universität freiburg

Algorithmen und Datenstrukturen SS 2025

Vorlesung 3: Untere Schranken für Sortieren

Dienstag, 6. Mai 2025

Prof. Dr. Hannah Bast
Professur für Algorithmen und Datenstrukturen
Institut für Informatik
Albert-Ludwigs-Universität Freiburg

Blick über die Vorlesung heute

- Organisatorisches
  - Erfahrungen mit dem **Ü2** — Laufzeitanalyse + die 5 Gruppen
  - Korrektur vom **Ü1** — Datei feedback-tutor.md
  - Vorgucker auf das **Ü3** — Noch mehr schöne Beweise 
  - Termin mit Ihrem Tutor — Buchbare Slots bei Bedarf
- Inhalt
  - Untere Schranken allgemein — Einfaches Beispiel
  - Was ist ein mathem. Beweis — Mathematisch ≠ viele Formeln
  - Vergleichsbasiertes Sortieren — Intuition, Definition, Beispiele
  - Untere Schranke — vergleichsbasiert $T(n) \geq n \cdot \log n$

Erfahrungen mit dem Ü2 1/2

- Auszüge aus Ihrem Feedback [halbwegs repräsentativ]

  "Das Ü2 war gut um wieder in das Beweisen reinzukommen"

  "Das Ü2 hat mir tatsächlich ziemlich viel Spaß gemacht"

  "Diese Vorlesung ist wirklich hervorragend gestaltet"

  "Stoff in der Vorlesung war richtig gut erklärt" [viele]

  "Aufzeichnungen von sehr guter Qualität und man kann top folgen"

  "Die majority Aufgabe war auch ganz spannend"

  "Bis jetzt immer gut ohne formelle Beweise durch die Prüfungen gekommen [...] ich finde den Ansatz immer extrem schwierig"

  "Habe wesentlich länger gebraucht als für das erste Blatt"

  "Ich habe noch nie mit LaTeX gearbeitet"

  "Ich möchte nicht in Schubladen denken" [drei mal]

Erfahrungen mit dem Ü2 2/2

- Statistik zu den fünf Gruppen (von Vorlesung 2, Folie 6)
  - Aus den 126 erfahrungen.txt für das Ü2 Stand 06.05.2025 12:30 Uhr

Gruppe	1 :	**3%**
Gruppe	1-2 :	**4%**
Gruppe	2 :	**29%**
Gruppe	2-3 :	**14%**
Gruppe	3 :	**35%**
Gruppe	3-4 :	**3%**
Gruppe	4 :	**2%**
Gruppe	5 :	**0%**
keine Angabe	:	**10%**

→ sogenannte "dunkle Materie"

Vielen Dank für die vielen (hoffentlich) ehrlichen Antworten

Tendenz: frühe Abgaben eher in Gruppe 1 o. 2, späte Abgaben eher in Gruppe 3 o. 4

Korrektur vom Ü1

- Feedback zu Ihrer Abgabe
  - Im SVN finden Sie im zum **Ü1** gehörigen Ordner eine Datei

    xy123/blatt-01/feedback-tutor.md
  - Machen Sie einfach **svn update** in Ihrer Arbeitskopie
  - Die Korrektur erfolgt in der Regel bis zum jeweiligen Freitag nach der Abgabe-Deadline
  - Falls Sie Wünsche oder Abneigungen in Bezug auf die Korrektur haben, teilen Sie dies Ihrem Tutor bitte mit

    Wenn Ihnen zum Beispiel ausführliches Feedback hilft, teilen Sie das mit

    Wenn Sie ausführliches Feedback eher nervt oder Sie es eh nicht lesen, ebenso

Vorgucker auf das Ü3

- Noch mehr schöne Beweise zum Üben und Lernen

  A1 Beweisen Sie, dass MinSort vergleichsbasiert ist

  Lassen Sie sich nicht von der langen Aufgabenstellung abschrecken, die soll Ihnen helfen. Sie sollen hier lernen, einen etwas komplexeren mathematisch Beweis maximal präzise zu führen. Nutzen Sie die Gelegenheit, Sie brauchen das sowohl für die Klausur als auch im weiteren Studium

  A2 Beweisen Sie, dass ZeroOneSort nicht vergleichsbasiert ist

  Die Aufgabe ist etwas einfacher, weil ein Gegenbeispiel reicht. Aber auch hier muss man genau verstanden haben, was es heißt, dass ein Sortieralgorithmus vergleichsbasiert ist. Ein ungefähres Verständnis reicht nicht

Persönliche Hilfestellung

Das ist wohlgemerkt
nur ein Angebot und in
keiner Weise verpflichtend

- Termine mit Ihrem Tutor
  - Es gibt in unseren Lehrveranstaltungen kein Präsenztutorat

    Wir haben damit schon sehr viel herumexperimentiert und unsere Erfahrung war immer wieder, dass nur wenige Studierende kommen (wenn man es nicht verpflichtend macht)

    Und gerade die, die Hilfe brauchen würden, kommen nicht oder nehmen nur passiv teil und stellen keine Fragen
  - Wir bieten allerdings Folgendes an (zusätzlich zu den Korrekturen)

    Ein Tool, mit dem sie niederschwellig (per Klick) einen Termin mit Ihrem Tutor buchen können, **Link auf Daphne**

    Das hilft hoffentlich denen, die es brauchen, einen Kontakt herzustellen (und wenn der Kontakt erstmal da ist, kann bei Bedarf auch ein zusätzlicher Termin vereinbart werden)

Untere Schranken allgemein 1/4

- Vorüberlegung
  - In der Vorlesung heute geht es um untere Schranken, die unabhängig von einem bestimmten Algorithmus gelten
  - Die Aussage ist dann: **egal mit welchem Algorithmus** man das Problem löst, man braucht $\geq$ xyz Operationen
  - Wie beweist man eine solche Aussage?

    Sieht schwierig aus, weil es eine Aussage über **alle möglichen** Algorithmen ist (ohne sie alle zu kennen)

Untere Schranken allgemein 2/4

- Ein einfaches Beispiel zum Warmwerden
  - Zu beweisen: Für das Sortieren von n Zahlen braucht man mindestens n Operationen (egal mit welchem Algorithmus)
  - Intuitives Argument:

    Damit ein Algorithmus korrekt ist, muss er sich zumindest jede Eingabezahl einmal anschauen

    Das alleine kostet schon n Operationen
  - Wichtig: das ist intuitiv, aber noch kein Beweis

    Auf der nächsten Folie, versuchen wir, dieses intuitive Argument formaler zu machen

    Generell ist in der Wissenschaft beides wichtig: Intuition, aber die muss man dann auch durch einen Beweis absichern

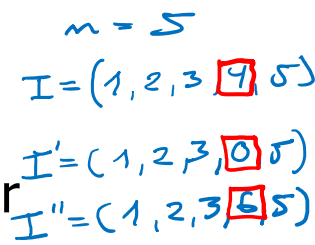
Untere Schranken allgemein 3/4

- Beweis: Sortieren von n Zahlen braucht $\geq n$ Op.

  1. Angenommen es gibt einen Algorithmus A, der weniger als n Operationen braucht, für ein $n \in \mathbf{N}$
  2. Für so ein n betrachten wir die Eingabe $I = (1, \ldots, n)$
  3. Da A für diese Eingabe weniger als n Operationen braucht, wird (mindestens) eine Eingabezahl überhaupt nicht gelesen
  4. Wir betrachten jetzt zwei Varianten der Eingabe I

     I' : ersetze die nicht gelesene Eingabezahl durch 0

     I'' : ersetze die nicht gelesene Eingabezahl durch $n + 1$
  5. A wird für I' und I'' (und I) dieselbe Ausgabe produzieren, die korrekten Ausgaben für I' und I'' sind aber verschieden
  6. Die Annahme ist also falsch, es gibt keinen solchen Alg. A



Untere Schranken allgemein 4/4

- Viele Probleme brauchen $\geq n$ Operationen
  - Aus demselben Grund, wie auf der vorherigen Folien:

    Man muss sich in der Regel mindestens jede Eingabezahl einmal anschauen, um das korrekte Ergebnis berechnen zu können
  - Es gibt aber auch Probleme, für die man zumindest für manche Eingaben der Größe n das korrekte Ergebnis mit echt weniger als n Operationen berechnen kann, z.B.

    Prüfe, ob die Eingabe sortiert ist [manchmal $< n$]

    Finde ein Element in einem gegebenen Feld [manchmal $< n$]

    Binäre Suche in einem sortierten Feld [immer $< n$, für große n]

    ...

Was ist ein mathematischer Beweis 1/2

- Mathematisch ≠ man muss alles in Formeln hinschreiben
  - Das ist ein häufiges Missverständnis!
  - Auf der vorvorherigen Folie steht viel Text, zum Beispiel:

    I' = wie I aber ersetze die nicht gelesene Eingabezahl durch 0
  - Wir hätten das auch formaler schreiben können:

    Eine Eingabe I der Größe n ist eine Folge von n Zahlen

    Für $j \in \{1, \ldots, n\}$ bezeichnen wir mit $I[j]$ die Zahl an Stelle j

    Sei $k \in \{1, \ldots, n\}$ eine Stelle, die nicht von A gelesen wird

    Definiere I' der Länge n mit $I'[j] := I[j]$ für $j \neq k$ und $I[k] = 0$

    Das ist formaler, aber weder genauer noch verständlicher
    (und es ginge auch noch formaler und unverständlicher)

Was ist ein mathematischer Beweis 2/2

- Allgemeine Struktur eines Beweises
  - Zentral wichtig ist, dass man eine Menge von Aussagen hat

    $A, B, C, E, D, F, G, \ldots$

    und eine Folge von Schlussfolgerungen

    $A \Rightarrow B,\ C \Rightarrow D,\ E \Rightarrow F,\ G \Rightarrow H,\ I \Rightarrow J,\ K \Rightarrow L, \ldots$

    und es gilt:

    1. Jede der Aussagen ist mathematisch klar
    2. Jedes "$\Rightarrow$" ist zweifelsfrei nachvollziehbar oder bekannt
    3. Jede Aussage auf der linken Seite der "$\Rightarrow$" ist entweder durch die Voraussetzungen gegeben oder bekannt oder steht auf der rechten Seite eines der vorherigen "$\Rightarrow$"
    4. Nach einem der "$\Rightarrow$" steht die zu beweisende Aussage

Vergleichsbasiertes Sortieren 1/8

- Vergleichsbasiertes Sortieren
  - **ZeroOneSort** und **CountingSort** laufen mit $\leq C \cdot n$ Operationen, sortieren die Elemente aber nicht durch "Umsortieren", sondern durch "Zählen"

    Das wollen wir für **ZeroOneSort** gerade zusammen coden
  - Wir wollen jetzt zeigen: wenn man "nur Umsortieren" zulässt, kann man nicht für alle Eingaben schneller sein als $C \cdot n \cdot \log n$ Operationen
  - Dazu müssen wir erst mal genauer fassen, was es heißt, dass ein Algorithmus "nur umsortiert"

    Es ist wichtig zu verstehen, was am Ende **genau** die Aussage ist: präzise Formulierung dazu auf Folie 16

Vergleichsbasiertes Sortieren 2/8

- Vorbetrachtung 1: Verzweigungen
  - In einem Programm können an diversen Stellen **Verzweigungen** auftreten, typischerweise so:

while (...): ...	Schleife fortführen oder beenden
for (...): ...	Schleife fortführen oder beenden
if (...): ... else: { ... }	If-Teil oder Else-Teil ausführen

  - Wir können alle Verzweigungen durch if (...): ... else: ... realisieren, zum Beispiel for i in range(n): ... durch

```
i = 0
while True:
    if i >= n: break
    ...
    i = i + 1
```

Wir schreiben **MinSort** gerade entsprechend um

(an der Laufzeit ändert das nur die Konstanten)

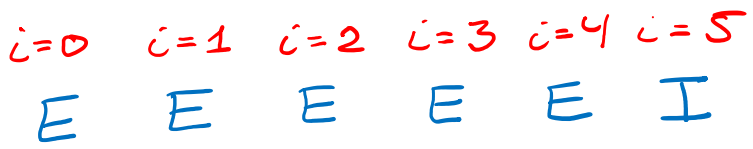
Vergleichsbasiertes Sortieren 3/8

- Vorbetrachtung 2: I/E-Sequenzen
  - Betrachten wir die Folge von Entscheidungen in den if (...) { ... } else { ... } Teilen im Ablauf eines Programms
  - Dann entspricht jeder Ablauf einer Sequenz **IEEIIIEIII**...

    I = **if**-Teil wird ausgeführt, **E** = **else**-Teil wird ausgeführt

    Falls der **if**-Teil nicht ausgeführt wird und es keinen **else**-Teil gibt, zählen wir das auch als **E**, zum Beispiel

```
i = 0
while True:
    if i >= 5: break
    ...
    i = i + 1
```



Vergleichsbasiertes Sortieren 4/8

- Vorbetrachtung 3: sortierende Permutationen
  - Eine Permutation einer Menge S ist eine bijektive Abbildung der Menge auf sich selber, zum Beispiel

    S = {A, B, C, D}

    f: S → S mit f(A) = C, f(B) = B, f(C) = D, f(D) = A

    A B C D
    C B D A
  - Eine sortieren Permutation für eine Eingabe $x_1, \ldots, x_n$ ist eine Permutation σ der Menge {1, …, n} die die Eingabe sortiert, das heißt $x_{\sigma(1)} \leq \ldots \leq x_{\sigma(n)}$, zum Beispiel:

    Eingabe: 16, 4, 23, 8, 42, 15 → 4, 8, 15, 16, 23, 42

    Sortierende Permutation σ von {1, 2, 3, 4, 5, 6}:

    σ(1) = 2, σ(2) = 4, σ(3) = 6, σ(4) = 1, σ(5) = 3, σ(6) = 5

    $\sigma^{-1}(1)=4$, $\sigma^{-1}(2)=1$, $\sigma^{-1}(3)=5$, $\sigma^{-1}(4)=2$, $\sigma^{-1}(5)=6$, $\sigma^{-1}(6)=3$

Vergleichsbasiertes Sortieren 5/8

- Definition von "vergleichsbasiert"
  - **Intuitiv:** Ein Algorithmus A heißt vergleichsbasiert, wenn die I/E-Sequenz eindeutig die sortierende Permutation bestimmt
  - **Formal:** Es gibt eine Funktion f, die jede mögliche I/E-Sequenz von A auf eine Permutation σ abbildet, so dass für jede Eingabe I gilt:

    Sei s die I/E-Sequenz, die sich ergibt, wenn man A auf I ausführt, dann ist $f(s)$ eine sortierende Permutation von I

    Das hört sich komplizierter an, als es ist; Beispiele und Erklärungen dazu auf den nächsten Folien

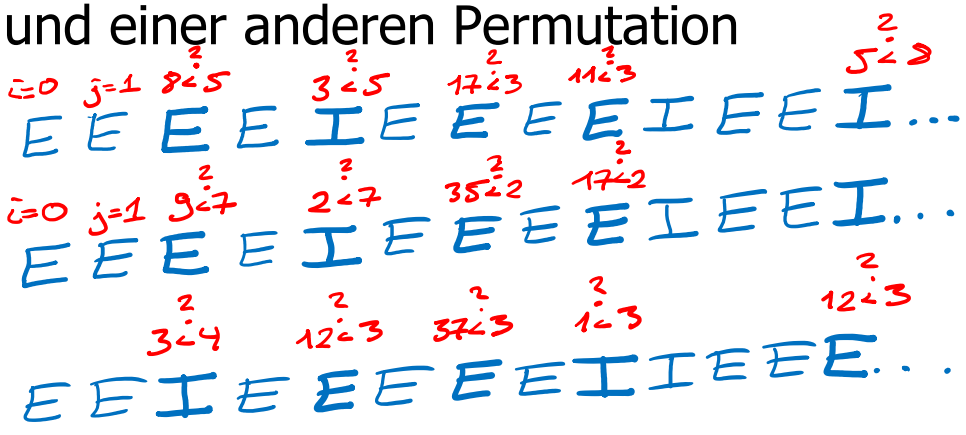
Vergleichsbasiertes Sortieren 6/8

- MinSort ist "vergleichsbasiert"
  - Beispiel: zwei Eingaben mit gleicher I/E-Sequenz und gleicher Permutation, und eine dritte Eingabe mit einer anderen I/E-Sequenz und einer anderen Permutation

σ_1^{-1} 2 3 1 5 4
5, 8, 3, 17, 11

σ_2^{-1} 2 3 1 5 4
7, 9, 2, 35, 17

σ_3^{-1} 3 2 4 5 1
4, 3, 12, 37, 1



Vergleichsbasiertes Sortieren 7/8

- MinSort ist "vergleichsbasiert"
  - Wie bestimmt die I/E-Sequenz die Permutation?
  - Das schauen wir uns an einem Beispiel an und für das Ü3 sollen Sie daraus einen mathematischen Beweis machen

    Beispieleingabe: 17, 32, 4, 11

    Wir schauen uns die I/E-Sequenz der ersten beiden Durchläufe der äußeren Schleife an und welche Permutation sie bedingen

D1: 17 32 4 11

D2: 4 32 17 11

„Transposition"

$\tau_1 = (1\ 3)$

i	1	2	3	4
$\sigma_1(i)$	3	2	1	4

...

Durchlauf 1: E E E E I E E I

Durchlauf 2: E E I E I I

Vergleichsbasiertes Sortieren 8/8

- ZeroOneSort ist nicht "vergleichsbasiert"
  - Beweis durch Gegenbeispiel:

    Zwei Eingaben mit der gleichen I/E-Sequenz, die aber **nicht** mit der gleichen Permutation sortiert werden können

    Das ist Aufgabe 2 vom Ü3

    Teil der Aufgabe ist es, das ZeroOneSort aus der Vorlesung so umzuschreiben, dass alle Verzweigungen explizit als if oder if-else implementiert sind

    Achtung: in dem "sum" steckt eine Schleife und damit auch eine Verzweigung, ebenso in dem Erzeugen des Ergebnisfelds am Ende

Untere Schranke Sortieren 1/4

- Beweis der unteren Schranke, Teil 1
  - Wir betrachten jetzt einen beliebigen vergleichsbasierten Algorithmus A auf Eingaben der Größe n
  - Sei T(n) eine **obere** Schranke für die Laufzeit, das heißt, A benötigt für Eingaben der Größe n immer $\leq T(n)$ Op.

    Wir wollen dann am Ende zeigen, dass $T(n) \geq C \cdot n \cdot \log n$
  - Für Eingabegröße n ist die Länge jeder I/E-Sequenz $\leq T(n)$, es gibt also höchstens $2^{T(n)}$ verschiedene I/E-Sequenzen

    Beachte: Wenn es eine Folge **IEEII** gibt, kann es nicht auch eine längere Folge geben, die mit **IEEII** anfängt
  - Der Algorithmus produziert also für alle möglichen Eingaben der Größe n höchstens $2^{T(n)}$ verschiedene Permutationen

Untere Schranke Sortieren 2/4

- Beweis der unteren Schranke, Teil 2
  - Ein korrekter Algorithmus muss für Eingabegröße n alle möglichen Permutationen produzieren können, das sind $n!$

    Wenn er eine Permutation nicht erzeugen könnte, würde er für die Eingabe, die **nur** mit dieser Permutation korrekt sortiert werden kann, nicht das richtige Ergebnis liefern
  - Auf der vorherigen Folie hatten wir gesehen, dass höchstens $2^{T(n)}$ verschiedene Permutationen produziert werden können
  - Wäre $2^{T(n)} < n!$, würden $< n!$ verschiedene Permutationen produziert werden können und der Alg. wäre nicht korrekt
  - Es muss also $2^{T(n)} \geq n!$ sein, oder äquivalent $T(n) \geq \log_2 (n!)$

Untere Schranke Sortieren 3/4

- Abschätzung von $\log_2(n!)$
  - Dafür wird oft die Stirling-Formel benutzt:

    $n! \geq \sqrt{2\cdot\pi\cdot n} \cdot (n/e)^n$

  - Wir können das aber auch (weniger genau, aber ausreichend) elementar-mathematisch abschätzen:

    $n! \geq (n/2)^{n/2}$

    $6! = \underbrace{6\cdot 5\cdot 4}_{\geq 3}\cdot 3\cdot 2\cdot 1 \geq 3^3$

    $5! = \underbrace{5\cdot 4}_{\geq 5/2 = 2.5}\cdot 3\cdot 2\cdot 1 \geq 2.5^{2.5}$

  - Daraus folgt:

    $\log_2(n!) \geq \log_2(n/2)^{n/2} = n/2 \cdot \underbrace{\log_2(n/2)}_{\log_2 n - 1} \geq \frac{1}{2}\cdot\log_2 n$ für $n \geq 4$

  - Das wiederum ist

    $\geq n/4 \cdot \log_2 n$ für $n \geq 4$

Untere Schranke Sortieren 4/4

- Beweis der unteren Schranke, Zusammenfassung

  Sei $T(n)$ eine **obere** Schranke für einen Algorithmus A, der n Elemente vergleichsbasiert sortiert

  Das heißt, A braucht für eine Eingabe der Größe n immer $\leq T(n)$ Operationen und die I/E-Sequenz bestimmt eindeutig eine Permutation σ, die die Eingabe korrekt sortiert

  Dann ist $T(n) \geq \log_2 (n!) \geq \frac{1}{4} \cdot n \cdot \log_2 n$ für $n \geq 4$

  Insbesondere heißt das: es gibt keinen vergleichsbasierten Algorithmus, der alle Eingaben der Größe n in Linearzeit, also mit $\leq C \cdot n$ Operationen korrekt sortieren kann

  Achtung: Das schließt nicht aus, dass ein vergleichsbasierter Algorithmus auf **manchen** Eingaben der Größe n schneller ist als $n \cdot \log n$

Literatur / Links

- Mehlhorn / Sanders
  - Laufzeit allgemein — 2.6 Basic Program Analysis
  - Untere Schranke — 5.3 A Lower Bound [for Sorting]
- Wikipedia
  - https://en.wikipedia.org/wiki/Comparison_sort