Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von Algorithmen (IEMS)
WS 2012 / 2013

Vorlesung 1, Donnerstag, 24. Oktober 2013
(Einführung, Organisatorisches, Sortieren)

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

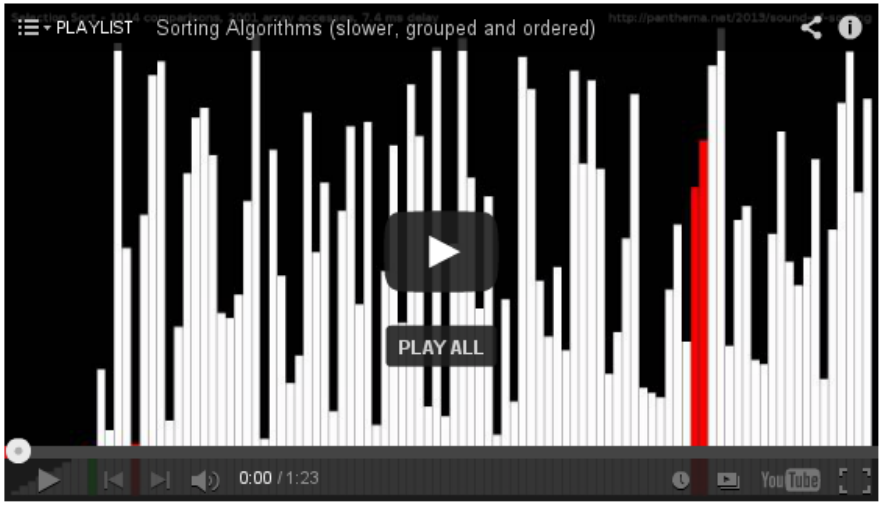
[basiert auf der letztjährigen Veranstaltung von Prof. Dr. Hannah Bast, Lehrstuhl für Algorithmen und Datenstrukturen]

Thema der Vorlesung

UNI FREIBURG

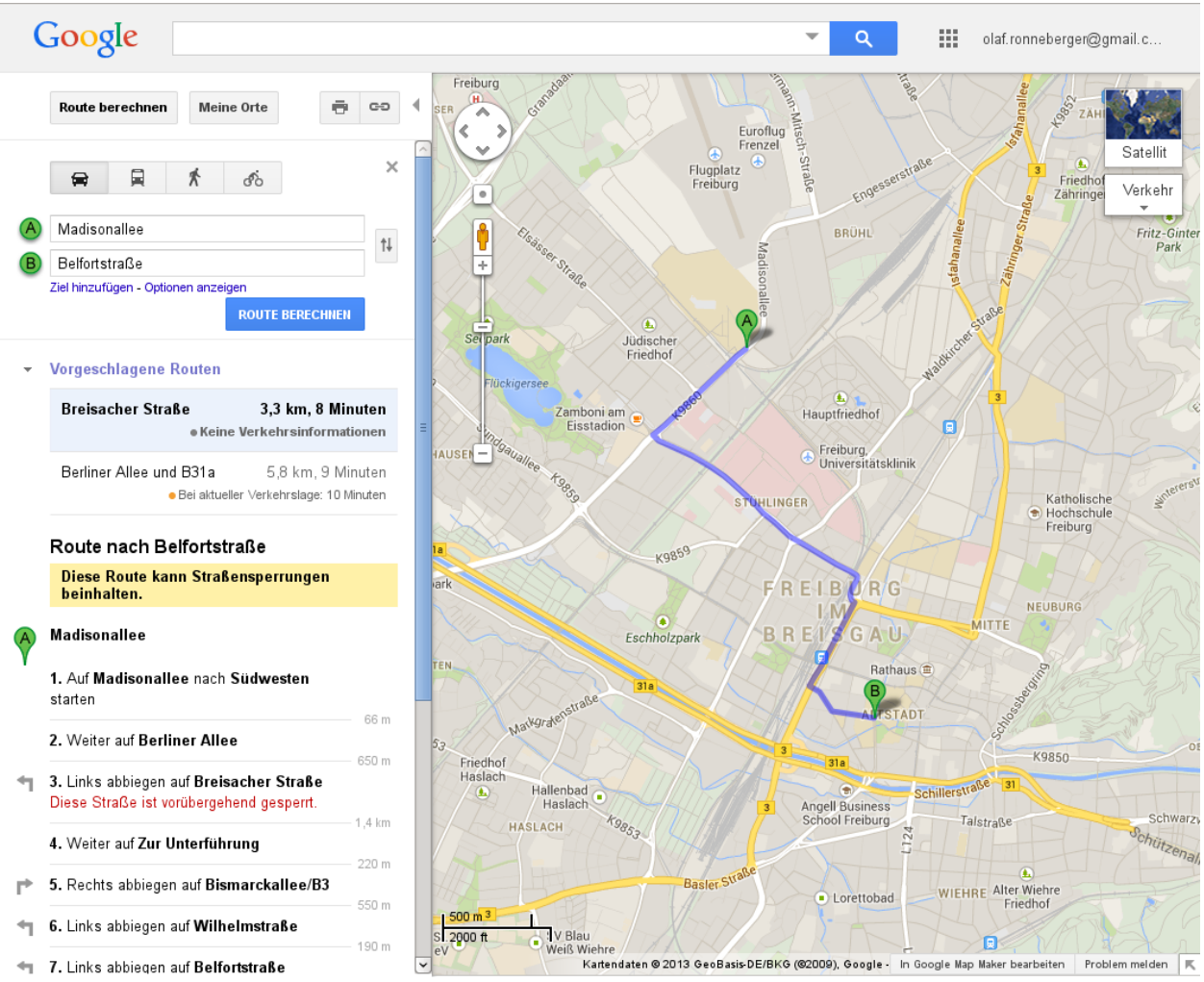
- Algorithmen und Datenstrukturen ...
  - ... für Probleme, die in praktisch **jedem** größeren Programm / Projekt auftauchen
  - Algorithmen = Verfahren um Probleme zu lösen
  - Datenstrukturen = Repräsentation der Daten dazu im Rechner

Beispiel 1: Sortieren



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

Beispiel 2: Routenplanung



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

Inhalt der Vorlesung 1/3

UNI FREIBURG

- 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

UNI FREIBURG

- Themen im Speziellen
  - Sortieren, dynamische Felder, assoziative Felder, Hashing, Prioritätswarteschlangen, Listen, Kürzeste Wege / Dijkstra's Algorithmus, Suchbäume, etc.
  - Sehr ähnlich zur Vorlesung vom WS 2012/2013 von Prof. Bast

    http://ad-wiki.informatik.uni-freiburg.de/teaching/AlgoDatEseIemsWS1213
- 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„

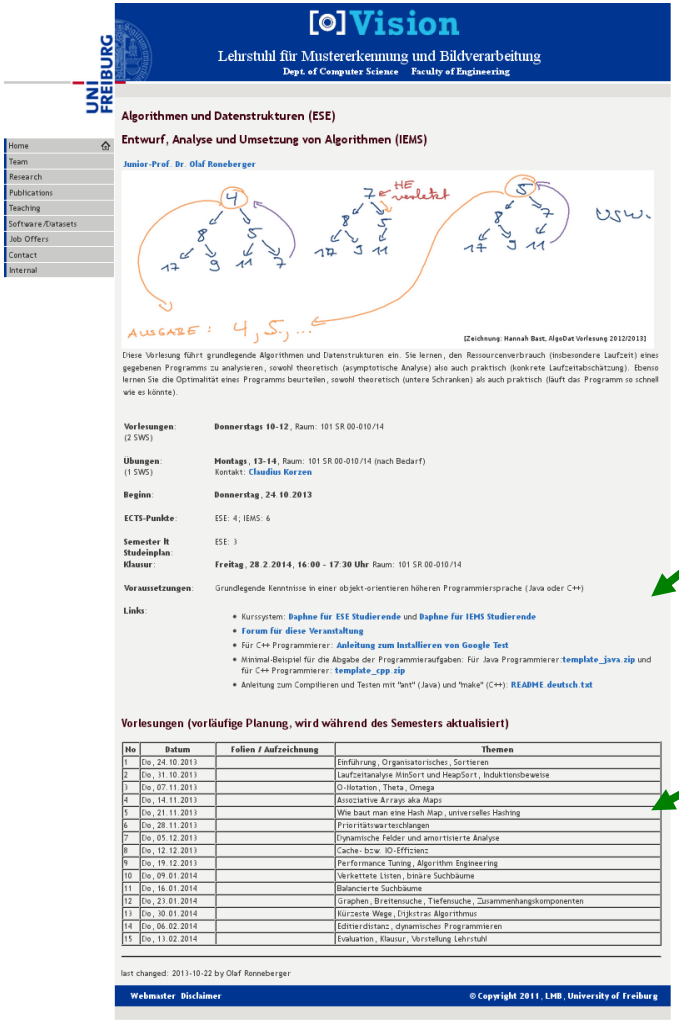
Inhalt der Vorlesung 3/3

UNI FREIBURG

- 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: Webseite

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



[◎]Vision

Lehrstuhl für Mustererkennung und Bildverarbeitung

Dept. of Computer Science Faculty of Engineering

UNI FREIBURG

Home
Team
Research
Publications
Teaching
Software/Datasets
Job Offers
Contact
Internal

Algorithmen und Datenstrukturen (ESE)

Entwurf, Analyse und Umsetzung von Algorithmen (IEMS)

Junior-Prof. Dr. Olaf Ronneberger

[Zeichnung: Hannah Bast, AlgoDat Vorlesung 2012/2013]

Diese Vorlesung führt grundlegende Algorithmen und Datenstrukturen ein. Sie lernen, den Ressourcenverbrauch (insbesondere Laufzeit) eines gegebenen Programms zu analysieren, sowohl theoretisch (asymptotische Analyse) also auch praktisch (konkrete Laufzeitabschätzung). Ebenso lernen Sie die Optimalität eines Programms beurteilen, sowohl theoretisch (untere Schranken) als auch praktisch (läuft das Programm so schnell wie es könnte).

Vorlesungen: **Donnerstags 10-12**, Raum: 101 SR 00-010/14 (2 SWS)

Übungen: **Montags, 13-14**, Raum: 101 SR 00-010/14 (nach Bedarf) (1 SWS) Kontakt: **Claudius Korzen**

Beginn: **Donnerstag, 24.10.2013**

ECTS-Punkte: ESE: 4; IEMS: 6

Semester lt Studienplan: ESE: 3

Klausur: **Freitag, 28.2.2014, 16:00 - 17:30 Uhr** Raum: 101 SR 00-010/14

Voraussetzungen: Grundlegende Kenntnisse in einer objekt-orientieren höheren Programmiersprache (Java oder C++)

Links:

- Kurssystem: **Daphne für ESE Studierende** und **Daphne für IEMS Studierende**
- **Forum für diese Veranstaltung**
- Für C++ Programmierer: **Anleitung zum Installieren von Google Test**
- Minimal-Beispiel für die Abgabe der Programmieraufgaben: Für Java Programmierer: **template_java.zip** und für C++ Programmierer: **template_cpp.zip**
- Anleitung zum Compilieren und Testen mit "ant" (Java) und "make" (C++): **README deutsch.txt**

Vorlesungen (vorläufige Planung, wird während des Semesters aktualisiert)

No	Datum	Folien / Aufzeichnung	Themen
1	Do, 24.10.2013		Einführung, Organisatorisches, Sortieren
2	Do, 31.10.2013		Laufzeitanalyse MinSort und HeapSort, Induktionsbeweise
3	Do, 07.11.2013		O-Notation, Theta, Omega
4	Do, 14.11.2013		Assoziative Arrays aka Maps
5	Do, 21.11.2013		Wie baut man eine Hash Map, universelles Hashing
6	Do, 28.11.2013		Prioritätswarteschlangen
7	Do, 05.12.2013		Dynamische Felder und amortisierte Analyse
8	Do, 12.12.2013		Cache- bzw. IO-Effizienz
9	Do, 19.12.2013		Performance Tuning, Algorithm Engineering
10	Do, 09.01.2014		Verkettete Listen, binäre Suchbäume
11	Do, 16.01.2014		Balancierte Suchbäume
12	Do, 23.01.2014		Graphen, Breitensuche, Tiefensuche, Zusammenhangskomponenten
13	Do, 30.01.2014		Kürzeste Wege, Dijkstras Algorithmus
14	Do, 06.02.2014		Editierdistanz, dynamisches Programmieren
15	Do, 13.02.2014		Evaluation, Klausur, Vorstellung Lehrstuhl

last changed: 2013-10-22 by Olaf Ronneberger

Webmaster Disclaimer © Copyright 2011, LMB, University of Freiburg

wichtige Links für die Übungen

Folien,
Vorlesungensaufzeichnungen,
Übungsblätter

Organisatorisches: Ablauf

UNI FREIBURG

- 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, 27.10. findet statt)
- Klausur:
  - Freitag, 28.2.2014, 16:00 - 17:30Uhr, Raum: 101 SR 00-010/014

Organisatorisches: Übungen 1/3

UNI FREIBURG

- Übungsblätter
  - Inhalt ca. 80% praktisch, 20% theoretisch
  - Wir erwarten, das jeder von Ihnen jedes Übungsblatt bearbeitet … wir werden aber niemanden dazu zwingen
  - Um sinnvoll an der Klausur teilnehmen zu können, sollten 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: Übungen 2/3

UNI FREIBURG

- Übungsbetrieb:
  - Die Tutoren sind: Mathieu Wacker, Stefan Köck
  - 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

UNI FREIBURG

- 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%

Organisatorisches: Aufwand

UNI FREIBURG

- 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

Daphne

UNI FREIBURG

- 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

Forum

UNI FREIBURG

- 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

SVN

UNI FREIBURG

- 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 ...

Jenkins

UNI FREIBURG

- 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 ...

Unit Tests

UNI FREIBURG

- 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 mindestes **einen kritischen** "Grenzfall", wenn es denn solche gibt ... z.B. leeres Feld beim Sortieren

Sortieren

- Problemdefinition
  - **Eingabe:** eine Folge von n Elementen $x_1, ..., x_n$
  - 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:
- Wo braucht man Sortieren?
  - In praktisch **jedem** größeren Programm
  - Beispiel: Bauen eines Indexes für eine Suchmaschine

MinSort — Algorithmus

UNI FREIBURG

- 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.

17, 4, 32, 19, 8, 44, 65

4, 17, 32, 19, 8, 44, 65

4, 8, 32, 19, 17, 44, 65

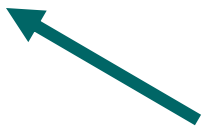
MinSort Implementierung in Java

UNI FREIBURG

```
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;
    }
}
```

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
```

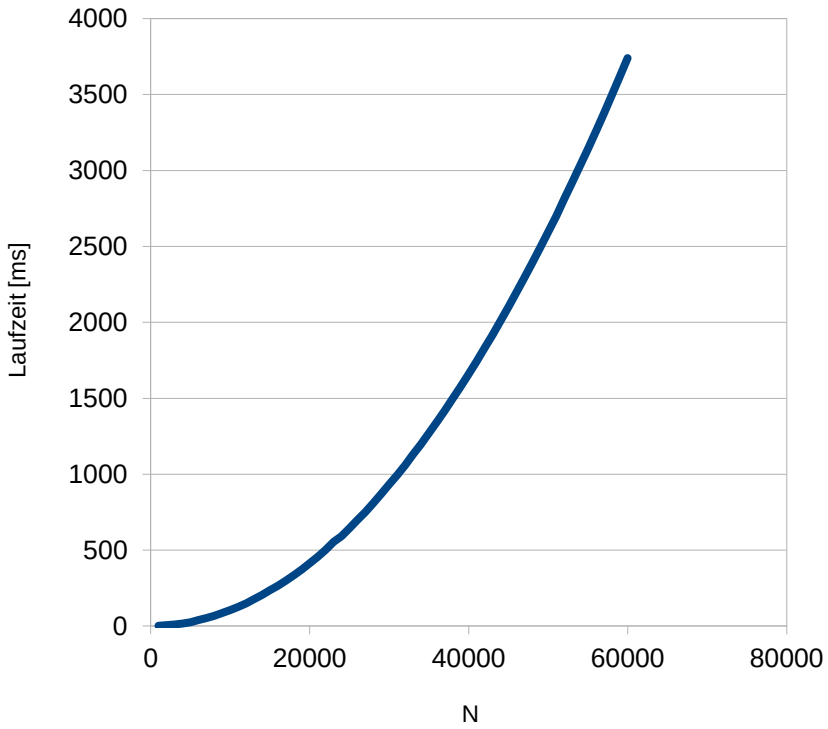


Warum fordert cpplint hier die Übergabe als Zeiger (anstelle einer Referenz)?

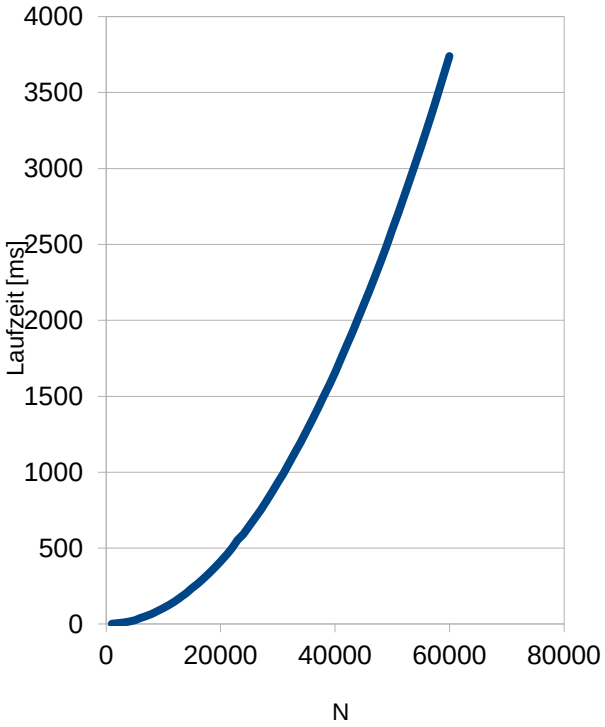
MinSort — Laufzeit

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



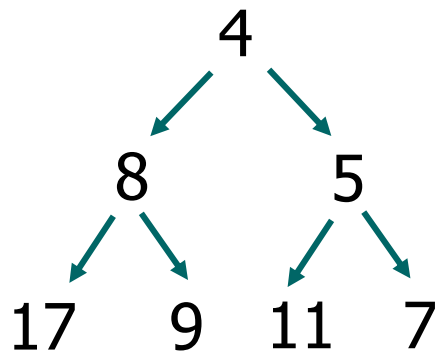
MinSort — Laufzeit



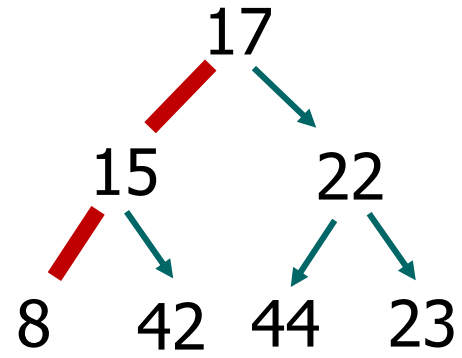
- Laufzeitananalyse
  - In der Vorlesung heute erstmal ein Schaubild
    - sowas sollen Sie auch für's 1. Übungsblatt machen !
  - Wir sehen an dem Schaubild
    - die Laufzeit "wächst schneller als linear"
    - D.h. für doppelt so viele Zahlen braucht es (viel) **mehr** als doppelt so viel Zeit
  -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

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

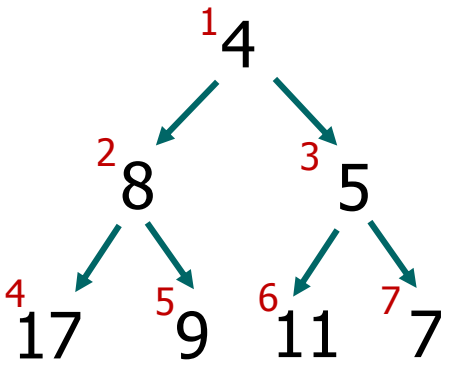


Heapeigenschaft an jedem Knoten erfüllt



Heapeigenschaft an zwei Stellen nicht erfüllt

HeapSort — Algorithmus 2/5



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

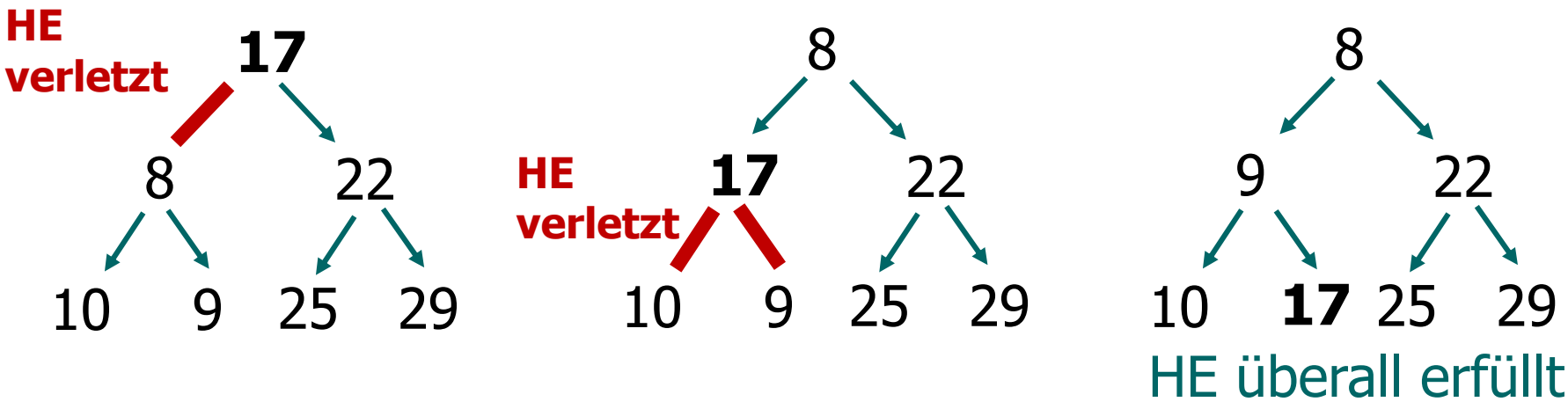
- Wie speichert man einen binären Heap
  - Wir numerieren die Knoten von oben nach unten und links nach rechts durch, beginnend mit **1** **...** dann gilt nämlich
    - die Kinder von Knoten i sind die Knoten 2i und 2i + 1
    - der Elternknoten von einem Knoten i ist floor(i/2)
  - Wir können die Elemente 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]

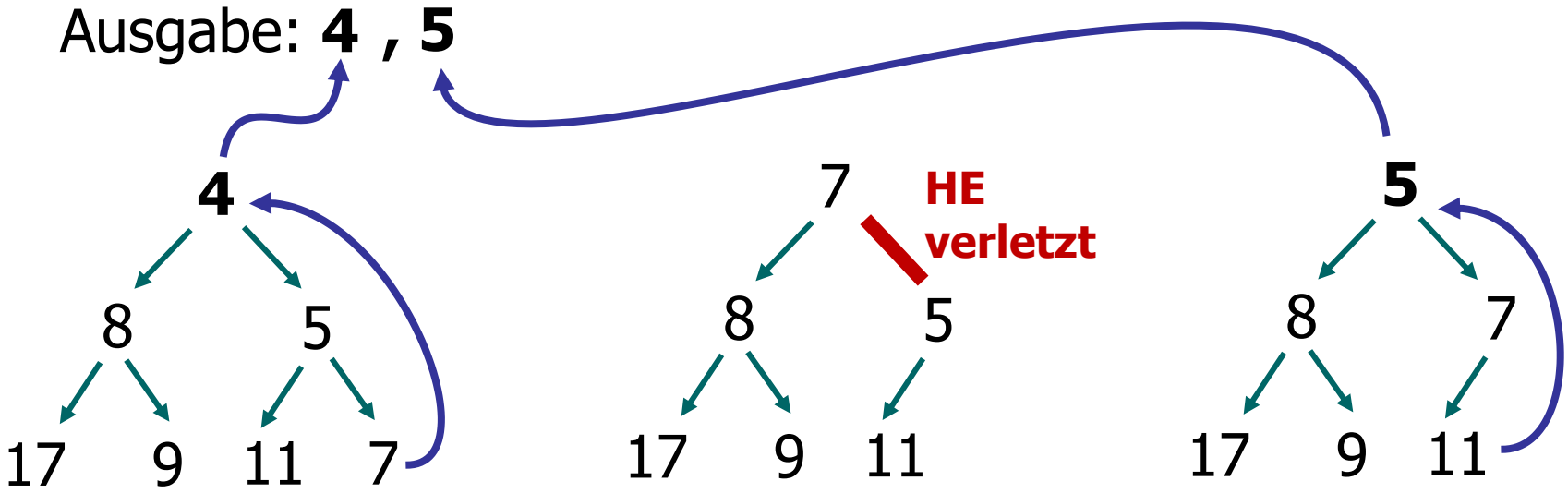
HeapSort — Algorithmus 3/5

- 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:



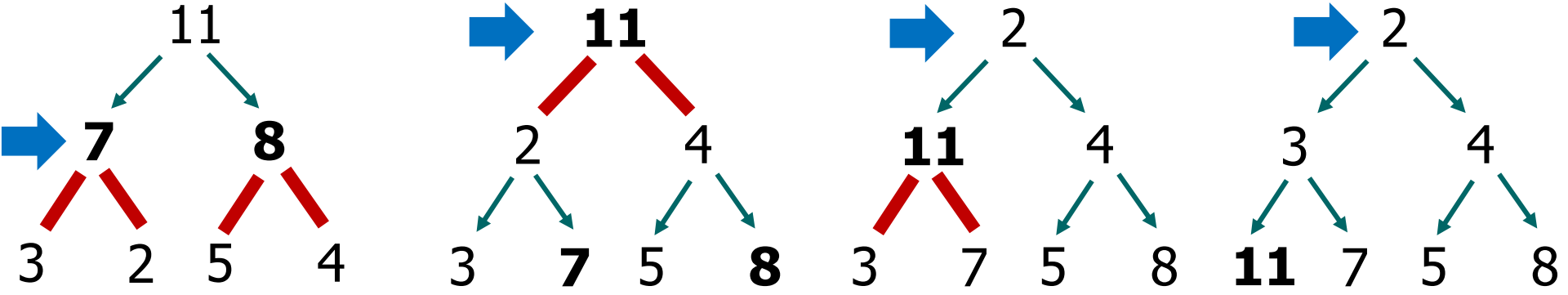
HeapSort — Algorithmus 4/5

- 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



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



HeapSort — Laufzeit

UNI FREIBURG

- 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 Minimus, 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 ...

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

Entwicklungsumgebung für die Übungsaufgaben

UNI FREIBURG

- 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

Schritt 1: Orderstruktur anlegen

UNI FREIBURG

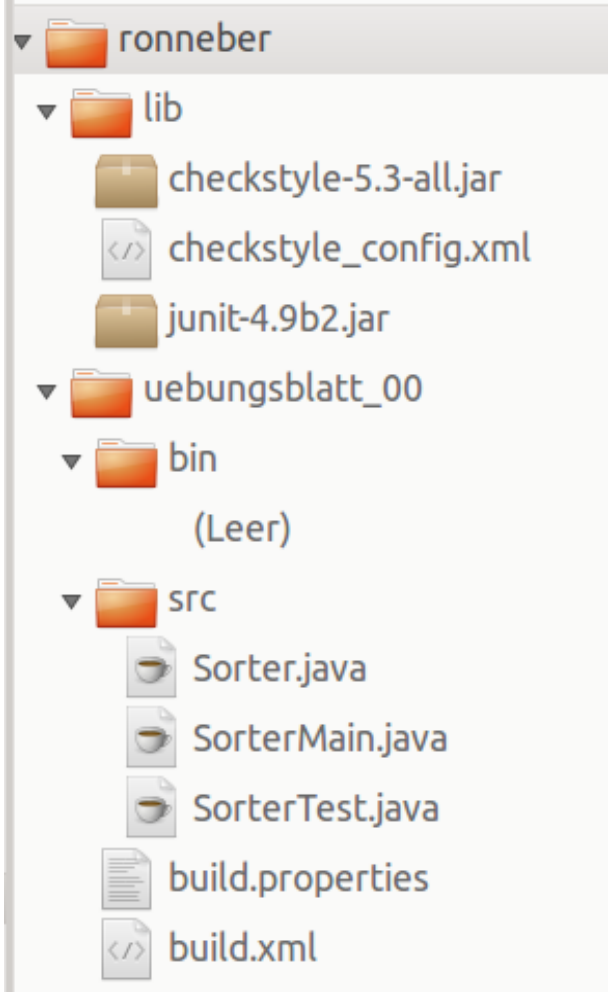
- Melden Sie sich bei Daphne an, damit im SVN ein Verzeichnis für Sie angelegt wird
- Checken Sie ihr (leeres) Verzeichnis aus (ersetzen Sie den ronneber durch ihren login)

  svn checkout https://daphne.informatik.uni-freiburg.de/svn/AlgoDatEseIemsWS1314/*ronneber* --username=*ronneber*
- Laden Sie das template für Java oder C++ herunter und packen Sie es im erzeugten Verzeichnis aus

  cd ronneber

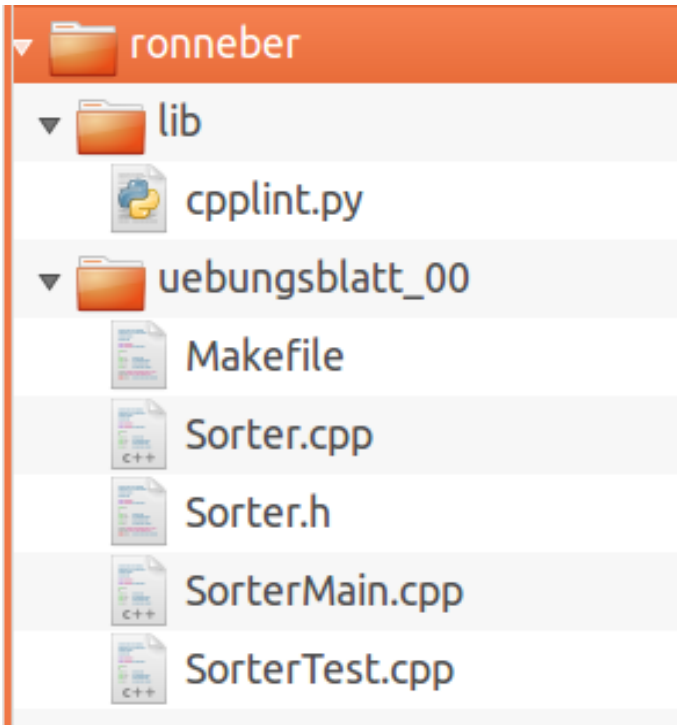
  unzip ../template_java.zip

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.

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

UNI FREIBURG

- 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
  - **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

UNI FREIBURG

- „Comitten“ Sie ihren Source-Code ins SVN
  - svn commit
- Loggen Sie sich auf der Jenkins-Seite ein und testen Sie, ob der Code auf unserem Rechner fehlerfrei kompiliert