

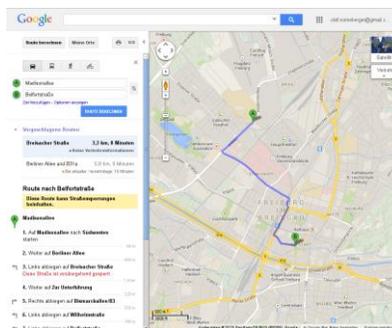
Algorithmen und Datenstrukturen (ESE)  
Entwurf, Analyse und Umsetzung von  
Algorithmen (IEMS)  
WS 2014 / 2015

Vorlesung 1, Donnerstag, 23. Oktober 2014  
(Einführung, Organisatorisches, Sortieren)

Junior-Prof. Dr. Olaf Ronneberger  
Image Analysis Lab  
Institut für Informatik  
Universität Freiburg

[basiert auf der Veranstaltung von Prof. Dr. Hannah Bast, Lehrstuhl für  
Algorithmen und Datenstrukturen]

Beispiel 2: Routenplanung

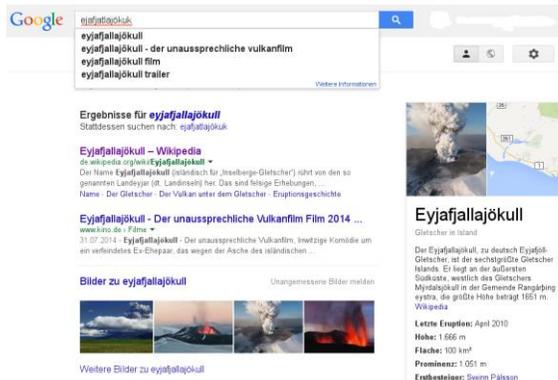


- **Datenstruktur:**
  - Wie repräsentiere ich die Karte im Computer?
- **Algorithmus:**
  - Wie finde ich den kürzesten / schnellsten Weg?

Thema der Vorlesung

- Algorithmen und Datenstrukturen ...
  - ... für Probleme, die in praktisch **jedem** größeren Programm / Projekt auftauchen
  - Algorithmen = Verfahren um Probleme zu lösen
  - Datenstrukturen = Repräsentation der Daten dazu im Rechner

Beispiel 3: Fehlertoleranter Textvergleich



Beispiel 1: Sortieren



<https://www.youtube.com/watch?v=kPRA0W1kECg>

Inhalt der Vorlesung 1/3

- **Thema allgemein**
  - Letztes Semester haben Sie (ESE) die Grundzüge des Programmierens gelernt (die meisten in Java)
  - Fragen der Performanz spielten noch kaum eine Rolle
  - Genau darum geht es jetzt in dieser Vorlesung
    - Wie schnell ist mein Programm?
    - 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 bei uns hier meistens um **Zeit**

## Inhalt der Vorlesung 2/3

- Themen im Speziellen
  - Sortieren, dynamische Felder, assoziative Felder, Hashing, Prioritätswarteschlangen, Listen, Kürzeste Wege / Dijkstra's Algorithmus, Suchbäume, etc.
  - Siehe Webseite von der Vorlesung:  
<http://lmb.informatik.uni-freiburg.de/lectures/AlgoDatESE/>
- Methodologisches
  - Laufzeitanalyse
  - O-Notation
  - Den einen oder anderen Korrektheits**beweis**
  - Die Mathematik, die wir in diesem Kurs verwenden, ist sehr "basic", aber es ist schon Mathematik, nicht nur "Rechnen,"

7

## Organisatorisches: Ablauf

- Vorlesung:
  - Donnerstags, 10:15 - 11:45 Uhr, Raum: 101 SR 00-010/014
  - Vorlesung wird aufgezeichnet und auf der Webseite bereitgestellt
- Übungen:
  - Jede Woche ein Übungsblatt
  - Betreuung/Abgabe/Korrektur alles online
  - Präsenzübung (nach Bedarf)  
Montags, 13:15 - 14:00 Uhr, Raum: 101 SR 00-010/014 (nächster Montag, 26.10. findet statt)
- Klausur:
  - voraussichtlich: Freitag, 27.2.2015, 16:00 – 18:00Uhr, Gebäude 082 Raum HS 00-006 (Kinohörsaal).

## Inhalt der Vorlesung 3/3

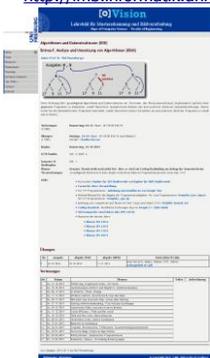
- Umsetzung
  - Sie sollen auch (weiter) **gutes Programmieren** lernen!
  - **Unit Tests** für alle nicht-trivialen Methoden ... **JUnit / Gtest**
    - **Grund** : Eine nicht-triviale Methode ohne Unit Test ist mit hoher Wahrscheinlichkeit nicht korrekt
  - Befolgen eines **Stylesheets** ... **Checkstyle / CppLint**
    - **Grund**: Damit ihr Code lesbar und verständlich ist
  - Standardisiertes **Build-Framework** ... **Ant / Make**
    - **Grund**: Damit wir nicht bei jeder Abgabe getrennt schauen müssen, wie man den Code testet / ausführt
  - Ob **Java** oder **C++** ist Ihnen überlassen

## Organisatorisches: Übungen 1/3

- Übungsblätter
  - Inhalt ca. **80%** praktisch, **20%** theoretisch
  - Wir erwarten, das **jeder** von Ihnen **jedes** Übungsblatt bearbeitet ... **wir werden aber niemanden dazu zwingen**
  - Um an der Klausur teilnehmen zu können, müssen Sie mindestens **50%** der Punkte in den Übungsblättern erreicht haben
  - Die Klausuraufgaben fragen das erworbene Wissen aus der Vorlesung **und aus den Übungen** ab

## Organisatorisches: Webseite

<http://lmb.informatik.uni-freiburg.de/lectures/AlgoDatESE/>



wichtige Links für die Übungen

Übungsblätter,  
Folien,  
Vorlesungsaufzeichnungen,

## Organisatorisches: Übungen 2/3

- Übungsbetrieb:
  - Die Tutoren sind: **Stefan Köck** und **Stefan Dilger**
  - Mitverantwortlich für die Übungsblätter ist: **Claudius Korzen**
  - Jede Woche am Ende der Vorlesung ein neues Übungsblatt
  - Bearbeitungszeit: **ESE: 1 Woche, IEMS: 3 Wochen**
  - Fragen dazu bitte im **Forum** stellen – bei Bedarf Präsenzübung am Montag
  - Abgabe der Übung durch „commit“ ins **SVN**
  - Testen, ob es bei uns compiliert, über **Jenkins-Webseite**
  - Korrektur und Bewertung bekommen Sie nach spätestens einer Woche per „update“ aus dem **SVN**
  - Überblick über ihre bisherigen Punkte, etc. auf der **Daphne-Webseite**

## Organisatorisches: Übungen 3/3

- Punktevergabe:
  - Für **Programmieraufgaben**:
    - Funktion: 60%
    - Tests: 20%
    - Doku, Checkstyle, etc. 20%
    - Wenn das Programm nicht kompiliert: 0 Punkte
  - Für **Theorieaufgaben (Beweise)**:
    - Grundidee / Ansatz: 40%
    - Ordentlich ausgearbeitet: 60%

## Forum

- Es gibt ein **Forum** zur Veranstaltung
  - Link dazu auf der Webseite und auf Ihrer Daphne Seite
  - Bitte fragen Sie, wann immer etwas nicht klar ist
  - **Nur keine Hemmungen**, auch wenn Sie denken, die Frage ist blöd ... *ist sie meistens nicht*
  - Und fragen Sie uns nicht einzeln, sondern auf dem Forum ... praktisch immer interessiert das auch andere
  - Entweder **Ich** oder **Claudius Korzen** oder einer der **TutorInnen** wird dann möglichst schnell antworten

16

## Organisatorisches: Aufwand

- Aufwand
  - Es gibt für die Veranstaltung **4 ECTS (ESE) / 6 ECTS (IEMS)**
  - Das entspricht **120 / 180** Arbeitsstunden für das Semester
  - **14** Vorlesungen à **6 / 8** Stunden Arbeit + Klausur am Ende
  - Also **4 / 6** Stunden / Übungsblatt (es gibt jede Woche eins)
  - Deadlines für **IEMS** flexibler, da berufsbegleitend

14

## SVN

- SVN = **Subversion** <http://subversion.apache.org/>
  - 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 benutzen das hier für
    - die Abgaben Ihrer Übungsblätter (Code + alles andere)
    - das Feedback von Ihrem Tutor / Ihrer Tutorin
    - Vorlesungsdateien / Musterlösungen
  - Ich werde es Ihnen heute einfach einmal vormachen ...

17

## Daphne

- Daphne ist unser **Kursverwaltungssystem**
  - Link auf der Webseite zum Kurs, **bitte anmelden!**
  - In Daphne haben Sie eine Übersicht über folgende Infos
    - Wer ihr/e Tutor/in ist
    - Ihre Punkte in den Übungsblättern
    - Info 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*

15

## Jenkins

- Jenkins ist unser **Build System**
  - Damit können Sie schauen, ob Ihr Code, so wie Sie ihn bei uns hochgeladen haben, kompiliert und läuft
    - Insbesondere ob die **Unit Tests** alle durchlaufen
    - Und ob **Checkstyle** mit allem zufrieden ist
  - Werde ich Ihnen auch heute vormachen ...

18

## Unit Tests

- Warum Unit Tests
  - **Grund 1:** Eine nicht-triviale Methode ohne Unit Test ist mit hoher Wahrscheinlichkeit nicht korrekt
  - **Grund 2:** Macht das Debuggen von größeren Programmen viel leichter und angenehmer
  - **Grund 3:** Wir und Sie selber können automatisch testen ob Ihr Code das tut was er soll
- Was ist ein "guter" Unit Test
  - Ein Unit Test soll überprüfen ob eine Methode für eine gegebene Eingabe, die gewünschte Ausgabe berechnet
  - Für mindestens **eine typische** Eingabe
  - Für mindestens **einen kritischen** "Grenzfall", wenn es denn solche gibt ... z.B. leeres Feld beim Sortieren

19

## MinSort Implementierung in Java

```
public static void minSort(int[] array) {
    int n = array.length;
    for (int i = 0; i < n - 1; i++) {
        // Compute the minimum among array[i], ..., array[n-1].
        int min = array[i];
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (array[j] < min) {
                min = array[j];
                minIndex = j;
            }
        }
        // Swap array[i] and array[minIndex].
        int tmp = array[i];
        array[i] = array[minIndex];
        array[minIndex] = tmp;
    }
}
```

## Sortieren

- Problemdefinition
  - **Eingabe:** eine Folge von  $n$  Elementen  $x_1, \dots, x_n$
  - Sowie ein (transitiver) Vergleichsoperator  $<$  der für zwei beliebige Elemente sagt, welches davon kleiner ist
    - Transitivität: Aus  $x < y$  und  $y < z$  folgt, dass  $x < z$  ist.
  - **Ausgabe:** die  $n$  Elemente in gemäß diesem Operator sortierter Reihenfolge, zum Beispiel  
 Eingabe: 17, 4, 32, 19, 8, 44, 65  
 Ausgabe: 4, 8, 17, 19, 32, 44, 65
- Wo braucht man Sortieren?
  - In praktisch **jedem** größeren Programm
  - Beispiel: Bauen eines Indexes für eine Suchmaschine

20

## MinSort Implementierung in C++

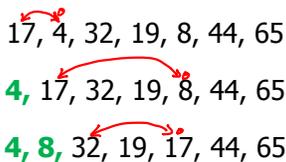
```
void Sorter::minSort(std::vector<int>* arrayP) {
    std::vector<int>& array = *arrayP;
    int n = array.size();
    // ... ab hier identisch zu Java
```

Warum fordert cplint hier die Übergabe als Zeiger (anstelle einer Referenz)?

*minSort(&arr);*

## MinSort — Algorithmus

- Informale Beschreibung
  - Finde das Minimum und tausche es an die erste Stelle
  - Finde das Minimum im Rest und tausche es an die zweite Stelle
  - Finde das Minimum im Rest und tausche es an die dritte Stelle
  - usw.



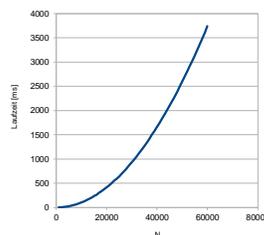
21

## MinSort — Laufzeit

1000	1,896
2000	5,237
3000	9,631
4000	16,916
5000	26,201
6000	39,112
7000	51,665
8000	67,804
9000	86,86
10000	105,503
11000	126,561
12000	150,375
13000	176,957
14000	203,978
15000	235,092
16000	265,975
17000	300,047
18000	334,941
19000	373,663
20000	412,961

- Wie lange läuft unser Programm?

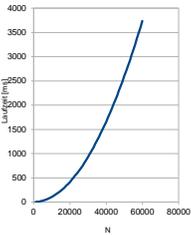
- Wir testen das mal für verschiedene Eingabegrößen
- **Beobachtung:** es wird "unproportional" langsamer, je mehr Zahlen sortiert werden



24

## MinSort — Laufzeit

### Laufzeitanalyse



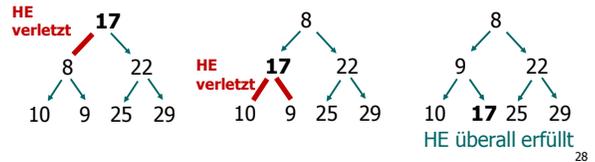
- In der Vorlesung heute erstmal ein Schaubild
  - sowas sollen Sie auch für's 1. Übungsblatt machen !
- Wir sehen an dem Schaubild
  - die Laufzeit "wächst schneller als linear"
  - D.h. für doppelt so viele Zahlen braucht es (viel) **mehr** als doppelt so viel Zeit
- In der nächsten Vorlesung werden wir genauer analysieren, was da passiert

25

## HeapSort — Algorithmus 3/5

### Reparieren nach Entfernen des Minimums

- Diese Routine wird uns im Folgenden nützlich sein
  - Entferne den Wurzelknoten (= kleinstes Element)
  - Setze den letzten Knoten an die Stelle der Wurzel
  - Lasse die neue Wurzel nach unten "durchsickern", bis die Wurzeleigenschaft wieder stimmt ... zum Beispiel:

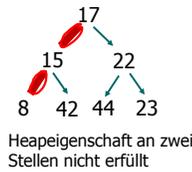
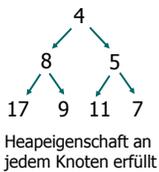


28

## HeapSort — Algorithmus 1/5

### Dasselbe Prinzip wie MinSort

- ... aber wir "verwalten" die Elemente geschickter, so dass wir das Minimum jeweils schneller berechnen können
- Genauer: in einem sogenannten **binären (Min-)Heap**
  - möglichst vollständiger binärer Baum
  - die **(Min-)Heapeigenschaft** ist erfüllt = jeder Knoten ist kleiner als seine (direkten) Kinder

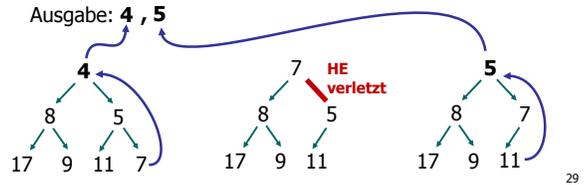


26

## HeapSort — Algorithmus 4/5

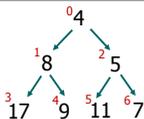
### HeapSort geht jetzt wie folgt vor

- Organisiere die  $n$  Elemente in einem Heap ... nächste Folie
- Solange noch Elemente im Heap sind:
  - Nimm das Minimum heraus
  - Setze das letzte Element an die Wurzel
  - Repariere den Heap wie gerade erklärt



29

## HeapSort — Algorithmus 2/5



0	1	2	3	4	5	6
4	8	5	17	9	11	7

### Wie speichert man einen binären Heap

- Wir numerieren die Knoten von oben nach unten und links nach rechts durch, beginnend mit 0 ... dann gilt nämlich
  - die Kinder von Knoten  $i$  sind die Knoten  $2i+1$  und  $2i+2$
  - der Elternknoten von einem Knoten  $i$  ist  $\text{floor}((i-1)/2)$
- Wir können die Elemente dann einfach in einem normalen Feld speichern:
 

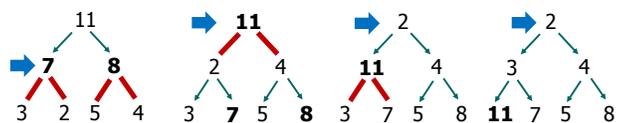
```
ArrayList<int> heap; // Java.
std::vector<int> heap; // C++.
```
- Zugriff auf den Knoten  $i$  einfach mit `heap.get(i)` bzw. `heap[i]`

27

## HeapSort — Algorithmus 5/5

### Organisieren der $n$ Elemente in einem Heap

- Diese Operation nennt man **heapify**
- Die Element stehen ja schon in einem Feld
- Fasse das einfach als binären Heap auf, bei dem die Heapeigenschaft (noch) nicht gilt
- Repariere diesen Heap "von unten nach oben", auch mit der "Durchsicker" Methode ... hier ist ein Beispiel



30

## HeapSort — Laufzeit

### ■ Intuitiv

- Zur Bestimmung des Minimums musste man bei **MinSort** in jeder Runde die ganzen übrigen Elemente durchgehen
- Bei **HeapSort** ist die Bestimmung des Minimums trivial
  - es ist einfach immer die Wurzel vom **Heap**
- Allerdings müssen wir nach dem Entfernen des Minimums, jedes Mal die **Heapeigenschaft** wieder reparieren
  - dazu müssen wir aber nur einen Teil des Baumes durchgehen, nicht alle Element in dem Baum
- Das machen wir in der nächsten Vorlesung formaler ...

31

## Schritt 1: Orderstruktur anlegen

- Melden Sie sich bei Daphne an, damit im SVN ein Verzeichnis für Sie angelegt wird (Rechenzentrums-login-Namel!)
  - Checken Sie ihr (leeres) Verzeichnis aus (ersetzen Sie den *ronneber* durch ihren Rechenzentrums-login)
- ```
svn checkout https://daphne.informatik.uni-  
freiburg.de/svn/AlgoDatEseLemsWS1415/ronneber --username=ronneber
```
- Laden Sie das template für Java oder C++ herunter und packen Sie es im erzeugten Verzeichnis aus (für C++: *template\_java.zip* durch *template\_cpp.zip* ersetzen)

```
cd ronneber  
wget http://lmb.informatik.uni-freiburg.de/lectures/AlgoDatESE/template_java.zip  
unzip template_java.zip
```

## Literatur / Links

### ■ Allgemein zur Vorlesung

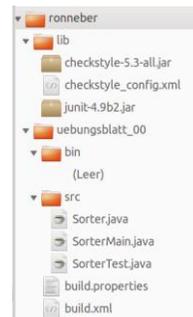
- Cormen / Leiserson / Rivest: Introduction to Algorithms  
[Klassisches Lehrbuch zu Algorithmen und Datenstrukturen. Nur das Inhaltsverzeichnis ist online, muss man kaufen oder ausleihen.]  
<http://mitpress.mit.edu/algorithms/>
- Mehlhorn / Sanders: Algorithms and Data Structures, The Basic Toolbox  
[Neueres Lehrbuch zu Algorithmen und Datenstrukturen, mit viel praktischerer Ausrichtung als der Cormen/Leiserson/Rivest. Das ganze Buch steht online!]  
<http://www.mpi-inf.mpg.de/~mehlhorn/Toolbox.html>

### ■ Sortieren

- In Mehlhorn/Sanders: 5. Sorting and Selection
- In Cormen/Leiserson/Rivest: II.7.1 HeapSort
- Wikipedia hat gute Artikel zu **MinSort** und **HeapSort**

32

## Ordnerstruktur Java



- In „lib“ stehen die Dateien für den Stylecheck und die unit-tests
- Die Abgaben für die Übungsblätter benennen Sie „uebungsblatt\_01“ etc.
- Darin: „bin“ und „src“-Ordner mit Ihren Dateien
- build.\* sind für **ant**. Sie müssen nur jeweils „build.properties“ anpassen.

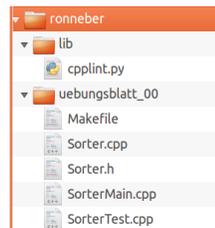
## Entwicklungsumgebung für die Übungsaufgaben

### ■ Am Beispiel von MinSort zeige ich Ihnen jetzt

- die Unit Tests ... **Junit / GTest**
- unsere Coding Standards ... **Checkstyle / CppLint**
- unser Build Framework ... **Ant / Make**
- unser Build System ... **Jenkins**
- unser **SVN**

33

## Ordnerstruktur C++



- In „lib“ stehen die Dateien für den Stylecheck
- Die Abgaben für die Übungsblätter benennen Sie „uebungsblatt\_01“ etc.
- Darin direkt ihre Dateien
- „Makefile“ für **make**. Sie müssen nichts anpassen.

## Schritt 2: Programmieren

- Schreiben Sie zuerst den Test (Dateiname muss „xyzTest.java“, bzw. „xyzTest.cpp“ sein)
- Implementieren Sie dann eine leere Klasse mit Dummy-Funktionen, damit der Test kompiliert (für C++ müssen Sie noch googletest <http://code.google.com/p/googletest> installieren)
  - Java: ant test
  - C++: make test
- Implementieren sie jetzt Stück für die Funktionalität ihrer Klasse. Stellen Sie dabei sicher, dass der Test weiterhin kompiliert
- Schreiben sie das Main-Programm (Dateiname muss „xyzMain.java“, bzw. „xyzMain.cpp“ lauten)
- Compilieren Sie das Programm und lassen es laufen
  - Java: ant all und java -jar SorterMain.jar
  - C++: make all und ./SorterMain

## Schritt 3: Übungen abgeben

- Stellen Sie ihren Source-Code unter Versionskontrolle (vorher Binärdateien löschen)
  - ant clean (in C++ make clean)
  - svn add lib (nur beim ersten Mal)
  - svn add uebungsblatt\_01
- „Committen“ Sie ihren Source-Code ins SVN
  - svn commit -m "this is my initial revision"
- Loggen Sie sich auf der Jenkins-Seite ein und testen Sie, ob der Code auf unserem Rechner fehlerfrei kompiliert

## Jenkins

The screenshot shows the Jenkins dashboard with a table of build jobs. The table has columns for 'S', 'W', 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The job 'AlgoChefWS1415-sonnaber' is circled in red.

| S | W | Name                           | Last Success | Last Failure     | Last Duration |
|---|---|--------------------------------|--------------|------------------|---------------|
| ● | ● | AlgoChefWS1415-sock1328        | 10 hr - #11  | 10 hr - #10      | 26 sec        |
| ● | ● | AlgoChefWS1415-mustentelmsagen | N/A          | 1 hr 13 min - #1 | 14 sec        |
| ● | ● | AlgoChefWS1415-sonnaber        | 10 hr - #3   | 10 hr - #1       | 26 sec        |
| ● | ● | AlgoChefWS1415-rl0             | N/A          | N/A              | N/A           |
| ● | ● | AlgoChefWS1415-ov11            | N/A          | N/A              | N/A           |
| ● | ● | AlgoChefWS1415-ak160           | N/A          | N/A              | N/A           |
| ● | ● | AlgoChefWS1415-veel            | N/A          | N/A              | N/A           |
| ● | ● | AlgoChefWS1415-ohl039          | N/A          | 11 hr - #2       | 14 sec        |