

Datenstrukturen¹

Sommersemester 2018

Herzlich willkommen!

¹basierend auf Folien von Prof. Georg Schnitger

Wer ist wer?

Wir: Professur für Algorithm Engineering www.ae.cs.uni-frankfurt.de

- Ulrich Meyer (Vorlesungen)
R 304 - RMS 11-15, umeyer AT ae.cs.uni-frankfurt.de
- Alex Schickedanz (Übungskoordination)
R 312 - RMS 11-15, alex AT ae.cs.uni-frankfurt.de
- Ehud Cesresnyes, Jaro Eichler, Stephan Gardoll, Adrian Kretz,
Vincent Kühn, Lukas Maurer, Marco Schmalhofer, Andreas Scholl,
Anh Duong Vo (Tutoren)
- Emails zur Vorlesung an [ds18 AT ae.cs.uni-frankfurt.de](mailto:ds18@ae.cs.uni-frankfurt.de)

Wer sind Sie?



Interaktives Clickersystem



- rund 90 Clicker verfügbar
- am Anfang jeder VL mitnehmen (2 Taschen auf der ersten Bank)
- am Ende jeder VL zurücklegen !!!
- 10 verschiedene Antwortmöglichkeiten
- Multiple-Choice Setting als Default
- „Channel“ und „?“ nicht drücken
- LED grün: Antwort registriert
- LED gelb mehrfach blinkend: senden
- LED gelb einmal blinkend: keine Umfrage
- LED rot: Empfangsproblem



Noch ein wichtiger Hinweis ...

Die Clicker sind *nicht* zum Verzehr geeignet.





Wer sind Sie? (1) demographic

Ich studiere:

- (1) Informatik Bachelor
- (2) Bioinformatik
- (3) Wirtschaftsinformatik
- (4) Informatik als Nebenfach
- (5) Informatik Lehramt
- (6) Sonstiges



Wer sind Sie? (2) demographic

Ich studiere **Informatik Bachelor** im

- (i) i-ten Fachsemester



Wer sind Sie? (3) demographic

Ich besuche diese Vorlesung (TheolInf 1) zum

- (i) i-ten Mal

Abstrakte Datentypen und Datenstrukturen

Ein **abstrakter Datentyp** besteht aus einer Sammlung von Operationen auf einer Menge von Objekten. Eine **Datenstruktur** ist eine Implementierung eines abstrakten Datentyps.

- Warum abstrakte Datentypen? In den Anwendungen treten dieselben Operationen für **verschiedenste** Datenmengen auf: Gib **eine** möglichst effiziente Implementierung an!
- Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
 - ▶ Wenn der jüngste Schlüssel zu entfernen ist: Wähle die Datenstruktur „**Stack**“.
 - ▶ Wenn der älteste Schlüssel zu entfernen ist: Wähle die Datenstruktur „**Schlange**“.
 - ▶ Wenn Schlüssel mit zugeordneter Prioritäten eingefügt werden und der Schlüssel mit jeweils höchster Priorität zu entfernen ist: Benutze einen **Heap**.

Worum geht's?

Entwerfe eine **möglichst effiziente** Datenstruktur für einen gegebenen abstrakten Datentyp.

- Was heißt Effizienz? Minimiere
 - ▶ den **Speicherplatzverbrauch** und
 - ▶ die **Laufzeiten** der einzelnen Operationen.
- Wie skalieren die Implementierungen mit wachsender Eingabelänge?
 - ▶ Führe eine asymptotische Analyse von Laufzeit und Speicherplatzverbrauch durch.

Ohne effiziente Datenstrukturen stehen bei der Verarbeitung großer Datenmengen in Datenbanken, Suchmaschinen, Peer-to-peer Netzwerken etc. alle Räder still !!!

Worauf wird aufgebaut?

- (1) Grundlagen der Programmierung 1: Kenntnis der elementaren Datenstrukturen.
- (2) Lineare Algebra und Analysis:
 - Logarithmen,
 - Grenzwerte und
 - asymptotische Notation.
- (3) Diskrete Modellierung: Mathematische Induktion und allgemeine Beweismethoden.

- Kapitel 3-5, 10-12 und 18 in T.H. Cormen, C.E. Leiserson, R.L. Rivest und C. Stein, “[Introduction to Algorithms](#)”, second edition, MIT Press, 2001.
- M. T. Goodrich, R. Tamassia und D. M. Mount, [Data Structures and Algorithms in C++](#), John Wiley, 2003.
- Kapitel 1-3 in J. Kleinberg und E. Tardos, “[Algorithm Design](#)”, Addison-Wesley, 2005.
- Ein Skript zur Vorlesung wie auch die Beamer-Dateien können von der Webseite <http://www.ae.cs.uni-frankfurt.de> → Teaching heruntergeladen werden
(ab 10.4.18. User: ds18 Passwd: ds18ds18).

Übungsbetrieb

- Webseite <http://www.ae.cs.uni-frankfurt.de> → Teaching erklärt Details wie die Notengebung, Anmeldung zu den regulären Übungsgruppen via AUGE (noch bis Freitag 13.04.) etc.
- Heute (10.04.) geben wir online das **0. Übungsblatt** heraus (ohne Abgabe).
- Besprechung ab nächster Woche, freie Wahl der Übungsgruppe falls noch keiner Gruppe zugeordnet.
- Regulärer Übungsbetrieb für 1., 2., ... Blatt Ende April gemäß Zuordnung.
- Übungen finden im 2-Wochen Rhythmus statt: Jeder Übungszettel wird in der Vorlesung ausgegeben und ist vor der Vorlesung nach zwei-wöchiger Bearbeitungszeit zurückzugeben.
- Bitte **UNBEDINGT** am Übungsbetrieb teilnehmen und Aufgaben bearbeiten!

- Die Erstklausur wird am 03.08., die Zweitklausur am 11.10. stattfinden.
- Die in den Übungen erreichten Punkte werden mit einem Maximalgewicht von 10% zu den Klausurpunkten hinzugezählt:

Wenn in der Klausur $x\%$ und in den Übungen $y\%$ erzielt wurden, dann wird $z = x + y/10$ als Gesamtpunktzahl angerechnet.

Die Note hängt nur von der Gesamtpunktzahl z ab. Die Veranstaltung ist bestanden, wenn $z \geq 50$.

- **Freiwillige Zusatzveranstaltung: Datenstrukturen in C++ Programmieren**
- C/C++ ist zentral für High-Performance-Computing da sehr maschinennah. Sehr gut für Datenstrukturen geeignet.
- Dozent: Manuel Penschuck (manuel AT ae.cs.uni-frankfurt.de)
- ab 20.04. Freitags 10:00 - 12:00, Magnus Hörsaal
(Robert-Mayer-Str. 11-15)
- Nicht klausurrelevant, aber durch Mitarbeit (Codeabgabe) können bis zu 30% fehlende Übungspunkte kompensiert werden.
- Bei knappem Zeitbudget lieber DS-Übungen priorisieren !!!

- Vor- und Nachbereitung der Vorlesungen & Übungen
- Teamarbeit vs. Einzelkämpfer
- Feedback an uns, Fragen stellen, RF Response Cards
- Studieren lernen – der richtige Umgang mit den Freiheiten
- Auslandssemester:

Infoveranstaltung des International Office am Campus Riedberg (speziell für NaturwissenschaftlerInnen):

Am Anfang der Vorlesungszeit, Termin wird noch bekannt gegeben.

Infos aus 2017: <http://www.uni-frankfurt.de/59064258/VortragStudPraktAuslandRiedberg-Stand-Okt17.pdf>



Wiederholung? (1) demographic

Mathematische Grundlagen:

Was genau ist die Lösung von $\sum_{i=1}^n i$???

- (1) häh ?
- (2) wusste ich mal, aber habe es vergessen ...
- (3) $\frac{n \cdot (n+1)}{2}$
- (4) $\frac{n \cdot (n-1)}{2}$
- (5) $\frac{n^2 - 1}{2}$

Auflösung: : (3) $\frac{n \cdot (n+1)}{2}$ (wie beweist man das?)

Die Summe der ersten n Zahlen

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2}.$$

- Mathematische Induktion nach n :

- ▶ Induktionsbasis für $n = 1$: $\sum_{i=1}^1 i = 1$ und $\frac{1 \cdot (1+1)}{2} = 1$. Stimmt!
- ▶ Induktionsschritt von n auf $n+1$:
 - ★ Wir können annehmen, dass $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$ gilt.
 - ★ $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{n \cdot (n+1) + 2 \cdot (n+1)}{2} = \frac{(n+2) \cdot (n+1)}{2} = \frac{(n+1) \cdot (n+2)}{2}$ und das war zu zeigen.

- Ein direktes Argument:

- ▶ Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
- ▶ Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
- ▶ Die Hauptdiagonale besitzt n Gitterpunkte und unterhalb der Hauptdiagonale befindet sich die Hälfte der verbleibenden $n^2 - n$ Gitterpunkte. Also folgt $\sum_{i=1}^n i = n + \frac{n^2 - n}{2} = \frac{n \cdot (n+1)}{2}$.



Wiederholung? (2) demographic

Mathematische Grundlagen:

Lösung für die geometrische Reihe $\sum_{i=0}^n a^i$ mit $a \neq 1$???

- (1) $2 \cdot a^n$
- (2) $\frac{a^{n+1}-1}{a-1}$
- (3) $\frac{a^{n+2}-1}{n}$
- (4) $\frac{a \cdot n \cdot (n-1)}{2}$
- (5) keine Ahnung

Auflösung: : (2) $\frac{a^{n+1}-1}{a-1}$ (wie beweist man das?)

Die geometrische Reihe

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ falls } a \neq 1.$$

- Mathematische Induktion nach n :

- ▶ Induktionsbasis für $n = 0$: $\sum_{i=0}^0 a^i = 1$ und $\frac{a^{0+1} - 1}{a - 1} = 1$. Stimmt!
- ▶ Induktionsschritt von n auf $n + 1$:
 - ★ Wir können annehmen, dass $\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1}$ gilt. Dann ist
 - ★ $\sum_{i=0}^{n+1} a^i = \sum_{i=0}^n a^i + a^{n+1} = \frac{a^{n+1} - 1}{a - 1} + a^{n+1} = \frac{a^{n+1} - 1 + a^{n+2} - a^{n+1}}{a - 1} = \frac{a^{n+2} - 1}{a - 1}$ und das war zu zeigen.

- Ein direktes Argument:

$$\begin{aligned}(a - 1) \cdot \sum_{i=0}^n a^i &= a \cdot \sum_{i=0}^n a^i - \sum_{i=0}^n a^i \\&= \sum_{i=1}^{n+1} a^i - \sum_{i=0}^n a^i = a^{n+1} - a^0 = a^{n+1} - 1\end{aligned}$$

und das war zu zeigen.

Rechnen mit Logarithmen

Seien $a > 1$ und $x > 0$ reelle Zahlen. $\log_a(x)$ ist der Logarithmus von x zur Basis a und stimmt mit y genau dann überein, wenn $a^y = x$ gilt.

(a) $\log_a(x \cdot y) = \log_a x + \log_a y$.

(b) $\log_a(x^y) = y \cdot \log_a(x)$.

(c) $a^{\log_a x} = x$.

(d) $\log_a x = (\log_a b) \cdot (\log_b x)$.

Was ist $4^{\log_2 n}$?

- $\log_2 n = (\log_2 4) \cdot (\log_4 n)$ mit (d).

- $4^{\log_2 n} = 4^{(\log_2 4) \cdot (\log_4 n)} = (4^{\log_4 n})^{\log_2 4} = n^2$.

Was ist $b^{\log_a x}$?

- $\log_a x = (\log_a b) \cdot (\log_b x)$ mit (d).

- $b^{\log_a x} = b^{(\log_a b) \cdot (\log_b x)} = (b^{\log_b x})^{\log_a b} = x^{\log_a b}$.

Binomialkoeffizienten

$$\binom{n}{k} = \begin{cases} \frac{n \cdot (n-1) \cdots (n-k+1)}{1 \cdot 2 \cdots k} = \frac{n!}{k! \cdot (n-k)!} & \text{falls } 1 \leq k \leq n-1, \\ 1 & \text{falls } k=0 \text{ oder } k=n. \end{cases}$$

- 1 Sei S eine Menge von n Elementen. Dann hat S genau $\binom{n}{k}$ Teilmengen der Größe k .
- 2 Binomischer Lehrsatz: Es gilt $(a+b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k}$. Insb. ist $\sum_{i=0}^n \binom{n}{i} = 2^n$: eine n -elementige Menge hat 2^n Teilmengen.
- 3 Sei $\text{Perm}(S)$ die Menge aller Permutationen von S . Dann ist $|\text{Perm}(S)| = n!$

Wieviele k -elementige Teilmengen hat die Menge $\{1, \dots, n\}$?

Wir führen eine Induktion nach n .

- Basis für $n = 1$:
 - ▶ $\binom{1}{0} = \binom{1}{1} = 1$.
 - ▶ Es gibt genau eine Teilmenge von $\{1\}$ mit keinem Element sowie genau eine Teilmenge mit einem Element.
- Der Induktionsschritt: Wir wissen, dass $\{1, \dots, n\}$ genau $\binom{n}{k}$ Teilmengen mit genau k Elementen hat.
 - ▶ Wieviele k -elementigen Teilmengen von $\{1, \dots, n, n+1\}$ besitzen das Element $n+1$ nicht?
Nach Induktionsannahme genau $\binom{n}{k}$ Teilmengen.
 - ▶ Wieviele k -elementigen Teilmengen von $\{1, \dots, n, n+1\}$ besitzen das Element $n+1$?
Nach Induktionsannahme genau $\binom{n}{k-1}$ Teilmengen.
 - ▶ Es ist $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$ und das war zu zeigen.

Für welchen Rechner sollen wir die Laufzeit berechnen?

- Analysiere die Laufzeit auf einem abstrakten Rechner:
 - ▶ Der Speicher besteht aus Registern, die eine ganze Zahl speichern können.
 - ▶ Eine CPU führt einfache boolesche und arithmetische Operationen aus.
 - ▶ Daten werden zwischen CPU und Speicher durch (indirekte) Lade- und Speicherbefehle ausgetauscht.
- Damit ist die Laufzeitberechnung für jeden modernen sequentiellen Rechner gültig, aber wir erhalten für einen speziellen Rechner nur eine **bis auf einen konstanten Faktor** exakte Schätzung.

Laufzeitmessung für welche Programmiersprache und welchen Compiler?

- Wir „sollten“ mit einer Assemblersprache arbeiten: Wir sind vom Compiler unabhängig und haben es mit einer geringen Anzahl verschiedener Anweisungen zu tun.
- Aber die Programmierung ist viel zu umständlich und wir wählen deshalb C++ bzw. Pseudocode.
- Wenn wir die Anzahl ausgeführter C++ Befehle zählen, erhalten wir dehalb nur eine **Schätzung** der tatsächlichen Anzahl ausgeführter Assembler Anweisungen.
- Unsere Schätzung ist deshalb nur **bis auf einen konstanten Faktor** exakt.

Laufzeitmessung: Skalierung unter wachsender Eingabelänge

Wie verhält sich die Datenstruktur oder der Algorithmus unter wachsender Eingabelänge?

- Eine Laufzeitbestimmung für alle Eingaben ist schwierig.
- Betrachte stattdessen die **worst-case** Laufzeit eines Algorithmus oder einer Datenstruktur **für jede Eingabelänge**.
 - ▶ Unsere Laufzeitbestimmung ist letztlich nur eine, bis auf einen konstanten Faktor exakte Schätzung.
 - ▶ Wir interessieren uns vor allen Dingen für die Laufzeit „großer“ Eingaben und „unterschlagen“ konstante Faktoren.

Wir erhalten eine von der jeweiligen Rechnerumgebung unabhängige Laufzeitanalyse, die das **Wachstumverhalten** der tatsächlichen Laufzeit verlässlich voraussagt.

Ein Beispiel: Das Teilfolgenproblem

- Die Eingabe besteht aus n Zahlen a_1, \dots, a_n (u. U. negativ).
- Definiere $f(i, j) = a_i + a_{i+1} + \dots + a_j$ für $1 \leq i \leq j \leq n$.
- Das Ziel ist die Berechnung von $\max\{f(i, j) | 1 \leq i \leq j \leq n\}$, also die maximale Teilfolgensumme.

Wir betrachten nur Algorithmen A , die ausschließlich Additionen und Vergleichsoperationen auf den Daten ausführen.

- $\text{Additionen}_A(n) = \max_{a_1, \dots, a_n \in \mathbb{Z}} \{ \text{Anzahl Additionen von } A \text{ für Eingabe } (a_1, \dots, a_n) \}$
- $\text{Vergleiche}_A(n) = \max_{a_1, \dots, a_n \in \mathbb{Z}} \{ \text{Anzahl Vergleiche von } A \text{ für Eingabe } (a_1, \dots, a_n) \}$.
- $\text{Zeit}_A(n) = \text{Additionen}_A(n) + \text{Vergleiche}_A(n)$ ist (obere Schranke für) die worst-case Rechenzeit von Algorithmus A .

Das Teilstufenproblem: Algorithmus A_1

```
Max = -∞;  
for (i=1 ; i <= n; i++)  
    for (j=i ; j <= n; j++)  
        {Berechne  $f(i,j)$  mit  $j - i$  Additionen;  
         Max = max{ $f(i,j)$ , Max}; }
```

Wir benötigen $j - i$ Additionen zur Berechnung von $f(i,j)$. Also ist

$$\text{Additionen}_{A_1}(n) = \sum_{i=1}^n \sum_{j=i}^n (j - i).$$

- Eine obere Schranke: $\text{Additionen}_{A_1}(n) \leq \sum_{i=1}^n \sum_{j=1}^n n = n^3$.
- Eine untere Schranke: $\text{Additionen}_{A_1}(n) \geq \sum_{i=1}^{n/4} \sum_{j=3n/4+1}^n \frac{n}{2}$ (warum???) und $\text{Additionen}_{A_1}(n) \geq \frac{n}{4} \cdot \frac{n}{4} \cdot \frac{n}{2} = \frac{n^3}{32}$ folgt.

Die worst-case Laufzeit von A_1

- Wieviele Vergleiche werden für die n Zahlen a_1, \dots, a_n ausgeführt?

$$\begin{aligned}\text{Vergleiche}_{A_1}(n) &= \sum_{i=1}^n \sum_{j=i}^n 1 = \sum_{i=1}^n (n - i + 1) \\ &= n + (n - 1) + \dots + 2 + 1 \\ &= \sum_{i=1}^n i = \frac{n \cdot (n + 1)}{2}.\end{aligned}$$

- Und die worst-case Laufzeit?
 - ▶ $\text{Zeit}_{A_1}(n) = \text{Additionen}_{A_1}(n) + \text{Vergleiche}_{A_1}(n)$.
 - ▶ $\frac{n^3}{32} \leq \text{Additionen}_{A_1}(n) \leq n^3$. Also ist
 $\frac{n^3}{32} + \frac{n \cdot (n + 1)}{2} \leq \text{Zeit}_{A_1}(n) \leq n^3 + \frac{n \cdot (n + 1)}{2}$.
- Wir möchten das wesentliche Ergebnis festhalten, nämlich, dass die Laufzeit **kubisch** ist.

Die asymptotische Notation (WICHTIG!!!)

$f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ seien Funktionen, die einer Eingabelänge $n \in \mathbb{N}$ eine nicht-negative Laufzeit $f(n)$, bzw. $g(n)$ zuweisen.

- Die Groß-Oh Notation: $f = O(g) \Leftrightarrow$ Es gibt eine positive Konstante $c > 0$ und eine natürliche Zahl $n_0 \in \mathbb{N}$, so dass

$$f(n) \leq c \cdot g(n)$$

für alle $n \geq n_0$ gilt: f wächst höchstens so schnell wie g .

- $f = \Omega(g) \Leftrightarrow g = O(f)$: f wächst mindestens so schnell wie g .
- $f = \Theta(g) \Leftrightarrow f = O(g)$ und $g = O(f)$: f und g wachsen gleich schnell.
- Die Klein-Oh Notation: $f = o(g) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$: f wächst langsamer als g .
- $f = \omega(g) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$: f wächst schneller als g .

Die Laufzeit von A_1 ist kubisch

Wir müssen $\text{Zeit}_{A_1}(n) = O(n^3)$ und $n^3 = O(\text{Zeit}_{A_1}(n))$ zeigen.

- Warum gilt $\text{Zeit}_{A_1}(n) = O(n^3)$?

- ▶ Wir wissen $\text{Zeit}_{A_1}(n) \leq n^3 + \frac{n \cdot (n+1)}{2}$.
- ▶ Also ist $\text{Zeit}_{A_1}(n) \leq n^3 + \frac{n \cdot 2n}{2} \leq n^3 + \frac{2n^3}{2} = 2 \cdot n^3$.
- ▶ $\text{Zeit}_{A_1}(n) = O(n^3)$: Setze $c = 2$ und $n_0 = 0$.

- Warum gilt $n^3 = O(\text{Zeit}_{A_1}(n))$?

- ▶ Wir wissen $\frac{n^3}{32} + \frac{n \cdot (n+1)}{2} \leq \text{Zeit}_{A_1}(n)$.
- ▶ Also ist $\frac{n^3}{32} \leq \text{Zeit}_{A_1}(n)$ und deshalb folgt $n^3 \leq 32 \cdot \text{Zeit}_{A_1}(n)$.
- ▶ $n^3 = O(\text{Zeit}_{A_1}(n))$: Setze $c = 32$ und $n_0 = 0$.

$\text{Zeit}_{A_1}(n) = \Theta(n^3)$: Die Laufzeit wächst (ungefähr) um den Faktor 8, wenn wir die Eingabelänge verdoppeln.

Wie schnell dominiert die Asymptotik?

Annahme: Ein einfacher Befehl benötigt 10^{-9} Sekunden.

n	n^2	n^3	n^{10}	2^n	$n!$
16	256	4.096	$\geq 10^{12}$	65536	$\geq 10^{13}$
32	1.024	32.768	$\geq 10^{15}$	$\geq 4 \cdot 10^9$	$\geq 10^{31}$
64	4.096	262.144	$\geq 10^{18}$	$\geq 6 \cdot 10^{19}$	
128	16.384	2.097.152	mehr als 10 Jahre	mehr als 600 Jahre	mehr als 10^{14} Jahre
256	65.536	16.777.216			
512	262.144	134.217.728			
1024	1.048.576	$\geq 10^9$			
1 Mio	$\geq 10^{12}$	$\geq 10^{18}$			
	mehr als 15 Minuten	mehr als 10 Jahre			

Grenzwerte

Grenzwerte sollten das Wachstum voraussagen!

Der Grenzwert der Folge $\frac{f(n)}{g(n)}$ existiere und es sei $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$.

- Wenn $c = 0$, dann ist $f = o(g)$. (f wächst langsamer als g .)
- Wenn $0 < c < \infty$, dann ist $f = \Theta(g)$.
(f und g wachsen gleich schnell.)
- Wenn $c = \infty$, dann ist $f = \omega(g)$. (f wächst schneller als g .)
- Wenn $0 \leq c < \infty$, dann ist $f = O(g)$.
(f wächst höchstens so schnell wie g .)
- Wenn $0 < c \leq \infty$, dann ist $f = \Omega(g)$.
(f wächst mindestens so schnell wie g .)

Rechnen mit Grenzwerten

$f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ seien Funktionen.

(a) Die Operation \circ bezeichne eine der Operationen $+$, $-$ oder $*$.
Dann gilt

$$\lim_{n \rightarrow \infty} (f(n) \circ g(n)) = \lim_{n \rightarrow \infty} f(n) \circ \lim_{n \rightarrow \infty} g(n),$$

falls die beiden Grenzwerte auf der rechten Seite existieren und endlich sind.

(b) $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \frac{\lim_{n \rightarrow \infty} f(n)}{\lim_{n \rightarrow \infty} g(n)}$, falls die beiden Grenzwerte auf der rechten Seite existieren, endlich sind und $\lim_{n \rightarrow \infty} g(n) \neq 0$ gilt.

- $\lim_{n \rightarrow \infty} \frac{n^3 + n \cdot (n+1)/2}{n^3} = \lim_{n \rightarrow \infty} \frac{n^3}{n^3} + \lim_{n \rightarrow \infty} \frac{n \cdot (n+1)/2}{n^3} = 1.$
- Also ist $n^3 + \frac{n \cdot (n+1)}{2} = \Theta(n^3)$.

Die Regel von de l'Hospital

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$, falls der letzte Grenzwert existiert
und falls $\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} g(n) \in \{0, \infty\}$.

- $\lim_{n \rightarrow \infty} \log_2 n = \lim_{n \rightarrow \infty} n = \infty$,
- der Grenzwert $\lim_{n \rightarrow \infty} \frac{\log'_2(n)}{n'} = \lim_{n \rightarrow \infty} \frac{(\log_2(e) \cdot \ln(n))'}{n'} = \lim_{n \rightarrow \infty} \log_2(e) \cdot \frac{1/n}{1} = 0$ existiert und
- $\lim_{n \rightarrow \infty} \frac{\log_2(n)}{n} = 0$ folgt mit der Regel von de l'Hospital.
- Weitere Anwendungen:
 - ▶ $\log_2 \log_2 n = o(\log_2 n)$.
 - ▶ $\log_2 \log_2 \log_2 n = o(\log_2 \log_2 n)$.
 - ▶ $\log_2^{(k+1)} n = o(\log_2^{(k)} n)$ für jedes k

Eine Wachstums-Hierarchie

- $f_1(n) = 1$, dann ist $f_1 = o(\log_2 \log_2 n)$,
 $\lim_{n \rightarrow \infty} \log_2 \log_2 n = \infty$.
- $\log_2 \log_2 n = o(\log_2 n)$,
wende de l'Hospital an.
- $\log_2 n = \Theta(\log_a n)$ für jedes $a > 1$,
 $\log_2 n = \log_2 a \cdot \log_a n$.
- $\log_2 n = o(n^{1/k})$ für jedes $k > 1$,
wende de l'Hospital an.
- $n^{1/k} = o(n)$ und $n = o(n \cdot \log_2 n)$ für jedes $k > 1$,
 $\lim_{n \rightarrow \infty} \frac{n^{1/k}}{n} = \lim_{n \rightarrow \infty} \frac{1}{n^{1-1/k}} = 0$ und $\lim_{n \rightarrow \infty} \log_2 n = \infty$.
- $n \cdot \log_2 n = o(n^k)$ für jedes $k > 1$.
- $n^k = o(b^n)$ für jedes $b > 1$,
 $n^k = b^{k \cdot \log_b n}$ und $\lim_{n \rightarrow \infty} \frac{k \cdot \log_b n}{n} = 0$
- und $b^n = o(n!)$ für jedes $b > 1$.



Asymptotische Notation (1) demographic

Seien $f(n) = n^{1.1} / \log n$, $g(n) = n \log^3 n$, $h(n) = 2^{\log_2 n}$

Ordne die Funktionen aufsteigend nach asympt. Wachstum.

- (1) f, g, h
- (2) f, h, g
- (3) g, f, h
- (4) g, h, f
- (5) h, f, g
- (6) h, g, f

Auflösung: (6)



Asymptotische Notation (2) multiple choice

Sei $f : N_{>0} \rightarrow N_{>0}$. Was gilt dann?

- (1) $f(n) + 10 = O(f(n))$
- (2) $f(n + 10) = O(f(n))$

Auflösung: (1)

Das Integralkriterium

Die Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ sei gegeben.

(a) Wenn f monoton wachsend ist, dann gilt

$$\int_0^n f(x)dx \leq \sum_{i=1}^n f(i) \leq \int_1^{n+1} f(x)dx.$$

(b) Wenn f monoton fallend ist, dann gilt

$$\int_1^{n+1} f(x)dx \leq \sum_{i=1}^n f(i) \leq \int_0^n f(x)dx.$$

Zwei wichtige Konsequenzen:

- $\sum_{i=1}^n \frac{1}{i} = \Theta(\ln(n)).$
- $\sum_{i=1}^n i^k = \Theta(n^{k+1})$, falls $k \geq 0$.

Das Teilfolgenproblem: Algorithmus A_2

```
Max = -∞;  
for (i=1 ; i <= n; i++)  
    { f(i, i - 1) = 0;  
     for (j=i ; j <= n; j++)  
         { f(i, j) = f(i, j - 1) + a_j;  
          Max = max{f(i, j), Max}; } }
```

Wir benötigen genau so viele Additionen wie Vergleiche. Also ist

$$\begin{aligned}\text{Zeit}_{A_2(n)} &= \text{Additionen}_{A_2}(n) + \text{Vergleiche}_{A_2}(n) \\ &= \frac{n \cdot (n + 1)}{2} + \frac{n \cdot (n + 1)}{2} = n \cdot (n + 1).\end{aligned}$$

- $\lim_{n \rightarrow \infty} \frac{n \cdot (n + 1)}{n^2} = 1$.
- $\text{Zeit}_{A_2}(n) = \Theta(n^2)$: die Laufzeit wächst um den Faktor vier, wenn sich die Eingabelänge verdoppelt.

- Mit Hilfe der asymptotischen Notation können wir sagen, dass
 - ▶ A_1 eine kubische Laufzeit und
 - ▶ A_2 eine quadratische Laufzeit besitzt.
- Es ist $\lim_{n \rightarrow \infty} \frac{n^2}{n^3} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0$ und damit folgt $n^2 = o(n^3)$:
 A_2 ist wesentlich schneller als A_1 .
- Aber es geht noch deutlich schneller!

Nun wird es lecker ...

★

DIVIDE & CONQUER

SHARING PLATTERS



The Mini Boss Sharing Platter

★ SIDEKICKS
Any 3 sides for €10*

House Salad
Frizzled Onions

THE MINI BOSS

A mouth-watering medley of our famous Jack Daniel's® ribs, slow-cooked bbq braised beef and grilled margarita chicken. Accompanied with



Ein Divide & Conquer Ansatz

- Angenommen wir kennen den maximalen $f(i, j)$ -Wert $\text{opt}_{\text{links}}$ für $1 \leq i \leq j \leq \lceil \frac{n}{2} \rceil$ sowie den maximalen $f(i, j)$ -Wert $\text{opt}_{\text{rechts}}$ für $\lceil \frac{n}{2} \rceil + 1 \leq i \leq j \leq n$.
 - Welche Möglichkeiten gibt es für den maximalen $f(i, j)$ -Wert opt für $1 \leq i \leq j \leq n$?
-
- Möglichkeit 1: $\text{opt} = \text{opt}_{\text{links}}$ und die maximale Teilfolge liegt in linken Hälfte.
 - Möglichkeit 2: $\text{opt} = \text{opt}_{\text{rechts}}$ und die maximale Teilfolge liegt in rechten Hälfte.
 - Möglichkeit 3: $\text{opt} = \max\{f(i, j) \mid 1 \leq i \leq \lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil + 1 \leq j \leq n\}$ und eine maximale Teilfolge beginnt in der linken und endet in der rechten Hälfte.

Das Teilfolgenproblem: Algorithmus A_3

$A_3(i, j)$ bestimmt den Wert einer maximalen Teilfolge für a_i, \dots, a_j .

- (1) Wenn $i = j$, dann gib a_i als Antwort aus.
- (2) Ansonsten setzte $\text{mitte} = \lfloor \frac{i+j}{2} \rfloor$;
- (3) $\text{opt}_1 = \max\{f(k, \text{mitte}) \mid i \leq k \leq \text{mitte}\}$;
 $\text{opt}_2 = \max\{f(\text{mitte} + 1, l) \mid \text{mitte} + 1 \leq l \leq j\}$;
- (4) $\text{opt} = \max\{A_3(i, \text{mitte}), A_3(\text{mitte} + 1, j), \text{opt}_1 + \text{opt}_2\}$ wird als Antwort ausgegeben.

- $n = j - i + 1$ ist die Eingabelänge. n sei eine Zweierpotenz.
- Wie bestimmt man die Laufzeit $\text{Zeit}_{A_3}(n)$?
 - ▶ $\text{Zeit}_{A_3}(1) = 1$,
 - ▶ In Schritt (4) werden zwei rekursive Aufrufe für Eingabelänge $\frac{n}{2}$ ausgeführt mit Laufzeit jeweils $\text{Zeit}_{A_3}(\frac{n}{2})$. Dazu kommen $t(n)$ weitere „nicht-rekursive“ Operationen aus den Schritten (3) und (4).
 - ▶ Also $\text{Zeit}_{A_3}(n) = 2 \cdot \text{Zeit}_{A_3}(\frac{n}{2}) + t(n)$.

Wie löst man Rekursionsgleichungen?

- Wie groß ist der „Overhead“ $t(n)$?
 - ▶ $\frac{n}{2} - 1 + \frac{n}{2} - 1 = n - 2$ Additionen und Vergleiche genügen für die Berechnung von opt_1 und opt_2 . Eine weitere Addition wird für $\text{opt}_1 + \text{opt}_2$ benötigt, zwei weitere Vergleiche fallen in Schritt (4) an.
 - ▶ $t(n) = (n - 2 + 1) + (n - 2 + 2) = 2n - 1$.
- Wir erhalten die Rekursionsgleichungen:
$$\text{Zeit}_{A_3}(1) = 1$$
$$\text{Zeit}_{A_3}(n) = 2 \cdot \text{Zeit}_{A_3}\left(\frac{n}{2}\right) + 2n - 1.$$
- Der allgemeinere Fall: Ein rekursives Programm A mit Eingabelänge n besitze a rekursive Aufrufe mit Eingabelänge $\frac{n}{b}$. Schließlich werden $t(n)$ nicht-rekursive Operationen ausgeführt. Die Rekursionsgleichung:

$$\text{Zeit}_A(n) = a \cdot \text{Zeit}_A\left(\frac{n}{b}\right) + t(n).$$

Der Baum der Rekursionsgleichung

Die Rekursionsgleichung

$$T(1) = c, T(n) = a \cdot T\left(\frac{n}{b}\right) + t(n)$$

ist zu lösen. n sei eine Potenz der Zahl $b > 1$ und $a \geq 1, c > 0$ gelte.

Der Baum der Rekursionsgleichung:

1. Wir sagen, dass die Wurzel `root` die Eingabelänge n hat.
 - ▶ Wenn $n = 1$, dann markiere `root` mit c und `root` wird zu Blatt.
 - ▶ Wenn $n > 1$, dann markiere `root` mit $t(n)$. `root` erhält a Kinder mit Eingabelänge n/b .
2. Sei v ein Knoten des Baums mit Eingabelänge m .
 - ▶ Wenn $m = 1$, dann markiere v mit c und v wird zu Blatt.
 - ▶ Wenn $m > 1$, dann markiere v mit $t(m)$ und v erhält a Kinder mit Eingabelänge m/b .

$T(n)$ stimmt überein mit der Summe aller Knotenmarkierungen.

Expansion und Vermutung

Die Rekursionsgleichung

$$T(1) = c, T(n) = a \cdot T\left(\frac{n}{b}\right) + t(n)$$

ist zu lösen. n sei eine Potenz der Zahl $b > 1$ und $a \geq 1, c > 0$ gelte.

Wir expandieren die Rekursionsgleichung und erhalten

$$\begin{aligned} T(n) &= a \cdot T\left(\frac{n}{b}\right) + t(n) \\ &= a \left(a \cdot T\left(\frac{n}{b^2}\right) + t\left(\frac{n}{b}\right) \right) + t(n) \\ &= a^2 \cdot T\left(\frac{n}{b^2}\right) + a \cdot t\left(\frac{n}{b}\right) + t(n) \\ &= a^3 \cdot T\left(\frac{n}{b^3}\right) + a^2 \cdot t\left(\frac{n}{b^2}\right) + a \cdot t\left(\frac{n}{b}\right) + t(n) = \dots \\ &\stackrel{?}{=} a^k \cdot T\left(\frac{n}{b^k}\right) + a^{k-1} \cdot t\left(\frac{n}{b^{k-1}}\right) + \dots + a \cdot t\left(\frac{n}{b}\right) + t(n). \end{aligned}$$

Eliminierung rekursiver Terme

Beweise die Vermutung mit Hilfe der mathematischen Induktion.

Setze $k = \log_b n$. Dann ist $b^k = n$ und $T(\frac{n}{b^k}) = T(1) = c$.

$$\begin{aligned}T(n) &= a^k \cdot T\left(\frac{n}{b^k}\right) + a^{k-1} \cdot t\left(\frac{n}{b^{k-1}}\right) + \cdots + a \cdot t\left(\frac{n}{b}\right) + t(n) \\&= a^{\log_b n} \cdot T(1) + \sum_{j=0}^{\log_b n - 1} a^j \cdot t\left(\frac{n}{b^j}\right) \\&= a^{\log_b n} \cdot c + \sum_{j=0}^{\log_b n - 1} a^j \cdot t\left(\frac{n}{b^j}\right) = n^{\log_b a} \cdot c + \sum_{j=0}^{\log_b n - 1} a^j \cdot t\left(\frac{n}{b^j}\right).\end{aligned}$$

Eine Beobachtung

Wenn $t(n) \leq d \cdot n^{\log_b(a) - \varepsilon}$ für Konstanten $\varepsilon > 0$ und $d > 0$, dann dominiert der erste Summand $n^{\log_b a} \cdot c$.

Zusammenfassung: Das Mastertheorem

Die Rekursion

$$T(1) = c, T(n) = a \cdot T\left(\frac{n}{b}\right) + t(n)$$

sei zu lösen. n ist eine Potenz der Zahl $b > 1$ und $a \geq 1, c > 0$ gelte.

- (a) Wenn $t(n) = O(n^{(\log_b a) - \varepsilon})$ für eine Konstante $\varepsilon > 0$, dann ist $T(n) = \Theta(n^{\log_b a})$. **Die Blätter dominieren.**
- (b) Wenn $t(n) = \Theta(n^{\log_b a})$, dann ist $T(n) = \Theta(n^{\log_b a} \cdot \log_b n)$. **Gleichgewicht zw. den $\log_b n$ Schichten des Rek.-Baums.**
- (c) Wenn $t(n) = \Omega(n^{(\log_b a) + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $a \cdot t\left(\frac{n}{b}\right) \leq \alpha t(n)$ für eine Konstante $\alpha < 1$, dann $T(n) = \Theta(t(n))$. **Die Wurzel dominiert.**

(Die zugehörigen Beweise finden sich im Skript.)

Die Laufzeit von Algorithmus A_3

Wir hatten die Rekursionsgleichungen

$$\text{Zeit}_{A_3}(1) = 1, \text{Zeit}_{A_3}(n) = 2 \cdot \text{Zeit}_{A_3}\left(\frac{n}{2}\right) + 2n - 1$$

erhalten. Wir bestimmen zuerst die asymptotische Lösung für die Rekursionsgleichung von Zeit_{A_3} .

- Um das Mastertheorem anzuwenden, müssen wir $c = 1, a = 2, b = 2$ und $t(n) = 2n - 1$ setzen.
- In welchem Fall befinden wir uns?
 - Es ist $\log_b a = \log_2 2 = 1$ und $t(n) = \Theta(n^{\log_b a})$.
 - Fall (b) ist anzuwenden und liefert $\text{Zeit}_{A_3}(n) = \Theta(n \log_2 n)$.
- Algorithmus A_3 ist schneller als Algorithmus A_2 , denn

$$\lim_{n \rightarrow \infty} \frac{\text{Zeit}_{A_3}(n)}{\text{Zeit}_{A_2}(n)} = \lim_{n \rightarrow \infty} \frac{n \log_2 n}{n^2} = \lim_{n \rightarrow \infty} \frac{\log_2 n}{n} = 0.$$



Und jetzt Sie . . . demogr.

Die Rekursion $T(1) = c$, $T(n) = a \cdot T\left(\frac{n}{b}\right) + t(n)$ sei zu lösen. n ist eine Potenz der Zahl $b > 1$ und $a \geq 1$, $c > 0$ gelte.

- (a) Wenn $t(n) = O\left(n^{(\log_b a) - \varepsilon}\right)$ für eine Konstante $\varepsilon > 0$, dann ist $T(n) = \Theta(n^{\log_b a})$.
- (b) Wenn $t(n) = \Theta(n^{\log_b a})$, dann ist $T(n) = \Theta(n^{\log_b a} \cdot \log_b n)$.
- (c) Wenn $t(n) = \Omega\left(n^{(\log_b a) + \varepsilon}\right)$ für eine Konstante $\varepsilon > 0$ und $a \cdot t\left(\frac{n}{b}\right) \leq \alpha \cdot t(n)$ für eine Konstante $\alpha < 1$, dann $T(n) = \Theta(t(n))$.

Lösung für $T(1) = \Theta(1)$, $T(n) = 4 \cdot T(n/2) + \Theta(n)$?

- (1) $T(n) = \Theta(n^4)$
- (2) $T(n) = \Theta(n^2 \cdot \log n)$
- (3) $T(n) = \Theta(n^2)$
- (4) $T(n) = \Theta(n \cdot \log^2 n)$
- (5) $T(n) = \Theta(n \cdot \log n)$ Lsg: (3)

Was kann das Mastertheorem nicht? I

Um das Mastertheorem anzuwenden, muss stets dieselbe Anzahl a von Teilproblemen mit derselben Eingabelänge $\frac{n}{b}$ rekursiv aufgerufen werden. Beachte, dass sich auch b nicht ändern darf.

Die Rekursion $T(1) = 1, T(n) = T(n - 1) + n$. Das Mastertheorem ist nicht anwendbar, da b sich ändert. Stattdessen:

- Expandiere, um eine Vermutung über die allgemeine Form der entwickelten Rekursion zu erhalten.

$$T(n) = T(n - 1) + n = T(n - 2) + (n - 1) + n = \dots \stackrel{?}{=} \\ T(n - k) + (n - (k - 1)) + \dots + n.$$

- Beweise deine Vermutung.
- Eliminiere rekursive Terme: Setze $k = n - 1$.
- Bestimme eine asymptotische Lösung:

$$T(n) = T(n - (n - 1)) + (n - (n - 2)) + \dots + n = \\ T(1) + 2 + 3 + \dots + n = \sum_{i=1}^n i = \frac{n \cdot (n+1)}{2} = \Theta(n^2).$$

Was kann das Mastertheorem nicht? II

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab3;
else {
    Hanoi(N-1,stab1,stab3,stab2);
    bewege einen Ring von Stab stab1 nach Stab stab3;
    Hanoi(N-1,stab2,stab1,stab3); }}
```

- Sei $T(N)$ die Anzahl der Ringbewegungen nach Aufruf des Programms $\text{Hanoi}(N, *, *, *)$.
- Dann gilt $T(1) = 1$ und $T(N) = 2 \cdot T(N - 1) + 1$.
- Das Mastertheorem ist nicht anwendbar, stattdessen expandiere

$$\begin{aligned}T(N) &= 2 \cdot T(N - 1) + 1 = 2^2 \cdot T(N - 2) + (1 + 2) \quad \dots \\&= 2^k \cdot T(N - k) + (1 + 2 + \dots + 2^{k-1}) \\&= 1 + 2 + \dots + 2^{N-1} = 2^N - 1.\end{aligned}$$

Das Teilstufenproblem: Algorithmus A₄

Setze $\text{Max}_k = \max\{f(i, j) | 1 \leq i \leq j \leq k\}$. Max_k ist der Wert einer optimalen Teilstufe für die ersten k Folgenelemente.
Wie berechnen wir Max_{k+1} ?

- Setze $\text{Max}_k^* = \max\{f(i, k) | 1 \leq i \leq k\}$.
- Dann ist $\text{Max}_{k+1} = \max\{\text{Max}_k, \text{Max}_{k+1}^*\}$. Warum?
 - ▶ Fall 1: Eine optimale Teilstufe enthält das Folgenelement a_{k+1} **nicht**. Dann ist $\text{Max}_{k+1} = \text{Max}_k$.
 - ▶ Fall 2: Eine optimale Teilstufe enthält das Folgenelement a_{k+1} . Dann ist $\text{Max}_{k+1} = \text{Max}_{k+1}^*$.
- $\text{Max}_{k+1}^* = \max\{\text{Max}_k^* + a_{k+1}, a_{k+1}\}$. Warum?
 - ▶ Fall 1: Eine optimale, mit dem $k+1$ sten Folgenelement endende Folge enthält nicht nur a_{k+1} . Dann ist $\text{Max}_{k+1}^* = \text{Max}_k^* + a_{k+1}$.
 - ▶ Fall 2: Sonst ist $\text{Max}_{k+1}^* = a_{k+1}$.

Algorithmus A_4

(1) $\text{Max}_1 = \text{Max}_1^* = a_1$.

(2) Für $k = 1, \dots, n - 1$ setze

$$\begin{aligned}\text{Max}_{k+1}^* &= \max\{\text{Max}_k^* + a_{k+1}, a_{k+1}\} \text{ und} \\ \text{Max}_{k+1} &= \max\{\text{Max}_k, \text{Max}_{k+1}^*\}.\end{aligned}$$

(3) Max_n wird ausgegeben.

- Pro Schleifendurchlauf: zwei Vergleiche und eine Addition.
Also insgesamt $2(n - 1)$ Vergleiche und $n - 1$ Additionen.
- $\text{Zeit}_{A_4}(n) = 3(n - 1)$ und A_4 hat lineare Laufzeit, da $\text{Zeit}_{A_4} = \Theta(n)$.
- A_4 ist schneller als A_3 , denn $\lim_{n \rightarrow \infty} \frac{n}{n \log_2 n} = \lim_{n \rightarrow \infty} \frac{1}{\log_2 n} = 0$.

- Zähle die Anzahl ausgeführter elementarer Anweisungen wie etwa Zuweisungen und Auswahl-Anweisungen (if-else und switch).
Weder iterative Anweisungen (for, while und do-while), noch Sprunganweisungen (break, continue, goto und return) oder Funktionsaufrufe sind zu zählen.
- Ist der so ermittelte Aufwand denn nicht zu klein?
 - ▶ Die in einer Schleife oder in einer Funktion ausgeführten elementaren Anweisungen sind sehr wohl zu zählen.
 - ▶ In vernünftigen Programmen wird die asymptotische Laufzeit korrekt ermittelt.
- In den meisten Fällen sind wir an der worst-case Laufzeit interessiert:
Bestimme für jede Eingabelänge n die maximale Laufzeit einer Eingabe der Länge n .

- **Zuweisungen:** Eine Zuweisung zu einer „einfachen“ Variablen ist einfach zu zählen, eine Zuweisung zu einer Array-Variablen ist mit der Länge des Arrays zu gewichten.
- **Auswahl-Anweisungen:** die Bedingung ist zu zählen.
 - ▶ Aber die Laufzeit hängt jetzt vom Wert der Bedingung ab!
 - ▶ Häufig genügt die Bestimmung des Gesamtaufwand für den **längsten** der alternativen Anweisungsblöcke.
- **Schleifen:** Der Aufwand kann in jedem Schleifendurchlauf variieren!
 - Häufig genügt die Bestimmung der maximalen Anzahl der ausführbaren Anweisungen innerhalb einer Schleife sowie die Anzahl der Schleifendurchläufe.

Beispiel: For-Schleifen

Matrizenmultiplikation $A \cdot B = C$

```
for (i=0; i < n; i++)
    for (j=0; j < n; j++)
        {C[i][j] = 0;
         for (k=0; k < n ; k++)
             C[i][j] += A[i][k]*B[k][j]; }
```

Analysiere die geschachtelten for-Schleifen durch geschachtelte Summen:

$$\begin{aligned}\text{Laufzeit} &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(1 + \sum_{k=0}^{n-1} 1\right) \\ &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (1 + n) \\ &= n^2 \cdot (1 + n) = \Theta(n^3).\end{aligned}$$

Beispiel: While Schleifen I

```
while (n >= 1)
    {n = n/2;
     Eine Menge von maximal c Anweisungen }
```

- $T(n)$ bezeichne die Laufzeit der while-Schleife für Parameter n .
Es gilt $T(1) \leq 1 + c$ und $T(n) \leq T(n/2) + 1 + c$.
- Wende das Mastertheorem an:
 - ▶ $a = 1$, $b = 2$ und $t(n) \leq 1 + c$.
 - ▶ Es ist $t(n) = \Theta(1) = \Theta(n^{\log_b a})$.
 - ▶ Fall 2 ist anzuwenden und $T(n) = \Theta(\log_2 n)$ folgt.
- Die Laufzeit ist logarithmisch.



while ($n \geq 10$)

{ $n = \sqrt{n}$;

Eine Menge von maximal c Anweisungen }

Wie hoch ist die Laufzeit?

- (1) $\Theta(\sqrt{n})$
- (2) $\Theta(\sqrt{n} \cdot \log n)$
- (3) $\Theta(\sqrt{\log n})$
- (4) $\Theta(\log \log n)$
- (5) $\Theta(\log n)$

Auflösung: (4)

Beispiel: While Schleifen II

```
while (n > 1)  
    n = ( n & 1 ) ? 3*n + 1 : n/2;
```

- Wenn n ungerade ist, dann ersetze n durch $3 \cdot n + 1$.
- Wenn n gerade ist, dann ersetze n durch $\frac{n}{2}$.
- Was ist die asymptotische Laufzeit in Abhängigkeit von n ?
 - ▶ Es ist bis heute nicht bekannt, ob die Schleife für jede Eingabe terminiert!
 - ▶ Die Laufzeit ist deshalb natürlich auch nicht bekannt.
- Die Analyse der Laufzeit kann schwierig sein!