2(n)$	2	64	2048	10240	20971520
n^2	4	256	65536	1048576	1,09951E+12
n^3	8	4096	16777216	1073741824	1,15292E+18
2^n	4	65536	1,1579E+77	sehr groß!	sehr groß!

Teilsumme: Idee „Zeit für Raum“

Hochschule Ludwigshafen am Rhein

folge	+5	-8	+3	+3	-5	+7	-2	-7	+3	+5

Matrix s für Zeit für Raum

i/j	0	1	2	3	4	5	6	7	8	9
0	+5	-3	+0	+3	-2	+5	+3	-4	-1	+4
1		-8	-5	-2	-7	+0	-2	-9	-6	-1
2			+3	+6	+1	+8	+6	-1	+2	+7
3				+3	-2	+5	+3	-4	-1	+4
4					-5	+2	+0	-7	-4	+1
5						+7	+5	-2	+1	+6
6							-2	-9	-6	-1
7								-7	-4	+1
8									+3	+8
9										+5

s_{ij} speichertSumme von *i* bis *j*

$$s_{ij} = s_{i,j-1} + folge[j]$$

Teilsumme: Programm Zeit für Raum

Hochschule Ludwigshafen am Rhein

```java
public class P4ZeitfuerRaum {
  public static void main(String[] args) {
    int[] folge = {5,-8,3,3,-5,7,-2,-7,3,5};

//  s definieren und leere Matrix erzeugen
    int[][] s = new int[folge.length][folge.length];

//  s füllen
    for (int anfang = 0; anfang < folge.length; anfang++) {
      s[anfang][anfang] = folge[anfang];
      for (int ende = anfang +1; ende < folge.length; ende++) {
        s[anfang][ende] = s[anfang][ende-1]+folge[ende];
      }
    }

 // maximale Teilsumme bestimmen
    int maxTeilsumme = 0;
    for (int anfang = 0; anfang < folge.length; anfang++) {
      for (int ende = anfang; ende < folge.length; ende++) {
        if (s[anfang][ende] > maxTeilsumme)
          maxTeilsumme = s[anfang][ende];
      }
    }
    System.out.println(maxTeilsumme);

  }
}
```

Lineare Suche…

Hochschule Ludwigshafen am Rhein

- … als Pseudocde

```
Setze Resultat = -1;
wiederhole für alle Elemente von folge und solange  Resultat == -1;
        falls Wert des Elementes = Sucheigenschaft
                dann Setze Resultat = Position des Elementes;
Gib Resultat aus;
```

- … und als Programm

```
int[] folge = { 3, 1, 5, 9, 7 };
int sucheigenschaft = 9;

int resultat = -1;
for (int i = 0; resultat == -1 && i < folge.length; i++) {
  if (folge[i] == sucheigenschaft) {
    resultat = i;
  }
}
System.out.println(resultat);
```

Binäre Suche (1)

Hochschule Ludwigshafen am Rhein

Sucheigenschaft: 9

Stelle	0	1	2	3	4	5	6	7	8	9
1. Schritt	unter				mitte					ober
Wert	0	1	2	4	5	8	9	12	13	18
2. Schritt						unter		mitte		ober
Wert	0	1	2	4	5	8	9	12	13	18
3. Schritt						unter	ober			
Wert	0	1	2	4	5	8	9	12	13	18
						mitte				
4. Schritt							unter ober			
Wert	0	1	2	4	5	8	9	12	13	18
							mitte			

Binäre Suche (2)

Hochschule Ludwigshafen am Rhein

Sucheigenschaft: 10

Stelle	0	1	2	3	4	5	6	7	8	9
1. Schritt	unter				mitte					ober
Wert	0	1	2	4	5	8	9	12	13	18
2. Schritt					unter		mitte			ober
Wert	0	1	2	4	5	8	9	12	13	18
3. Schritt					unter	ober				
Wert	0	1	2	4	5	8	9	12	13	18
					mitte					
4. Schritt						unter ober				
Wert	0	1	2	4	5	8	9	12	13	18
						mitte				
5. Schritt						ober	unter			
Wert	0	1	2	4	5	8	9	12	13	18

Abbruch der Suche: Sucheigenschaft nicht gefunden

Binäre Suche…

Hochschule Ludwigshafen am Rhein

→ **als Programm in den Übungen**

Binäre Suche: Anzahl Schritte

Hochschule Ludwigshafen am Rhein

Verfahren	10	100	1.000	10.000	100.000
sequentiell (n/2)	5	50	500	5.000	50.000
binär ($\log_2(n)$)	3,3	6,6	10,0	13,3	16,6

Selection Sort

Hochschule Ludwigshafen am Rhein

1. Durchlauf	5	1	8	3	9	2
2. Durchlauf	1	5	8	3	9	2
3. Durchlauf	1	2	8	3	9	5
4. Durchlauf	1	2	3	8	9	5
5. Durchlauf	1	2	3	5	9	8
Ergebnis	1	2	3	5	8	9

- Legende:
  - in dem Eingerahmten wird das kleinste Element gesucht
  - die beiden gelben Elemente werden vertauscht

Bubble Sort

Hochschule Ludwigshafen am Rhein

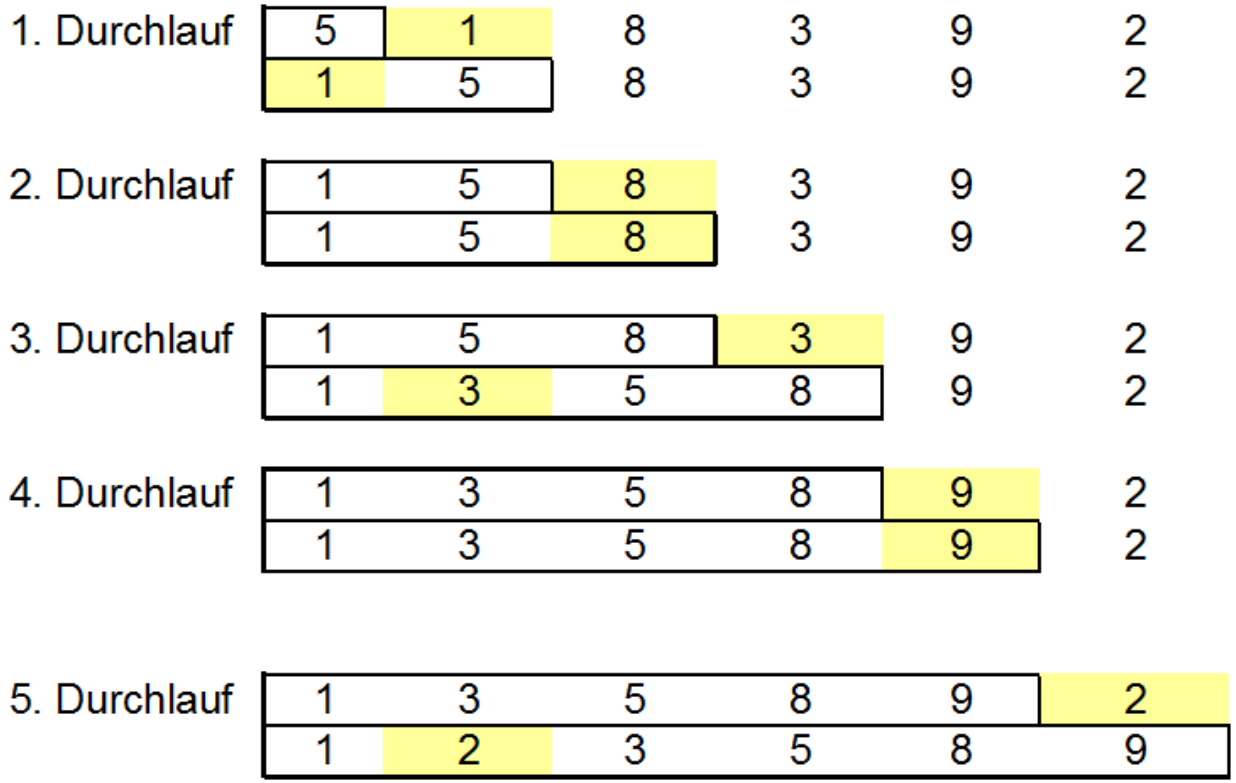
1. Durchlauf	5	3	8	9	2	1
	3	5	8	9	2	1
	3	5	8	2	9	1
	3	5	8	2	1	9
2. Durchlauf	3	5	8	2	1	9
	3	5	2	8	1	9
	3	5	2	1	8	9
3. Durchlauf	3	5	2	1	8	9
	3	2	5	1	8	9
	3	2	1	5	8	9
4. Durchlauf	3	2	1	5	8	9
	2	3	1	5	8	9
	2	1	3	5	8	9
5. Durchlauf	2	1	3	5	8	9
	1	2	3	5	8	9
Ergebnis	1	2	3	5	8	9

Erklärung:

In dem Eingerahmten wird von links nach rechts paarweise verglichen und ggf. vertauscht

Insertion Sort

Hochschule Ludwigshafen am Rhein



- Legende:
  - Das gelbe Element wird in das Eingerahmte einsortiert
  - Dabei werden die größeren Elemente nach rechts verschoben

Rekursive und iterative Algorithmen

Hochschule Ludwigshafen am Rhein

- Iteration = Wiederholung
- Bisher haben wir Algorithmen durch wiederholte Anweisungen z.B. in einer Schleife dargestellt → iterative Algorithmen
- Definition Rekursion (Quelle: Sedgewick)
  - Ein rekursiver Algorithmus löst ein Problem, in dem er eine oder mehrere Instanzen des gleichen Problems löst
  - Um rekursive Algorithmen in Java zu implementieren, verwenden wir rekursive Methoden, d.h. Methoden, die sich selbst aufrufen
- Viele Aufgaben haben eine iterative und eine rekursive Lösung

Fakultät

Hochschule Ludwigshafen am Rhein

- Definition
  - n! = 1 * 2 * … * (n-1) * n
- Werte

n	1	2	3	4	5	6	7	8	9
Fak(n)	1	2	6	24	120	720	5040	40320	362880

- Programm

```
long fak = 1;
for (long i = 2; i <= n; i++)
    fak = fak * i;
```

Fakultät rekursiv

Hochschule Ludwigshafen am Rhein

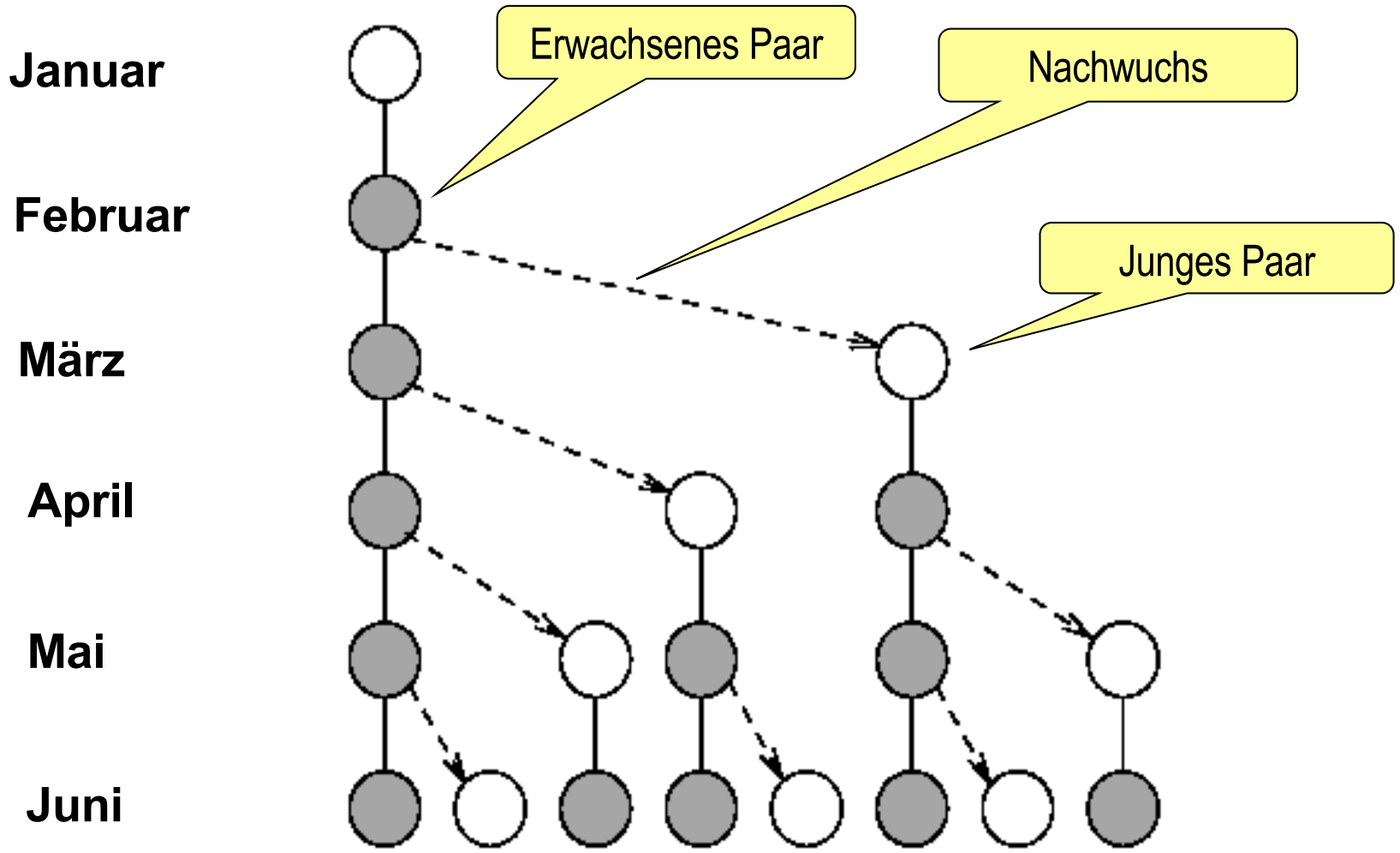
- Pseudocode

```
Modul Fakultät (n)
falls n = 1
        dann Fakultät = 1;
        sonst Fakultät = n * Fakultät(n-1)
```

- Programm

```
public class Fakultät {
  static long fak(long n) {
    if (n == 1)
      return 1;
    else
      return n* fak(n-1);
    }

  public static void main(String[] args) {
       System.out.println(fak(6));
  }
}
```

Fibonacci



n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Fib(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377

Fibonacci rekursiv

Hochschule Ludwigshafen am Rhein

- Pseudocode

  Modul Fibonacci (n)
  falls n = 0 oder n=1
  dann Fibonacci = n;
  sonst Fibonacci = Fibonacci(n-1) + Fibonacci (n-2);

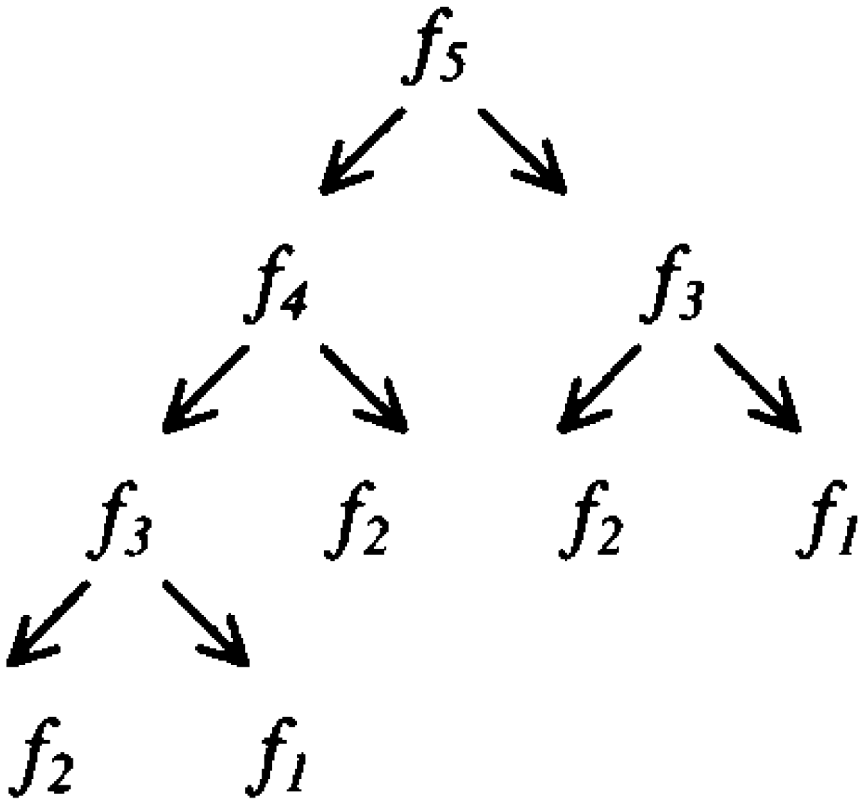
- Programm

```
public class Fibonacci {
   static long fib(long n) {
    if (n == 0 || n == 1)
      return n;
    else
      return fib(n-2)+ fib(n-1);
    }

  public static void main(String[] args) {
     System.out.println(fib(9));
  }
}
```

Fibonacci rekursiv - Rechenweg

Hochschule Ludwigshafen am Rhein

- Zwischenergebnisse werden mehrfach berechnet



Fibonacci rekursiv - Zeitkomplexität

Hochschule Ludwigshafen am Rhein

- Formel von Moivre-Binet

$$\text{fib}(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

  - Beweis: vgl. http://de.wikipedia.org/wiki/Fibonacci-Folge#Formel_von_Moivre-Binet

- Aufwand(n) = Aufwand(n - 1) + Aufwand(n - 2).
  - genau wie die Werte selbst: **exponentiell**

$$O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right) \quad \text{wobei} \quad \left(\frac{1+\sqrt{5}}{2}\right) = 1{,}618...$$

n	((1+WURZEL(5))/2)^n
30	1.860.498
31	3.010.349
32	4.870.847
33	7.881.196
34	12.752.043
35	20.633.239
36	33.385.282
37	54.018.521
38	87.403.803
39	141.422.324
40	228.826.127
41	370.248.451
42	599.074.578
43	969.323.029
44	1.568.397.607
45	2.537.720.636

Fibonacci iterativ

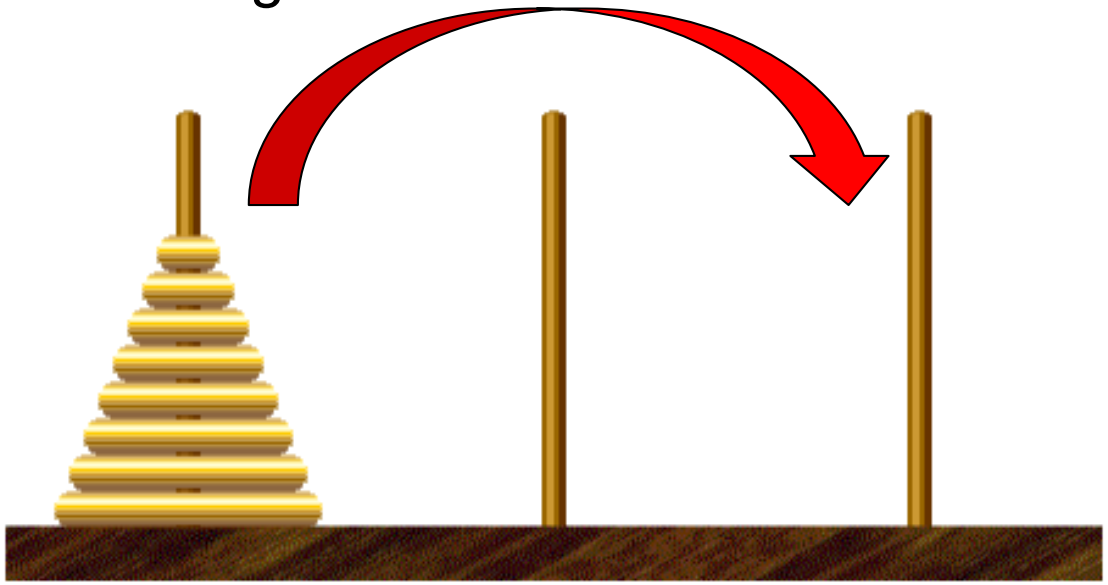
Hochschule Ludwigshafen am Rhein

→ in den Übungen

Türme von Hanoi: Problemstellung

Hochschule Ludwigshafen am Rhein

- 64 goldene Scheiben verschiedener Größe sind gestapelt.
- Es darf immer nur eine kleinere Scheibe auf einer größeren liegen.
- Scheiben dürfen einzeln von einem “Turm” zu einem anderen Turm gebracht werden. Am Ende sollen alle Scheiben korrekt auf dem rechten Turm liegen.



https://www.mathematik.ch/spiele/hanoi_mit_grafik/

Türme von Hanoi

Hochschule Ludwigshafen am Rhein

- Die einzige Möglichkeit, die unterste (größte) Scheibe von links nach rechts zu bewegen:

Türme von Hanoi: Programmieransatz

Hochschule Ludwigshafen am Rhein

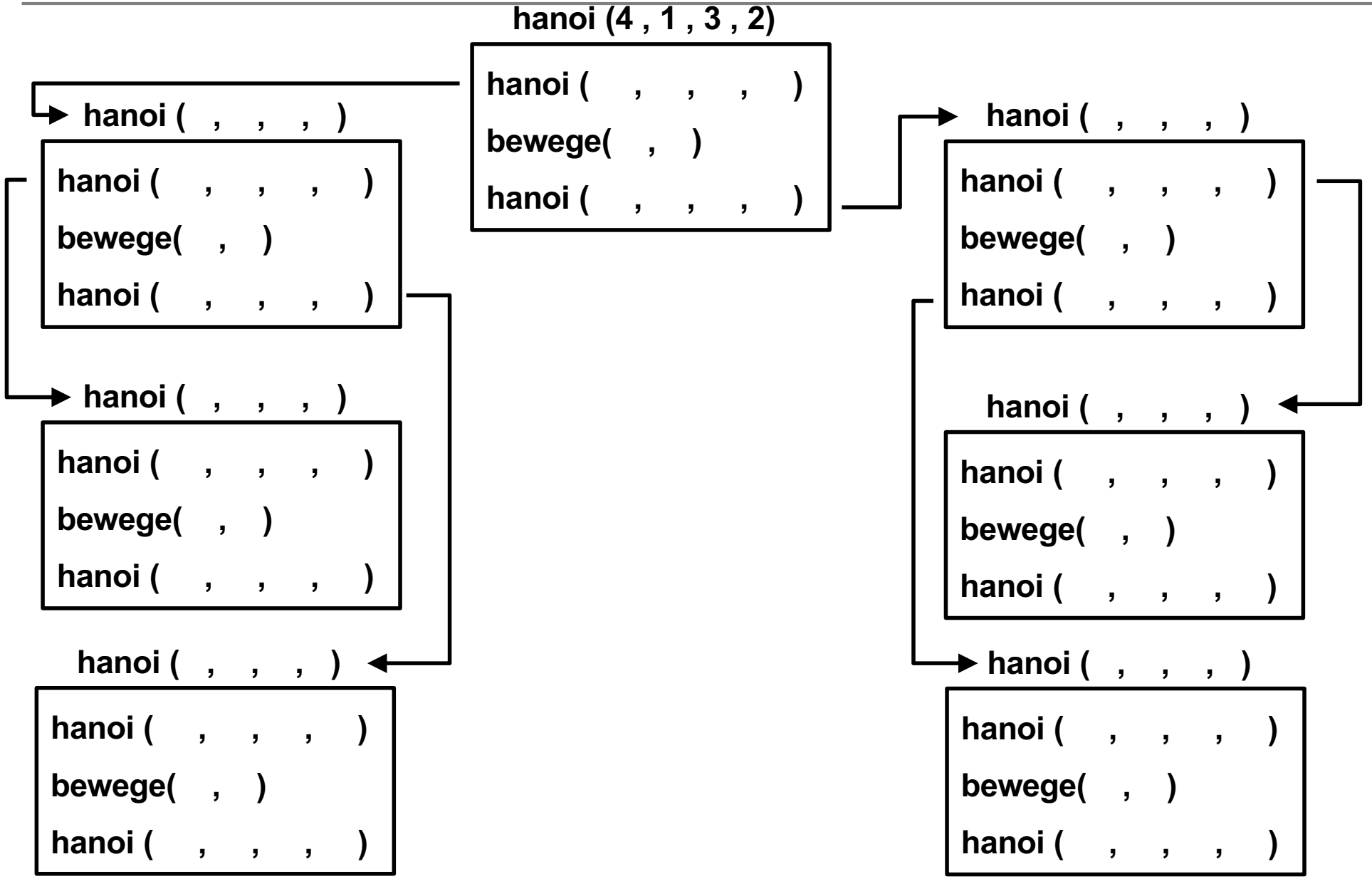
```
public class HanoiAnsatz {

      // Bewege eine Scheibe
  public static void bewege1 (int von, int nach) {
      System.out.println("Eine Scheibe von Stapel " + von + " nach Stapel "
+ nach);
  }

  // Bewege zwei Scheiben. Benutze die Lösung für 1 Scheibe
  public static void bewege2 (int von, int nach, int arbber) {
    bewege1 (von, arbber);        //  (I)
    bewege1 (von, nach);        //  (II)
    bewege1 (arbber, nach);         //  (III)
  }

  // Bewege drei Scheiben. Benutze die Lösung für 2 Scheiben
  public static void bewege3 (int von, int nach, int arbber) {
    bewege2 (von, arbber, nach); //  (I)
    bewege1 (von, nach);        //  (II)
    bewege2 (arbber, nach, von); //  (III)
  }

  public static void main (String[] args) {
    // von = Stapel 0, nach = Stapel 2, arbber = Stapel 1
    bewege3 (0, 2, 1);
  }
}
```

Türme von Hanoi: Programm

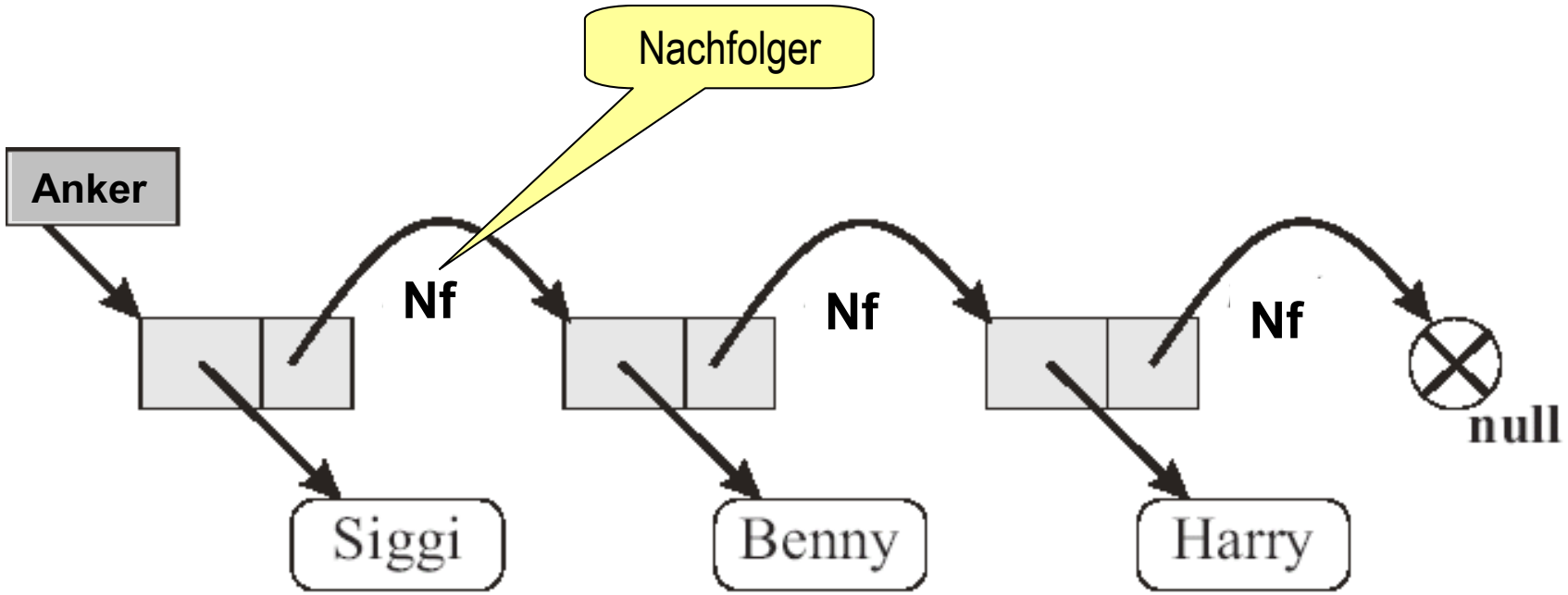
Hochschule Ludwigshafen am Rhein

→ in den Übungen

Beispiel einer Aufrufkette



Prinzip der verketteten Liste



Knoten und verketten

Hochschule Ludwigshafen am Rhein

- Knoten

```
public class Knoten {
  public int Zahl;
  public Knoten Nf;     //Nachfolger

  public Knoten(int Zahl, Knoten Nf) {
    this.Zahl = Zahl;
    this.Nf = Nf;
  }
}
```

Nutzinhalt (beliebiges Objekt möglich)

rekursiv

- verketten:

```
    Knoten k1, k2;

    k1 = new Knoten(1, null);
    k2 = new Knoten(2, null);
    k1.Nf = k2;

// noch einen Konten davorhängen
    Knoten k0 = new Knoten(0, k1);
```

Verkettete Liste: Programmcode

Hochschule Ludwigshafen am Rhein

- Knoten

```
public class Knoten {
  public int Zahl;
  public Knoten Nf;     //Nachfolger

  public Knoten(int Zahl, Knoten Nf) {
    this.Zahl = Zahl;
    this.Nf = Nf;
  }
}
```

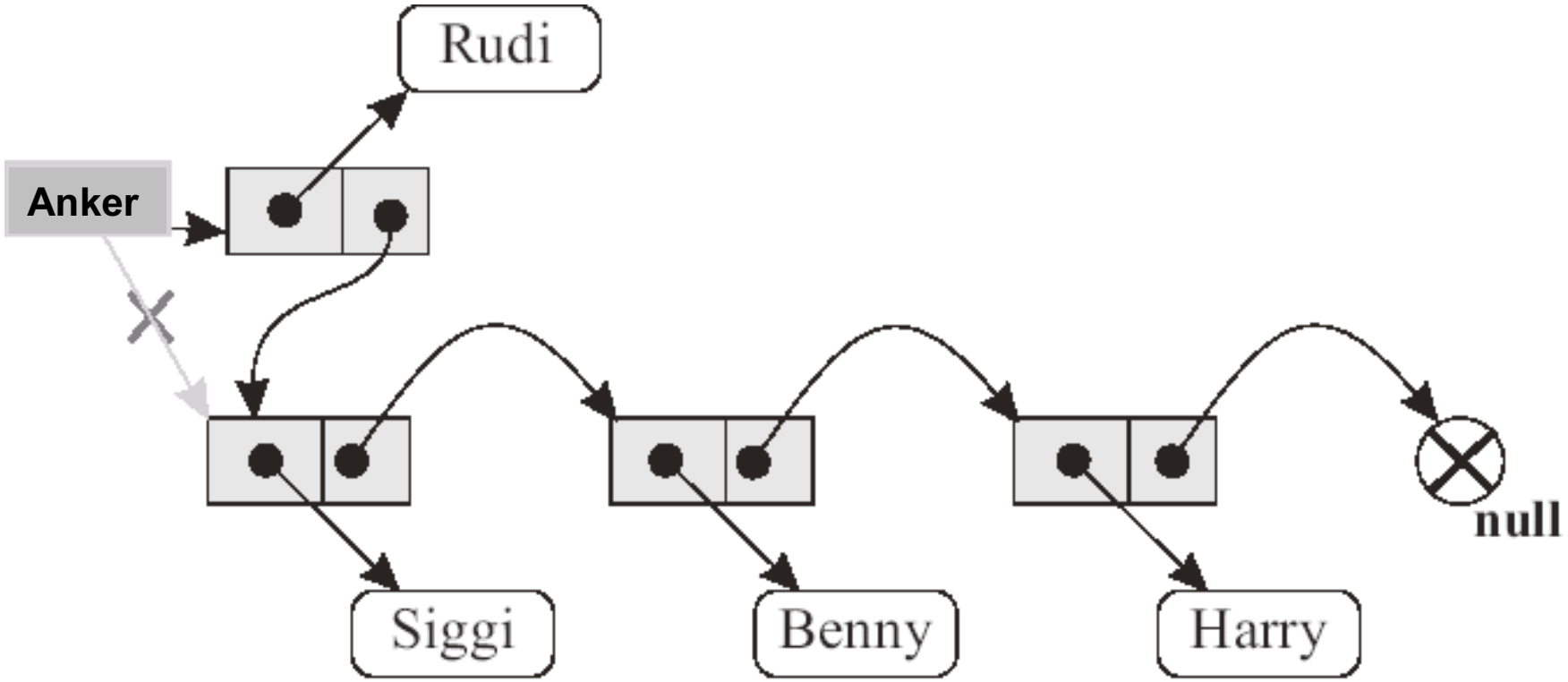
Nutzinhalt (beliebiges Objekt möglich)

rekursiv

- Liste

```
public class Liste {
Knoten anker;

        public Liste() {
              this.anker = null;
        }
 }
```

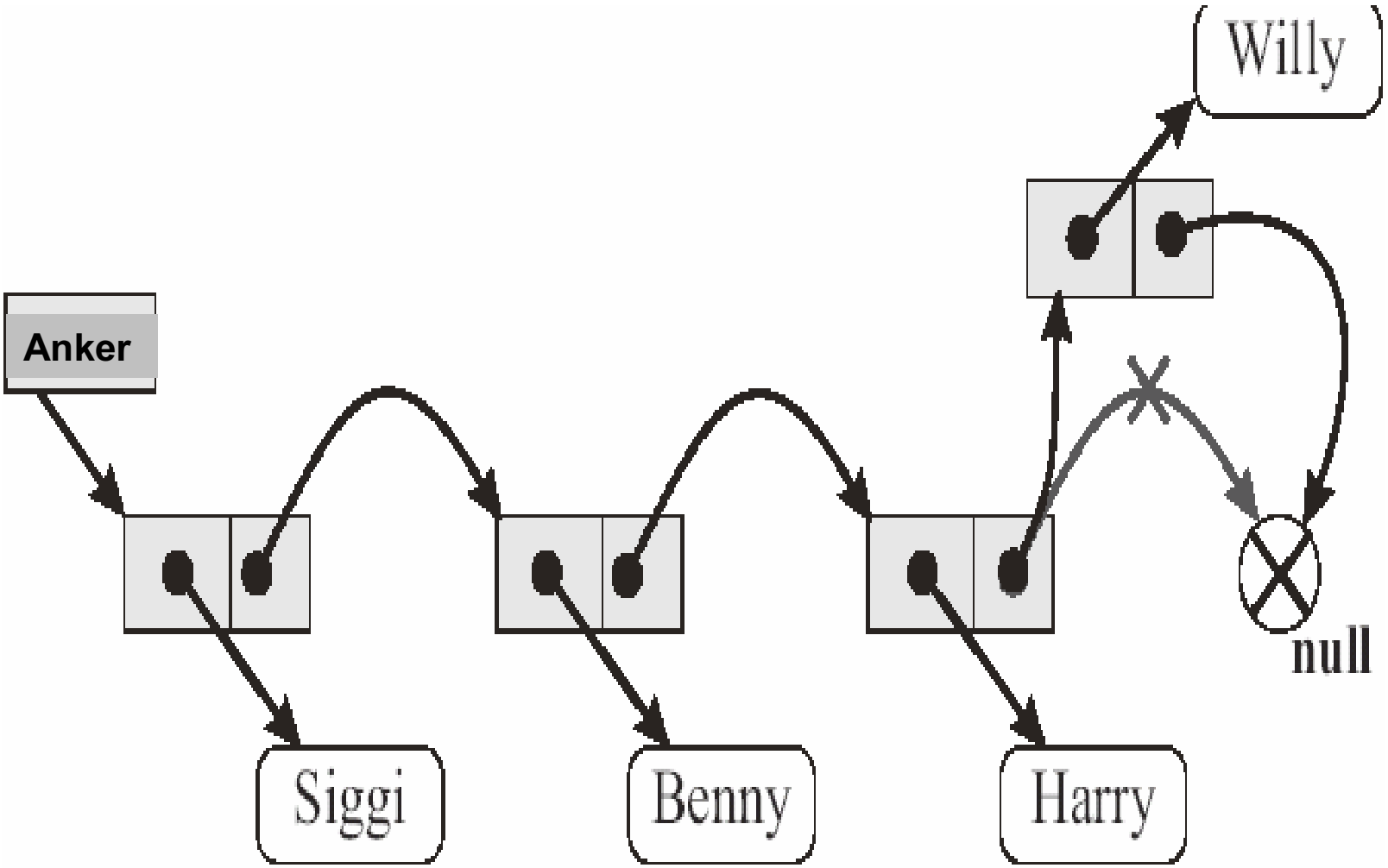
Einfügen zu Beginn




```
void einfuegenAnfang (int Zahl) {
    anker = new Knoten (Zahl, anker);
}
```

Einfügen am Ende

Hochschule Ludwigshafen am Rhein



Einfügen am Ende: Programmcode

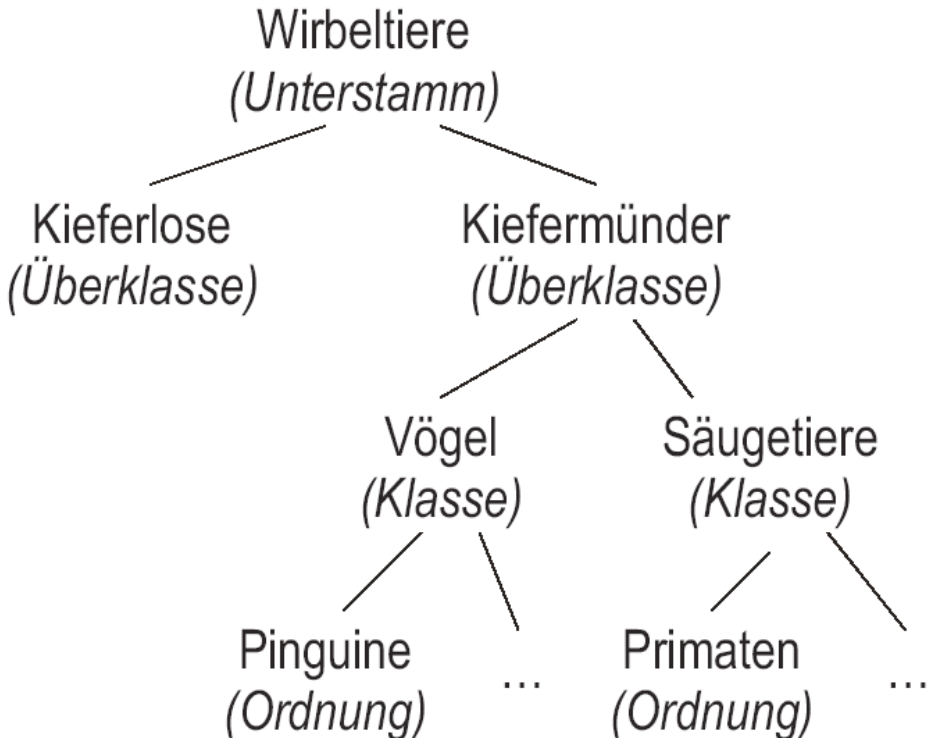
Hochschule Ludwigshafen am Rhein

```
public void einfuegenEnde(int Zahl) {
      if (this.anker != null) {
            // Suche Knoten, dessen Nachfolger auf null zeigt
            Knoten k = this.anker;
            while (k.Nf != null)
                  k = k.Nf;
            // Knoten gefunden --> bekommt nun einen Knoten angehängt
            k.Nf = new Knoten(Zahl, null);
      } else
            // Sonderbehandlung zur Vermeidung von
            // java.lang.NullPointerException
            einfuegenAnfang(Zahl);
}
```

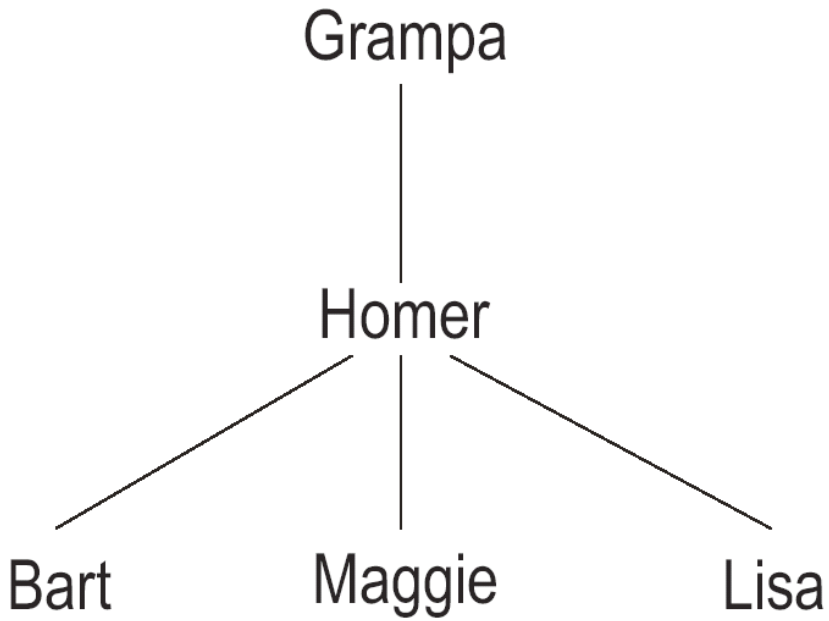
Beispiele für Bäume (1)

Hochschule Ludwigshafen am Rhein

Vererbungsbeziehungen



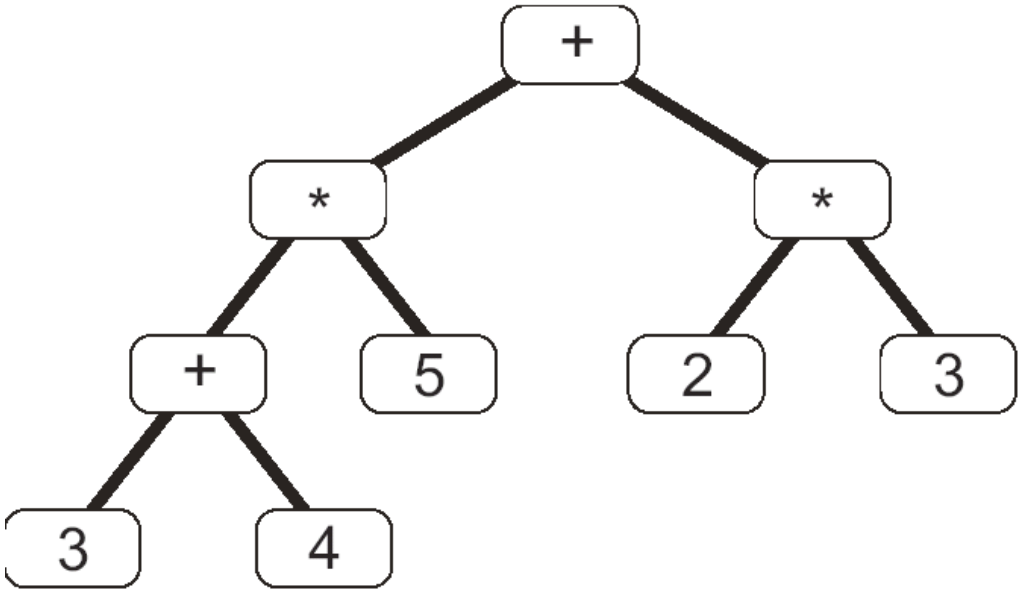
Stammbaum



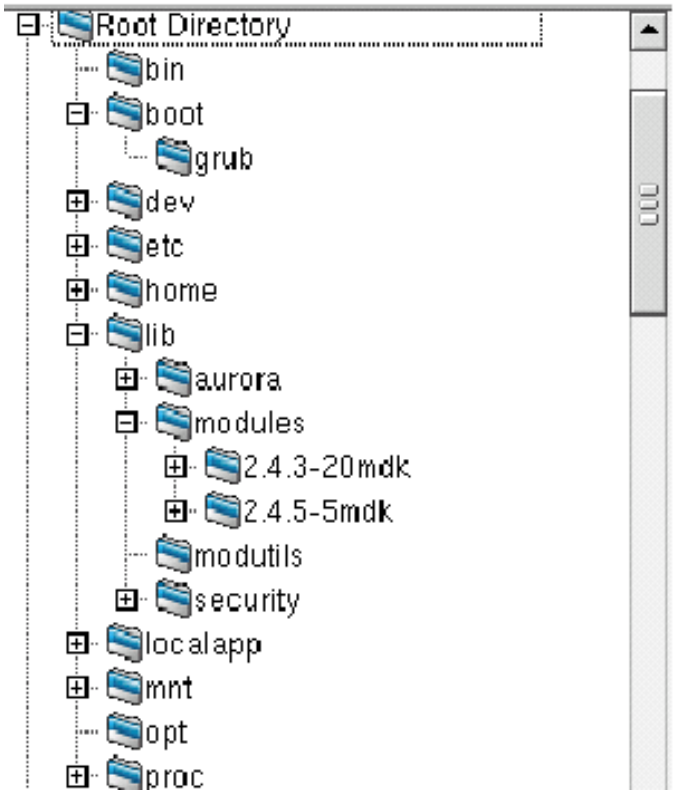
Beispiele für Bäume (2)

Hochschule Ludwigshafen am Rhein

Arithmetischer Ausdruck

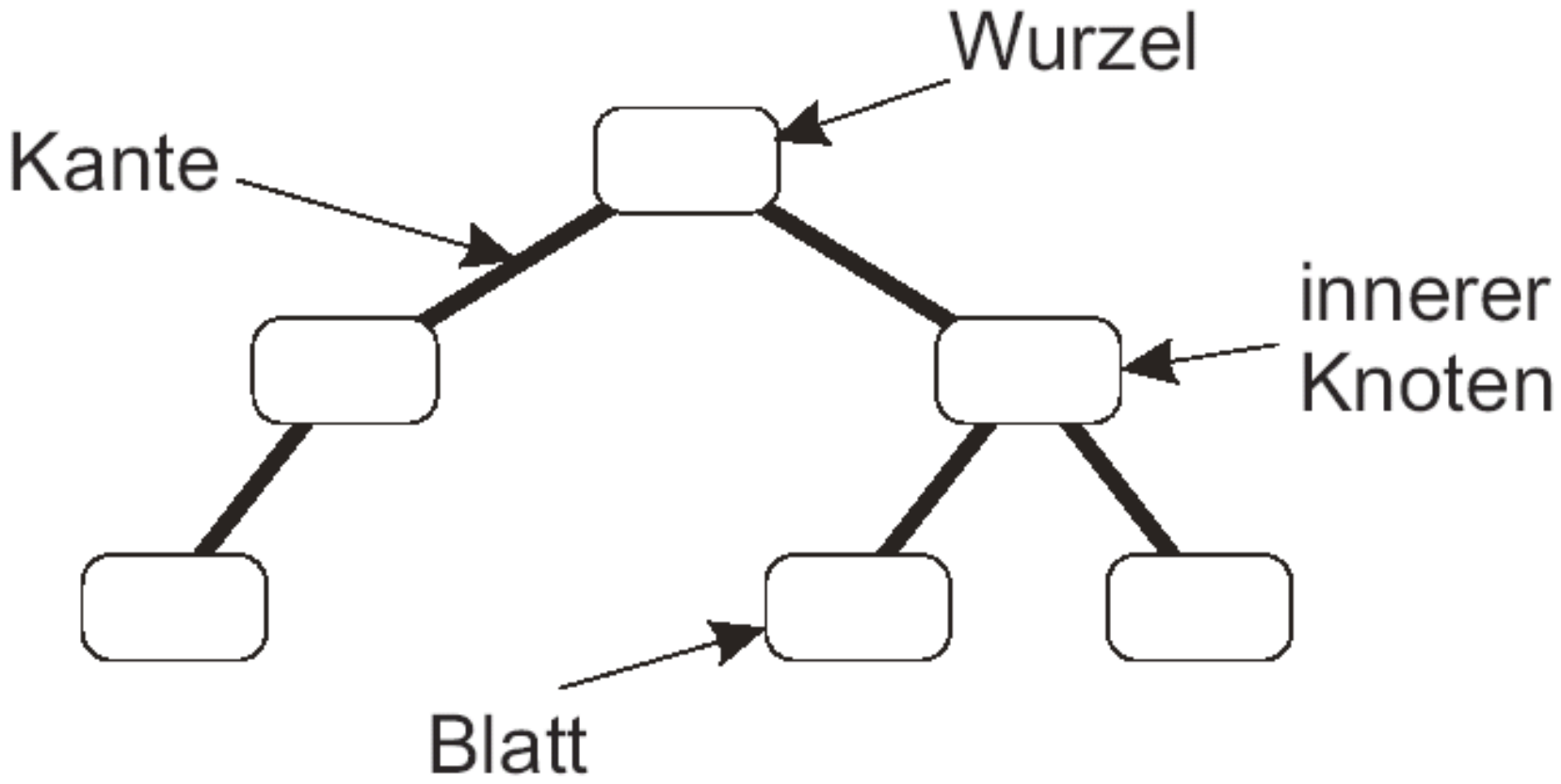


Dateisystem



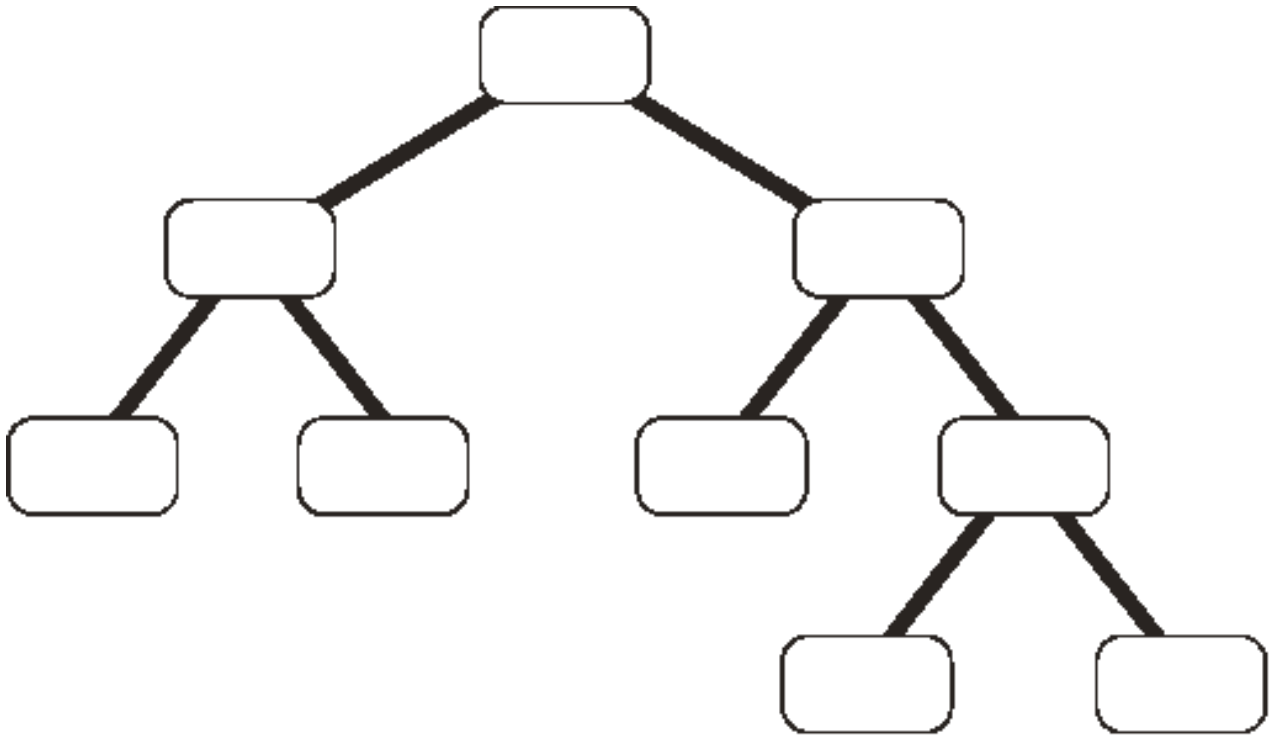
Begriffe in einem Baum

Hochschule Ludwigshafen am Rhein



Ebene und Höhe

Hochschule Ludwigshafen am Rhein



Ebene 0

Ebene 1

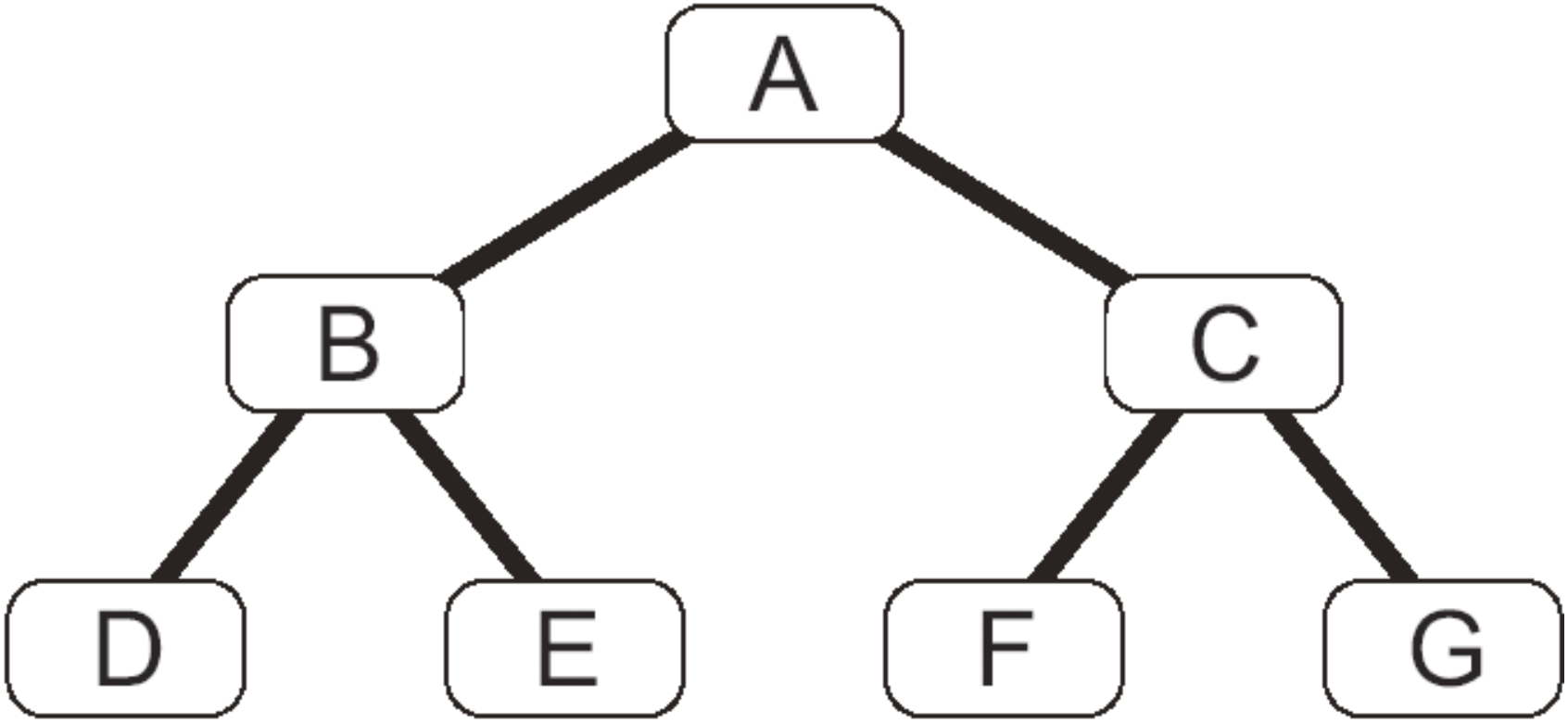
Ebene 2

Ebene 3

Höhe 4

Binärbaum

Hochschule Ludwigshafen am Rhein



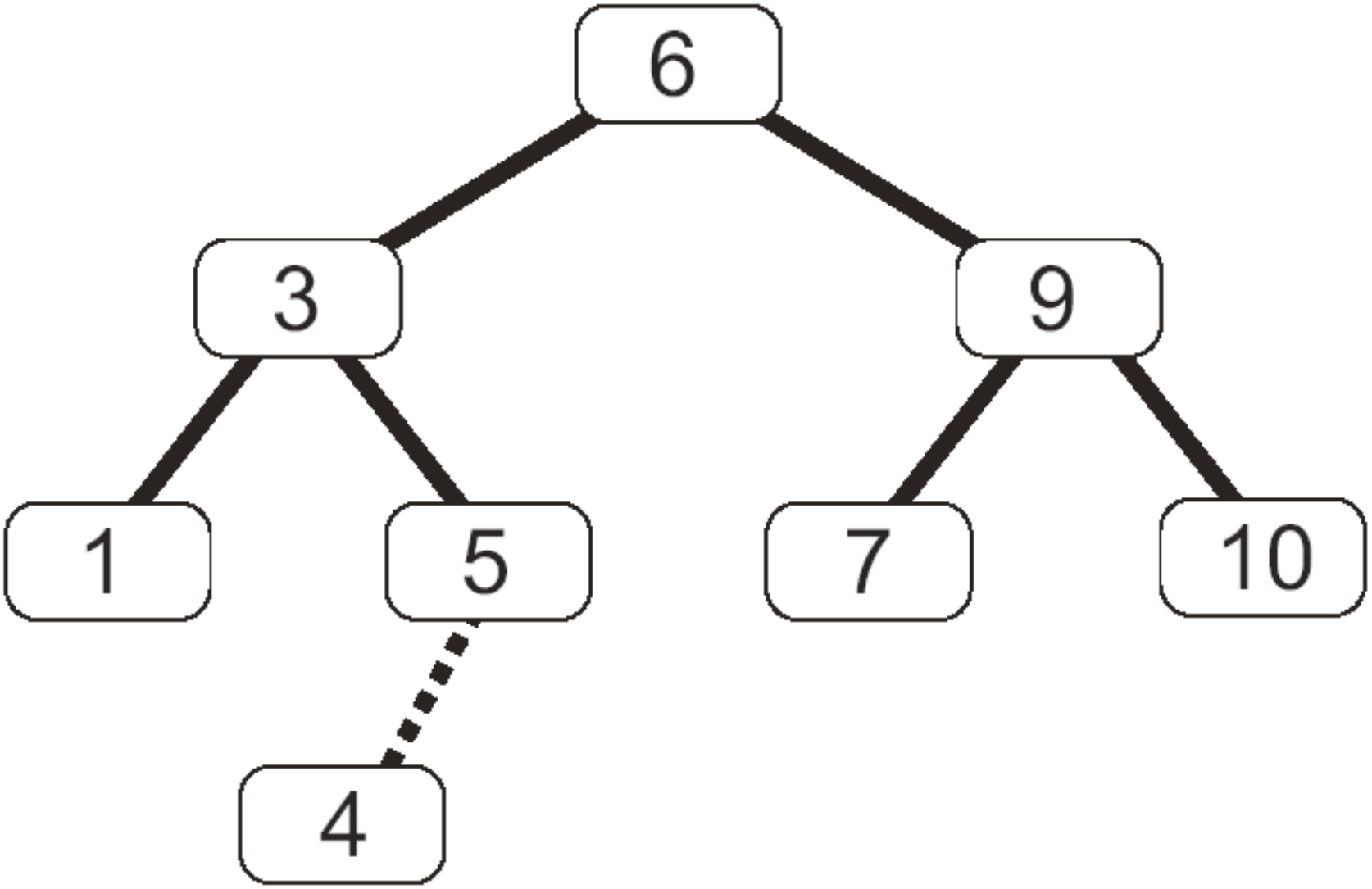
Binärbaum Code

Hochschule Ludwigshafen am Rhein

```
public class Knoten {
   char wert;        // Nutzinhalt
   Knoten links, rechts;

   public String toString() {
      return " "+this.wert;
   }

   public Knoten (char wert, Knoten links, Knoten rechts) {   // Konstruktor
     this.wert = wert;
     this.links = links;
     this.rechts = rechts;
   }
}

public class Binaerbaum {
  Knoten wurzel;

  public Binaerbaum() {
    Knoten d = new Knoten('D', null, null);
    Knoten e = new Knoten('E', null, null);
    Knoten b = new Knoten('B', d, e);
    Knoten f = new Knoten('F', null, null);
    Knoten g = new Knoten('G', null, null);
    Knoten c = new Knoten('C', f, g);
    this.wurzel = new Knoten('A', b, c);
  }
}
```

Einfügen in einen Suchbaum

Hochschule Ludwigshafen am Rhein



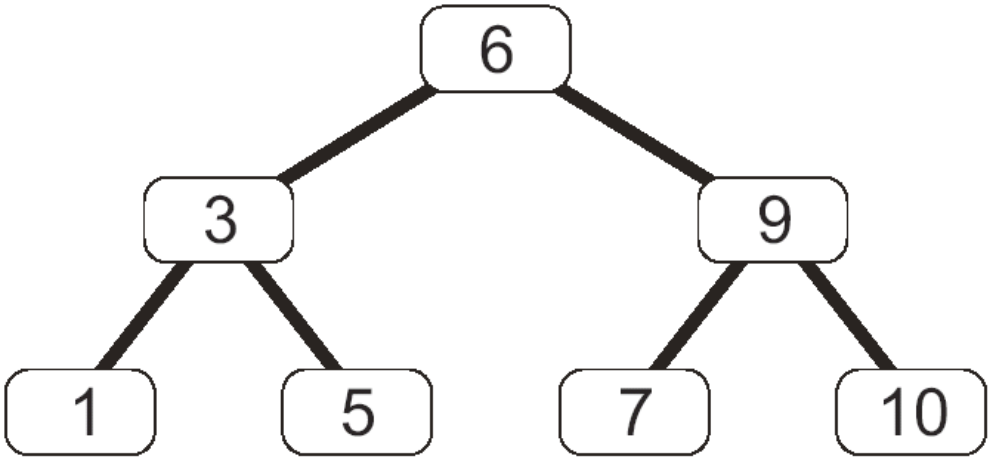
Einfügen in einen Suchbaum

Hochschule Ludwigshafen am Rhein

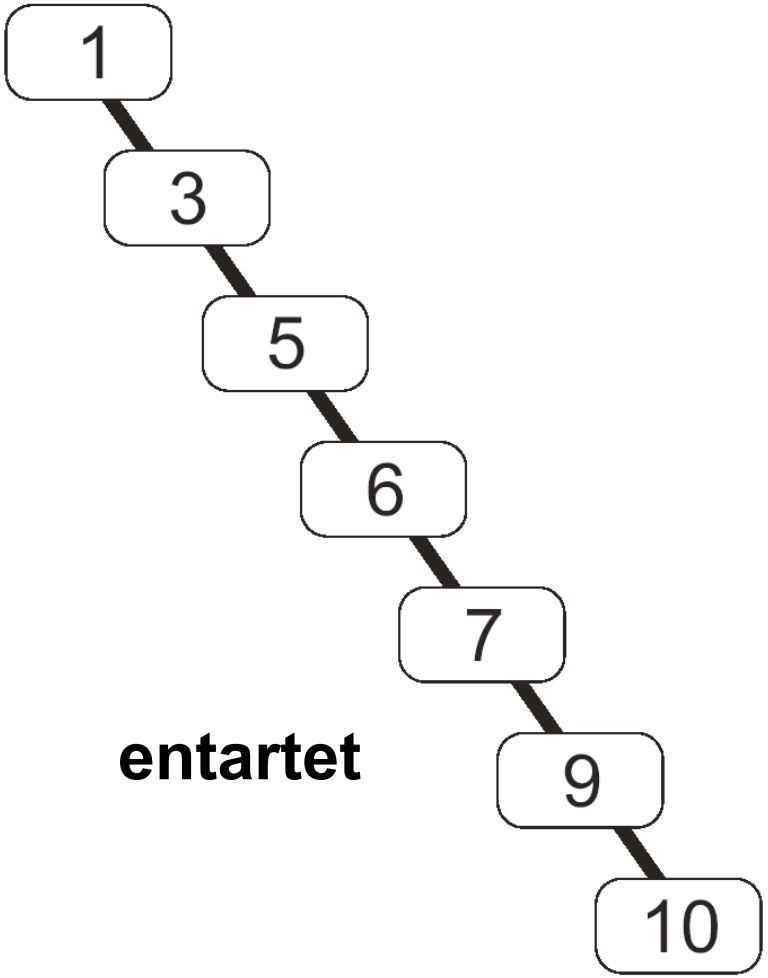
```
public void einfuegen(int wert) {
    Knoten neuerKnoten = new Knoten(wert, null, null);
    if (this.wurzel != null) {
      Knoten temp = this.wurzel;
      Knoten parent = null;
//suche einen freien Platz
      while (temp != null) {
        parent = temp;
        if ( neuerKnoten.wert > temp.wert )
          temp = temp.rechts;
        else   //wenn der Wert schon vorhanden wird links eingetragen
          temp = temp.links;
      }
//    neuen Knoten eintragen:
      if ( neuerKnoten.wert > parent.wert )
          parent.rechts = neuerKnoten;
        else
          parent.links = neuerKnoten;
   }
    else
      this.wurzel = neuerKnoten;

  }
```

„Gute“ und „schlechte“ Suchbäume

Hochschule Ludwigshafen am Rhein



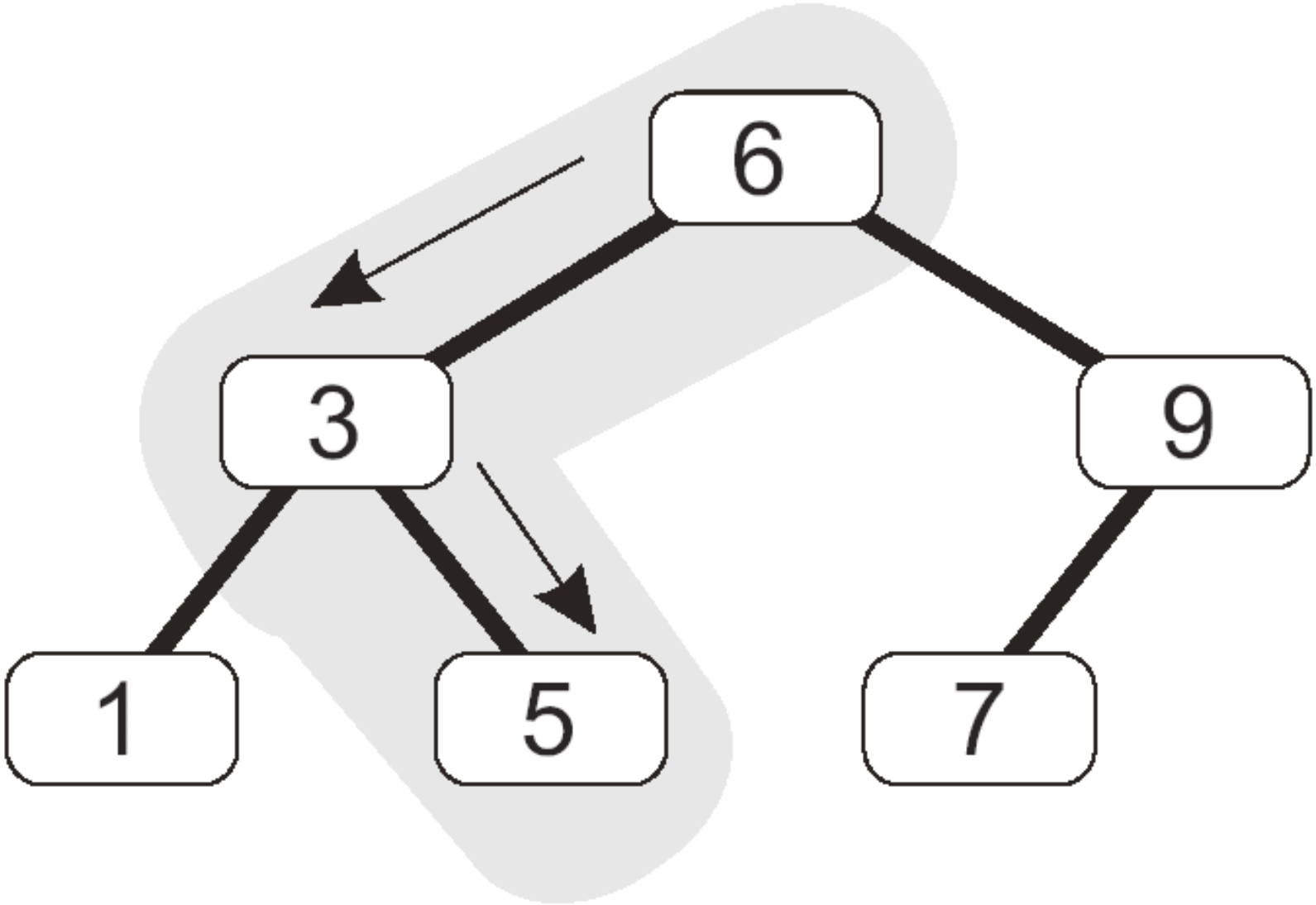
ausgeglichen



entartet

Suche im Suchbaum

Hochschule Ludwigshafen am Rhein



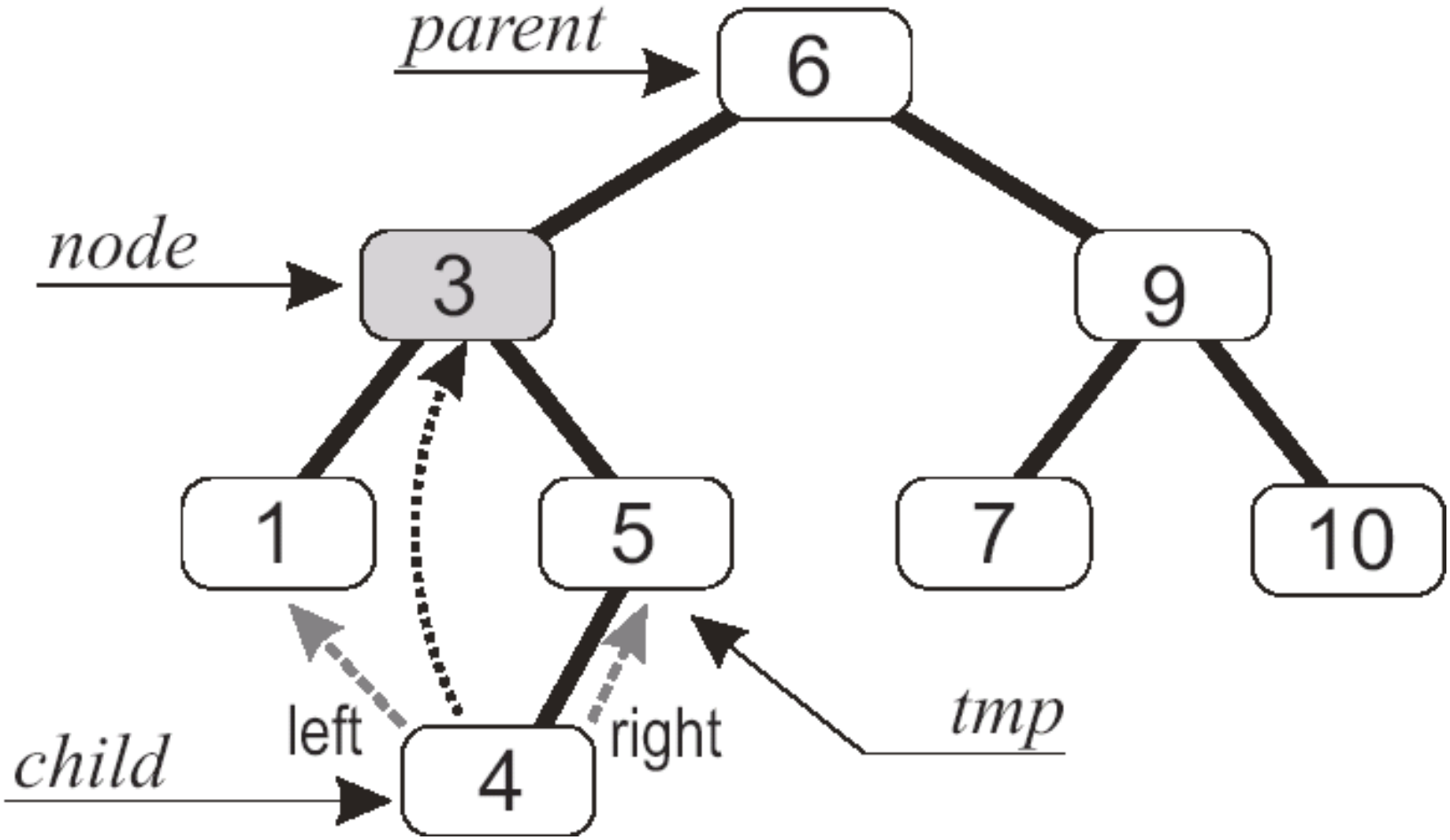
Suche im Suchbaum

Hochschule Ludwigshafen am Rhein

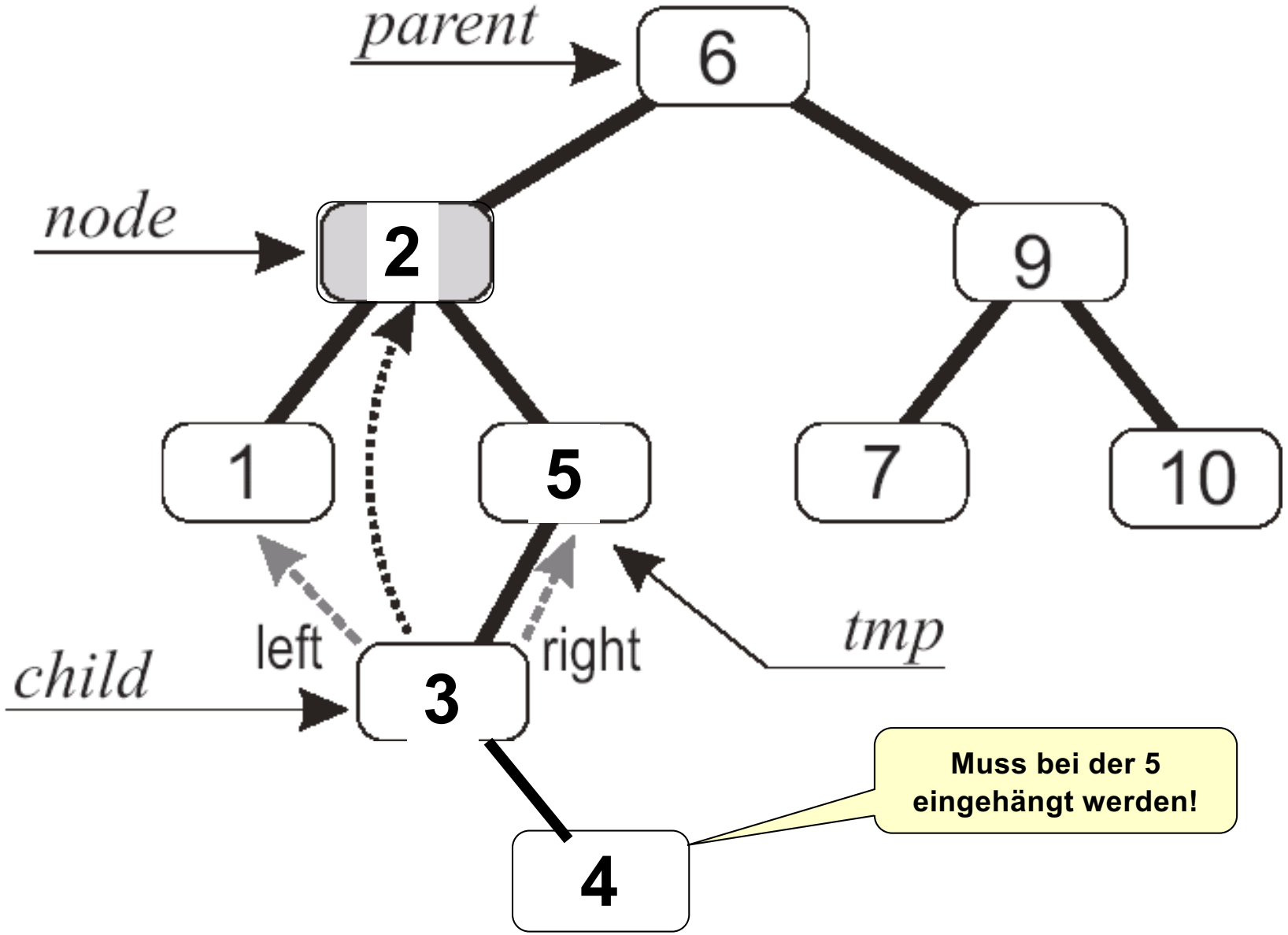
```
private Knoten sucheKnoten(int wert) {
  Knoten temp = this.wurzel;
  while ( temp !=  null) {
    if (wert == temp.wert )
      return temp;
    else if (wert > temp.wert)
      temp = temp.rechts;
    else
      temp = temp.links;
  }
  return null;
}
public boolean suche(int wert){
  return (sucheKnoten(wert) != null);
}
```

Löschen im Suchbaum (1)

Hochschule Ludwigshafen am Rhein

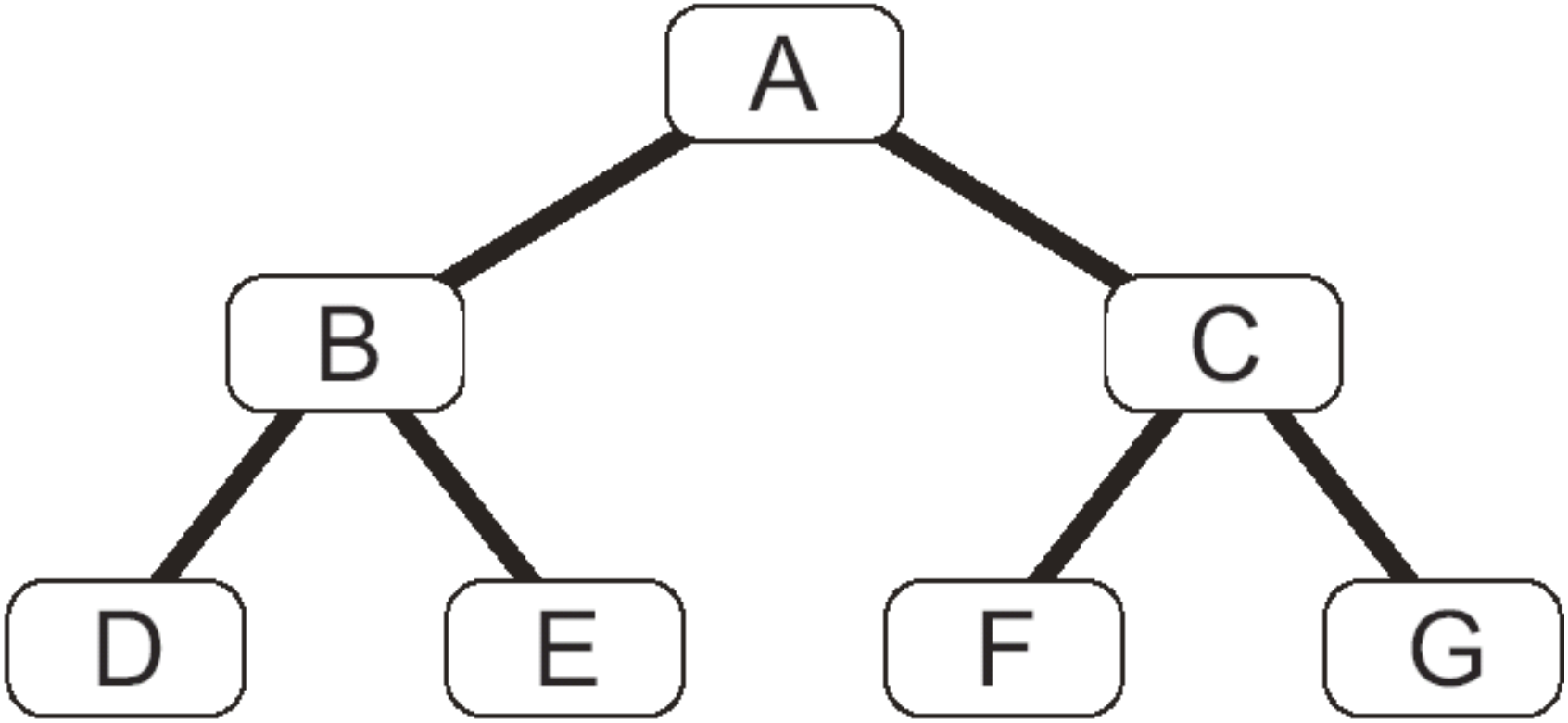


Löschen im Suchbaum (2)



Baumdurchläufe

Hochschule Ludwigshafen am Rhein



Inorder-Durchlauf

Hochschule Ludwigshafen am Rhein

```
private void printInorder (Knoten k) {
  if ( k != null ) {
    printInorder(k.links);
    System.out.print(k);
    printInorder(k.rechts);
  }
}
public void printInorder() {
  printInorder(this.wurzel);
  System.out.println();
}
```

Preorder-Durchlauf

Hochschule Ludwigshafen am Rhein

```
private void printPreorder (Knoten k) {
  if ( k != null ) {
    System.out.print(k);
    printPreorder(k.links);
    printPreorder(k.rechts);
  }
}
public void printPreorder() {
  printPreorder(this.wurzel);
  System.out.println();
}
```

Postorder-Durchlauf

Hochschule Ludwigshafen am Rhein

```
private void printPostorder (Knoten k) {
  if ( k != null ) {
    printPostorder(k.links);
    printPostorder(k.rechts);
    System.out.print(k);
  }
}
public void printPostorder() {
  printPostorder(this.wurzel);
  System.out.println();
}
```

Divide and Conquer: Pseudocode

Hochschule Ludwigshafen am Rhein

Methode DIVANDCONQ (Problem)
falls [Problem klein]
dann [explizite Lösung];
sonst [Teile Problem auf in P_1 , ... P_k];
DIVANDCONQ (P_1);
…
DIVANDCONQ (P_k);
[Setze Lösung für Problem aus Lösungen für P_1 ,.., P_k zusammen]

Quicksort Pseudocode

Hochschule Ludwigshafen am Rhein

```
Modul QuickSort1 (Liste)
bestimme Teilungselement PIVOT;
teile L in L1 und L2 so daß gilt:
        alle Elemente in L1 sind kleiner als PIVOT, und
        alle Elemente in L2 sind größer als PIVOT;
falls L1 oder L2 hat mehr als ein Element
dann
        setze L1 = QuickSort (L1);
        setze L2 = QuickSort (L2);
Gib L1 + [ PIVOT ] + L2 zurück;
```

Falls das Pivotelement nicht in L1 oder L2 enthalten ist

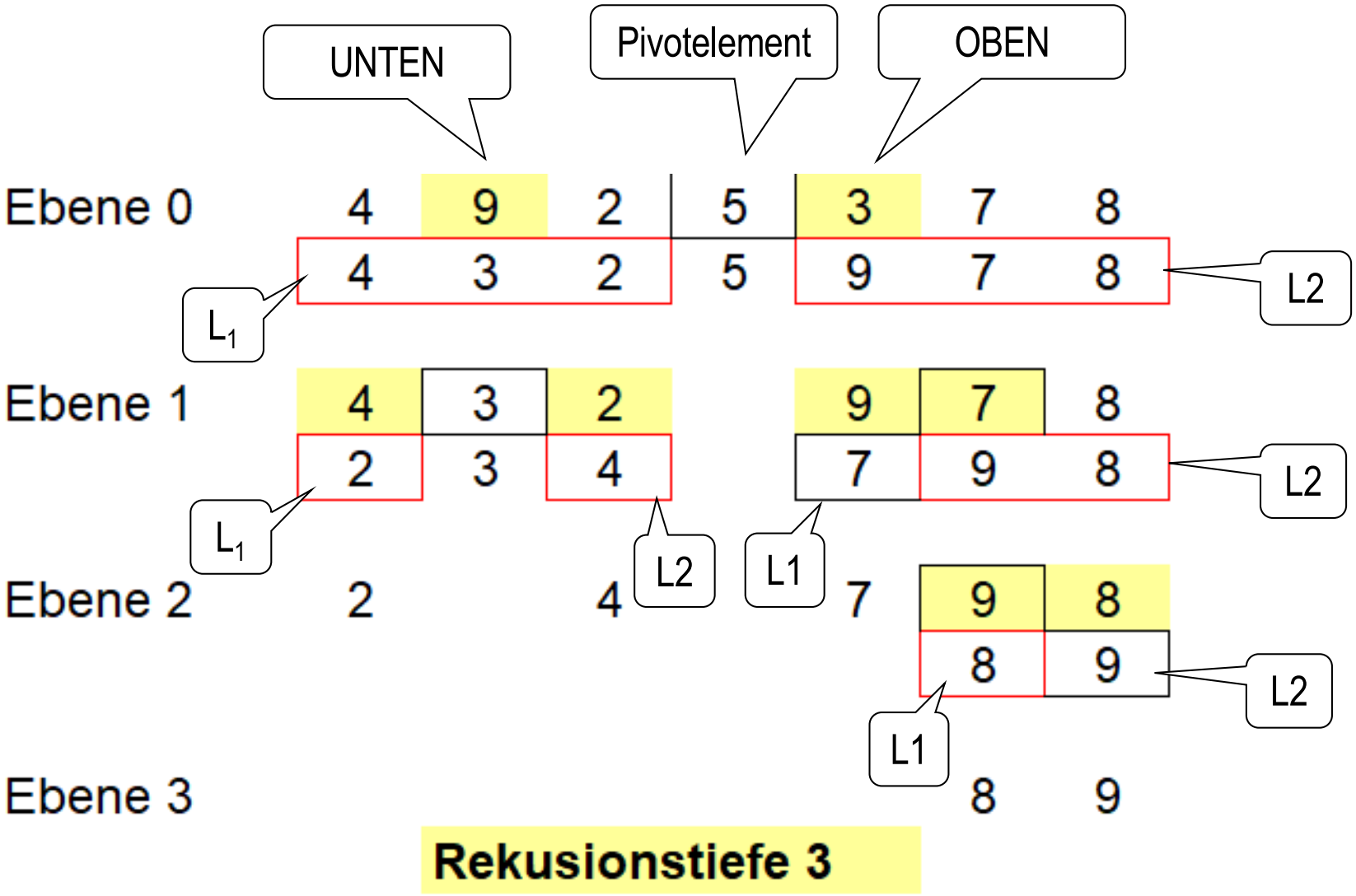
Quicksort Aufteilung

Hochschule Ludwigshafen am Rhein

Falls n die Anzahl der Elemente des Feldes und 0 der untere Index;
setze PIVOT = Feld[beliebig aus 0 bis n-1, z.B. Mitte];
setze UNTEN = 0;
setze OBEN = n-1;
Solange UNTEN $\leq$ OBEN
führe aus
    solange Feld[UNTEN] < PIVOT
    führe aus
        UNTEN = UNTEN + 1;
    solange Feld[OBEN] > PIVOT
    führe aus
        OBEN = OBEN - 1;
    falls UNTEN $\leq$ OBEN
        vertausche Inhalte von Feld[UNTEN] und Feld[OBEN];
        UNTEN = UNTEN + 1;
        OBEN = OBEN - 1;

Danach kann wie folgt aufgeteilt werden:
L1 = Feld[0..OBEN]
L2 = Feld[UNTEN..n-1]

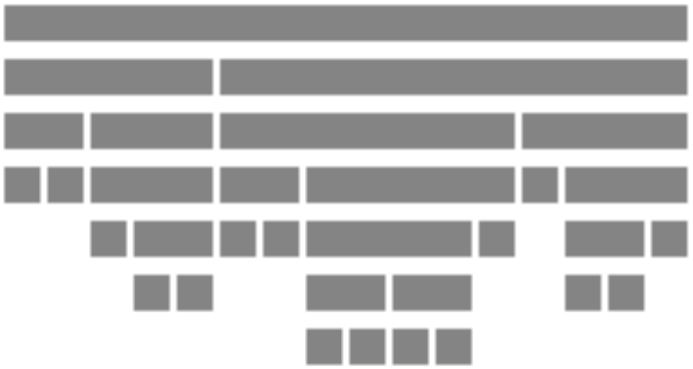
Quicksort Aufteilung



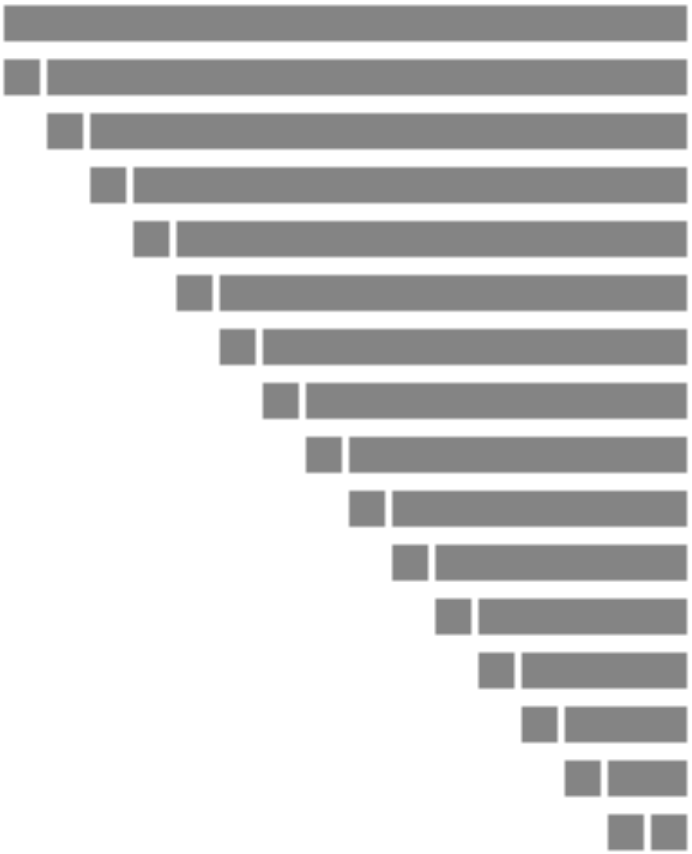
Quicksort: Rekursionstiefe

Hochschule Ludwigshafen am Rhein

(a)



(b)



(c)

Binäre Suche: Divide and Conquer

Hochschule Ludwigshafen am Rhein

Stelle	0	1	2	3	4	5	6	7	8	9
1. Schritt	unter				mitte					ober
Wert	0	1	2	4	5	8	9	12	13	18
2. Schritt						unter		mitte		ober
Wert	0	1	2	4	5	8	9	12	13	18
3. Schritt						unter	ober			
Wert	0	1	2	4	5	8	9	12	13	18
						mitte				

- **Idee Divide and Conquer:**
  - **Teile Array in 2 Hälften und suche in der Hälfte weiter, welche die Sucheeigenschaft enthalten muss**
  - **fahre in dieser Weise fort**

Binäre Suche Divide and Conquer: Pseudcode

Hochschule Ludwigshafen am Rhein

```
Modul BinäreSuche (folge, Sucheigenschaft, UntereGrenze, ObereGrenze)
Setze Mitte = ( UntereGrenze + ObereGrenze) / 2
Falls Element von Mitte = Sucheigenschaft
        dann   Gib Position des Elementes zurück;
Falls UntereGrenze >= ObereGrenze      //nicht gefunden
        dann   Gib -1 zurück;

Falls Element von Mitte > Sucheigenschaft   //unten weitersuchen
        dann  Gib  BinäreSuche  (folge,  Sucheigenschaft,  UntereGrenze,  Mitte  -  1)
              zurück;
        sonst Gib  BinäreSuche  (folge,  Sucheigenschaft,  Mitte  +  1,  ObereGrenze)
              zurück;
```