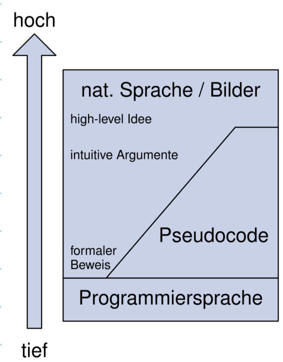
Algo Zusammenfassung

- Abstraktionsebenen

Unterschied zw. natürliche Sprache,

Pseudocode und Programmiersprache (Implementierung).
↳ Abstrahiert z.B. Paradigmen → gut zu lesen



- „Beschreibe" bedeutet nicht Pseudocode, sondern high-level Idee!

- Asymptotische Analyse
  - ↳ Motivation: Effizienzmaß unabhängig von Hardware
  - ↳ „Wie schnell wächst f?"
  - ↳ auch „Landau-Notation"

Notation	Asymptotischer Vergleich	Formale Definition
$f(n) \in \omega(g(n))$	$f(n)$ wächst schneller als $g(n)$	$\forall c\, \exists n_0\, \forall n > n_0\;\; f(n) > c \cdot g(n)$
		(egal wie viel schneller der Rechner für $f(n)$, für genügend großes n ist $f(n)$ größer als $g(n)$)
$f(n) \in \Omega(g(n))$	$f(n)$ wächst min. so schnell wie $g(n)$	$\exists c\, \exists n_0\, \forall n > n_0\;\; c \cdot f(n) \geq g(n)$
		(wenn der Rechner für $f(n)$ langsam genug und n groß genug ist, dann ist $f(n)$ größer als $g(n)$)
$f(n) \in \Theta(g(n))$	$f(n)$ und $g(n)$ wachsen gleich schnell	$f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n))$
		(es hängt vom Rechner ab, welche Laufzeit für großes n schneller wächst)
$f(n) \in O(g(n))$	$f(n)$ wächst max. so schnell wie $g(n)$	$\exists c\, \exists n_0\, \forall n > n_0\;\; f(n) \leq c \cdot g(n)$
		(wenn der Rechner für $f(n)$ schnell genug und n groß genug ist, dann ist $f(n)$ kleiner als $g(n)$)
$f(n) \in o(g(n))$	$f(n)$ wächst langsamer als $g(n)$	$\forall c\, \exists n_0\, \forall n > n_0\;\; c \cdot f(n) < g(n)$
		(egal wieviel langsamer der Rechner für $f(n)$, für genügend großes n ist $f(n)$ kleiner als $g(n)$)

- Definition durch z.B. Limes:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} \infty \iff f(n) \in \omega(g(n)) \\ c \in \mathbb{R} \iff f(n) \in \Theta(g(n)) \\ 0 \iff f(n) \in o(g(n)) \end{cases}$$

- „×" ist eine Äquivalenzrelation!

- Rechenregeln:

Konstante Faktoren
$a \cdot f(n) \in \Theta(f(n))$

Monome
- $a \leq b \Rightarrow n^a \in O(n^b)$
- $n^a \in \Theta(n^b) \Leftrightarrow a = b$

Transitivität
$f_1(n) \in O(f_2(n)) \wedge f_2(n) \in O(f_3(n)) \Rightarrow f_1(n) \in O(f_3(n))$

Summen
$f_1(n) \in O(f_3(n)) \wedge f_2(n) \in O(f_3(n)) \Rightarrow f_1(n) + f_2(n) \in O(f_3(n))$

Produkte
$f_1(n) \in O(g_1(n)) \wedge f_2(n) \in O(g_2(n)) \Rightarrow f_1(n) \cdot f_2(n) \in O(g_1(n) \cdot g_2(n))$

Teile und Herrsche (divide and conquer)

- Idee: zerteile ein Problem solange in kleinere Teile, bis das Problem trivial ist.
  ↳ Nachteil: Laufzeit schwieriger abzuschätzen
- Rekursionsformel:

$$T(n) = \begin{cases} \Theta(1) & \text{, Problem trivial} \\ a \cdot T(\frac{n}{b}) + \Theta(n^c), & \text{sonst} \end{cases}$$

a: „in wie viele Teilprobleme wird jedes Problem zerlegt?"

b: „wie groß ist die nächste Probleminstanz?"

c: „was ist die Laufzeit einer Instanz?" (wenn polynomiell!)

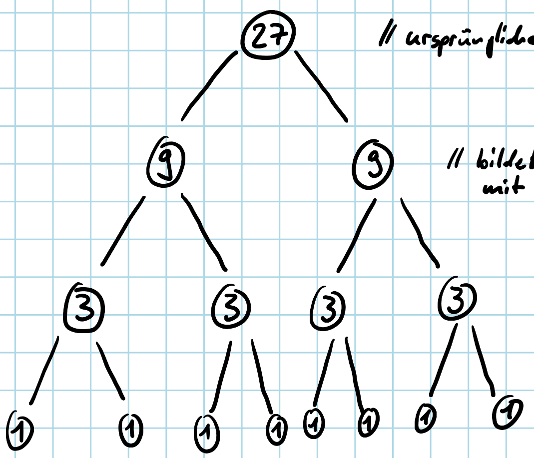
- Bäume

Es bilden sich a-äre Instanzbäume (jeder Knoten hat a Kinder).

z.B.

$$T(n) = \begin{cases} \Theta(1) & \text{, trivial} \\ 2 \cdot T(\frac{n}{3}) + \Theta(n^1), & \text{sonst} \end{cases}$$

bildet mit $n = 27$ den folgenden Baum:



// ursprüngliches Problem

// bildet 2 kleinere Probleme, mit der Größe $\frac{27}{3}$

// Alle Instanzen haben die Laufzeit $\Theta(n)$

// insg. $\Theta(27) + 2 \cdot \Theta(9) + 4 \cdot \Theta(3) + 8 \cdot \Theta(1)$

Allgemeiner: $T(n) \in \Theta\left(\sum_{i=0}^{\log_3(n)} 2^i \cdot \Theta((\frac{n}{3})^1) \right)$

... aber wieso?

- Ein Baum, in welchem jeder Knoten genau a Kinder hat, gibt es in der i-ten Ebene genau a^i Instanzen.
- Wird die Problemgröße jeder Unterinstanz durch b geteilt, so ist jede Problemgröße auf der i-ten Ebene $\frac{n}{b^i}$.
- Ist b wie oben definiert, so gibt es $\log_b(n)$ Ebenen

$\rightarrow$ Wir können über die Ebenen iterieren, und alle Laufzeiten aufaddieren!

Anzahl Ebenen

Laufzeit für jede Instanz

$$T(n) \asymp \sum_{i=0}^{\log_b(n)} a^i \cdot \Theta\left(\left[\frac{n}{b^i}\right]^c\right)$$

Anzahl Instanzen pro Ebene

Problemgröße pro Ebene

Alternativ: Mastertheorem

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

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