Vorlesungsmitschrift

Höhere Algorithmik

gelesen von Prof. Dr. Günter Rote

Tobias Höppner

Wintersemester 2014/2015

Inhaltsverzeichnis

1 Einführung (Vorlesung 1 am 17.10.) **1**
- 1.1 Organisatorisches . . . 1
- 1.2 Kuchen teilen . . . 1
  - 1.2.1 1. Algorithmus (für 2 Personen) . . . 1
  - 1.2.2 2. Algorithmus (für 3 Personen) . . . 1
  - 1.2.3 3. Teilen und Trimmen . . . 2
  - 1.2.4 4. Teilen mit bewegtem Messer . . . 2
  - 1.2.5 5. Simuliertes bewegtes Messer . . . 2
  - 1.2.6 6. Simuliertes Messer + Zufall . . . 2
  - 1.2.7 7. Divide & Conquer . . . 3
  - 1.2.8 8. Divde & Conquer + Zufall . . . 3

2 Einführung Teil 2 (Vorlesung 2 am 20.10.) **4**
- 2.1 Ziele der Vorlesung . . . 4
- 2.2 Rechnermodelle . . . 4
  - 2.2.1 Turing-Maschine . . . 4
  - 2.2.2 Registermaschine (RAM - random access machine) . . . 4
  - 2.2.3 Berechnung der Laufzeit . . . 5
- 2.3 Laufzeit eines Algorithmus . . . 6

3 Rechnermodelle (Fortsetzung) (Vorlesung 3 am 24.10.) **7**
- 3.1 Warum nicht die Turingmaschine? . . . 7
- 3.2 Elementare Operationen . . . 7
- 3.3 Teile und Herrsche . . . 8
  - 3.3.1 Beispiel A: Quicksort . . . 8
  - 3.3.2 Beispiel B: Mergesort (Sortieren durch Verschmelzen) . . . 8
  - 3.3.3 Analysemöglichkeiten . . . 8

4 Rekursion (Fortsetzung) (Vorlesung 4 am 31.10.) **10**
- 4.1 Motivation Master-Theorem . . . 10
- 4.2 Master-Theorem für divide and conquer-Rekursion . . . 10
  - 4.2.1 Bemerkungen . . . 11
- 4.3 Beweis: Master-Theorem . . . 11

5 Master Theorem (Fortsetzung) (Vorlesung 6 am 3.11.) **13**
- 5.1 Beweis Fortsetzung . . . 13
- 5.2 Zählen von Fehlständen (Inversion) . . . 14
  - 5.2.1 Divide and Conquer - oder - Warum Mergesort so wichtig ist! . . . 14
  - 5.2.2 Variante . . . 14
  - 5.2.3 Laufzeit . . . 15

6 Median (Vorlesung 7 am 7.11.) **16**
6.1 Bestimmung des k-kleinsten Elements (Medians) . . . 16
6.2 Quickselect . . . 16
6.2.1 Laufzeit . . . 16
6.3 randomisiertes Quickselect . . . 17
6.3.1 Laufzeit . . . 17
6.4 Quickselect nach Blum, Floyd, Pratt, Rivest, Tarjan (1973) . . . 17
6.4.1 Laufzeit . . . 18

7 Das Rucksackproblem (Vorlesung 8 am 10.11.) **20**
7.1 Lösung: Dynamisches Programmierung / Optimierung . . . 20
7.1.1 Laufzeit und Speicherbedarf . . . 21
7.2 Dynamische Programmierung . . . 21
7.3 Die Tabelle als Netzwerk . . . 22
7.4 Speicheroptimierung . . . 22

1 Einführung (Vorlesung 1 am 17.10.)

1.1 Organisatorisches

Mitschrift wird von Studenten erstellt.

Korrekturfarbe für Gummipunkte: Grün!

Voraussetzungen

- O-Notation
- Turing-Maschine
- Sortieralgorithmen
- Schubfachprinzip
- Gauß-Nummer
- Harmonische Reihe

1.2 Kuchen teilen

Problem: Ein Kuchen soll unter zwei Personen aufgeteilt werden.
Zwei Lösungsideen:
- perfektes Teilen
- einer teilt den Kuchen und der andere sucht sich eine Hälfte aus.

Was passiert, wenn jemand die Teile des Kuchens unterschiedlich bewertet? (z.B. Kirsche auf einer Seite, viel Sahne auf der anderen Seite)
Perfektes teilen bedeutet, dass jemand *für sich* perfekt teilt. (nach seinem Maßstab)
Ziel: Fairness Jeder will $\frac{1}{n}$ des Kuchens nach ihrem Maßstab. ($n = \#$Personen)

1.2.1 1. Algorithmus (für 2 Personen)

1. Erste teilt
2. Zweite sucht aus

Der Algorithmus ist toll, aber es gibt zu viele Schritte. Daher wollen wir den Algorithmus verbessern.
Ziel: möglichst wenige Schritte.

1.2.2 2. Algorithmus (für 3 Personen)

Anton, Berta und Clara:

1. Anton teilt $\frac{1}{3}|\frac{2}{3}$
2. Berta teilt $\frac{\frac{2}{3}}{2}|\frac{\frac{2}{3}}{2}$
3. Clara sucht aus.
4. Anton sucht aus.

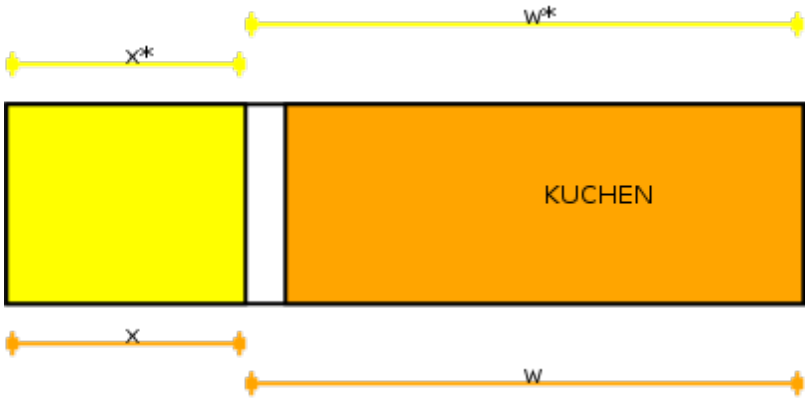
   Fall 1: Clara nimmt eines der rechten Stücke $\Rightarrow$ Anton nimmt linkes Stück.

Fall 2: Clara nimmt linkes Stück.
Schubfachprinzip: eines der rechten Stücke ist mindestens $\frac{1}{3}$

5. Berta):

1.2.3 3. Teilen und Trimmen

1. Anton teilt:



2. Berta:

Fall 1: Berta denkt $x \leq \frac{1}{3}$

Fall 2: Berta denkt $x > \frac{1}{3} \Rightarrow$ Trimmen

3. Clara darf sich entscheiden:

Fall 1: will x^* dann Algorithmus 1. für den Rest

Fall 2: will x^* nicht.
$\Rightarrow w^* \geq \frac{2}{3}$ für Clara und Anton

1.2.4 4. Teilen mit bewegtem Messer

Man nimmt ein Messer und jede Person sagt einfach Stop, wenn die *perfekte Wahl* für die Person getroffen ist.
#Schritte $= n - 1$

1.2.5 5. Simuliertes bewegtes Messer

- Jeder macht bei $\frac{1}{n}$ eine Markierung
- der/die Linkeste bekommt das Stück
  #Schritte $= n + (n-1) + ...3 + 2 = \theta(n^2)$ (Gauß-Nummer)

1.2.6 6. Simuliertes Messer + Zufall

Wie 5., aber

1. Reihenfolge zufällig
2. nur neue Linkeste Markierung werden gemacht

3. $T(n) = \#$erwartete Markierungen
   $= \underbrace{\frac{1}{n}}_{\text{Erwartete Anzahl der letzten Markierung}} + \underbrace{T(n-1)}_{\text{Erwartete Anzahl von Markierungen aller Anderen.}}$
4. $T(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \theta(\log n)$ (harmonische Reihe)
5. Gesamtlaufzeit $\leq n * O(\log n) = O(n * \log n)$

1.2.7 7. Divide & Conquer

n Personen
n Markierungen bei $\frac{k}{n}$
#Schritte im Worst Case $T(n) = n + 2$

1.2.8 8. Divde & Conquer + Zufall

(erwartete) Laufzeit pro Teilen $\theta(\log n)$ also insgesamt $\theta(n)$

2 Einführung Teil 2 (Vorlesung 2 am 20.10.)

2.1 Ziele der Vorlesung

- Algorithmen nach den wichtigsten Entwurfsprinzipien entwerfen:
  - Devide and Conquer
  - dynamisches Programmieren
  - bound and bound
  - greedy-Algorithmen
- Algorithmen mit Analysetechniken analysieren im Bezug auf Laufzeit und Speicherbedarf (Stromverbrauch)
  - randomisierte Analyse
  - amortisierte randomisierte Analyse
  - Rekursionsgleichungen
- Vergleich und Beurteilung von Algorithmen nach Einsatzzweck
- Theorie der NP Vollständigkeit verstehen und einfache Vollständigkeitsbeweise führen

(Stromverbrauch ist zunehmend wichtig, aber nicht Teil der Vorlesung. Allgemein sind Algorithmen mit weniger Laufzeit besser.)

2.2 Rechnermodelle

2.2.1 Turing-Maschine

Eine Turing-Maschine ist ein theoretisches Modell. Es handelt sich um ein unendliches Band mit Symbolen aus einem endlichen Alphabet mit endlichem Zustandsraum. In jedem Schritt wird ein Symbol gelesen, das Band entsprechend der Eingabe beschrieben und der Zustand verändert. Prinzipiell ist alles mit einer Turing-Maschine berechenbar, jedoch teilweise sehr umständlich, weil immer nur ein Symbol gelesen werden kann.

2.2.2 Registermaschine (RAM - random access machine)

RAM ist auch als random access memory als Arbeitsspeicher bekannt

Eine RAM funktioniert nach einem ähnlichen Prinzip wie moderne Rechner arbeiten. Es gibt eine potentiell unendliche (unbeschränkte) Anzahl von Registern `R0, R1, R2, ...` wobei jedes Register eine ganze Zahl enthalten kann. Die Programmiersprache ist ähnlich wie Assembler.

1. Befehle

Zuweisung `R4 = R17`

Rechenbefehl `R1 = R2 + R3`

```
R1 = R2 - R3
R1 = R2 * R3
R1 = R2 / R3
```

Operanden der Befehle

1. Register `R17`
2. direkte Operanden (Zahlen) `250`
3. indirekte Adressen: `(R1)`

den Inhalt des Registers, dessen Nummer in Register R1 steht.

2. Sprünge

```
GOTO x
IF R_i = 0 THEN GOTO x

GZ R1, label ;if R1 is greater 0, goto label
```

Es sind nur die drei Vergleichsoperationen GLZ: < 0 , GGZ: > 0 , GZ: = 0 erlaubt!

x ist eine Sprungmarke im Programm.

```
loop:
  \\ some commands
  GOTO loop
```

3. HALT

Ein Programm endet immer mit **HALT**

Ein- und Ausgabe

Eingabe: R0 = n= die Länge der Eingabe R1, R2, ... Rn. Alle andere Zellen sind auf 0 initialisiert.
Ausgabe steht am Ende im Speicher!

2.2.3 Berechnung der Laufzeit

a) Einheitskostenmaß (EKM)

Jede Operation dauert eine Zeiteinheit.
unfair, weil es Operationen gibt, die offensichtlich komplizierter sind.

b) logarithmisches Kostenmaß (LKM)

Laufzeit = Summe der Längen aller vorkommenden Adressen und Operanden.

$$l(x) = \lfloor \log_2 \max\{|x|, 1\} \rfloor + 1$$

$$\begin{aligned} \texttt{R2} &= (\texttt{R0}) + 250 \\ \text{...Kosten} &= l(2) + l(0) + \underbrace{l(\texttt{R0})}_{\text{Adresse}} + \underbrace{l((\texttt{R0}))}_{\text{Operand}} + \underbrace{l(250)}_{\text{Operanden}} \end{aligned}$$

Das LKM ist gerechter, als das EKM.

Im EKM kann man schwindeln:
Operationen auf langen Daten können in einem Schritt erledigt werden.
Andererseits ist das EKM näher an einem tatsächlichen Prozessor. Sofern die Operanden in ein Wort eines konventionellen Speichers (64 Bit) passen.
Abschätzung: LKM $\leq O$(EKM . l(längster vorkommender Operand oder Adresse))
Wenn die größten vorkommenden Zahlen nicht zu groß sind, dann ist das EKM realistisch.
LKM ist fairer, wenn es um sehr unterschiedliche Operanden geht (verschieden lang)

2.3 Laufzeit eines Algorithmus

Man muss den möglichen Eingaben eine Länge zuordnen.
x.. Eingabe $L(x)$
Bsp. n Zahlen $x_1, x_2, ..., x_n$ sortieren: $L = n$
Bsp. Multiplikation von langen Zahlen x, y: $L = \#$ Bits in der Eingabe.
Bsp. Lösen eines linearen Gleichungssystems: $Ax = bA \in \mathbb{Z}^{n \times x} b \in \mathbb{Z}^n x \in \mathbb{Q}^n$
Länge der Eingabe: n^2
Gauß-Elimination $O(n^3)$ Zeit, erfordert Rechnen mit rationalen Zahlen.
Man kann Zeigen, dass die Länge der Zähler und Nenner in den Zwischenergebnissen höchstens n-Mal ($\leq n$) ist, wenn man Brüche immer kürzt.
Laufzeit im LKM: $O(n^4, l(\text{größte Eingabezahl}))$

Tendenziell kompliziertes Beispiel, um zu illustrieren, dass LKM nicht immer leicht zu berechnen ist.

Was ist die Laufzeit eines Algorithmus?

$$T(x) = \text{Laufzeit des Algorithmus bei Eingabe} x$$

$$(Analyse im schlimmsten Fall). T(n) = \max\{T(x) | L(x) = n\}$$

Andere Möglichkeiten

Analyse im Durchschnitt, Erwartungswert der Laufzeit
Benötigt eine Wahrscheinlichkeitsverteilung auf der Menge der Eingaben.

3 Rechnermodelle (Fortsetzung) (Vorlesung 3 am 24.10.)

3.1 Warum nicht die Turingmaschine?

Die Registermaschine ist näher am heutigen Rechnermodell. Die Turingmaschine ist viel primitiver.
Satz:

a) Ein Alogrithmus, der auf einer Registermaschine Laufzeit $T(n)$ im logarithmischen Kosteneinheitsmaß hat, kann auf einer Turingmaschine in Laufzeit $O((T(n))^3)$ simuliuert werden.

b) Ein Alogirhtmus mit Laufzeit $U(n)$ auf einer Turingmaschine kann mit Laufzeit $O(U(n) \log U(n))$ auf einer Registermaschine im LKM simuliert werden.

zu b) In Zeit $U(n)$ kann die Maschine höchstens die Felder $-U(n) ... + U(n)$ beschreiben. Adressen sind durch $2U(n)$ beschränkt jeden Schritt der TM kann in konstant vielen Operationen der Registermaschine simuliert werden.
$\rightarrow O(\log U(n)$

zu a) Speicherinhalt auf dem Band notieren.
i : (Inhalt von Register i).($i+1$:Inhalt von Register($i+1$). ...
Register mit Inhalt 0 können weggelassen werden. Register werden in natürlicher Reihenfolge aufgeschrieben. Alle Zahlen binär oder dezimal (nach Belieben).

Die Länge des Bandes $= L$ ist durch $T(n)$ beschränkt.
Jede Adresse, jede Registereinheit wurde bei der letzten Benutzung in voller Länge bei $T(n)$ berücksichtigt.

3.2 Elementare Operationen

1. Adresse im Speicher suchen; (Adresse steht im linken Zwischenbereich)
2. entsprechenden Inhalt zwischen Speicher und Zwischenbereich übertragen
3. Rechenoperationen im Zwischenbereich

```
1 R2 = (R17)
```

Jede Stelle die verglichen wird, erfordert im schlimmsten Fall ein Wandern über das gesamte Band.

Operation 1 dauert $O(L^2)$ Schritte, wobei L die Länge des Bandes ist.

Operation 2 ist ähnlich. Gegebenenfalls muss man den rechten Teil des Bandinhalts verschieben (Um eine Stelle verschieben dauert $O(L)$ Zeit, $\leq O(L^2)$ insgesamt).

Operation 3 $\leq O(L^2)$

$O(L^2)$ für 1 Schritt der Registermaschine $= O(T(n))^2$

3.3 Teile und Herrsche

(eng. divide and conquer) (lat. divide et impera)

1. Zerlege das Problem P in Teilprobleme $P_1, P2, ..., P_k$ (typischerweise $k = 2$)
2. Löse die Teilprobleme rekursiv.
3. Füge die Teillösung zur Lösung von P zusammen.

3.3.1 Beispiel A: Quicksort

1. Wahl eines Pivotelementes a:
   Zerlegung in Elemente $\underbrace{< a}_{\text{Teilproblem}}, = a, > a$
3. Teilfolgen aneinanderhängen.

Bei Quicksort ist der erste Schritt aufwändiger, bei Mergesort der letzte Schritt.

3.3.2 Beispiel B: Mergesort (Sortieren durch Verschmelzen)

1. Zerlegung in 2 gleich große Teile
3. Verschmelzen der beiden sortierten Teillisten.

Laufzeit $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n)$

n gerade $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

Lösung $T(n) = O(n \log n)$

3.3.3 Analysemöglichkeiten

I. Lösung <u>erraten</u> und durch vollständige Induktion beweisen.

II. Wiederholtes einsetzen auf der rechten Seite:

$$
\begin{aligned}
T(n) &\leq 2T(\frac{n}{2}) + cn \quad (c > 0) \\
T(\frac{n}{2}) &\leq 2T(\frac{n}{4}) + c * \frac{n}{2} \\
T(n) &\leq 2(2T(\frac{n}{4}) + c\frac{n}{2}) + cn \\
&= 4 \underbrace{T(\frac{n}{4})}_{=2T(\frac{n}{8})+c\frac{n}{4}} + cn + cn \\
&\leq 8T(\frac{n}{8}) + cn + cn + cn \\
&\leq 2^k T(\frac{n}{2^k}) + k.c.n
\end{aligned}
$$

Annahme $n = 2^l$ ist eine Zweierpotenz $l = \log_2 n$

$$T(n) = \underbrace{2^l}_{n}\underbrace{T(1)}_{\text{konst.}} + \underbrace{l}_{\log_2 n}.c.n = O(n \log n)$$
$$= O(n) + O(n \log n)$$

nur gültig für Zweierpotenzen.

Möglichkeit a) n auf die nächste $n' = 2^l$ aufrunden.
$n \leq n' < 2n$
Sortieren von n Elementen kann nicht länger dauern als Sortieren von n' Elementen.
(zu beweisen! z.B. mit vollst. Indunktion anhand der Rekursion)
$T(n) \leq T(n') = O(n' \log n') = O(2n. \log(2n)) = O(n \log n)$✓

Möglichkeit b) Als Inspiration, um auf die Vermutung $O(n \log n)$ zu bekommen. Beweis mit Methode I.

III. Rekursionsbaum $\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \rfloor = \lfloor \frac{n}{4} \rfloor$ Laufzeit: 2^l Probleme konstanter Größe. $T(1), T(2) \leq c'$

$$\text{Ebene } 0 : \leq \Theta(n)$$
$$\text{Ebene } 1 : \leq 2\Theta(\lceil \frac{n}{2} \rceil)$$
$$\text{Ebene } 2 : \leq 4\Theta(\lceil \frac{n}{4} \rceil)$$
$$\Theta(n) \leq c.n$$
$$\text{Summe} \leq cn + 2c\lceil \frac{n}{2} \rceil + 4c\lceil \frac{n}{4} \rceil + ... + 2^{l-1}c\lceil \frac{n}{2^{l-1}} \rceil + 2^l c'$$
$$\leq cn + 2c(\frac{n}{2} + 1) + 4c(\frac{n}{4} + 1) + ...$$
$$= \underbrace{cn + cn + ... + cn}_{\text{l-mal}} + \underbrace{2c + 4c + 8c + ... + 2^{l-1}c}_{(2^l - 2)c} + 2^l c'$$

4 Rekursion (Fortsetzung) (Vorlesung 4 am 31.10.)

4.1 Motivation Master-Theorem

$$T(n) = \underbrace{T(\frac{n}{b})}_{T(\lfloor \frac{n}{b} \rfloor)+\cdots+T(\lceil \frac{n}{b} \rceil)} *a + f(n)$$

Für Probleme $\leq n_0$ wird das Problem irgendwie direkt gelöst.
Startbedingung: $1 \leq T(n) \leq M$ für $n \leq n_0$

In der Praxis muss man natürlich irgendwann das n_0 ausrechnen und kann nicht beliebig lange aufteilen.
Die Konstanten $a \geq 1$ und $b > 1$ müssen erfüllt sein und außerdem müssen wir fordern:

$$\begin{aligned} \lceil \frac{n}{b} \rceil &\leq n-1 \text{ für } n > n_0 \\ \Leftrightarrow \frac{n}{b} &\leq n-1 \\ n(1-\frac{1}{b}) &\geq 1 \\ n &\geq \frac{b}{b-1} \\ \Rightarrow n_0 &\geq \frac{b}{b-1} \end{aligned}$$

sonst werden die Probleme nicht kleiner und die Rekursion kann nicht gelöst werden.

$n \log_b n$ Elemente $\begin{cases} 1 \text{ Problem der Größe } n & \text{Aufwand } 1f(n)n^k \text{ Annahme } f(n) = n^k \\ 2 \text{ Probleme der Größe } \frac{n}{b} & \text{Aufwand } a * f(\frac{n}{b})a(\frac{n}{b})^k \\ 3 \text{ Probleme der Größe } \frac{n}{b^2} & \text{Aufwand } a^2 * f(\frac{n}{b^2})a^2(\frac{n}{b})^k \\ \vdots & \vdots \end{cases}$

Beispiel: Mergesort

$$\begin{aligned} a = b &= 2 \\ \gamma &= \log_2 2 = 1 \end{aligned}$$

4.2 Master-Theorem für divide and conquer-Rekursion

$$a \geq 1, b > 1, M, n_0 \geq 1 (\frac{n_0}{b} \leq n_0 - 1)$$

$f(n), T(n)$ Funktionen auf den natürlichen Zahlen

$$f(n) \geq 0$$

Es gelten die Rekursionsbedingungen

$$\begin{aligned} T(n) &\leq aT(\lceil \frac{n}{b} \rceil) + f(n) && (n > n_0) \\ T(n) &\geq aT(\lfloor \frac{n}{b} \rfloor) + f(n) && (n > n_0) \\ 1 &\leq T(n) \leq M \end{aligned}$$

Dann definieren wir den kritischen Exponenten

$$n = \log a > 0$$

(-) Wenn $f(n) = \mathcal{O}(n^{\gamma-\epsilon})$ für ein $\epsilon > 0$, dann
$T(n) = \Theta(n^{\gamma})$

(=) Wenn $f(n) = \Theta(n^{\gamma})$ ist, dann
$T(n) = \Theta(n^{\gamma} \log n)$

(+) Wenn $f(n) = \Theta(n^{\gamma+\epsilon})$ für ein $\epsilon > 0$ ist oder wenn die Reularitätsbedingung erfüllt ist
$\exists c <^1$:
(*) $a.f(\lceil \frac{n}{b} \rceil) < c.f(n)$ für alle $n > n_0$
dann gilt: $T(n) = \Theta(f(n))$

4.2.1 Bemerkungen

1. Wenn f monoton ist, dann gelten die Schlussfolgerungen auch für beliebig gemischtes Auf- und Abrunden.
2. Mit (*) kann man auch Funktionen wie $f(n) = 2^n$ oder $f(n) = 2^{\sqrt{n}}$ erfassen.
3. $\Omega(n^{\gamma+\epsilon})$ im Fall (+) reicht leider nicht.
4. $f(n) = n \log n, \gamma = 1$ wird nicht erfasst.

4.3 Beweis: Master-Theorem

a.) Wir betrachten die oberen Schranken für die Fälle (-) und (=)

(a) Ersetze $f(n)$ durch die oberen Schranke $\underline{u}.n^k$
$f(n) \leq u.n^k$ Finde eine Funktion $P(n)$ mit $(***)P(n) \geq aP(\lceil \frac{n}{b} \rceil) + un^k$ für $n \geq n_0$
und $P(n) \geq M$ für $n \geq n_0$
Dann ergibt sich durch vollständige Induktion: $T(n) \leq P(n)$
Basis: $(n \leq n_0)$

$$\begin{aligned} T(n) &\leq aT(\lceil \frac{n}{b} \rceil) + f(n) \leq \text{(I.V.)} \\ &\leq aP(\lceil \frac{n}{b} \rceil) + f(n) \\ &\leq aP(\lceil \frac{n}{b} \rceil) + un^k \leq P(n) \end{aligned}$$

(b) Verschiebung des Definitionsbereiches, um $\lceil \rceil$ los zu werden.
$v = \frac{b}{b-1} \Rightarrow -\frac{v}{b} = 1 - v$
$P(n) = T'(n - v)$ bzw. $T'(n) = P(n + v)$
$T' ist jetzt auf \mathbb{R}_{>0}$ definiert.
Wir bestimmten dann T' so, dass
$(**)T'(n) = aT'(\frac{n}{b}) + u'n^k$ (u ist eine Konstante)

Behauptung: aus $(**)$ folgt $(***)$, falls T' monoton wächst

$$\begin{aligned}
\underbrace{P(n)} &\geq aP(\lceil\frac{n}{b}\rceil) + un^k \\
\text{L.S.} &= P(n) = T'(n-v) = aT'(\frac{n}{b} - \frac{v}{b}) + u'(n-v)^k \\
\text{R.S.} &= aP(\lceil\frac{n}{b}\rceil) + un^k \\
&= aT'(\lceil\frac{n}{b}\rceil - v) + un^k \\
&< aT'(\frac{n}{b} + 1 - v) + un^k \\
&= aT'(\frac{n-v}{b}) + un^k
\end{aligned}$$

Jetzt müssen wir nur noch u' so wählen, dass $u'(n-v)^k \geq un^k$ für $n \geq n_0 u' \geq u\frac{n_0^k}{(n_0-v)^k}$

Lösen von $(**)$ durch Ansatz:

Fall (-) $k = \gamma - \epsilon : T'(n) = Dn^\gamma + En^k$ Einsetzen in $(**)$

$$\begin{aligned}
Dn^\gamma + En^k &= aD(\frac{n}{b})^\gamma + aE(\frac{n}{b})^k + u'n^k \\
&= Dn^\gamma \underbrace{\frac{a}{b^\gamma}}_{1} + n^k(aE\frac{1}{b^k} + u')
\end{aligned}$$

$$\begin{aligned}
E(1 - \frac{a}{b^k} = u', E = \frac{u'}{1 - \frac{a}{b^2}}) \\
E(1 - \frac{b^\gamma}{b^{\gamma-\epsilon}}) = u' \\
E(1 - b^\epsilon) = u' \\
\underline{E} = \frac{-u'}{b^\epsilon - 1} < 0
\end{aligned}$$

D ist noch frei: Wähle D groß genug, dass $P(n) = T'(n-v) = D(n-v)^\gamma + E(n-v)^k \geq M$ für $n \leq n_0$ ist.

Fall (=)

$$\begin{aligned}
T'(n) &= Dn^\gamma + En^\gamma \log_b n \\
\cdots &\Rightarrow E = u', \text{ D bleibt frei. - D groß genung.}
\end{aligned}$$

Ergebnis im Fall (-) $T(n) \leq D(n-v)^\gamma + E(n-v)^{\gamma-k} = \mathcal{O}(\backslash^\gamma)$

Ergebnis im Fall (=) $= \mathcal{O}(\backslash^\gamma \log \frac{\backslash}{\gamma})$

5 Master Theorem (Fortsetzung) (Vorlesung 6 am 3.11.)

5.1 Beweis Fortsetzung

Fall (+)

$$T(n) \leq T(\lceil \frac{n}{b} \rceil) + f(n)$$
$$T(n) \geq T(\lfloor \frac{n}{b} \rfloor) + f(n)$$
$$f(n) = \Theta(n^{\gamma+\epsilon})$$
$$\gamma = \log_b a$$
$$\text{oder: } \forall n > n_0 : \quad a.f(\lceil \frac{n}{b} \rceil) < c.f(n)$$

$c < 1$ ist eine Konstante

$$\Rightarrow T(n) = \Theta(f(n))$$

Beweis (Induktion)

untere Schranke $T(n) \geq f(n)$ (aus der Rekursion) $\Rightarrow T(n) = \Omega(f(n))$
obere Schranke: Ansatz: $T(n) \leq D.f(n)$
Versuch eines Beweises durch Induktion.
n_0 groß genug machen, dass $\frac{n_0}{b} \leq n_0 - 1 \Rightarrow \frac{n}{b} \leq n - 1 \forall n \geq n_0$
$\Rightarrow \lceil \frac{n}{b} \rceil < n$ Induktion kann funktionieren.
Induktionsschritt: $n \geq n_0$ für $i < n$ sei $T(i) \leq D.f(i)$ schon bewiesen.

$$\begin{aligned} T(n) &\leq aT(\lceil \frac{n}{b} \rceil) + f(n) \\ &\leq a.D.f(\lceil \frac{n}{b} \rceil) + f(a) \quad \text{nach I.V.} \\ &\leq D.cf(n) + f(n) \quad \text{Regularitätsbedingung} \\ &\leq D.f(n) \end{aligned}$$

$$\underbrace{Dc + 1 \leq D}_{\text{notwendig}}$$
$$\leftrightarrow D(1-c) \leftrightarrow D \geq \frac{1}{1-c}$$

Induktionsbasis: Wähle D groß genug, dass $T(i) \leq Df(i)$ für $i = 1, 2, ..., n_0 - 1$ gilt.
(Voraussetzung: $f(i) > 0$)
$D = \max\{\frac{T(1)}{f(1)}, \frac{T(2)}{f(2)}, \ldots, \frac{T(n_0)}{f(n_0)}, \frac{1}{1-c}\}$
2. Fall: $f(n) = \Theta(n\gamma + \epsilon), \epsilon > 0$

Obere Schranke (a) Ersetze $f(n)$ durch $u.n^{\gamma+\epsilon}$
Beweise, dass $f(n) = u.n^{\gamma+\epsilon}$ die Regularitätsbedingung erfüllt. (zunächst ohne Aufrunden, weil leichter).
$a.f(\frac{n}{b}) < c.f(n)$ L.S. $= a.u.(\frac{n}{b})^{\gamma+\epsilon} = \frac{a.un^{\gamma+\epsilon}}{b^{\gamma}.b^{\epsilon}}$
R.S. $= c.u.n^{\gamma+\epsilon}$

n_0 so groß wählen, dass $\frac{(\frac{n}{b}+1)^{\gamma+\epsilon}}{(\frac{n}{b})^{\gamma+\epsilon}}$ nahe genug bei 1 ist, sodass die L.S. immer noch $< cf(n)$ ist.

$\Leftarrow (1+\frac{b}{n_0})^{\gamma+\epsilon} < b^{\epsilon} \leftarrow n_0$ groß genug wählen.

5.2 Zählen von Fehlständen (Inversion)

Ein Fehlstand ist ein Paar $a_i > a_j$ mit $i > j$.
$(7,3,17,12,16,20) = (a_1,\ldots,a_n)$
$0 \leq$ #Fehlstände $\leq \binom{n}{2}$

5.2.1 Divide and Conquer - oder - Warum Mergesort so wichtig ist!

Fehlstände können zwischen linker und rechter Hälfte leicht bestimmt werden, wenn man die beiden sortierten Listen verschmelzt.
$(15610)(2479)$ Anzahl der Fehlstände = Anzahl der Fehlstände links + Anzahl der Fehlstände rechts
$F((a_1,\ldots,a_n)\ldots$ Ausgabe: Sortierte Liste$(b_1,\ldots,b_n), k$ wobei $k =$ #Fehlstände

```
if n=0:return (a_1),0
n'=\lfloor \frac{n}{2} \rfloor; n'' = n-n'
(b_1,...,b_n),F_L = F(a_1,...,a_n')
(c1,...,c_n''),F_R = F(a_{n'+1},...,a_n)
k = F_L + F_R
i = j = 1;
for l = 1,2,...,n
  if (b_i \leq c_j or j = n'' +1) and i \leq n'
    d_l = b_i; k = k + (j-1)
    i ++
  else
    d_l = c_j
    j++
  return (d_1,...,d_n),k
```

5.2.2 Variante

Länge des Fehlstands ist $j - i (a_i > a_j, j > i)$
$p =$ Gesamtlänge alle Fehlstände; wir brauchen zusätzlich zu jeden Element die Position in der ursprünglichen Liste.
Eingabe: $a_1,...,a_n...$ Ausgabe ist $(b_1,...,b_n),(q1,...,q_n),k,p$
q_i ist die Position von b_i in der Liste $(a_1,...,a_n)...$ (q_i) ist eine Permutation von $(1,...,n)$
Rekursive Aufrufe....
$(b_1,...,b_n),(q1,...,q'_n),F_L,P_L =$ rekursiv $(c_1,...,c_n),(r1,...,r''_n),F_R,P_R =$

```
i = j = 1
for l = 1,...,n
  if i <= n' and (j = n''+1 or b_i \leq c_j)
    d_l = b_i
    s_l = q_i
    k = k + j -1
    // eckige klammer rechts neben die oberen 3 ausdrücken
    p = p + (j-1) (n'-q_i)+T
    // ende
    i++
  else
    d_l = c_j
    s_l = r_j + n'
```

```
    // eckige Klammer rechts neben der beiden oberen ausdrücke:
    T = T + r_j
    // ende
    j ++
return (d_1,...,d_n)(s_1,...,s_n),k,p
```

5.2.3 Laufzeit

Nach Master-Theorem:

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(\underbrace{n}_{n^{\gamma}(=)})$$
$$a = 2, b = 2, \gamma = \log_2 2 = 1$$
$$T(n) = \Theta(n \log n)$$

Oft teilt das Problem auf, dass man Größen in zwei (ungefähre) gleich große Teile zerlegen möchte, einen Teil mit den kleineren Werten, und einen Teil mit den größeren Werten.
Der **Median** ($= \frac{n}{2}$) - größtes Element ist der ideale Trennungspunkt.

6 Median (Vorlesung 7 am 7.11.)

6.1 Bestimmung des k-kleinsten Elements (Medians)

Eingabe: $(a_1, a_2, \ldots, a_n)$ Liste mit Werten, k, $1 \leq k \leq n$
Bestimme das k-kleinste Element in sortierter Reihenfolge.
Sortierte Reihenfolge $a^{(1)} \leq a^{(2)} \leq \cdots \leq a^{(n)}$
Gesucht ist $a^{(k)}$; $k =$ Stelle in der sortierten Reihenfolge heißt der **Rang** des Elements
Beispiel: $(\underline{4}, 2, 1, 7, 9)$ Rang von $a_r 4$ ist 3.
Das Element, das in der Mitte steht heißt der Median.
Oft hat man versucht das Element in der Mitte zu bestimmen, in dem man alle Werte aufsummiert und dann durch die Anzahl der Werte teilt. Das Ergebnis sollte dann der Mittelwert sein. Das Problem sind allerdings Werte, die im Verhältnis zu allen anderen deutlich größer sind (bsp. $(1, 2, 3, 4, 2, 3, 9000)$), weil sie den Mittelwert ungünstig verschieben, sodass er keinen Sinn ergibt.

Der **Median** kann wie folgt bestimmt werden:
für ungerade n

$$a^{\frac{n+1}{2}}$$

für gerade n

$$\frac{1}{2}(a^{\frac{n}{2}} + a^{\frac{n}{2}+1})$$

6.2 Quickselect

Algorithmus: `Quickselect(k,l)` mit $l = (a_1, ..., a_n)$

1. Wähle Pivotelement a
2. Zähle, wie viele Elemente $<, =, > a$ sind. Der Rang von a ist zwischen $n_< + 1$ und $n_< + n_=$
3. **if** $k \leq n_<$ **then** `Quickselect(k,l_kleiner)`, wobei $|l_<| = n_<$ und `l_kleiner` enthält die Elemente $< a$.
4. **if** $k > n_< + n_=$ **then** `Quickselect(k-nkleiner-ngleich, l_groesser)`
5. return a

6.2.1 Laufzeit

Laufzeit im schlimmsten Fall: Pivotelement immer das kleinste Element oder größte.
Liste wird nur um 1 kleiner in jeder Rekursion $\rightarrow \Theta(n^2)$

Laufzeit im besten Fall:
- Rang(a)$= k$, keine Rekursion notwendig $\rightarrow \Theta(n)$ (GLÜCK!)
- Teilung in der Mitte: $n_<, n_> \leq \frac{n}{2}$: $T(n) = T(\frac{n}{2}) + \Theta(n)$ Ein bisschen Mastertheorem:

$$T(n) = 1 * T(\frac{n}{2}) + \Theta(n)$$
$$a = 1$$
$$b = 2$$
$$f(n) = \Theta(n^1) 1 > 0$$
$$\gamma = \log_b a = 0 \rightarrow \text{ Fall(+) } \Rightarrow T(n) = \Theta(f(n)) = \Theta(n)$$

Alternative (Einsetzen:)

$$T(n) = \Theta(n) + \Theta(\frac{n}{2}) + \Theta(\frac{n}{4}) \cdots = \Theta(n)$$

Der Algorithmus ist also stark davon abhängig welches Pivotelement wir wählen. Ideal wäre es den Median zu finden. Da wir aber hier versuchen den Median zu finden ist das ein Zirkelschluss. Dabei muss es nicht mal genau das Element genau in der Mitte sein, es reicht, wenn es nahe genug dran ist.

6.3 randomisiertes Quickselect

Wähle a zufällig aus der Liste.
Rang(a) ist gleich verteilt auf $1, 2, ..., n$.

6.3.1 Laufzeit

Analyse der erwarteten Laufzeit:
Wir nennen den Aufruf von Quickselect erfolgreich, wenn:

$$n_< + n_= = \frac{1}{4}n$$
$$n_> + n_= = \frac{1}{4}n$$

in der obersten Aufrufebene ist.

$$[\frac{1}{4}n \leq \text{rang}(a) \leq \frac{3}{4}n]$$

wenn (a) eindeutig ist.
Wahrscheinlichkeit(erfolgreich) $\geq \frac{1}{2}$
Bei einem erfolgreichen Aufruf wird die Liste auf höchstens $\frac{3}{4}n$ reduziert.
$T(n)$ =erwartete Laufzeit.

$$\begin{aligned} T(n) &\leq E(\#\text{Läufe bis zu einem erfoglreichen Lauf}).\mathcal{O}(n) + T(\frac{3}{4}) \\ &= \frac{1}{p} \text{ wobei } p = \frac{1}{2} \text{ die Erfolgswahrscheinlichkeit ist.} \\ &=\leq 2 \end{aligned}$$

$$T(n) \leq T(\frac{3}{4}n) + \mathcal{O}(n) \Rightarrow T(n) = \mathcal{O}(n)$$

6.4 Quickselect nach Blum, Floyd, Pratt, Rivest, Tarjan (1973)

Determinitische Auswahl in $\mathcal{O}(n)$ Zeit.

1. Falls $n \leq n_0$, sortiere

2. Andernfalls zerlege Folge in 5er-Gruppen und bestimme in jeder Gruppe den Median $m_1, m_2, ...m_{\lfloor \frac{n}{5} \rfloor}$

3. Bestimme den Median m^* dieser Mediane rekursiv.

4. Wähle das Pivotelement $a := m^*$ und verfahre weiter wie bei Quickselect.

6.4.1 Laufzeit

Welche Aussagen treffen jetzt auf $n_< + n_=$ und $n_> + n_=$zu?

$$
\begin{aligned}
n_< + n_= &\geq 3\frac{\lfloor \frac{n}{5} \rfloor}{2} \\
n_> + n_= &\geq 3\frac{\lfloor \frac{n}{5} \rfloor}{2} \\
n_< &= n - (n_< + n_=) \\
&= n - \frac{3}{2} * \lfloor \frac{n}{5} \rfloor
\end{aligned}
$$

Annahme $n = 5l$

$$
\begin{aligned}
n_< &\leq n - 0,3 = 0,7n \\
n &= 5l + i \qquad\qquad |i = 0,1,2,3,4 \quad l = \lfloor \frac{n}{5} \rfloor \\
n_< &\leq n - \frac{3}{2}l = n - \frac{3}{2}(\frac{n-i}{5}) \\
&= n - \frac{3}{10}n + \frac{3}{10}i \\
&\leq \frac{7}{10}n + \frac{12}{10} \leq \frac{7}{10}n + 3 \\
n_< &\leq \frac{7}{10}n + 3
\end{aligned}
$$

Behauptung: $T(n) = \mathcal{O}(n)$
Beweis: Annahme:

$$T(n) \leq \mathcal{C}n + T(\lfloor \frac{n}{5} \rfloor) + T(\lfloor 0,7n \rfloor + 3) \text{ für } n \geq 100$$

Behauptung $T(n) \leq \mathcal{C}'n$, wenn $\mathcal{C}' \geq 20C$ ist und $\mathcal{C}'$ so groß ist, dass $T(n) \leq \mathcal{C}'n$ für $n \geq 100$ ist.
Beweis mit vollständiger Induktion: $n \geq 100$ geht!

für $n > 100$:

$$\begin{aligned}
T(n) &\leq \mathcal{C}n + T(\lfloor \frac{n}{5} \rfloor) + T(\lfloor 0,7n \rfloor + 3) \\
&\leq \mathcal{C}n + \mathcal{C}'\frac{n}{5} + \mathcal{C}' * 0,7n + \mathcal{C}'3 \\
&\leq \mathcal{C}'\frac{n}{20} \\
&\leq \mathcal{C}'n(0,05 + 0,2 + 0,7) + \mathcal{C}'.3 \\
&= \mathcal{C}'(0,95n + 3) \leq \mathcal{C}'n \\
0,95n + 3 &\leq n \\
3 &\leq n.0,05 \quad (n \geq 100 \rightarrow n0,05 \geq 5)
\end{aligned}$$

7 Das Rucksackproblem (Vorlesung 8 am 10.11.)

Gegeben sind n Gegenstände.
Jeder Gegenstand hat einen Wert w und ein Gewicht g_i.
Es gibt eine Gewichtsschranke G.

Problem

Finde eine Teilmenge mit möglichst großem Wert und Gesamtgewicht $\leq G$

Beispiel n = 5, G = 12

i	1	2	3	4	5
g_i	4	3	5	2	6
w_i	7	8	6	3	9

Formulierung mit Variablen:

$$\text{maximiere} \quad \sum_{i=1}^{n} x_i w_i \qquad |x_i \text{ gibt an, ob Gegenstand ausgewählt wird}$$

$$\text{unter} \quad \sum_{i=1}^{n} x_i g_i \leq G$$

$$x_i \in \{0,1\}$$

$\rightarrow 2^n$ Möglichkeiten.

- ganzzahliges RP: $x_i \in \mathbb{N}$
- gebrochenes RP: $0 \leq x_i \leq 1$

7.1 Lösung: Dynamisches Programmierung / Optimierung

Löse das Problem durch *systematisches Lösen von Teilproblemen.*
Große Teilprobleme werden auf kleinere zurückgeführt, die man schon vorher gelöst hat.

Teilprobleme?

Betrachte nur die ersten i Gegenstände.
zusätzlich: muss man das zulässige Gesamtgewicht variieren.

$$f(i,b) = \text{optimaler Wert mit den ersten } i \text{ Gegenständen und das Gesamtgewicht } \leq b$$
$$= \max\{\sum_{j=1}^{i} w_j x_j | \sum_{j=1}^{i} g_j x_j \leq b, x_j \in \{0,1\}\}$$

$$f(i,b) = \max\{f(i-1,b), f(i-1,b-g_i) + w_i, \text{ falls } b \geq g_i\}$$
$$= f(i-1,b), \text{ falls } g_i > b$$

Lösung mit Tabelle: $f(i,b)$ mit $i = 0,...,n$ und $b = 0,...,G$

g_i		4	3	5	2	6
i	0	1	2	3	4	5
$b = 0$	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	8	8	8	8
4	0	7^+	8	8	8	8
5	0	7	8	8	11	11
6	0	7	8	8^-	11	
7	0	7	15^+	15^-	15	
8	0	7	15	15	15	
9	0	7	15	15	18	
10	0	7	15	15	18	
11	0	7	15	15	18	
12	0	7	15	21^+	21^-	21^-

Mit $^+$ markierte Einträge in der Tabelle werden zur optimalen Gesamtlösung hinzugefügt.
$x_5 = 0 \to x_4 = 0 \to x_3 = 1 \to x_2 = 1 \to x_1 = 1$ Tabelle liefert $f(5,12) = 21 = f(n,G)$ den Wert der Optimallösung.
Um die Lösung selbst zu finden, müssen wir zurückverfolgen, wie dieser Wert zustande gekommen ist.

Zurückverfolgen der Lösung

a) man merkt sich bloß die Tabelle und rechnet beim Zurückgehen jeden Eintrag noch einmal nach. (Programmieraufwand)

b) man speichert sich schon beim Berechnen Zusatzinformationen, wie der Wert zustande gekommen ist. (viel zusätzlicher Speicheraufwand)

7.1.1 Laufzeit und Speicherbedarf

$\Theta(nG)$ = Größe der Tabelle =Speicherbedarf
Der Speicher lässt sich auf $\Theta(G)$ reduzieren (allerdings verliert man die Möglichkeit der Rücknachvollziehbarkeit)

7.2 Dynamische Programmierung

- Definition der Teilprobleme
  nicht eindeutig vorgegeben.

- Rekursion (+ Anfangsbedingungen)
  Variante mit Gesamtgewicht $= b$
  ($f(i,b) = -\infty$ falls es keine Lösung gibt.)
  (Rekursion bleibt unverändert, Anfangsbedingung ändert sich. Optimallösung in der ganzen Spalte suche)

- systematisches Ausfüllen(Zeilen- oder Spaltenweise) der Tabelle aller Teilprobleme

- Rückverfolgen der Lösung

7.3 Die Tabelle als Netzwerk

Betrachte die Tabelle als gerichteten Graphen. Jeder Eintrag = 1 Knoten.
Vorgänger = Einträge, von denen der Knoten abhängt.
Kantengewicht = Wert, der in Rekursion addiert und Knotenwert = $f(i, b)$ = Längster Weg von der linken oberen Ecke $(0, 0)$ zum Knoten (i, b).
Sehr oft lässt sich eine DP-Rekursion als Wegeproblem in einem azyklischen Graphen modellieren. (kürzeste / längste Wege von einer Ecke zur anderen)

7.4 Speicheroptimierung

Der Speicher lässt sich optimieren(?) in dem man einen Faktor $\log n$ zur Laufzeit hinzufügt. (unklar...)