
II. Imperative und objektorientierte Programmierung

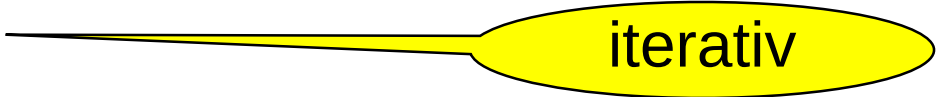
- 1. Grundelemente der Programmierung
- 2. Objekte, Klassen und Methoden
- 3. Rekursion und dynamische Datenstrukturen
- 4. Erweiterung von Klassen und fortgeschrittene Konzepte

II.3. Rekursion und dynamische Datenstrukturen

- 1. Rekursive Algorithmen
- 2. Rekursive (dynamische) Datenstrukturen

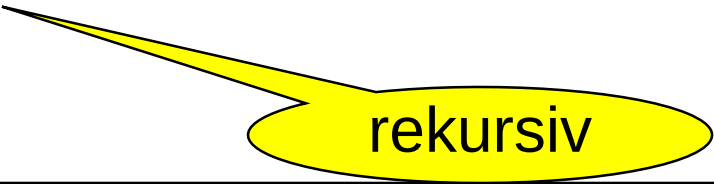
Fakultät

```
public static int fak (int x) {  
    int res = 1;  
    while (x > 1) {  
        res = x * res;  
        x = x - 1;  
    }  
    return res;  
}
```



iterativ

```
public static int fak (int x) {  
    if (x > 1) return x * fak (x - 1);  
    else      return 1;  
}
```



rekursiv

Fibonacci-Zahlen (nicht-lineare Rekursion)

$$\text{fib}(x) = \begin{cases} 0, & x < 1 \\ 1 & x = 1 \\ \text{fib}(x-1) + \text{fib}(x-2) & x \geq 2 \end{cases}$$

```
public static int fib (int x) {  
  
    if      (x < 1)      return 0;  
    else if (x == 1)     return 1;  
    else                return fib (x - 1) + fib (x - 2);  
  
}
```



nicht-linear

even & odd (verschränkte Rekursion)

```
public static boolean even (int x) {  
  
    if      (x == 0)      return true;  
    else if (x > 0)      return odd  (x - 1);  
    else                return odd  (x + 1);  
  
}
```



```
public static boolean odd  (int x) {  
  
    if      (x == 0)      return false;  
    else if (x > 0)      return even (x - 1);  
    else                return even (x + 1);  
  
}
```



sqrt (Endrekursion)

```
public static float sqrt (float uG, float oG, float x) {  
    float m, epsilon = 1e-3f;  
  
    m = (uG + oG) / 2;  
    if (oG - uG <= epsilon) return m;  
    else if (m * m > x) return sqrt (uG, m, x);  
    else return sqrt (m, oG, x);  
}
```

Endrekursion

```
public static int fak (int x) {  
    if (x > 1) return x * fak (x - 1);  
    else return 1;  
}
```

sqrt (Endrekursion)

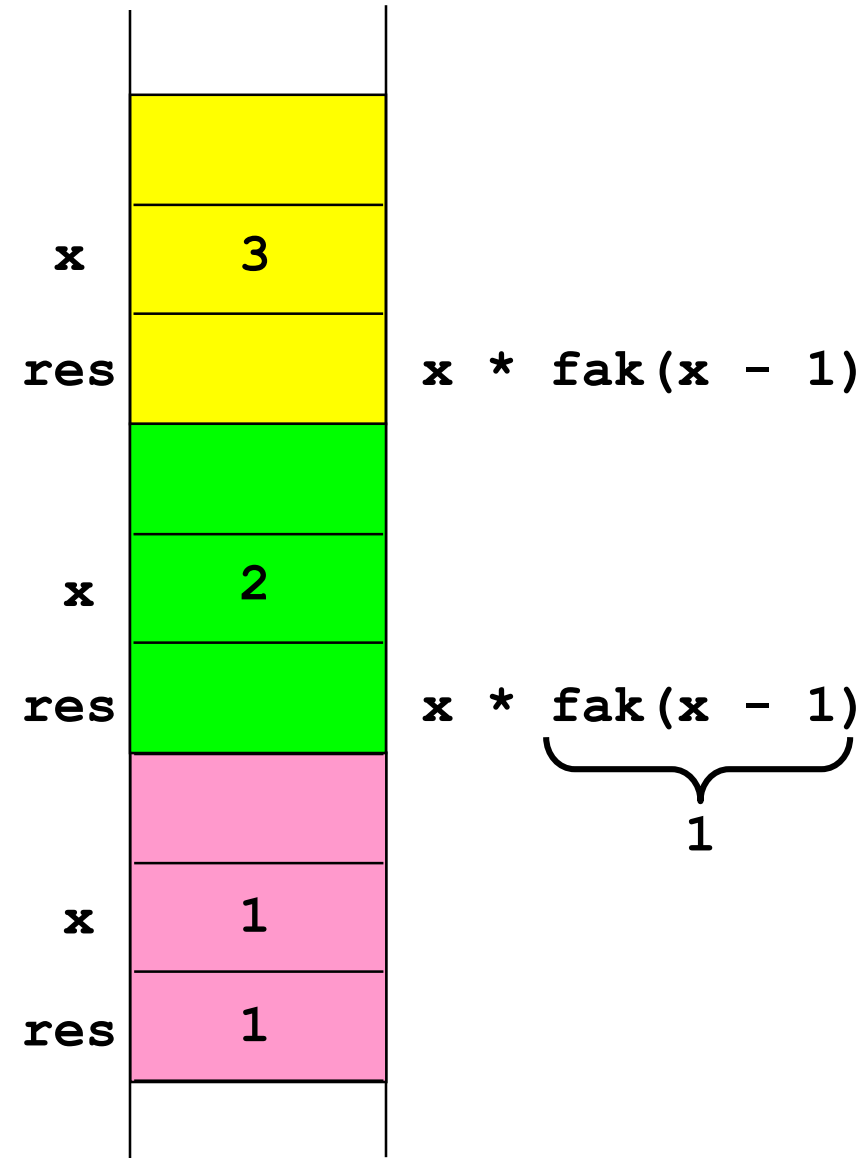
```
public static float sqrt (float uG, float oG, float x) {  
    float    m, epsilon = 1e-3f;  
  
    m = (uG + oG) / 2;  
    if      (oG - uG <= epsilon) return m;  
    else if (m * m > x)           return sqrt (uG, m, x);  
    else                                     return sqrt (m, oG, x);  
}
```

```
float    m, epsilon = 1e-3f;  
  
for (;;) {  
    m = (uG + oG) / 2;  
    if      (oG - uG <= epsilon) return m;  
    else if (m * m > x)           oG = m;  
    else                                     uG = m;  
}
```

Speicherorganisation bei Rekursion

```
public static int fak (int x) {  
    if (x > 1)  
        return x * fak (x - 1);  
    else return 1;  
}
```

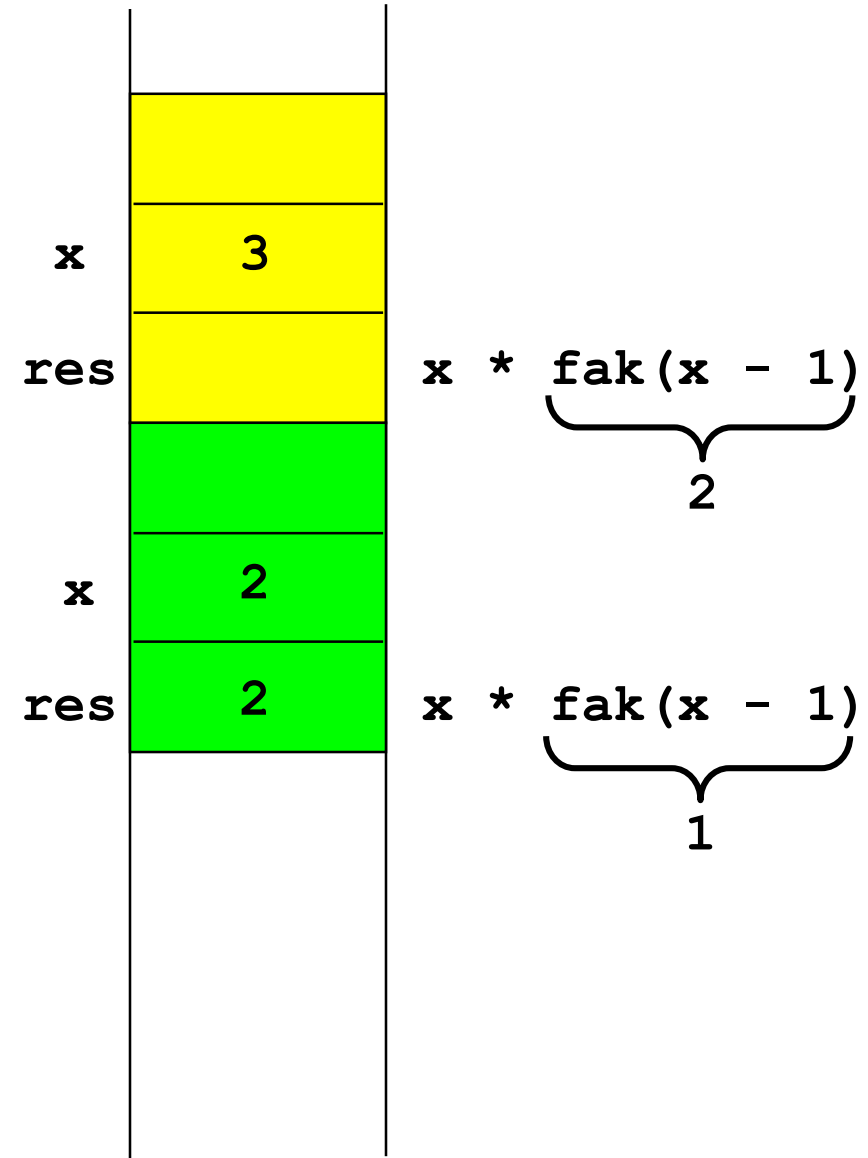
Aufruf: **fak (3)**



Speicherorganisation bei Rekursion

```
public static int fak (int x) {  
    if (x > 1)  
        return x * fak (x - 1);  
    else return 1;  
}
```

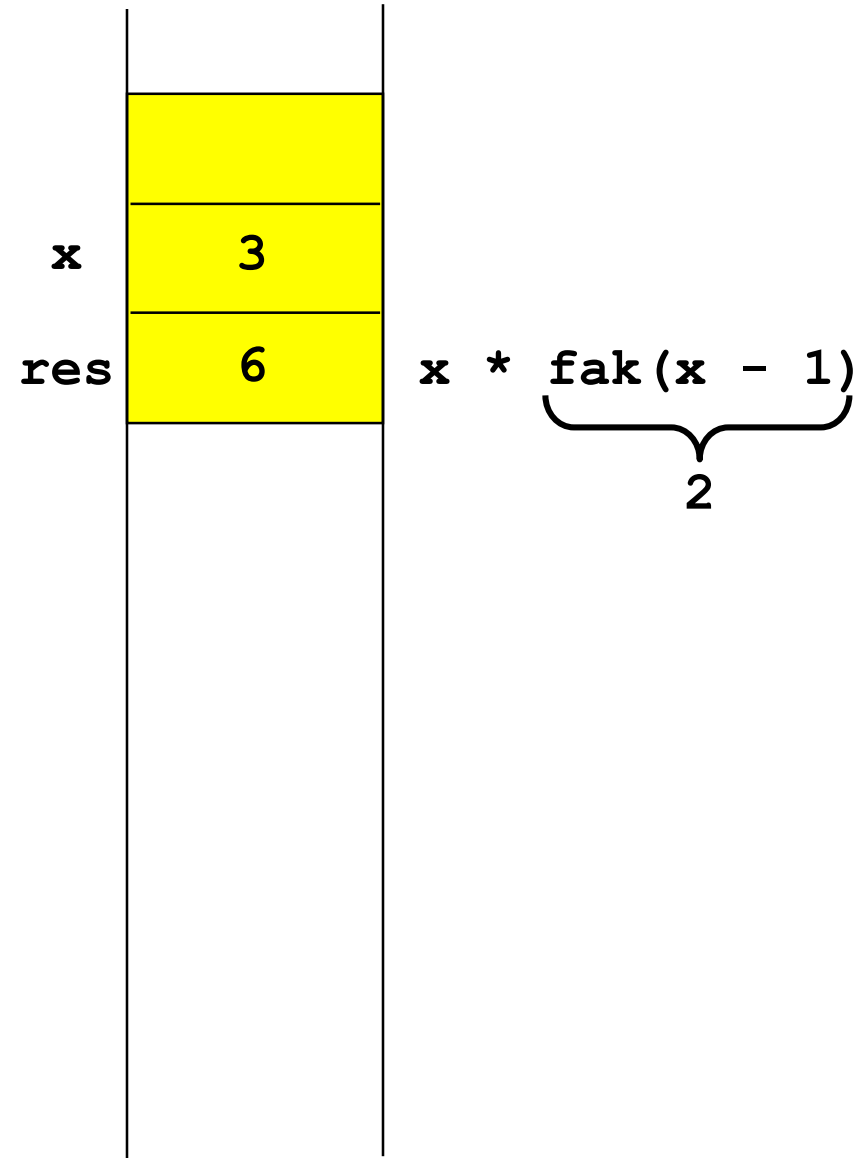
Aufruf: **fak (3)**



Speicherorganisation bei Rekursion

```
public static int fak (int x) {  
    if (x > 1)  
        return x * fak (x - 1);  
    else return 1;  
}
```

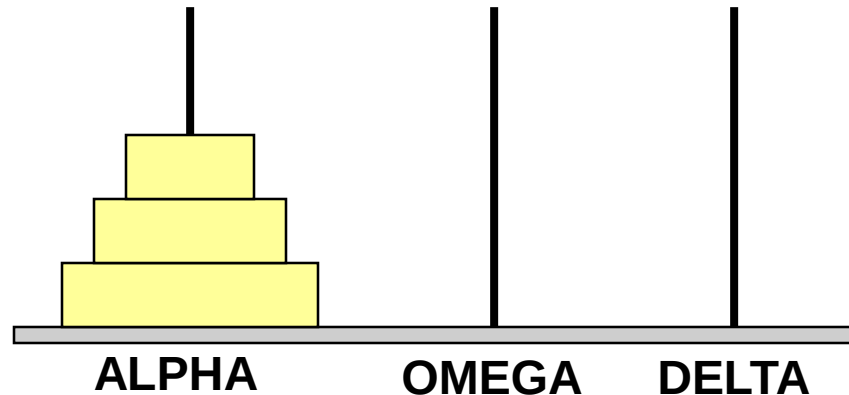
Aufruf: **fak (3)**



Türme von Hanoi

■ Aufgabe:

- bewege die Scheiben von ALPHA über DELTA nach OMEGA
- es darf immer **nur eine Scheibe** bewegt werden
- niemals darf eine Scheibe auf eine kleinere bewegt werden



■ Lösung (Divide & Conquer):

- allgemeine Lösung für einen Turm der Höhe h von ALPHA nach OMEGA
 - ◆ $h = 0$ gar nichts machen
 - ◆ $h > 0$
 1. Turm der Höhe $h-1$ von ALPHA über OMEGA nach DELTA
 2. (Unterste) Scheibe von ALPHA nach OMEGA legen
 3. Turm der Höhe $h-1$ von DELTA über ALPHA nach OMEGA

Türme von Hanoi

```
public class Hanoi {
```

*1 = oberste Scheibe
h = unterste Scheibe*

```
    private static void bewegeTurm (int hoehe,  
                                    String von, String ueber, String nach) {  
        if (hoehe > 0) { bewegeTurm (hoehe-1, von, nach, ueber);  
                        druckeZug  (hoehe, von, nach);  
                        bewegeTurm (hoehe-1, ueber, von, nach);  
        }  
    }
```

```
    private static void druckeZug (int hoehe, String von, String nach) {  
        System.out.println ("Scheibe " + hoehe + " von " + von + " nach " + nach);  
    }
```

```
    public static void main (String [] args) {  
        bewegeTurm(Integer.parseInt(args[0]), "ALPHA", "DELTA", "OMEGA");  
    }  
}
```

Türme von Hanoi

3 ALPHA DELTA OMEGA

① 2 ALPHA OMEGA DELTA

① 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

② **Scheibe 2 von ALPHA nach DELTA**

③ 1 OMEGA ALPHA DELTA

① 0 OMEGA DELTA ALPHA

② **Scheibe 1 von OMEGA nach DELTA**

③ 0 ALPHA OMEGA DELTA

② **Scheibe 3 von ALPHA nach OMEGA**

③ 2 DELTA ALPHA OMEGA

① 1 DELTA OMEGA ALPHA

① 0 DELTA ALPHA OMEGA

② **Scheibe 1 von DELTA nach ALPHA**

③ 0 OMEGA DELTA ALPHA

② **Scheibe 2 von DELTA nach OMEGA**

③ 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

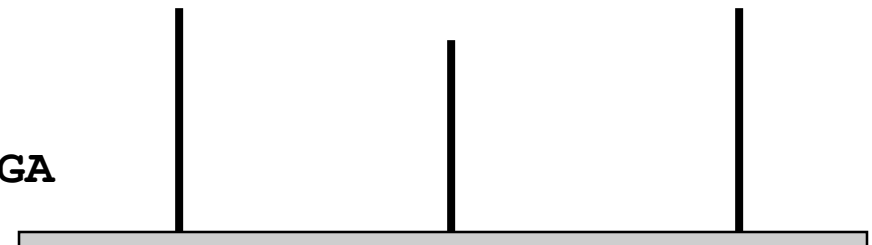
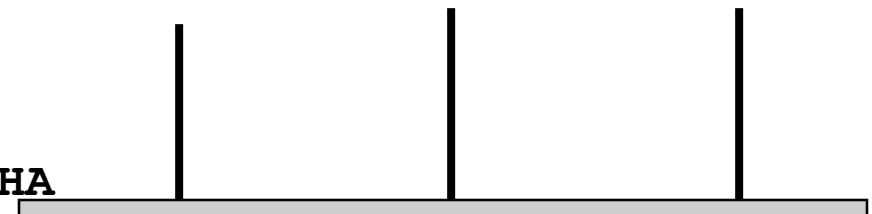
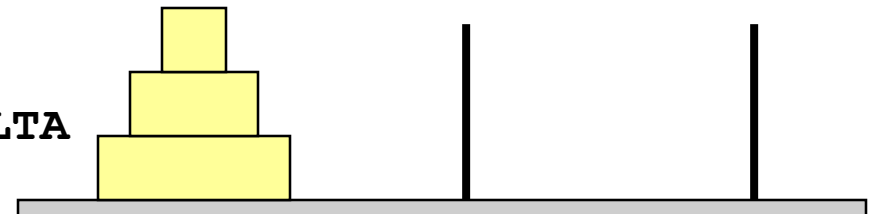
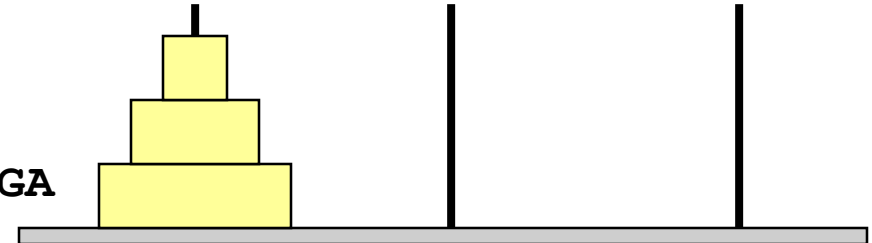
② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

ALPHA

OMEGA

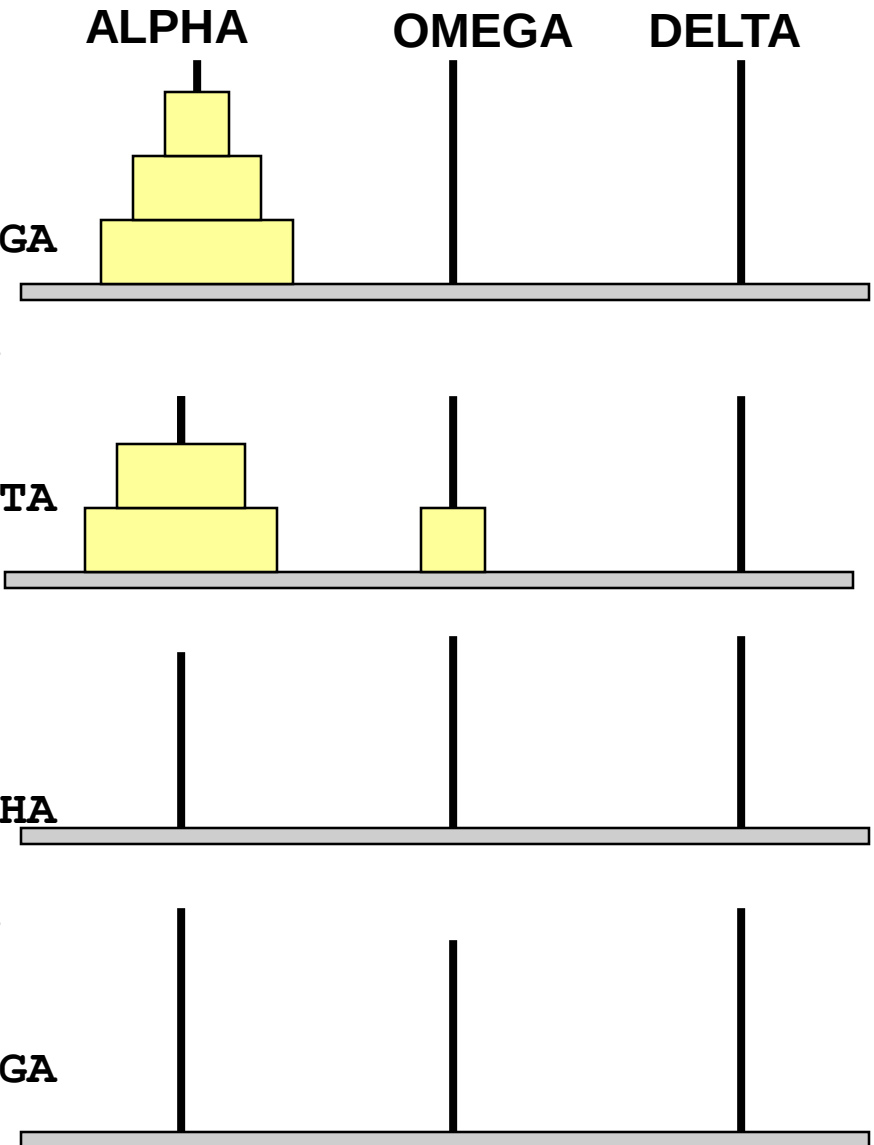
DELTA



Türme von Hanoi

3 ALPHA DELTA OMEGA

- ① 2 ALPHA OMEGA DELTA
 - ① 1 ALPHA DELTA OMEGA
 - ① 0 ALPHA OMEGA DELTA
 - ② **Scheibe 1 von ALPHA nach OMEGA**
 - ③ 0 DELTA ALPHA OMEGA
 - ② **Scheibe 2 von ALPHA nach DELTA**
 - ③ 1 OMEGA ALPHA DELTA
 - ① 0 OMEGA DELTA ALPHA
 - ② **Scheibe 1 von OMEGA nach DELTA**
 - ③ 0 ALPHA OMEGA DELTA
- ② **Scheibe 3 von ALPHA nach OMEGA**
- ③ 2 DELTA ALPHA OMEGA
 - ① 1 DELTA OMEGA ALPHA
 - ① 0 DELTA ALPHA OMEGA
 - ② **Scheibe 1 von DELTA nach ALPHA**
 - ③ 0 OMEGA DELTA ALPHA
 - ② **Scheibe 2 von DELTA nach OMEGA**
 - ③ 1 ALPHA DELTA OMEGA
 - ① 0 ALPHA OMEGA DELTA
 - ② **Scheibe 1 von ALPHA nach OMEGA**
 - ③ 0 DELTA ALPHA OMEGA



Türme von Hanoi

3 ALPHA DELTA OMEGA

① 2 ALPHA OMEGA DELTA

① 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

② **Scheibe 2 von ALPHA nach DELTA**

③ 1 OMEGA ALPHA DELTA

① 0 OMEGA DELTA ALPHA

② **Scheibe 1 von OMEGA nach DELTA**

③ 0 ALPHA OMEGA DELTA

② **Scheibe 3 von ALPHA nach OMEGA**

③ 2 DELTA ALPHA OMEGA

① 1 DELTA OMEGA ALPHA

① 0 DELTA ALPHA OMEGA

② **Scheibe 1 von DELTA nach ALPHA**

③ 0 OMEGA DELTA ALPHA

② **Scheibe 2 von DELTA nach OMEGA**

③ 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

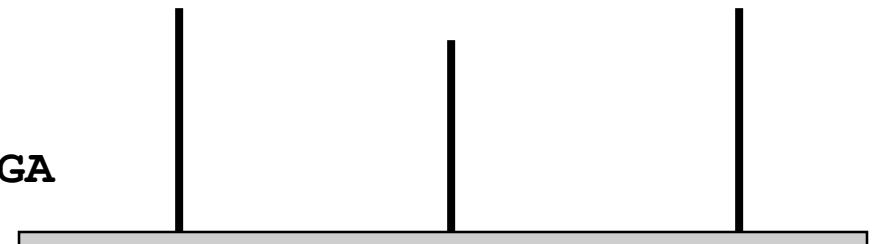
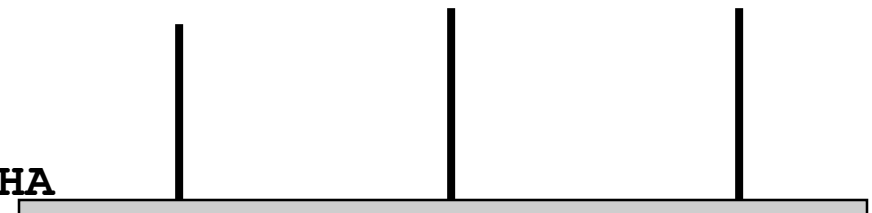
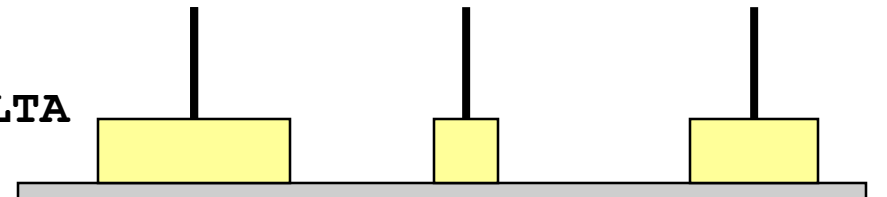
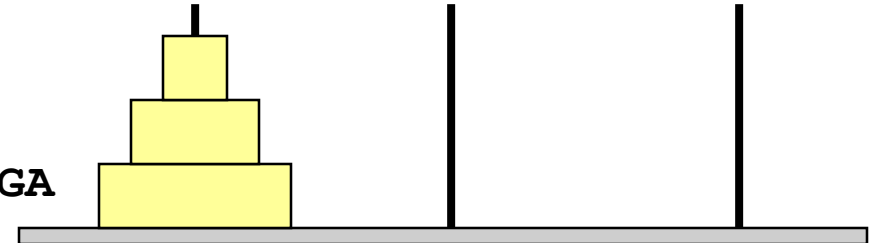
② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

ALPHA

OMEGA

DELTA



Türme von Hanoi

3 ALPHA DELTA OMEGA

① 2 ALPHA OMEGA DELTA

① 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

② **Scheibe 2 von ALPHA nach DELTA**

③ 1 OMEGA ALPHA DELTA

① 0 OMEGA DELTA ALPHA

② **Scheibe 1 von OMEGA nach DELTA**

③ 0 ALPHA OMEGA DELTA

② **Scheibe 3 von ALPHA nach OMEGA**

③ 2 DELTA ALPHA OMEGA

① 1 DELTA OMEGA ALPHA

① 0 DELTA ALPHA OMEGA

② **Scheibe 1 von DELTA nach ALPHA**

③ 0 OMEGA DELTA ALPHA

② **Scheibe 2 von DELTA nach OMEGA**

③ 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

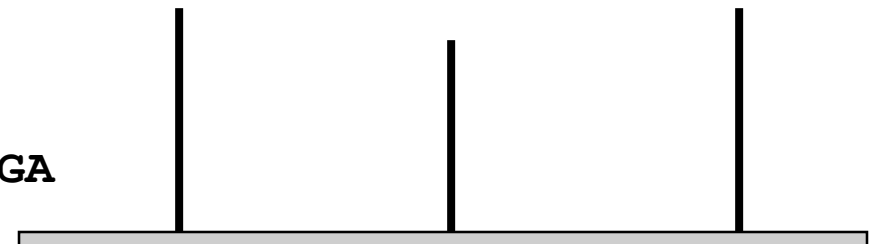
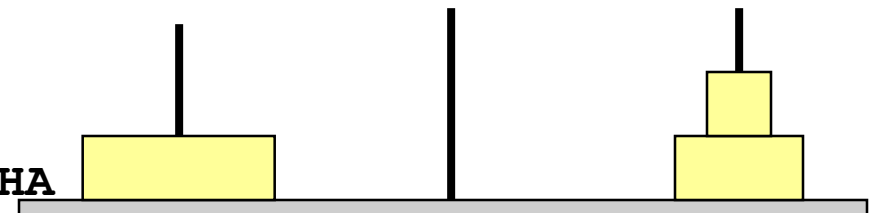
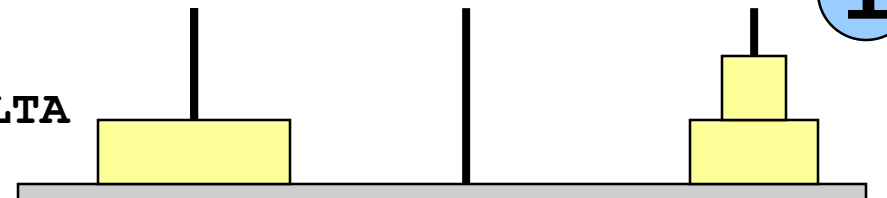
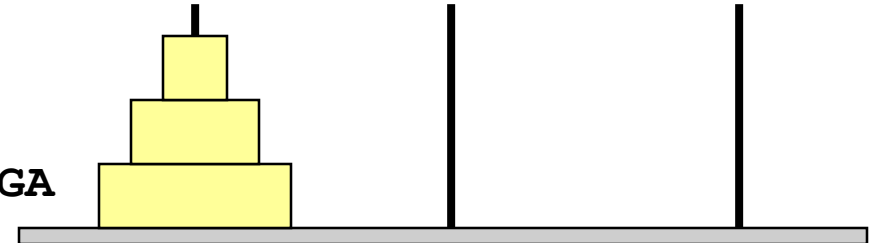
② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

ALPHA

OMEGA

DELTA



Türme von Hanoi

3 ALPHA DELTA OMEGA

① 2 ALPHA OMEGA DELTA

① 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

② **Scheibe 2 von ALPHA nach DELTA**

③ 1 OMEGA ALPHA DELTA

① 0 OMEGA DELTA ALPHA

② **Scheibe 1 von OMEGA nach DELTA**

③ 0 ALPHA OMEGA DELTA

② **Scheibe 3 von ALPHA nach OMEGA**

③ 2 DELTA ALPHA OMEGA

① 1 DELTA OMEGA ALPHA

① 0 DELTA ALPHA OMEGA

② **Scheibe 1 von DELTA nach ALPHA**

③ 0 OMEGA DELTA ALPHA

② **Scheibe 2 von DELTA nach OMEGA**

③ 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

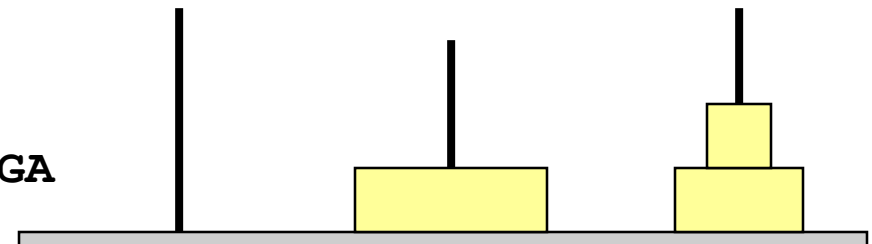
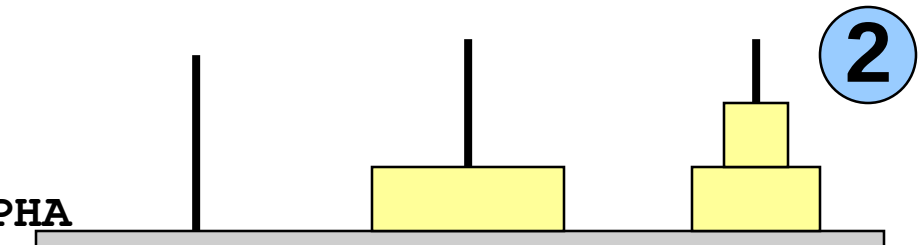
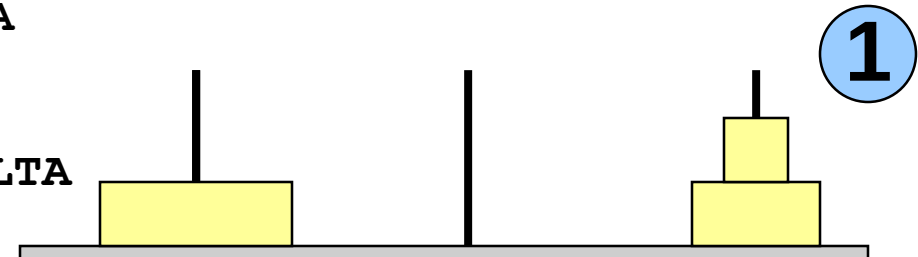
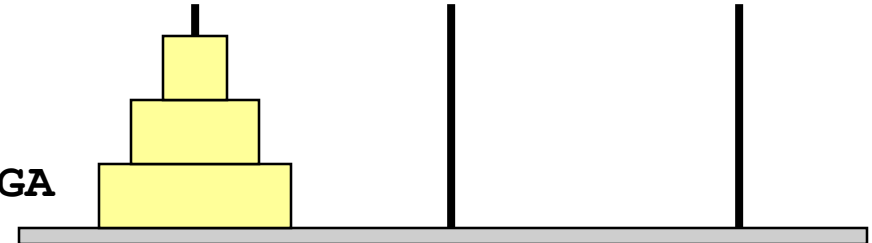
② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

ALPHA

OMEGA

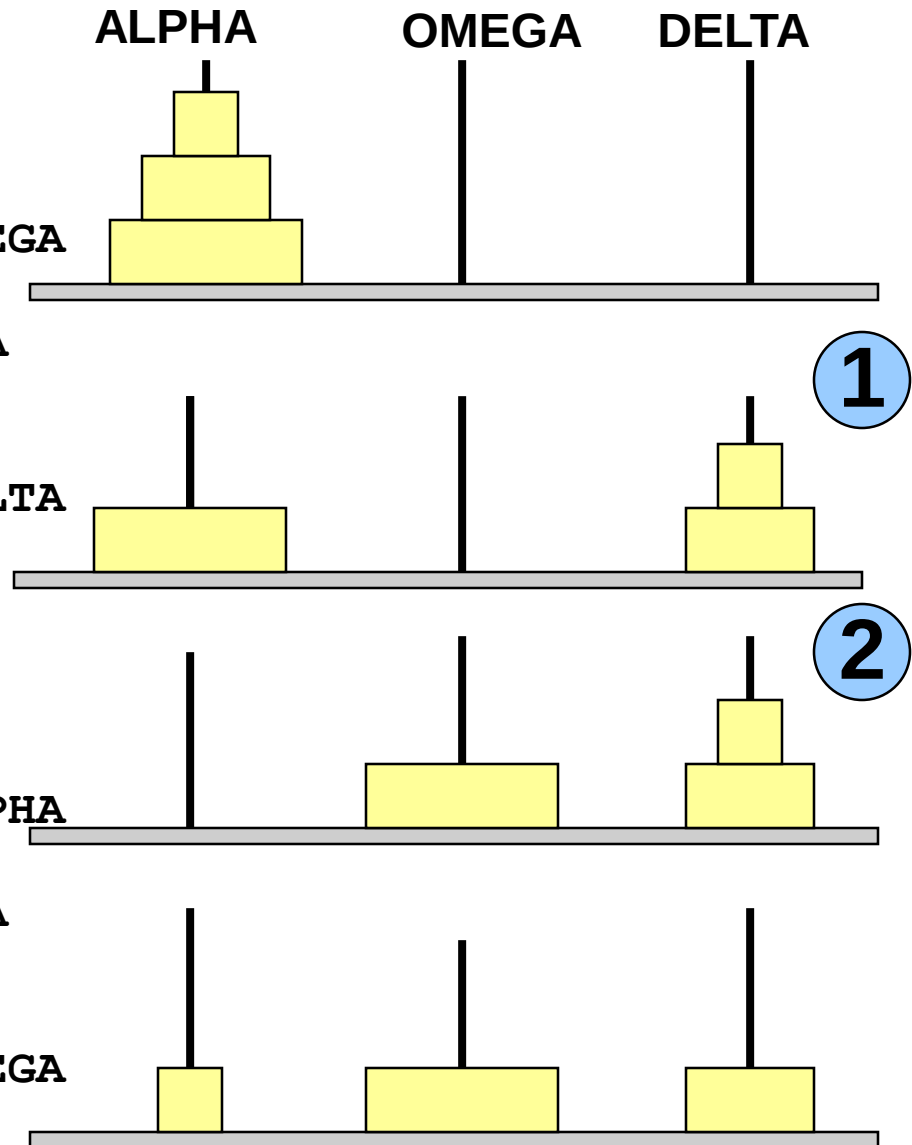
DELTA



Türme von Hanoi

3 ALPHA DELTA OMEGA

- ① 2 ALPHA OMEGA DELTA
 - ① 1 ALPHA DELTA OMEGA
 - ① 0 ALPHA OMEGA DELTA
 - ② **Scheibe 1 von ALPHA nach OMEGA**
 - ③ 0 DELTA ALPHA OMEGA
 - ② **Scheibe 2 von ALPHA nach DELTA**
 - ③ 1 OMEGA ALPHA DELTA
 - ① 0 OMEGA DELTA ALPHA
 - ② **Scheibe 1 von OMEGA nach DELTA**
 - ③ 0 ALPHA OMEGA DELTA
- ② **Scheibe 3 von ALPHA nach OMEGA**
- ③ 2 DELTA ALPHA OMEGA
 - ① 1 DELTA OMEGA ALPHA
 - ① 0 DELTA ALPHA OMEGA
 - ② **Scheibe 1 von DELTA nach ALPHA**
 - ③ 0 OMEGA DELTA ALPHA
 - ② **Scheibe 2 von DELTA nach OMEGA**
 - ③ 1 ALPHA DELTA OMEGA
 - ① 0 ALPHA OMEGA DELTA
 - ② **Scheibe 1 von ALPHA nach OMEGA**
 - ③ 0 DELTA ALPHA OMEGA



Türme von Hanoi

3 ALPHA DELTA OMEGA

① 2 ALPHA OMEGA DELTA

① 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

② **Scheibe 2 von ALPHA nach DELTA**

③ 1 OMEGA ALPHA DELTA

① 0 OMEGA DELTA ALPHA

② **Scheibe 1 von OMEGA nach DELTA**

③ 0 ALPHA OMEGA DELTA

② **Scheibe 3 von ALPHA nach OMEGA**

③ 2 DELTA ALPHA OMEGA

① 1 DELTA OMEGA ALPHA

① 0 DELTA ALPHA OMEGA

② **Scheibe 1 von DELTA nach ALPHA**

③ 0 OMEGA DELTA ALPHA

② **Scheibe 2 von DELTA nach OMEGA**

③ 1 ALPHA DELTA OMEGA

① 0 ALPHA OMEGA DELTA

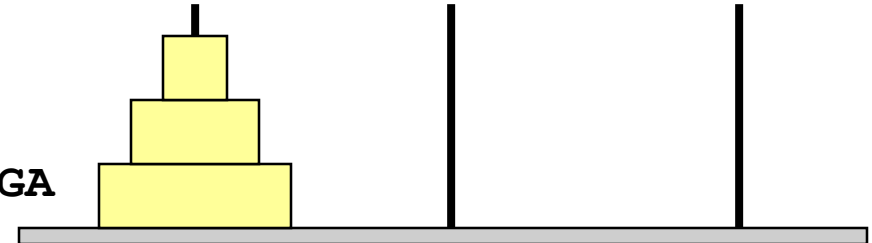
② **Scheibe 1 von ALPHA nach OMEGA**

③ 0 DELTA ALPHA OMEGA

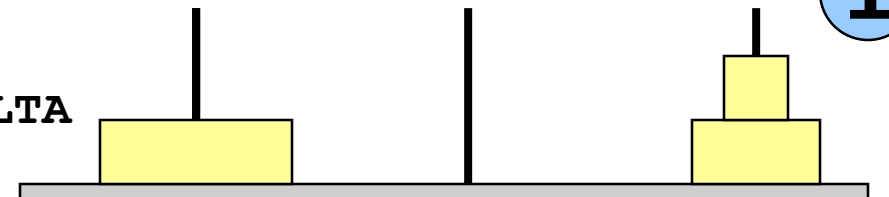
ALPHA

OMEGA

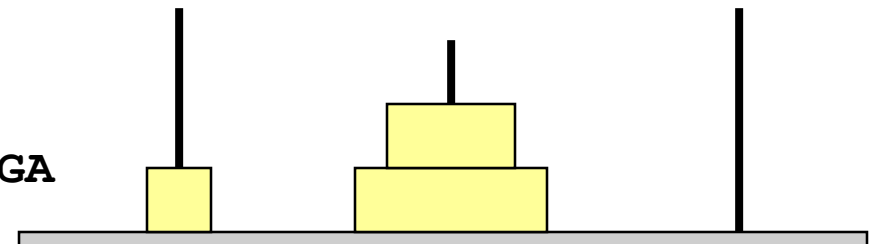
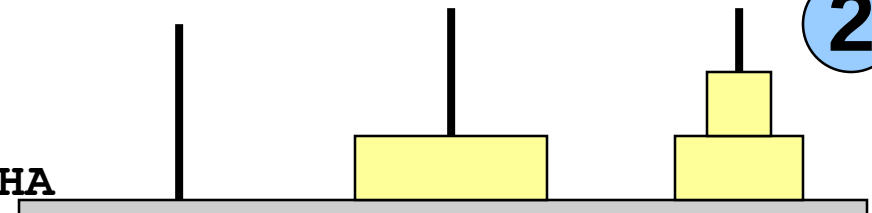
DELTA



1



2

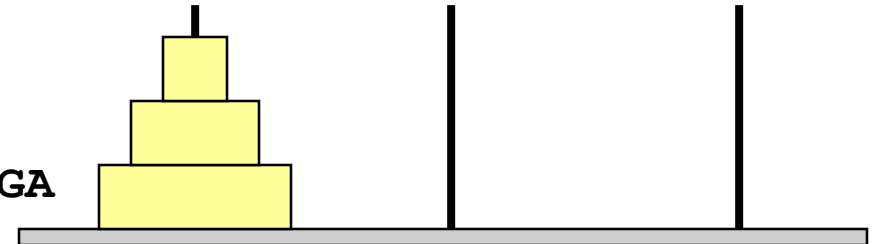


Türme von Hanoi

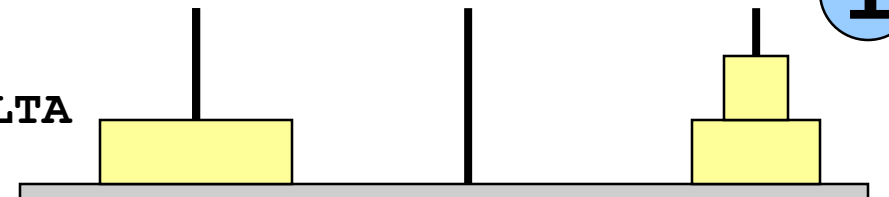
3 ALPHA DELTA OMEGA

- ① 2 ALPHA OMEGA DELTA
 - ① 1 ALPHA DELTA OMEGA
 - ① 0 ALPHA OMEGA DELTA
 - ② **Scheibe 1 von ALPHA nach OMEGA**
 - ③ 0 DELTA ALPHA OMEGA
 - ② **Scheibe 2 von ALPHA nach DELTA**
 - ③ 1 OMEGA ALPHA DELTA
 - ① 0 OMEGA DELTA ALPHA
 - ② **Scheibe 1 von OMEGA nach DELTA**
 - ③ 0 ALPHA OMEGA DELTA
- ② **Scheibe 3 von ALPHA nach OMEGA**
- ③ 2 DELTA ALPHA OMEGA
 - ① 1 DELTA OMEGA ALPHA
 - ① 0 DELTA ALPHA OMEGA
 - ② **Scheibe 1 von DELTA nach ALPHA**
 - ③ 0 OMEGA DELTA ALPHA
 - ② **Scheibe 2 von DELTA nach OMEGA**
 - ③ 1 ALPHA DELTA OMEGA
 - ① 0 ALPHA OMEGA DELTA
 - ② **Scheibe 1 von ALPHA nach OMEGA**
 - ③ 0 DELTA ALPHA OMEGA

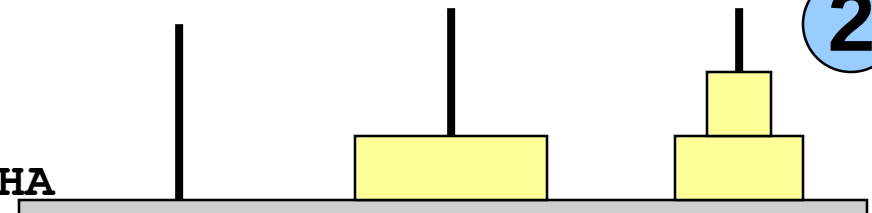
ALPHA OMEGA DELTA



1



2



3

