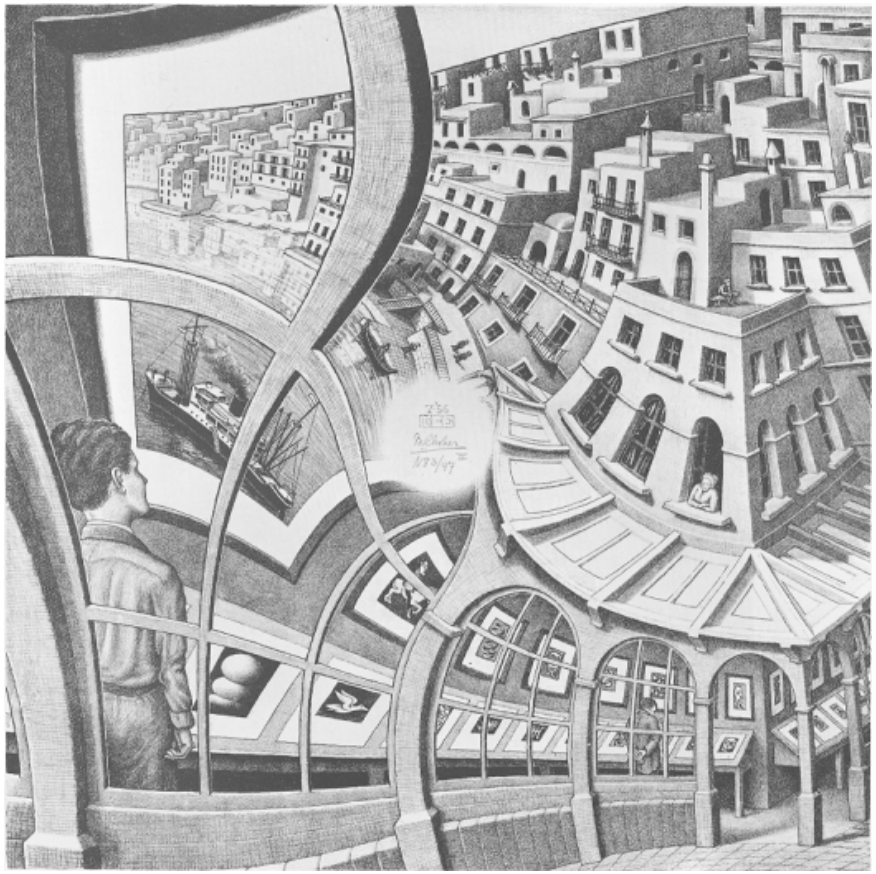
Kapitel 7: Rekursion

- Grundbegriffe
- Beispiele
  - Türme von Hanoi
  - Größter gemeinsamer Teiler
  - Graphische Darstellung von Bäumen
  - Linear verkettete Listen
- Teile-und-Herrsche-Verfahren
  - Potenzfunktion
  - Binäre Suche
- Endrekursion
- Rekursion und Keller

Rekursion

- Rekursion bedeutet wörtlich Zurückführen.
- Rekursion liegt dann vor, wenn eine Funktion, ein Algorithmus, eine Datenstruktur, ein Begriff, etc. durch sich selbst definiert wird.



M. C. Escher, Bildergalerie, 1956.

Rekursive Datentypen und Funktionen

- Linear verkettete Listen und Bäume (später) sind Beispiele für rekursiv definierte Datentypen.
- Beispiel: linear verkettete Liste

```
class Node {
    Node next;
    int  data;
    // ...
}
```

Node wird durch sich selbst definiert.

- Eine rekursive Funktion ist eine Funktion, die sich selbst aufruft.
- Beispiel: Fakultätsfunktion fak(n) = n*(n-1)*…*2*1

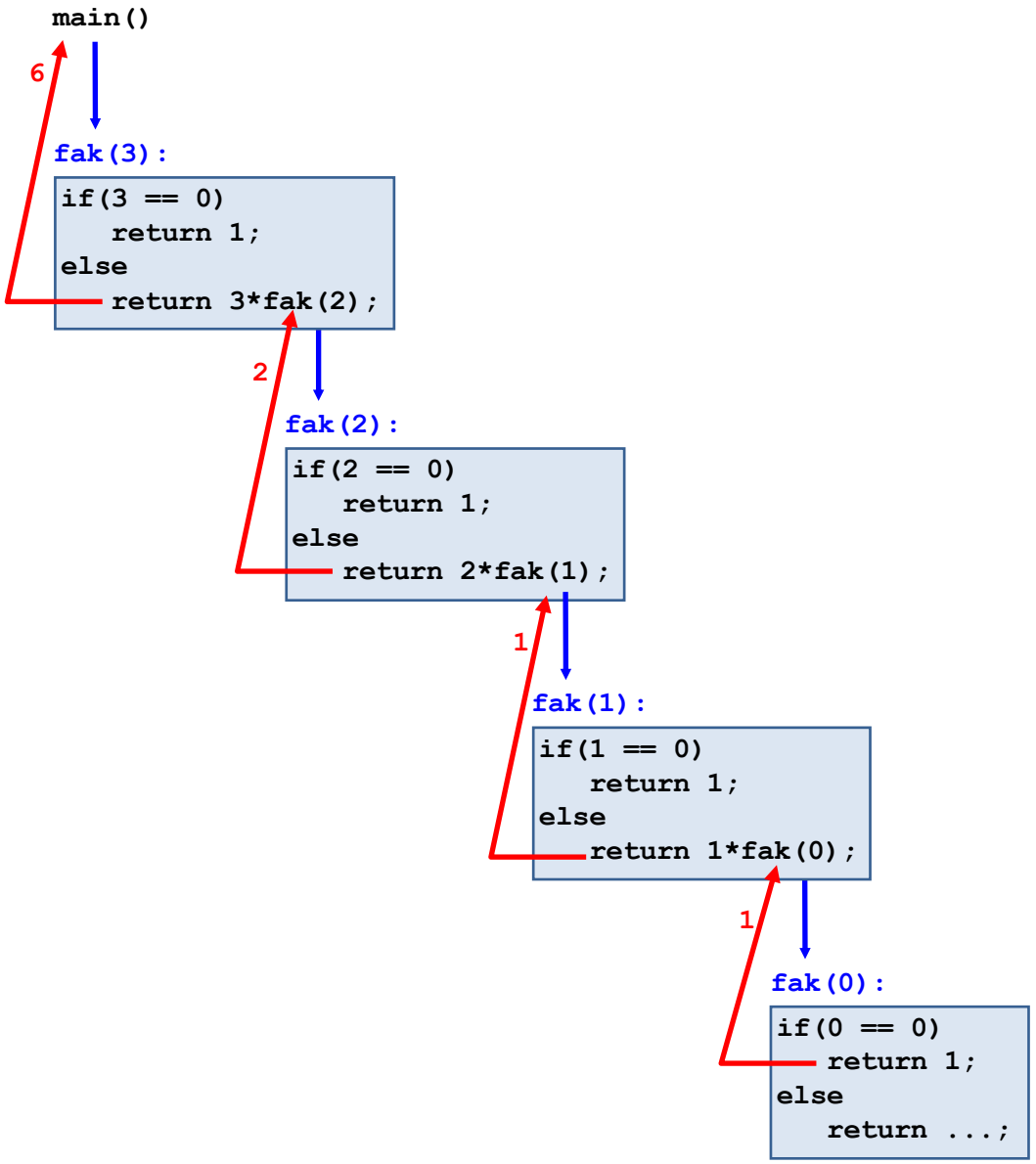
```
int fak(int n) {
    if (n == 0)
        return 1;
    else
        return n * fak(n-1) ;
}
```

fak ruft sich selbst auf.

Aufruf einer rekursiven Funktion - Beispiel

```
void main(){
    fak(3);
}

int fak(int n){
    if (n == 0)
        return 1;
    else
        return n*fak(n-1) ;
}
```



Aufrufstruktur und Rekusionstiefe

Aufrufstruktur

Kompakte Darstellung sämtlicher rekursiver Aufrufe einer rekursiven Funktion

Rekursionstiefe

Anzahl der geschachtelten Aufrufe einer rekursiven Funktion.

Wir werden uns oft für die maximale Rekursionstiefe interessieren.

Beispiel: fak(3)

Aufrufstruktur:	**Rekursionstiefe:**
fak(3)	0
fak(2)	1
fak(1)	2
fak(0)	3

Rekusionstiefe und Laufzeitstack

- Zur Laufzeit wird bei jedem Funktionsaufruf ein Call-Frame bestehend aus
  - Parameter,
  - Rücksprungadresse und
  - lokale Variablen

  in den Laufzeit-Stack abgelegt.

- Das bedeutet, dass große Rekursionstiefen den Laufzeit-Stack belasten und bei einer zu großen Rekursionstiefe der Laufzeit-Stack überläuft (Stack Overflow Error Exception).

- Zu große Rekursionstiefen vermeiden und insbesondere auf Endlos-Rekursion achten:

```
void main() {
    fak(3);
}

int fak(int n) {
    if (n == 0)
        return 1;
    else
        return n*fak(n) ;
}
```

Endlos-Rekursion:
Stack Overflow Error Exception

Vorgehensweise bei rekursiver Programmierung

Problemstellung

- Gesucht ist eine rekursive Funktion zur Lösung eines Problems P der Größe n ($n \geq 0$).
- Beispiele: fak(n), Sortieren von n Zahlen, Suchen von x in n Zahlen, ...

Vorgehensweise

- Rekursionsfall:

  Reduziere Problem der Größe n auf ein Problem der Größe k mit $0 \leq k < n$ (oder evtl. mehrere Probleme).

  Beispiel: bei der Fakultätsfunktion wird fak(n) zurückgeführt auf n*fak(n-1)

- Basisfall (bzw. Basisfälle):

  Löse P für alle Werte n direkt, die sich im Rekursionsfall nicht weiter reduzieren lassen.

  Beispiel: bei der Fakultätsfunktion ist der Basisfall fak(0).

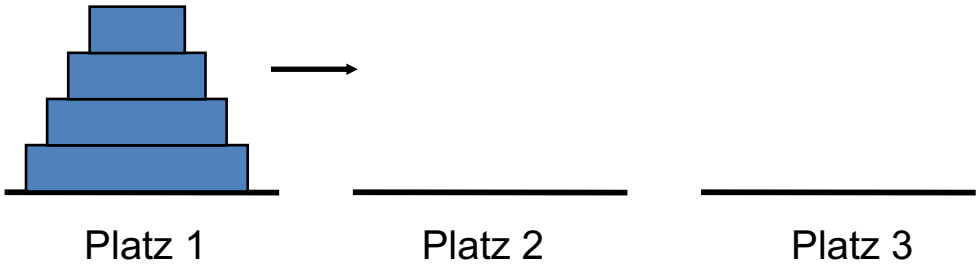
Kapitel 7: Rekursion

- Grundbegriffe
- Beispiele
  - Türme von Hanoi
  - Größter gemeinsamer Teiler
  - Graphische Darstellung von Bäumen
  - Linear verkettete Listen
- Teile-und-Herrsche-Verfahren
  - Potenzfunktion
  - Binäre Suche
- Endrekursion
- Rekursion und Keller

Türme von Hanoi (1)

Aufgabenstellung

- n Scheiben unterschiedlichen Durchmessers, die der Größe nach sortiert übereinander liegen, bilden mit der größten Scheibe unten einen Turm. Der Turm soll von einem Platz 1 nach einem Platz 2 transportiert werden.
- Dabei steht ein Hilfsplatz 3 zur Verfügung.
- Es darf jeweils nur die oberste Scheibe eines Turms bewegt werden.
- Außerdem darf auf eine Scheibe nur eine kleinere Scheibe gelegt werden.

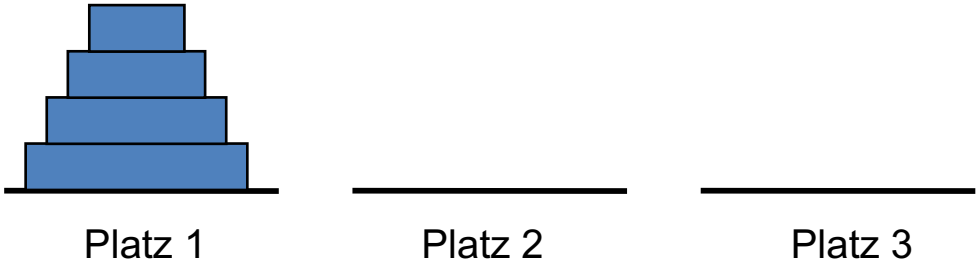


Türme von Hanoi (2)

Methode bewegeTurm

```
void bewegeTurm(int n, int s, int z, int h);
```

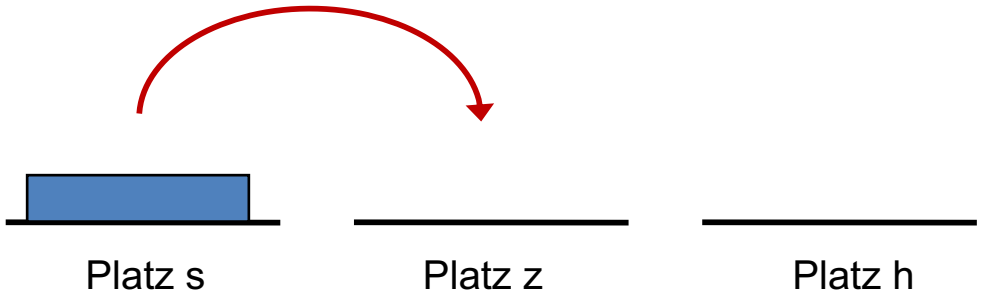
gibt die notwendigen Scheibenbewegungen aus, um ein Turm mit n Scheiben vom Startplatz s zum Zielplatz z zu bewegen. Dabei ist h ein zusätzlicher Hilfsplatz.



Ziel: Rekursive Lösung für bewegeTurm

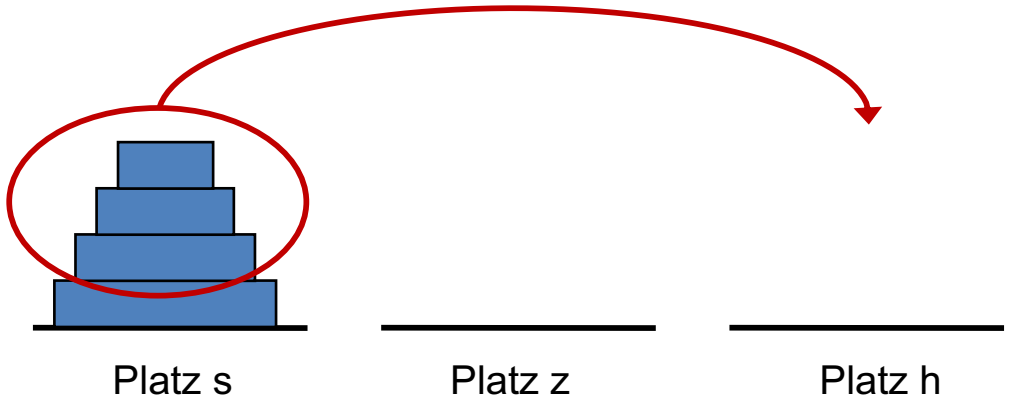
Türme von Hanoi – Rekursive Lösung

```
static void bewegeTurm(int n, int s, int z, int h)
{
   if (n == 1)
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
   else {
      bewegeTurm(n-1,s,h,z);
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
      bewegeTurm(n-1,h,z,s);
   }
}
```



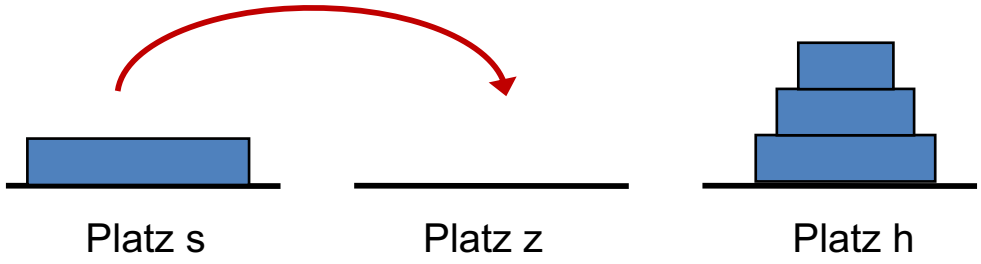
Türme von Hanoi – Rekursive Lösung

```
static void bewegeTurm(int n, int s, int z, int h)
{
   if (n == 1)
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
   else {
      bewegeTurm(n-1,s,h,z);
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
      bewegeTurm(n-1,h,z,s);
   }
}
```



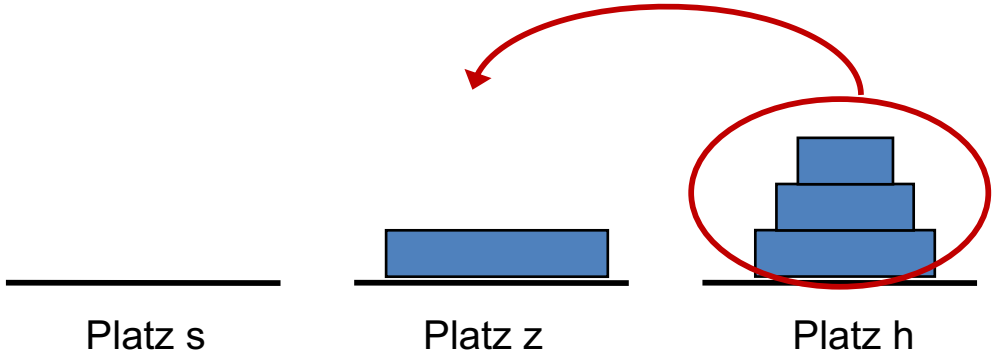
Türme von Hanoi – Rekursive Lösung

```
static void bewegeTurm(int n, int s, int z, int h)
{
   if (n == 1)
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
   else {
      bewegeTurm(n-1,s,h,z);
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
      bewegeTurm(n-1,h,z,s);
   }
}
```



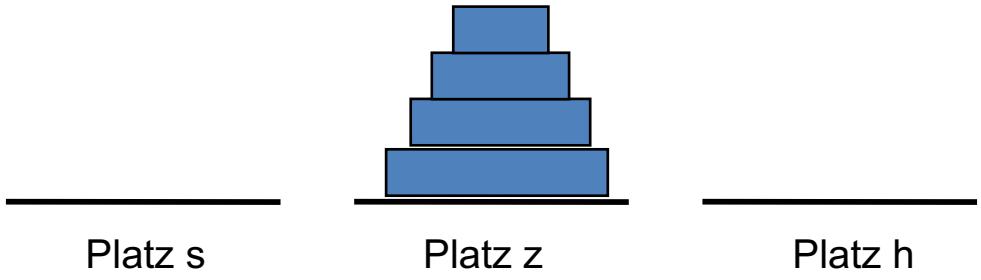
Türme von Hanoi – Rekursive Lösung

```
static void bewegeTurm(int n, int s, int z, int h)
{
   if (n == 1)
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
   else {
      bewegeTurm(n-1,s,h,z);
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
      bewegeTurm(n-1,h,z,s);
   }
}
```

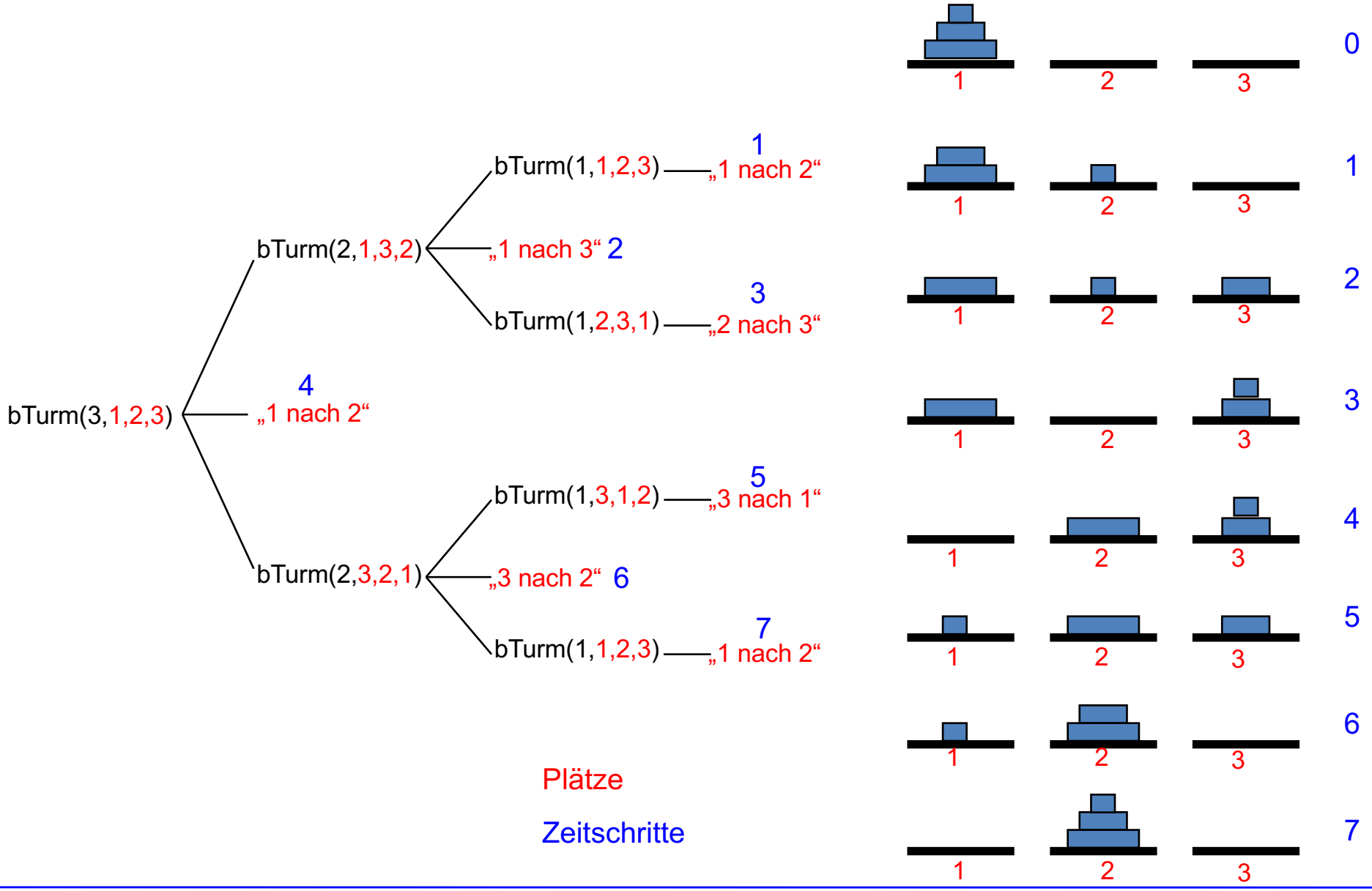


Türme von Hanoi – Rekursive Lösung

```
static void bewegeTurm(int n, int s, int z, int h)
{
   if (n == 1)
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
   else {
      bewegeTurm(n-1,s,h,z);
      System.out.println("Bewege Scheibe von " + s + " nach " + z);
      bewegeTurm(n-1,h,z,s);
   }
}
```



Animation des Aufrufs bewegeTurm(3,1,2,3)



Türme von Hanoi – Aufgabe 7.1

a) Wie groß ist die maximale Rekursionstiefe R(n) bei Aufruf von bewegeTurm(n,1,2,3)?

b) Wieviel Scheiben S(n) müssen transportiert werden, um einen Turm der Größe n vom Start- zum Zielplatz zu bewegen?

Größter gemeinsamer Teiler - ggT

Aufgabenstellung

Gesucht ist eine rekursive Funktion ggt(n,m) zur Berechnung des größten gemeinsamen Teilers zweier ganzer Zahlen n, m $\geq$ 0.

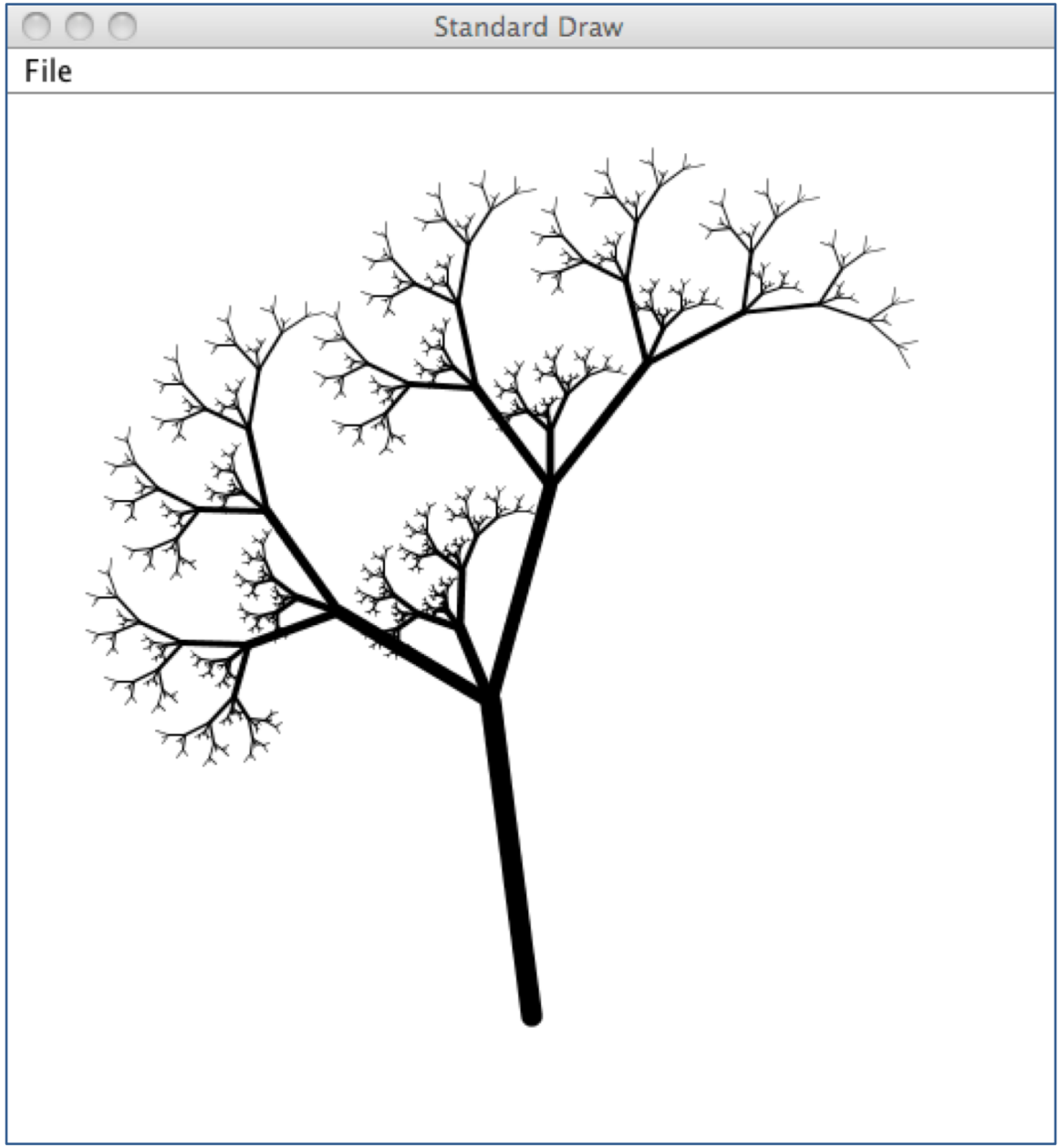
Es gilt folgende Eigenschaft von ggt:

ggt(m, n) = ggt(n, m mod n) für n > 0

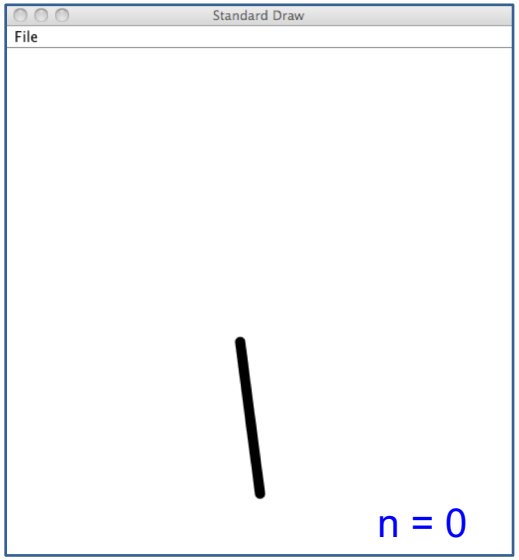
Rekursive Lösung

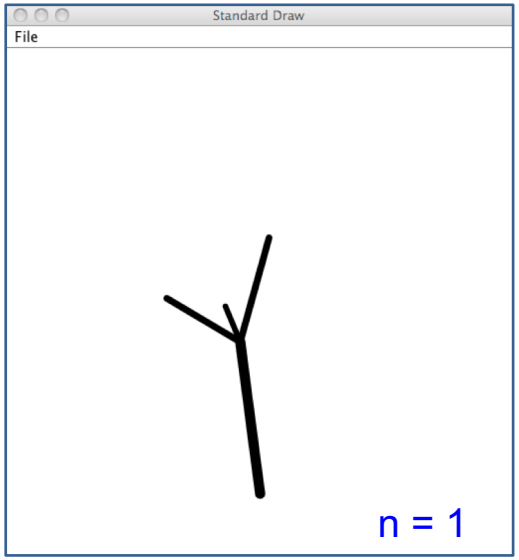
```
static int ggt(int m, int n)
{
   if (n == 0)
      return m;
   else
      return ggt(n, m%n);
}
```

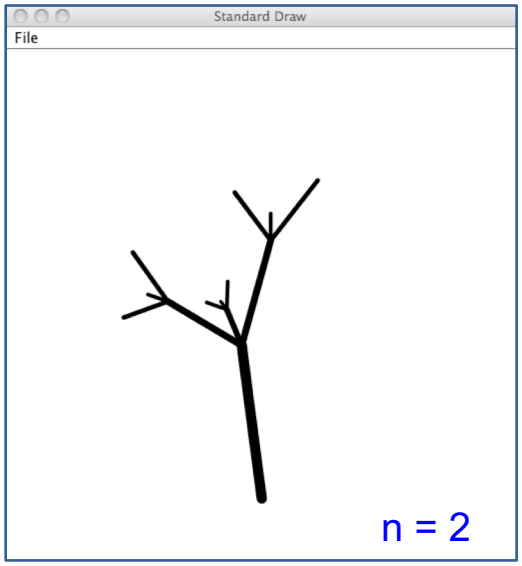
Grafische Darstellung eines Baums (1)



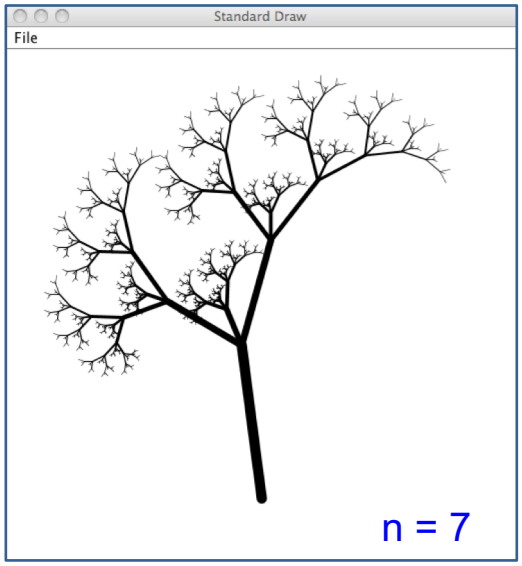
Grafische Darstellung eines Baums (2)







...

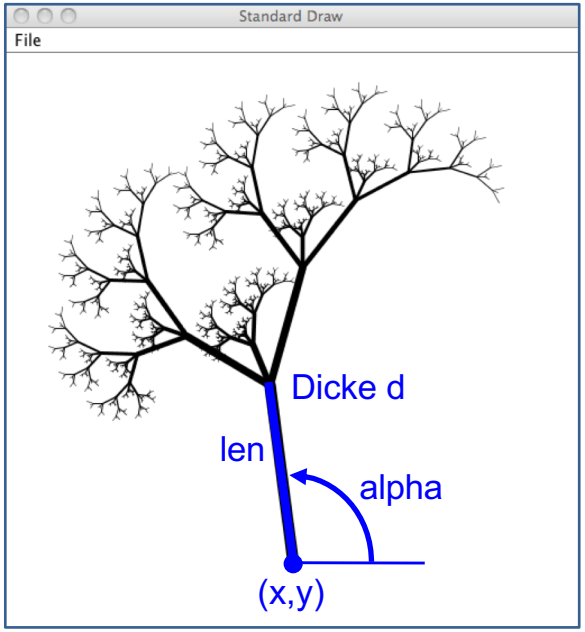


Verästelungstiefe n

Grafische Darstellung eines Baums (3)

```
static void draw(double x, double y, double alpha,
                 double len, double d, int n) {
   if (n >= 0) {
      double xe = x + len*Math.cos(alpha);
      double ye = y + len*Math.sin(alpha);
      StdDraw.setPenRadius(d);
      StdDraw.line(x, y, xe, ye);
      draw(xe, ye, alpha+0.90, len*0.55, d/1.5, n-1);
      draw(xe, ye, alpha+0.25, len*0.25, d/1.8, n-1);
      draw(xe, ye, alpha-0.40, len*0.70, d/1.5, n-1);
   }
}

public static void main() {
   StdDraw.setXscale(-6, +6);
   StdDraw.setYscale(-1, +11);
   draw(0, 0, 1.7, 4.0, 0.02, 7);
}
```



Die drei Äste sind gegenüber aktuellem Ast gedreht um:

alpha + 0.90 = alpha + 51.6°
alpha + 0.25 = alpha + 14.3°
alpha − 0.40 = alpha – 22.9°

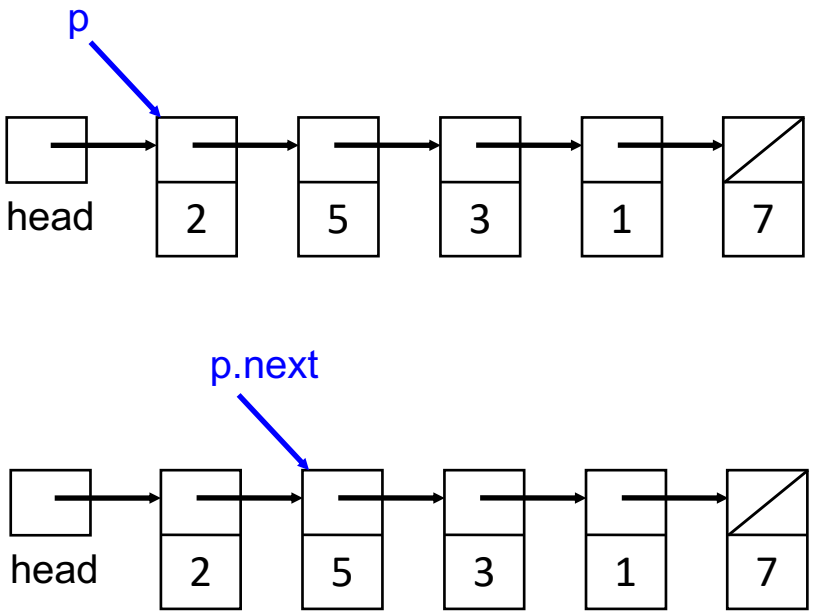
Neigung des Baumstamms ist 1.7 = 97.4°.

- draw zeichnet einen Ast vom Startpunkt (x,y) mit der Neigung alpha (in rad), der Länge len und der Dicke d und zeichnet am Endpunkt des Asts (xe,ye) drei weitere Äste mit jeweils Verästelungstiefe n-1.
- StdDraw ist eine Klasse von http://introcs.cs.princeton.edu/home/ und gestattet einfache Zeichenoperationen in einem Fenster.

Rekursion über linear verkettete Listen (1)

Ansatz

Problem für Liste p wird zurückgeführt auf Problem für Liste p.next.
Beachte, dass Liste p.next ein Knoten weniger enthält.



Beispiel:

rekursives Ausgeben aller Knoten

```
void printR(Node p) {
    if (p != null) {
        System.out.println(p.data);
        printR(p.next);
    }
}
```

Rekursion über linear verkettete Listen (2)

```
public class LinkedList {

    private static class Node {
        int data;
        Node next;
        Node(Node p, int x) {
            data = x;
            next = p;
        }
    }
    private Node head;

    public LinkedList() {head = null;}

    public void insert(int x) {
        head = new Node(head,x);
    }

    // ...
```

Rekursion über linear verkettete Listen (3)

```
// ...

public void printR() {
    printR(head);
}

private void printR(Node p) {
    if (p != null) {
        System.out.println(p.data);
        printR(p.next);
    }
}

public void print() {
    for (Node p = head; p != null; p = p.next)
        System.out.println(p.data);
}

// ...
```

printR:

Rekursive Ausgabe der linear verketteten Liste.

print:

Zum Vergleich iterative Ausgabe der linear verketteten Liste.

Rekursion über linear verkettete Listen (4)

```
public void eraseR(int x) {
     head = eraseR(head, x);
}

private Node eraseR(Node p, int x) {
     if (p == null)
          return null;
     else if (p.data == x)
          return p.next;
     else {
          p.next = eraseR(p.next,x);
          return p;
     }
}
```

eraseR:

rekursives Löschen des ersten Vorkommens von x in der Liste

```
public void erase(int x) {
     if (head == null)
          return;
     else if (head.data == x)
            head = head.next;
     else {
          Node p = head;
          while (p.next != null && x != p.next.data)
               p = p.next;
          if (p.next != null)
               p.next = p.next.next;
     }
}
```

erase:

Zum Vergleich iteratives Löschen des ersten Vorkommens von x in der Liste

Kapitel 7: Rekursion

- Grundbegriffe
- Beispiele
  - Türme von Hanoi
  - Größter gemeinsamer Teiler
  - Graphische Darstellung von Bäumen
  - Linear verkettete Listen
- Teile-und-Herrsche-Verfahren
  - Potenzfunktion
  - Binäre Suche
- Endrekursion
- Rekursion und Keller

Teile-und-Herrsche-Verfahren

Motivation

- Bei zahlreichen Problemstellungen (z.B. Sortieren, Suchen, geometrische Algorithmen) führen Teile-und-Herrsche-Verfahren zu sehr effizienten Lösungen.

Vorgehensweise

```
if (Problem ist einfach zu lösen; z.B. Problemgröße n == 1) {
    löse Problem direkt;
}
else {
    // Teileschritt:
    Teile Problem in 2 Teilprobleme (oder 1 Teilproblem)
    in etwa der Größe n/2, wobei Teilprobleme derselben Art sind
    wie Ausgangsproblem;
    // Herrscheschritt:
    Löse Teilprobleme rekursiv;
    Setze Teillösungen zu einer Gesamtlösung zusammen;
}
```

Beispiel Potenzfunktion (1)

Aufgabenstellung

Gesucht ist eine Teile-und-Herrsche-Funktion

$$pot(x,n) = x^n \text{ , } n \in N$$

Idee für Teile- und Herrsche-Schritt:

$x^n = x^{n/2} * x^{n/2}$, falls n gerade und $n \geq 2$

$x^n = x * x^{n/2} * x^{n/2}$, falls n ungerade und $n \geq 3$

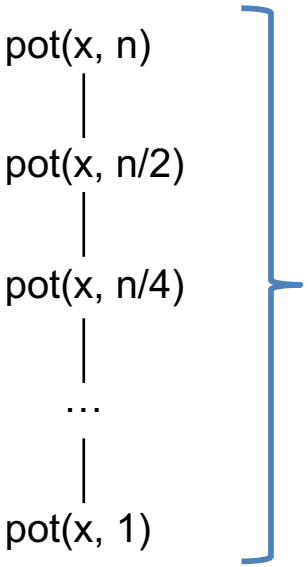
Bei n/2 wird ganzzahlige Division vorausgesetzt.

Rekursive Teile-und-Herrsche-Funktion

```
static double pot(double x, int n) {
    if (n == 1)
        return x;
    else  {
        double p = pot(x,n/2);
        if (n%2 == 0)    // n gerade
            return p*p;
        else
            return x*p*p;
    }
}
```

Beispiel Potenzfunktion (2)

Aufrufstruktur und maximale Rekursionstiefe



Maximale Rekursionstiefe:

$$R(n) = \lfloor \log_2 n \rfloor$$

($\lfloor x \rfloor$ = x abgerundet)

Beispiel:

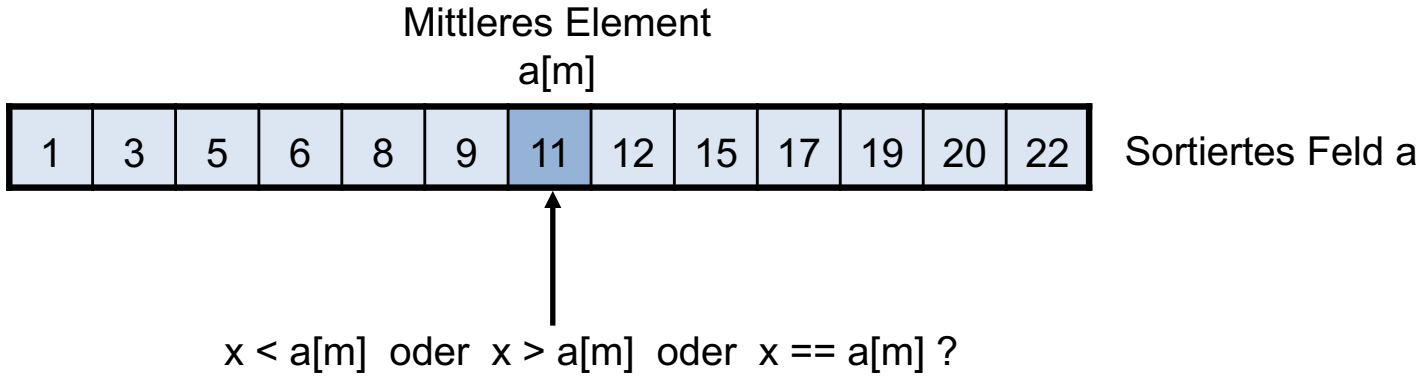
- Der Aufruf von pot(x, 1000) führt zu einer maximalen Rekursionstiefe von:
  $R(1000) = \lfloor \log_2 1000 \rfloor = 9$.

Beispiel Binäre Suche (1)

Aufgabenstellung

- Suche x in einem sortierten und lückenlos gefülltem Feld a.
- Falls x gefunden wird, dann soll der Index zurückgeliefert werden und sonst -1 (nicht gefunden).

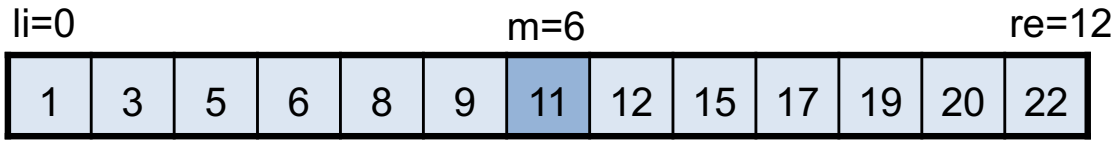
Idee für Teile-und-Herrsche-Schritt:

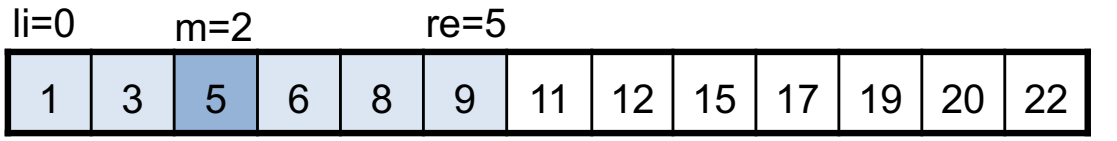


Falls x == a[m],	dann gefunden.
Falls x < a[m],	dann suche in linker Hälfte weiter.
Falls x > a[m],	dann suche in rechter Hälfte weiter

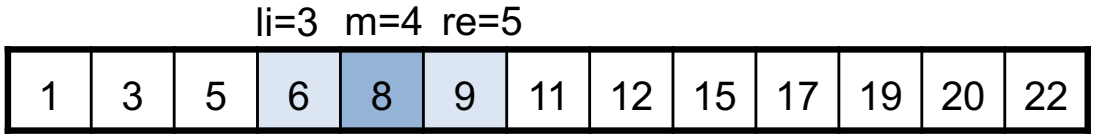
Beispiel Binäre Suche (2)

Beispiel: suche x = 8





Suche in linker Hälfte



Suche in rechter Hälfte;
x wird gefunden!

Zu durchsuchender Bereich geht von a[li] bis a[re]

Mittleres Element m = (li + re)/2

Beispiel Binäre Suche (3)

```
private static int binSuche(int[] a, int li, int re, int x) {

    if (re < li)
        return -1;
    else {
        int m = (li + re)/2;
        if (x < a[m])
            return binSuche(a, li, m-1, x);
        else if (x > a[m])
            return binSuche(a, m+1, re, x);
        else // x == a[m]
            return m;
    }
}

public static int binSuche(int[] a, int x) {
    assert isSorted(a);
    return binSuche(a,0,a.length-1,x);
}
private static boolean isSorted(int[] a) {
    for (int i = 0; i < a.length-1; i++)
        if (a[i+1] < a[i])
            return false;
    return true;
}
```

binSuche durchsucht
a[li], a[li+1], ..., a[re] nach x und liefert i zurück, falls a[i] == x, sonst -1.

Basisfall: leeres Teilfeld

Das Feld a muss aufsteigend sortiert sein!

(aufgrund der Laufzeit assert statt Exception; Prüfung lässt sich abschalten)

Kapitel 7: Rekursion

- Grundbegriffe
- Beispiele
  - Türme von Hanoi
  - Größter gemeinsamer Teiler
  - Graphische Darstellung von Bäumen
  - Linear verkettete Listen
- Teile-und-Herrsche-Verfahren
  - Potenzfunktion
  - Binäre Suche
- Endrekursion
- Rekursion und Keller

Endrekursion

Definition

Ein rekursiver Aufruf heißt endrekursiv, falls unmittelbar nach dem Aufruf die Funktion verlassen wird.

(Endrekursion auf engl.: tail recursion)

Beispiel:

```
static void print(Node p)
{
    if (p != null)
    {
        System.out.println(p.data);
        print(p.next);
    }
}
```

Aufruf ist endrekursiv.

Aufgabe 7.2

Untersuchen Sie einige der bisher besprochenen rekursiven Funktionen auf Endrekursion.

Eliminierung der Endrekursion (1)

- Ein endrekursiver Aufruf verhält sich wie eine Schleife und kann daher durch eine Schleife ersetzt werden.
- Man beachte, dass die iterative Funktion (d.h. Funktion ohne Rekursion) resourcensparender ist. Warum?

```
static void print(Node p)
{
     if (p != null)
     {
          System.out.println(p.data);
          print(p.next);
     }
}
```

```
static void print(Node p)
{
     while (p != null)
     {
          System.out.println(p.data);
          p = p.next;
     }
}
```

Eliminierung der Endrekursion (2)

Allgemeines Schema

```
RT fun(T x) {
    if (Basisfall)
        return r;
    else {
        A
        return fun(a);
    }
}
```

```
RT fun(T x) {
    while(!Basisfall)
        A
        x = a;
    }
    return r;
}
```

- RT steht für einen beliebigen Rückgabewerttyp.
  Der Rückgabewerttyp kann auch void sein.
- T steht für einen beliebigen Parametertyp.
  Im allgemeinen kann die Funktion fun auch mehrere Parameter haben.
- A steht für einen beliebigen Anweisungsblock.

Aufgabe

Aufgabe 7.3

Beseitigen Sie die Endrekursion in der binären Suche.

```
private static int binSuche(int[] a, int li, int re, int x) {
    if (re < li)
        return -1;
    else {
        int m = (li + re)/2;
        if (x < a[m])
            return binSuche(a, li, m-1, x);
        else if (x > a[m])
            return binSuche(a, m+1, re, x);
        else // x == a[m]
            return m;
    }
}
```

Keller und Rekursion (1)

- Endrekursive Aufrufe lassen sich einfach (d.h. schematisch) durch eine Schleife ersetzen.
- Nicht-endrekursive Aufrufe lassen sich prinzipiell mit Hilfe eines Kellers beseitigen. Manchmal kann Rekursion auch ohne Hilfe eines Kellers beseitigt werden
- Beseitigung von nicht-endrekursiven Funktionen mit Hilfe eines Kellers ist in der Regel nicht ratsam, soll aber trotzdem am Beispiel der Türme von Hanoi gezeigt werden.

```
static void bewegeTurm(int n, int s, int z, int h)
{
      if (n == 1)
            System.out.println("Bewege Scheibe von " + s + " nach " + z);
      else {
            bewegeTurm(n-1,s,h,z);
            System.out.println("Bewege Scheibe von " + s + " nach " + z);
            bewegeTurm(n-1,h,z,s);
  }
}
```

Keller und Rekursion (2)

- Idee: Keller als Aufgabenstapel.
- Speichere im Keller zu erledigende Aufgaben als Quadrupel (n, s, z, h) ab:
  bewege n Scheiben von s nach z mit Hilfsplatz h.
- Beachte LIFO-Organisation des Kellers:
  Reihenfolge beim Auskellern eines Quadrupels ist umgekehrt zum Einkellern.

```
private static void bewegeTurm(int n, int s, int z, int h) {

        Deque<Integer> stack = new LinkedList<>();
        stack.push(n); stack.push(s); stack.push(z); stack.push(h);

        while (! stack.isEmpty()) {
                h = stack.pop(); z = stack.pop(); s = stack.pop(); n = stack.pop();
                if (n == 1)
                        System.out.println("Bewege Scheibe von " + s + " nach " + z);
                else {
                        stack.push(n-1); stack.push(h); stack.push(z); stack.push(s);
                        stack.push(1);    stack.push(s); stack.push(z); stack.push(h);
                        stack.push(n-1); stack.push(s); stack.push(h); stack.push(z);
                }
        }
}
```

Initiale Aufgabe einkellern

Solange Keller nicht leer ist, hole die oberste Aufgabe vom Keller und erledige sie

Neue Aufgaben einkellern.