5 Rekursion

Lernziele

- Was ist Rekursion?
- Beispiele
- Lohnt sich Rekursion?

Einführung

Was ist Rekursion?

- Lateinisch: recurrere (zurücklaufen)
- Selbstdefinition einer Funktion, eines Verfahrens oder einer Datenstruktur
- Oft sehr elegant und leicht verständlich
- Paradebeispiele: Mathematische Funktionen
- Weitere Beispiele: Suche, Traversierung von Bäumen, Spiele, ...
- Aber: Rekursion kann oft vermieden werden

Einfache Rekursion: Beispiel Fakultät

Definition

$$n! = \begin{cases} 1 & f\text{ür } n = 0 \\ n \cdot (n-1)! & f\text{ür } n > 0 \end{cases}$$

Beobachtung

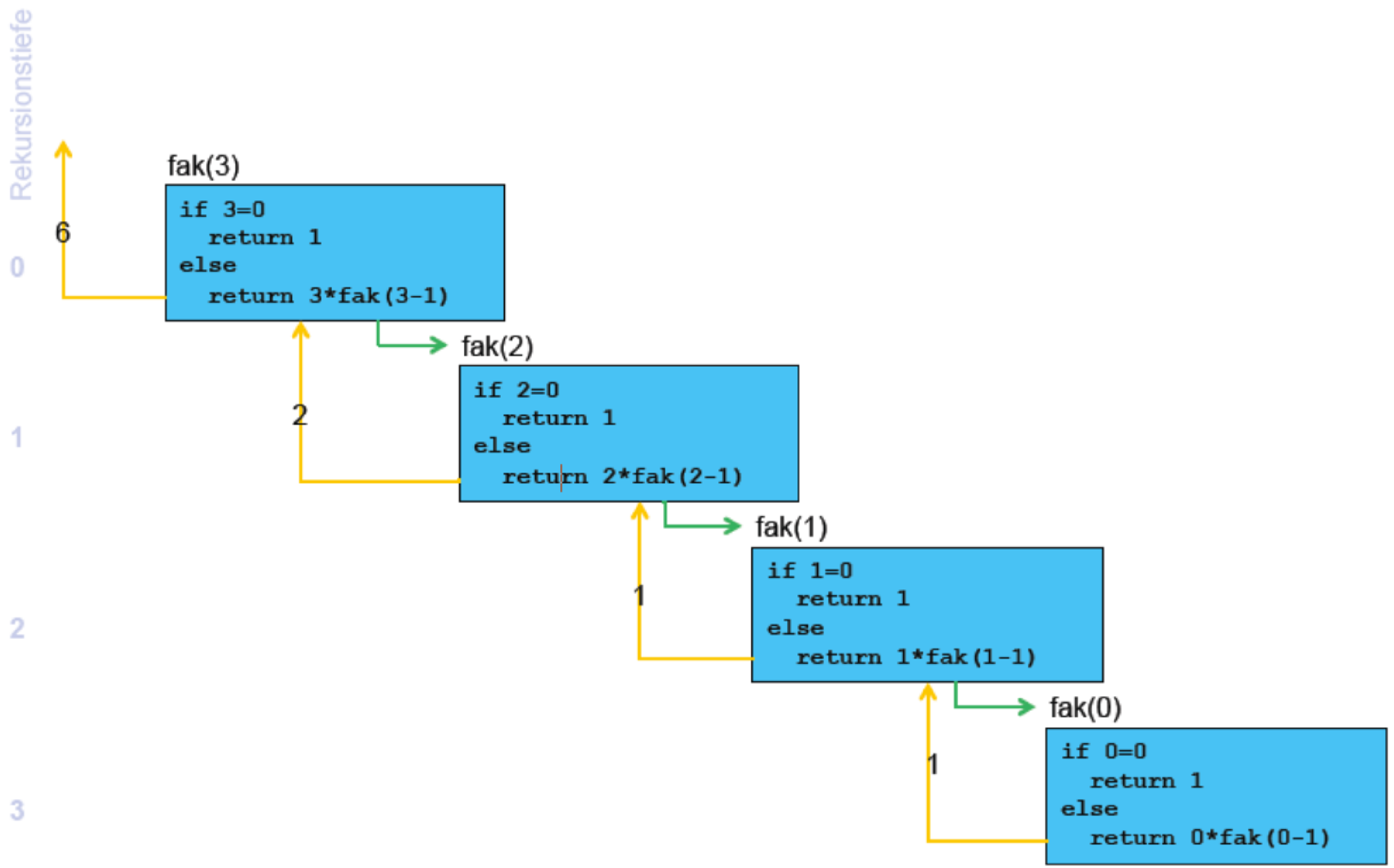
- Rekursive Definition ist an Bedingung geknüpft
- Bedingung stellt Ende sicher
- Rekursion erfolgt direkt

Einführendes Beispiel: Fakultät

Algorithmus

```
function FAKULTAET(n)
    if n = 0 then
        return 1
    else
        return n * FAKULTAET(n-1)
    end if
end function
```

Einführendes Beispiel: Fakultät



Einführendes Beispiel: Fakultät

Definition (iterativ)

- Rekursion kann (hier) vermieden werden
- Fakultät kann auch iterativ definiert werden:

$$n! = \begin{cases} 1 & für\ n = 0 \\ \prod_{i=1}^{n} i & für\ n > 0 \end{cases}$$

Einführendes Beispiel: Fakultät

Algorithmus (iterativ)

```
function FAKULTAET(n)
    if n = 0 then
        return 1
    end if
    f = 1
    for i = 2 to n do
        f = f * i
    end for
    return f
end function
```

Direkte und indirekte Rekursion

Direkte Rekursion

```
function F
    if Rekursionsbedingung then
        F
    end if
    ...
end function
```

Indirekte Rekursion

```
function F
    if Rekursionsbedingung then
        G
    end if
    ...
end function

function G
  if Rekursionsbedingung then
        F
    end if
    ...
end function
```

Beispiel: Binäre Suche

Eingabe: Sortiertes Feld A, Grenzen l, r, und gesuchte Zahl x

Ausgabe: Position der Zahl x in A oder NIL

Grundidee

- Betrachte die Mitte
- Falls gewünschte Zahl nicht gefunden: Suche in der linken oder rechten Hälfte weiter
- Rekursiv leicht lösbar

Beispiel: Binäre Suche

```
function BINAERESUCHE(A, l, r, x)
    m = ⌊(l + r) / 2⌋
    if A[m] == x then
        return m
    end if
    if l >= r then
        return NIL
    end if
    if x > A[m] then
        return BINAERESUCHE(A, m+1, r, x)
    else
        return BINAERESUCHE(A, l, m-1, x)
    end if
end function
```

Hochschule Aalen

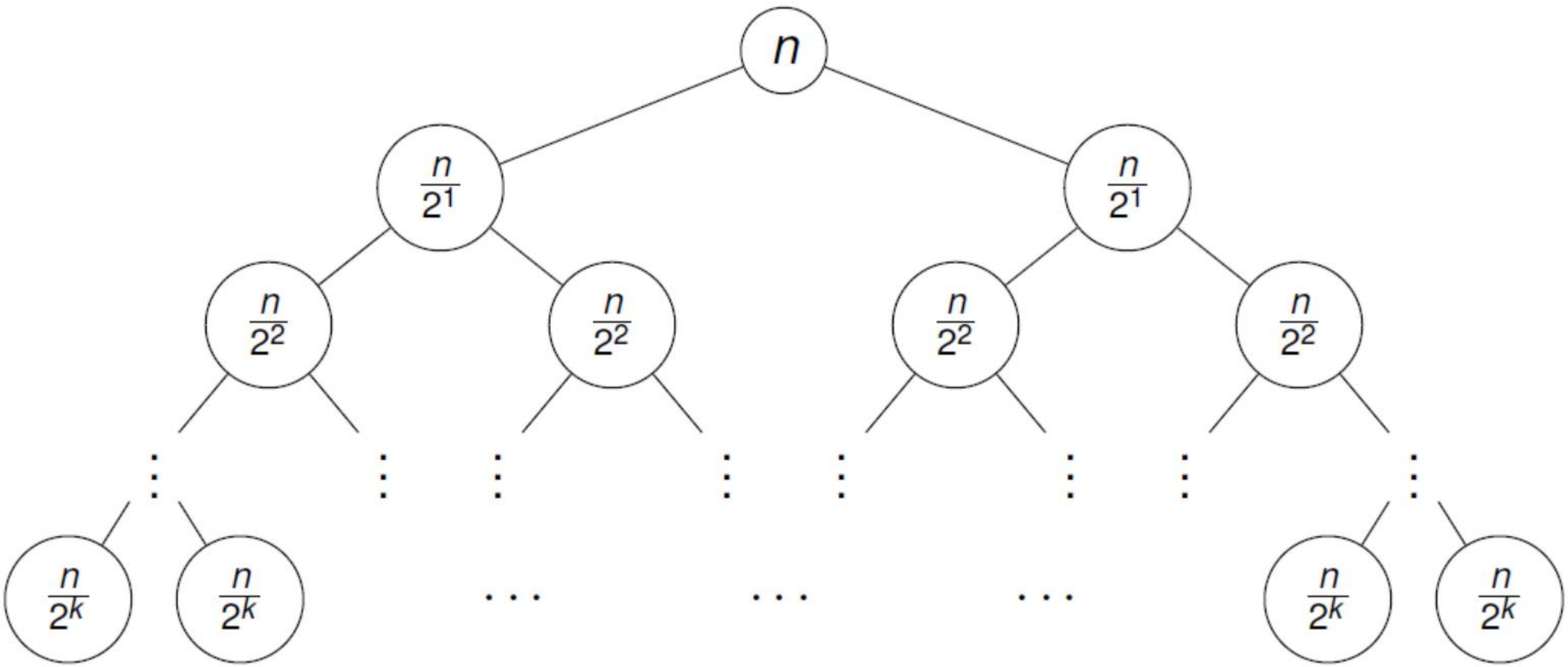
Laufzeit

- Wie kann die Laufzeit abgeschätzt werden?

  Beobachtung

  - Lösung trivial, wenn x in der Mitte steht ($O(1)$ oder sogar $\Theta(1)$)
  - Sonst: Zerlegung in 2 Teilprobleme halber größe
  - Nur ein Teilproblem (= 1 Seite) wird rekursiv weiter behandelt
  - Jeder Vergleich reduziert die zu durchsuchende Restmenge n_i um den Faktor 2
  - Rekursionsgleichung für Laufzeit: $T(n) = T(n/2) + \Theta(1)$
- Allgemein kann Laufzeit abgeschätzt werden durch
  - Substitutionsmethode
  - Analyse des Rekursionsbaums
  - Mastertheorem

Analyse des Rekursionsbaums



Mastertheorem

- Seien a und b Konstanten mit $a \geq 1$ und $b > 1$, $n \in$ N und $f(n)$ eine von $T(n)$ unabhängige Funktion, dann ist $T(n)$ durch die Rekursionsgleichung

$$T(n) = a \cdot T(\frac{n}{b}) + f(n)$$

definiert.

- Interpretation:

a	Anzahl der Unterprobleme
n/b	Teil des Originalproblems, beinhaltet wiederum Unterprobleme
f(n)	Aufwand (Nebenkosten) der durch Aufteilung des Problems und Kombination der Teillösungen entsteht.

Mastertheorem

- Interpretation (für Binäre Suche):

a	Anzahl der Unterprobleme = 1 (nur eine Seite des aufgeteilten Gesamtproblems wird weiter betrachtet)
n/b	Teil des Originalproblems, beinhaltet wiederum Unterprobleme = n/2 (Originalproblem waren n zu vergleichende Elemente, nach Teilung noch max. n/2)
f(n)	Aufwand (Nebenkosten) der durch Aufteilung des Problems und Kombination der Teillösungen entsteht. = 1 bzw $\Theta(1)$ (in diesem Beispiel Abschätzung des Aufwands)

Mastertheorem: Definition

- Seien a und b Konstanten mit $a \geq 1$ und $b > 1$, $n \in \mathbb{N}$ und $f(n)$ eine von $T(n)$ unabhängige Funktion, dann ist $T(n)$ durch die Rekursionsgleichung

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

definiert.

- Dann unterscheidet das Mastertheorem für T(n) in Abhängigkeit von n drei Fälle.

Mastertheorem: Fall 1

Falls gilt

$$f(n) = O(n^{log_b a - \epsilon})\ mit\ \epsilon > 0$$

Dann folgt

$$T(n) = \ \Theta(n^{log_b a})$$

Bedeutung

Der notwendige Aufwand (Nebenkosten) ist gegenüber der Berechnung der Unterprobleme vernachlässigbar. $(a \ \cdot T(\frac{n}{b}) \ \gg f(n))$

Mastertheorem: Fall 2

Falls gilt

$$f(n) = \Theta(n^{log_b a})$$

Dann folgt

$$T(n) = \Theta(n^{log_b a} \cdot \log(n))$$

Bedeutung

Der notwendige Aufwand (Nebenkosten) ist vergleichbar mit der Berechnung der Unterprobleme. $(a \cdot T(\frac{n}{b}) \approx f(n))$

Mastertheorem: Fall 3

Falls gilt

$$f(n) = \Omega\left(n^{log_b a+\epsilon}\right) f\text{ür } ein\ \epsilon > 0$$

$$a \cdot f\left(\frac{n}{b}\right) \leq c\ \cdot f(n) f\text{ür } ein\ 0 < c < 1\ und\ hinreichend\ große\ n$$

Dann folgt

$$T(n) =\ \Theta(f(n))$$

Bedeutung

Die Berechnung der Unterprobleme ist gegenüber des Aufwands (Nebenkosten) vernachlässigbar. $(a\ \cdot T(\frac{n}{b})\ \ll f(n))$

Mastertheorem: Definition

- Seien *a* und *b* Konstanten mit *a* ≥ 1 und *b* > 1, *n* ∈ N und *f* (*n*) eine von *T*(*n*) unabhängige Funktion, dann ist *T*(*n*) durch die Rekursionsgleichung

  $$T(n) = a \cdot T\frac{n}{b} + f(n)$$

  definiert.

- Dann unterscheidet das Mastertheorem für T(n) in Abhängigkeit von n drei Fälle.

- $$T(n) = \begin{cases} \Theta\left(n^{log_b a}\right) & wenn\ f(n) = O\left(n^{log_b a - \epsilon}\right) mit\ \epsilon > 0 \\ \Theta(n^{log_b a} \cdot \log(n)) & wenn\ f(n) = \Theta\left(n^{log_b a}\right) \\ \Theta(f(n)) & wenn\ f(n) = \Omega\left(n^{log_b a + \epsilon}\right) mit\ \epsilon > 0\ und \\ & a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) f\text{ür}\ ein\ 0 < c < 1\ und\ große\ n \end{cases}$$

Mastertheorem: Beispiele

Fall 1

Sei $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n$

$\Longrightarrow a = 9, b = 3, f(n) = n$

$n^{log_b a} = n^{log_3 9} = n^2$

Es gilt $f(n) = n \in O\left(n^{2-1}\right) = O(n) f\text{ür}\ \epsilon = 1$

$\Longrightarrow$ ist Fall 1 des Mastertheorems

Folglich ist $T(n) \in \Theta(n^2)$

Mastertheorem: Beispiele

Fall 2

Sei $$T(n) = T\left(\frac{2n}{3}\right) + 1$$

$$\Longrightarrow a = 1, b = \frac{3}{2},\ f(n) = 1$$

$$n^{log_b a} = n^{log_{3/2} 1} = n^0 = 1$$

Es gilt $f(n) = 1\ \in \Theta(1)$

$\Longrightarrow$ ist Fall 2 des Mastertheorems

Folglich ist $T(n) \in \Theta(\log(n))$

Mastertheorem: Beispiele

Fall 3

Sei $$T(n) = 3 \cdot T\left(\frac{n}{4}\right) + n\, logn$$

$$\Longrightarrow a = 3, b = 4,\ f(n) = n\, logn$$

$$n^{log_b a} = n^{log_4 3} = n^{0{,}793}$$

Es gilt $$f(n) = n\, logn\ \in \Omega(n)\ f\text{ür}\ \epsilon \approx 0{,}2$$

$\Longrightarrow$ ist Fall 3, falls gilt $$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

$$3\left(\frac{n}{4} log\, \frac{n}{4}\right) \leq c\, n\, logn$$

$$\frac{3n}{4} log\, \frac{n}{4} \leq \left(\frac{3}{4}\right) n\, logn = c \cdot f(n) f\text{ür}\ c = \frac{3}{4}$$

Folglich ist $$T(n) \in \Theta(\log(n))$$

Mastertheorem: Beispiele

Fall 3

Sei $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n\ logn$

$$\Longrightarrow a = 2, b = 2, f(n) = n\ logn$$

$$n^{log_b a} = n^{log_2 2} = n$$

Es muss gelten $f(n) = n\ logn\ \in \Omega(n^{1+\epsilon})$

$$\frac{f(n)}{n^{1+\epsilon}} = \frac{n\ logn}{n\ \cdot\ n^{\epsilon}} = \frac{logn}{n^{\epsilon}}$$

Geht asymptotisch gegen 0, folglich ist Fall 3 nicht anwendbar

Man kann daraus schließen, dass das Mastertheorem nicht in allen Fällen zur Lösung der Rekurrenz benutzt werden kann.

Mastertheorem: Alternative Betrachtung

Annahme: $f(n) \in \Theta(n^d)$

Dann gelten:

Fall 1: $n^{log_b a - \epsilon} \Longrightarrow d < log_b a$

Fall 2: $n^{log_b a} \Longrightarrow d = log_b a$

Fall 3: $n^{log_b a + \epsilon} \Longrightarrow d > log_b a$

mit $\epsilon > 0, a \geq 0, b > 1 \; und \; d \geq 0$

Mastertheorem: Alternative Betrachtung

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta\left(n^d\right)$$

mit $\mathrm{a} \geq 0, \mathrm{b} > 1 \; und \; d \geq 0$

$$T(n) = \begin{cases} Fall\ 1\text{:}\ \Theta\left(n^{log_b a}\right) & wenn\ d < log_b a \\ Fall\ 2\text{:}\ \Theta\left(n^d logn\right) & wenn\ d = log_b a \\ Fall\ 3\text{:}\ \Theta\left(n^d\right) & wenn\ d > log_b a \end{cases}$$

Aber: In Abhängigkeit von T(n) nicht immer anwendbar!

Mastertheorem: Beispiel Binäre Suche

Laufzeit

- Zahl steht in der Mitte: Problem ist trivial gelöst in $\Theta(1)$
- Sonst: Zerlegung in zwei Teilprobleme halber Größe, eins wird weiter betrachtet

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \Theta(1)$$

$$\Longrightarrow a = 1, b = 2$$

$$n^{log_b a} = n^{log_2 1} = n^0 = 1 \in \Theta(1)$$

$\Longrightarrow$ Fall 2 des Mastertheorems

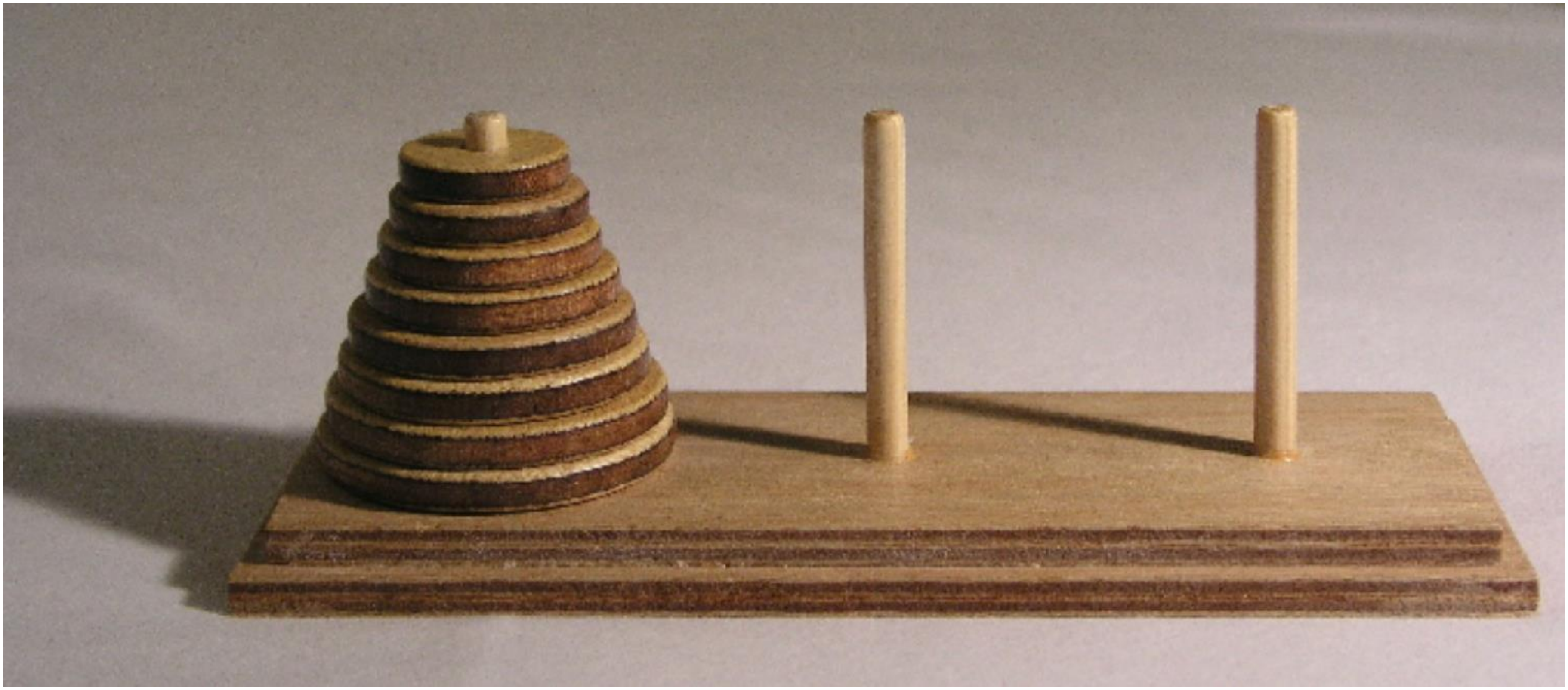
Folglich ist $T(n) \in \Theta(log n)$

Die Türme von Hanoi

- Vermutlich 1883 erfunden vom französischen Mathematiker Édouard Lucas
- Dachte sich eine Geschichte dazu aus:
  - Im großen Tempel zu Benares im Mittelpunkt der Welt, soll ein Turm aus 64 goldenen Scheiben versetzt werden. Wenn dies vollbracht ist, ist das Ende der Welt gekommen.
- Lösung der Mönche:
  - Der älteste Mönch bittet den zweitältesten Mönch, die 63 oberen Scheiben zu versetzen. Es selbst versetzt dann die untere Scheibe.
  - Der zweitälteste Mönch bittet den drittältesten Mönch die 62 oberen Scheiben zu versetzen. Er selbst versetzt dann die zweitunterste Scheibe.
  - . . .

- Da die 64 Mönche im Kloster viel Zeit haben, nehmen sie sich der Aufgabe an.

Die Türme von Hanoi



Quelle: Wikipedia

Die Türme von Hanoi

```
function HANOI(h, A, B, C)
    if h>0 then
        HANOI(h-1, A, C, B)
        Verschiebe Scheibe von A nach C
        HANOI(h-1, B, A, C)
    end if
end function
```

h: Scheibengröße

A: Ausgangsturm

B: Hilfsturm

C: Ziel

Die Türme von Hanoi

```c
#include <stdio.h>

void Hanoi ( int h, int A, int B, int C)
{
        if (h > 0)
        {
                Hanoi (h -1, A, C, B);
                printf (" Move disc of size %d: stack %d --> stack %d\r\n", h, A, C);
                Hanoi (h -1, B, A, C);
        }
}

int main ()
{
        Hanoi (3, 1, 2, 3);
        return 0;
}
```

Die Türme von Hanoi

Ausgabe

```
Move disc of size 1: stack 1 --> stack 3
Move disc of size 2: stack 1 --> stack 2
Move disc of size 1: stack 3 --> stack 2
Move disc of size 3: stack 1 --> stack 3
Move disc of size 1: stack 2 --> stack 1
Move disc of size 2: stack 2 --> stack 3
Move disc of size 1: stack 1 --> stack 3
```

Die Türme von Hanoi

Laufzeitbestimmung: Wann ist das Ende der Welt gekommen?

n	T(n)
1	1
2	1 + 1 + 1 = 3
3	3 + 1+ 3 = 7
4	7 + 1 + 7 = 15
⋮	⋮
n	T(n-1) + 1 + T(n-1)

Laufzeit: $T(n) = 2 \cdot T(n-1) + 1$

$= 2^n - 1$ (Mersenne-Zahl)

Wenn die Mönche pro Sekunde einen Stein bewegen, ist das Ende der Welt in 585 Milliarden Jahren gekommen

Die Türme von Hanoi

Substitutionsmethode

Aus der vorherigen Tabelle lässt sich vermuten, dass

$$T(n) = 2 \cdot T(n-1) + 1 = 2^n - 1 \in O(2^n)$$

Zu zeigen ist also, dass $T(n) \leq c \cdot 2^n$

$$\begin{aligned} T(n) = \quad 2 \cdot (2^{n-1} - 1) + 1 &\leq c \cdot 2^n \\ 2^n - 2 + 1 &\leq c \cdot 2^n \\ 2^n - 1 &\leq c \cdot 2^n \end{aligned}$$

Die Türme von Hanoi

Beweis mit vollständiger Induktion

Behauptung: $T(n) = 2 \cdot T(n-1) + 1 = 2^n - 1$

Induktionsanfang: $T(1) = 2^1 - 1 = 1$

Induktionsschritt: Falls die Behauptung für n gilt, dann gilt sie auch für n+1

$$T(n) = 2^n - 1$$

$$\begin{aligned} T(n+1) &= 2 \cdot T(n) + 1 \\ &= 2 \cdot (2^n - 1) + 1 \\ &= 2 \cdot 2^n - 2 + 1 \\ &= 2^{n+1} - 1 \end{aligned}$$