Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Tanya Braun und Felix Kuhr (Übungen)

sowie viele Tutoren

Teilnehmerkreis und Voraussetzungen

Studiengänge

- Bachelor **Informatik**
- Bachelor/Master **Mathematik in Medizin und Lebenswissenschaften**
- Bachelor **Medieninformatik**
- Bachelor **Medizinische Ingenieurwissenschaft**
- Bachelor **Medizinische Informatik**
- Bachelor **IT-Sicherheit**
- Bachelor **Robotik und Autonome Systeme**

Voraussetzungen

- Einführung in die Programmierung
- Lineare Algebra und Diskrete Strukturen 1

Organisatorisches: Übungen

- **Start**: Montag, 24. April 2017
- **Übungen**: Montags
  Anmeldung über Moodle nach dieser Veranstaltung
- **Übungsaufgaben** stehen jeweils kurz nach der Vorlesung am Freitag über Moodle bereit
- Aufgaben sollen in einer **2-er Gruppe** bearbeitet werden
- **Abgabe der Lösungen** erfolgt bis Donnerstag in der jeweils folgenden Woche nach Ausgabe bis 12 Uhr in der IFIS-Teeküche (1 Kasten pro Gruppe)
- Bitte unbedingt Namen, Matrikelnummern und Übungsgruppennummern auf Abgaben vermerken

Organisatorisches: Prüfung

- Die **Eintragung in den Kurs** und in eine Übungsgruppe ist **Voraussetzung**, um an dem Modul Algorithmen und Datenstrukturen teilnehmen zu können und Zugriff auf die Unterlagen zu erhalten
- Am Ende des Semesters findet eine **Klausur** statt
- **Voraussetzung** zur Teilnahme an der Klausur sind mindestens **50% der gesamtmöglichen Punkte aller Übungszettel**

Literatur

Th. Cormen, C.E. Leiserson, R. Rivest, C. Stein,
Algorithmen: Eine Einführung,
4. Auflage, Oldenbourg, 2013

M. Dietzfelbinger, K. Mehlhorn, P. Sanders
Algorithmen und Datenstrukturen - Die Grundwerkzeuge,
Springer, 2014

R. Sedgewick, K. Wayne,
Algorithmen und Datenstrukturen,
4. Auflage, Pearson, 2014

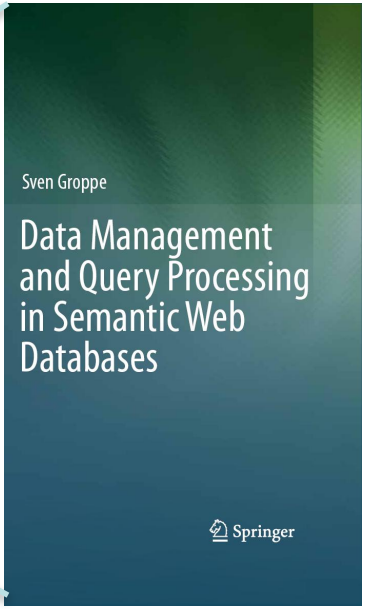
Literatur

T. Ottmann, P. Widmayer,
Algorithmen und Datenstrukturen,
Spektrum 1997

U. Schöning,
Algorithmik,
Spektrum, 2011

Ausblick über IFIS Module

- **Bachelor-Programm**
  - Algorithmen und Datenstrukturen
  - Datenbanken
  - Non-Standard-Datenbanken
- **Master-Programm**
  - Webbasierte Informationssysteme
  - Datenmanagement
    - Mobile und verteilte Datenbanken
    - Semantic Web
  - Web and Data Science
    - Foundations of Ontologies and Databases for Information Systems
    - Web Mining Agents



Allgemeine Lernziele in diesem Kurs

- Weg **vom Problem zum Algorithmus** gehen können
  - **Auswahl** eines Algorithmus aus Alternativen unter Bezugnahme auf vorliegende Daten und deren Struktur
  - **Entwicklung** eines Algorithmus mitsamt geeigneter Datenstrukturen (Terminierung, Korrektheit, ...)
- **Analyse von Algorithmen** durchführen
  - Anwachsen der Laufzeit bei Vergrößerung der Eingabe
- Erste Schritte in Bezug auf die **Analyse von Problemen** gehen können
  - Ja, Probleme sind etwas anderes als Algorithmen!
  - Probleme können in gewisser Weise „schwer“ sein
  - Prüfung, ob Algorithmus optimal

Beispielproblem: Summe der Elemente eines Feldes A[1..n] bestimmen

- Algorithmus?

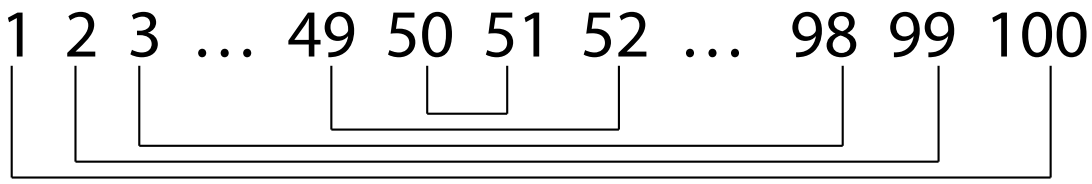
- summe(A) = $\sum_{i=1}^{n} A[i]$
- Aufwand?
- Wenn A n Elemente hat, n Schritte!
- Der Aufwand wird *linear* genannt

Spezialisierung des Problems

- Vorwissen: A[i] = i

- Das Problem wird sehr viel einfacher!

Ausnutzen der Einschränkung



Summe jedes Paares: 101

50 Paare: $101 * 50 = 5050$

- Lösungsverfahren:

```
function summe-2(A)
    n ← length(A)
    return (n+1)*(n/2)
```

- Nach Carl Friedrich Gauß (ca. 1786)
- Aufwand?
- Konstant, d.h. hängt (idealisiert!) nicht von n ab
- Entwurfsmuster: Ein-Schritt-Berechnung

Algorithmen: Notation durch Programme

- Annahme: Serielle Ausführung
- Vgl. **Vorlesung „Einführung in die Programmierung“**
  - Variablen, Felder A[...]
  - Zuweisungen ← (oder auch **:=**)
  - Fallunterscheidungen **if** ... **then** ... **else** ...
    - Vergleich und Berechnungen für Bedingungstest
  - Schleifen **while** ... **do**, **for** ... **do**
    - Vergleich und Berechnungen für Bedingungstest
  - **procedure**, **function**
  - Auf Folien wird der jeweilige Skopus durch Einrückung ausgedrückt

Ein erstes Problem: Summe der Elemente

- **Gegeben: A[1..n] : N**
  - Feld (Array) A von n Zahlen aus N (natürliche Zahlen)
- **Gesucht:**
  - Transformation **S** von A, so dass gilt:
    - summe = $\Sigma_{i \in \{1,...,n\}}$ A[i]
  - Also: Gesucht ist ein Verfahren **S**, so dass **{ P } S { Q }** gilt (Notation nach Hoare)
    - Vorbedingung: **P**: true (keine Einschränkung)
    - Nachbedingung: **Q**: summe = $\Sigma_{i \in \{1,...,n\}}$ A[i]

C.A.R. Hoare, An axiomatic basis for computer programming.
Communications of the ACM, Vol. 12, S. 567 - 583, **1969**

Ein zweites Problem: Summe der Elemente

- **Gegeben: A[1..n] : N**
  - Feld (Array) A von n Zahlen aus N (natürliche Zahlen)
- **Gesucht:**
  - Transformation **S** von A, so dass gilt:
    - summe-2 = $\Sigma_{i \in \{1,...,n\}}$ A[i]
  - Also: Gesucht ist ein Verfahren **S**, so dass **{ P } S { Q }** gilt (Notation nach Hoare)
    - Vorbedingung: **P**: **A[i] = i**
    - Nachbedingung: **Q**: summe-2 = $\Sigma_{i \in \{1,...,n\}}$ A[i]

C.A.R. Hoare, An axiomatic basis for computer programming.
Communications of the ACM, Vol. 12, S. 567 - 583, **1969**

Ein erstes Problem: In-situ-Sortierproblem

- **Gegeben: A[1..n] : N**
  - Feld (Array) A von n Zahlen aus N (natürliche Zahlen)
- **Gesucht:**
  - Transformation S von A, so dass gilt: $\forall\, 1 \le i < j \le n$: $A[i] \le A[j]$
  - Nebenbedingung: Es wird intern kein weiteres Feld gleicher (oder auch nur fast gleicher Größe) verwendet
  - Also: Gesucht ist ein Verfahren **S**, so dass **{ P } S { Q }** gilt (Notation nach Hoare)
    - Vorbedingung: **P = true** (keine Einschränkung)
    - Nachbedingung: **Q =** $\forall$ **1≤i<j≤n: A[i] ≤ A[j]**
    - Nebenbedingung: nur „konstant“ viel zusätzlicher Speicher (feste Anzahl von Hilfsvariablen)

C.A.R. Hoare, An axiomatic basis for computer programming. Communications of the ACM, Vol. 12, S. 567 - 583, **1969**

In-situ-Sortieren: Problemanalyse

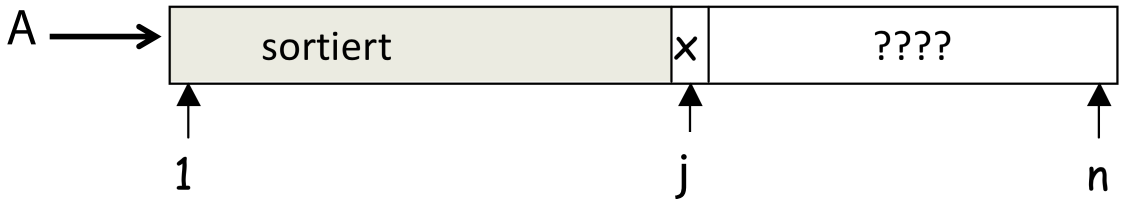
- Felder erlauben **wahlfreien** Zugriff auf Elemente
  - **Zugriffszeit** für ein Feld **konstant**
    (d.h. sie hängt nicht vom Indexwert ab)
  - Idealisierende Annahme (gilt nicht für moderne Computer)
- Es gibt keine Aussage darüber, ob die Feldinhalte schon sortiert sind, eine willkürliche Reihenfolge haben, oder umgekehrt sortiert sind
  - Vielleicht lassen sich solche „**erwarteten Eingaben**“ aber in der Praxis feststellen
- **Aufwand das Problem zu lösen**: Man kann leicht sehen, dass jedes Element „falsch positioniert“ sein kann
  - **Mindestaufwand** im allgemeinen Fall: n Bewegungen
  - **Maximalaufwand** in Abhängigkeit von n?

Aufwand zur Lösung eines Problems

- Gegeben ein **Problem** (hier: In-situ-Sortierproblem)
  - Damit verbundene Fragen:
    - Wie „langsam“ muss ein Algorithmus sein, damit alle möglichen Probleminstanzen korrekt gelöst werden?
    - Oder: Wenn wir schon einen Algorithmus haben, können wir noch einen „substantiell besseren“ finden?
- Jedes Eingabefeld A stellt eine **Probleminstanz** dar
- Notwendiger Aufwand in Abhängigkeit von der Eingabegröße heißt **Komplexität eines Problems**
  - Anzahl der notwendigen Verarbeitungsschritte in Abhängigkeit der Größe der Eingabe (hier: Anzahl der Elemente des Feldes A)
  - Komplexität durch jeweils „schlimmste“ Probleminstanz bestimmt
  - Einzelne Probleminstanzen können evtl. weniger Schritte benötigen

Beispiel 2: Sortierung

- Gegeben: A = [4, 7, 3, 5, 9, 1]
- Gesucht: In-situ-Sortierverfahren (aufsteigend)
- Aufgabe: Entwickle "Idee"

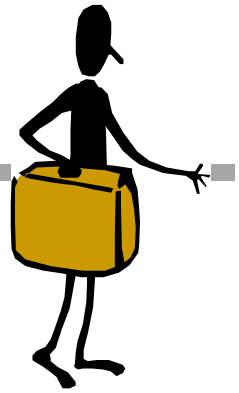



```
1: procedure INSERTION-SORT(A)
2:     for j ← 2 to length(A) do
3:         key ← A[j]
4:         ▷ Insert A[j] into the sorted sequence A[1..j − 1]
5:         i ← j − 1
6:         while i > 0 and A[i] > key do
7:             A[i + 1] ← A[i]
8:             i ← i − 1
9:         A[i + 1] ← key
```

Entwurfsmuster / Entwurfsverfahren

- Schrittweise Berechnung
  - Beispiel: Bestimmung der Summe eines Feldes durch Aufsummierung von Feldelementen
- Ein-Schritt-Berechnng
  - Beispiel: Bestimmung der Summe eines Feldes ohne die Feldelemente selbst zu betrachten (geht nur unter Annahmen)
- Verkleinerungsprinzip
  - Beispiel: Sortierung eines Feldes
    - Unsortierter Teil wird immer kleiner, letztlich leer
    - Umgekehrt: Sortierter Teil wird immer größer, umfasst am Ende alles → Sortierung erreicht

Zusammenfassung: Entwurfsmuster



- In dieser Vorlesungseinheit:
  - Schrittweise Berechnung
  - Ein-Schritt-Berechnung
  - Verkleinerungsprinzip
- Nächste Vorlesung
  - Teile und Herrsche
- „Später“:
  - Vollständige Suchverfahren (z.B. Rücksetzen, Verzweigen und Begrenzen)
  - Approximative Such- und Berechnungsverfahren (z.B. gierige Suche)
  - Schrittweise Annäherung
  - Dynamisches Programmieren (Berechnung von Teilen und deren Kombination, Wiederverwendung von Zwischenergebnissen)