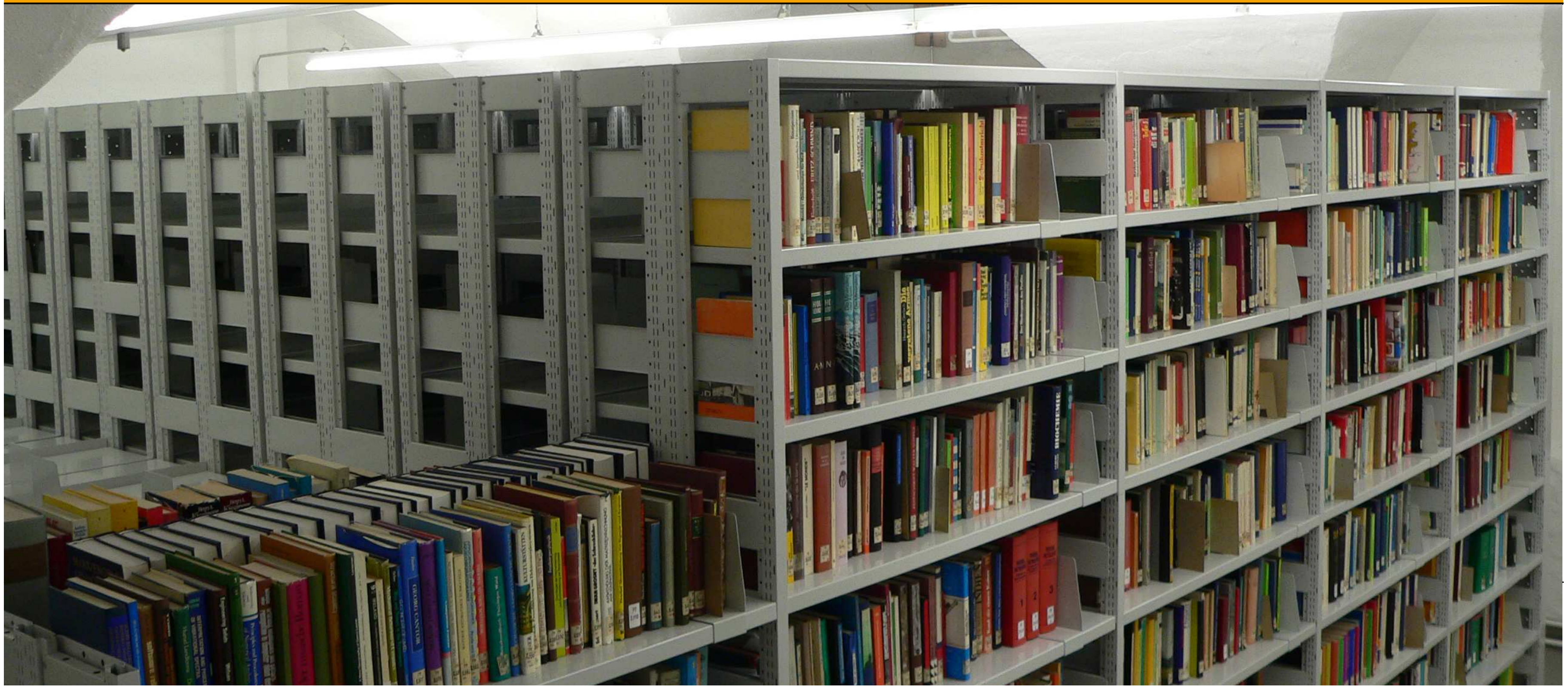
Martin Ziegler

TECHNISCHE UNIVERSITÄT DARMSTADT

Willkommen zur *Komplexitätstheorie*



Formalien

Informatiker willkommen!

TECHNISCHE UNIVERSITÄT DARMSTADT

Komplexitätstheorie

- Bachelor 3. Jahr oder Master Ergänzung
- Vorkenntnisse: Algo.diskr.Mathe, Grundk. Logik
- V2 Mittwochs
- Ü2 Freitags

Ausnahme: am 27.10. (Mittwoch) ist Übung; am 29.10. bin ich auf Dienstreise

- Hausaufgaben-Ausgabe:
  Mittwochnachmittags im Internet
- Einsammeln der *handschriftl.* Lösungen:
  Mittwochs vor der nächsten Vorlesung
- Besprechung der Lösungen:
  2 Tage später (Freitag) in der Übung
- 6 ETCS: Klausur (30min) **+** mündl. (7min)

Lernen Lernen

TECHNISCHE UNIVERSITÄT DARMSTADT
Komplexitätstheorie

Berücksichtigen Sie die Lern-Physiologie!

- Verstehen ≠ Auswendiglernen
- Kurzzeit-, Mittel- und Langzeitgedächtnis
  - Transfer durch Wiederholung und Schlaf
- Bearbeiten Sie die Übungszettel!
  - Selbstkontrolle, Wiederholung, Verständnis
- Verteilen Sie die Bearbeitung auf möglichst viele Tage!

`http://www3.mathematik.tu-darmstadt.de/evs/916`

Lehrbuchsammlung Mathe-Bibliothek: Papadimitrou „*Computational Complexity*“

Erinnerung: Asymptotik

- Landau: Für $f,g : \mathbb{N} \to \mathbb{R}_+$ schreibe
  - $f=O(g) \quad \Leftrightarrow \quad \exists M\ \forall n \geq M\colon f(n) \leq M \cdot g(n)$
  - $f=\Omega(g) \quad \Leftrightarrow \quad \exists M\ \forall n \geq M\colon f(n) \geq g(n)/M$
  - $f=\Theta(g) \quad \Leftrightarrow \quad f=O(g) \wedge f=\Omega(g)$
- Diese Notation vernachlässigt (und vereinfacht so) Terme niederer Ordnung
- z.B. $5 \cdot n^3 - 27 \cdot n^2 + 933 \cdot n + 2197 = \Theta(n^3)$
- weitere Beispiele: Übung
- f wächst polynomiell $\Leftrightarrow \exists k\colon f=O(n^k)$

Asymptotische Laufzeiten

n	$\log_2 n \cdot 10$s	$n \cdot \log n$ sec	n^2 msec	n^3 µsec	2^n nsec
10	33sec	33sec	0.1sec	1msec	1msec
100	≈1min	11min	10sec	1sec	40 Mrd. Y
1000	≈1.5min	≈3h	17min	17min	
10 000	≈2min	1.5 Tage	≈1 Tag	11 Tage	
100 000	≈2.5min	19 Tage	4 Monate	32 Jahre	

- Laufzeit von Algorithmen, z.B. Sortieren
  - BubbleSort: $O(n^2)$ Vergleichs- und Kopier-Op.
  - QuickSort: typisch $O(n \cdot \log n)$ Operationen, aber $O(n^2)$ Operationen im *worst-case*
  - HeapSort: immer $O(n \cdot \log n)$ Operationen
- Hier: stets worst-case Betrachtungen!
  - bzgl. n = Eingabegröße (z.B. Bitlänge) $\to \infty$

Beispiel Matrixmultiplikation

- Eingabe: Zwei $n\times n$-Matrizen A und B,
- Ausgabe: die $n\times n$-Matrix $C := A \cdot B$.
- n^2-mal "*Zeile-mal-Spalte*" á $O(n)$: $O(n^3)$

Multiplikation von $n\times n$-Matrizen mittels

$$\begin{array}{|c|c|}\hline C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \\ \hline \end{array} = \begin{array}{|c|c|}\hline A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \\ \hline \end{array} \cdot \begin{array}{|c|c|}\hline B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \\ \hline \end{array}$$

7 Multiplikationen +18 Additionen von $(n/2)\times(n/2)$-Matrizen

$$L(n) = 7\cdot L(\lceil n/2 \rceil) + 18\cdot(n/2)^2$$

$$L(n) = O(n^{\log_2 7}), \quad \log_2 7 \approx 2{,}8$$

Optimalität und Rechenmodell

- Matrix-Multipl. zählt arithmet. Operationen
  - $2n^2$ Eingaben, n^2 Ausgaben: $\Omega(n^2)$.
- HeapSort: $O(n \cdot \log n)$ Operationen
  - Geht es (asymptotisch) schneller?
  - Ja: mit nur 1 Operation `sort`$(x_1,\ldots,x_n)$
- Komplexität immer bzgl. *Rechenmodell*:
- mathem. Formalisierung+ Idealisierung
  - Welche Operationen werden unterstützt
  - und wie viele Ressourcen (ver-)brauchen sie.
- "Ressourcen": z.B. Laufzeit, Speicherplatz, #Prozessoren (bei Parallelcomputing)

Hier fast ausschließlich: *Turingmaschine*

2.1 Turingmaschine

Alan M. Turing [1937]

- mathematische Idealisierung/ Abstraktion seiner Zuarbeiter (sog. *„computer“*)
- heutzutage gemeinhin akzeptiert als Modell für Digitalrechner (PCs)