

# Algorithmen & Datenstrukturen

Sebastian Wild  
wild@cs.uni-kl.de



*Fachbereich Informatik*

05. Januar 2017

0

# Organisatorisches

News

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)

News

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen

News


- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT

News




- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!


# News


- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 



News



- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.
  - danach in diesem Slot: Übungsstunde

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.
  - danach in diesem Slot: Übungsstunde
- voraussichtlich kein Donnerstagstermin für Vorlesungen (außer heute)

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.
  - danach in diesem Slot: Übungsstunde
- voraussichtlich kein Donnerstagstermin für Vorlesungen (außer heute)
- jetzt offiziell: 



# Überblick über den Kurs

**START**

**1 – Motivation**

**2 – Maschinen und Modelle**

**3 – Analyse von Algorithmen**

**4 – Elementare Datenstrukturen**

**5 – Sortieren**

**6 – Binäre Suchbäume**

**7 – Hashing**

**8 – Graph Basics**

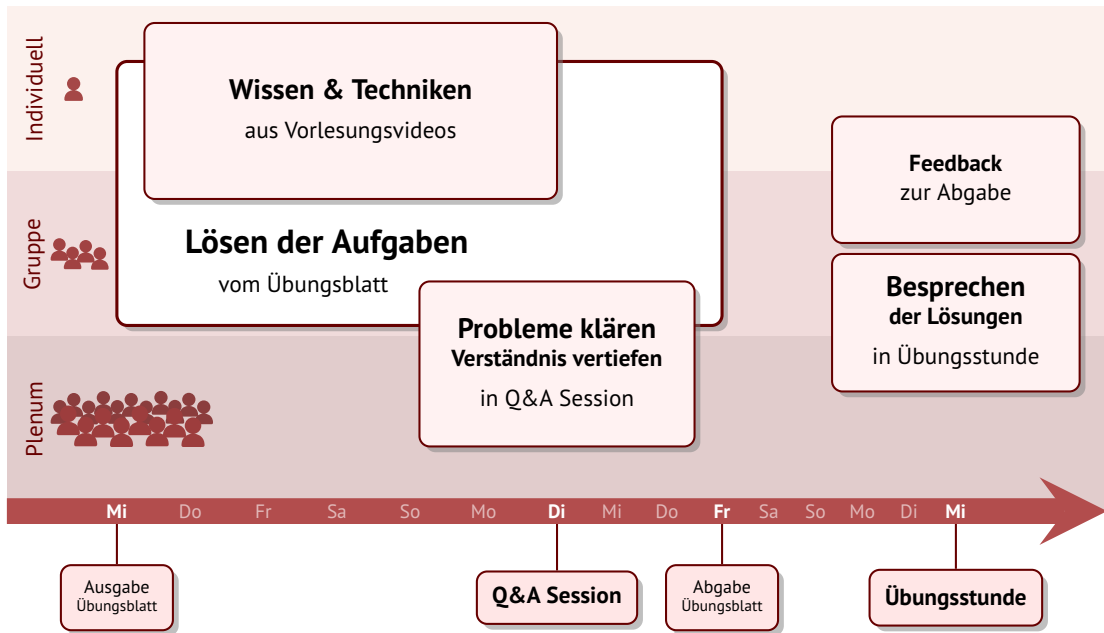
**9 – Shortest Paths**

**10 – Dynamic Programming**

**11 – Computability & Intractability**

Images from <https://www.flickr.com/photos/drywall/747943436/>, pixabay, [http://www.palmpedia.net/wiki/Hyphaene\\_compressa](http://www.palmpedia.net/wiki/Hyphaene_compressa)

# Ablaufplan





## • Aufgaben auf Übungsblättern

Problem-zentriertes Arbeiten (→ deshalb Übungsblatt ganz am Anfang),  
Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe

- **Aufgaben auf Übungsblättern**

Problem-zentriertes Arbeiten (→ deshalb Übungsblatt ganz am Anfang),  
Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe

- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch

Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)

- **Aufgaben auf Übungsblättern**

Problem-zentriertes Arbeiten (→ deshalb Übungsblatt ganz am Anfang),  
Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe

- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch

Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)

- **„Aufgaben zur Vorbereitung“** zu Videos

**Mindestaktivierung** während des Zuhörens

„das solltet ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich

⚡ Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)  
(sonst ist die Zeit vertan)

- **Aufgaben auf Übungsblättern**

Problem-zentriertes Arbeiten (→ deshalb Übungsblatt ganz am Anfang),  
Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe

- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch

Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)

- **„Aufgaben zur Vorbereitung“** zu Videos

**Mindestaktivierung** während des Zuhörens

„das solltet ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich

⚡ Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)  
(sonst ist die Zeit vertan)

- **Kollaborative Fragensammlung** (Piratenpad)

Unklarheiten und **weiterführende Fragen** sammeln, direkt beim Schauen

**Feedback** zu Fragen und Antworten **von Mitstudis** (geben und bekommen!)

Ranking der Fragen für Q&A Session

## • **Aufgaben auf Übungsblättern**

Problem-zentriertes Arbeiten (→ deshalb Übungsblatt ganz am Anfang),  
Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe

## • **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch

Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)

## • **„Aufgaben zur Vorbereitung“** zu Videos

**Mindestaktivierung** während des Zuhörens

„das solltet ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich

⚡ Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)  
(sonst ist die Zeit vertan)

## • **Kollaborative Fragensammlung** (Piratenpad)

Unklarheiten und **weiterführende Fragen** sammeln, direkt beim Schauen

**Feedback** zu Fragen und Antworten **von Mitstudis** (geben und bekommen!)

Ranking der Fragen für Q&A Session

*Evtl. fortführen während Q&A Session als Gedächtnisstütze / grobes Protokoll?*

## • Aufgaben auf Übungsblättern

Problem-zentriertes Arbeiten (→ deshalb Übungsblatt ganz am Anfang),  
Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe

## • Videos (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch

Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)

## • „Aufgaben zur Vorbereitung“ zu Videos

**Mindestaktivierung** während des Zuhörens

„das solltet ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich

📌 Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)  
(sonst ist die Zeit vertan)

## • Kollaborative Fragensammlung (Piratenpad)

Unklarheiten und **weiterführende Fragen** sammeln, direkt beim Schauen

**Feedback** zu Fragen und Antworten **von Mitstudis** (geben und bekommen!)

Ranking der Fragen für Q&A Session

*Evtl. fortführen während Q&A Session als Gedächtnisstütze / grobes Protokoll?*

## • Q&A Sessions

**Klären** von schwierigen Fragen, **Vertiefen** des Verständnisses

nach den Videos, während Bearbeitung der Übungsaufgaben → ihr seid „im Stoff“



## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

$\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

$\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

$\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

$\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.



## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

$\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

$\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.

$\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.  
 $\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Vorlesungsmodus

**Ziel:** neuen Stoff lernen  
 $\rightsquigarrow$  konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,  
Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...

## Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiiere!**  
Ich habe (etwas) Verständnis für Faulheit,  
aber keins für Ideenklau!

## Q&A-Modus

**Ziel:** Verständnis vertiefen  $\rightsquigarrow$  Ihr seid **aktiv**.  
 $\neq$  alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$  hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet  $\rightsquigarrow$  keine Scheu!
- Wir sind alle erwachsen  
 $\rightsquigarrow$  keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Vorlesungsmodus

**Ziel:** neuen Stoff lernen  
 $\rightsquigarrow$  konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,  
Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...

## Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiiere!**  
Ich habe (etwas) Verständnis für Faulheit,  
aber keins für Ideenklau!

## Q&A-Modus

**Ziel:** Verständnis vertiefen ~> Ihr seid **aktiv**.  
≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

~> hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet ~> keine Scheu!
- Wir sind alle erwachsen  
~> keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Vorlesungsmodus

**Ziel:** neuen Stoff lernen  
~> konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,  
Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...



## Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiiere!**  
Ich habe (etwas) Verständnis für Faulheit,  
aber keins für Ideenklau!

## Q&A-Modus

**Ziel:** Verständnis vertiefen ~> Ihr seid **aktiv**.  
≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.  
inklusive „Aufgaben zur Vorbereitung“!

~> hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet ~> keine Scheu!
- Wir sind alle erwachsen  
~> keine Anwesenheitspflicht  
(auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

## Vorlesungsmodus

**Ziel:** neuen Stoff lernen  
~> konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,  
Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...



## Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiiere!**  
Ich habe (etwas) Verständnis für Faulheit,  
aber keins für Ideenklau!

1

START







## Why study algorithms?

---

Their impact is broad and far-reaching.

**Internet.** Web search, packet routing, distributed file sharing, ...

**Biology.** Human genome project, protein folding, ...

**Computers.** Circuit layout, file system, compilers, ...

**Computer graphics.** Movies, video games, virtual reality, ...

**Security.** Cell phones, e-commerce, voting machines, ...

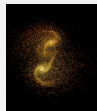
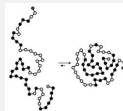
**Multimedia.** MP3, JPG, DivX, HDTV, face recognition, ...

**Social networks.** Recommendations, news feeds, advertisements, ...

**Physics.** N-body simulation, particle collision simulation, ...

⋮

Google  
YAHOO!  
bing



3

<http://algs4.cs.princeton.edu/lectures/00Intro.pdf>, page 3

## Why study algorithms?

Their impact is broad and far-reaching.

### Mysterious algorithm was 4% of trading activity last week

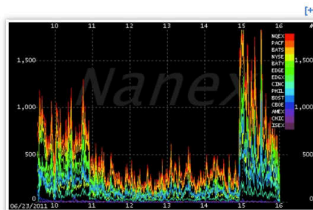
October 11, 2012

A single mysterious computer program that placed orders — and then subsequently canceled them — made up 4 percent of all quote traffic in the U.S. stock market last week, according to the top tracker of [high-frequency trading](#) activity.

The motive of the algorithm is still unclear, [CNBC](#) reports.

The program placed orders in 25-millisecond bursts involving about 500 stocks, according to Nanex, a market data firm. The algorithm never executed a single trade, and it abruptly ended at about 10:30 a.m. ET Friday.

"My guess is that the algo was testing the market, as high-frequency frequently does," says Jon Najarian, co-founder of TradeMonster.com. "As soon as they add bandwidth, the HFT crowd sees how quickly they can top out to create latency." ([Read More: Unclear What Caused Kraft Spike: Nanex Founder.](#))



Generic high frequency trading chart (credit: Nanex)

## Why study algorithms?

---

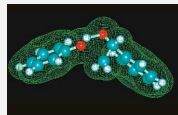
They may unlock the secrets of life and of the universe.

*“ Computer models mirroring real life have become crucial for most advances made in chemistry today.... Today the computer is just as important a tool for chemists as the test tube. ”*

— *Royal Swedish Academy of Sciences  
(Nobel Prize in Chemistry 2013)*



Martin Karplus, Michael Levitt, and Arieh Warshel



## Why study algorithms?

For fun and profit.



## Why study algorithms?

---

- Their impact is broad and far-reaching.
- Old roots, new opportunities.
- For intellectual stimulation.
- To become a proficient programmer.
- They may unlock the secrets of life and of the universe.
- To solve problems that could not otherwise be addressed.
- Everybody else is doing it.
- For fun and profit.

Why study anything else?



## Ziele dieser Vorlesung:

- 1 **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
  - **best practices**: Liste von A. & DS. für Standardprobleme
  - intellektuelle Errungenschaften der Informatik
- 2 Algorithmen **bewerten** können
- 3 mit anderen Experten über Algorithmen **reden**
  - Computational Thinking prägt unsere Welt.
  - Vokabular aufbauen (→ Baukasten)
  - präzise Beschreibungen verstehen und erstellen, z.B. Javadoc der Java API (siehe nächste Folie)
  - erkennen, wenn jemand „Mist“ erzählt

## Ziele dieser Vorlesung:

- 1 **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
  - **best practices:** Liste von A. & DS. für Standardprobleme
  - intellektuelle Errungenschaften der Informatik
- 2 Algorithmen **bewerten** können
- 3 mit anderen Experten über Algorithmen **reden**
  - Computational Thinking prägt unsere Welt.
  - Vokabular aufbauen (→ Baukasten)
  - präzise Beschreibungen verstehen und erstellen, z.B. Javadoc der Java API (siehe nächste Folie)
  - erkennen, wenn jemand „Mist“ erzählt



## Ziele dieser Vorlesung:

- 1 **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
  - **best practices**: Liste von A. & DS. für Standardprobleme
  - intellektuelle Errungenschaften der Informatik
- 2 Algorithmen **bewerten** können
- 3 mit anderen Experten über Algorithmen **reden**
  - Computational Thinking prägt unsere Welt.
  - Vokabular aufbauen (→ Baukasten)
  - präzise Beschreibungen verstehen und erstellen, z.B. Javadoc der Java API (siehe nächste Folie)
  - erkennen, wenn jemand „Mist“ erzählt

## Ziele dieser Vorlesung:

- 1 **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
  - **best practices**: Liste von A. & DS. für Standardprobleme
  - intellektuelle Errungenschaften der Informatik
- 2 Algorithmen **bewerten** können
- 3 mit anderen Experten über Algorithmen **reden**
  - Computational Thinking prägt unsere Welt.
  - Vokabular aufbauen (↔ Baukasten)
  - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
  - erkennen, wenn jemand „Mist“ erzählt

## Ziele dieser Vorlesung:

- 1 **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
  - **best practices:** Liste von A. & DS. für Standardprobleme
  - intellektuelle Errungenschaften der Informatik
- 2 Algorithmen **bewerten** können
- 3 mit anderen Experten über Algorithmen **reden**
  - Computational Thinking prägt unsere Welt.
  - Vokabular aufbauen (→ Baukasten)
  - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
  - erkennen, wenn jemand „Mist“ erzählt



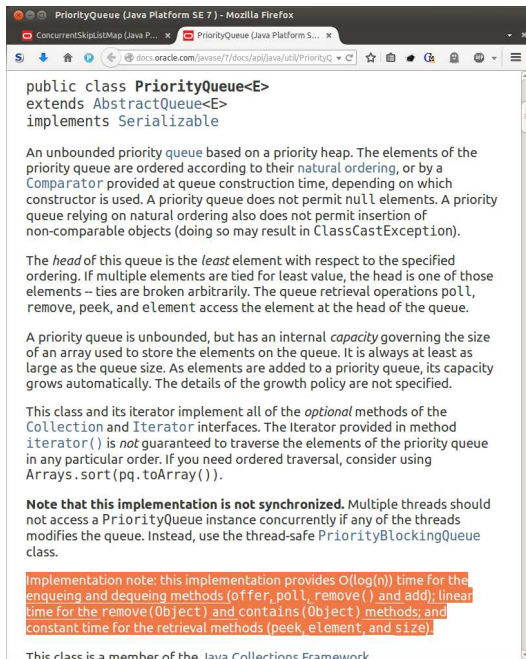
## Ziele dieser Vorlesung:

- 1 **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
  - **best practices:** Liste von A. & DS. für Standardprobleme
  - intellektuelle Errungenschaften der Informatik
- 2 Algorithmen **bewerten** können
- 3 mit anderen Experten über Algorithmen **reden**
  - Computational Thinking prägt unsere Welt.
  - Vokabular aufbauen (↪ Baukasten)
  - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
  - erkennen, wenn jemand „Mist“ erzählt



10/13

# Beispiele aus der Java API



PriorityQueue (Java Platform SE 7) - Mozilla Firefox

```
public class PriorityQueue<E>  
    extends AbstractQueue<E>  
    implements Serializable
```

An unbounded priority queue based on a priority heap. The elements of the priority queue are ordered according to their natural ordering, or by a `Comparator` provided at queue construction time, depending on which constructor is used. A priority queue does not permit null elements. A priority queue relying on natural ordering also does not permit insertion of non-comparable objects (doing so may result in `ClassCastException`).

The *head* of this queue is the *least* element with respect to the specified ordering. If multiple elements are tied for least value, the head is one of those elements – ties are broken arbitrarily. The queue retrieval operations `poll`, `remove`, `peek`, and `element` access the element at the head of the queue.

A priority queue is unbounded, but has an internal *capacity* governing the size of an array used to store the elements on the queue. It is always at least as large as the queue size. As elements are added to a priority queue, its capacity grows automatically. The details of the growth policy are not specified.

This class and its iterator implement all of the *optional* methods of the `Collection` and `Iterator` interfaces. The Iterator provided in method `iterator()` is *not* guaranteed to traverse the elements of the priority queue in any particular order. If you need ordered traversal, consider using `Arrays.sort(pq.toArray())`.

**Note that this implementation is not synchronized.** Multiple threads should not access a `PriorityQueue` instance concurrently if any of the threads modifies the queue. Instead, use the thread-safe `PriorityBlockingQueue` class.

Implementation note: this implementation provides  $O(\log(n))$  time for the enqueueing and dequeuing methods (`offer`, `poll`, `remove()` and `add`); linear time for the `remove(Object)` and `contains(Object)` methods; and constant time for the retrieval methods (`peek`, `element`, and `size`).

This class is a member of the Java Collections Framework



ConcurrentSkipListMap (Java Platform SE 7) - Mozilla Firefox

```
java.util.concurrent  
Class ConcurrentSkipListMap<K,V>  
  
java.lang.Object  
    java.util.AbstractMap<K,V>  
        java.util.concurrent.ConcurrentSkipListMap<K,V>
```

**Type Parameters:**

- K - the type of keys maintained by this map
- V - the type of mapped values

**All Implemented Interfaces:**

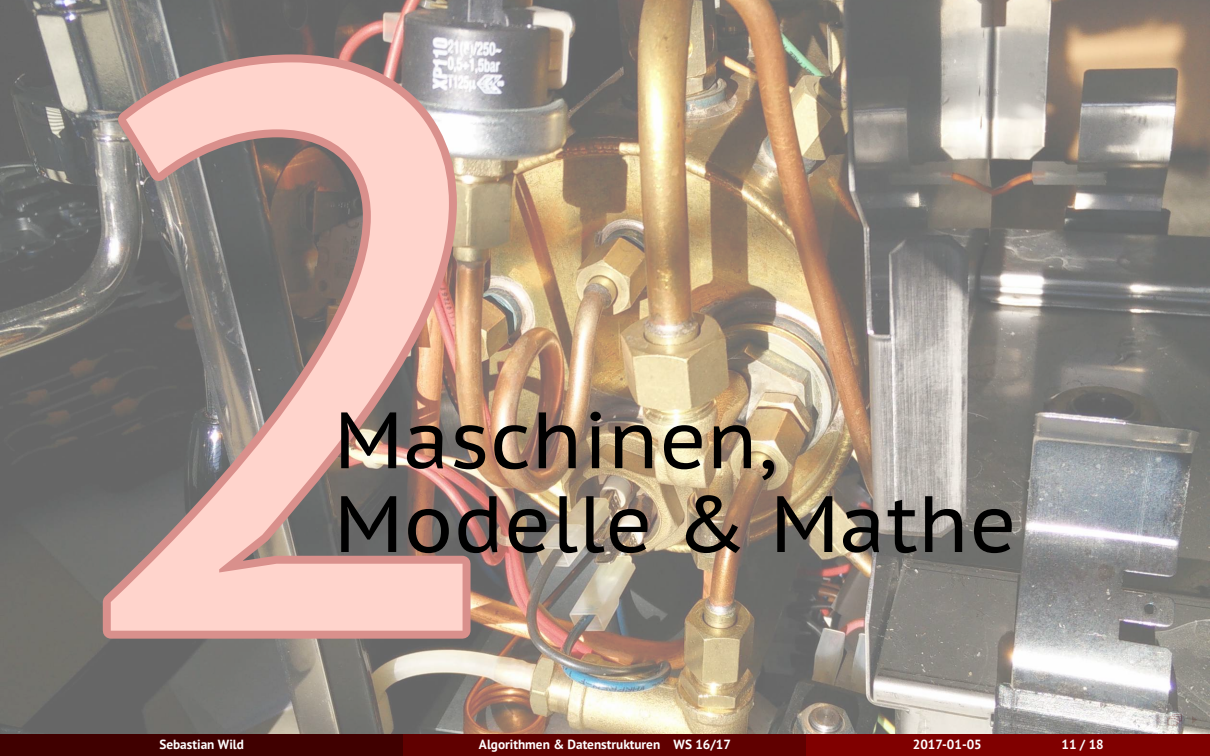
- Serializable, Cloneable, ConcurrentMap<K,V>, ConcurrentNavigableMap<K,V>, Map<K,V>, NavigableMap<K,V>, SortedMap<K,V>

---

```
public class ConcurrentSkipListMap<K,V>  
    extends AbstractMap<K,V>  
    implements ConcurrentNavigableMap<K,V>, Cloneable, Serializ
```

A scalable concurrent `ConcurrentNavigableMap` implementation. The map is sorted according to the natural ordering of its keys, or by a `Comparator` provided at map creation time, depending on which constructor is used.

This class implements a concurrent variant of `SkipLists` providing expected average  $\log(n)$  time cost for the `containsKey`, `get`, `put` and `remove` operations and their variants. Insertion, removal, update, and access operations safely execute concurrently by multiple threads. Iterators are *weakly consistent*, returning elements reflecting the state of the map at some point at or since the creation of the iterator. They do *not* throw `ConcurrentModificationException`, and may proceed concurrently with other operations. Ascending key ordered views and their iterators are faster than descending ones.



# 2

## Maschinen, Modelle & Mathe

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

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

Weniger ...



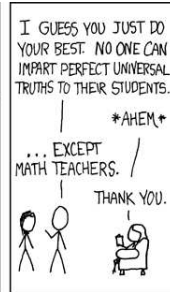
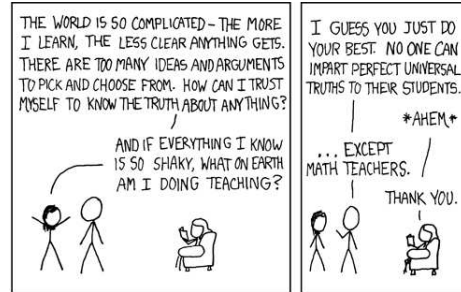


A&DS ist Teil eines **wissenschaftlichen** Studiengangs

Weniger ...

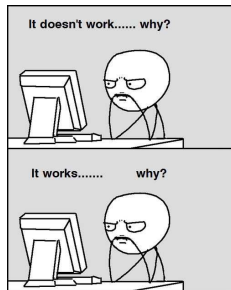


...und mehr

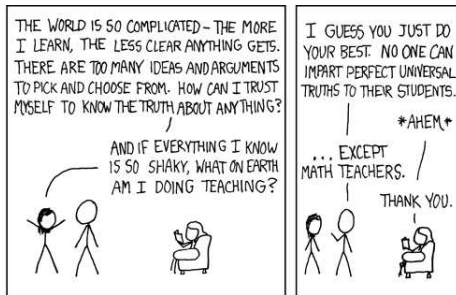


A&DS ist Teil eines **wissenschaftlichen** Studiengangs

Weniger ...



...und mehr



↪ Fokus auf “universal truths” der Algorithmik

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

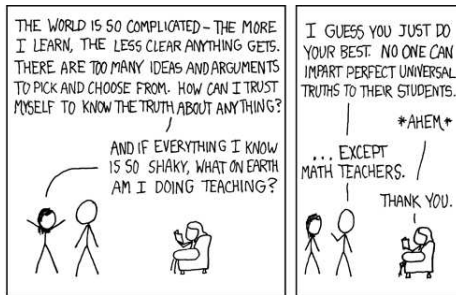
↪ Wir brauchen ein paar mathematische / theoretische Grundlagen.

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

↪ Wir brauchen ein paar mathematische / theoretische Grundlagen.

# Was ist ein Algorithmus?





**Welche Definitionen für „Algorithmus“ gaben die Videos?**

# Was ist ein Algorithmus?



**Anschaulich:**  
Folge von Anweisungen  $\approx$  Kochrezept



# Was ist ein Algorithmus?



## Anschaulich:

Folge von Anweisungen  $\approx$  Kochrezept

## Genauer:

- 1 mechanisch nachvollziehbar  
~ kein „gesunder Menschenverstand“ nötig
- 2 endliche Beschreibung
- 3 löst eine Klasse von Problemen







## Anschaulich:

Folge von Anweisungen  $\approx$  Kochrezept

## Genauer:

- 1 mechanisch nachvollziehbar  
     $\rightsquigarrow$  kein „gesunder Menschenverstand“ nötig
- 2 endliche Beschreibung
- 3 löst eine Klasse von Problemen





# Was ist ein Algorithmus?



## Anschaulich:

Folge von Anweisungen  $\approx$  Kochrezept

## Genauer:

- 1 mechanisch nachvollziehbar  
     $\rightsquigarrow$  kein „gesunder Menschenverstand“ nötig
- 2 endliche Beschreibung
- 3 löst eine Klasse von Problemen

# Was ist ein Algorithmus?



## Anschaulich:

Folge von Anweisungen  $\approx$  Kochrezept

## Genauer:

- 1 mechanisch nachvollziehbar  
     $\rightsquigarrow$  kein „gesunder Menschenverstand“ nötig
- 2 endliche Beschreibung  $\neq$  endliche Berechnung!
- 3 löst eine Klasse von Problemen



# Was ist ein Algorithmus?



## **Anschaulich:**

Folge von Anweisungen  $\approx$  Kochrezept

## **Genauer:**

- 1 mechanisch nachvollziehbar  
 $\rightsquigarrow$  kein „gesunder Menschenverstand“ nötig
- 2 endliche Beschreibung  $\neq$  endliche Berechnung!
- 3 löst eine Klasse von Problemen

# Was ist ein 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 eine Klasse von Problemen

# Was ist ein 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 eine Klasse von Problemen

$x + y$ , nicht nur  $17 + 4$



# Was ist ein 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 eine Klasse von Problemen

$x + y$ , nicht nur  $17 + 4$

typisches Beispiel: *Bubblesort*

nicht ein spezielles Programm,  
sondern die zugrundeliegende Idee

**Anschaulich:**

Folge von Anweisungen  $\approx$  Kochrezept



**Welche Algorithmen wurden in den Videos vorgestellt?**

**Genauer:**

z.B. Java Programm

- 1 mechanisch nachvollziehbar
- 2 endliche Beschreibung  $\neq$  endliche Berechnung!
- 3 löst eine Klasse von Problemen

$x + y$ , nicht nur  $17 + 4$

typisches Beispiel: *Bubblesort*

nicht ein spezielles Programm,  
sondern die zugrundeliegende Idee

# 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 eine Klasse von Problemen

$x + y$ , nicht nur  $17 + 4$

typisches Beispiel: *Bubblesort*

nicht ein spezielles Programm,  
sondern die zugrundeliegende Idee



# Was ist eine Datenstruktur?



# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

↪ sehr schwammiger Begriff!

# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

→ sehr schwammiger Begriff!





# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

→ sehr schwammiger Begriff!

Für uns:



# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“  
→ sehr schwammiger Begriff!

Für uns:

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

*application programming interface*



# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

↪ sehr schwammiger Begriff!

Für uns:

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

Beispiel: *Union-Find*

*application programming interface*



# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

→ sehr schwammiger Begriff!

Für uns:

*application programming interface*

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

Beispiel: *Union-Find*

- **Implementierung** eines Interface:  
pro Operation ein Algorithmus, der die Spezifikation erfüllt





# Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“  
→ sehr schwammiger Begriff!

Für uns:

*application programming interface*

- **Interface:** Spezifikation von Operationen API  
= *abstract data type (ADT)*  
Beispiel: *Union-Find*
- **Implementierung** eines Interface:  
pro Operation ein Algorithmus, der die Spezifikation erfüllt  
Beispiel: *Quick Union*



Warum nicht einfach: Algorithmus = Java Programm?



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert! ↗

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

(beide stehen zu Beginn)

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

## Mensch (zu Fuß) oder Zug: Wer ist schneller?

(beide stehen zu Beginn)

- Kommt darauf an! Am Anfang sicher der Mensch ...



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert! ↗

**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

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert! ↗

**Beispiel:** Wie viel Speicher brauchen  $n$  natürliche Zahlen?

↖  
als Funktion in  $n$

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**Beispiel:** Wie viel Speicher brauchen  $n$  natürliche Zahlen?

- **Java:**  $32n$  bit, da int eine 32 Bit Zahl ist.

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert! ↗

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.
- ↪ Halten Wortbreite variabel.



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.
- ↪ Halten Wortbreite variabel. ← kann sogar mit  $n$  mitwachsen

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.
- ↪ Halten Wortbreite variabel. ← kann sogar mit  $n$  mitwachsen  
← „Überlebt Pentium 4.“

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.
- ↪ Halten Wortbreite variabel. ← kann sogar mit  $n$  mitwachsen  
← „Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.
- ↗ Halten Wortbreite variabel. ↗ kann sogar mit  $n$  mitwachsen  
↗ „Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

- selbst-modifizierender Code



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert! ↗

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.
- ↪ Halten Wortbreite variabel. ↗ kann sogar mit  $n$  mitwachsen  
↗ „Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

- selbst-modifizierender Code
- Rechnen mit reellen Zahlen  
(≠ floating-point) ↗

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**  
Sind am Verhalten für große Eingaben interessiert!

**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, würde Programmierer aus Effizienzgründen trotzdem 32-bit ints wählen.
- $\rightsquigarrow$  Halten Wortbreite variabel. kann sogar mit  $n$  mitwachsen  
„Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

- selbst-modifizierender Code
- Rechnen mit reellen Zahlen  
( $\neq$  floating-point)
- Magie

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ~ erst über Algorithmen nachdenken

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↷ erst über Algorithmen nachdenken



- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

↪ erst über Algorithmen nachdenken

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↳ erst über Algorithmen nachdenken

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↳ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↳ “look into the box”

top down funktioniert nur, wenn wir überblicken können,  
dass und wie Schritte implementiert werden können!

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↪ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↪ “look into the box”

top down funktioniert nur, wenn wir überblicken können,  
dass und wie Schritte implementiert werden können!

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↪ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↪ “look into the box”

top down funktioniert nur, wenn wir überblicken können,  
dass und wie Schritte implementiert werden können!

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↪ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↪ “look into the box”

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

## Pseudocode

- für Informatiker lesbar und mit etwas Aufwand in (Java/...)-Code übersetzbar
- keine strenge Syntax, aber eindeutig
- ↪ nicht direkt verwendbar für Computer, also *pseudo code*

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↪ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↪ “look into the box”

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

## Pseudocode

- für Informatiker lesbar und mit etwas Aufwand in (Java/...)-Code übersetzbar
- keine strenge Syntax, aber eindeutig
- ↪ nicht direkt verwendbar für Computer, also *pseudo code*



- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↪ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↪ “look into the box”

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

## Pseudocode

- für Informatiker lesbar und mit etwas Aufwand in (Java/...)-Code übersetzbar
- keine strenge Syntax, aber eindeutig
- ↪ nicht direkt verwendbar für Computer, also *pseudo code*

“Überlebt Java.”



- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↪ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↪ “look into the box”

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

## Pseudocode

- für Informatiker lesbar und mit etwas Aufwand in (Java/...)-Code übersetzbar
- keine strenge Syntax, aber eindeutig

↪ nicht direkt verwendbar für Computer, also *pseudo code*

“Überlebt Java.”



- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ↪ erst über Algorithmen nachdenken



## Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank?

Antwort

↪ “look into the box”

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

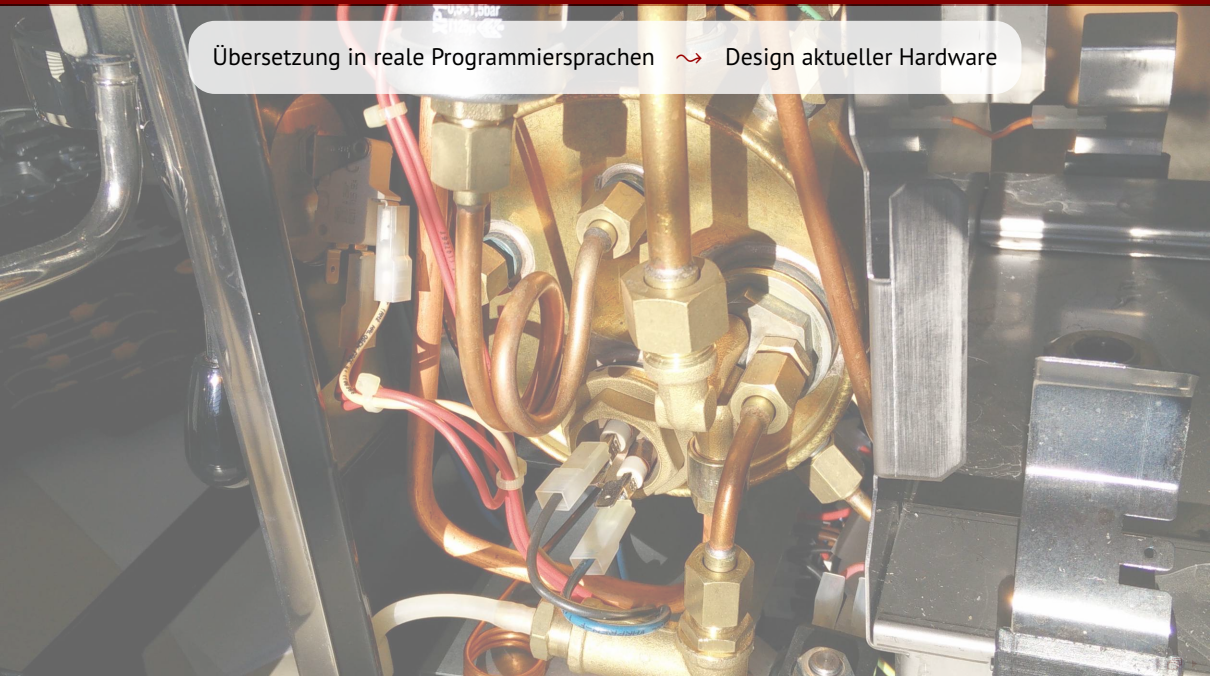
## Pseudocode

- für Informatiker lesbar und mit etwas Aufwand in (Java/...)-Code übersetzbar
- keine strenge Syntax, aber eindeutig
- ↪ nicht direkt verwendbar für Computer, also *pseudo code*

“Überlebt Java.”



Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware



Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

**Abstraktes Maschinenmodell: RAM**  $\leftarrow$  random access machine

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

**Abstraktes Maschinenmodell: RAM** ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

**Abstraktes Maschinenmodell: RAM** ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße



Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: **RAM** ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: **RAM** ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B.  $MEM[2] := MEM[0] + MEM[1]$

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: **RAM** ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
  - jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
  - Befehle der Maschine sind:
    - Operationen auf 2 Zahlen, z.B.  $MEM[2] := MEM[0] + MEM[1]$
    - statt  $x + y$  auch  $x - y, x \cdot y, x \text{ div } y, x \text{ mod } y$  sowie bitweise logische Operationen
- ← Achtung! Alle Operationen modulo  $2^w$

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: RAM ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B.  $MEM[2] := MEM[0] + MEM[1]$
  - statt  $x + y$  auch  $x - y, x \cdot y, x \text{ div } y, x \text{ mod } y$  sowie bitweise logische Operationen
- bedingte und unbedingte Sprünge

← Achtung! Alle Operationen modulo  $2^w$



Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: **RAM** ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B.  $MEM[2] := MEM[0] + MEM[1]$
  - statt  $x + y$  auch  $x - y, x \cdot y, x \text{ div } y, x \text{ mod } y$  sowie bitweise logische Operationen
- bedingte und unbedingte Sprünge

← Achtung! Alle Operationen modulo  $2^w$

oben: vereinfachte Version ohne Register

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

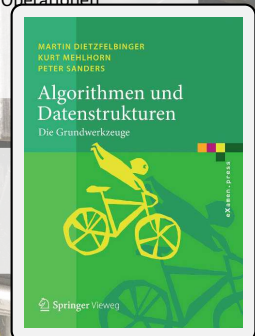
## Abstraktes Maschinenmodell: **RAM** ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B.  $MEM[2] := MEM[0] + MEM[1]$
  - statt  $x + y$  auch  $x - y, x \cdot y, x \text{ div } y, x \text{ mod } y$  sowie bitweise logische Operationen
- bedingte und unbedingte Sprünge

← Achtung! Alle Operationen modulo  $2^w$

oben: vereinfachte Version ohne Register

**alle Details:** Kapitel 2.2 aus  
Dietzfelbinger, Mehlhorn, Sanders: *Algorithmen und Datenstrukturen*



Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: RAM ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad ...  
So will man nicht programmieren müssen!
  - Meistens reicht uns eine abstraktere Sichtweise.
  - **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!
- $\rightsquigarrow$  große Errungenschaft der theoretischen Informatik,  
da wir mathematische Beweise führen können.

↑  
eternal truths

oben: V  
alle De



Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: RAM ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad ...

So will man nicht programmieren müssen!

- Meistens reicht uns eine abstraktere Sichtweise.

- **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!

$\rightsquigarrow$  große Errungenschaft der theoretischen Informatik,  
da wir mathematische Beweise führen können.

↑  
eternal truths

oben: V  
alle De

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: RAM ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad ...  
So will man nicht programmieren müssen!
  - Meistens reicht uns eine abstraktere Sichtweise.
  - **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!
- $\rightsquigarrow$  große Errungenschaft der theoretischen Informatik,  
da wir mathematische Beweise führen können.

eternal truths

oben: V  
alle De

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: RAM ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad ...  
So will man nicht programmieren müssen!
- Meistens reicht uns eine abstraktere Sichtweise.
- **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!

$\rightsquigarrow$  große Errungenschaft der theoretischen Informatik,  
da wir mathematische Beweise führen können.

↑  
eternal truths

oben: V  
alle De

Übersetzung in reale Programmiersprachen  $\rightsquigarrow$  Design aktueller Hardware

## Abstraktes Maschinenmodell: RAM ← random access machine

- unbeschränkt großer Speicher  $MEM[0], MEM[1], MEM[2], \dots$
- jede Zelle  $MEM[i]$  speichert eine  $w$ -Bit Zahl, also ganze Zahl aus  $[0..2^w - 1]$   
 $w$  ist die Wortbreite; typischerweise  $2^w \approx n$ , die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad ...  
So will man nicht programmieren müssen!
  - Meistens reicht uns eine abstraktere Sichtweise.
  - **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!
- $\rightsquigarrow$  große Errungenschaft der theoretischen Informatik,  
da wir mathematische Beweise führen können.

←  
eternal truths

oben: v  
alle De

