

# Algorithmen und Datenstrukturen

## Sommersemester 2021

Stefan Näher  
Universität Trier  
naeher@uni-trier.de

## Gesamtübersicht

- Organisatorisches / Einführung
- Grundlagen: RAM, Pseudo-Code,  $\mathcal{O}$ -Notation, Rekursion, Datenstrukturen
- Sortieren
- Wörterbücher und Mengen
- Graphen und Graphalgorithmen

## Organisatorisches

**Vorlesungen** MI 8:30-10:00 DO 8:30-10:00

Die Vorlesungen werden als MP4-Video und PDF-Skript jeweils zu den offiziellen Terminen

<http://chomsky.uni-trier.de/Lehre/Algorithmen-Sommer2021/>  
zum Download zur Verfügung stehen.

**Übungsbetrieb** Die Übungen werden wöchentlich besprochen.  
Beginn: in der zweiten Semesterwoche.

**Sprechstunde**

Moritz Gobbert [gobbert@uni-trier.de](mailto:gobbert@uni-trier.de)

## **Abschlussklausur**

Termin Abschlussklausur: Letzte Semesterwoche

Nachklausur zu Beginn des nächsten Semesters.

Klausurzulassung bei 50% der Punkte aus den Übungen.

Genaueres wird noch bekanntgegeben.

## Übungsaufgaben

Werden jede Woche auf StudIP bereitgestellt sollen (falls möglich) in Gruppen zu 2-3 Personen bearbeitet werden. Eine Woche später (in der Regel) werden die korrigierten und bewerteten Hausaufgaben wieder zurückgegeben (vor den Übungen).

Verantwortlich für den Übungsbetrieb ist **Moritz Gobbert** (Email: gobbert@uni-trier.de)

Er wird in den nächsten Tagen weitere Informationen über den Ablauf liefern insbesondere wie die Übungen per Videokonferenz stattfinden werden.

## **Literatur**

**Ralf H. Güting, Stefan Dieker: Algorithmen und Datenstrukturen**

**Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms**

**Sedgewick: Algorithmen**

**Mehlhorn: Datenstrukturen und Algorithmen**

## Algorithmen und Programmierung

Das *Programmieren* (das üblicherweise mit der Informatik als Erstes in Zusammenhang gebracht wird) kann man als konkretes Ausprägen / Implementieren eines vorher entwickelten Algorithmus ansehen.

~> Algorithmen sind möglichst programmiersprachen- und natürlich auch maschinenunabhängig zu beschreiben.

~> Verwendung von “Pseudocode” zur Notierung von Algorithmen sowie eines allgemeinen abstrakten Maschinenmodells (bei uns einer “RAM”) zur besseren Analyse (Vergleich) von Algorithmen

## **Datenstrukturen**

Die Definition von Datenstrukturen erfolgt im Allgemeinen durch die Angabe einer konkreten Spezifikation zur Datenhaltung und der dazu nötigen Operationen (für Zugriff und Manipulation der Daten).

Diese konkrete Spezifikation legt das allgemeine Verhalten der Operationen fest und abstrahiert damit von der konkreten Implementierung der Datenstruktur.

Beispiele für Datenstrukturen: Felder (Arrays), Listen, Bäume, ...



## Grundverständnis der Vorlesung

Die Vorlesung beschäftigt sich mit **Grundbegriffen der Informatik**

- Gutes Programmieren und ebenfalls gutes Software-Entwickeln ist ohne gute Kenntnisse über Algorithmen unvorstellbar.
- “Algorithmik” zeigt Ihnen auch die Nützlichkeit und Anwendbarkeit (vorher) gelernter mathematischer Kenntnisse.
- Viele Professuren beschäftigen sich gerade in Trier mit der Entwicklung von Algorithmen.

## Lernziele

- Gute Algorithmen und Datenstrukturen für wichtige Probleme
- Paradigmen für den Entwurf von Algorithmen
- Korrektheitsbeweise
- Laufzeit- und Speicherplatzabschätzungen

## Kapitel I: Grundlagen

- Maschinenmodell
- PseudoCode
- Laufzeit/Kosten
- Einfache Algorithmen
- Asymptotische Laufzeitanalyse (O-Notation)
- Rekursive Algorithmen
- Einfache Datenstrukturen

## Das Maschinenmodell: RAM

RAM: Random Access Machine

Eine RAM hat einen *Prozessor* d.h. sie arbeitet sequentiell (keine Parallelverarbeitung).

Der *Speicher* der Maschine kann als unbeschränkt langes Feld (Array) aufgefasst werden. Jede Speicherzelle kann Werte beliebiger Typen speichern. In dieser Vorlesung wird meistens mit (ganzen) Zahlen gearbeitet.

Jeder *Schritt* der Maschine verursacht *Kosten 1*, unabhängig von der Größe der Operanden (*Einheitskostenmaß*). Zu den Schritten der Maschine zählen neben direkten ebenfalls *indirekte Speicherzugriffe*.

## **Pseudo-Code** Eine abstrakte Programmiersprache

**Variablen:** Namen für Speicherzellen

$x \leftarrow 17;$

$y \leftarrow x;$

**Felder:** Abschnitt aufeinanderfolgender Speicherzellen

$A[\ell..r]$   $\ell$ : Anfangsindex (links)  $r$ : End-Index (rechts)

$A[0..99]$  Feld der Länge 100 (Indexbereich beginnt bei 0)

$A[1..100]$  Indexbereich beginnt bei 1

**Feldzugriff**

$x \leftarrow A[17];$

$y \leftarrow A[x];$

## Kontrollstrukturen

**if** <Bedingung> **then** ... **fi**

**if** <Bedingung>  
**then** ...  
**else** ...  
**fi**

**while** <Bedingung> **do... od**

**repeat** ... **until** <Bedingung>

**for**  $i = 1$  **to**  $n$  **do... od**

## Laufzeit (Kosten)

Jede Operation (Rechenschritt) im Pseudo Code benötigt 1 Zeiteinheit  
alternativ: hat Kosten 1

### Analyse:

Schätze die Zahl der Schritte im **schlechtesten Fall** *worst case* ab  
d.h. worst case Kosten

Im Gegensatz zur **mittleren** Analyse  
d.h. mittlere oder erwartete Kosten

## Erstes Beispiel: Lineare Suche

Feld  $A[1..n]$  von  $n$  (ganzen) Zahlen

`LINEAR_SEARCH(x)` liefert die erste Position  $i$  von  $x$  in  $A$   
d.h. kleinstes  $i$  mit  $A[i] = x$  bzw. 0 falls  $i \notin A$

1.  $i \leftarrow 1$ ;
2. **while**  $i \leq n \wedge A[i] \neq x$  **do**
3.    $i \leftarrow i + 1$ ;
4. **od**
5. **if**  $i > n$  **then**  $i \leftarrow 0$ ; **fi**



## Laufzeitanalyse

Wieviele Schritte führt LINEAR\_SEARCH im schlimmsten Fall aus ?  
d.h. wenn  $x \notin A$

**Zeile 1:** 1 mal Wertzuweisung  $i \leftarrow 1$

**Zeile 2:**  $n + 1$  mal  $i \leq n$  Vergleich und  
     $n$  mal Feldzugriff  $A[i]$  und  
     $n$  mal Vergleich  $\neq x$   
     $n$  mal logische  $\wedge$  Operation

**Zeile 3:**  $n$  mal Addition  $i + 1$  und  
     $n$  mal Wertzuweisung  $i \leftarrow \dots$

**Zeile 5:** 1 Vergleich + 1 Wertzuweisung

Worst-Case Laufzeit:  **$6n + 4$**

## Lineare Laufzeit

Die Laufzeit von LINEAR\_SEARCH ist  $\leq 6n + 4$

$6n + 4 \leq 7n$  für Felder der Länge  $n \geq 4$

d.h. linear oder proportional zur Länge des Feldes (mit Faktor 7).

**Eine Verdoppelung der Länge von A führt zu einer Verdoppelung der Laufzeit.**

## 2. Binäre Suche

Vorraussetzung:  $A[1..n]$  ist **aufsteigend sortiert**

d.h.  $A[i] \leq A[i + 1]$  für  $i = 1, \dots, n - 1$

Zunächst nehmen wir an, dass sogar  $A[i] < A[i + 1]$

d.h. es gibt keine Wiederholungen (allgemeiner Fall: Übung)

### Idee des Algorithmus:

Betrachte immer 2 Positionen  $\ell$  und  $r$  mit:

Falls  $x = A[i]$  für eine Position  $i$  dann gilt  $\ell \leq i \leq r$

$\leadsto$  **Invariante** des Algorithmus

## Initialisierung

$\ell \leftarrow 1$  und  $r \leftarrow n$ .

## Schritt des Algorithmus

sei  $m \leftarrow \lfloor (\ell + r)/2 \rfloor$  d.h.  $m$  ist mittlere Position in  $[\ell..r]$

Dann gibt es 3 mögliche Fälle:

- a)  $x = A[m]$  **fertig** ( $x$  gefunden)
- b)  $x < A[m]$  suche weiter im Teilfeld  $A[\ell..m-1]$  ( $r \leftarrow m-1$ )
- c)  $x > A[m]$  suche weiter im Teilfeld  $A[m+1..r]$  ( $\ell \leftarrow m+1$ );

BINARY\_SEARCH ( $x$ )

1.  $\ell \leftarrow 1$ ;
2.  $r \leftarrow n$ ;
3. **while**  $\ell \leq r$  **do**
4.    $m \leftarrow \lfloor (\ell + r)/2 \rfloor$ ;
5.   **if**  $A[m] = x$  **then**
6.     **return**  $m$ ;
7.   **fi**
8.   **if**  $x < A[m]$
9.     **then**  $r \leftarrow m - 1$ ;
10.    **else**  $\ell \leftarrow m + 1$ ;
11.   **fi**
12. **od**
13. **return** 0;

## Korrektheit

1. Die Invariante ist stets erfüllt.

2. Termination

Die Länge des Teilfeldes  $A[\ell..r]$  wird in jedem Durchlauf der while-Schleife vermindert (sogar halbiert !)

## **Laufzeitanalyse 1** (Anzahl der Iterationen)

Wieviele Schritte führt BINARY\_SEARCH im schlimmsten Fall aus ?  
d.h. wenn  $x \notin A$

1. Die Länge des Intervalls  $[\ell..r]$  ist am Anfang  $n$ .
2. Bei jeder Ausführung der while-Schleife wird das Intervall mindestens halbiert.
3. Der Algorithmus terminiert, wenn dieses Intervall leer ist (Zeile 3).

### **Frage:**

Wie oft muss man eine Zahl  $n$  wiederholt durch 2 teilen bis das Resultat  $< 1$  ist.

### **Antwort:**

Dies ist eine mögliche Definition des Logarithmus zur Basis 2.

Also wird die while-Schleife maximal  $\log_2(n)$  mal ausgeführt.

## **Laufzeitanalyse 2** (Gesamtkosten)

Wir schätzen die worst-case Kosten einer Ausführung der Schleife ab:

Maximal 11 Operationen pro Schleife  
(bitte Nachzählen und evtl. korrigieren)

Die worst-case Laufzeit von BINARY\_SEARCH ist somit  $11 \cdot \log_2(n) + 3$

Falls  $\log_2(n) \geq 3$  d.h.  $n \geq 8$  ist die Laufzeit also  $12 \cdot \log_2(n)$   
d.h. logarithmisch .

**Eine Verdoppelung der Länge von A führt nur zu einer Erhöhung der Laufzeit um eine Konstante (nämlich 11).**