

# Informatik II - SS 2016

## (Algorithmen & Datenstrukturen)

Vorlesung 1 (20.4.2016)

Organisation, Übungen, Sortieren I



**UNI  
FREIBURG**

Fabian Kuhn

Algorithmen und Komplexität

- Letztes Semester haben Sie (die Informatiker/innen) die Grundzüge des Programmierens gelernt
- Fragen der Effizienz spielten eine untergeordnete Rolle
- Dies ist das generelle Thema dieser Vorlesung
  - **Wie schnell ist mein Programm? Wie misst man das?**
  - **Wie kann ich es schneller machen?**
  - **Wie kann ich beweisen, dass es immer so schnell läuft?**
- Manchmal geht es auch um Sparsamkeit im Platzverbrauch oder andere Ressourcen, aber hier meistens um **Zeit**

- Sie können schon **kleinere Programme schreiben**
  - z. B.: die kleinste Zahl in einem Array finden, einen gegebenen einfachen Algorithmus implementieren, ...
- Sie haben **Erfahrung in einer Programmiersprache** gesammelt
  - Offiziell werden wir in dieser Vorlesung **Python** unterstützen
  - Falls Sie möchten, können Sie aber auch Java oder C++ verwenden
- Verständnis von **wichtigen Grundkonzepten der Programmierung**
  - Variablen, Objekte, Schleifen, Ein- und Ausgabe, Funktionen, Rekursion
- **Falls Sie da Lücken haben**, müssen Sie das parallel zur Vorlesung aufarbeiten und haben dann **deutlich mehr Aufwand!**

## Algorithmen:

- Algorithmenentwurf: Wie löst man ein geg. Problem effizient?
- Wie analysiert man einen Algorithmus?
  - O-Notation, Laufzeitanalyse, Korrektheit (auch math. Beweise)
  - Sie sollten z. B. nach der Vorlesung verstehen, ob Ihr Programm
    - in Linearzeit oder “fast-Linearzeit” läuft (gut!)
    - quadratische oder höhere polynomielle Laufzeit hat (oft schlecht!)
    - exponentielle Laufzeit hat (immer schlecht!)
- Beispiele: Sortieren, Suchen, Graphenalgorithmen (kürzeste Wege, Spannbäume, Breiten-/Tiefen-Suche), Editierdistanz

## Datenstrukturen:

- Wie legt man Daten ab, so dass man schnell darauf zugreifen kann?
- Gute/geeignete Datenstrukturen  $\Leftrightarrow$  effiziente Algorithmen
  - Wir müssen gute Algorithmen haben, um effizient auf Daten zuzugreifen
  - Daten müssen geschickt abgelegt werden, damit effiziente Alg. Existieren
  - Die Laufzeit von Algorithmen für grössere Probleme hängt oft eng damit zusammen, welche Datenstrukturen man verwendet, um die anfallenden Daten zu verwalten
- Beispiele: Hashtabellen, Suchbäume, (Prioritäts-)Warteschlangen, dynamische Arrays, Repräsentation von Graphen

## Vorlesung (101-00-036)

- Mittwoch: 14:15-15:45
- Freitag: 14:15-15:45

**Webpage:** [http://ac.informatik.uni-freiburg.de/teaching/ss\\_16/info2.php](http://ac.informatik.uni-freiburg.de/teaching/ss_16/info2.php)

## Übungen

- Ein Übungsblatt pro Woche (12 Übungsblätter)
- Übungen ca.  $\frac{1}{2}$  theoretisch und  $\frac{1}{2}$  praktisch
- Ablauf der Übungen alles online → gleich mehr...

## Aufwand

- 8 ECTS (für die Informatiker) = ca. 240 Arbeitsstunden
- Bei 160h Vorlesungen/Übungen und 80h Klausurvorbereitung
  - ca. **10 Stunden pro Übung...**

## Übungsblätter

- 12 Übungsblätter, pro Übungsblatt max. 20 Punkte
- Um an der Klausur teilzunehmen, müssen Sie **50% der Punkte** aus den Übungen machen.
- Übungen sind grundsätzlich die beste Klausurvorbereitung!!

## Übungsgruppen

- Keine regelmässigen Übungsgruppentermine (alles online)
- Assistenten:  
[Yannic Maus](#) (theor. Übungen), [Oleksii \(Alex\) Saukh](#) (prakt. Ü.)
- Tutoren:  
Julian Bürklin, Sebastian Holler, Daniel Kemen,  
Betim Musa, Rezart Qelibari, Frank Schüssele

## Die Übungen sind der wichtigste Teil der Vorlesung!

- Wenn Sie die Übungen alle sorgfältig gemacht und verstanden haben, sollte die Klausur gut machbar sein.



## benutzen **Daphne** als **Kursverwaltungssystem**

- Link auf der Vorlesungswebpage: **bitte anmelden!**
- In Daphne haben Sie eine Übersicht über folgende Infos
  - Wer ihr/e Tutor/in ist
  - Ihre Punkte in den Übungsblättern
  - Infos zum aktuellen Übungsblatt
  - Link zum **Forum** ... gleich mehr dazu
  - Link zum **SVN** ... gleich mehr dazu
  - Link zu unseren **Coding Standards** ... gleich mehr dazu
  - Link zu unserem **Build System** ... gleich mehr dazu
- Daphne kennen Sie von Informatik I ?
- Das gleiche System wird auch in “Programmieren in C++” verwendet...

## Forum zu Vorlesung / Übungen

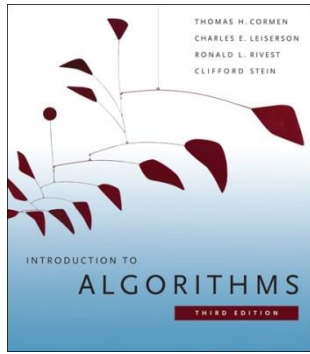
- Link dazu auf der Vorlesungswebseite / Daphne-Seite
- (Alle) Fragen zu Übungen / Vorlesung bitte übers Forum fragen!
  - Wir können so am schnellsten antworten...
  - Antwort hilft oft vielen anderen auch...
- Nutzen Sie die Gelegenheit, über's Forum zu fragen!
- Stellen Sie konkrete Fragen!
  - Auf Fragen wie “Programm läuft nicht”, “Ich habe keine Ahnung, wie ich das beweisen soll” können Sie keine Antwort erwarten
  - Bei Fragen zum Programmieren: Bitte Code-Segmente angeben, welche nicht funktionieren, jedoch nicht ganze Programme ins Forum posten!
  - Angabe der Fehlermeldung
- Falls nötig, können Sie sich natürlich mit Ihrem Tutor auch mal persönlich treffen...

SVN : <http://subversion.apache.org>

- Kennen Sie bereits aus Informatik I?
- Dateien liegen auf einem zentralen Server, in einem sogenannten **repository**, die typische Operationen sind
  - **Update**: neuste Version vom Server ziehen
  - **Commit**: letzte Änderungen auf den Server hochladen
- Vollständige Historie von allen Änderungen an den Dateien
- Insbesondere nützlich für das Schreiben von Code
- Wir werden dies benutzen für
  - die **Abgaben Ihrer Übungsblätter** (Code + alles andere)
  - das **Feedback von Ihrem Tutor**
  - Vorlesungsdateien / Musterlösungen

## Richtlinien für die Abgabe der Programmierübungen:

- Sie können die Übungen in Python, Java oder C++ Machen
  - Offiziell unterstützen wir allerdings nur Python (z.B. Musterlösungen)
- **Unit tests** für alle nichttrivialen Methoden
  - Ohne Test ist die Chance gross, dass die Methode falsch ist...
  - Immer **mind. eine typische Eingabe** und **einen kritischen Grenzfall** testen!
  - Vereinfacht das Debuggen von grösseren Projekten
  - Wir/Sie können so testen, ob das Programm das Richtige tut
- Befolgen eines **einheitlichen Programmierstils** (style checker)
  - Macht den Code leserlich (auch für andere!)
- Standardisiertes **Build-Framework** (jenkins)
  - So können wir effizient Ihre Programme ausprobieren und testen
- Ich werde dies heute und am Freitag alles demonstrieren...



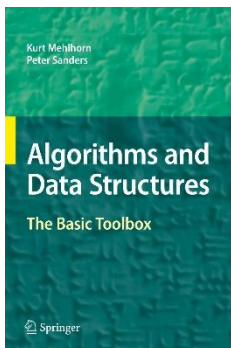
T. Cormen, C. Leiserson, R. Rivest, C. Stein  
Introduction to Algorithms, Third Edition,  
MIT Press, 2009

## **Klassisches Lehrbuch zum Thema (auf Englisch)**



T. Ottmann, P. Widmayer  
Algorithmen und Datenstrukturen, 5. Auflage,  
Spektrum Akademischer Verlag, Heidelberg, 2012

## **Deutsche Alternative (aus Freiburg / Zürich)**



K. Mehlhorn, P. Sanders  
Algorithms and Data Structures,  
Springer, 2008, online verfügbar unter

<http://www.mpi-inf.mpg.de/~mehlhorn/Toolbox.html>

## Wikipedia:

- ist bei Standardalgorithmen / -datenstrukturen recht gut...
- z.T. auch für weitergehende Algorithmen

## Alte Vorlesungen, z.B.:

- verfügbares Material von früheren Informatik II Vorlesungen
  - 2015: <https://ad-wiki.informatik.uni-freiburg.de/teaching/AlgoDatSS2015>
  - 2014: [http://ac.informatik.uni-freiburg.de/teaching/ss\\_14/info2.php](http://ac.informatik.uni-freiburg.de/teaching/ss_14/info2.php)
  - ...
- Vorlesungen von anderen Unis, z.B.:  
MIT Courseware: <http://ocw.mit.edu>
  - u.a. mit Aufzeichnungen von 2011

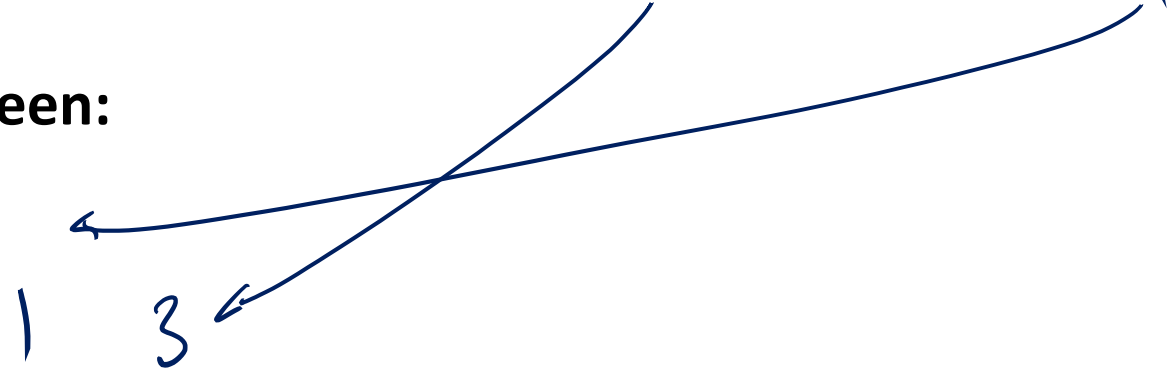
## Problemdefinition

- **Eingabe:** Sequenz von  $n$  Elementen  $x_1, \dots, x_n$
- Ordnungsrelation  $\leq$  auf den Elementen
  - Vergleichsoperation, mit welcher zwei beliebige Elemente miteinander verglichen werden können
  - Bsp. 1: Sequenz von Zahlen mit üblicher  $\leq$ -Relation
  - Bsp. 2: Sequenz von Strings mit lexikographischer (alphabetischer) Ordnung
- **Ausgabe:** Gemäss  $\leq$ -Ordnung sortierte Sequenz der  $n$  Elemente
- **Beispiel:**
  - Eingabe: [15, 3, 27, 49, 23, 18, 6, 1, 31]
  - Ausgabe: [1, 3, 6, 15, 18, 23, 27, 31, 49]
- Sortieren wird in (fast) allen grösseren Programmen gebraucht!

# Algorithmus zum Sortieren?

**Aufgabe:** Sortiere Array  $A$  (z.B.  $A = [15, 3, 27, 49, 23, 18, 6, \cancel{X}, 31]$ )

**Algorithmus, Ideen:**

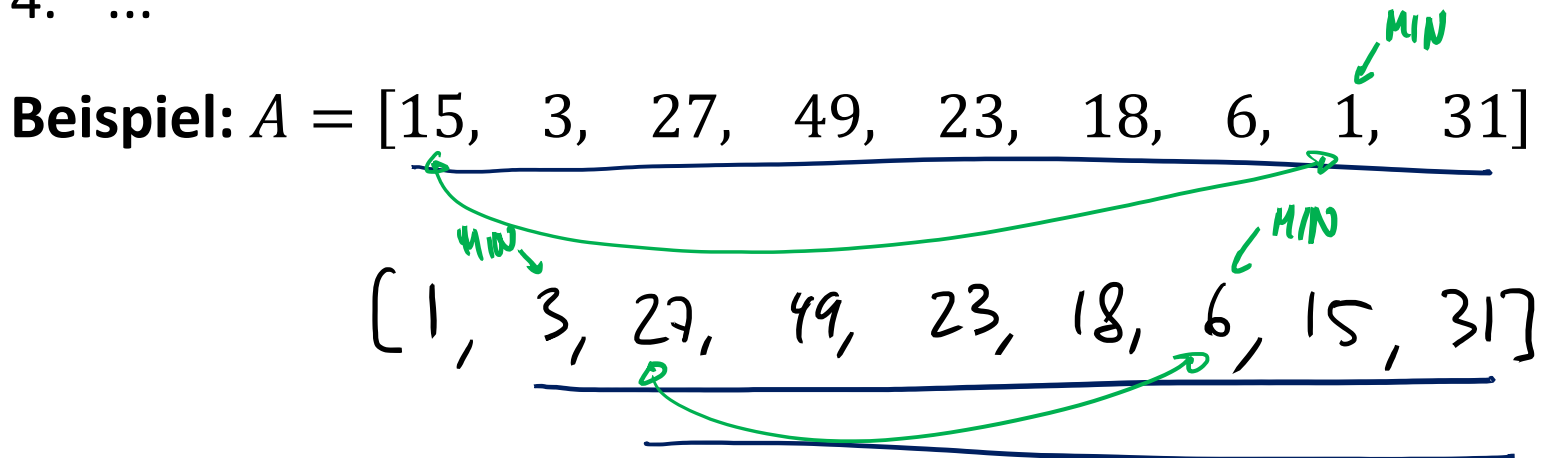




# Selection Sort Algorithmus

## SelectionSort (informell):

1. Finde kleinstes Element im Array, vertausche es an 1. Stelle
2. Finde kleinstes Element im Rest, vertausche es an 2. Stelle
3. Finde kleinstes Element im Rest, vertausche es an 3. Stelle
4. ...



# Selection Sort: Pseudocode

Eingabe: Array  $A$  der Grösse  $n$

SelectionSort( $A$ ):

**for**  $i=0$  **to**  $n-2$  **do**

*// find min in  $A[i..n-1]$*

*// swap  $A[i]$  with min of  $A[i..n-1]$*

# Selection Sort: Pseudocode

Eingabe: Array  $A$  der Grösse  $n$

SelectionSort( $A$ ):

```
1: for  $i=0$  to  $n-2$  do
2:   // find min in  $A[i..n-1]$ 
3:    $minIdx = i$ 
4:   for  $j=i$  to  $n-1$  do
5:     if  $A[j] < A[minIdx]$  then
6:        $minIdx = j$ 
7:   // swap  $A[i]$  with min of  $A[i..n-1]$ 
8:    $tmp = A[i]$ 
9:    $A[i] = A[minIdx]$ 
10:   $A[minIdx] = tmp$ 
```

# SelectionSort: Programm

Schreiben wir doch das gleich mal als Python-Programm um...

Dabei sehen wir gleich die Umgebung für die Übungen:

- **Unit Tests** : `doctest` (Java / C++: `junit` / `gtest`)
- **Coding Standards** : `flake8` (Java / C++: `checkstyle` / `cpplint`)
- **Build-Framework** : `make` (Java / C++: `ant` / `make`)
- **Build-System** : `jenkins`
- **File-Repository** : `SVN`

```
import time
```

```
...
```

```
start_time = time.time();
```

```
// code segment for which you want to measure  
// the running time.
```

```
run_time = (time.time() - start_time) * 1000;
```

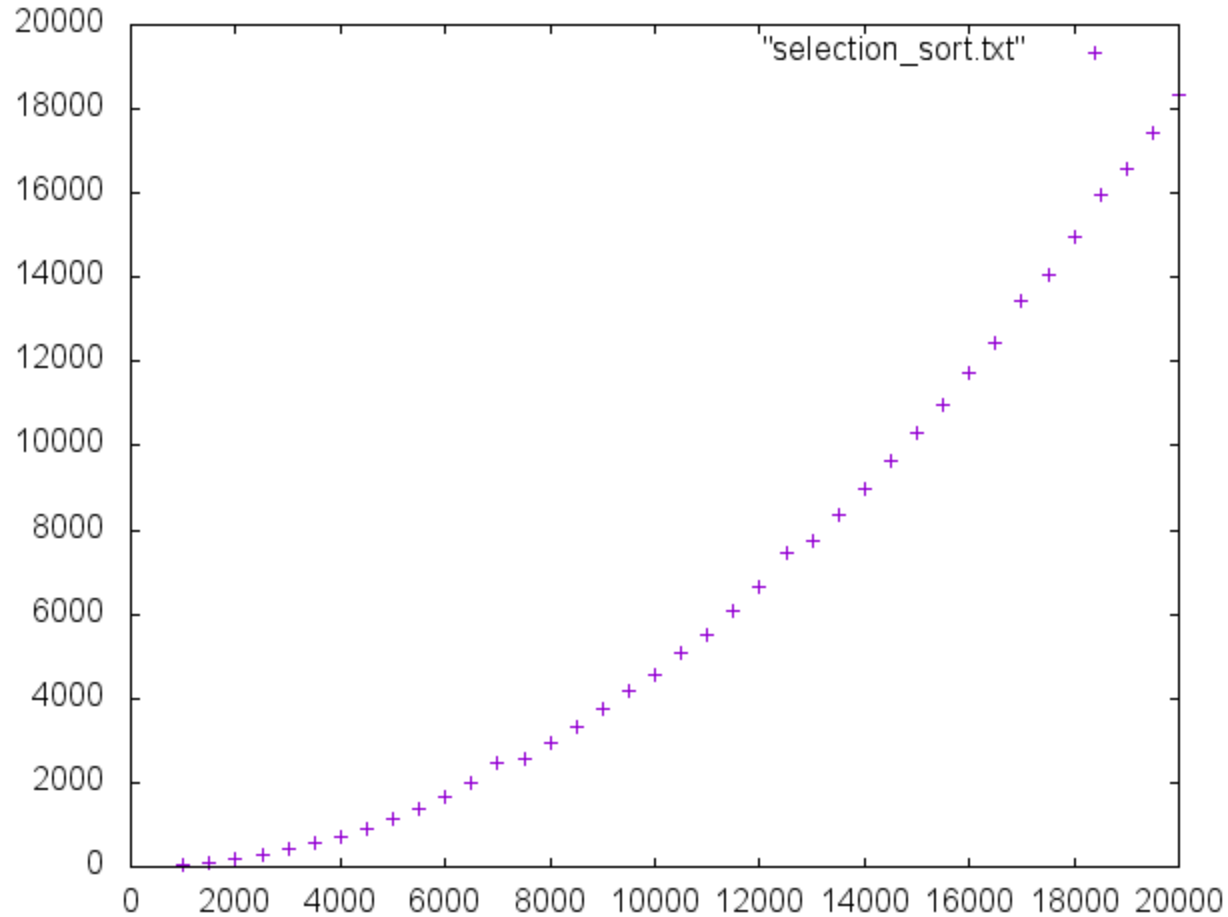
## Java:

```
long startTime = System.currentTimeMillis();  
// code segment for which you want to measure  
// the running time.  
long runTime = System.currentTimeMillis() - startTime;
```

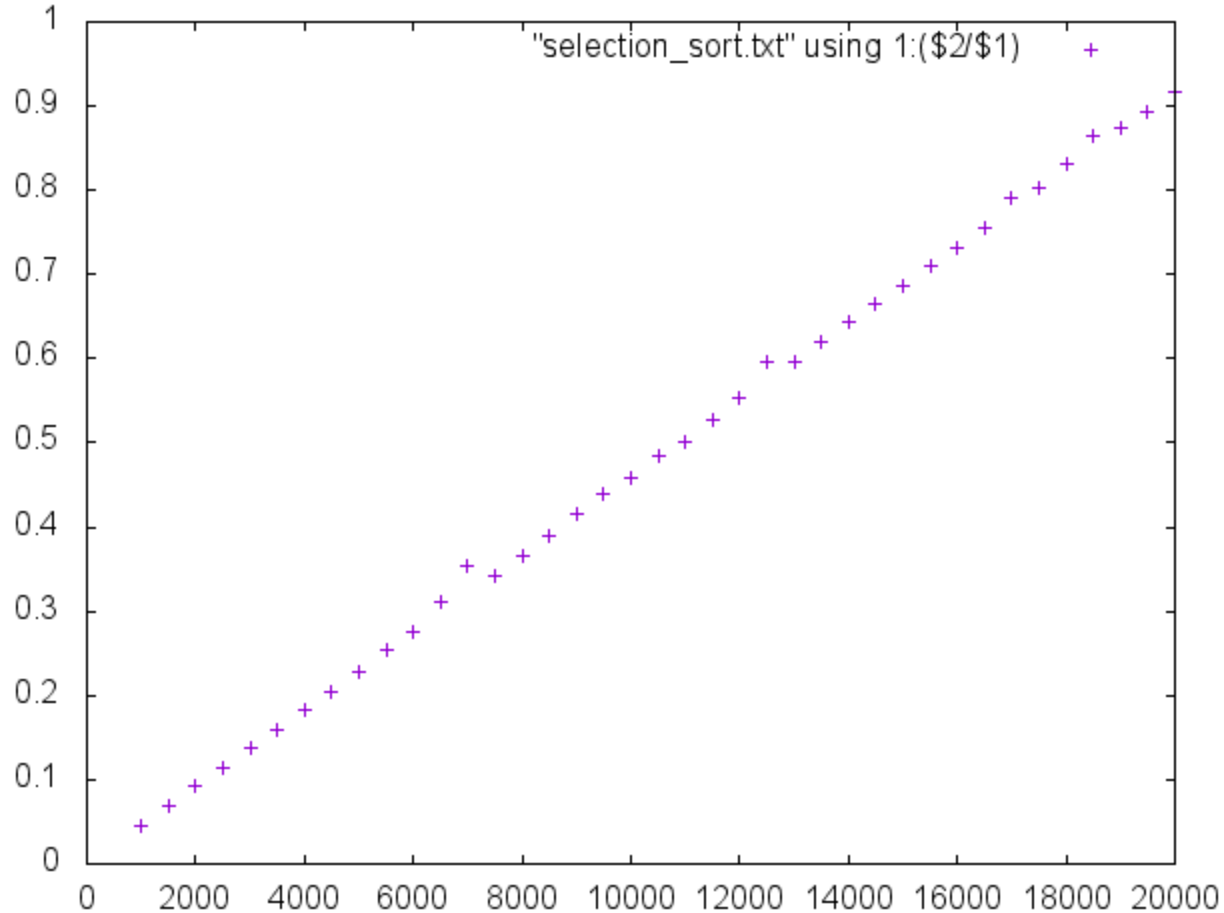
## C++:

```
#include <ctime>  
  
...  
std::clock_t startTime = std::clock();  
// code segment for which you want to measure  
// the running time.  
std::clock_t endTime = std::clock();  
int runTime = static_cast<int>(1000*(endTime - startTime));
```

# Zeitmessung SelectionSort



# Zeitmessung SelectionSort





# Zeitmessung SelectionSort

