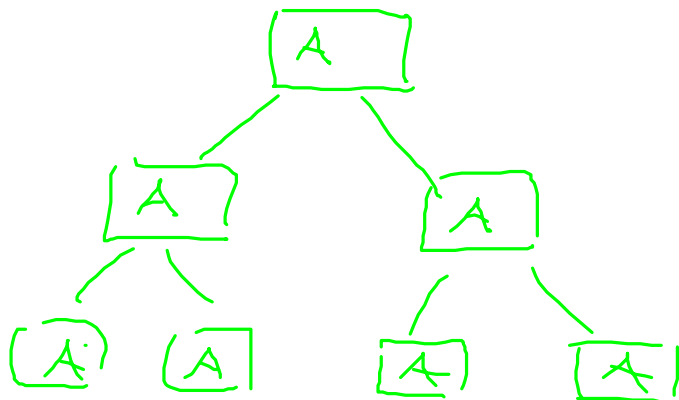


# Kap 8. Rekursion

Rekursion:

Methoden können sich selbst aufrufen



Rekursionstiefe

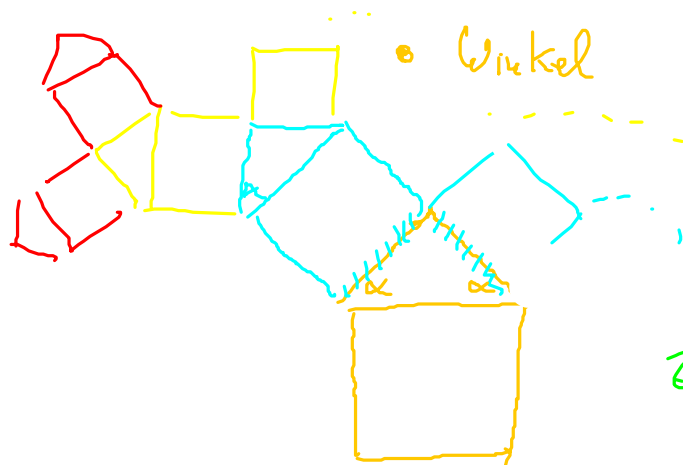
= Höhe (Rekursionsbaum) + 1

$\hat{=}$  Maß für Speicher

# Knoten  $\hat{=}$  Maß für Laufzeit

Bsp: Pythagoras - Baum

Methode: Input: • Strecke im  $\mathbb{R}^2$  (Seitenlänge Quadrat)



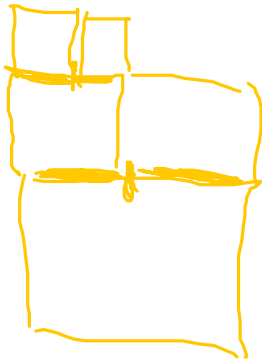
Zeichnung  $\hat{=}$  Rekursionsbaum umgedreht

Funktionsweise: Zeichnet Quadrat + Dreieck mit Winkel  $\alpha$

+ rekursive Aufrufe für die Seiten  
des Dreiecks, die nicht zum  
Quadrat gehören

nichts machen, falls Länge Strecke  
< 2mm ist

$$f_n(x) = 0$$

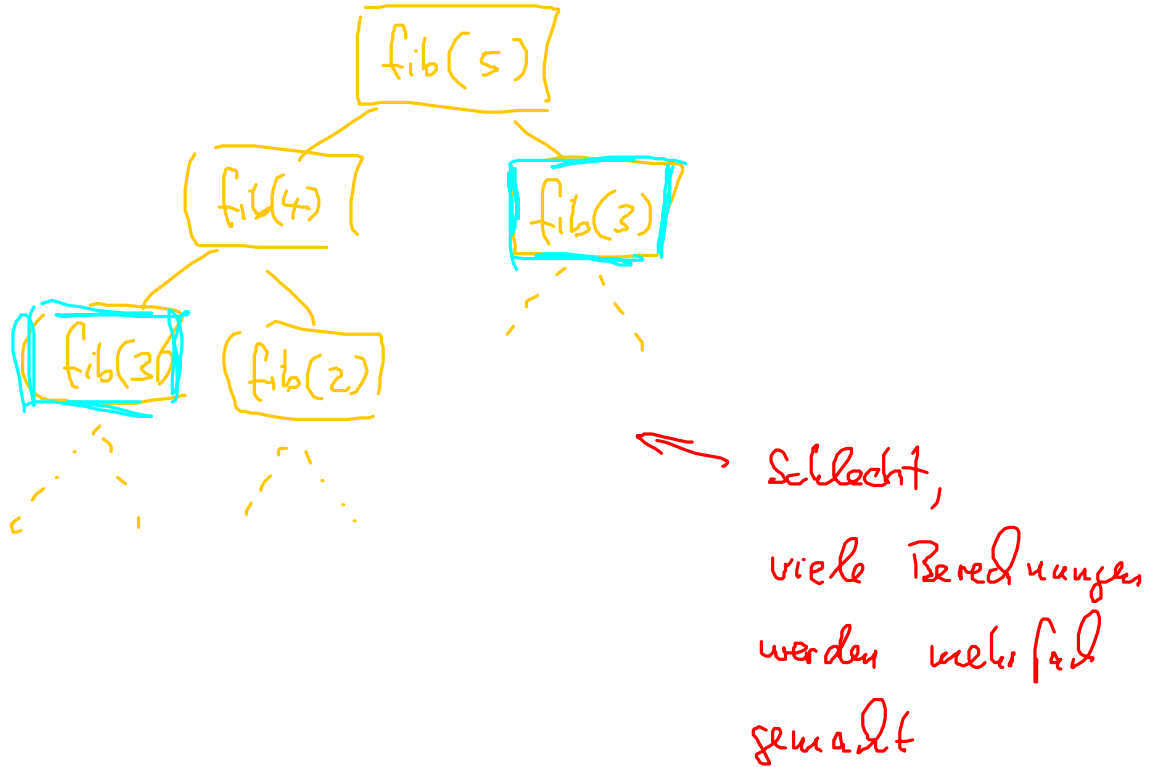


## Fibonacci Zahlen

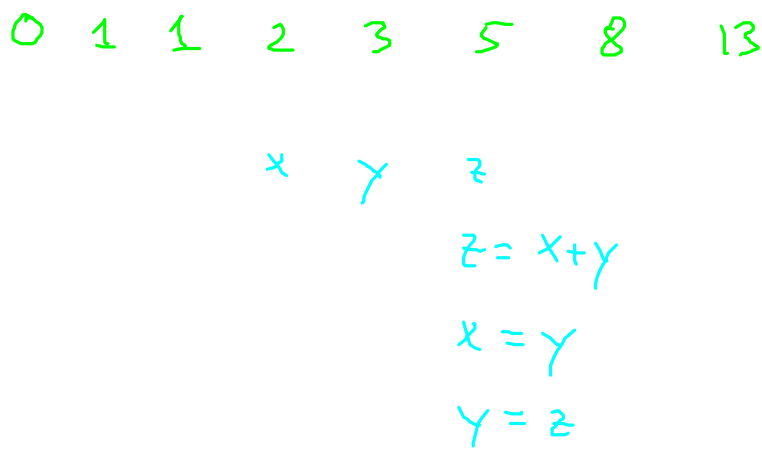
$$f_0 = 0 \quad f_1 = 1$$

$$f_{n+1} = f_n + f_{n-1}$$

0 1 1 2 3 5 8 ...



hier Iteration besser



## Kap 9 Die Analyse von Algorithmen

Ziel Effizienzuntersuchung von Algorithmen

↑  
unabhängig von Technologie



Worst case:  $n$  Vergleiche

- average case analysis  
mitteln über alle möglichen Inputs
- untere Schranken für Laufzeit

Bsp: Matrizenmultiplikation von  $n \times n$  Matrizen



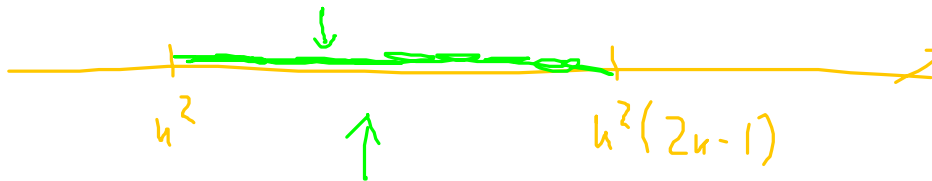
$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

$n$  Mult.,  $n-1$  Add.

$n^2$  Ergebnis-Elemente.

$n^2(2n-1)$

natürliche untere Schranke:  $n^2$  (da  $n^2$  Ergebniszahlen berechnet werden müssen)



Bereich für Laufzeit von Algos

- A posteriori Analyse

Laufzeitmessungen mit Implementations des Algo  
abh. von Prog. Sprache, Rechner, Computer

A priori Analyse interessiert sich nicht um "Konstanten"  
nur Größenordnungen interessieren

↓

wie messen wir Größenordnungen?

## 9.2 Die asymptotische Notation

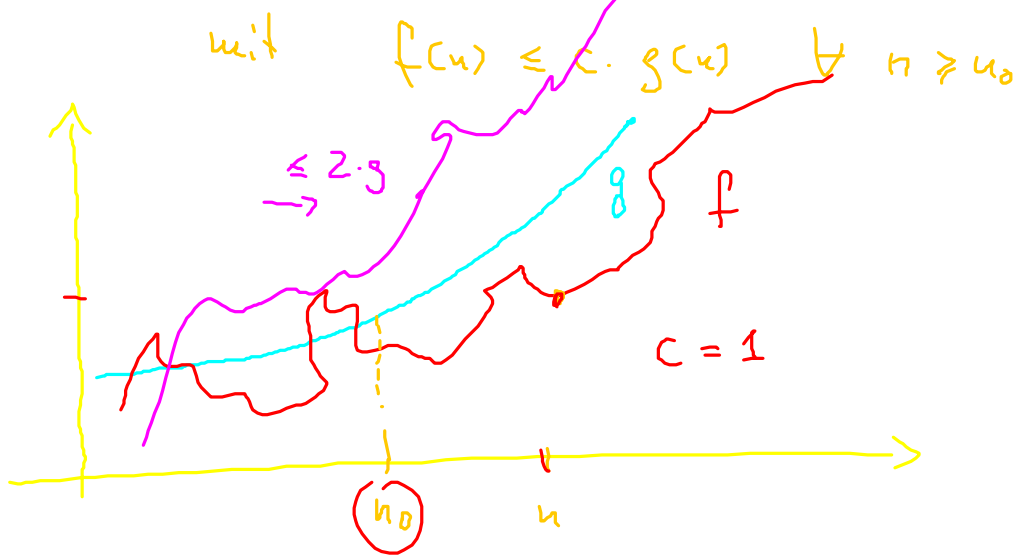
Betrachte Funktionen  $f: \mathbb{N} \rightarrow \mathbb{N}$

↑ ↑  
da Laufzeit  $f(n)$  von Algorithmen

Sei  $g: \mathbb{N} \rightarrow \mathbb{N}$

$$O(g) := \left\{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c \exists n_0 \in \mathbb{N} : \forall n \geq n_0 \right. \\ \left. f(n) \leq c \cdot g(n) \right\}$$

↑  
zu einem solchen  $f$  existiert Konstante  $c$ , Zahl  $n_0$   
( $c, n_0$  abh. von  $f$ )



Sprechweise  $f$  ist in  $O$  von  $g$ ,  $f$  ist von der Größenordnung  $g$

Schreibweise  $f(n) \in O(g(n)) \quad f(n) = O(g(n))$

SATZ: Sei  $f$  ein Polynom,  $f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$   
 $a_m \neq 0$

$$[\text{Bsp } f(n) = 3n^4 - 7n^3 + n - 17]$$

Dann gilt:  $f \in O(n^m)$

$$[\text{Bsp } f \in O(n^4)]$$

Beweis: Überprüfen Def von  $O(g)$

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$$

$$\leq |a_m| n^m + |a_{m-1}| n^{m-1} + \dots + |a_1| n + |a_0|$$

$$= \underbrace{\left( |a_m| + \frac{|a_{m-1}|}{n} + \dots + \frac{|a_1|}{n^{m-1}} + \frac{|a_0|}{n^m} \right)}_{n \geq n_0 := 1} n^m$$

$$\leq C := |a_m| + |a_{m-1}| + \dots + |a_0|$$

$$\Rightarrow f(n) \leq \underline{C} \cdot n^m \quad \text{für alle } n \geq \underline{n_0} = 1 \quad \Rightarrow f \in O(n^m)$$

$$\begin{array}{cc} 100 n^2 & \frac{1}{1000} n^3 \\ \downarrow & \downarrow \\ O(n^2) & O(n^3) \end{array}$$

für große  $n$  sind kleinere  
Größenordnungen zu bevorzugen

Wie kann man Größenordnungen unterscheiden?

Mit  $O$ -Notation

$$\text{Sei } g: \mathbb{N} \rightarrow \mathbb{N}$$

$$O(g) = \{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \forall c \exists n_0 \in \mathbb{N} : f(n) \leq c \cdot g(n) \forall n \geq n_0 \}$$

Sprechweise  $f$  ist kleiner als  $g$

$f$  ist eine kleinere Größenordnung als  $g$



Schreibweisen  $f(n) \in o(g(n))$   $f(n) = o(g(n)) \dots$

Gängige Größenordnungen:

$$O(1) < O(\log n) < O(n) < O(n \cdot \log n) < O(n^2) < O(n^3) \\ < O(n^k) \text{ für festes } k$$

$$< O(n^{\log n}) < O(2^n)$$

" $<$ "  $\hat{=}$   $f \in o(g)$

polynomiale  
Größenordnungen

Algo von dieser Größenordnung  
heißt "von polynomieller Laufzeit"  
oder "effizient"

keine polynomiale  
Laufzeit

$O(2^n) \hat{=}$  exponentielle Laufzeit

Wie feststellen ob  $f \in o(g)$ ?

Nützliches Kriterium der Analysis

SATZ: Seien  $f, g: \mathbb{N} \rightarrow \mathbb{N}$ . Dann gilt:

$$f \in o(g) \iff \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

Beweis: " $\Rightarrow$ " Sei  $\varepsilon > 0$  beliebig, fest

$$\text{Def von } o(g) \Rightarrow \text{zu } \varepsilon \exists n_0 \text{ mit } f(n) \leq \varepsilon \cdot g(n) \quad \forall n \geq n_0$$

$$\Rightarrow \frac{f(n)}{g(n)} \leq \varepsilon \quad \forall n \geq n_0$$

$$\Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

" $\Leftarrow$ " Sei  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow \text{zu } c > 0 \exists n_0 \text{ mit}$

$$\frac{f(n)}{g(n)} \leq c \quad \forall n \geq n_0$$

$$\Rightarrow f(n) \leq c \cdot g(n) \quad \forall n \geq n_0 \Rightarrow f \in o(g) \quad \square$$

Beispiel: Unterschied polynomiale Laufzeit vs. exponentielle Laufzeit

haben Problem mit Inputgröße  $n$ , Laufzeit  $f(n)$

heutiger Rechner braucht 1 Stunde für Problemgröße  $n_0$

$$f(n_0) = 1 \text{ Std}$$

Bessere Reduzier 100 mal so schnell

$$f(n_{\text{neu}}) = 1 \text{ Std} = 100 \cdot f(n_0)$$

$\uparrow$   
gesucht

$$\Rightarrow f(n_{\text{neu}}) = 100 \cdot f(n_0)$$

Fall 1:  $f$  ist polynomial, etwa  $f(n) = n^3$

$$n_{\text{neu}}^3 = 100 \cdot n_0^3 \Rightarrow n_{\text{neu}} = \sqrt[3]{100} n_0$$

$\underbrace{\hspace{2cm}}$   
4, ...

Faktor

Fall 2:  $f$  ist exponentiell, etwa  $f(n) = 2^n$

$$2^{n_{\text{neu}}} = 100 \cdot 2^{n_0}$$

$$n_{\text{neu}} = \underbrace{\log(100)}_{\text{zwischen 6 und 7}} + n_0$$

$\leftarrow$  nur additiv

zwischen 6 und 7