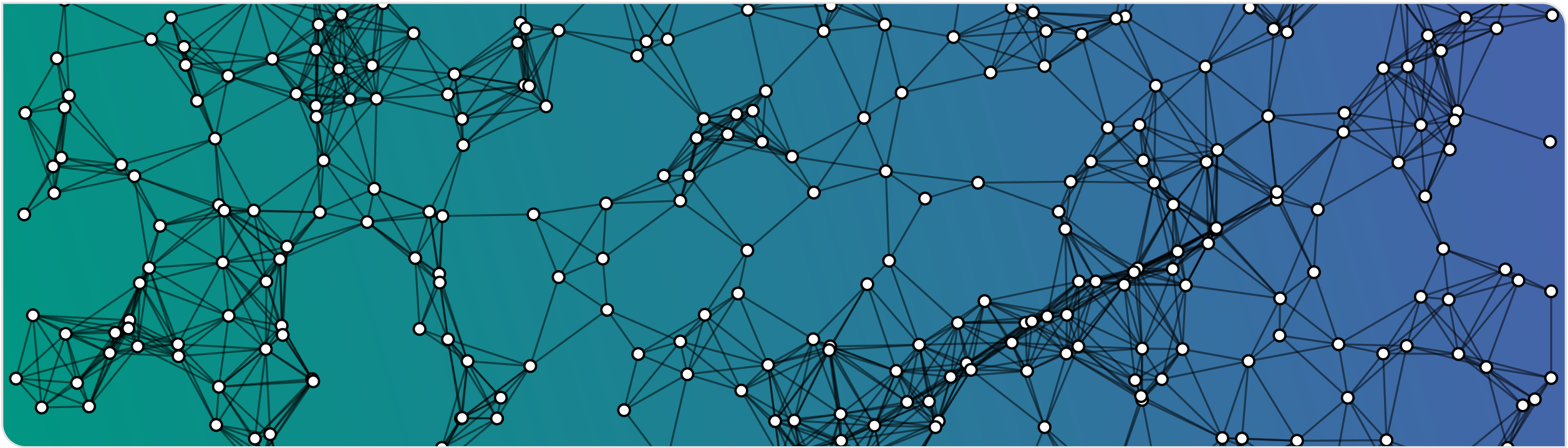
Karlsruher Institut für Technologie

Algorithmen 1

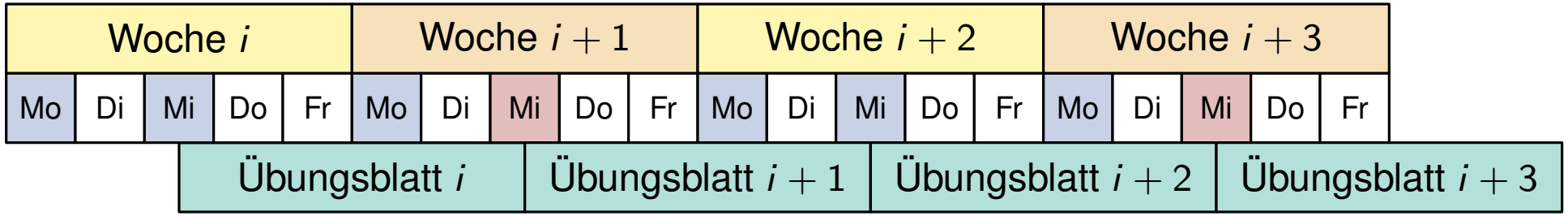
Einführung
Schriftliche Multiplikation und Landau-Notation



Ablauf

Grundsätzlich gilt: mehr Details auch nochmal auf der Homepage

`https://scale.iti.kit.edu/teaching/2022ss/algo1/start`



i ungerade
wir sind in Woche 1

Übungsblätter & Tutorien

- sehr wichtig für den Lernerfolg
- Klausurbonus
- Einteilung bis **morgen 18:00 Uhr**

Übung

- machen Max und Marcus
- vertieft den Stoff der Vorlesung
- mehr dazu nächsten Mittwoch

Fragen $\rightarrow$ Discord (Link im Ilias)

app.klicker.uzh.ch/join/algo1

Techniktest

Wie formell soll es sein?

Algorithmus? Kann man das essen?

al-Chwarizmi

- persischer Rechenmeister und Astronom
- lebte ca. 780–840 in Bagdad
- latinisierter Name: Algorismi
- Herkunft des Begriffs „Algorithmus“

Moderne Definition (Wikipedia)

- Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen.
- Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten.
- Damit können sie zur Ausführung in ein Computerprogramm implementiert, aber auch in menschlicher Sprache formuliert werden.
- Bei der Problemlösung wird eine bestimmte Eingabe in eine bestimmte Ausgabe überführt.

Was lernen wir hier?

Wissen, verstehen, anwenden

- Vokabular an Fachbegriffen
- algorithmische Bausteine
- algorithmische Techniken

Was ist eine amortisierte Analyse?

Was macht Teile und Herrsche?

Was ist eine Rekurrenz und wie löst man das?

Was ist ein Heap? Was ist ein Suchbaum? Was kann ich damit machen?

Was ist ein dynamisches Programm?

Was ist ein greedy Algorithmus?

Analysieren, evaluieren, erschaffen

- Laufzeitanalyse
- Entwicklung eigener Algorithmen
- Auswahl geeigneter Techniken und Bausteine
- Wechsel zwischen Abstraktionsebenen
- Aufbau mentaler Shortcuts

Ist der Algorithmus schnell genug?

Wie kann ich ein gegebenes Problem effizient lösen?

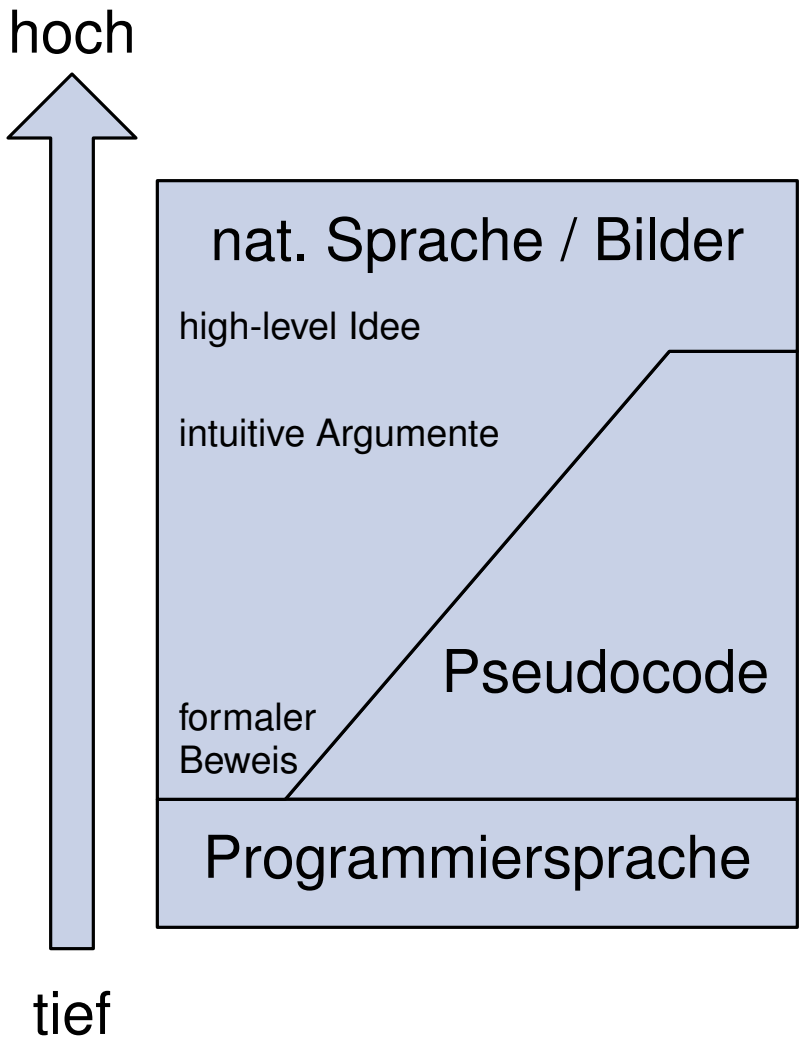
Welche Datenstruktur sollte ich verwenden?

Welche algorithmische Idee könnte funktionieren?

Ist meine Idee wirklich umsetzbar?

Ist mein Algorithmus korrekt?

Abstraktionsebene? Mentale Shortcuts?



Wahl der richtigen Abstraktionsebene

- beim Programmieren will man nicht über Transistoren nachdenken müssen
- die Programmiersprache abstrahiert von der Hardware

Algorithmenentwurf

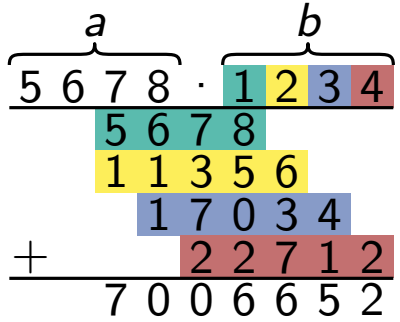
- beim Ausdenken des Algorithmus will man zunächst nicht über das Programmieren nachdenken
- wir brauchen zusätzliche Abstraktionsebenen

Problem & Lösung

- viele high-level Ideen gehen beim konkretisieren kaputt
- Konkretisierung oft gar nicht so einfach
- mentale Shortcuts: Intuition was geht und was nicht
- bekommt man nur mit viel Training

Zum warm werden: schriftliche Multiplikation

Wie geht das nochmal?

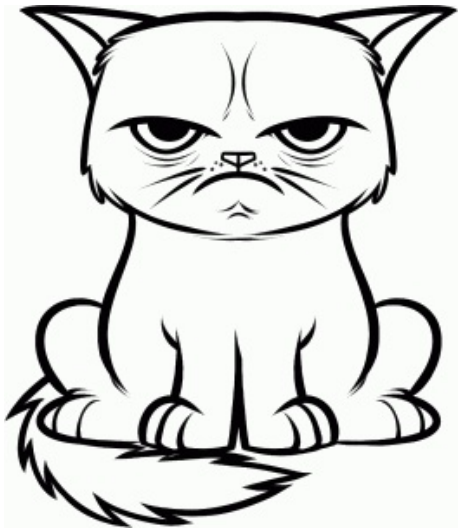


- berechne $a \cdot b_i$ für jede Ziffer b_i von $b = (b_{n-1} \dots b_0)$
- verschiebe Ergebnis um i Stellen nach links
- addiere Ergebnisse

Wie viele Schritte braucht das, wenn a und b jeweils n Ziffern haben?

- Berechnung von $a \cdot b_i$ für ein i $\rightarrow 2n-1$ Operationen insgesamt (alle i): $2n^2 - n$
  - n Multiplikationen
  - zusätzlich: Übertrag $\rightarrow n-1$ Additionen (oder $2n-2$?)
- schriftliche Addition am Ende $n-1$ Schritte $\rightarrow (n-1)(2n+1)$
  - pro Schritt: addiere eine Zeile auf bisheriges Ergebnis drauf $\rightarrow n+1$ Additionen
  - zusätzlich: Übertrag $\rightarrow n$ Additionen
  - beachte: bisheriges Ergebnis hat nicht zu viele Stellen

Je genauer desto besser?



Warum ist das so kompliziert?
Und wen interessiert das überhaupt so genau?

Eine zu genaue Betrachtungsweise…

- erschwert die Analyse
- bringt wenig relevante Einsicht
- versperrt den Blick auf das Wesentliche bei der Suche nach einem besseren Algorithmus

Wie ungenau darf's denn sein?

- hängt zum Teil von der Situation ab
- fast immer gilt:
  - Terme niedriger Ordnung interessieren nicht $(4n^2 - 2n - 1 \to 4n^2)$
  - konstante Faktoren interessieren nicht $(4n^2 \to n^2)$
  - formales Werkzeug: Landau-Symbole (O-Notation)

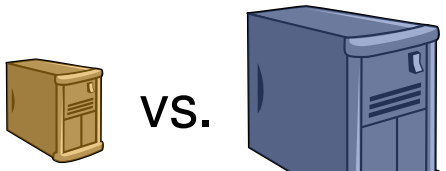
Bild: Grumpy cat line art / XXspiritwolf2000XX / Creative Commons

Welcher Algorithmus ist schneller?

Alice und Bob haben einen Algorithmus entwickelt

- ihre Laufzeit hängt von der Problemgröße n ab
- Bobs Algorithmus benötigt $f(n) = n^2 + 2$ Schritte
- Alice Algorithmus benötigt $g(n) = 4n + 1$ Schritte

Und wenn Bobs Rechner schneller ist?



vs.

Noch schneller?

Landau-Notation: Vergleich von Laufzeiten

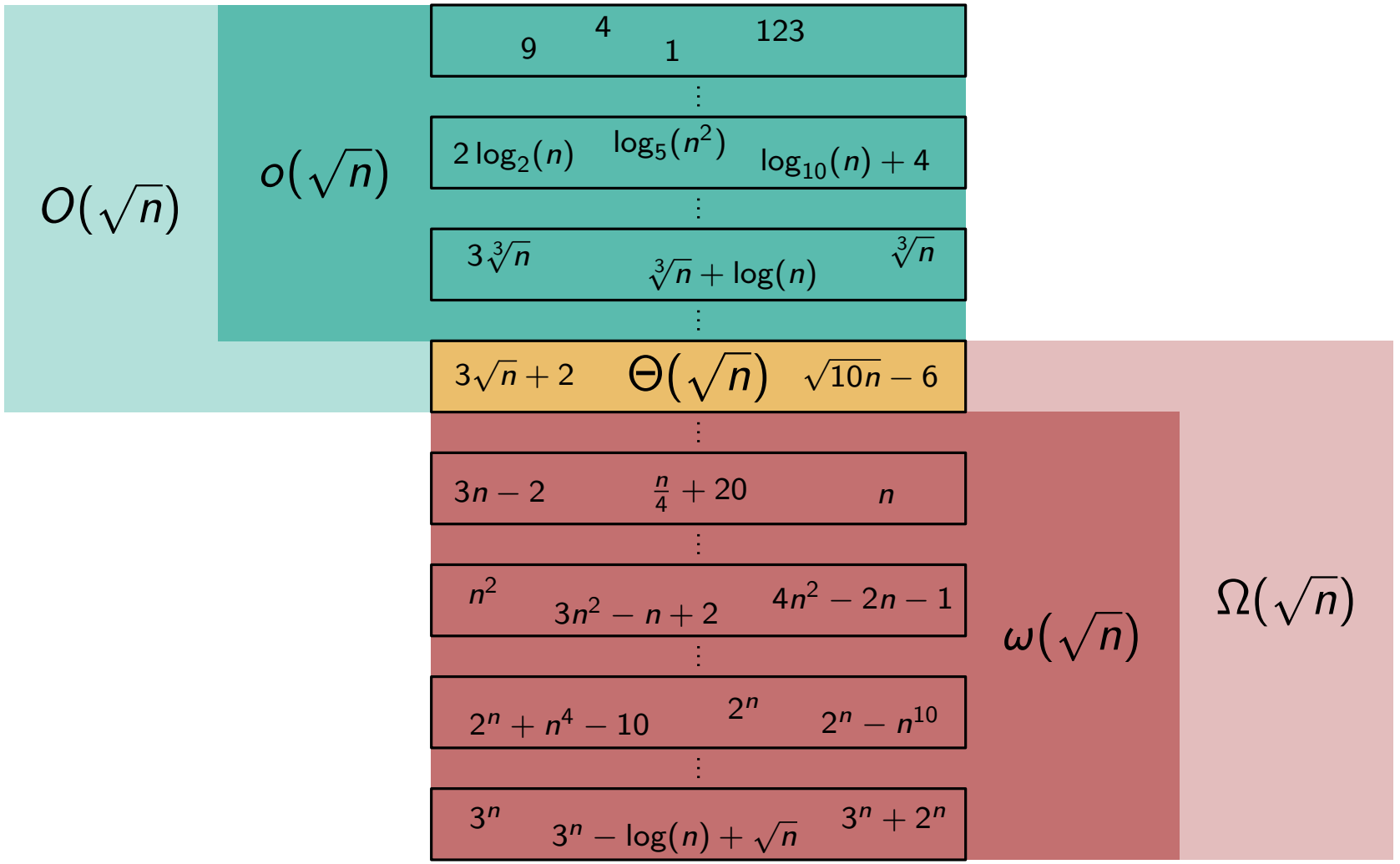
Notation	Asymptotischer Vergleich	Formale Definition
$f(n) \in \omega(g(n))$	$f(n)$ wächst schneller als $g(n)$	$\forall c\, \exists n_0\, \forall n > n_0\ f(n) > c \cdot g(n)$
	(egal wie viel schneller der Rechner für $f(n)$, für genügend großes n ist $f(n)$ größer als $g(n)$)	
$f(n) \in \Omega(g(n))$	$f(n)$ 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 $g(n)$)	

Anmerkung

- asymptotisch gleich schnell zu wachsen ist eine Äquivalenzrelation
- $\Theta(g(n))$ ist die Äquivalenzklasse

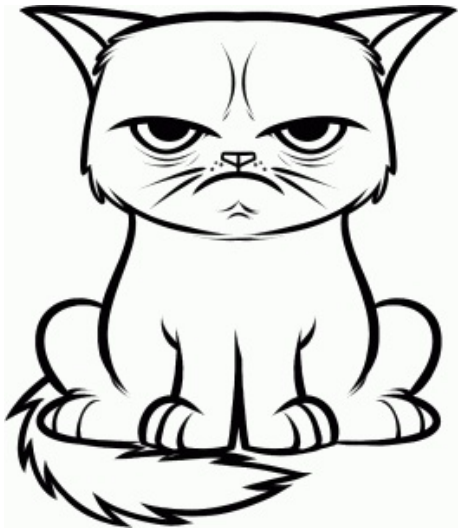
Landau-Notation: Ordnung auf Funktionen

kleine Funkionen
- langsam wachsend
- schnelle Laufzeit



großе Funkionen
- schnell wachsend
- langsame Laufzeit

Landau-Notation: Warum so kompliziert?



Warum ist das so kompliziert?

Wollten wir uns das Leben nicht einfacher machen?

Anmerkung

- mit der formalen Definition zu arbeiten ist mühsam
- was wir brauchen sind einfache Rechenregeln
- und etwas Übung

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

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

(sehr ähnliche Regeln gelten auch für die anderen Symbole)

Bild: Grumpy cat line art / XXspiritwolf2000XX / Creative Commons

app.klicker.uzh.ch/join/algo1

Landau-Notation: Klickern

Welche der Aussagen ist korrekt?

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

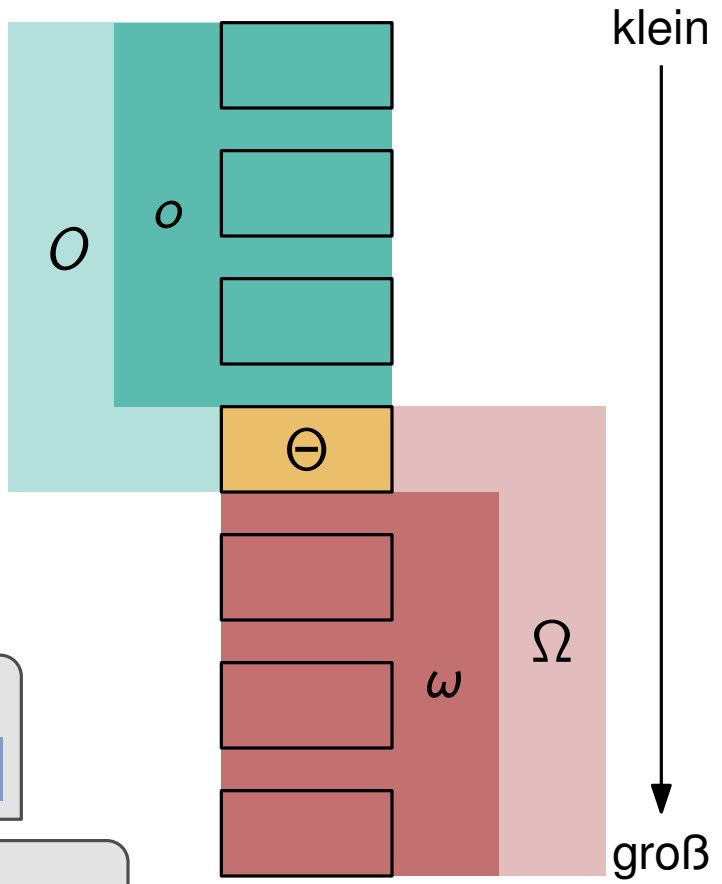
$f_1(n) \in O(f_2(n)) \wedge f_2(n) \in O(f_3(n)) \Rightarrow f_1(n) \in O(f_3(n))$

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

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



app.klicker.uzh.ch/join/algo1

Landau-Notation: alternative Definition

(Annahmen: Grenzwert existiert, $f(n)$ und $g(n)$ positiv)

Betrachte Grenzwert $\ell = \lim_{n\to\infty} \frac{f(n)}{g(n)}$

	$\ell = 0$	$\ell < \infty$	$0 < \ell < \infty$	$0 < \ell$	$\ell = \infty$
$f(n) \in$	$o(g(n))$	$O(g(n))$	$\Theta(g(n))$	$\Omega(g(n))$	$\omega(g(n))$

Welche Sichtweise soll ich denn jetzt verwenden?

Welche der Aussagen ist korrekt?

- immer die, die gerade am angenehmsten ist
- mentaler Shortcut: mit der Zeit kennt man Θ-Klassen wichtiger elementarer Funktionen
- die Rechenregeln von vorhin reichen dann aus
- wenn der mentale Shortcut fehlt: Grenzwertbetrachtung

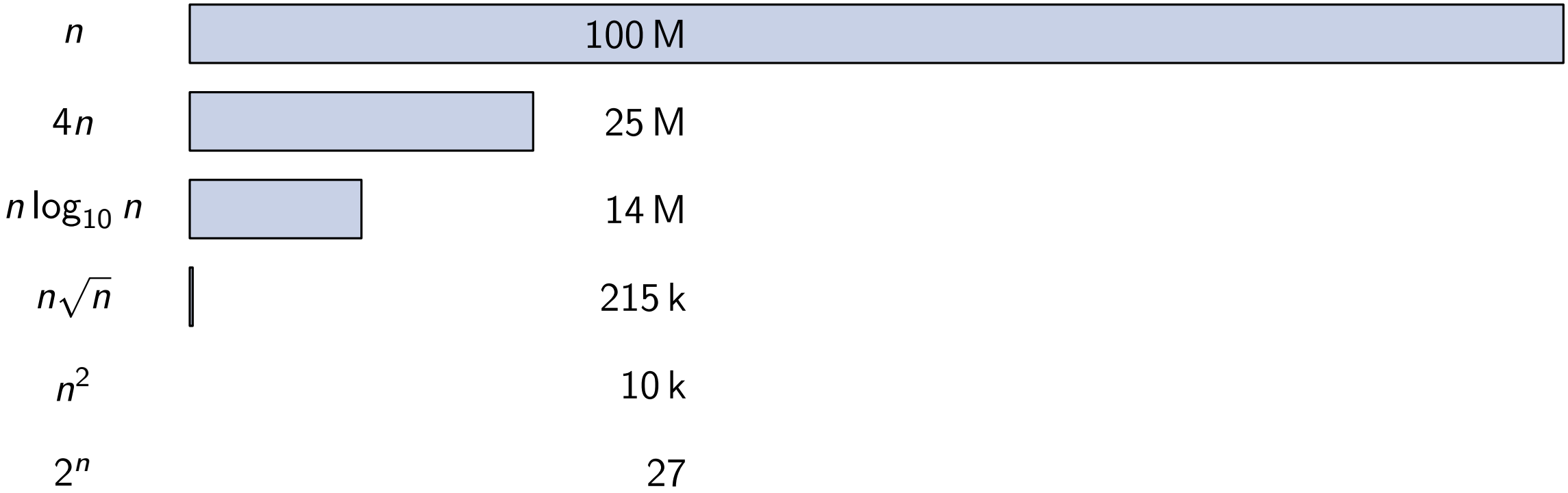
Ein paar elementare Funktionen

$$\left| 1 \right| \left| \log^* n \right| \left| \log n \right| \left| \log^2 n \right| \left| \sqrt[3]{n} \right| \left| \sqrt{n} \right| \left| n \right| \left| n^2 \right| \left| n^3 \right| \left| n^{\log n} \right| \left| 2^{\sqrt{n}} \right| \left| 2^n \right| \left| 3^n \right| \left| 4^n \right| \left| n! \right| \left| 2^{n^2} \right|$$

Asymptotik und tatsächliche Laufzeiten

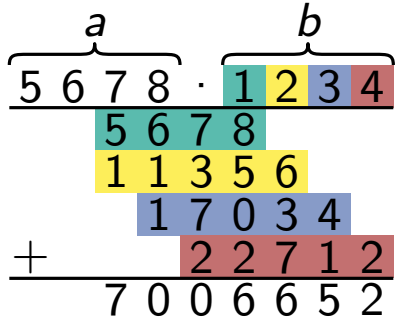
Gedankenexperiment

- Algorithmus braucht $f(n)$ Schritte
- Annahme: ca. $10\,\mathrm{M}$ Schritte pro Sekunde
- Wie groß darf die Eingabe sein, damit der Algorithmus nach $10\,\mathrm{s}$ fertig ist?



Zurück zur schriftlichen Multiplikation

Wie geht das nochmal?



- berechne $a \cdot b_i$ für jede Ziffer b_i von $b = b_n \ldots b_1$
- verschiebe Ergebnis um $i - 1$ Stellen nach links
- addiere Ergebnisse

Wie viele Schritte braucht das, wenn *a* und *b* jeweils *n* Ziffern haben?

- Berechnung von $a \cdot b_i$ für ein $i \rightarrow \Theta(n)$ Operationen — insgesamt (alle i): $\Theta(n^2)$
- schriftliche Addition am Ende — insgesamt: $\Theta(n^2)$
  - addiere in jedem Schritt zwei Zahlen mit $\Theta(n)$ Ziffern
  - nach $\Theta(n)$ Schritten hat man das Ergebnis

$$\Rightarrow \Theta(n^2)$$

Landau-Notation: zusätzliche Anmerkungen

Schreibweise mit $=$

- $O(g(n))$ ist eine Menge von Funktionen
- daher schreiben wir: $\left(4n^{0.98} - \sqrt{n}\right) \in O\left(n^{0.98}\right) \subseteq O(n)$
- oft sieht man auch: $\left(4n^{0.98} - \sqrt{n}\right) = O\left(n^{0.98}\right) = O(n)$

O in der Formel

- manchmal sieht man z.B.: $2^{\Theta(n)}$
- damit ist die Menge der Funktionen der Form $2^{f(n)}$ für $f(n) \in \Theta(n)$ gemeint
- also: Menge der Exponentialfunktionen ($f(n) \in 2^{\Theta(n)}$ wenn $c_1^n < f(n) < c_2^n$ für Konstanten c_1, c_2)
- damit kann man Laufzeiten noch gröber abschätzen ($\Theta(2^n) \neq \Theta(3^n)$ aber $2^{\Theta(n)} = 3^{\Theta(n)}$)

Schrumpfende Funktionen

- manchmal betrachtet man auch in n schrumpfende Funktionen
- z.B.: Wahrscheinlichkeit $1 - O\left(\frac{1}{n}\right)$ geht gegen 1 für $n \to \infty$
- das $O\left(\frac{1}{n}\right)$ gibt die Konvergenzgeschwindigkeit an

Zusammenfassung

Asymptotische Betrachtungsweise von Laufzeiten

- vereinfacht die Laufzeitanalyse durch Weglassen unwichtiger Informationen
- verhindert Ablenkung vom Wesentlichen beim Entwurf von Algorithmen
- liefert meist sehr gute Vorhersage für die Praxis ($\Theta(n)$ ist (fast) immer besser als $\Theta(n^2)$)
- tolles Werkzeug für faule Programmierer:
  - schließe asymptotisch langsamere Algorithmen direkt aus
  - weniger Arbeit beim Implementieren und Testen

Werkzeug für diese Betrachtungsweise: Landau-Notation

- mentale Shortcuts: einfache Rechenregeln + Eingruppierung elementarer Funktionen
- formale Definition(en) für Fälle in denen die Shortcuts nicht ausreichen

Schriftliche Multiplikation

- asymptotische Laufzeit $\Theta(n^2)$
- nächste Vorlesung: schnellerer Algorithmus