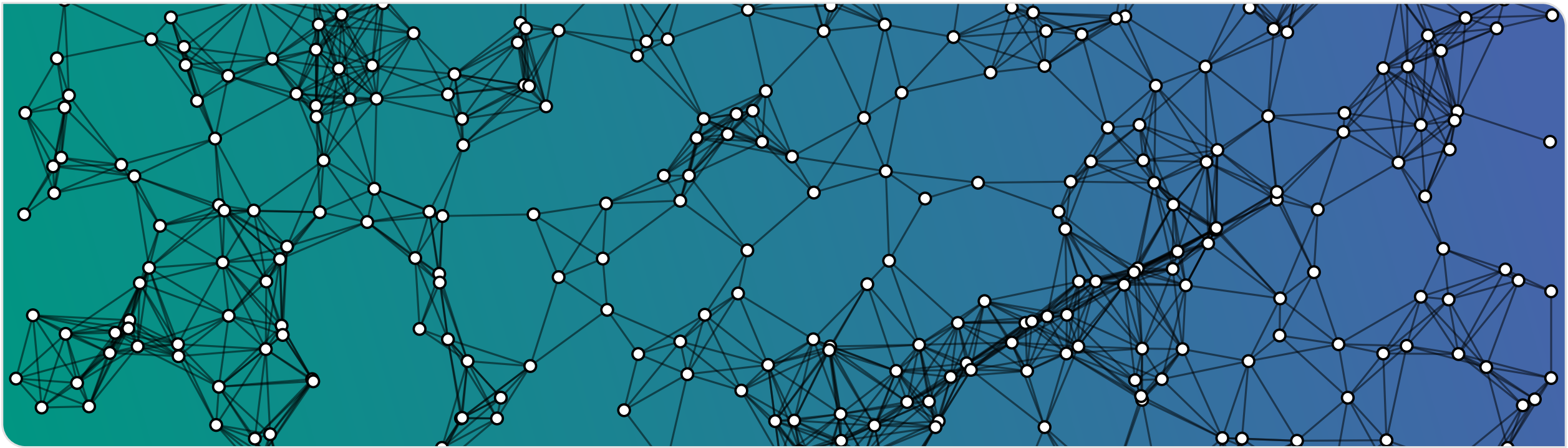
Karlsruher Institut für Technologie

# Algorithmen 1

## Übung 1
## Asymptotik, Pseudocode & Master-Theorem



# Organisatorisches

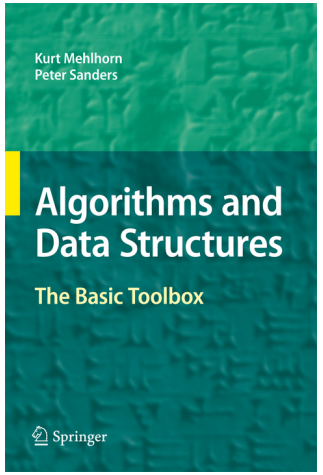
- Tretet gern dem Discord Server bei `https://scale.iti.kit.edu/teaching/2022ss/algo1/start`

## Literatur zur Vorlesung

- Vorlesung richtet sich nicht strikt nach einem konkreten Buch
- Folien sollten zum Verständnis ausreichend detailliert sein

Für weitergehende Recherche

- Algorithms and Data Structures – The Basic Toolbox
  Kurt Mehlhorn, Peter Sanders
- Algorithmen und Datenstrukturen: Die Grundwerkzeuge
  Martin Dietzfelbinger, Kurt Mehlhorn, Peter Sanders





## Konventionen

- Algorithmen sollen immer so effizient wie möglich sein
- Wenn nicht anders angegeben, ist $\log(\cdot)$ immer zur Basis 2

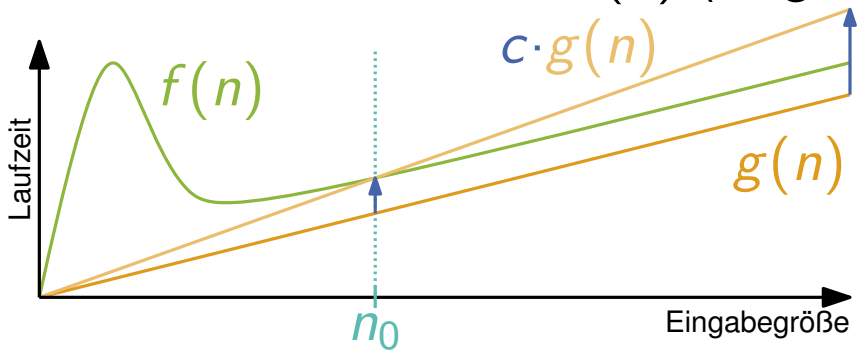
# Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

**Landau-Notation, *O*-Notation, big-*O* notation**

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c\, \exists n_0\, \forall n > n_0\; f(n) \leq c \cdot g(n)$

- $n$ bezeichnet die Größe der Eingabe
- Intuitiv: Wenn die Eingabe ausreichend groß ist (mindestens $n_0$), dann braucht ein Algorithmus mit Laufzeit $f(n)$ (ungefähr) höchstens so lang wie einer mit Laufzeit $g(n)$.




$f(n) \in \omega(g(n))$ $f(n)$ wächst schneller als $g(n)$

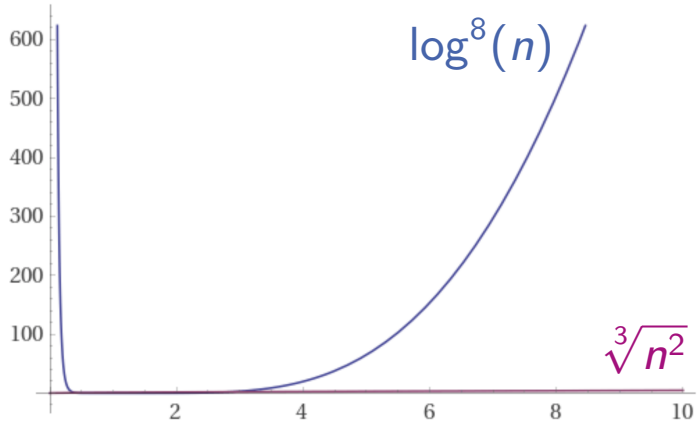
$f(n) \in \Omega(g(n))$ $f(n)$ wächst min. so schnell wie $g(n)$

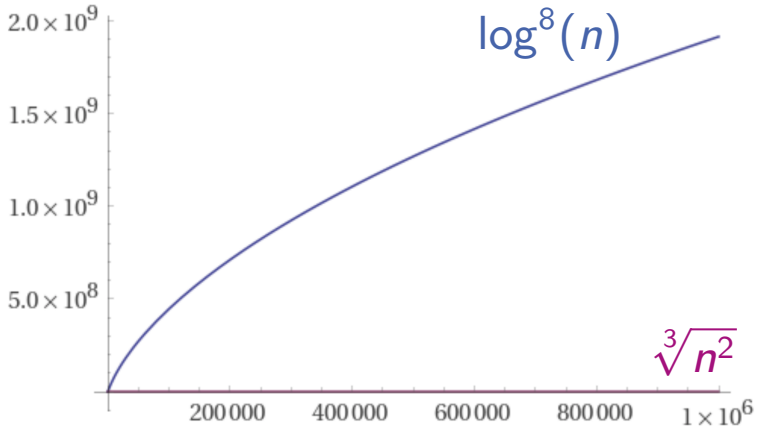
$f(n) \in \Theta(g(n))$ $f(n)$ und $g(n)$ wachsen gleich schnell

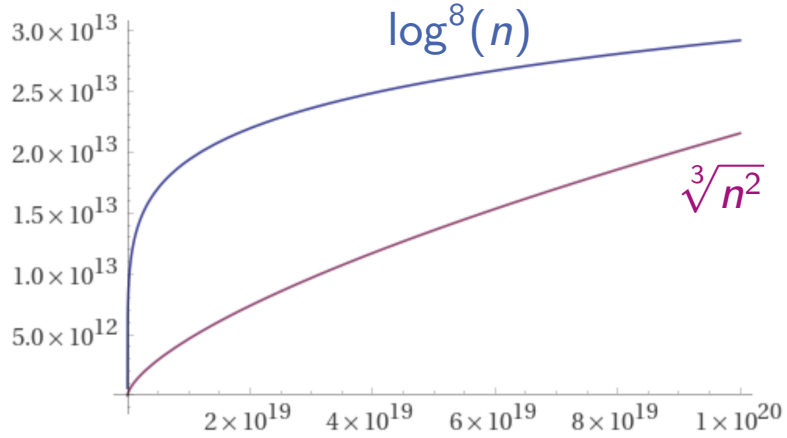
$f(n) \in o(g(n))$ $f(n)$ wächst langsamer als $g(n)$

# Asymptotik – Tipps

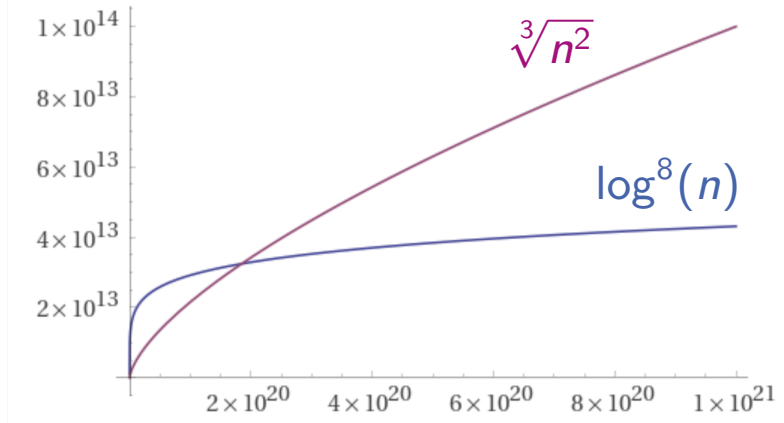
- Plots helfen, um ein Gefühl zu bekommen, können aber auch trügerisch sein




- Mentaler Shortcut: Alle Logarithmen wachsen langsamer als alle Polynome
- Der Logarithmus ist unheimlich klein
  ca. $6 \cdot 10^{79}$ Atome im Universum $\rightarrow \log(6 \cdot 10^{79}) \approx 265$
- Eine Laufzeit in $O(n \log(n))$ nennt man **quasi-linear**
  manchmal auch $O(n \log^k(n))$




- In der Praxis ist ab quadratisch nur bedingt brauchbar, exponentiell geht gar nicht

# Asymptotik – Beispiele

$$O(\log\log(n)) \subseteq O(\log(n)) \subseteq O(n^{1/x}), x \geq 1 \subseteq O(n) \subseteq O(n^x), x \geq 1, \subseteq O(x^n), x > 1$$

$$\exists c\, \exists n_0\, \forall n > n_0\ f(n) \leq c \cdot g(n)$$

**Zeige: $10n \in O(n^2)$**

$$\begin{aligned} 10n &\leq c \cdot n^2 \\ n &\leq \tfrac{c}{10} \cdot n^2 \quad c = 10 \\ n &\leq n^2 \\ 1 &\leq n \qquad n_0 = 1 \end{aligned}$$

$$\begin{aligned} \log(10) + \log(n) &\leq \log(c) + 2\log(n) \\ \log(10) - \log(c) &\leq \log(n) \\ \tfrac{10}{c} &\leq n \quad c = 10\ n_0 = 1 \qquad c = 5\ n_0 = 2 \end{aligned}$$

**Zeige: $n^{1+\varepsilon} \in \Omega(n \log(n) \log\log(n)), \varepsilon > 0$**

$$\exists c\, \exists n_0\, \forall n > n_0\ c \cdot f(n) \geq g(n)$$

$$\begin{aligned} c \cdot n^{1+\varepsilon} &\geq n\log(n)\log\log(n) \\ c \cdot n^{\varepsilon} &\geq \log(n)\log\log(n) \\ c \cdot n^{\varepsilon} &\geq \log^2(n) \\ \log(c) + \varepsilon\log(n) &\geq 2\log\log(n) \quad c = 1 \\ \varepsilon\log(n) &\geq 2\log\log(n) \\ \tfrac{\varepsilon}{2} &\geq \frac{\log\log(n)}{\log(n)} \end{aligned}$$

Alles was diese Ungleichung erfüllt, erfüllt auch diese.

$n = 2^{2^m}$

$$\begin{aligned} \tfrac{\varepsilon}{2} &\geq \tfrac{m}{2^m} \\ \varepsilon &\geq \tfrac{m}{2^{m-1}} \\ \varepsilon &\geq \tfrac{2^{m/2}}{2^{m-1}} \\ \varepsilon &\geq 2^{-m/2+1} \\ \log(\varepsilon) &\geq -m/2 + 1 \\ 2 - 2\log(\varepsilon) &\leq m \end{aligned}$$

Alles was diese Ungleichung erfüllt, erfüllt auch diese.

$\log\log(n) = m$

$$\begin{aligned} 2 - 2\log(\varepsilon) &\leq \log\log(n) \\ 2^{2-2\log(\varepsilon)} &\leq \log(n) \\ 2^{2^{2-2\log(\varepsilon)}} &\leq n \end{aligned}$$

$$n_0 = 2^{2^{2-2\log(\varepsilon)}}$$

# Asymptotik – Beispiele (2)

$$O(\log\log(n)) \subseteq O(\log(n)) \subseteq O(n^{1/x}), x \geq 1 \subseteq O(n) \subseteq O(n^x), x \geq 1, \subseteq O(x^n), x > 1$$

**Zeige: $2^{O(\log(n))} \subseteq O((1+\varepsilon)^n), \varepsilon > 0$**

$$\exists c\, \exists n_0\, \forall n > n_0\ f(n) \leq c \cdot g(n)$$

- Zu zeigen: Für jedes $f(n) \in 2^{O(\log(n))}$ gilt: $f(n) \in O((1+\varepsilon)^n)$

$$\exists c'\, \exists n_0'\, \forall n > n_0'\ f(n) \leq 2^{c' \log(n)} = n^{c'}$$

$$n^{c'} \leq c \cdot (1+\varepsilon)^n$$

$$c' \log(n) \leq \log(c) + n \log(1+\varepsilon) \quad c = 1$$

$$\log(n) \leq \frac{\log(1+\varepsilon)}{c'} n \quad n = 2^m$$

$$\frac{m}{2^m} \leq \frac{\log(1+\varepsilon)}{c'}$$

$$\frac{2^{m/2}}{2^m} \leq \frac{\log(1+\varepsilon)}{c'}$$

$$2^{-m/2} \leq \frac{\log(1+\varepsilon)}{c'}$$

$$m \geq 2 \log\left(\frac{c'}{\log(1+\varepsilon)}\right) \quad m = \log(n)$$

$$\log(n) \geq \log\left(\left(\frac{c'}{\log(1+\varepsilon)}\right)^2\right)$$

$$n \geq \left(\frac{c'}{\log(1+\varepsilon)}\right)^2 =: n_0$$

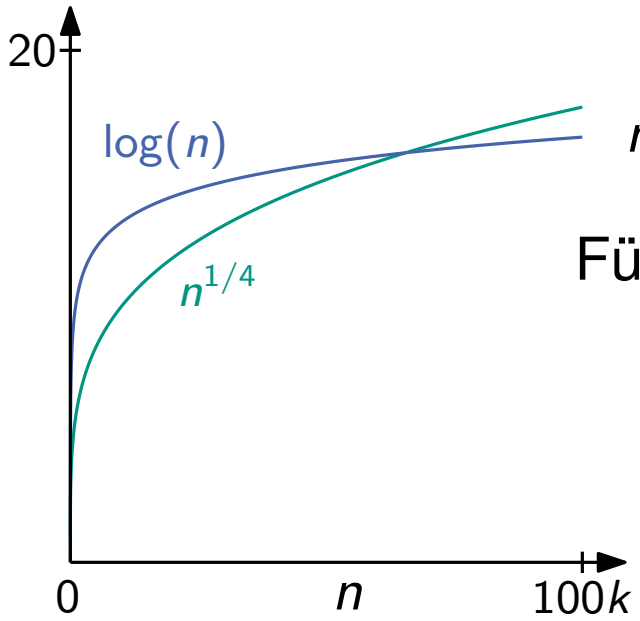
# Asymptotik – Beispiele (3)

**Zeige:** $\frac{n}{\log(n)} \in \omega(\sqrt{n})$

$$f(n) \in \omega(g(n)) \Rightarrow \ell = \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

$$\ell = \lim_{n\to\infty} \frac{\frac{n}{\log(n)}}{\sqrt{n}} = \lim_{n\to\infty} \frac{n}{\log(n)\sqrt{n}} = \lim_{n\to\infty} \frac{\sqrt{n}}{\log(n)} \geq \lim_{n\to\infty} \frac{\sqrt{n}}{n^{1/4}} = \lim_{n\to\infty} n^{1/4} = \infty$$

$\log(n) \leq n^{1/4}$ (für ausreichend großes $n$)




$n_0 = 2^{16}$ $\log(n_0) = 16$ $n_0^{1/4} = (2^{16})^{1/4} = 2^4 = 16$ $(n \geq n_0)$

Für $n_0 = 2^{16}$ gilt $\log(n_0) = n_0^{1/4}$.

Wie können wir zeigen, dass eine Funktion schneller wächst als eine andere? → Anstieg → Ableitung

$(\log(n))\frac{d}{dn} = n^{-1}$ $(n^{1/4})\frac{d}{dn} = \frac{1}{4}n^{-3/4}$

$$\ell = \lim_{n\to\infty} \frac{\frac{1}{4}n^{-3/4}}{n^{-1}} = \lim_{n\to\infty} \frac{1}{4}n^{1-3/4} = \lim_{n\to\infty} \frac{1}{4}n^{1/4}$$

# Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

| | | |
|---|---|---|
| Algorithmusidee | Wir iterieren alle Ziffern und ... | |
| Pseudocode | **for** $i$ in $\{0, 1, \ldots, n-1\}$ **do** | `https://scale.iti.kit.edu/teaching/2022ss/algo1/start` |
| Code | `for (int i = 0; i < n; ++i) { ... }` | |

- für Menschen gut lesbar
- richtige Abstraktionsebene

~~**wrongAlgo**(*input*)~~
~~  **return** correct output~~

- Beispiel: Was macht der von diesem Pseudocode beschriebene Algorithmus?

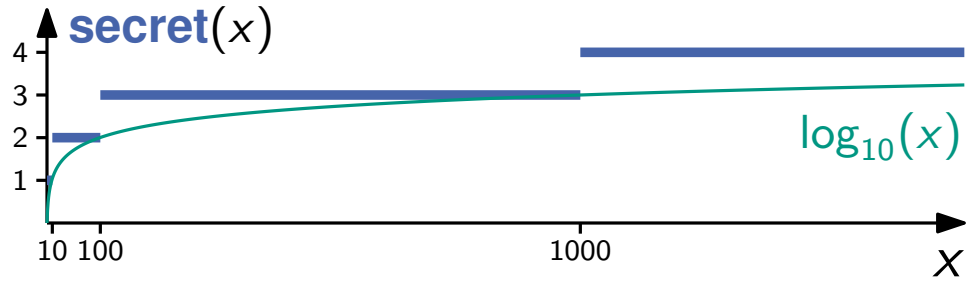
**secret**$(x)$
  **if** $x \leq 10$ **return** 1
  **return** $1 +$ **secret**$(x/10)$

Beispiel Eingaben:

$x \in [1, 10] \to 1$

$x \in [11, 100] \to 2$

$x \in [101, 1000] \to 3$




Gegeben eine Zahl $x \geq 1$, berechnet der Algorithmus $\lceil \log_{10}(x) \rceil$

# Master–Theorem

- Werkzeug zur Berechnung der Laufzeit von rekursiven Algorithmen

**Master–Theorem**

Sei $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ mit $f(n) \in \Theta(n^c)$ und $T(1) \in \Theta(1)$. Dann gilt

$$T(n) \in \begin{cases} \Theta(n^c) & \text{wenn } a < b^c, \\ \Theta(n^c \log n) & \text{wenn } a = b^c, \\ \Theta\left(n^{\log_b(a)}\right) & \text{wenn } a > b^c. \end{cases}$$

**log10**$(x)$

```
if x ≤ 10 return 1
return 1 + log10(x/10)
```

$T(1) \in \Theta(1)$ $\quad a \quad b \quad c$

$T(n) = f(n) + 1 \cdot T(n/10)$

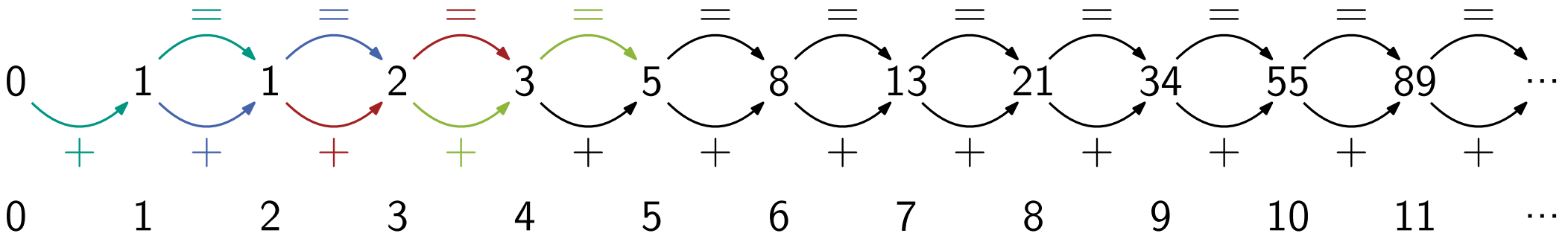
$f(n) \in \Theta(1) = \Theta(n^0)$

$b^c = 10^0 = 1 = a$

$T(n) = \Theta(n^0 \log(n))$

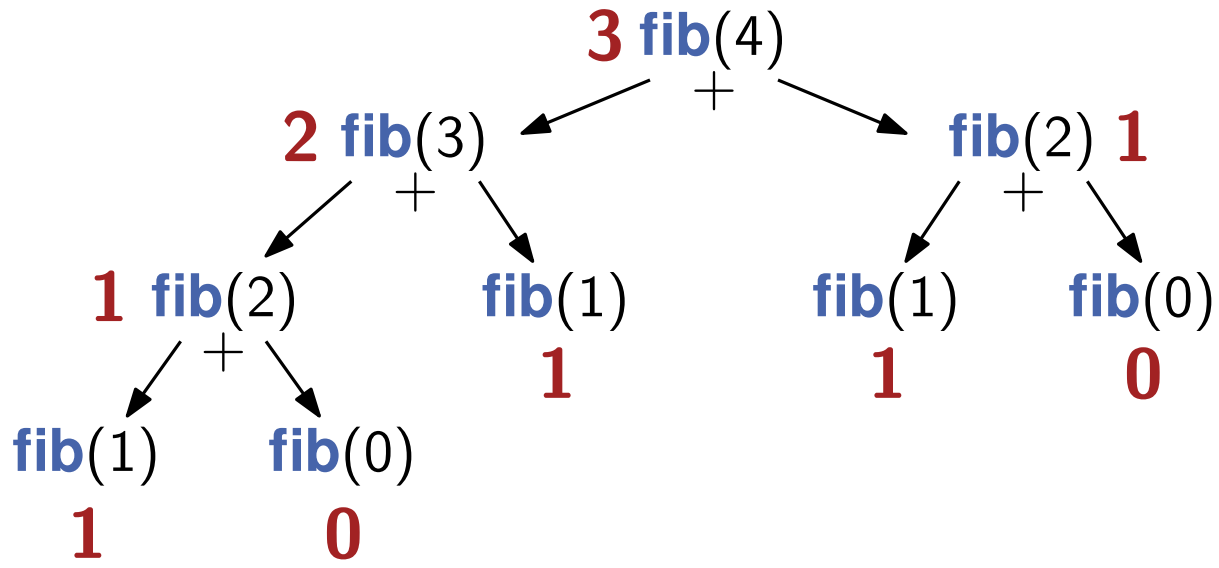
$= \Theta(\log(n))$

# Fibonacci




**fib**$(n)$

> **if** $n \leq 1$ **return** $n$
> **return** **fib**$(n-1)+$
> **fib**$(n-2)$



# Fibonacci – Laufzeit

**Master–Theorem**

Sei $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ mit $f(n) \in \Theta(n^c)$ und $T(1) \in \Theta(1)$. Dann gilt

$$T(n) \in \begin{cases} \Theta(n^c) & \text{wenn } a < b^c, \\ \Theta(n^c \log n) & \text{wenn } a = b^c, \\ \Theta\left(n^{\log_b(a)}\right) & \text{wenn } a > b^c. \end{cases}$$

```
fib(n)
    if n ≤ 1 return n
    return fib(n − 1) +
           fib(n − 2)
```

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &\leq 2 \cdot T(n-1) + 1 \\ &\leq 2 \cdot (2 \cdot T(n-2) + 1) + 1 \\ &\leq 4 \cdot T(n-2) + 3 \\ &\leq 8 \cdot T(n-3) + 7 \\ &\dots \\ \text{Induktion} \quad k = n \quad &\leq 2^k \cdot T(n-k) + (2^k - 1) \\ &\leq 2^n \cdot 1 + (2^n - 1) \in O(2^n) \end{aligned}$$

$T(n-2) \leq T(n-1)$

$$\begin{aligned} &\geq 2 \cdot T(n-2) + 1 \\ &\geq 2 \cdot (2 \cdot T(n-4) + 1) + 1 \\ &\geq 4T(n-4) + 3 \\ &\geq 8T(n-6) + 7 \\ &\dots \\ &\geq 2^k \cdot T(n-2k) + (2^k - 1) \quad k = \tfrac{n}{2} \\ &\geq 2^{n/2} \cdot 1 + (2^{n/2} - 1) \in \Omega(\sqrt{2}^n) \end{aligned}$$

# Fibonacci – Laufzeit (bessere obere Schranke)

- Was wir bereits wissen
  - $T(n) = T(n-1) + T(n-2) + 1$
  - $T(0) = T(1) = 1$
  - $T(n) \in \Omega(\sqrt{2}^n)$, wobei $\sqrt{2} \approx 1.4142$ und
  - $T(n) \in O(2^n)$
- Idee: $T(n) \leq c^n$, $c \in [\sqrt{2}, 2]$ konstant
  $\rightarrow T(n) \leq c^{n-1} + c^{n-2} + 1$
  Bedingung: $c^n \geq c^{n-1} + c^{n-2} + 1$
- Idee 2: $T(n) \leq c^n - 1$
  $\rightarrow T(n) \leq c^{n-1} - 1 + c^{n-2} - 1 + 1$
  Bedingung: $c^n - 1 \geq c^{n-1} - 1 + c^{n-2}$

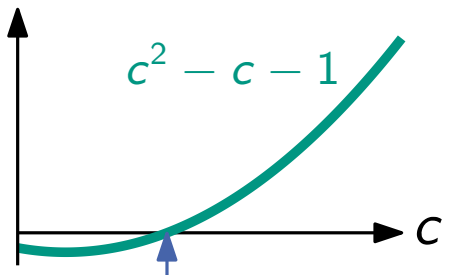
$$c^n \geq c^{n-1} + c^{n-2}$$
$$c^n - c^{n-1} - c^{n-2} \geq 0$$
$$c^2 - c - 1 \geq 0$$

Test: $T(0) \leq \phi^0 - 1 = 0, T(1) \leq \phi^1 - 1 \approx 0.618$
$\not\geq 1$ $\not\geq 1$

- Idee 3: $T(n) \leq 2 \cdot c^n - 1$
  $\rightarrow T(n) \leq 2c^{n-1} - 1 + 2c^{n-2} - 1 + 1$
  Bed.: $2c^n - 1 \geq 2c^{n-1} - 1 + 2c^{n-2}$

$$2c^n \geq 2c^{n-1} + 2c^{n-2}$$
$$c^n \geq c^{n-1} + c^{n-2} \qquad c = \phi$$

Test: $T(0) \leq 2\phi^0 - 1 = 1 \geq 1$
$T(1) \leq 2\phi^1 - 1 \approx 2.236 \geq 1$




$c = \frac{1+\sqrt{5}}{2} =: \phi \approx 1.6180$

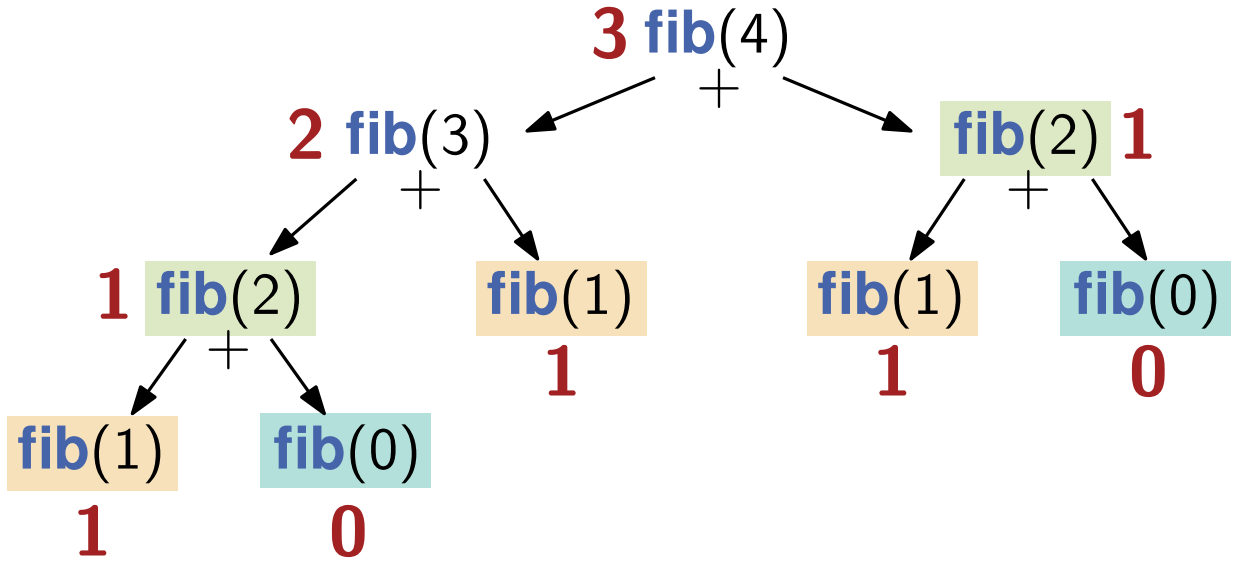
$T(n) \leq 2 \cdot \phi^n - 1$

$T(n) \in O(\phi^n)$
$\subseteq O(1.6181^n)$

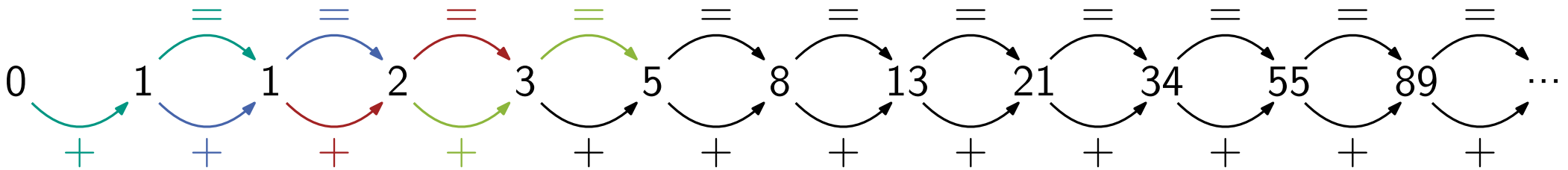
# Fibonacci – Das Problem

- Im Rekursionsbaum verdoppeln sich die Knoten mit jeder weiteren Lage (ungefähr)
- Mit wachsender Anzahl von Lagen schrumpft $n$ nicht schnell genug.
  → Viele Lagen!
- Viele Teilergebnisse werden mehrfach berechnet

```
fib(n)
    if n ≤ 1 return n
    return fib(n − 1) +
           fib(n − 2)
```



# Fibonacci – Schneller




**fib**$(n)$

```
fib := 0                          // O(1)
fibNext := 1                      // O(1)
for i from 0 to n − 1 do          // O(n)
    temp := fib                   // O(1)
    fib := fibNext                // O(1)
    fibNext := fib + temp         // O(1)
return fib                        // O(n)
```

**fib**$(0)$

| | |
|---|---|
| 0 | |
| 1 | |
| 0 | -1 |

**fib**$(1)$

| | | |
|---|---|---|
| 0 | | 1 |
| 1 | | 1 |
| 0 | 0 | 1 0 |
| 0 | | |
| 1 | | |
| 1 | | |

**fib**$(2)$

| | | |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 0 1 | 1 1 | 2 1 |
| 0 | 1 | |
| 1 | 1 | |
| 1 | 2 | |

- Fibonacci rekursiv: exponentielle Laufzeit. Fibonacci iterativ: lineare Laufzeit.