

5

Maschinen & Modelle

Algorithmen & Datenstrukturen · Sommersemester 2026

Prof. Dr. Sebastian Wild

5 Maschinen & Modelle

5.1 Grundbegriffe

5.2 Datenmodelle

5.3 Asymptotische Vergleiche

5.4 Mathematische Maschinenmodelle

5.5 Kostenmodelle

5.1 Grundbegriffe

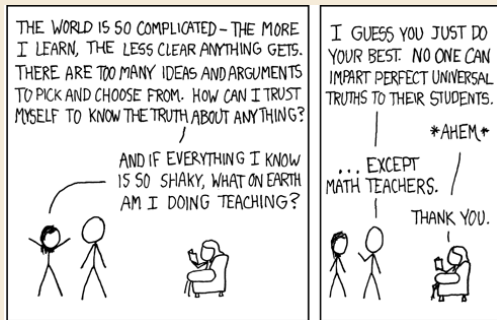
Philosophie

A&DS ist Teil eines **wissenschaftlichen** Studiengangs

Weniger ...



... und mehr



↪ Fokus auf "universal truths" der Algorithmik

- ▶ Model der Realität
- ▶ quantitative Vorhersagen
- ▶ Validierung der Modelle in Experimenten

↪ Müssen Algorithmen *mathematisch* modellieren

Was ist ein Algorithmus?



al-Chwarizmi
Namensgeber für
„Algorithmus“

Anschaulich:

Folge von Anweisungen \approx Kochrezept

Genauer:

z. B. Java Programm

1. mechanisch nachvollziehbar
 \rightsquigarrow kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung \neq endliche Berechnung!
3. löst ein *Problem*, $\leftarrow x + y$, nicht nur $17 + 4$
d. h. ganze Klasse von *Probleminstanzen*

Typisches Beispiel: *Bubblesort*

nicht ein spezielles Programm,
sondern die zugrundeliegende Idee

Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

↪ sehr schwammiger Begriff!

Für uns:

was soll passieren

- ▶ **Interface:** Spezifikation von Operationen API
= *abstract data type (ADT)*

application programming interface

Beispiel: *Union-Find*

- ▶ **Implementierung** eines Interface:
welche Daten werden gespeichert (oft: Invarianten)
(pro Operation) *Algorithmus*, der Spezifikation erfüllt

Beispiel: *QuickUnion*

wie machen wir das

Algorithmen für die Ewigkeit

Gute algorithmische Ideen sind zeitlos!

▶ Euklidischer Algorithmus für ggT

- ▶ von ≈ 300 v. Chr.
- ▶ i.W. unverändert in Verwendung

```
1 static int gcd(int p, int q) {  
2     if (q == 0) return p;  
3     else return gcd(q, p % q);  
4 }
```

▶ Mergesort (siehe Unit 7)

- ▶ 1945 beschrieben
- ▶ Programmiersprachen und Computer von damals und heute haben wenig gemeinsam
- ▶ trotzdem ist Mergesort einer der besten Sortieralgorithmen

↪ *Wie sollen wir „Algorithmen für die Ewigkeit“ formulieren?*

- ▶ sicher nicht in einer proprietären Sprache, die nur eine aktuelle Maschinengeneration funktioniert ...

Java? Ja, aber . . .

Warum nicht einfach: *Algorithmus = Java-Programm?*

- ▶ („Java ist doof, ich mag lieber C/C++/Python/Rust/OCAML/Groovy/Fortran/Haskell/Cobol/...“)
- ▶ um Algorithmen zu beschreiben wollen wir manchmal Details offen lassen
 - ▶ das geht in einem ausführbaren Java-Programm nicht!
 - ▶ aber grundsätzlich möglich, z. B. durch eine Interface-Methode, die wir nur aufrufen
- ▶ konkrete Programmiersprachen spiegeln Restriktionen **aktueller** Hardware und Performance-Tradeoffs wider
 - ▶ z. B. maximale Arraygröße in Java: $2^{31} - 1$ \rightsquigarrow auch maximale String-Länge
 - ▶ maximale Code-Größe einer Methode: 64 KB (kompilierter Bytecode)
- ▶ Werden oft *Java-Implementierung* unserer abstrakten algorithmischen Idee angeben
 - ▶ Java-Code als eindeutige **Beschreibung** der Idee
 - ▶ Algorithmus bleibt aber die **Idee**, nicht der (potentiell Java-spezifisch beschränkte) Code

5.2 Datenmodelle

Ein Wettrennen

Mensch (zu Fuß) oder Zug: Wer ist schneller *am Horizont?*

(beide stehen zu Beginn)

- ▶ Kommt darauf an! Am Anfang sicher der Mensch ...
- ▶ aber auf lange Sicht der Zug

Eingabegröße

„Zug vs. zu Fuß“ nur sinnvoll zu beantworten, wenn Gesamtstrecke bekannt ist

Analog: Zwei Algorithmen auf unterschiedlichen Eingabe vergleichen potentiell unfair

- ▶ falls A dann schneller ist als B , ist A wirklich besser, oder war A s Eingabe einfacher?

Für jedes algorithmische Problem legen wir **Parameter** der Eingabe fest die grob einfangen, wie schwierig/aufwändig diese Eingabe ist.

- ▶ einfachste Variante: **Eingabegröße n**
 - ▶ Was n genau ist, hängt vom Problem ab
 - ▶ Teil der Problembeschreibung
- ▶ **Beispiele:**
 - ▶ 3SUM: n = Anzahl Zahlen in der Eingabe
 - ▶ Union-Find: N = Anzahl Objekte, M = Anzahl von union und find Aufrufen

↪

Kosten eines Algorithmus A sind eine Funktion in n

- ▶ **Beispiel:** Brute-force 3SUM Algorithmus hat Laufzeit $T(n) = \frac{(n-2)(n-1)n}{6}$

Datenmodelle

Für viele Algorithmen ist selbst bei festem n die Laufzeit datenabhängig.

In der Algorithmik verwendet man deshalb einfache Datenmodelle (Annahmen an Daten):

▶ **Worst-Case-Performance:**

betrachte stets die für diesen Algorithmus *schlimmste* Eingabe der Größe n

▶ **Best-Case-Performance:**

betrachte stets die für diesen Algorithmus *günstigste* Eingabe der Größe n

▶ **Average-Case-Performance:**

bestimme den Durchschnitt/Erwartungswert für eine *zufällige* Eingabe der Größe n

↪ Kosten wieder nur eine Funktion in n

5.3 Asymptotische Vergleiche

Jeder Vergleich braucht einen Referenzpunkt

Mensch (zu Fuß) oder Zug: Wer ist schneller *am Horizont?*

Referenzpunkt: $n \rightarrow \infty$



Uns interessieren Schnellzüge!

Warum Asymptotik?

„Aber meine Eingabe ist doch nicht unendlich!“

- ▶ Stimmt.
- ▶ Aber: Verhalten für $n \rightarrow \infty$ ist (meistens) gute Approximation für moderate n
- ↔ Asymptotisch schnellerer Algorithmus (= bessere Skalierbarkeit!)
(meist) auch in der Praxis schneller

Standard-Vorgehen in der Algorithmik:

Vergleiche Θ -Klasse der Worst-Case-Laufzeit

Für präzisere Vergleiche: \sim Asymptotik

Schummelnde Algorithmen

Außerdem: Algorithmen-Design für feste Eingabegröße theoretisch langweilig:

▶ Für festes n gibt es nur **endlich viele verschiedene** Eingaben

↪ Trivialer Algorithmus: alle **vorab berechnen** und passende Lösung ausgeben!

▶ meist nicht praktikabel (Riesiger Code mit allen Fällen)

▶ aber nach Algorithmen-Definition nicht verboten 🐱

⚡ verhindert jegliche **Unmöglichkeitsergebnisse** ("*lower bounds*")

(= Beweise, dass algorithmisches Problem nicht schneller als mit gewisser Laufzeit lösbar ist)

große Errungenschaft der Informatik!

↪ Brauchen Asymptotik um Suche nach **optimalen Algorithmen** sinnvoll zu definieren

5.4 Mathematische Maschinenmodelle

Was ist schon ein int?

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- ▶ **Java:** $32n$ bit, da int eine 32 Bit Zahl ist.
= typische *Wortgröße* aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- ▶ **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \lg(M + 1) \rceil$ bits
also $n \cdot \lceil \lg(M + 1) \rceil$ bits
- ▶ Wenn M klein ist, würden wir aus Effizienzgründen trotzdem 32-bit ints wählen.

Wortbreite

Wir denken ja aber asymptotisch . . . was ist denn bitte „ $\lim_{n \rightarrow \infty} \text{int}$ “?

- ▶ Allgemeine Version: Halten *Wortbreite* w **variabel** (ein Parameter der Maschine)
- ↔ Mit w -Bit Zahlen können wir wie mit `int` rechnen (Addition in $O(1)$)
- ▶ Für größere Zahlen müssen wir Algorithmen wie das schriftliche Addieren bemühen

Aber: Da `int` auch Indizes und Länge von Arrays (allgemein: Speicheradressen) beschreibt, können wir nur Eingaben bis Größe $n \leq 2^w$ verarbeiten!

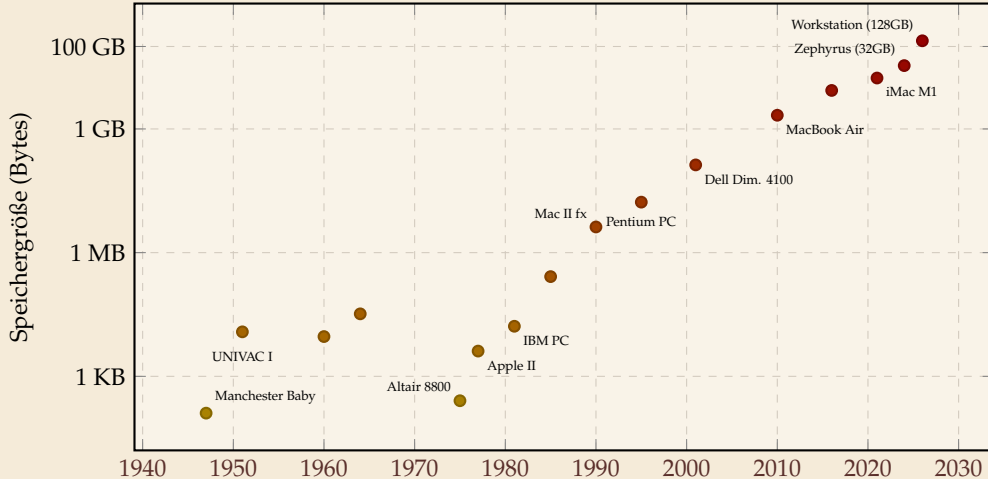
- ▶ Einziger Ausweg: Wortbreite w muss mit n wachsen(!)
- ▶ übliche Annahme: $w = \Theta(\log n)$

↔ Das heißt, unsere Maschine wächst mit den Eingaben mit!? 🤖
Wie ergibt das Sinn?

Arbeitsspeichergröße über die Zeit

Arbeitsspeicher ist gute Näherung für größte Eingabegrößen

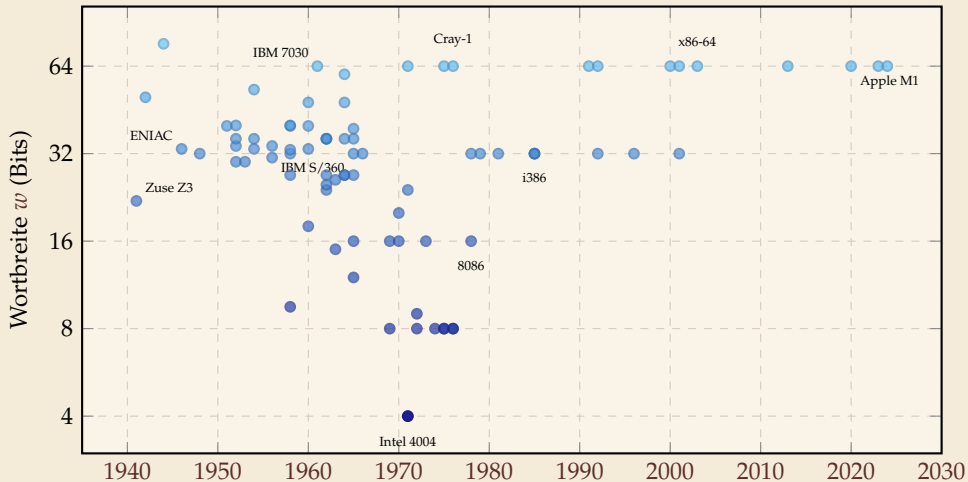
Größe des Arbeitsspeicher (RAM) für typische Systeme



Wortbreite über die Zeit

Entwicklung der Wortbreite in der gleichen Zeit ([https://en.wikipedia.org/wiki/Word_\(computer_architecture\)](https://en.wikipedia.org/wiki/Word_(computer_architecture)))

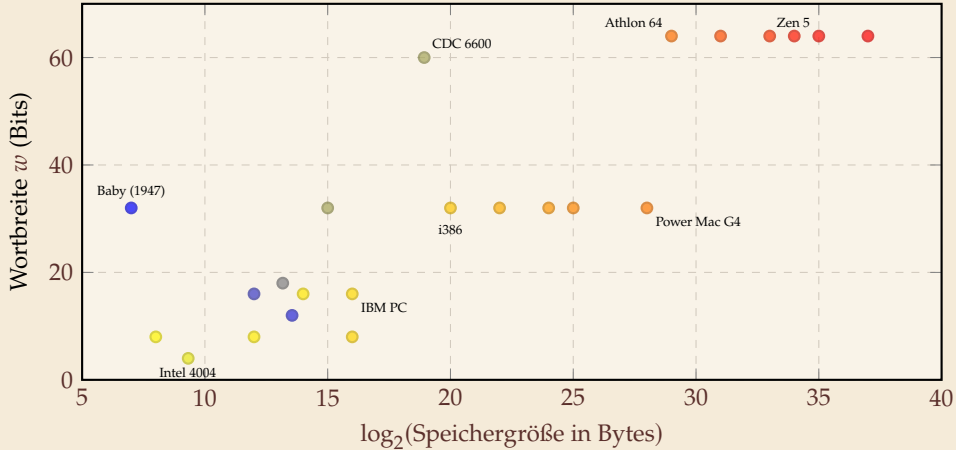
Wortbreite diverser Computersysteme (1941–2026)



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

Das Verhältnis von $\log_2 n$ und w ist recht stabil $\rightsquigarrow w = \Theta(\log n)$ ist plausibel!

Wortbreite vs. Speichergröße diverser Systeme (1947–2026)



Farbe =

Erscheinungsjahr (von blau zu rot)

word-RAM

Falls es auf solche Details ankommt, ist $w = \Theta(\log n)$ die „richtige“ Annahme

- ↪ mit „kleinen“ (polynomiell beschränkten) ganzen Zahlen können wir in Konstantzeit rechnen
mit „großen“ ganzen Zahlen ist die Laufzeit von der Anzahl Ziffern abhängig
- ▶ kann zu einem vollständigen, mathematischen Modell eines einfachen, abstrakten Computers ergänzt werden
- ↪ Standard-Annahme in der Algorithmik-Forschung:
das **word-RAM Modell** (RAM = random access machine, nicht Arbeitsspeicher)

5.5 **Kostenmodelle**

Was kostet ein Algorithmus?

Typischerweise interessiert uns für einen Algorithmus

bzw. Worst-Case-Eingabe der Größe n

- ▶ die **Laufzeit** (auf einem Computer und mit einer Eingabe)
- ▶ der **Speicherbedarf** (zusätzlich zur Eingabe)

Präzise Laufzeit für **mathematische Analyse** und Vorhersage ungeeignet

- ▶ hängt von vielen Details der Hardware und Implementierung ab
- ▶ nicht perfekt reproduzierbar
- ▶ sehr mühselig bis unmöglich genau zu analysieren

↪ Suchen ein **abstraktes Kostenmaß** für einen Algorithmus

- ▶ (nahezu) proportional zur tatsächlichen Laufzeit ↪ erlaubt Prognosen!
- ▶ **zählt** Vorkommen von diskreten Events (z. B. Anzahl Ausführungen einer Code-Zeile)

↪ exakt reproduzierbar, hardwareunabhängig

- ▶ (hoffentlich) leicht(er) zu analysieren

Zusammenfassung

Durch die Formulierung konkreter Modelle wird die Analyse von Algorithmen auf ein solides Fundament gestellt

- ▶ Datenmodelle für die Eingaben
- ▶ Maschinenmodelle für die erlaubten Algorithmen
- ▶ Kostenmodelle für das Maß, das wir optimieren möchten

↪ Voraussetzung für **wissenschaftlichen Vergleich von Algorithmen** (*algorithm science*)