



Technische
Universität
Braunschweig



Algorithmen und Datenstrukturen

Große Übung 0

Ramin Kosfeld und Chek-Manh Loi

22.10.26

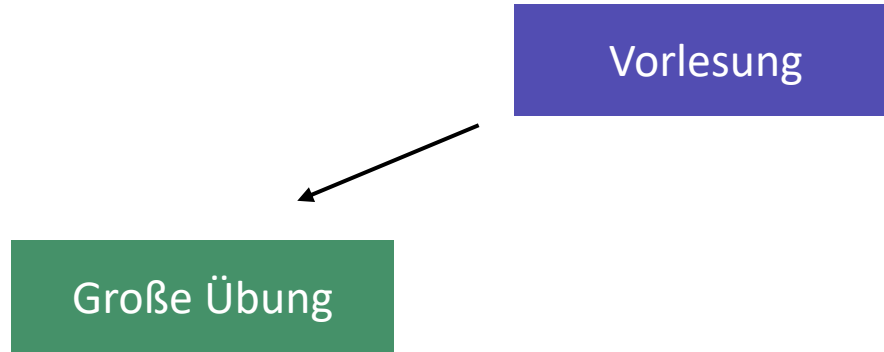
Programm heute

- Organisation
- Einstieg
 - Algorithmen und Pseudocode
- (Logarithmen)

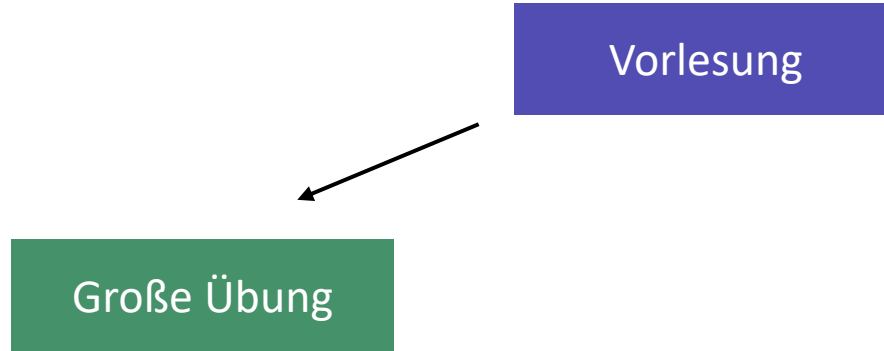
Organisation

Vorlesung

Formate

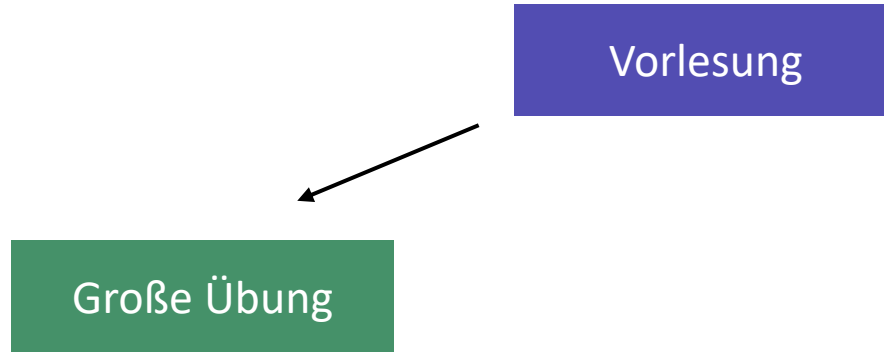


Formate



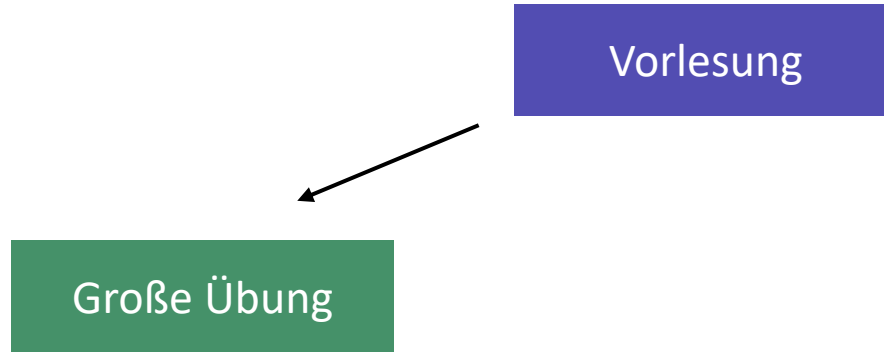
Tafelübungen im Hörsaal (mit allen)

Formate



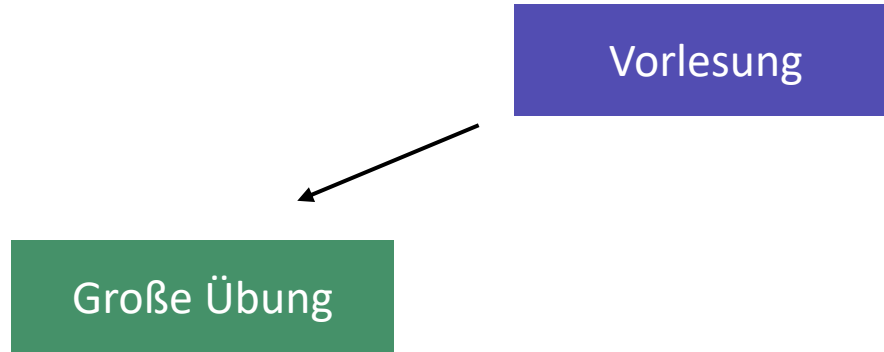
Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen



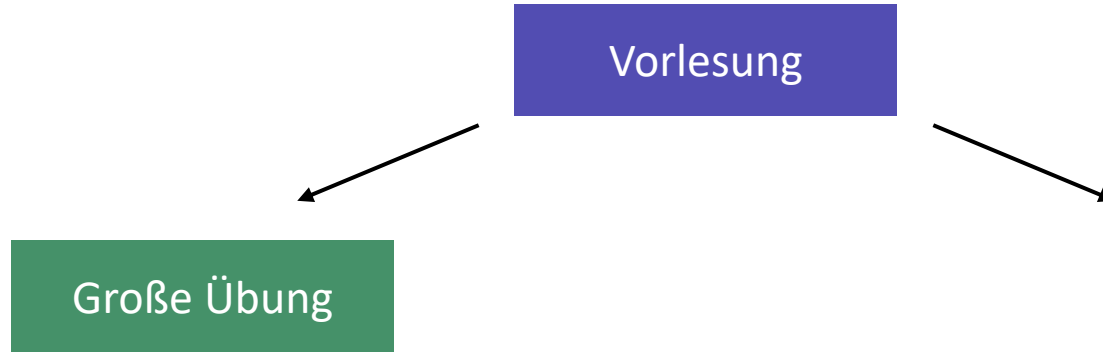
Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen



Tafelübungen im Hörsaal (mit allen)

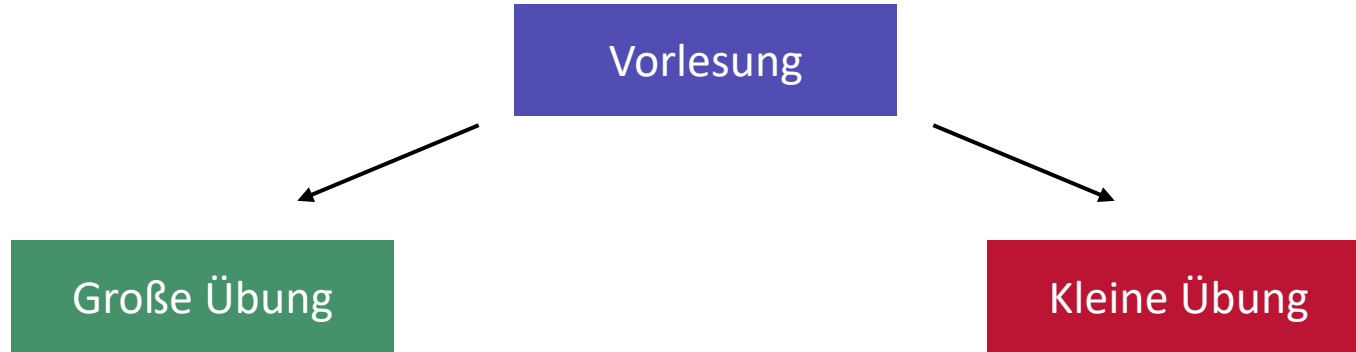
- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen
- Fragen klären



Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen
- Fragen klären

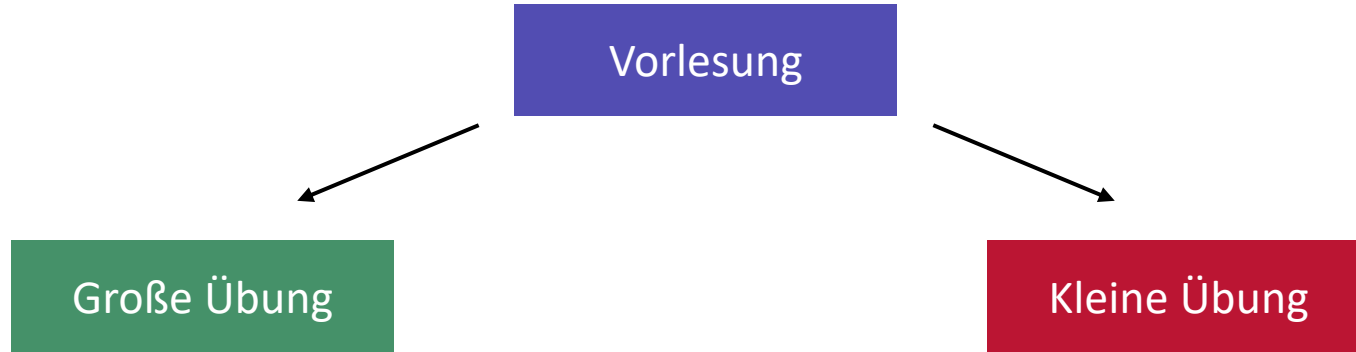
Formate



Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen
- Fragen klären

Kleingruppen (Seminarräume)

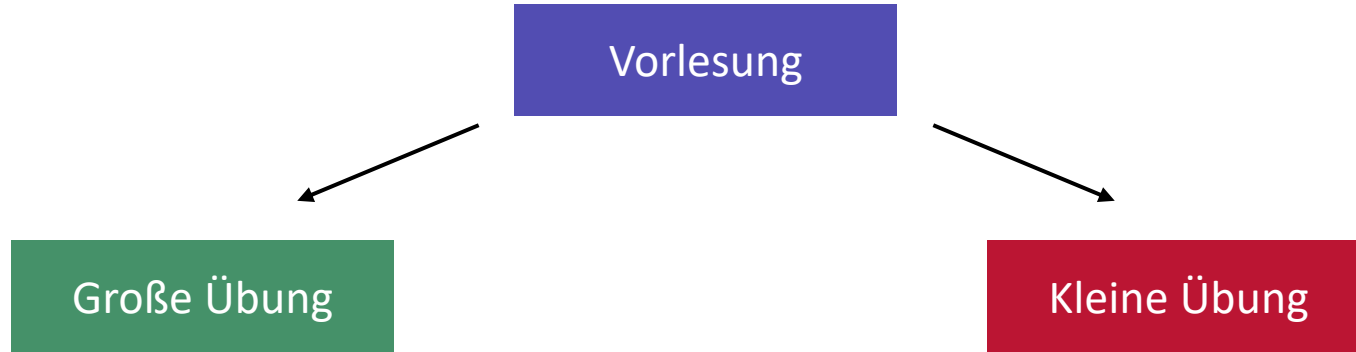


Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen
- Fragen klären

Kleingruppen (Seminarräume)

- Alle 2 Wochen

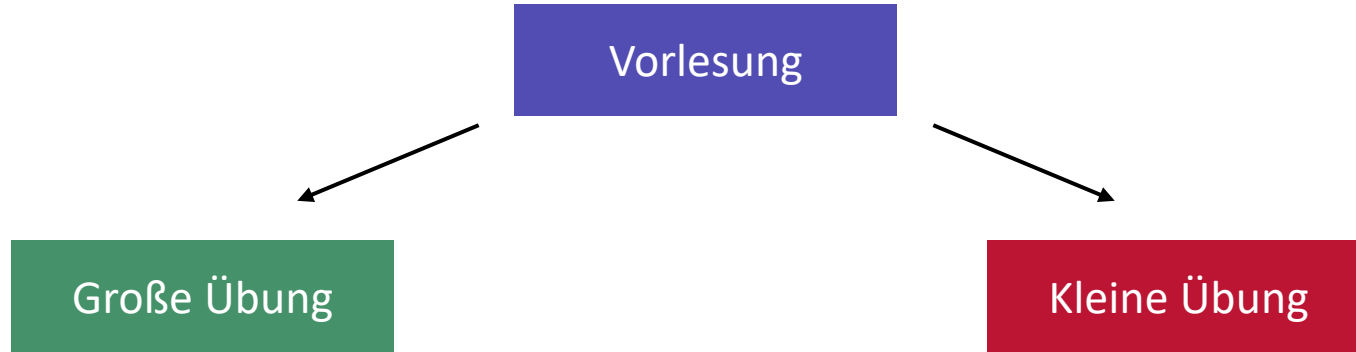


Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen
- Fragen klären

Kleingruppen (Seminarräume)

- Alle 2 Wochen
- Betreut von Übungsleiter*in

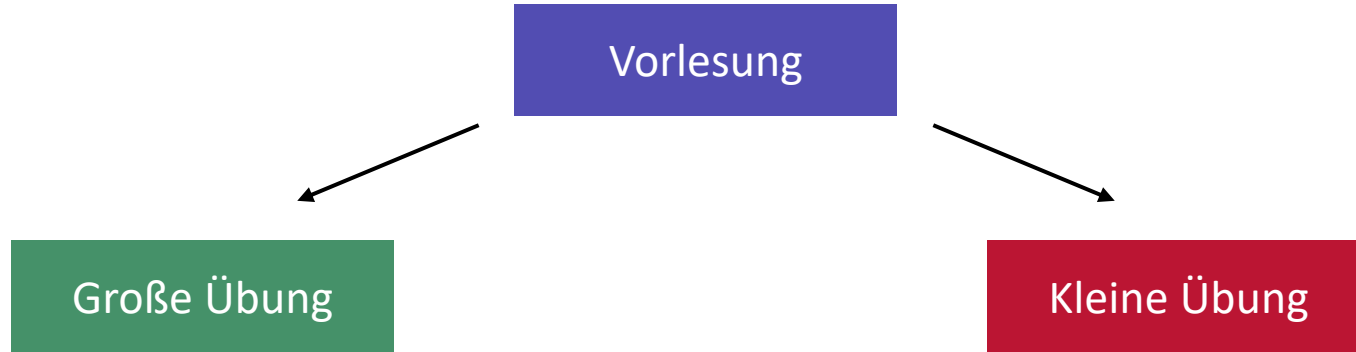


Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen
- Fragen klären

Kleingruppen (Seminarräume)

- Alle 2 Wochen
- Betreut von Übungsleiter*in
- Besprechung von Hausaufgaben



Tafelübungen im Hörsaal (mit allen)

- Ca. alle 2 Wochen
- Wiederholung und Vertiefung der Inhalte, Beispiele, Ergänzungen
- Fragen klären

Kleingruppen (Seminarräume)

- Alle 2 Wochen
- Betreut von Übungsleiter*in
- Besprechung von Hausaufgaben
- Fragen klären und Inhalte üben

Wie besteht man AuD?

Wie besteht man AuD?

Prüfungsleistung

Wie besteht man AuD?

Prüfungsleistung
→ Klausur

Wie besteht man AuD?

Prüfungsleistung
→ Klausur

Studienleistung

Wie besteht man AuD?

Prüfungsleistung
→ Klausur

Studienleistung
→ Hausaufgaben

Wie besteht man AuD?

Prüfungsleistung
→ Klausur

Studienleistung
→ Hausaufgaben

- Beide unabhängig voneinander
(kann an Klausur teilnehmen ohne bestandene HA und andersherum)

Wie besteht man AuD?

Prüfungsleistung
→ Klausur

Studienleistung
→ Hausaufgaben

- Beide unabhängig voneinander
(kann an Klausur teilnehmen ohne bestandene HA und andersherum)
- Beide bestanden: Veranstaltung geschafft :)

Klausur

- Voraussichtlich 17.02.27
- Dauer: 2 Stunden
- Inhalt: Prinzipiell alles aus VL und Übung
- ...mehr, wenn die Klausur näher rückt :)



Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Wintersemester 2023/2024

Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Klausur
Algorithmen und Datenstrukturen
13.02.2024

Name:

Vorname:

Matr.-Nr.:

Studiengang:

☐ Bachelor ☐ Master ☐ Andere

Klausurcode:

Dieser wird benötigt, um das Ergebnis der Klausur abzurufen.

A large, empty rectangular box for entering the Klausurcode.

Hausaufgaben

Hausaufgaben

5 Zettel

Hausaufgaben

5 Zettel

Einzelabgaben

Hausaufgaben

5 Zettel

Einzelabgaben

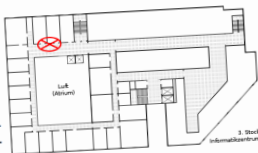
Algorithmen und Datenstrukturen
Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Winter 2024/25
Abgabe: 12.11.2024
Rückgabe: ab 18.11.2024

Hausaufgabenblatt 1

Abgabe der Lösungen bis zum Dienstag, den 12.11.2024 um 13:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

Schreibe auf die Abgabe unbedingt deinen Namen, Matrikel- und Gruppennummer! Mehrere Blätter tackern!



Hausaufgabe 1 (Hamilton):

Betrachte die in Abbildung 1 dargestellten Graphen. Untersuche, welcher dieser Graphen einen Hamiltonkreis besitzt. Begründe deine Antworten für beide Graphen.

(5 Punkte)

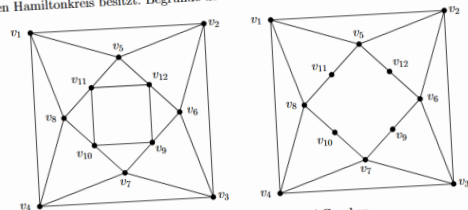


Abbildung 1: Abbildung von zwei Graphen.

Hausaufgabe 2 (Graphen):

Zeichne einen einfachen, zusammenhängenden Graphen mit $n = 7$ Knoten und $m = 9$ Kanten, sodass dieser eine Eulertour, aber keinen Hamiltonpfad bzw. Hamiltonkreis besitzt.

(3 Punkte)

Hausaufgaben

5 Zettel

Einzelabgaben

Aufwand: Überschaubar

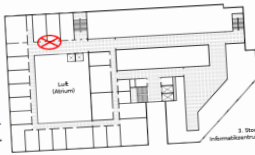
Algorithmen und Datenstrukturen
Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Winter 2024/25
Abgabe: 12.11.2024
Rückgabe: ab 18.11.2024

Hausaufgabenblatt 1

Abgabe der Lösungen bis zum Dienstag, den 12.11.2024 um 13:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

Schreibe auf die Abgabe unbedingt deinen Namen, Matrikel- und Gruppennummer! Mehrere Blätter tackern!



Hausaufgabe 1 (Hamilton):

(5 Punkte)
Betrachte die in Abbildung 1 dargestellten Graphen. Untersuche, welcher dieser Graphen einen Hamiltonkreis besitzt. Begründe deine Antworten für beide Graphen.

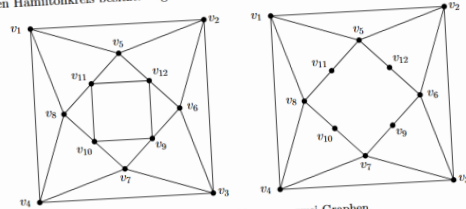


Abbildung 1: Abbildung von zwei Graphen.

Hausaufgabe 2 (Graphen):

(3 Punkte)
Zeichne einen einfachen, zusammenhängenden Graphen mit $n = 7$ Knoten und $m = 9$ Kanten, sodass dieser eine Eulertour, aber keinen Hamiltonpfad bzw. Hamiltonkreis besitzt.

Hausaufgaben

5 Zettel Einzelabgaben

Aufwand: Überschaubar

Neues Blatt: Alle 2 Wochen

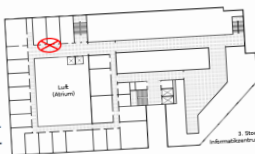
Algorithmen und Datenstrukturen
Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Winter 2024/25
Abgabe: 12.11.2024
Rückgabe: ab 18.11.2024

Hausaufgabenblatt 1

Abgabe der Lösungen bis zum Dienstag, den 12.11.2024 um 13:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

Schreibe auf die Abgabe unbedingt deinen Namen, Matrikel- und Gruppennummer! Mehrere Blätter tackern!



Hausaufgabe 1 (Hamilton):

Betrachte die in Abbildung 1 dargestellten Graphen. Untersuche, welcher dieser Graphen einen Hamiltonkreis besitzt. Begründe deine Antworten für beide Graphen.

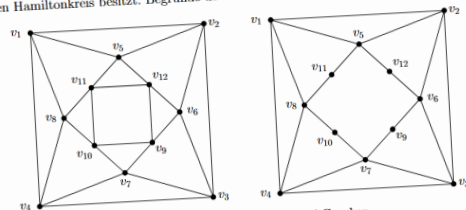


Abbildung 1: Abbildung von zwei Graphen.

Hausaufgabe 2 (Graphen):

Zeichne einen einfachen, zusammenhängenden Graphen mit $n = 7$ Knoten und $m = 9$ Kanten, sodass dieser eine Eulertour, aber keinen Hamiltonpfad bzw. Hamiltonkreis besitzt.

Hausaufgaben

Einzelabgaben

5 Zettel

Aufwand: Überschaubar

Neues Blatt: Alle 2 Wochen

Bearbeitungszeit: 2 Wochen

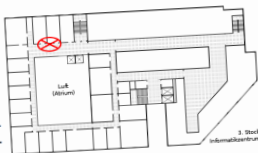
Algorithmen und Datenstrukturen
Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Winter 2024/25
Abgabe: 12.11.2024
Rückgabe: ab 18.11.2024

Hausaufgabenblatt 1

Abgabe der Lösungen bis zum Dienstag, den 12.11.2024 um 13:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

Schreibe auf die Abgabe unbedingt deinen Namen, Matrikel- und Gruppennummer! Mehrere Blätter tackern!



Hausaufgabe 1 (Hamilton):

Betrachte die in Abbildung 1 dargestellten Graphen. Untersuche, welcher dieser Graphen einen Hamiltonkreis besitzt. Begründe deine Antworten für beide Graphen.

(5 Punkte)

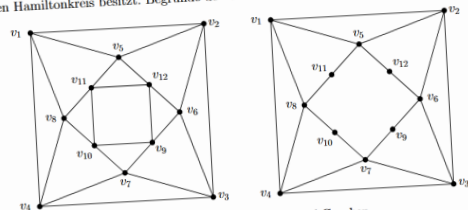


Abbildung 1: Abbildung von zwei Graphen.

Hausaufgabe 2 (Graphen):

Zeichne einen einfachen, zusammenhängenden Graphen mit $n = 7$ Knoten und $m = 9$ Kanten, sodass dieser eine Eulertour, aber keinen Hamiltonpfad bzw. Hamiltonkreis besitzt.

(3 Punkte)

Hausaufgaben

Einzelabgaben

5 Zettel

Aufwand: Überschaubar

Neues Blatt: Alle 2 Wochen

Bearbeitungszeit: 2 Wochen

Pro Blatt: 20 Punkte

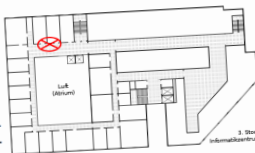
Algorithmen und Datenstrukturen
Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Winter 2024/25
Abgabe: 12.11.2024
Rückgabe: ab 18.11.2024

Hausaufgabenblatt 1

Abgabe der Lösungen bis zum Dienstag, den 12.11.2024 um 13:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

Schreibe auf die Abgabe unbedingt deinen Namen, Matrikel- und Gruppennummer! Mehrere Blätter tackern!



Hausaufgabe 1 (Hamilton):

Betrachte die in Abbildung 1 dargestellten Graphen. Untersuche, welcher dieser Graphen einen Hamiltonkreis besitzt. Begründe deine Antworten für beide Graphen.

(5 Punkte)

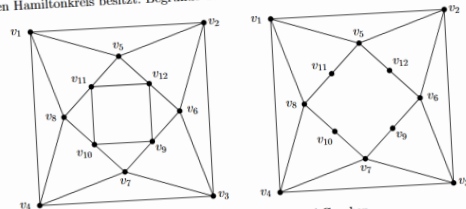


Abbildung 1: Abbildung von zwei Graphen.

Hausaufgabe 2 (Graphen):

Zeichne einen einfachen, zusammenhängenden Graphen mit $n = 7$ Knoten und $m = 9$ Kanten, sodass dieser eine Eulertour, aber keinen Hamiltonpfad bzw. Hamiltonkreis besitzt.

(3 Punkte)

Hausaufgaben

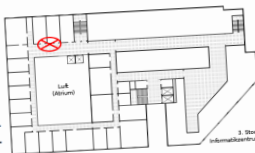
Algorithmen und Datenstrukturen
Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Winter 2024/25
Abgabe: 12.11.2024
Rückgabe: ab 18.11.2024

Hausaufgabenblatt 1

Abgabe der Lösungen bis zum Dienstag, den 12.11.2024 um 13:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

Schreibe auf die Abgabe unbedingt deinen Namen, Matrikel- und Gruppennummer! Mehrere Blätter tackern!



Hausaufgabe 1 (Hamilton):

Betrachte die in Abbildung 1 dargestellten Graphen. Untersuche, welcher dieser Graphen einen Hamiltonkreis besitzt. Begründe deine Antworten für beide Graphen.

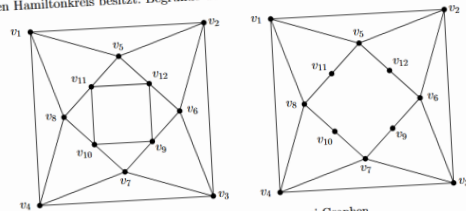


Abbildung 1: Abbildung von zwei Graphen.

Hausaufgabe 2 (Graphen):

Zeichne einen einfachen, zusammenhängenden Graphen mit $n = 7$ Knoten und $m = 9$ Kanten, sodass dieser eine Eulertour, aber keinen Hamiltonpfad bzw. Hamiltonkreis besitzt.

5 Zettel

Einzelabgaben

Aufwand: Überschaubar

Neues Blatt: Alle 2 Wochen

Bearbeitungszeit: 2 Wochen

Pro Blatt: 20 Punkte

Benötigt: 50 Punkte
zum Bestehen der Studienleistung

Hausaufgaben

Algorithmen und Datenstrukturen
Prof. Dr. Sándor P. Fekete
Ramin Kosfeld
Chek-Manh Loi

Winter 2024/25
Abgabe: 12.11.2024
Rückgabe: ab 18.11.2024

Hausaufgabenblatt 1

Abgabe der Lösungen bis zum Dienstag, den 12.11.2024 um 13:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet.

Schreibe auf die Abgabe unbedingt deinen Namen, Matrikel- und Gruppennummer! Mehrere Blätter tackern!



Hausaufgabe 1 (Hamilton):

(5 Punkte)
Betrachte die in Abbildung 1 dargestellten Graphen. Untersuche, welcher dieser Graphen einen Hamiltonkreis besitzt. Begründe deine Antworten für beide Graphen.

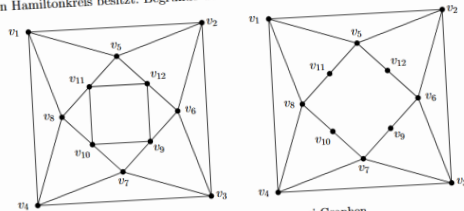


Abbildung 1: Abbildung von zwei Graphen.

Hausaufgabe 2 (Graphen):

(3 Punkte)
Zeichne einen einfachen, zusammenhängenden Graphen mit $n = 7$ Knoten und $m = 9$ Kanten, sodass dieser eine Eulertour, aber keinen Hamiltonpfad bzw. Hamiltonkreis besitzt.

5 Zettel

Einzelabgaben

Aufwand: Überschaubar

Neues Blatt: Alle 2 Wochen

Bearbeitungszeit: 2 Wochen

Pro Blatt: 20 Punkte

Unbenotet

Benötigt: 50 Punkte
zum Bestehen der Studienleistung

Wie gebe ich meine Hausaufgaben ab?

Wie gebe ich meine Hausaufgaben ab?

- Abgabe in Papierform

Wie gebe ich meine Hausaufgaben ab?



- Abgabe in Papierform
- Einwurf in HA-Schrank, 3. Stock IZ

Hausaufgabenschrank

Wie gebe ich meine Hausaufgaben ab?



- Abgabe in Papierform
- Einwurf in HA-Schrank, 3. Stock IZ

Hausaufgabenschrank

Wie gebe ich meine Hausaufgaben ab?

- Abgabe in Papierform
- Einwurf in HA-Schrank, 3. Stock IZ



Hausaufgabenschrank



Wie gebe ich meine Hausaufgaben ab?

- Abgabe in Papierform
- Einwurf in HA-Schrank, 3. Stock IZ



Hausaufgabenschrank



Wie gebe ich meine Hausaufgaben ab?

- Abgabe in Papierform
- Einwurf in HA-Schrank, 3. Stock IZ
- Vorsicht: Bis Freitag 9:00 Uhr!



Hausaufgabenschrank



Wie gebe ich meine Hausaufgaben ab?



Hausaufgabenschrank

- Abgabe in Papierform
- Einwurf in HA-Schrank, 3. Stock IZ
- Vorsicht: Bis Freitag 9:00 Uhr!
 - Danach: Korrektur, aber keine Punkte.



Hausaufgaben und Präsenzblätter

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung
- Rückgabe & Besprechung in den KÜ

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung
- Rückgabe & Besprechung in den KÜ
- Sie sind korrigiert:
Ihr seht eure Fehler und eure Punkte :)

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

6 freiwillige Präsenzblätter

- Studienleistung
- Rückgabe & Besprechung in den KÜ
- Sie sind korrigiert:
Ihr seht eure Fehler und eure Punkte :)

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung
- Rückgabe & Besprechung in den KÜ
- Sie sind korrigiert:
Ihr seht eure Fehler und eure Punkte :)

6 freiwillige Präsenzblätter

- Müssen nicht abgegeben werden :)

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung
- Rückgabe & Besprechung in den KÜ
- Sie sind korrigiert:
Ihr seht eure Fehler und eure Punkte :)

6 freiwillige Präsenzblätter

- Müssen nicht abgegeben werden :)
- Gemeinsam bearbeiten in den KÜ

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung
- Rückgabe & Besprechung in den KÜ
- Sie sind korrigiert:
Ihr seht eure Fehler und eure Punkte :)

6 freiwillige Präsenzblätter

- Müssen nicht abgegeben werden :)
- Gemeinsam bearbeiten in den KÜ
- Zusätzliche Vertiefung

Hausaufgaben und Präsenzblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung
- Rückgabe & Besprechung in den KÜ
- Sie sind korrigiert:
Ihr seht eure Fehler und eure Punkte :)

6 freiwillige Präsenzblätter

- Müssen nicht abgegeben werden :)
- Gemeinsam bearbeiten in den KÜ
- Zusätzliche Vertiefung
- Prüfungsvorbereitung

Hausaufgaben

Hausaufgaben

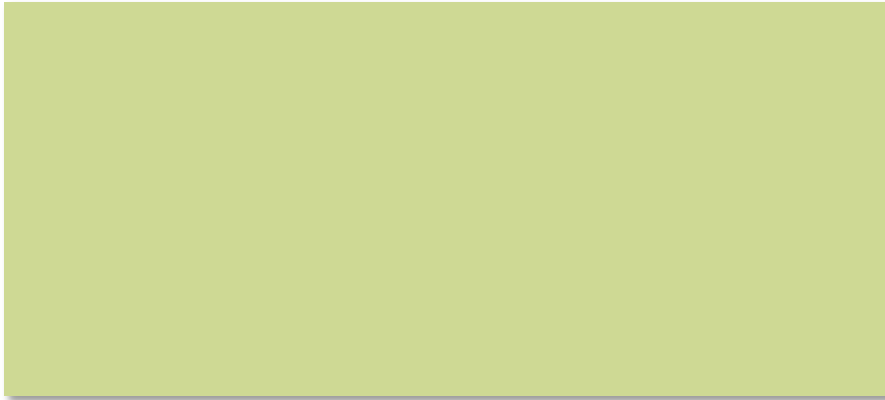
Wozu Hausaufgaben?

Die Hausaufgaben dienen euch zur Vorbereitung auf die Klausur.

Hausaufgaben

Wozu Hausaufgaben?

Die Hausaufgaben dienen euch zur Vorbereitung auf die Klausur.



Wozu Hausaufgaben?

Die Hausaufgaben dienen euch zur Vorbereitung auf die Klausur.

- Ideale Nachbereitung der Vorlesungsinhalte

Wozu Hausaufgaben?

Die Hausaufgaben dienen euch zur Vorbereitung auf die Klausur.

- Ideale Nachbereitung der Vorlesungsinhalte
- Zeitersparnis bei der Prüfungsvorbereitung

Wozu Hausaufgaben?

Die Hausaufgaben dienen euch zur Vorbereitung auf die Klausur.

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

Wozu Hausaufgaben?

Die Hausaufgaben dienen euch zur Vorbereitung auf die Klausur.

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

Wozu Hausaufgaben?

Die Hausaufgaben dienen euch zur Vorbereitung auf die Klausur.

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

- Zusammen überlegen ist gut, aber:
*Einzel*n aufschreiben und abgeben.
Sonst: **Plagiat** = keine Punkte.

Hausaufgaben

Hausaufgaben

- Alle wichtigen Infos stehen auch ...

Hausaufgaben

- Alle wichtigen Infos stehen auch ...
 - Teilweise auf den HA-Blättern selber

Hausaufgaben

- Alle wichtigen Infos stehen auch ...
 - Teilweise auf den HA-Blättern selber
 - Der Website

Hausaufgaben

- Alle wichtigen Infos stehen auch ...
 - Teilweise auf den HA-Blättern selber
 - Der Website
- „Hausaufgaben-Merkblatt“

Hausaufgaben

- Alle wichtigen Infos stehen auch ...
 - Teilweise auf den HA-Blättern selber
 - Der Website→ „Hausaufgaben-Merkblatt“
- Bei Fragen / Problemen mit den HA und den Abgaben:

Hausaufgaben

- Alle wichtigen Infos stehen auch ...
 - Teilweise auf den HA-Blättern selber
 - Der Website→ „Hausaufgaben-Merkblatt“
- Bei Fragen / Problemen mit den HA und den Abgaben:
 - Schreibt *direkt* euren Tutor*innen eine Mail, die können oft helfen :)

Homepage und Anmeldung

TU Braunschweig

Was suchen Sie?

15

STUD.IP

Vorlesung/Übung: Algorithmen und Datenstrukturen - Details

Aktionen

- Drucken
- Zugang zur Veranstaltung
- Nur im Stundenplan vormerken

Teilen

- Link zu dieser Veranstaltung kopieren

Die Anmeldung zu dieser Veranstaltung folgt speziellen Regeln. Lesen Sie den Hinweistext.

Allgemeine Informationen

Veranstaltungsname	Vorlesung/Übung: Algorithmen und Datenstrukturen
Veranstaltungsnummer	4227001
Semester	WiSe 2023/24
Aktuelle Anzahl der Teilnehmenden	0
erwartete Teilnehmendenanzahl	430
Heimat-Einrichtung	Abteilung Algorithmik (ALG)
Veranstaltungstyp	Vorlesung/Übung in der Kategorie Lehre

Stud.IP?



Homepage und Anmeldung

Unsere Kursseite: aud.ibr.cs.tu-bs.de

... Folien (ja, auch diese hier), Hausaufgaben, Altklausuren, **Übungsanmeldungen**:

The screenshot shows the homepage of the 'Algorithmen und Datenstrukturen' course for the winter semester 2023/24. The header is red with the course title and logo. A navigation bar includes links for 'Startseite', 'Veranstaltungen', 'Organisation', 'Kapitel', 'Knowledge Space', 'FAQ', 'Kontakt', and 'Archiv'. The main content area is divided into two columns. The left column, titled 'Startseite', contains a welcome message, a description of the course as a mandatory event for students in Informatics, and a list of topics: Algorithmenbegriff, Graphen, Suche in Graphen, Korrektheit und Komplexität von Algorithmen, and Datenstrukturen. The right column contains sections for 'Zeiten' (times) and 'Neuigkeiten' (news). The 'Zeiten' section lists lecture times (Dienstag 09:45-11:15 Uhr SN 19.1, Mittwoch 11:30-13:00 Uhr SN 19.1) and the first lecture date (25.10.2023). The 'Große Übung' (Large Exercise) section lists the time (Donnerstag 11:30-13:00 Uhr SN 19.1) and the first exercise date (02.11.2023). The 'Neuigkeiten' section mentions a mailing list and registration details.

Algorithmen und Datenstrukturen
WS 2023/24

[Startseite](#) [Veranstaltungen](#) [Organisation](#) [Kapitel](#) [Knowledge Space](#) [FAQ](#) [Kontakt](#) [Archiv](#)

Startseite

Algorithmen und Datenstrukturen

Die Vorlesung **Algorithmen und Datenstrukturen** ist eine Pflichtveranstaltung für Studierende der Informatik, Wirtschaftsinformatik, Informations- und Systemtechnik; außerdem ist sie wichtig und von Interesse für Studierende anderer Studiengänge, die mit Informatik zu tun haben.

Algorithmen sind das methodische Herz der theoretischen und praktischen Informatik; Datenstrukturen ermöglichen die effiziente Umsetzung von Algorithmen und den effizienten Zugriff auf Input- und Outputdaten. In dieser Einstiegsvorlesung werden die folgenden grundlegenden Begriffe erarbeitet:

- Algorithmenbegriff
- Graphen
- Suche in Graphen
- Korrektheit und Komplexität von Algorithmen
- Datenstrukturen

Zeiten

Vorlesung

Dienstag 09:45-11:15 Uhr SN 19.1
Mittwoch 11:30-13:00 Uhr SN 19.1
Erste Vorlesung: 25.10.2023

Große Übung

Donnerstag 11:30-13:00 Uhr SN 19.1
(unregelmäßig)
Erste Übung: 02.11.2023

Für Details siehe [Gesamtübersicht](#).

Neuigkeiten

Es gibt eine [Mailingliste](#). Bitte registriert euch dort mit eurer tu-bs-Adresse. Solltet

Homepage und Anmeldung

- Anmeldung für die kleinen Übungen auf der Kursseite
- Zuweisung erfolgt (voraussichtlich) am 02.11.26

Homepage und Anmeldung

- Anmeldung für die kleinen Übungen auf der Kursseite
- Zuweisung erfolgt (voraussichtlich) am 02.11.26

Kleine Übungen

Anmeldung zu den kleinen Übungen

Die Teilnahme an den kleinen Übungen ist Voraussetzung für das Bestehen der Studienleistung! Die Zuteilung in die Übungsgruppen (und damit auch die Teilnahme an den kleinen Übungen) erfolgt ausschließlich über dieses Formular – also tragt euch schnell ein 😊

Die Anmeldung für die kleinen Übungen ist bis einschließlich 29.10.24 geöffnet, danach ist die Anmeldung geschlossen.

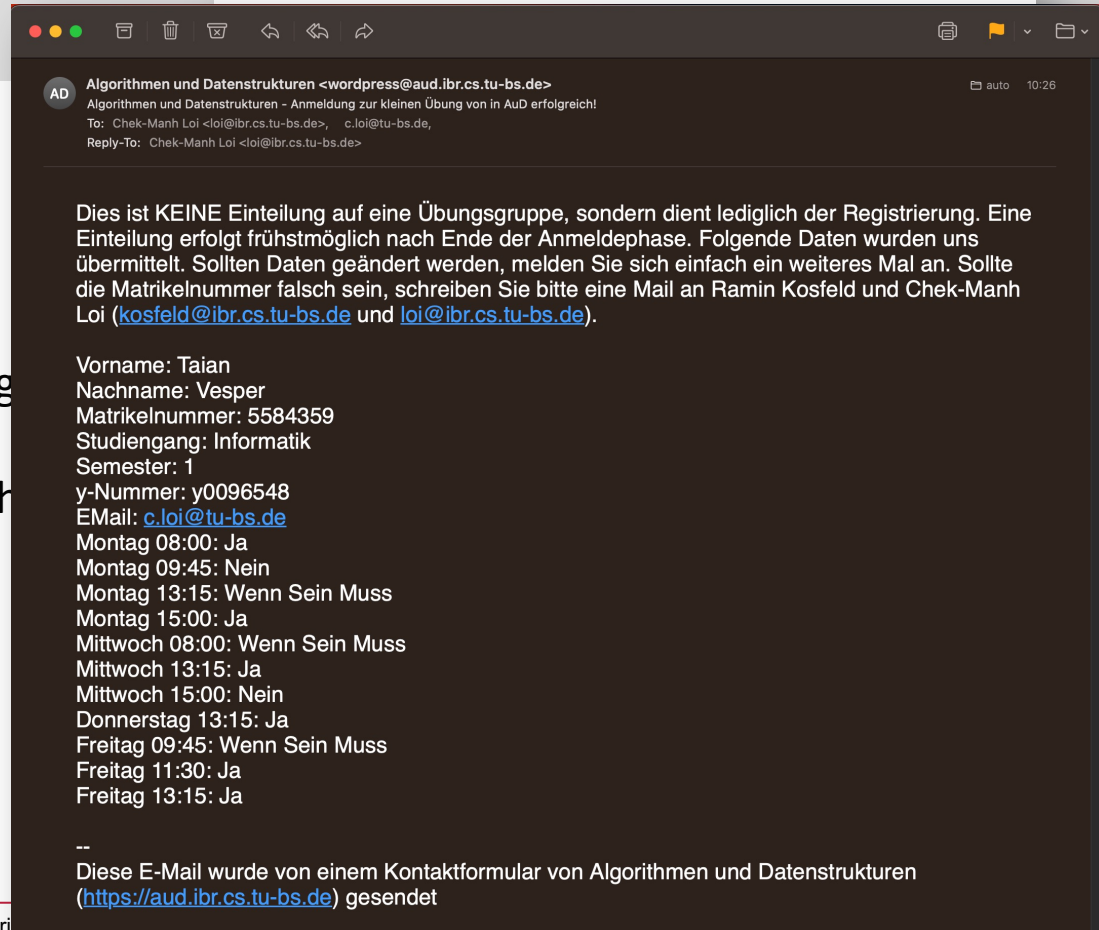
Bitte beachtet folgende Punkte:

- Bitte wählt **wirklich alle Termine, an denen ihr könnt**, damit wir am Ende eine faire Zuteilung garantieren können.
- Bitte nutzt eure **TU-Email!**
- Bei mehreren Anmeldung derselben Person wir werten immer **die neueste Anmeldung!**
 - Wenn ihr also **später eure Angaben korrigieren** müsst, schickt das Formular einfach erneut ab!
- Überprüft **vor** dem Absenden des Formulars, ob eure Daten (vor allem **Matrikelnummer und E-Mail**) korrekt sind.
- Bei erfolgreicher Anmeldung erhaltet ihr eine Mail mit einer Übersicht der eingegebenen Daten. Das ist noch keine Zuweisung der Gruppen!
- Die Einteilung in die Gruppen findet voraussichtlich am 30.10.24 statt. Ihr erhaltet dazu dann eine separate Mail.

Homepage und Anmeldung

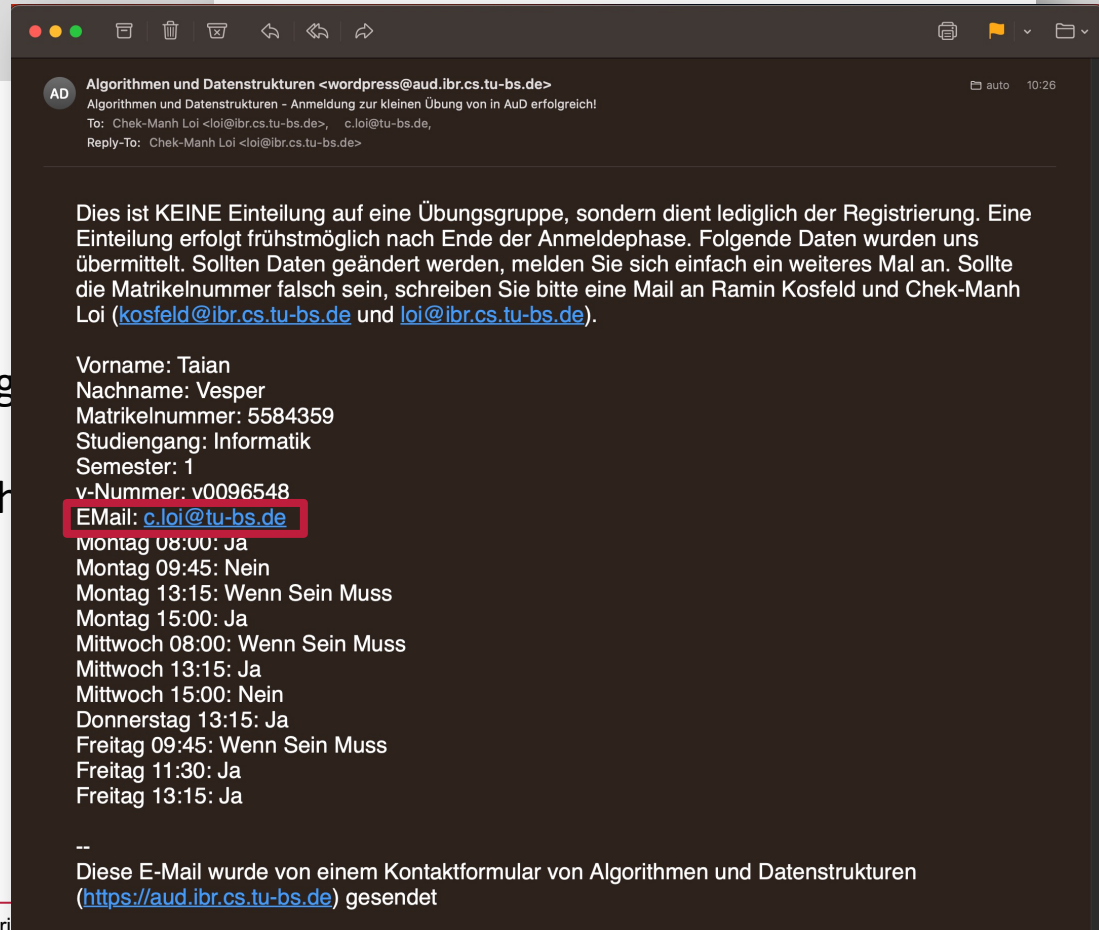
- Anmeldung für die kleinen Übungen auf der Kursseite
- Zuweisung erfolgt (voraussichtlich) 02.11.26

22.10.26 | Ramin Kosfeld und Chek-Manh Loi | Algori



Homepage und Anmeldung

- Anmeldung für die kleinen Übungen auf der Kursseite
- Zuweisung erfolgt (voraussichtlich) 02.11.26



Semesterplan

Semesterplan



Algorithmen und Datenstrukturen

Im Wintersemester 2024/25

[Startseite](#) [Vorlesungen](#) [Kleine Übung & Hausaufgaben](#) [Kapitel](#) [Knowledge Space](#) [Kontakt](#) [Archiv](#)

Startseite

Algorithmen und Datenstrukturen

Die Vorlesung **Algorithmen und Datenstrukturen** ist eine Pflichtveranstaltung für Studierende der Informatik, Wirtschaftsinformatik, Informations- und Systemtechnik; außerdem ist sie wichtig und von Interesse für Studierende anderer Studiengänge, die mit Informatik zu tun haben.

Algorithmen sind das methodische Herz der theoretischen und praktischen Informatik; Datenstrukturen ermöglichen die effiziente Umsetzung von Algorithmen und den effizienten Zugriff auf Input- und Outputdaten. In dieser Einstiegsvorlesung werden die folgenden grundlegenden Begriffe erarbeitet:

- Rekursionen
- Algorithmenbegriff
- Graphen
- Suche in Graphen
- Korrektheit und Komplexität von Algorithmen
- Datenstrukturen
- Sortieren

Anmeldung zur Veranstaltung

Diese Veranstaltung verwendet *kein* Stud.IP, entsprechend muss sich für den Besuch der Vorlesung in keine Veranstaltung eingetragen werden. Sehr wohl müssen aber später die Teilnahme in der kleinen Übung sowie die Teilnahme an der Klausur angemeldet werden, das ist wichtig!

Organisation

Vorlesung

Dienstag 09:45-11:15 Uhr SN 19.1
Mittwoch 11:30-13:00 Uhr SN 19.1
Erste Vorlesung: 16.10.2024

Große Übung

Donnerstag 11:30-13:00 Uhr UP 3.007 (Bunker)
Erste Übung: 21.10.2024

Kleine Übung (hier klicken)

Erste Woche mit kleinen Übungen: ab 04.11.2024

Klausur

Alle Infos zur Klausur geben wir im Laufe des Semesters bekannt.

Wann findet welche Vorlesung, große oder kleine Übung statt?

Semesterplan (hier klicken)
Das ist die offizielle Quelle zu allen Terminen der Veranstaltung und wird von uns immer aktuell gehalten.

Semesterplan



Semesterplan AuD WS26/27

Woche (KW)	Woche (Datum)	Vorlesung (Di./Mi.)	Gr. Übung (Do.)	Kl. Übung	HA Ausgabe (Fr.)	HA Abgabe (bis Fr. 09:00)	HA und Präsenz Besprechung
43	19.10.	0,1	0				
44	26.10.	2,3					
45	02.11.	4,5	1		HA1		
46	09.11.	6,7		0			P0
47	16.11.	8,9	2		HA2	HA1	
48	23.11.	10,11		1			HA1+P1
49	30.11.	12,13	3		HA3	HA2	
50	07.12.	14,15		2			HA2+P2
51	14.12.	16,17	4		HA4	HA3	
52	21.12.	/,/					
1	28.12.	Weihnachtsferien					
2	04.01.	/,/					
3	11.01.	18,19	5	3	HA5	HA4	HA3+P3
4	18.01.	20,21		4			HA4+P4
5	25.01.	22,23	6			HA5	
6	01.02.	24,25	7	5			HA5+P5

Organisation

Vorlesung

Dienstag 09:45-11:15 Uhr SN 19.1
Mittwoch 11:30-13:00 Uhr SN 19.1
Erste Vorlesung: 16.10.2024

Große Übung

Donnerstag 11:30-13:00 Uhr UP 3.007 (Bunker)
Erste Übung: 21.10.2024

Kleine Übung (hier klicken)

Erste Woche mit kleinen Übungen: ab 04.11.2024

Klausur

Alle Infos zur Klausur geben wir im Laufe des Semesters bekannt.

Wann findet welche Vorlesung, große oder kleine Übung statt?

Semesterplan (hier klicken)

Dies ist die offizielle Quelle zu allen Terminen der Veranstaltung und wird von uns immer aktuell gehalten.

Mailingliste

- Anmeldungslink auf der Homepage
- Für Informationen wie
 - Raumänderungen,
 - Ausfälle,
 - etc.
- Möglichkeit für Fragen
- Checkt regelmäßig eure Mails!
(Generelles Ding an der Uni)

The screenshot shows the Postorius web interface for a mailing list named 'Aud' with the email address 'aud@ibr.cs.tu-bs.de'. The interface includes a navigation bar with 'Postorius', 'Lists', and 'Archives' links, and a 'Login' / 'Sign Up' button. The main content area has a 'Summary' section with the description 'Mailingliste für Algorithmen und Datenstrukturen' and contact information 'aud-owner@ibr.cs.tu-bs.de'. Below this is a 'Subscription / Unsubscription' section with a 'Log In' button and a form for subscribing without an account, including fields for 'Your email address' and 'Your name (optional)', and a 'Subscribe' button. The footer contains links to 'Postorius Documentation', 'GNU Mailman', and the version 'Postorius Version 1.3.6'.

Postorius Lists Archives Login Sign Up

Aud aud@ibr.cs.tu-bs.de

Summary

Mailingliste für Algorithmen und Datenstrukturen

To contact the list owners, use the following email address: aud-owner@ibr.cs.tu-bs.de

You have to login to visit the archives of this list.

Subscription / Unsubscription

To subscribe or unsubscribe from this list, please log in first. If you have not previously logged in, you may need to set up an account with the appropriate email address.

[Log In](#)

You can also subscribe without creating an account. If you wish to do so, please use the form below.

Your email address

Your name (optional)

[Subscribe](#)

[Postorius Documentation](#) • [GNU Mailman](#) • Postorius Version 1.3.6

Mailingliste

- Anmeldungslink auf der Homepage
- Für Informationen wie
 - Raumänderungen,
 - Ausfälle,
 - etc.
- Möglichkeit für Fragen
- Checkt regelmäßig eure (Generelles Ding an der



Postorius Lists Archives Login Sign Up

Aud aud@ibr.cs.tu-bs.de

uren
email address: aud-owner@ibr.cs.tu-bs.de
list.

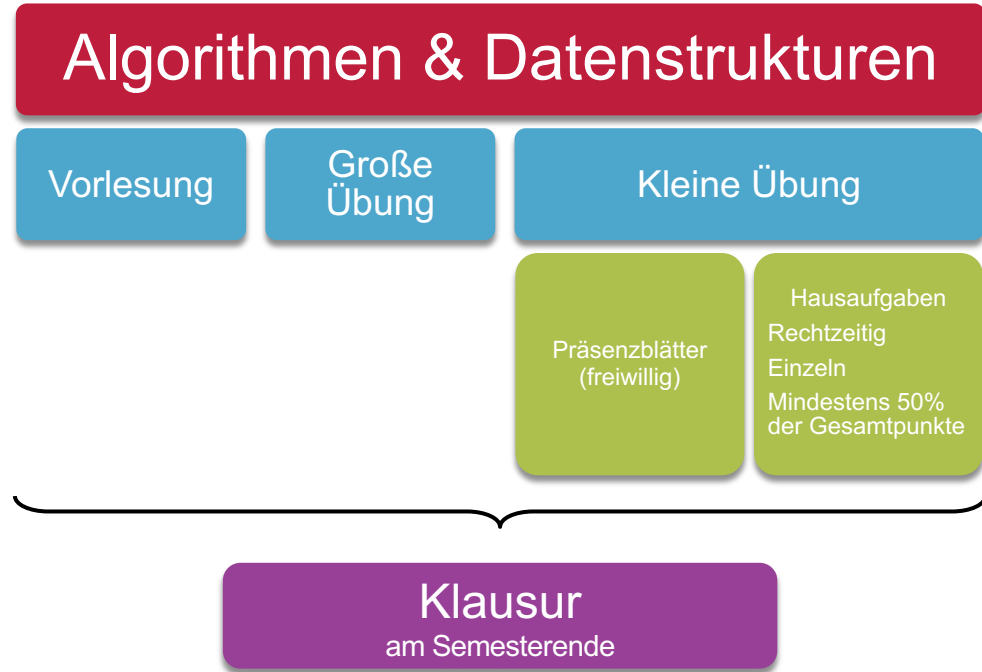
Description
Please log in first. If you have not previously logged in, you may need to set up an account with the appropriate email address.

account. If you wish to do so, please use the form below.

Postorius Documentation • GNU Mailman • Postorius Version 1.3.6

Zusammenfassung

- Kursseite
 - Semesterplan
 - Anmeldung kleine Übung
- Kommunikation
 - In Person
 - Mailingliste!
 - Mail an uns beide



Fragen



Fragen

Stellt Eure Fragen ...



Fragen

Stellt Eure Fragen ...

- Während oder nach den Vorlesungen und Übungen



Fragen

Stellt Eure Fragen ...

- Während oder nach den Vorlesungen und Übungen
- Euren Tutoren in den kleinen Übungen!



Fragen

Stellt Eure Fragen ...

- Während oder nach den Vorlesungen und Übungen
- Euren Tutoren in den kleinen Übungen!
- Über die Mailingliste (ins Plenum)



Fragen

Stellt Eure Fragen ...

- Während oder nach den Vorlesungen und Übungen
- Euren Tutoren in den kleinen Übungen!
- Über die Mailingliste (ins Plenum)
- Per Mail (kosfeld@ibr.cs.tu-bs.de & loi@ibr.cs.tu-bs.de)
 - Sprechstunde von Ramin und Chek-Manh (nach Vereinbarung)

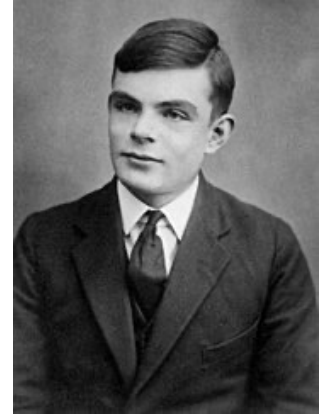


Fragen?

Algorithmen & Pseudocode



Donald Knuth



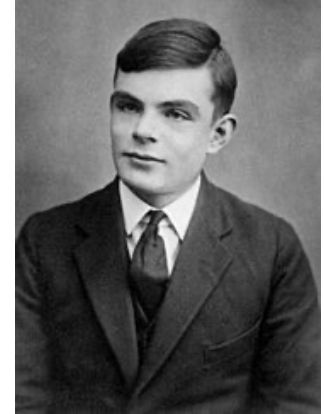
Alan Turing

Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems



Donald Knuth



Alan Turing

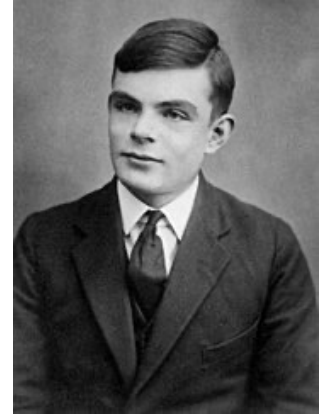
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

- Eigenschaften



Donald Knuth



Alan Turing

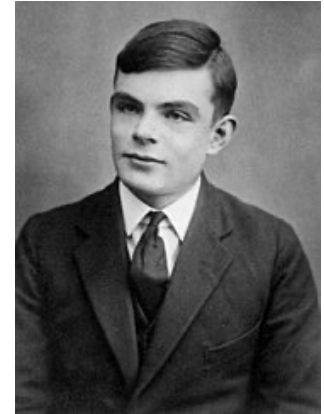
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

- Eigenschaften
 - Ausführbarkeit



Donald Knuth



Alan Turing

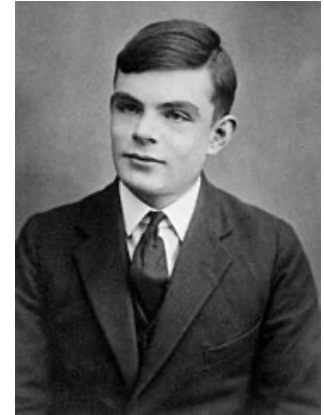
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

- Eigenschaften
 - Ausführbarkeit
 - Endlichkeit



Donald Knuth



Alan Turing

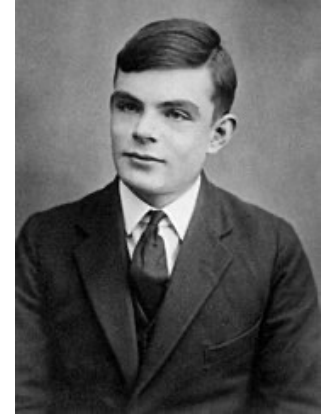
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

- Eigenschaften
 - Ausführbarkeit
 - Endlichkeit
 - Endliche Ausführzeit



Donald Knuth



Alan Turing

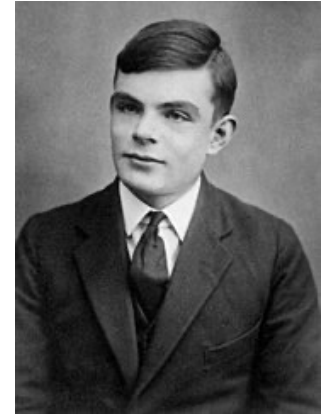
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

- Eigenschaften
 - Ausführbarkeit
 - Endlichkeit
 - Endliche Ausführzeit
 - Endlicher Speicherbedarf



Donald Knuth



Alan Turing

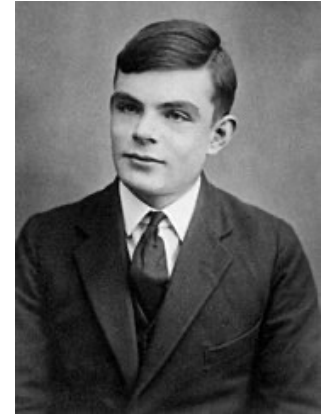
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

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



Donald Knuth



Alan Turing

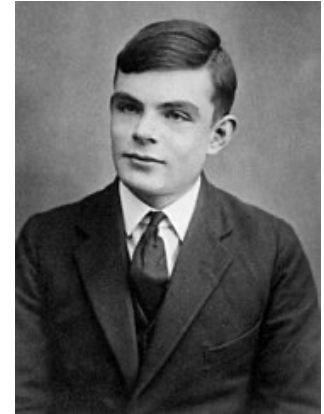
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

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



Donald Knuth



Alan Turing

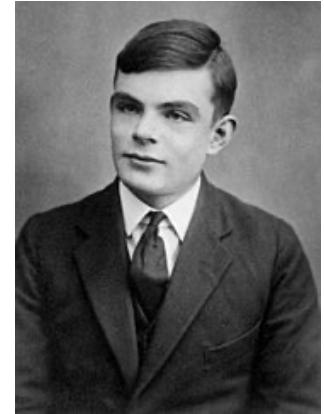
Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

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



Donald Knuth



Alan Turing

Pseudocode

Pseudocode

Pseudocode

Wörterbuch



pseu·do

/pseüdo/

Adjektiv UMGANGSSPRACHLICH

nicht echt, nur nachgemacht, nachgeahmt

Pseudocode

Wörterbuch



pseu·do

/pseüdo/

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 HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Warum Pseudocode?

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Ist dieser Code effizient zu lesen?

Warum Pseudocode?

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Ist dieser Code effizient zu lesen?

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

Warum Pseudocode?

Java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Pseudocode

```
function main(args)  
    print Hello World!  
end function
```

Beispiel: Euklidischer Algorithmus

Beispiel: Euklidischer Algorithmus

Algorithmus zur Berechnung des größten gemeinsamen Teilers zwei ganzer Zahlen

Beispiel: Euklidischer Algorithmus

Algorithmus zur Berechnung des größten gemeinsamen Teilers zwei ganzer Zahlen

Als 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.

Beispiel: Euklidischer Algorithmus

Algorithmus zur Berechnung des größten gemeinsamen Teilers zwei ganzer Zahlen

Als Pseudocode

Als 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.

Beispiel: Euklidischer Algorithmus

Algorithmus zur Berechnung des größten gemeinsamen Teilers zwei ganzer Zahlen

Als 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.

Als Pseudocode

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

Beispiel: Euklidischer Algorithmus

Algorithmus zur Berechnung des größten gemeinsamen Teilers zwei ganzer Zahlen

Als 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.

$$7 \bmod 3 = 1,$$

Als Pseudocode

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

Beispiel: Euklidischer Algorithmus

Algorithmus zur Berechnung des größten gemeinsamen Teilers zwei ganzer Zahlen

Als 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.

$$7 \bmod 3 = 1,$$

$$7 = 2 * 3 + 1$$

Als Pseudocode

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

Beispiel: Euklidischer Algorithmus

Algorithmus zur Berechnung des größten gemeinsamen Teilers zwei ganzer Zahlen

Als 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.

$$7 \bmod 3 = 1,$$

$$7 = 2 * 3 + 1$$

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

Als Pseudocode

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

Beispiel: Euklidischer Algorithmus

Pseudocode

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



Beispiel: Euklidischer Algorithmus

Pseudocode

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



Schlüsselwörter

Beispiel: Euklidischer Algorithmus

Pseudocode

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



Schlüsselwörter

- Anlehnung an höhere Programmiersprachen

Beispiel: Euklidischer Algorithmus

Pseudocode

```
function Euklid( $a, b$ )  
  while  $b \neq 0$  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.)

Beispiel: Euklidischer Algorithmus

Pseudocode

```
function Euklid( $a, b$ )  
  while  $b \neq 0$  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.)
- Können sowohl auf Deutsch als auch auf Englisch benutzt werden

Pseudocode - Bausteine

Pseudocode - Bausteine

Methoden

```
function NAME (Param1, Param2, ...)  
    ...  
end function
```

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)
```

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
```

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  $a$  zurück und beendet die Funktion)
```

Pseudocode - Bausteine

Pseudocode - Bausteine

Schleifen - 1

while Bedingung **do**

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

end while

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

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 $a := b$

a wird nach jeder Iteration um 1 erhöht. ($a := a + 1$)

Wiederhole bis a den Wert c erreicht hat.

end for

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 $a := b$

a wird nach jeder Iteration um 1 erhöht. ($a := a + 1$)

Wiederhole bis a den Wert c erreicht hat.

end for

Schleifen - 4

for $a := b$ **down to** c

Starte mit $a := b$

a wird in jeder Iteration um 1 verringert. ($a := a - 1$)

Wiederhole bis a den Wert c erreicht hat.

end for

Pseudocode - Varianten

Pseudocode - Varianten

- Auf Deutsch
 - Wenn ... Dann ... Sonst ...
 - Solange ...
 - Für $i \leftarrow 1, \dots, k$...

Pseudocode - Varianten

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

Pseudocode - Varianten

- Auf Deutsch
 - Wenn ... Dann ... Sonst ...
 - Solange ...
 - Für $i \leftarrow 1, \dots, k$...
- Statt $\leftarrow$ schreibt man gelegentlich $:=$ oder sogar $=$
- Mischung aus Prosa und Pseudocode:
 1. **function** Verhältnis(*liste*)
 2. Sortiere die Zahlen in *liste* aufsteigend
 3. **return** *liste*[1] / *liste*[länge(*liste*)]
 4. **end function**

Pseudocode - Varianten

- Auf Deutsch
 - Wenn ... Dann ... Sonst ...
 - Solange ...
 - Für $i \leftarrow 1, \dots, k$...
- Statt $\leftarrow$ schreibt man gelegentlich $:=$ oder sogar $=$
- Mischung aus Prosa und Pseudocode:
 1. **function** Verhältnis(*liste*)
 2. Sortiere die Zahlen in *liste* aufsteigend
 3. **return** *liste*[1] / *liste*[länge(*liste*)]
 4. **end function**

self driving cars aren't even hard to make lol
just program it not to hit stuff

```
if(goingToHitStuff)
{
    don't();
}
```

(so bitte nicht.)

Pseudocode - Varianten

- Auf Deutsch
 - Wenn ... Dann ... Sonst ...
 - Solange ...
 - Für $i \leftarrow 1, \dots, k$...
- Statt $\leftarrow$ schreibt man gelegentlich $:=$ oder sogar $=$
- Mischung aus Prosa und Pseudocode:
 1. **function** Verhältnis(*liste*)
 2. Sortiere die Zahlen in *liste* aufsteigend
 3. **return** *liste*[1] / *liste*[länge(*liste*)]
 4. **end function**
- end... statements dürfen weggelassen werden, aber Einrückung muss korrekt sein!

self driving cars aren't even hard to make lol
just program it not to hit stuff

```
if(goingToHitStuff)
{
    don't();
}
```

(so bitte nicht.)

Pseudocode - Varianten

- Auf Deutsch
 - Wenn ... Dann ... Sonst ...
 - Solange ...
 - Für $i \leftarrow 1, \dots, k$...
- Statt $\leftarrow$ schreibt man gelegentlich $:=$ oder sogar $=$
- Mischung aus Prosa und Pseudocode:
 1. **function** Verhältnis(*liste*)
 2. Sortiere die Zahlen in *liste* aufsteigend
 3. **return** *liste*[1] / *liste*[länge(*liste*)]
 4. **end function**
- end... statements dürfen weggelassen werden, aber Einrückung muss korrekt sein!

Wichtig

Pseudocode muss immer verständlich und eindeutig sein!

self driving cars aren't even hard to make lol
just program it not to hit stuff

```
if(goingToHitStuff)
{
    don't();
}
```

(so bitte nicht.)

Pseudocode - Varianten

- Auf Deutsch
 - Wenn ... Dann ... Sonst ...
 - Solange ...
 - Für $i \leftarrow 1, \dots, k$...
- Statt $\leftarrow$ schreibt man gelegentlich $:=$ oder sogar $=$
- Mischung aus Prosa und Pseudocode:
 1. **function** Verhältnis(*liste*)
 2. Sortiere die Zahlen in *liste* aufsteigend
 3. **return** *liste*[1] / *liste*[länge(*liste*)]
 4. **end function**
- end... statements dürfen weggelassen werden, aber Einrückung muss korrekt sein!

Wichtig

Pseudocode muss immer verständlich und eindeutig sein!

... und konsistent!

self driving cars aren't even hard to make lol
just program it not to hit stuff

```
if(goingToHitStuff)
{
    don't();
}
```

(so bitte nicht.)

Problem und Instanz

Problem

Instanz

Problem und Instanz

Problem

Instanz

- Allgemeine Fragestellung

Problem und Instanz

Problem

Instanz

- Allgemeine Fragestellung
- Lösung: Angabe eines Algorithmus

Problem und Instanz

Problem

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

Instanz

Problem und Instanz

Problem



Instanz

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

Problem und Instanz

Problem



Instanz

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

Problem und Instanz

Problem



Instanz

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

- Konkrete Fragestellung (Zahlen etc.)
- Lösung: Angabe einer konkreten Ausgabe

Problem und Instanz

Problem

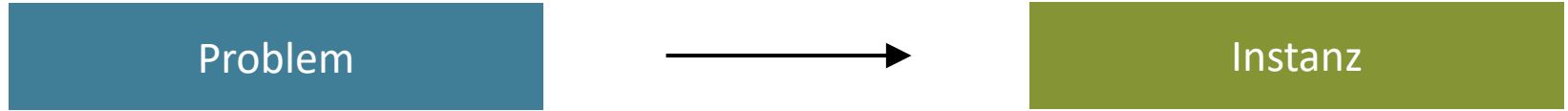


Instanz

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

- Konkrete Fragestellung (Zahlen etc.)
- Lösung: Angabe einer konkreten Ausgabe
- Formuliert durch eine konkrete Eingabe eines bestimmten Problems

Problem und Instanz Beispiel



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 (gerade gesehen)

Problem und Instanz Beispiel

Problem

Größter gemeinsamer Teiler zweier Zahlen

Eingabe: Zwei Zahlen a und b .

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

Lösung: Euklidischer Algorithmus (gerade gesehen)



Instanz

Größter gemeinsamer Teiler zweier Zahlen

- $ggT(5, 102)$; **Lösung:** 1
- $ggT(8, 64)$; **Lösung:** 8
- ...

Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
5. **end for**
6. **return** c
7. **end function**

Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
5. **end for**
6. **return** c
7. **end function**

Instanz: Berechne das Produkt von 9 und 3

Aufruf: Mult(9, 3)

Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
5. **end for**
6. **return** c
7. **end function**

Variablen nach jeder Iteration der for-Schleife:

Instanz: Berechne das Produkt von 9 und 3

Aufruf: Mult(9, 3)

Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
5. **end for**
6. **return** c
7. **end function**

Variablen nach jeder Iteration der for-Schleife:

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

Instanz: Berechne das Produkt von 9 und 3

Aufruf: Mult(9, 3)

Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
5. **end for**
6. **return** c
7. **end function**

Variablen nach jeder Iteration der for-Schleife:

Iteration	a	b	c	d
1	9	3	9	3

Instanz: Berechne das Produkt von 9 und 3

Aufruf: Mult(9, 3)

Pseudocode – In Aktion

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

Variablen nach jeder Iteration der for-Schleife:

Iteration	a	b	c	d
1	9	3	9	3
2	9	3	18	2

Instanz: Berechne das Produkt von 9 und 3

Aufruf: Mult(9, 3)

Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
5. **end for**
6. **return** c
7. **end function**

Instanz: Berechne das Produkt von 9 und 3

Aufruf: Mult(9, 3)

Variablen nach jeder Iteration der for-Schleife:

Iteration	a	b	c	d
1	9	3	9	3
2	9	3	18	2
3	9	3	27	1

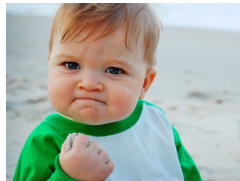
Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
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
1	9	3	9	3
2	9	3	18	2
3	9	3	27	1

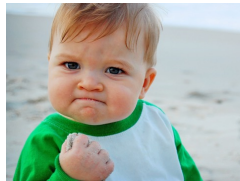
Pseudocode – In Aktion

1. **function** Mult(a, b)
2. $c \leftarrow 0$
3. **for** $d \leftarrow b$ **downto** 1
4. $c \leftarrow c + a$
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
1	9	3	9	3
2	9	3	18	2
3	9	3	27	1

*Das ist hier nur eine Demo für Pseudocode, natürlich dürft ihr in eurem Pseudocode einfach direkt Multiplikationen hinschreiben.

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )  
2.   while  $b \neq 0$  do  
3.      $h \leftarrow a \bmod b$   
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)

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

Zeile	Iteration	a	b	h
-------	-----------	-----	-----	-----

Pseudocode – In Aktion

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

Zeile	Iteration	a	b	h
1	-	49	21	-

Instanz: Berechne ggT von $a = 49, b = 21$

Aufruf: Euklid(49, 21)

Pseudocode – In Aktion

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

Zeile	Iteration	a	b	h
1	-	49	21	-
2	-	49	21	-

Instanz: Berechne ggT von $a = 49, b = 21$

Aufruf: Euklid(49, 21)

Pseudocode – In Aktion

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

Zeile	Iteration	a	b	h
1	-	49	21	-
2	-	49	21	-
3	1	49	21	7

Instanz: Berechne ggT von $a = 49, b = 21$

Aufruf: Euklid(49, 21)

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

Zeile	Iteration	a	b	h
1	-	49	21	-
2	-	49	21	-
3	1	49	21	7
4	1	21	21	7

Pseudocode – In Aktion

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

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

Instanz: Berechne ggT von $a = 49, b = 21$

Aufruf: Euklid(49, 21)

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

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

Pseudocode – In Aktion

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

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

Instanz: Berechne ggT von $a = 49, b = 21$

Aufruf: Euklid(49, 21)

Pseudocode – In Aktion

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

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

Instanz: Berechne ggT von $a = 49, b = 21$

Aufruf: Euklid(49, 21)

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

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

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

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

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

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

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

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

Pseudocode – In Aktion

```
1. function Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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)

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 Euklid( $a, b$ )
2.   while  $b \neq 0$  do
3.      $h \leftarrow a \bmod b$ 
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** AbsoluteSum(a, b)
2. **return** Absolute(a) + Absolute(b)
3. **end function**

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

1. **function** AbsoluteSum(a, b)
2. **return** Absolute(a) + Absolute(b)
3. **end function**

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 AbsoluteSum(a, b)
2.    return Absolute(a) + Absolute(b)
3.  end function
```

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 AbsoluteSum(a, b)  
2.   return Absolute(a) + Absolute(b)  
3. end function
```

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

1. **function** Absolute(a)
2. **if** $a < 0$ **then**
3. $a \leftarrow -a$
4. **end if**
5. **return** a
6. **end function**

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	a (Absolute)	b
------------------------	---------------------	----------------------	-------------------	-----

1. **function** AbsoluteSum(a, b)
2. **return** Absolute(a) + Absolute(b)
3. **end function**

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

```
1. function Absolute(a)
2.   if  $a < 0$  then
3.      $a \leftarrow -a$ 
4.   end if
5.   return  $a$ 
6. end function
```

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	a (Absolute)	b
1	-	-10	-	3

```
1. function AbsoluteSum( $a, b$ )
2.   return Absolute( $a$ ) + Absolute( $b$ )
3. end function
```

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

1. **function** Absolute(a)
2. **if** $a < 0$ **then**
3. $a \leftarrow -a$
4. **end if**
5. **return** a
6. **end function**

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	a (Absolute)	b
1	-	-10	-	3
2	1	-10	-10	3

1. **function** AbsoluteSum(a, b)
2. **return** Absolute(a) + Absolute(b)
3. **end function**

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

```
1. function Absolute(a)
2.   if  $a < 0$  then
3.      $a \leftarrow -a$ 
4.   end if
5.   return  $a$ 
6. end function
```

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	a (Absolute)	b
1	-	-10	-	3
2	1	-10	-10	3
2	2	-10	-10	3

```
1. function AbsoluteSum( $a, b$ )
2.   return Absolute( $a$ ) + Absolute( $b$ )
3. end function
```

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

```
1. function Absolute(a)
2.   if  $a < 0$  then
3.      $a \leftarrow -a$ 
4.   end if
5.   return  $a$ 
6. end function
```

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	a (Absolute)	b
1	-	-10	-	3
2	1	-10	-10	3
2	2	-10	-10	3
2	3	-10	10	3

```
1. function AbsoluteSum( $a, b$ )
2.   return Absolute( $a$ ) + Absolute( $b$ )
3. end function
```

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

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

```
1. function AbsoluteSum(a, b)
2.   return Absolute(a) + Absolute(b)
3. end function
```

Zeile (AbsoluteSum)	Zeile (Absolute)	<i>a</i> (AbsoluteSum)	<i>a</i> (Absolute)	<i>b</i>
1	-	-10	-	3
2	1	-10	-10	3
2	2	-10	-10	3
2	3	-10	10	3
2	4	-10	10	3

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

```
1. function Absolute(a)
2.   if  $a < 0$  then
3.      $a \leftarrow -a$ 
4.   end if
5.   return  $a$ 
6. end function
```

```
1. function AbsoluteSum( $a, b$ )
2.   return Absolute( $a$ ) + Absolute( $b$ )
3. end function
```

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	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

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

1. **function** Absolute(a)

2. **if** $a < 0$ **then**

3. $a \leftarrow -a$

4. **end if**

5. **return** a

6. **end function**

1. **function** AbsoluteSum(a, b)

2. **return** Absolute(a) + Absolute(b)

3. **end function**

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	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

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

1. **function** Absolute(a)

2. **if** $a < 0$ **then**

3. $a \leftarrow -a$

4. **end if**

5. **return** a

6. **end function**

1. **function** AbsoluteSum(a, b)

2. **return** Absolute(a) + Absolute(b)

3. **end function**

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	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

Aufruf: AbsoluteSum(-10, 3)

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

```
1. function Absolute(a)
2.   if  $a < 0$  then
3.      $a \leftarrow -a$ 
4.   end if
5.   return  $a$ 
6. end function
```

```
1. function AbsoluteSum( $a, b$ )
2.   return Absolute( $a$ ) + Absolute( $b$ )
3. end function
```

Aufruf: AbsoluteSum(-10, 3)

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	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

Pseudocode – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

1. **function** Absolute(a)

2. **if** $a < 0$ **then**

3. $a \leftarrow -a$

4. **end if**

5. **return** a

6. **end function**

1. **function** AbsoluteSum(a, b)

2. **return** Absolute(a) + Absolute(b)

3. **end function**

Aufruf: AbsoluteSum(-10, 3)

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	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 – In Aktion

Der absolute Wert einer Zahl ist sein positiver Wert:

$$\text{abs}(-5) = 5, \quad \text{abs}(11) = 11, \quad \text{abs}(0) = 0$$

1. **function** Absolute(a)

2. **if** $a < 0$ **then**

3. $a \leftarrow -a$

4. **end if**

5. **return** a

6. **end function**

1. **function** AbsoluteSum(a, b)

2. **return** Absolute(a) + Absolute(b)

3. **end function**

Aufruf: AbsoluteSum(-10, 3)

Rückgabe: 13

Zeile (AbsoluteSum)	Zeile (Absolute)	a (AbsoluteSum)	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

Merkzettel Pseudocode

(Auf der Kursseite verlinkt)



Merkzettel ▷ Pseudocode

Version – 12. April 2021

Pseudocode dient der übersichtlichen und eindeutigen Darstellung von Algorithmen. Er ist meist an moderne höhere Programmiersprachen angelehnt, kann aber auch mit natürlicher Sprache gemischt werden. Pseudocode verzichtet auf technische Besonderheiten realer Programmiersprachen und ist dadurch kompakter, sollte sich aber dennoch ohne viel Aufwand in eine reale Implementierung überführen lassen.

Methoden

Funktionale Einheiten von Code werden zu benannten Blöcken (Funktionen) zusammengefasst.

- Funktions-Körper:
function NAME(Parameter1, Parameter2,...)
...
end function

Ein einfacher Algorithmus für den Logarithmus

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: $x = \log_b n$)

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?

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

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. $\log \leftarrow 0$
3. **while** $n > 1$ **do**
4. $n \leftarrow n/b$
5. $\log \leftarrow \log + 1$
6. **return** $\log$

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. $\log \leftarrow 0$
3. **while** $n > 1$ **do**
4. $n \leftarrow n/b$
5. $\log \leftarrow \log + 1$
6. **return** $\log$

Beispiel: Log(64, 2)

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?

log	n
-----	-----

Betrachten wir zunächst den einfachen Fall: x ist ganzzahlig

1. **function** Log(n, b)
2. $\log \leftarrow 0$
3. **while** $n > 1$ **do**
4. $n \leftarrow n/b$
5. $\log \leftarrow \log + 1$
6. **return** log

Beispiel: Log(64, 2)

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: $x = \log_b n$)

log	n
0	64

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. $\log \leftarrow 0$
3. **while** $n > 1$ **do**
4. $n \leftarrow n/b$
5. $\log \leftarrow \log + 1$
6. **return** $\log$

Beispiel: Log(64, 2)

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?

log	n
0	64
1	32

Betrachten wir zunächst den einfachen Fall: x ist ganzzahlig

1. **function** Log(n, b)
2. $\log \leftarrow 0$
3. **while** $n > 1$ **do**
4. $n \leftarrow n/b$
5. $\log \leftarrow \log + 1$
6. **return** $\log$

Beispiel: Log(64, 2)

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?

log	n
0	64
1	32
2	16

Betrachten wir zunächst den einfachen Fall: x ist ganzzahlig

1. **function** Log(n, b)
2. $\text{log} \leftarrow 0$
3. **while** $n > 1$ **do**
4. $n \leftarrow n/b$
5. $\text{log} \leftarrow \text{log} + 1$
6. **return** log

Beispiel: Log(64, 2)

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.    $\log \leftarrow 0$ 
3.   while  $n > 1$  do
4.      $n \leftarrow n/b$ 
5.      $\log \leftarrow \log + 1$ 
6.   return  $\log$ 
```

Beispiel: $\text{Log}(64, 2)$

log	n
0	64
1	32
2	16
3	8

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.    $\log \leftarrow 0$ 
3.   while  $n > 1$  do
4.      $n \leftarrow n/b$ 
5.      $\log \leftarrow \log + 1$ 
6.   return  $\log$ 
```

Beispiel: $\text{Log}(64, 2)$

log	n
0	64
1	32
2	16
3	8
4	4

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.    $\log \leftarrow 0$ 
3.   while  $n > 1$  do
4.      $n \leftarrow n/b$ 
5.      $\log \leftarrow \log + 1$ 
6.   return  $\log$ 
```

Beispiel: $\text{Log}(64, 2)$

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

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.    $\log \leftarrow 0$ 
3.   while  $n > 1$  do
4.      $n \leftarrow n/b$ 
5.      $\log \leftarrow \log + 1$ 
6.   return  $\log$ 
```

Beispiel: $\text{Log}(64, 2)$

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

Ein Algorithmus für den Logarithmus

Ein Algorithmus für den Logarithmus

Was passiert, wenn x nicht ganzzahlig wird?

Problem: Irrationale Ergebnisse

Lösung: Bestimme den Logarithmus nur auf k Stellen genau.

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.

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.

$$\log_b n$$

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.

$$10^k \log_b n$$

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.

$$10^k \log_b n = \log_b (n^{(10^k)})$$

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

$$10^k \log_b n = \log_b (n^{(10^k)})$$

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

Problem 1: Schon für kleine k wird n sehr groß

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

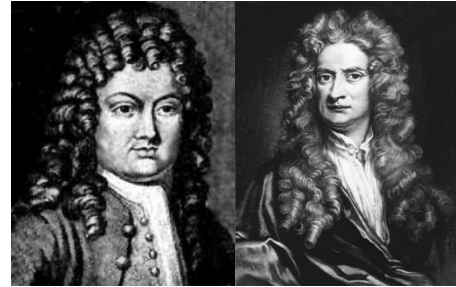
Problem 1: Schon für kleine k wird n sehr groß

Problem 2: Der Algorithmus benötigt exponentiell viele Schritte

Mehr zu Logarithmen

Logarithmen lassen sich deutlich schneller und präziser berechnen.

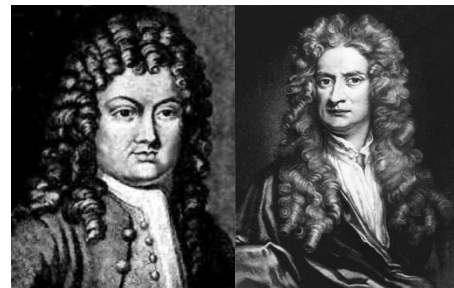
- Taylor-Entwicklung (aus der Analysis)
- Newton-Verfahren (aus der Numerik)



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. 🎉

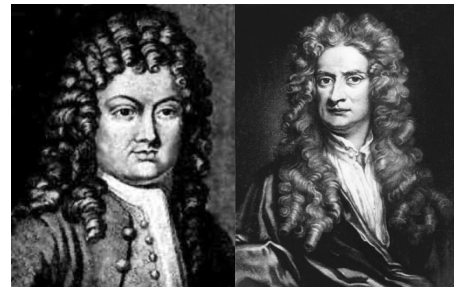
Logarithmen werden aber dennoch benötigt, beispielsweise

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

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?

... nächstes Mal

Beweistechniken

Logische Verknüpfungen

Negation („Nicht“, $\neg$)	Konjunktion („Und“, $\wedge$)	Disjunktion („Oder“, $\vee$)	Implikation („wenn... dann“, $\Rightarrow$)	Äquivalenz („genau dann wenn“, $\Leftrightarrow$)
---------------------------------------	--	---	--	--

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

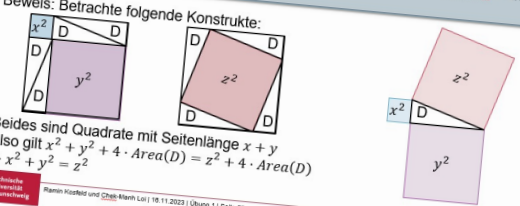
1: Aussage ist wahr
0: Aussage ist falsch

Bew

Beweise – Direkter Beweis

Satz des Pythagoras: Für ein rechtwinkliges Dreieck D mit Kathetenlängen x und y und Hypotenusenlänge z gilt $x^2 + y^2 = z^2$.

Beweis: Betrachte folgende Konstrukte:



Beides sind Quadrate mit Seitenlänge $x + y$
Also gilt $x^2 + y^2 + 4 \cdot \text{Area}(D) = z^2 + 4 \cdot \text{Area}(D)$
 $\Rightarrow x^2 + y^2 = z^2$

Beweise – Direkter Beweis

Aussagen oft in der Form $A \Rightarrow B$. Unterscheide zwischen **Voraussetzung** (A) und **Schlussfolgerung** (B)

Wird die Schlussfolgerung durch eine logische Folgerungskette aus den Voraussetzungen hergeleitet, so spricht man von einem **direkten Beweis**.

Beispiel:
Wenn $x, y \in \mathbb{R}$ und $x, y \geq 0$, dann gilt $\sqrt{xy} \leq \frac{x+y}{2}$



... nächste Woche , am 06.11.!