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

Vorlesung 1, Dienstag, 23. Oktober 2012
(Einführung, Organisatorisches, Sortieren)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Allgemeines zu dieser 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**

Allgemeines zu dieser 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 2011/2012

    http://ad-wiki.informatik.uni-freiburg.de/teaching/AlgoDatEseWS1112
- 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"

Allgemeines zu dieser Vorlesung 3/3

UNI FREIBURG

- Art und Weise
  - Großer Praxisbezug (wie in allen meinen Vorlesungen)
  - Aber auch Theorie (sofern sie der Praxis nutzt)
  - Übungsblätter 80% praktisch, 20% theoretisch
  - Zum Ablauf des Übungsbetriebs gleich mehr …
- 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

Übungen

UNI FREIBURG

- Übungsblätter
  - Um an der Klausur teilnehmen zu können, brauchen Sie mindestens 50% der Punkte in den Übungsblättern
  - Wer die Übungsblätter alle macht, braucht am Ende kaum noch was zu lernen für die Klausur
  - Abgabe / Korrektur über unser SVN … später mehr dazu
- Übungsgruppen
  - Die Tutoren sind: Katja Faist, Markus Näther, Mathieu Wacker
  - Mitverantwortlich für die Übungsblätter ist: Claudius Korzen
  - Kein Übungsgruppentermin, es läuft alles **online**
    - für alle Fragen etc. gibt es das **Forum** … gleich mehr dazu
    - bei Bedarf auch eins-zu-eins Treffen mit TutorIn möglich

Note / Klausur

UNI FREIBURG

- Die **Endnote** ergibt sich folgendermaßen
  - Die Punkte aus den Übungsblättern werden auf max. 20 Punkte skaliert → Ü
  - In der Abschlussklausur wird es 4 Aufgaben à jeweils 20 Punkte geben → K1, K2, K3, K4
    - Termin (alle): Samstag, 23. März 2013, 11 – 13 Uhr
  - Von Ü, K1, K2, K3, K4 werden die besten 4 genommen und aufsummiert (also maximal 80 Punkte)
  - Aus dieser Summe wird die Endnote ermittelt
  - Schummeln / Plagiat bei den Übungsblättern bzw. bei der Klausur hat wie immer Nicht-Bestehen zur Folge

    (einmal reicht, das tut man ja nicht aus Versehen)

Daphne

- Daphne ist unser **Kursverwaltungssystem**
  - Link auf dem Wiki 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 dem Wiki 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 …

Coding Standards

UNI FREIBURG

- Neben dem eigentlichen Thema (AlgoDat)
  - ... sollen Sie auch (weiter) **gutes Programmieren** lernen **!**
  - Dazu ein paar bewährte Standards
    - Unit Tests für alle nicht-trivialen Methoden ... Junit / GTest

      Grund : Siehe nächste Folie
    - 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
  - Ich werde Ihnen das heute alles einmal vormachen ...
  - Ob Java oder C++ ist Ihnen überlassen !

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

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 …

Sortieren

UNI FREIBURG

- Problemdefinition
  - **Eingabe:** eine Folge von n Elementen $x_1, \ldots, x_n$
  - Sowie ein (transitiver) Vergleichsoperator $<$ der für zwei beliebige Elemente sagt, welches davon kleiner ist

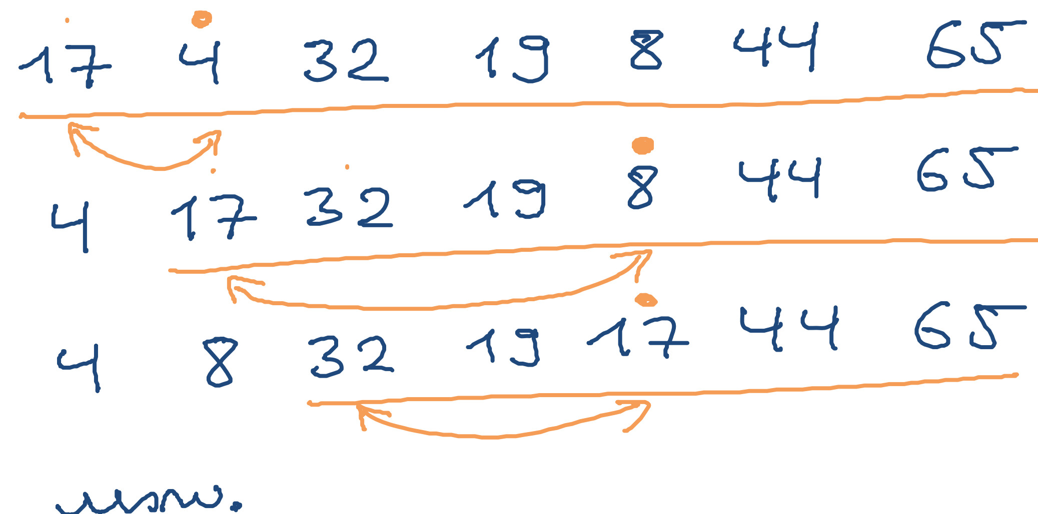
    $x < y$ und $y < z$ $\Rightarrow$ $x < z$
  - **Ausgabe:** die n Elemente in gemäß diesem Operator sortierter Reihenfolge, zum Beispiel

    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

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.



MinSort — Programm

- Das schreiben wir jetzt zusammen
  - Bei der Gelegenheit zeige ich Ihnen dann auch gleich wie das alles geht mit
    - den Unit Tests … Junit / GTest
    - unseren Coding Standards … Checkstyle / Cpplint
    - unserem Build Framework … Ant / Make
    - unserem Build System … Jenkins
    - unserem SVN

MinSort — Laufzeit

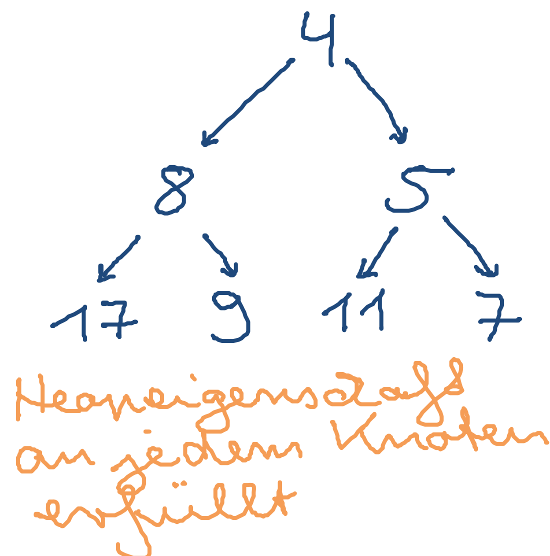
UNI FREIBURG

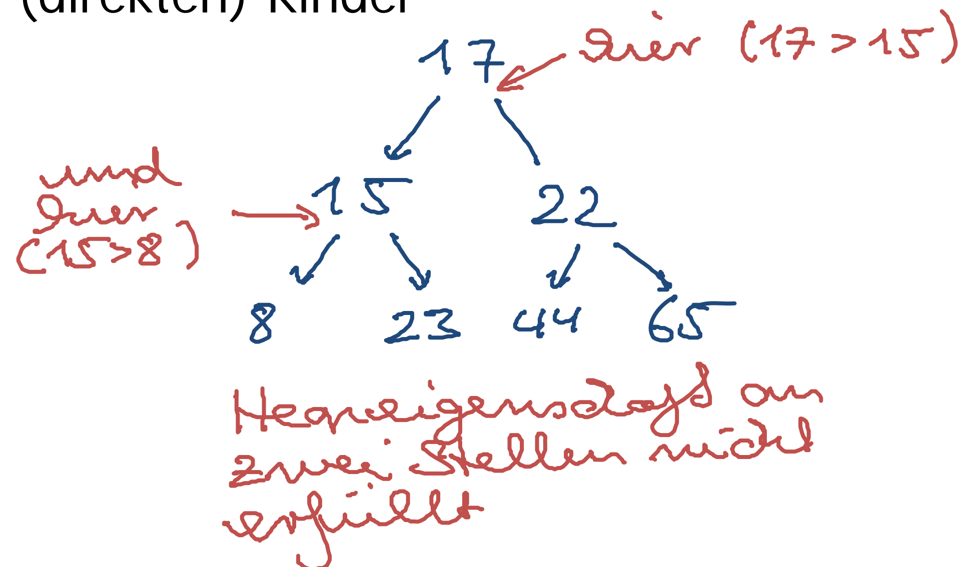
- Wie lange läuft unser Programm?
  - Wir testen das mal für verschiedene Eingabegrößen
  - **Beobachtung:** es wird "unproportional" langsamer, je mehr Zahlen sortiert werden
  - In der nächsten Vorlesung werden wir genauer analysieren, was da passiert
  - In der Vorlesung heute erstmal ein Schaubild
    - 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

HeapSort — Algorithmus 1/5

UNI FREIBURG

- 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



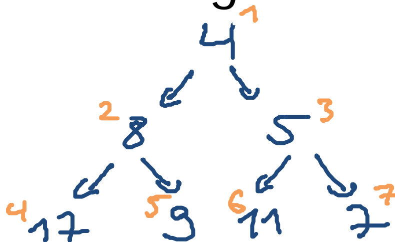


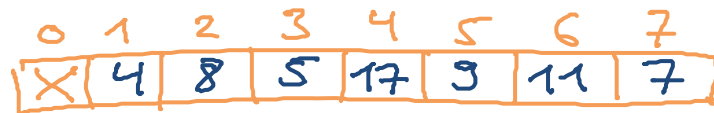
HeapSort — Algorithmus 2/5

- 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.

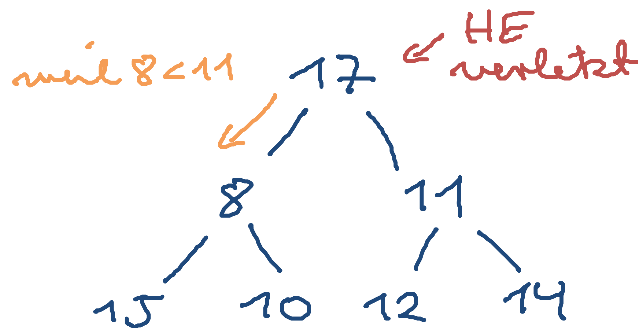
    std::vector<int> heap; // C++.
  - Zugriff auf den Knoten i einfach mit heap.get(i) bzw. heap[i]

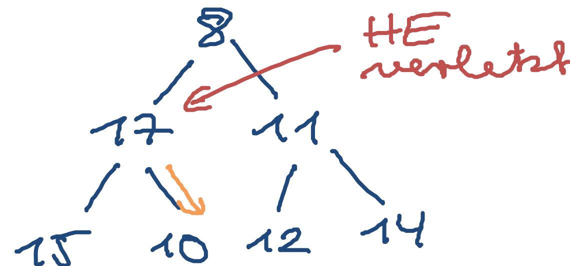


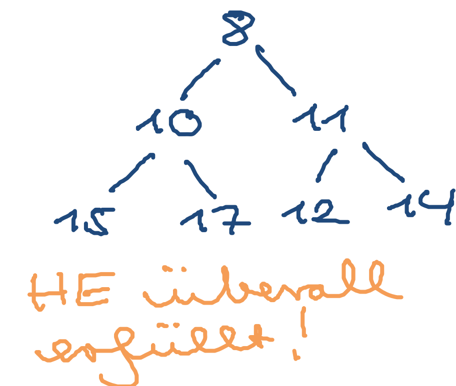


HeapSort — Algorithmus 3/5

- 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:

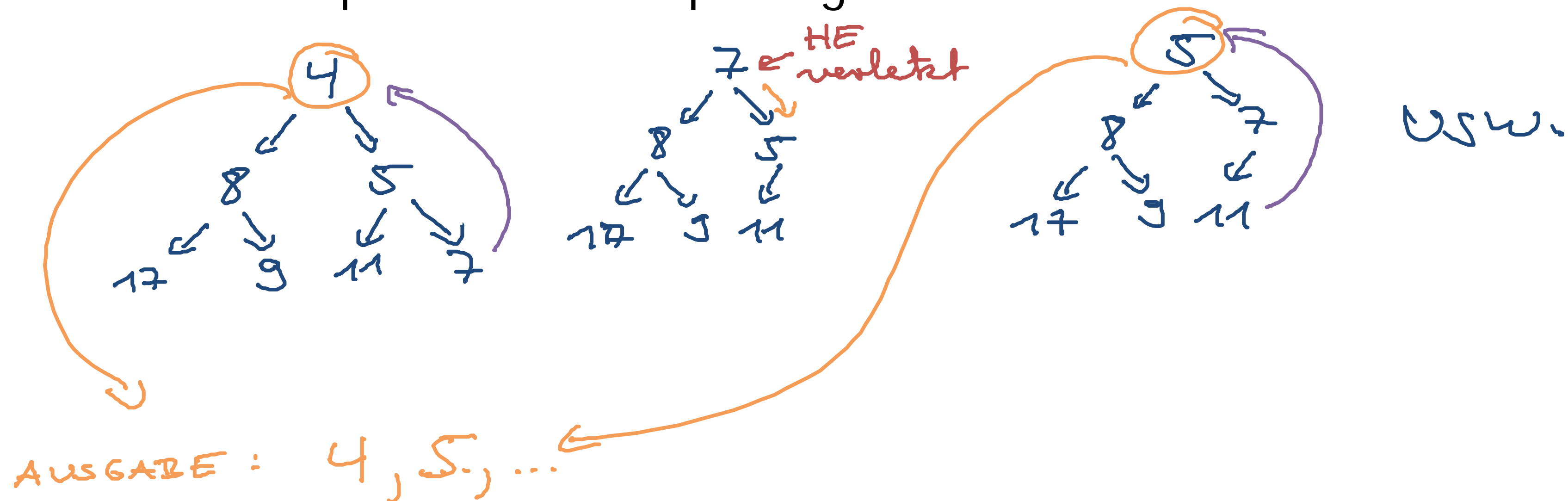






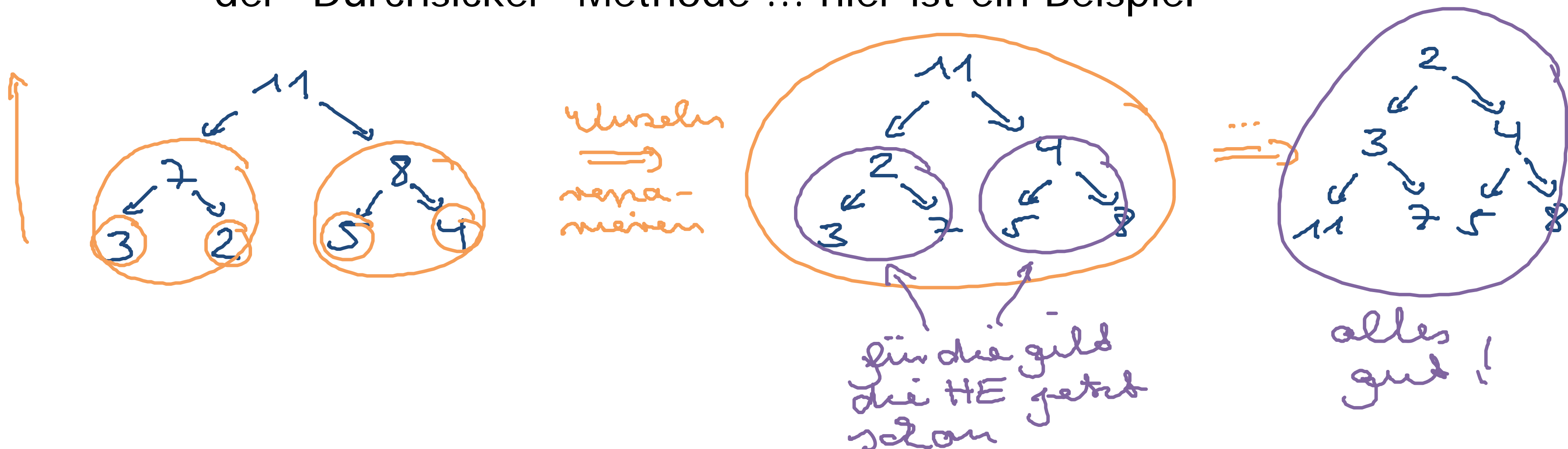
HeapSort — Algorithmus 4/5

- 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