Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 2, Montag, 25. Oktober 2010
(Sortieren, Sortieren, Sortieren)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Neuer Übungsgruppentermin am Do oder Fr
  - Ihre Erfahrungen mit dem 1. Übungsblatt
- Sortieren
  - MinSort: Implementierung + Analyse
  - QuickSort: Implementierung + Analyse
  - MergeSort: Ihre Aufgabe für das 2. Übungsblatt
  - Eine untere Schranke für Sortieren

Neuer Übungsgruppentermin

UNI FREIBURG

- Grund
  - Vorlesung und Abgabetermin ist am Montag
  - Der aktuelle Übungsgruppentermin ist am Dienstag
  - So schnell können die Tutoren nicht korrigieren
  - Aber von Mo bis Do oder Fr ginge
  - Alternative: Übungsgruppentermin weiterhin am Dienstag, aber 8 Tage nach dem Abgabetermin = 15 Tage nach dem Ausgabetermin, finde ich persönlich sehr lange
- Neuer Termin
  - Umfrage auf dem Forum ergab keinen eindeutigen Gewinner
  - Die Frage ist, wann Sie können, nicht wann Sie wollen

Ihre Erfahrungen mit dem 1. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Viele fanden es zu mathematisch und zu schwierig
  - Allgemeines Problem mit mathematischen Beweisen
  - Knobelaufgaben seien doof als Übungsaufgaben
  - Beweis wurde in der Vorlesung nicht gut genug erklärt
- Mein Fazit daraus
  - Wir werden schon weiterhin (auch) Beweise machen
  - Ich hoffe, Sie werden sehen, dass man das braucht
  - Ich werde aber darauf achten, Sie nicht zu überfordern
  - Wir müssen halt zusammen nachholen, was offenbar an anderer Stelle versäumt wurde
  - Das in der 1. Vorlesung war ja elementare Mathematik (und wir werden auch selten Tieferes benötigen)

Sortieren

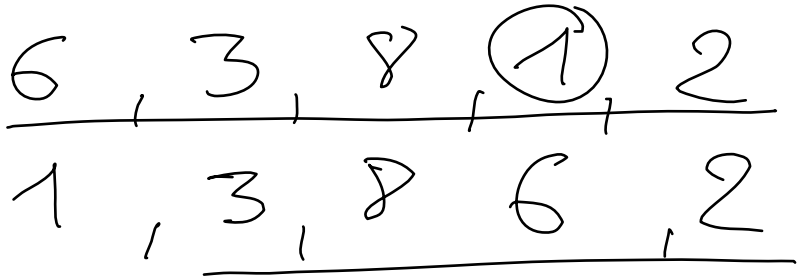
- Problemdefinition
  - Gegeben eine Folge von n Elementen $x_1, \ldots, x_n$
  - Sowie ein (transitiver) Vergleichsoperator $<$ der für zwei beliebige Elemente sagt, welches davon kleiner ist
  - Ausgabe: die n Elemente in sortierter Reihenfolge, also für eine Permutation σ der Zahlen $1..n$ eine Folge $x_{\sigma(1)}, \ldots, x_{\sigma(n)}$ so dass $x_{\sigma(1)} < \cdots < x_{\sigma(n)}$
- Wo braucht man Sortieren?
  - In jedem größeren Programm
  - Zum Beispiel unser AnagramFinder aus C++ SS2010, Ü9

Eingabe: 5, 7, 1, 3, 10 (n=5)

Ausgabe: 1, 3, 5, 7, 10

MinSort — Algorithmus und Programm

- Informaler Algorithmus
  - Finde das Minimum und setze es an die erste Stelle
  - Finde das Minimum im Rest und setze es an die zweite Stelle
  - Finde das Minimum im Rest und setze es an die dritte Stelle
  - usw.
- Schreiben wir erstmal das Progamm zusammen
  - Tests sind schon vorbereitet
  - Ebenso eine einfache Visualisierung des Algorithmus



MinSort — Analyse

- Wir interessieren uns für die Laufzeit T
  - Wir nehmen an $A_1 \cdot m \leq T \leq A_2 \cdot m$, wobei m die Anzahl der Elementvergleiche und A_1 und A_2 Konstanten sind
  - Lemma: Es gilt $A_1 / 4 \cdot n^2 \leq T(n) \leq A_2 / 2 \cdot n^2$ für $n \geq 2$
  - Das nennt man "quadratische Laufzeit" (wegen dem n^2)
  - Beweis des Lemmas:

Runde 1: $n-1$ Vergleiche

Runde 2: $n-2$ Vergleiche

usw.

Insgesamt $\sum_{i=0}^{n-1} i = \frac{1}{2} \overbrace{(n-1)}^{\geq n/2} \cdot \overbrace{n}^{\leq n} \leq n^2/2$, $\geq n^2/4$ für $n \geq 2$

Quadratische Laufzeit

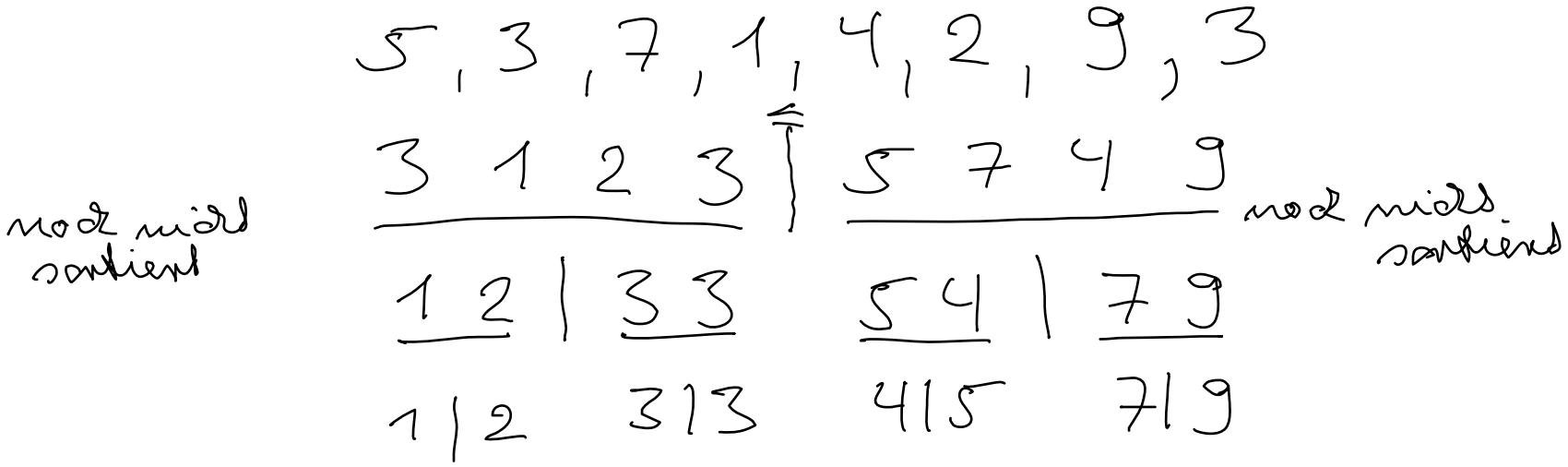
UNI FREIBURG

- Definition
  - Die Laufzeit T hängt von der Eingabegröße n ab
  - Es gibt Konstanten A_1 und A_2 mit $A_1 \cdot n^2 \leq T(n) \leq A_2 \cdot n^2$
- Betrachtungen dazu $(2n)^2 = 4n^2$
  - Doppelt so große Eingabe → viermal so große Laufzeit
  - Unabhängig von den Konstanten wird das schnell sehr teuer
    - $A_1 = 1$ ns (1 Nanosekunde ≈ 1 einfache Anweisung)
    - $n = 10^6$ (1 Millionen Zahlen = 4 MB) [bei 4 Bytes/Zahl]
      - $A_1 \cdot n^2 = 10^{-9} \cdot 10^{12} = 10^3$ s = 16.7 Minuten
    - $n = 10^9$ (1 Milliarde Zahlen = 4 GB)
      - $A_1 \cdot n^2 = 10^{-9} \cdot 10^{18} = 10^9$ s = 31.7 Jahre

Geht Sortieren auch schneller als quadratisch?

QuickSort — Algorithmus und Programm

- Informaler Algorithmus
  - Teile die Eingabe in zwei Hälften, links und rechts
  - Alle Elemente links sind kleiner als alle Elemente recht
  - Dann sortiere die beiden Hälften "rekursiv"
  - Rekursiv = ein kleineres Unterproblem von derselben Sorte mit demselben Verfahren lösen
- Beispiel:



QuickSort — Analyse 1/2

- Wir interessieren uns wieder für die Laufzeit T(n)
  - Wieder $A_1 \cdot m \leq T \leq A_2 \cdot m$, wobei m die Anzahl der Elementvergleiche und A_1 und A_2 Konstanten sind
  - Lemma:
  - Beweis:

Wir nehmen an (der Einfachheit halber), erstmal
dass das Feld immer in zwei (fast)
gleich große Hälften geteilt wird.
Genauer $n \to \lceil \frac{n}{2} \rceil$, $\lfloor \frac{n}{2} \rfloor$

QuickSort — Analyse 2/2

Dann gilt: $T(n) \leq T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + A \cdot n$

Das nennt man eine „Rekursionsgleichung"

Sei (wieder der Einfachheit halber) $n = 2^k$

$T(n) = T(2^k) \leq T(2^{k-1}) + T(2^{k-1}) + A \cdot 2^k$

$= 2 \cdot T(2^{k-1}) + A \cdot 2^k$

$T(1) \leq A$

$\leq 2 \cdot \left(2 \cdot T(2^{k-2}) + A \cdot 2^{k-1} \right) + A \cdot 2^k$

$k = \log_2 n$

$= 4 \cdot T(2^{k-2}) + A \cdot 2^k + A \cdot 2^k$

$\leq \log_2 n$

$(*) = A \cdot n \cdot (1 + \log_2 n)$

$\leq 8 \cdot T(2^{k-3}) + A \cdot 2^k + A \cdot 2^k + A \cdot 2^k$

$\leq 2A \cdot n \cdot \log_2 n$ für $n \geq 2$

$\leq 2^k \cdot T(1) + k \cdot A \cdot 2^k$

$\leq A \cdot n + \log_2 n \cdot A \cdot n$ $(*)$

Laufzeit proportional zu n · log n

- Schauen wir uns wieder Zahlenbeispiele an
  - Nehmen wir also an, es gibt Konstanten A_1 und A_2 mit
    $A_1 \cdot n \cdot \log n \leq T(n) \leq A_2 \cdot n \cdot \log n$ für $n \geq 2$
  - Dann dauert es bei doppelt so großer Eingabe nur geringfügig mehr als doppelt so lange
  - Im Folgenden sei log der Logarithmus zur Basis 2
  - $A_1 = 1$ ns (1 Nanosekunde ≈ 1 einfache Anweisung)
  - $n = 2^{20}$ (≈ 1 Millionen Zahlen = 4 MB) [bei 4 Bytes/Zahl]
    - $A_1 \cdot n \cdot \log n = 10^{-9} \cdot 2^{20} \cdot 20$ s = 21 Millisekunden
  - $n = 2^{30}$ (≈ 1 Milliarde Zahlen = 4 GB)
    - $A_1 \cdot n \cdot \log n = 10^{-9} \cdot 2^{30} \cdot 30$ s = 32 Sekunden

$$\log_b x = \frac{\ln x}{\ln b}$$

Laufzeit n · log n ist also fast so gut wie linear!

QuickSort — Worst Case

UNI FREIBURG

- Die $n \cdot \log n$ Laufzeit galt unter einer Voraussetzung
  - Nämlich dass wir es in jedem Schritt schaffen, die Zahlen immer genau in zwei Hälften zu teilen
  - Unsere Implementierung nimmt aber irgendein Element als Splitter m und teilt dann in $\leq m$ und $> m$
  - Im schlechtesten Fall sind alle Elemente außer dem Splitter selber $> m$
  - Dann haben wir ein Subproblem der Größe $n - 1$
  - Und wenn es im nächsten Schritt wieder so schlecht läuft, ein Subproblem der Größe $n - 2$ usw.
  - Dann sind wir wieder bei quadratischer Laufzeit, zumindest wenn es schlecht läuft

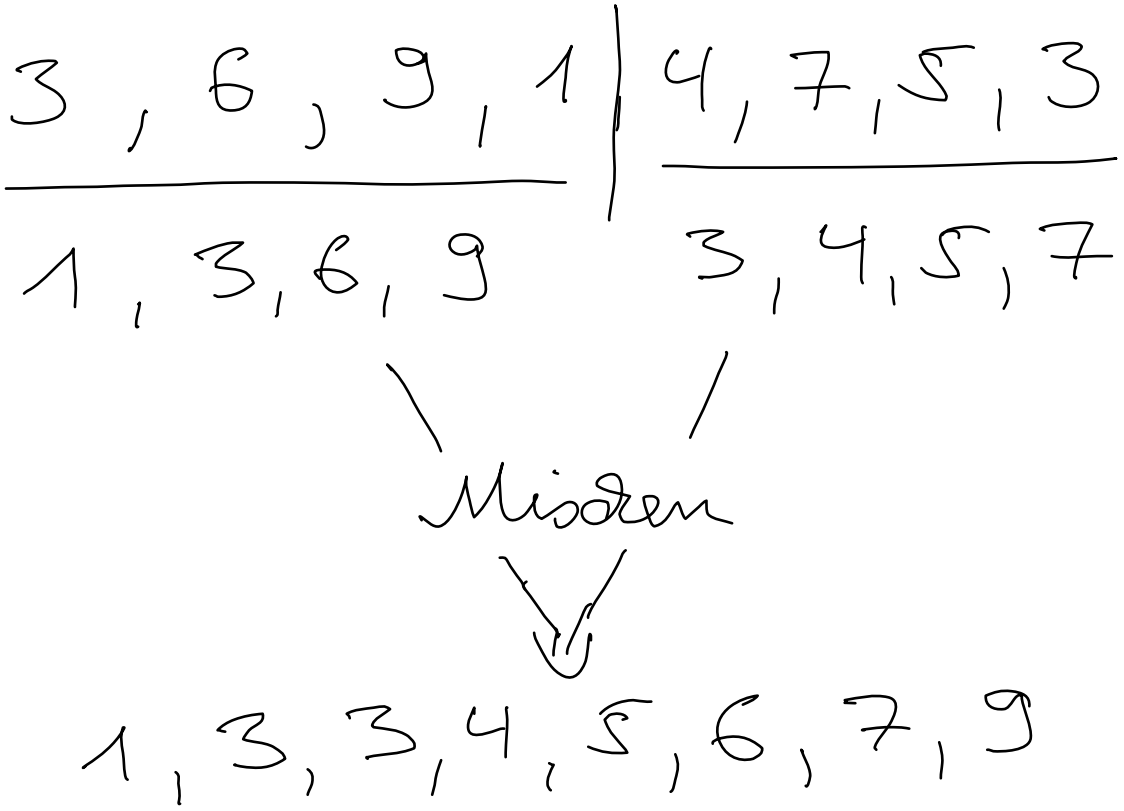
Geht auch Sortieren in $n \cdot \log n$ in jedem Fall?

MergeSort

- Informaler Algorithmus
  - Teile wieder die Eingabe in zwei Hälften, aber diesmal muss nicht gelten, dass alle links $\leq$ sind als alle rechts
  - Dann können wir einfach das Eingabefeld in der Mitte teilen
  - Wenn wir die beiden Hälften sortiert haben, sind wir jetzt aber noch nicht fertig
  - Wir müssen die beiden sortierten Teilfolgen noch zu einer einzigen sortierten Folge "verarbeiten"
  - Das nennt man "Mischen" (Englisch: "merge")
  - Mischen geht in linearer Zeit
  - Für die Laufzeit gilt dann wie im besten Fall von QuickSort:

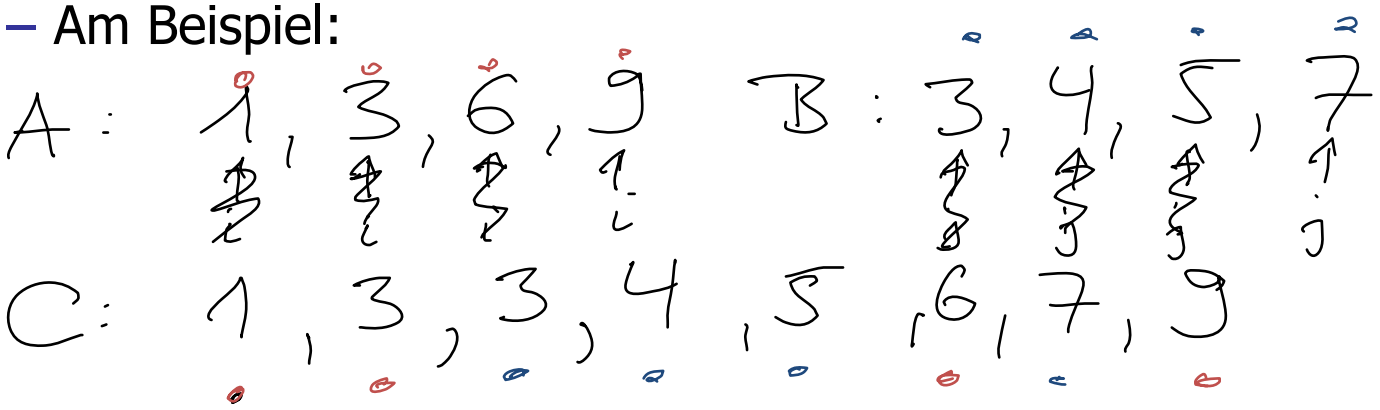
    $T(n) \leq T(\text{floor}(n/2)) + T(\text{ceil}(n/2)) + A \cdot n$ und $T(1) \leq A$
  - Und damit wie dort schon gezeigt $T(n) \leq 2A \cdot n \cdot \log n$

MergeSort — Beispiel



Mischen — Algorithmus

- Informaler Algorithmus
  - Gegeben zwei Folgen A und B, die schon sortiert sind
  - Die wollen wir zu einer sortierten Folge C mischen
  - Wir merken uns in jeder Folge eine Position, i für A und j für B, am Anfang sind beide am Anfang, also 0
  - Jetzt gehen wir immer in der Folge weiter nach rechts, wo das kleinere Element steht und schreiben es nach C
  - Am Beispiel:



MergeSort — Programm

UNI FREIBURG

- Das ist Ihre Übungsaufgabe
  - Erstmal eine Methode für Mischen
  - Und dann MergeSort
  - Nächste Woche Montag ist Allerheiligen, da gibt es keine Vorlesung und auch kein neues Übungsblatt
  - Deshalb haben Sie für dieses Übungsblatt zwei Wochen Zeit, bis zum 8. November um 16 Uhr
  - Tipp: fangen Sie nicht erst am 7. November damit an

Sortieren — Untere Schranke

UNI FREIBURG

- Geht es vielleicht noch schneller als $n \cdot \log n$?
  - Wenn man über die Elemente nicht mehr weiß, als dass man Sie miteinander vergleichen kann, dann nicht
    - informations-theoretische untere Schranke
    - machen wir vielleicht später im Semester noch
  - Wenn die Elemente ganze Zahlen sind, gibt es aber Tricks mit denen man in linearer Zeit sortieren kann
  - In der Praxis gehören aber Varianten von QuickSort trotzdem zu den schnellsten Sortierverfahren
  - Beispiel für so eine Variante: Splitter zufällig wählen
  - Mit Sortieren alleine kann man sich ein ganzes (Forscher)Leben beschäftigen ...

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 etwas praktischere 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: 1.3 MergeSort, 8. Quicksort
  - Wikipedia hat Artikel zu MinSort, QuickSort und MergeSort