Technische Universität Braunschweig

# Algorithmen und Datenstrukturen
## Große Übung 0

Arne Schmidt

03.11.2022

# Vorstellung

Interessen

- Geometrische Optimierung,
- Programmierbare Materie,
- Komplexitätstheorie

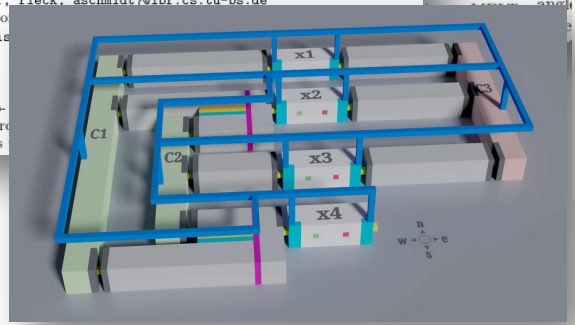
Computing MaxMin Edge Length Triangulations

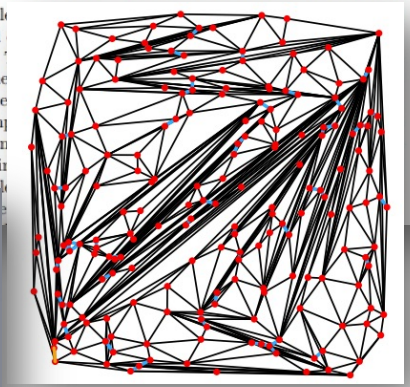
Sándor P. Fekete * Winfried Hellmann* Michael Hemmer* Arne Schmidt*
Julian Troegel*

Particle-Based Assembly
Using Precise Global Control*

Computing Nonsimple Polygons
of Minimum Perimeter

Sándor P. Fekete[1], Andreas Haas[1], Michael Hemmer[1], Michael Hoffmann[2], Irina Kostitsyna[3], Dominik Krupke[1], Florian Maurer[1], Joseph S. B. Mitchell[4], Arne Schmidt[1], Christiane Schmidt[5], and Julian Troegel[1]



# Organisation

# Homepage und Anmeldung

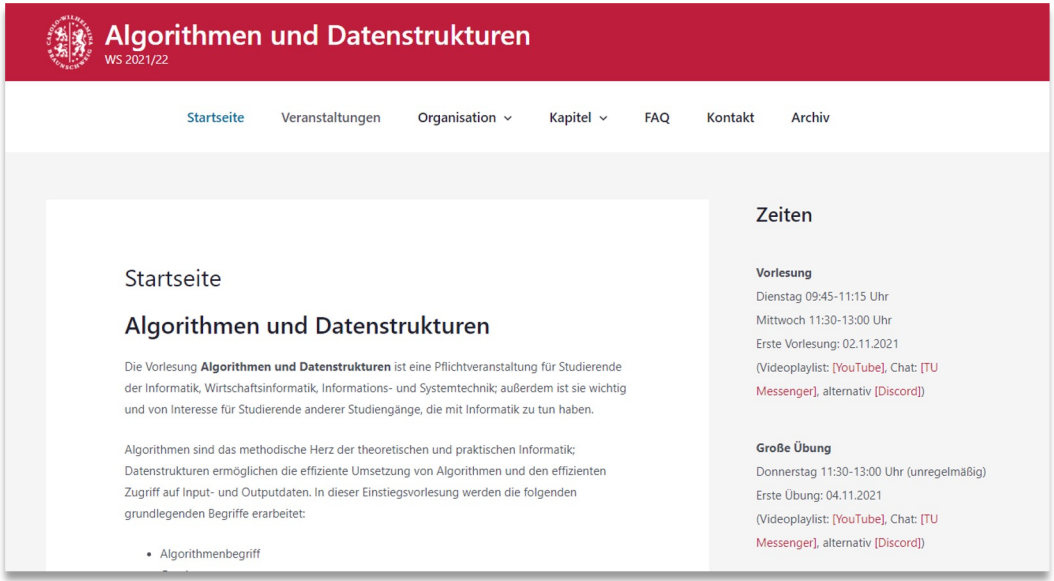
- Veranstaltungsübersicht:

  *https://www.ibr.cs.tu-bs.de/courses/ws2223/aud/index.html*

# Homepage und Anmeldung

- Folien, Hausaufgaben, Vorlesungsvideos, Altklausuren, Übungsanmeldungen:

  *https://www.aud.ibr.cs.tu-bs.de*



# Homepage und Anmeldung

- Anmeldung:

  *https://www.aud.ibr.cs.tu-bs.de/organisation*

- Zuweisung erfolgt (voraussichtlich) am 09.11.22

## Anmeldung

- **Mailingliste:** HIER. ***Es werden nur Mailadressen der TU Braunschweig freigeschaltet.*** Bei Fragen oder technischen Problemen wendet euch einfach per Mail an Arne.
- **Übungsgruppen:** Bis einschließlich 08.11.22 auf der Beitragsseite.
- **Klausur:** Beim jeweiligen Prüfungsamt

## Anmeldung Kleine Übungen

Allgemein / 26. Oktober 2022

Die Anmeldung für die kleinen Übungen ist ab sofort bis einschließlich 08.11.22 geöffnet.

Bitte beachtet folgende Punkte:

- Bitte wählt so viele Termine wie möglich, damit wir am Ende eine faire Zuteilung garantieren können.
- Wir werten immer die neueste Anmeldung (einfach das Formular erneut abschicken).
- Bei erfolgreicher Anmeldung erhält man eine Mail mit einer Übersicht der eingegebenen Daten. Das ist noch keine Zuweisung der Gruppen!
- Überprüft **vor** dem Absenden des Formulars, ob eure Daten (vor allem Matrikelnummer und eMail) korrekt sind.
- Die **Einteilung** in die Gruppen findet voraussichtlich am 09.11.22 statt. Ihr erhaltet dazu eine separate Mail.

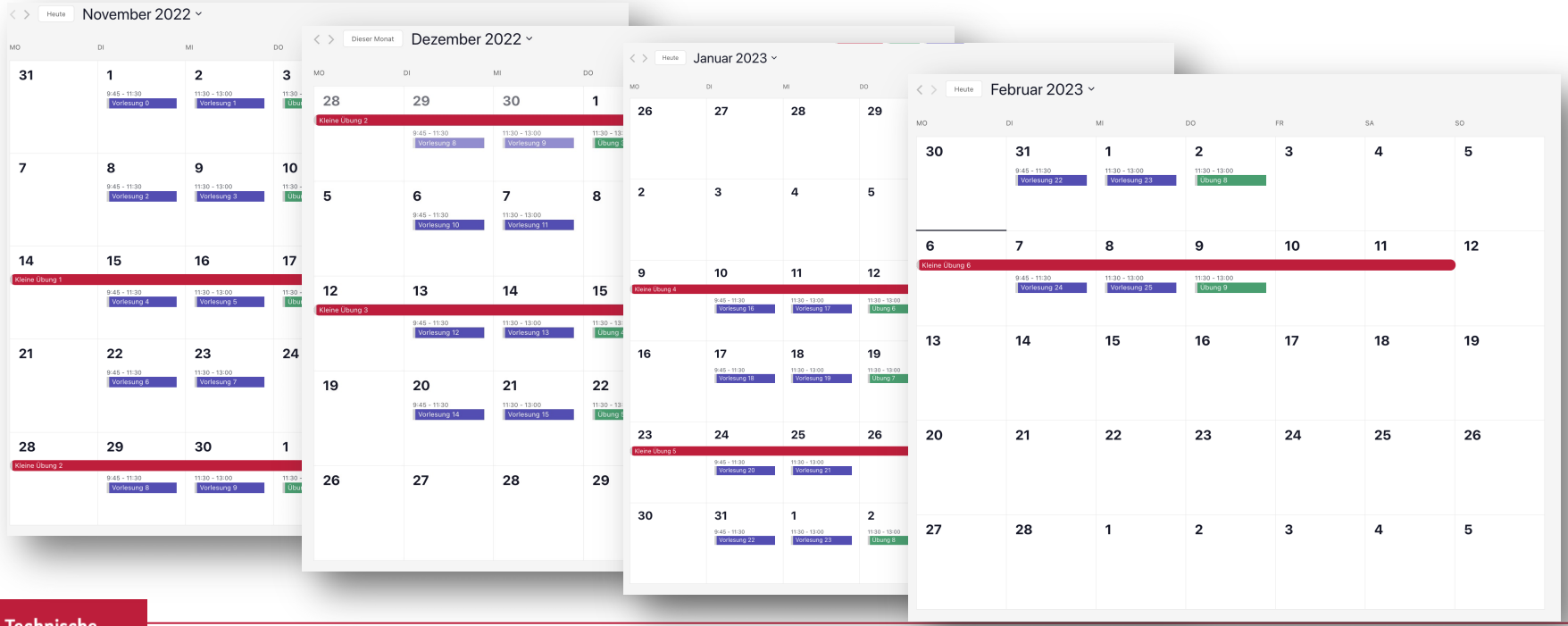
Nachname

Vorname

Matrikelnummer

# Semesterkalender

https://aud.ibr.cs.tu-bs.de/events/



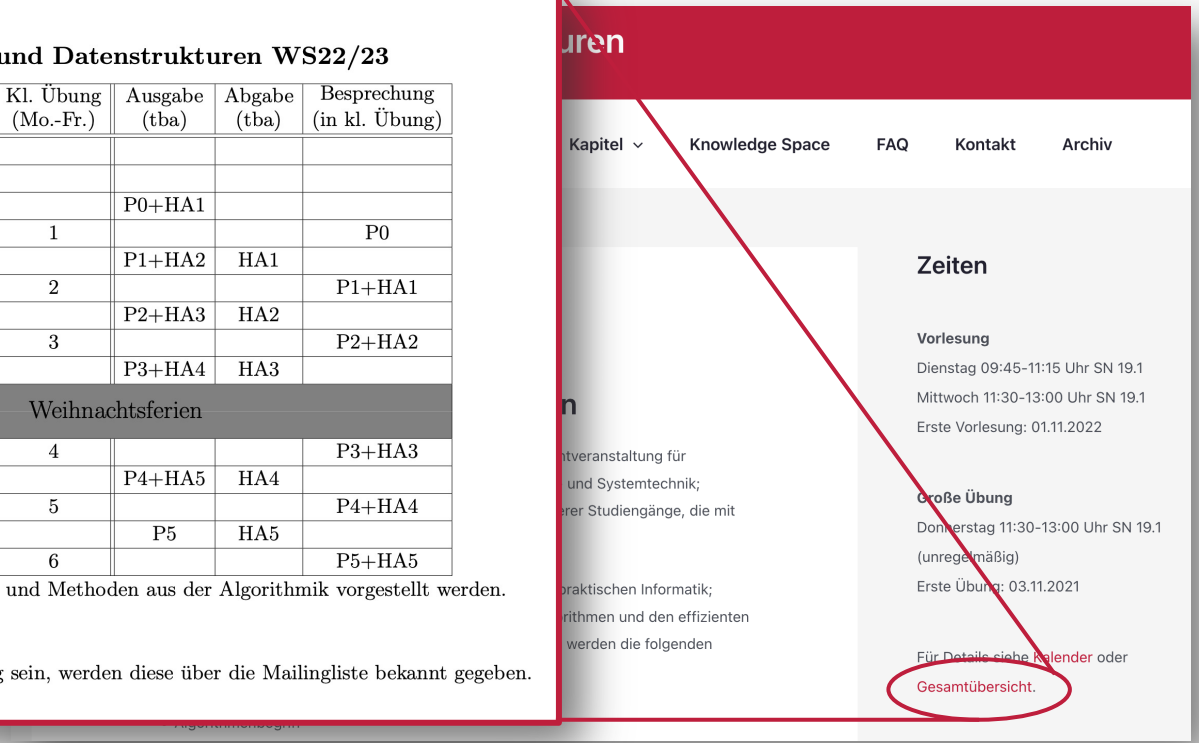
# Semesterplan

**Semesterplan Algorithmen und Datenstrukturen WS22/23**

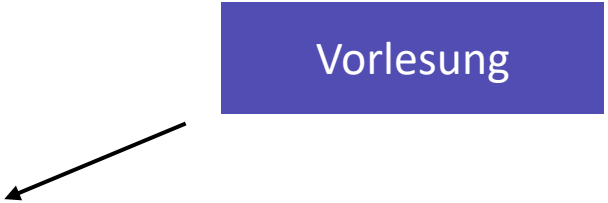
| Woche (KW) | Woche (Datum) | Vorlesung (Di./Mi.) | Gr. Übung (Do.) | Kl. Übung (Mo.-Fr.) | Ausgabe (tba) | Abgabe (tba) | Besprechung (in kl. Übung) |
|---|---|---|---|---|---|---|---|
| 44 | 24.10. | | | | | | |
| 45 | 31.10. | 0,1 | 0 | | | | |
| 46 | 07.11. | 2,3 | 1 | | P0+HA1 | | |
| 47 | 14.11. | 4,5 | 2* | 1 | | | P0 |
| 48 | 21.11. | 6,7 | | | P1+HA2 | HA1 | |
| 49 | 28.11. | 8,9 | 3 | 2 | | | P1+HA1 |
| 50 | 05.12. | 10,11 | | | P2+HA3 | HA2 | |
| 51 | 12.12. | 12,13 | 4 | 3 | | | P2+HA2 |
| 52 | 19.12. | 14,15 | 5* | | P3+HA4 | HA3 | |
| 53 | 26.12. | Weihnachtsferien | | | | | |
| 1 | 02.01. | | | | | | |
| 2 | 09.01. | 16,17 | 6 | 4 | | | P3+HA3 |
| 3 | 16.01. | 18,19 | 7 | | P4+HA5 | HA4 | |
| 4 | 23.01. | 20,21 | | 5 | | | P4+HA4 |
| 5 | 30.01. | 22,23 | 8 | | P5 | HA5 | |
| 6 | 06.02. | 24,25 | 9(*) | 6 | | | P5+HA5 |

*: Eine Exkurs-Übung, in der spannende Probleme und Methoden aus der Algorithmik vorgestellt werden.

Sollten kurzfristig Änderungen im Zeitplan notwendig sein, werden diese über die Mailingliste bekannt gegeben.



# Inhalte




**Große Übung**

- Aufarbeitung der Inhalte
- Exkurse
- Beantwortung von Fragen
- Interaktion!

**Kleine Übung**

- Vertiefung der Inhalte
- Selbständiges Arbeiten
- Besprechung von Hausaufgaben

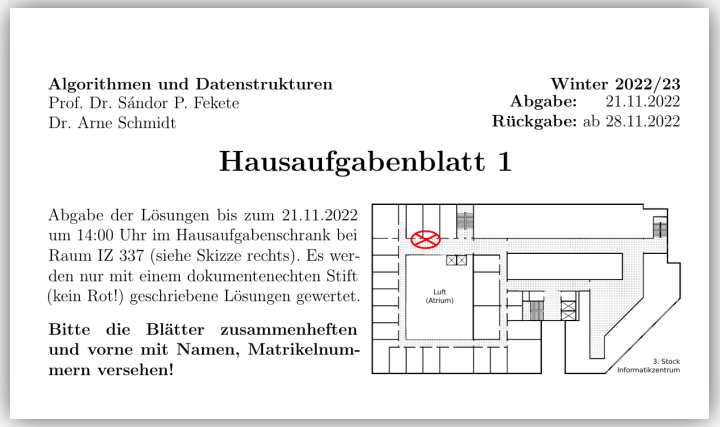
# Hausaufgaben und Übungsblätter

**5 verpflichtende Hausaufgabenblätter**

- Studienleistung
- 20 Punkte pro Blatt

**6 freiwillige Übungsblätter**

- Zusätzliche Vertiefung
- Prüfungsvorbereitung



**Algorithmen und Datenstrukturen**
Prof. Dr. Sándor P. Fekete
Dr. Arne Schmidt

**Winter 2022/23**
**Abgabe:** 21.11.2022
**Rückgabe:** ab 28.11.2022

## Hausaufgabenblatt 1

Abgabe der Lösungen bis zum 21.11.2022 um 14:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

**Bitte die Blätter zusammenheften und vorne mit Namen, Matrikelnummern versehen!**



- Studienleistung: 50% der Gesamtpunkte
  - Studienleistung ist **keine** Voraussetzung, um an der Prüfung teilzunehmen.
  - Studienleistung ist **eine** Voraussetzung, um das Modul abzuschließen.
  - Studienleistung ist nicht benotet und fließt nicht in die Prüfung ein.

# Hausaufgaben

**Wozu Hausaufgaben?**
Die Hausaufgaben dienen Euch (nicht uns) zur Vorbereitung auf die Klausur.

- Ideale Nachbereitung der Vorlesungsinhalte
- Zeitersparnis bei der Prüfungsvorbereitung
- Direktes Feedback, über euren aktuellen Lernstand.

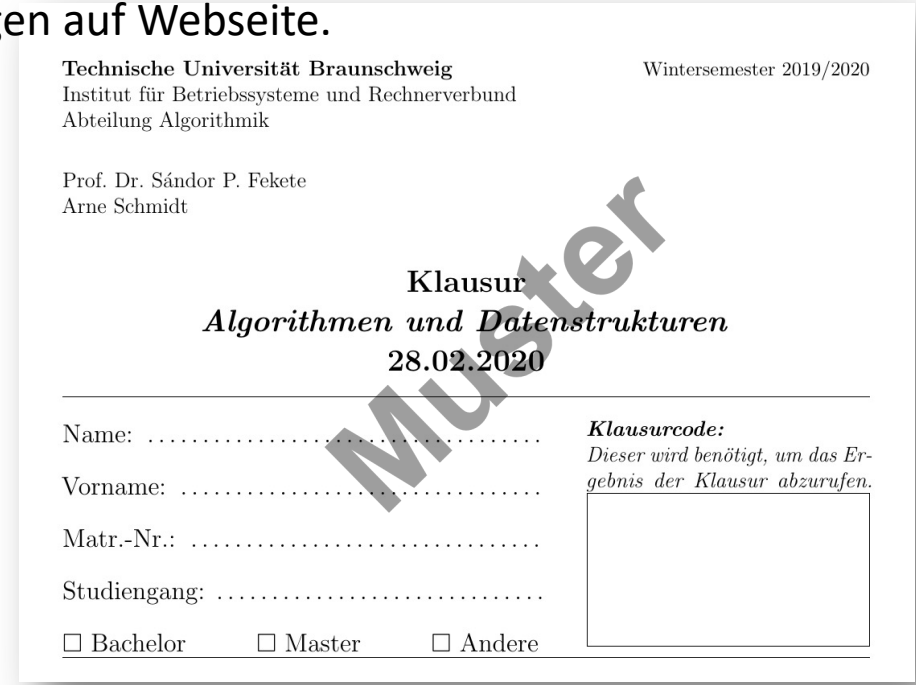
- Zu späte Abgaben: 0 Punkte
- Zusammen überlegen, **ABER:** einzeln aufschreiben und abgeben, sonst: 0 Punkte.

# Hausaufgaben


Hausaufgabenschrank

# Klausur

- Voraussichtlich 03.03.23, zwischen 10:30 Uhr und 13:30 Uhr.
- Raumaufteilung und Beginn der Klausur folgen auf Webseite.
- Inhalt: Prinzipiell alles aus VL und Übung
- Dauer: 2 Stunden

**Technische Universität Braunschweig** Wintersemester 2019/2020
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Prof. Dr. Sándor P. Fekete
Arne Schmidt

**Klausur**
***Algorithmen und Datenstrukturen***
**28.02.2020**

Muster

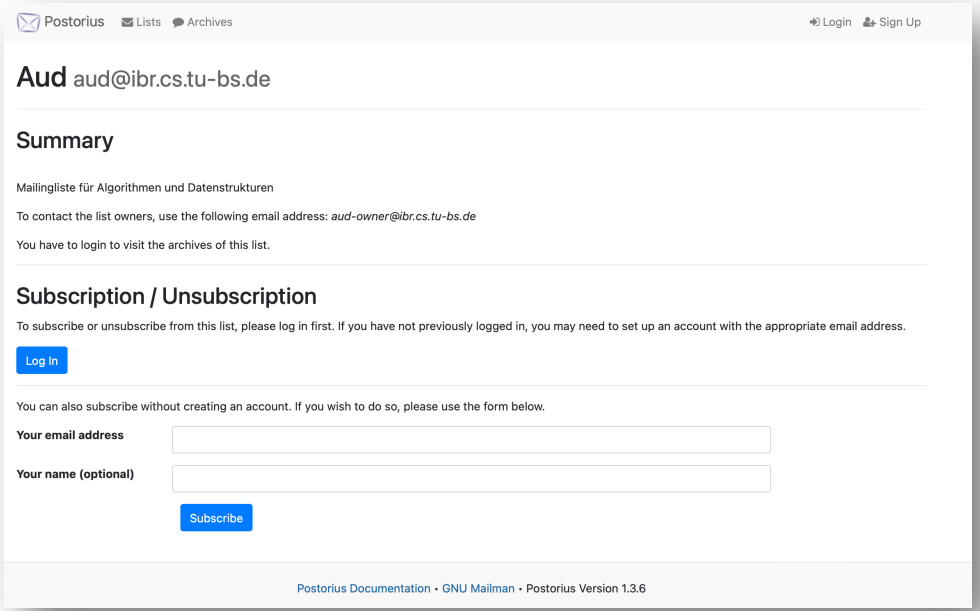
Name: .....
Vorname: .....
Matr.-Nr.: .....
Studiengang: .....

☐ Bachelor ☐ Master ☐ Andere

***Klausurcode:***
*Dieser wird benötigt, um das Ergebnis der Klausur abzurufen.*

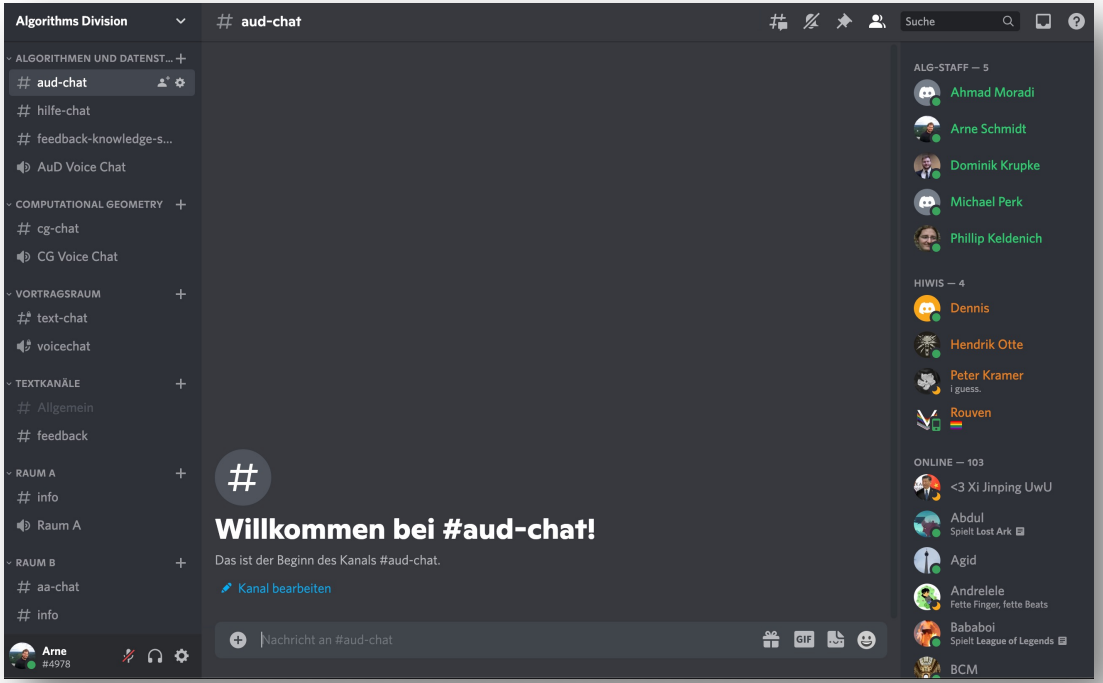
# Mailingliste

- Anmeldung über Homepage
- Für Informationen wie
  - Raumänderungen,
  - Ausfälle,
  - Etc.
- Möglichkeit für Fragen



# Discord

Discord




Kommunikation abseits der Vorlesung und Übung und Austausch mit Mitstudierenden.

# Fragen

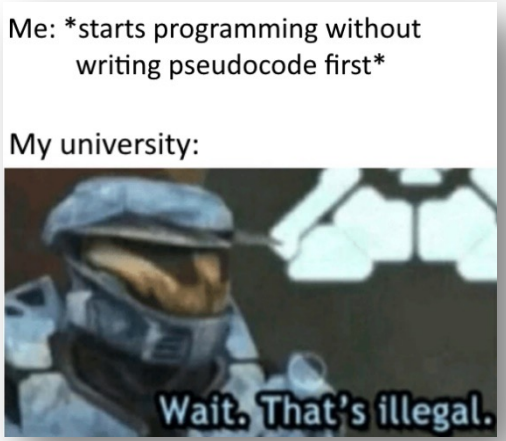
Stellt Eure Fragen

- Über die Mailingliste
- Euren Tutoren
- Sprechstunde von Arne (Mo. 9:45 – 10:30 Uhr)
- Per Mail an Arne (aschmidt@ibr.cs.tu-bs.de)

# Fragen?

# Pseudocode

https://www.ibr.cs.tu-bs.de/alg/Merkzettel/pseudocode-booklet.pdf



# Pseudocode

Wörterbuch

pseu·do

/pseúdo/

*Adjektiv* **UMGANGSSPRACHLICH**

nicht echt, nur nachgemacht, nachgeahmt

Wörterbuch

Quell·code

/Quéllcode/

*Substantiv, maskulin* [der] **EDV**

in einer [höheren] Programmiersprache geschriebene Abfolge von Programmanweisungen, die vom Menschen gelesen, aber erst nach einer elektronischen Übersetzung vom Computer verarbeitet werden können

# Warum Pseudocode?

```
public class HalloWelt {
  public static void main(String[] args) {
    System.out.println("Hallo Welt!");
  }
}
```

Ist dieser Code effizient zu lesen?

Was können wir weglassen, wenn ein Mensch (kein Computer) diesen Code verstehen soll?

# Warum Pseudocode?

Java

```
public class HalloWelt {
  public static void main(String[] args) {
    System.out.println("Hallo Welt!");
  }
}
```

Pseudocode

```
function main(args)
    print Hallo Welt!
end function
```

# Problem und Instanz

**Problem**

- Allgemeine Fragestellung
- Lösung: Angabe eines Algorithmus
- (Meist) Formuliert durch
  - Eingabe: Was ist gegeben?
  - Ausgabe: Was ist gesucht?

**Instanz**

- Konkrete Fragestellung
- Lösung: Angabe einer konkreten Ausgabe
- Formuliert durch eine konkrete Eingabe eines bestimmten Problems

# Problem und Instanz Beispiel

Problem → Instanz

**Größter gemeinsamer Teiler zweier Zahlen**

Eingabe: Zwei Zahlen *a* und *b*.

Ausgabe: Der größte gemeinsame Teiler *q* von *a* und *b*.

Lösung: Euklidischer Algorithmus (gleich mehr)

**Größter gemeinsamer Teiler zweier Zahlen**

- ggT(5, 102); Lösung: 1
- ggT(8, 64); Lösung: 8
- …

# Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

- Eigenschaften
  - Ausführbarkeit
  - Endlichkeit
  - Endliche Ausführzeit
  - Endlicher Speicherbedarf
- Beschrieben durch
  - Prosatext
  - Pseudocode

Donald Knuth

Alan Turing

# Beispiel: Euklidischer Algorithmus

**Prosatext**

Solange die Zahl *b* nicht 0 ist, wählen wir $h = a \bmod b$, setzen *a* auf *b* und *b* auf h. Nachdem $b = 0$, geben wir a zurück.

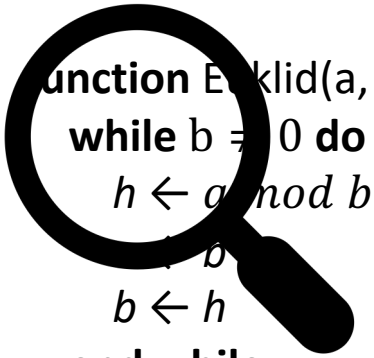
**Pseudocode**

```
function Euklid(a, b)
    while b ≠ 0 do
        h ← a mod b
        a ← b
        b ← h
    end while
    return a
end function
```

(Für modulo gilt $r = a \, mod \, b$, sodass $a = q \cdot b + r$ für ein $q \in \mathbb{Z}$ gilt mit $0 \leq r < b$.)

# Beispiel: Euklidischer Algorithmus

**Pseudocode**



**function** Euklid(a, b)
  **while** b ≠ 0 **do**
    $h \leftarrow a \bmod b$
    $a \leftarrow b$
    $b \leftarrow h$
  **end while**
  **return** *a*
**end function**

**Schlüsselwörter**

- Anlehnung an höhere Programmiersprachen
- Vereinfachung zur Übersichtlichkeit (keine Klammern, Typen, etc.)
- ABER: **end** statements
- Können sowohl auf Deutsch als auch auf Englisch benutzt werden

# Pseudocode - Bausteine

## Methoden

**function** NAME (Param1, Param2, …)
…
**end function**

## Zuweisungen

$a := b$
$a \leftarrow b$
(Weist $a$ den Wert $b$ zu)

$a := b + c$ (Weist a die Summe aus b und c zu)
$A := B$ (Weist der Menge A die Menge B zu)

## Bedingungen

**if** Bedingung **then**
Aktion falls Bedingung wahr ist
**else**
Aktion, falls Bedingung falsch ist
**end if**

## Ausgaben / Rückgaben

**print** $a$ (Gibt $a$ aus)
**return** $a$ (Gibt $a$ aus und beendet die Funktion)

# Pseudocode - Bausteine

## Schleifen - 1

**while** Bedingung **do**

Führe Aktion aus, *solange* eine Bedingung wahr ist.

**end while**

## Schleifen - 2

**repeat**

Führe Aktion aus, *bis* eine Bedingung wahr ist.

**until** Bedingung

## Schleifen - 3

**for** a **in** $b, \dots, c$ **do**

Starte mit $a \coloneqq b$

$a$ wird nach jeder Iteration um 1 erhöht. ($a \coloneqq a + 1$)

Wiederhole bis $a$ den Wert $c$ erreicht hat.

**end for**

## Schleifen - 4

**for** $a \coloneqq b$ **down to** $c$

Starte mit $a \coloneqq b$

$a$ wird in jeder Iteration um 1 verringert. ($a \coloneqq a - 1$)

Wiederhole bis $a$ den Wert $c$ erreicht hat.

**end for**

# Pseudocode – In Aktion

1. **function** Euklid(a, b)
2. **while** b $\neq 0$ **do**
3. $h \leftarrow a \; mod \; b$
4. $a \leftarrow b$
5. $b \leftarrow h$
6. **end while**
7. **return** $a$
8. **end function**

Instanz: Berechne ggT von $a = 49, b = 21$
Aufruf: Euklid(49, 21)
Rückgabe: 7

| Zeile | Iteration | a | b | h |
|---|---|---|---|---|
| 1 | - | 49 | 21 | - |
| 2 | - | 49 | 21 | - |
| 3 | 1 | 49 | 21 | 7 |
| 4 | 1 | 21 | 21 | 7 |
| 5 | 1 | 21 | 7 | 7 |
| 6 | 1 | 21 | 7 | 7 |
| 2 | 1 | 21 | 7 | 7 |
| 3 | 2 | 21 | 7 | 0 |
| 4 | 2 | 7 | 7 | 0 |
| 5 | 2 | 7 | 0 | 0 |
| 6 | 2 | 7 | 0 | 0 |
| 2 | 2 | 7 | 0 | 0 |
| 7 | 2 | 7 | 0 | 0 |

# Pseudocode – In Aktion

```
1.  function Mult(a, b)
2.      c ← 0
3.      for d := b downto 1
4.          c ← c + a
5.      end for
6.      return c
7.  end function
```

Instanz: Berechne das Produkt von $9$ und $3$
Aufruf: Mult(9, 3)

Rückgabe: 27

Variablen nach jeder Iteration der for-Schleife:

| Iteration | a | b | c | d |
|---|---|---|---|---|

# Pseudocode – In Aktion

```
1.  function Absolute(a)
2.      if a < 0 then
3.          a ← −a
4.      end if
5.      return a
6.  end function
```

```
1.  function AbsoluteDiff(a, b)
2.      return Absolute(a) – Absolute(b)
3.  end function
```

Aufruf: AbsoluteDiff(-10, 3)

Rückgabe: 7

| Zeile (AbsoluteDiff ) | Zeile (Absolute) | a (AbsoluteDiff ) | a (Absolute) | b |
|---|---|---|---|---|
| 1 | - | -10 | - | 3 |
| 2 | 1 | -10 | -10 | 3 |
| 2 | 2 | -10 | -10 | 3 |
| 2 | 3 | -10 | 10 | 3 |
| 2 | 4 | -10 | 10 | 3 |
| 2 | 5 | -10 | 10 | 3 |
| 2 | 1 | -10 | 3 | 3 |
| 2 | 2 | -10 | 3 | 3 |
| 2 | 4 | -10 | 3 | 3 |
| 2 | 5 | -10 | 3 | 3 |

# Pseudocode - Varianten

- Auf Deutsch
  - Wenn … Dann … Sonst …
  - Solange …
  - Für $i \leftarrow 1, \ldots, k$ …
- Statt $\leftarrow$ schreibt man gelegentlich $:=$ oder sogar $=$

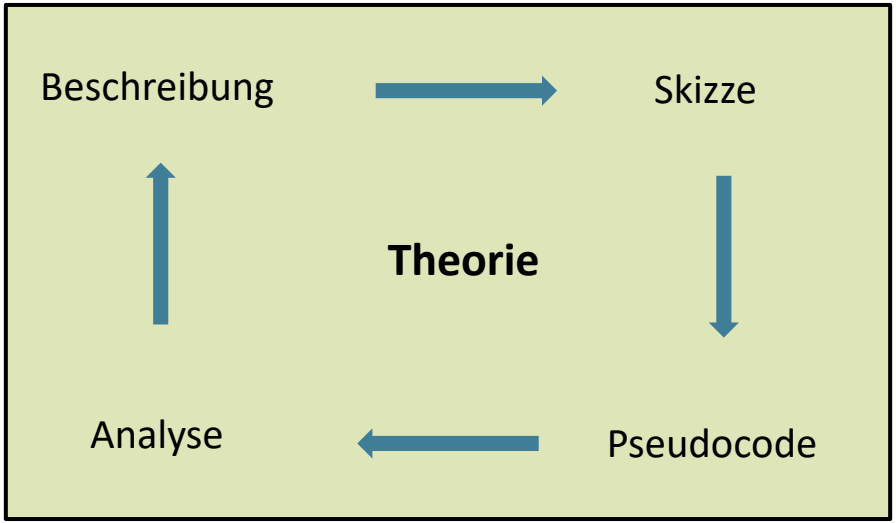
**Wichtig**
Pseudocode muss immer verständlich und eindeutig sein!

- Mischung aus Prosa und Pseudocode:

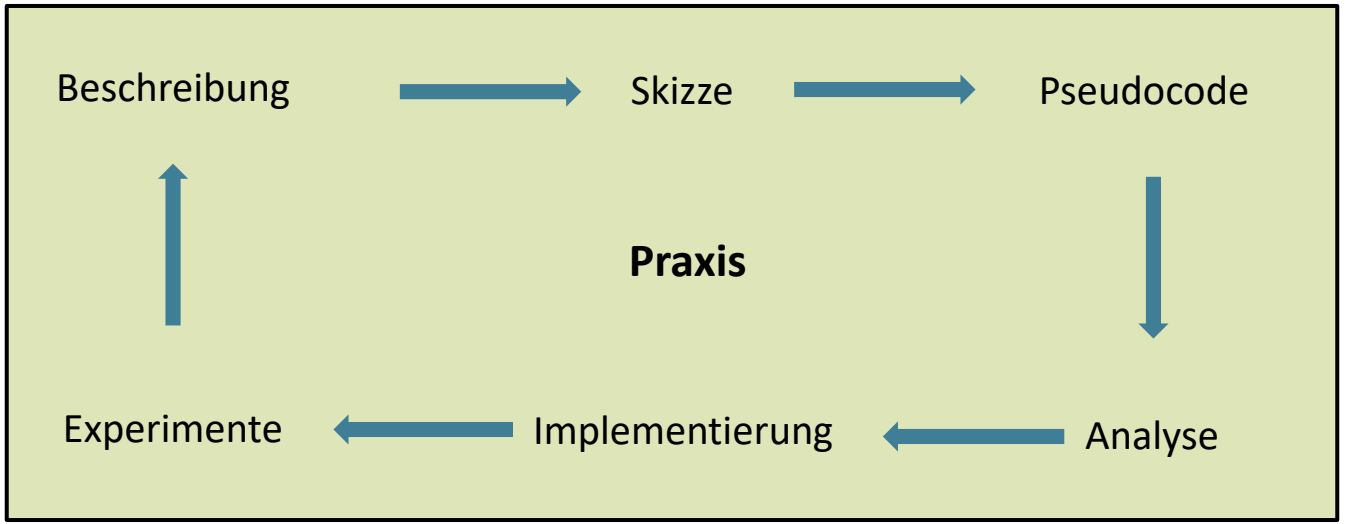
```
1.  function Verhältnis(liste)
2.      Sortiere die Zahlen in liste aufsteigend
3.      return liste[0] / liste[länge(liste)]
4.  end function
```

- end… statements dürfen weggelassen werden, aber Einrückung muss korrekt sein!

# Algorithmenentwurf in der Theorie



# Algorithmenentwurf in der Theorie



# Ein einfacher Algorithmus für den Logarithmus

Betrachten wir den Logarithmus:

Gegeben Zwei (ganzzahlige) Zahlen n und b

Gesucht Eine (nicht-ganzzahlige) Zahl x, sodass $n = b^x$ (Oder: $x = \log_b n$)

*Einfach ausgedrückt: Wie oft muss man n durch b teilen, damit man auf 1 kommt?*

Betrachten wir zunächst den einfachen Fall: $x$ ist ganzzahlig

```
1.  function Log(n, b)
2.      log ← 0
3.      while n > 1 do
4.          log ← log + 1
5.          n ← n/b
6.      return log
```

**Beispiel**: Log(64, 2)

| log | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| n | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

# Ein Algorithmus für den Logarithmus

Was passiert, wenn *x* nicht ganzzahlig wird?

Problem: Irrationale Ergebnisse

Lösung: Bestimme den Logarithmus nur auf *k* Stellen genau.

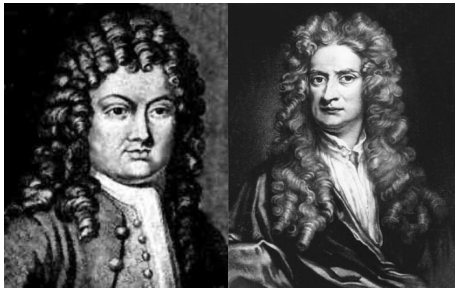
Idee: Nutze den alten Algorithmus, indem wir die Stellen vor das Dezimalkomma verschieben. $10^k \log_b n$ gibt uns *k* Stellen mehr vor dem Komma.

1. **function** Log(n, b, k)
2. $n \leftarrow n^{(10^k)}$ # $10^k \log_b n = \log_b(n^{(10^k)})$
3. $\log \leftarrow Log(n, b)$
4. **return** $\log \cdot 10^{-k}$
5. **end function**

Problem 1: Schon für kleine *k* wird *n* sehr groß

Problem 2: Der Algorithmus benötigt exponentiell viele Schritte

# Mehr zu Logarithmen



Logarithmen lassen sich deutlich schneller und präziser berechnen.

- Taylor-Entwicklung (aus der Analysis)
- Newton-Verfahren (aus der Numerik)

Für diese Vorlesung ist die genaue Berechnung von (nicht ganzzahligen) Logarithmen nicht relevant.

Logarithmen werden aber dennoch benötigt, beispielsweise

- für Laufzeiten, z.B. beim Sortieren (Kapitel 5)
- zum Bestimmen der Höhe eines Suchbaumes (Kapitel 4)

**Wichtig:** Rechenregeln für Logarithmen!

# Fragen?