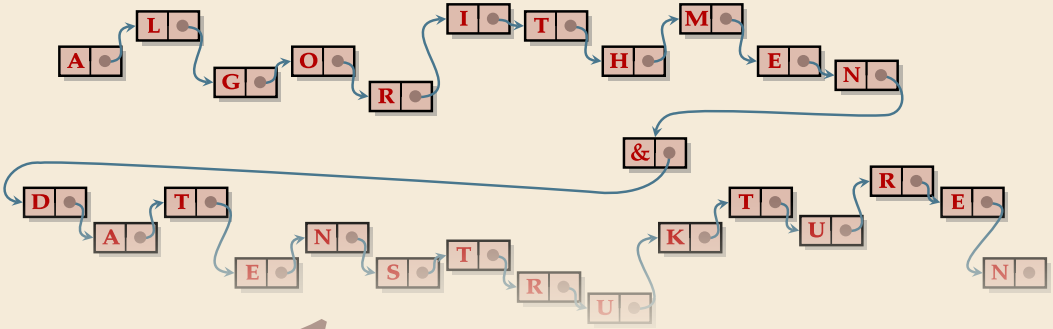


1 Motivation

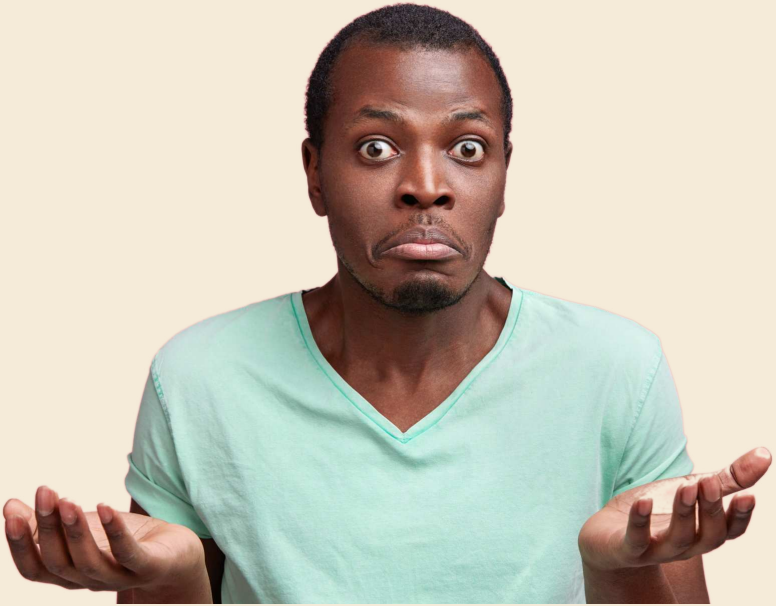
Algorithmen & Datenstrukturen · Sommersemester 2026

Prof. Dr. Sebastian Wild

CS210 (Sommersemester 2026)
Philipps-Universität Marburg
version `2026-04-20 09:22`

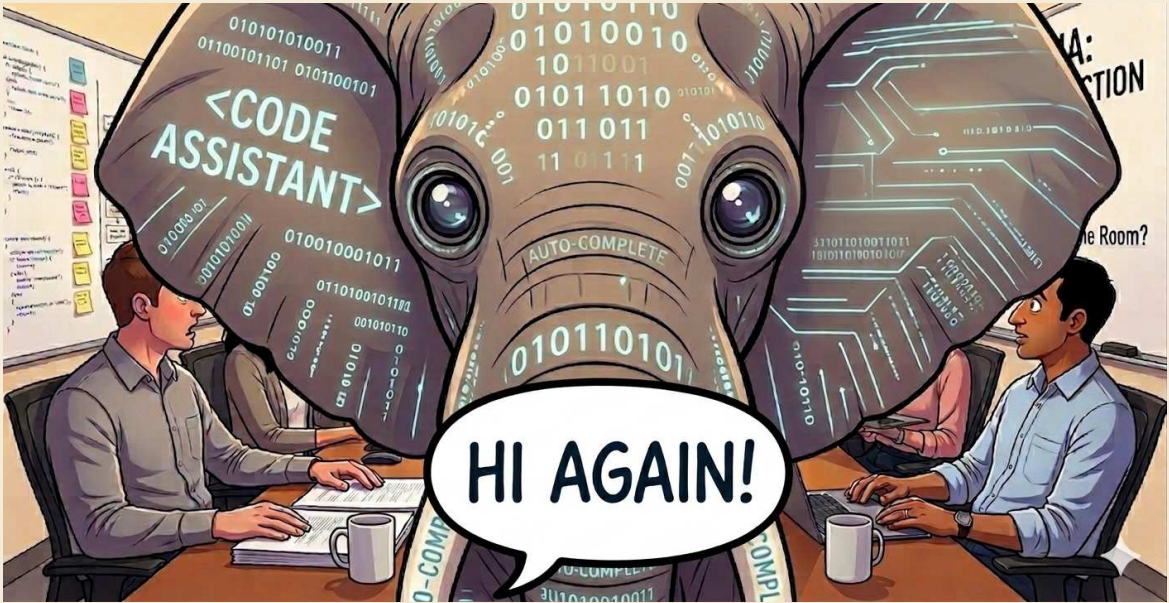
Outline

Wozu Algorithmen und Datenstrukturen?



1.1 Ketzerische Fragen

Kann das nicht die KI machen?



Kann das nicht die KI machen?

- Ja und Nein.
- Mit präzisen und passenden Anweisungen kann KI Code generieren.

⇝ Genau hier profitiert man aber davon, die richtigen Abstraktionen zu verwenden!
Dafür muss man wissen, wovon man redet.

Kann das nicht die KI machen?

- Ja und Nein.
- Mit präzisen und passenden Anweisungen kann KI Code generieren.

⇝ Genau hier profitiert man aber davon, die richtigen Abstraktionen zu verwenden!
Dafür muss man wissen, wovon man redet.

- KI liefert die häufigste Lösung (aus Training), die ungefähr passt.
  Nur weil ein Ansatz in Medium-Blogs am häufigsten vorkommen ist er nicht unbedingt korrekt oder effizient!

⇝ **Für die Unterscheidung brauchen Sie das ADS-Wissen!**

- Was tun bei *Performance Leaks*?

Nicht nur mein Ratschlag

> *"**Master the fundamentals first.** This is critical, because you can't evaluate code quality without understanding what good architecture looks like. You can't write effective tests without knowing what can break. You can't catch hallucinations without domain expertise.*
>
> *It's important for experienced developers to keep these skills from atrophying, and **junior developers to build them in the first place**."*
>
> *Eira May, Mind the gap: Closing the AI trust gap for developers,* `stackoverflow.blog/2026/02/18/closing-the-developer-ai-trust-gap`

Nicht nur mein Ratschlag

> "***Master the fundamentals first.*** *This is critical, because you can't evaluate code quality without understanding what good architecture looks like. You can't write effective tests without knowing what can break. You can't catch hallucinations without domain expertise.*
>
> *It's important for experienced developers to keep these skills from atrophying, and* ***junior developers to build them in the first place****."*
>
> *Eira May, Mind the gap: Closing the AI trust gap for developers,* `stackoverflow.blog/2026/02/18/closing-the-developer-ai-trust-gap`

> "***What you want to get to, particularly as your career evolves, is mastery****. That's how you kind of escape the thing that everybody can do and get more differentiation. The concern I have is this culture of, 'Well, I'm not even going to try to understand what's going on. I'm just going to spend some tokens, and maybe it'll be great.'*
> *[. . .]*
> *Getting us out of writing boilerplate, getting us out of memorizing APIs, getting us out of looking up that thing from Stack Overflow; I think this is really profound. This is a good use. The thing that I get* ***concerned*** *about is if you go so far as to not care about what you're looking up on Stack Overflow and why it works that way and* ***not learning from it****."*
>
> *Chris Lattner,* `fast.ai/posts/2025-10-30-build-to-last.html`

Kann ich nicht einfach Unit Testen und gut?

- (nicht für sicherheitsrelevante Software)
- Unit Tests prüfen **Korrektheit**
- Skalierbarkeit kann man so nicht testen!

Von 99% zu 100%

- Zwischen “fast komplett richtig” und “100% korrekt” liegen oft Welten

Von 99% zu 100%

- Zwischen “fast komplett richtig” und “100% korrekt” liegen oft Welten
- ***Hierarchische Abstraktion*** und Blackbox-Reuse sind omnipräsent in der Informatik
  - “algorithmic thinking”
  - Problemlösestrategien

Von 99% zu 100%

- Zwischen “fast komplett richtig” und “100% korrekt” liegen oft Welten
- ***Hierarchische Abstraktion*** und Blackbox-Reuse sind omnipräsent in der Informatik
  - “algorithmic thinking”
  - Problemlösestrategien

  unsere einzige erwiesenermaßen effektive Waffe für wachsende Komplexität von Systemen

⇝ kleinste Fehler in der Realisierung propagieren sich über viele Abstraktionsschichten hinweg zu Systemversagen

- *Blackbox-Reuse braucht pedantische Korrektheit*

1.2 Effizienz in Anwendungen

Skalierbarkeit

Analogie: Architektur

- 1m Brücke: Holzplanke genügt!
- 10x $\rightsquigarrow$ Konstruktion nötig
- 1000x $\rightsquigarrow$ Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.

Skalierbarkeit

Analogie: Architektur

- 1m Brücke: Holzplanke genügt!
- 10x $\rightsquigarrow$ Konstruktion nötig
- 1000x $\rightsquigarrow$ Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.

Skalierbarkeit

Analogie: Architektur

- 1m Brücke: Holzplanke genügt!
- 10x $\rightsquigarrow$ Konstruktion nötig
- 1000x $\rightsquigarrow$ Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.

Skalierbarkeit

Analogie: Architektur

- 1m Brücke: Holzplanke genügt!
- 10x $\rightsquigarrow$ Konstruktion nötig
- 1000x $\rightsquigarrow$ Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.




2m	~10x →	12m	~10x →	170m	~10x →	1280m
€ 50 000	~10x →	€ 350 000	~50x →	€ 22 000 000	~200x →	€ 4 500 000 000

(Kosten sind Schätzungen der heutigen Kosten)

Skalierbarkeit

Analogie: Architektur

- 1m Brücke: Holzplanke genügt!
- 10x ⇝ Konstruktion nötig
- 1000x ⇝ Meisterleistung der Bauingenieurskunst

Informatik

- Eingabegrößen in Informatiksystemen erreichen oft Faktor 1 000 000 000!

  1 Byte ⟶ 1 GB

⇝ *Können uns keine Holzbretter leisten.*

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.




2m	~10x ⟶	12m	~10x ⟶	170m	~10x ⟶	1280m
€ 50 000	~10x ⟶	€ 350 000	~50x ⟶	€ 22 000 000	~200x ⟶	€ 4 500 000 000

(Kosten sind Schätzungen der heutigen Kosten)

Gibt's das nicht als fertige Library?

- klassische Datenstrukturen (wie die in ADS!) gibt es in der Regel als Software Library

⇝ nicht das Rad neu erfinden, wenn es nicht nötig ist!

- **Aber:** Diese Bibliotheken rein als ***"magic black boxes"*** zu begreifen ist unzureichend.
  - müssen Performancecharakteristika auf Anwendung anpassen
  - Fehlersuche bei "Performance leaks"
  - Erweiterungen

Gibt's das nicht als fertige Library?

- klassische Datenstrukturen (wie die in ADS!) gibt es in der Regel als Software Library

⇝ nicht das Rad neu erfinden, wenn es nicht nötig ist!

- **Aber:** Diese Bibliotheken rein als ***"magic black boxes"*** zu begreifen ist unzureichend.
  - müssen Performancecharakteristika auf Anwendung anpassen
  - Fehlersuche bei "Performance leaks"
  - Erweiterungen

Wissen um Performancecharakteristika ist Ihr *Bullsh*t Detector*

Gibt's das nicht als fertige Library?

- klassische Datenstrukturen (wie die in ADS!) gibt es in der Regel als Software Library

⇝ nicht das Rad neu erfinden, wenn es nicht nötig ist!

- **Aber:** Diese Bibliotheken rein als ***"magic black boxes"*** zu begreifen ist unzureichend.
  - müssen Performancecharakteristika auf Anwendung anpassen
  - Fehlersuche bei "Performance leaks"
  - Erweiterungen

Wissen um Performancecharakteristika ist Ihr *Bullsh*t Detector*

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important.
Bad programmers worry about the code.
***Good programmers worry about data structures and their relationships.**"*

Linus Torvalds

„Effizienz“ ist kontextabhängig

1.3 Abstraktion

Power of Abstraction

Sie erinnern sich an eben:

- ***Hierarchische Abstraktion*** und Blackbox-Reuse sind omnipräsent in der Informatik
  - "algorithmic thinking"
  - Problemlösestrategien

  unsere einzige erwiesenermaßen effektive Waffe für wachsende Komplexität von Systemen

Power of Abstraction

Sie erinnern sich an eben:

- ***Hierarchische Abstraktion*** und Blackbox-Reuse sind omnipräsent in der Informatik
  - "algorithmic thinking"
  - Problemlösestrategien

  unsere einzige erwiesenermaßen effektive Waffe für wachsende Komplexität von Systemen

- typische Beispiele: Dateisysteme, TCP-Protokoll, String-Funktionen in Java, ...
  - Oft sind diese Abstraktionen gerade Algorithmen und Datenstrukturen!

char[]

UTF-16

Power of Abstraction

Sie erinnern sich an eben:

- ***Hierarchische Abstraktion*** und Blackbox-Reuse sind omnipräsent in der Informatik
  - "algorithmic thinking"
  - Problemlösestrategien

  unsere einzige erwiesenermaßen effektive Waffe für wachsende Komplexität von Systemen

- typische Beispiele: Dateisysteme, TCP-Protokoll, String-Funktionen in Java, . . .
  - Oft sind diese Abstraktionen gerade Algorithmen und Datenstrukturen!

- Algorithmen sind der ultimative **wiederverwendbare Code**
  - quasi per Definition ein wiederkehrendes Problem, das wir ein für alle mal lösen

- Abstrakte Datentype (ADTs) sind die ultimative Form von **semantischer Kapselung** (*encapsulation and information hiding*)
  - Datenstrukturen als austauschbare Implementierungen entkoppeln Realisierung von Verwendung

Unter die Motorhaube

- Wenn die Abstraktion perfekt passt, reicht die Library-Implementierung
- aber Achtung: früher oder später muss man in die Blackbox reinschauen

"All non-trivial abstractions, to some degree, are leaky."

Leaky abstractions

Joel Spolsky, joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions

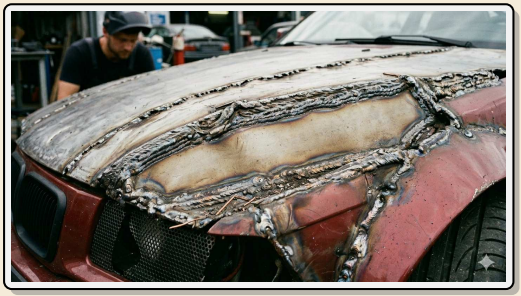
Unter die Motorhaube

- Wenn die Abstraktion perfekt passt, reicht die Library-Implementierung
- aber Achtung: früher oder später muss man in die Blackbox reinschauen

> *"All non-trivial abstractions, to some degree, are leaky."*
>
> *Joel Spolsky, joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions*

- "Autofahrer oder Mechaniker"

⇝ keine Autos mit **zugeschweißter Motorhaube** bauen



Ratschlag aus dem Jahre 2002

"The law of leaky abstractions means that whenever somebody comes up with a ***wizzy new code-generation tool*** *that is supposed to make us all ever-so-efficient, you hear a lot of people saying 'learn how to do it manually first, then use the wizzy tool to save time.'* ***Code generation tools which pretend to abstract out something, like all abstractions, leak, and the only way to deal with the leaks competently is to learn about how the abstractions work and what they are abstracting.***

So the ***abstractions save us time working, but they don't save us time learning.***

And all this means that paradoxically, even as we have higher and higher level programming tools with better and better abstractions, becoming a proficient programmer is getting harder and harder."

Joel Spolsky, `joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions`

1.4 Die Sprache der Experten

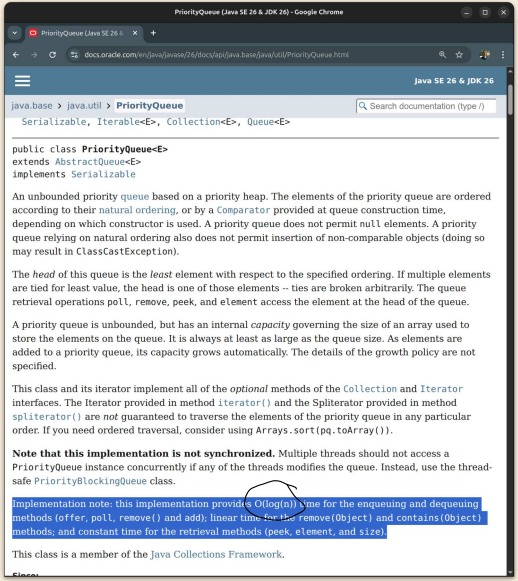
Abstraktion wird Vokabel

- Unter Experten ersetzen einzelne Begriffe ganze komplexe Themenfelder

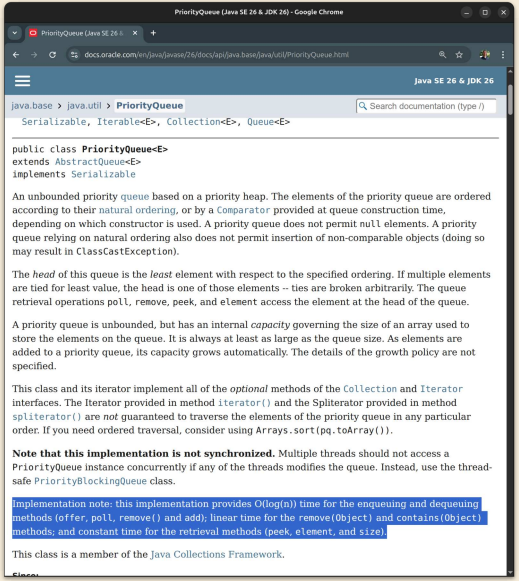
⇝ wiederverwertbares *Verständnis*

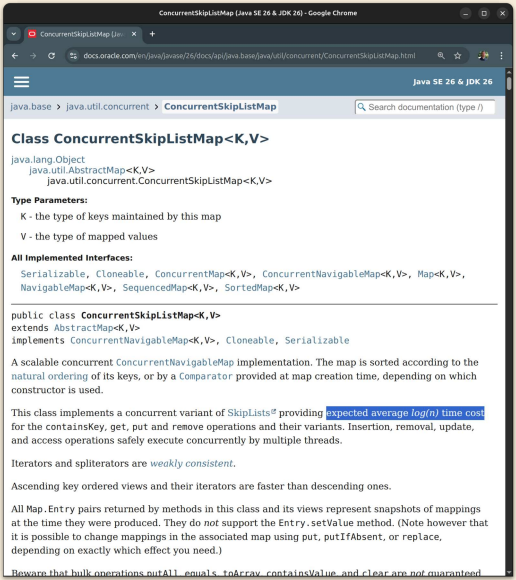
- Es wird erwartet, dass Sie damit zurechtkommen!

Beispiele aus der Java API



Beispiele aus der Java API





Gatekeeper: Coding Interviews

Immer noch nicht überzeugt, dass ADS für Sie wertvoll ist?

Gatekeeper: Coding Interviews

Immer noch nicht überzeugt, dass ADS für Sie wertvoll ist?

Für die meisten Karrieren in der Tech Industrie sind *technische Interviews* Standard.

- Fragen zu Algorithmen und Datenstrukturen (sowie ihrer Implementierung) sind die häufigsten Themen
- Standard-Kanon an Universitäten
- gut testbar
- auch allen Interviewern bekannt!

xkcd.com/2483/

Zusammenfassung

Ziele von ADS:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - *best practices:* Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
   - Anpassungen und Erweiterungen ermöglichen

Zusammenfassung

Ziele von ADS:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - *best practices:* Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
   - Anpassungen und Erweiterungen ermöglichen
2. Algorithmen & Datenstrukturen **bewerten** können

Zusammenfassung

Ziele von ADS:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - *best practices:* Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
   - Anpassungen und Erweiterungen ermöglichen
2. Algorithmen & Datenstrukturen **bewerten** können
3. mit anderen Experten über Algorithmen & Datenstrukturen **reden** können
   - Vokabular aufbauen ($\rightsquigarrow$ Baukasten)
   - präzise Beschreibungen (von Performance) verstehen und erstellen
   - erkennen, wenn jemand „Mist“ erzählt