Organisatorisches

- Vorlesung am Dienstag von 16:15 bis 17:45 Uhr im Hörsaal O25 - H1
- Vorlesung am Donnerstag von 14:15 bis 15:45 Uhr im Hörsaal 45 - H 45.2 (Uni West)
- Bei erfolgreicher Teilnahme an den Übungen erhalten Sie einen Notenbonus.
- Einteilung der Tutorien erfolgt im Moodle

Die Übungen leitet Lisa-Marie Jaser.

Algorithmus

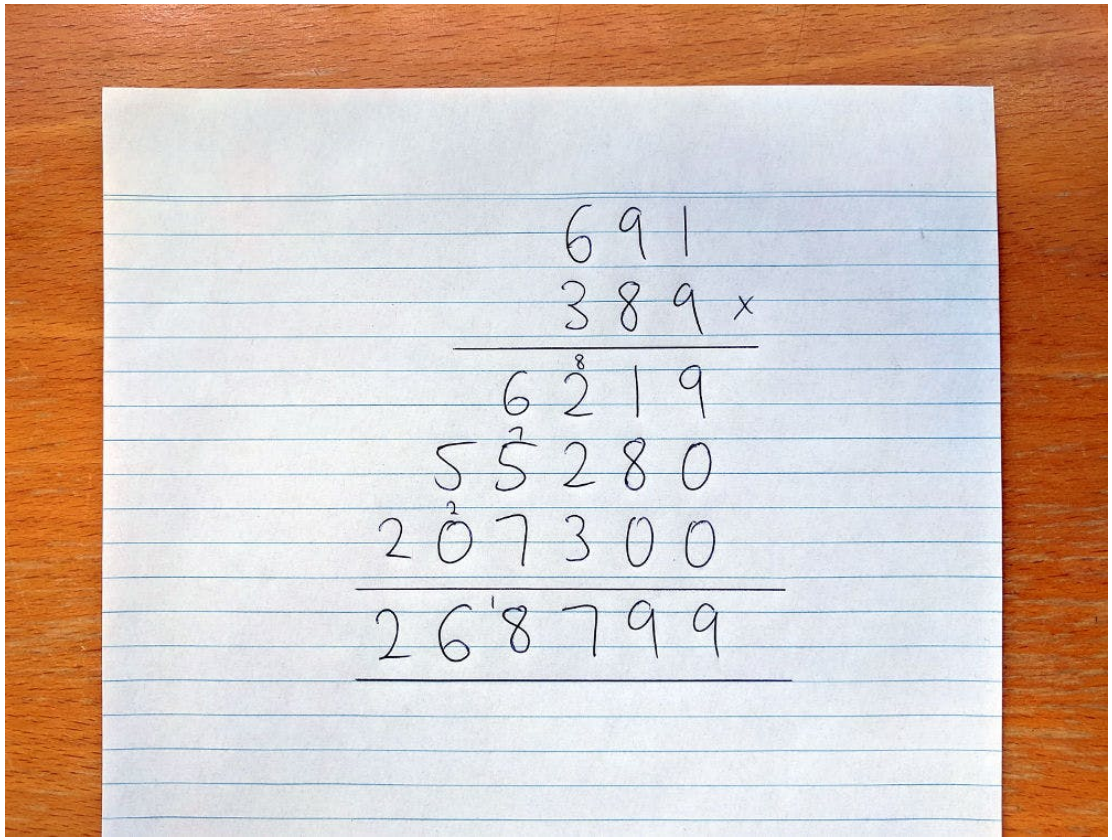
Allgemeines (eindeutiges) Verfahren zur Lösung einer Klasse gleichartiger Probleme (z.B. zur Berechnung einer Funktion für verschiedene Argumente), gegeben durch eines aus elementaren Anweisungen an einen (menschlichen oder maschinellen) Rechner bestehenden Text.

`Lexikon der Informatik und Datenverarbeitung, 1991`

Algorithmus

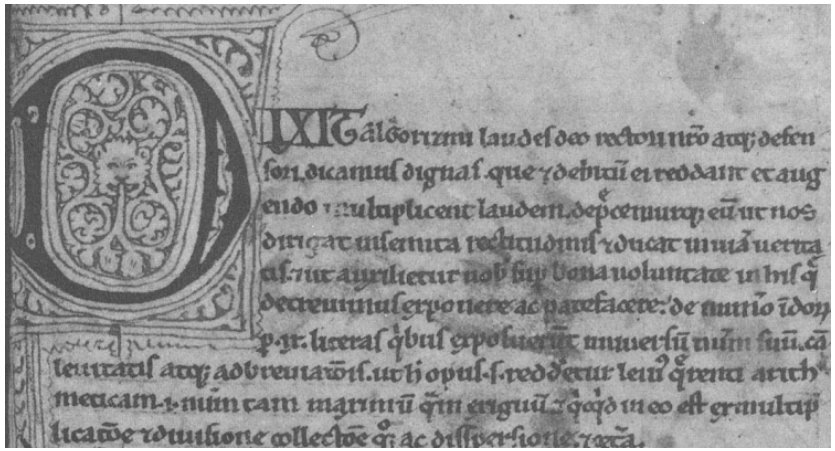
- Die Rechenvorschrift besteht aus einem endlichen Text.
- Der Ablauf einer Berechnung geschieht schrittweise als Folge von elementaren Rechenschritten.
- In jedem Stadium einer Rechnung ist eindeutig bestimmt, welcher Rechenschritt als nächster durchgeführt wird.
- Der nächste Rechenschritt hängt nur von der Eingabe und den bisher berechneten Zwischenergebnissen ab.

Multiplikation



Al-Khowarizmi

Abu Ja'far Mohammed ibn Musa **al-Khowarizmi** (geboren um 780)



(Mohammed, Vater des Ja'far, Sohn des Mose, geboren in Khowarizmi)

Algorithmus von Euklid

Algorithmus zur Berechnung des größten gemeinsamen Teilers (etwa 300vC.)

Procedure Euklid(a, b)

`If` $b = 0$ `Then` **return** a

`Else` **return** Euklid($b, a \mod b$)

Für $a = 21$ und $b = 30$ ergibt sich

$$\begin{aligned} \text{Euklid}(21, 30) &= \text{Euklid}(30, 21) \\ &= \text{Euklid}(21, 9) \\ &= \text{Euklid}(9, 3) \\ &= \text{Euklid}(3, 0) \\ &= 3 \end{aligned}$$

Inhalt

- Grundlagen/Konzepte
- Sortier- und Selektionsalgorithmen
- Hashing
- Dynamisches Programmieren
- Greedy-Algorithmen
- Algorithmen auf Graphen
- Algebraische und zahlentheoretische Algorithmen
- String Matching Verfahren

Literatur

- H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2009.
- U. Schöning, Algorithmik, Spektrum Akademischer Verlag, 2001.

Analyse von Algorithmen

Ein Algorithmus A wird mit einer Eingabe x gestartet.

- Die Eingabe ist z.B. ein Graph G, eine Matrix M, eine Folge von Zahlen $(a_1, a_2, \ldots, a_n)$, eine Zahl L (Primzahltest).

Die Eingabe x hat eine Länge $n = |x|$. Zum Beispiel ist n

- die Anzahl der Knoten und Kanten von G
- $k \cdot l$ für eine $k \times l$ Matrix M
- die Anzahl der Zahlen $a_1, a_2, \ldots, a_n$
- die Länge der Binärdarstellung von L

Uniformes Komplexitätsmaß

- Wird die Komplexität der Elementaroperationen (z.B. die Addition zweier Zahlen) als konstant angenommen, so spricht man vom uniformen Komplexitätsmaß.
- Gerechtfertigt, wenn alle in einem Programm vorkommenden Zahlen in jeweils einer Speicherzelle untergebracht werden können.
- Diese Annahme vereinfacht die Komplexitätsanalyse von Algorithmen ganz entscheidend.
- Für einen deterministischen Algorithmus A ist mit der Eingabe x der Rechenablauf und damit dessen Länge festgelegt.
- $\text{zeit}_A(x)$ ist die Laufzeit (d.h. Anzahl der durchlaufenen Elementarschritte).

Bit-Komplexitätsmaß

- Beim Bit-Komplexitätsmaß messen wir die Eingabelänge anhand der Anzahl der Bits, die die Eingabe (in codierter Form) in Anspruch nimmt.
- Bei natürlichen Zahlen ist dies die Länge der Binärdarstellung.
- Dann können wir auch nicht mehr annehmen, dass elementare Rechenoperationen, wie die Addition, mit konstantem Aufwand ausführbar sind.
- Vielmehr müssen wir jetzt den Aufwand jeder Operationen bis auf Bitebene ausrechnen.
- Zum Beispiel kann man n-Bit-Zahlen mit Aufwand $O(n)$ addieren und mit Aufwand $O(n^2)$ multiplizieren.

worst-case vs. average-case

Die *worst-case Komplexität* von A ist die Laufzeit bei der ungünstigsten Eingabe der Länge n:

$$\text{wc-zeit}_A(n) = \max\{\text{zeit}_A(x) \mid |x| = n\}.$$

Bei der *average-case Komplexität* wird Gleichverteilung auf allen Eingaben der Länge n angenommen.

- Eingaben werden zufällig ausgewählt.
- Damit wird die Laufzeit zu einer Zufallsvariablen $X_{A,n}$.
- Erwartungswertoperator E.

$$\begin{aligned}\text{av-zeit}_A(n) &= E[X_{A,n}] \\ &= \frac{1}{|\{x : |x| = n\}|} \sum_{x:|x|=n} \text{zeit}_A(x),\end{aligned}$$

Zeitkomplexität

Sei A ein Algorithmus und f und g Funktionen von $\mathbb{N}$ auf $\mathbb{R}_+$.

- Falls wc-zeit$_A(n) \leq f(n)$ dann macht der Algorithmus für jede Eingabe x der Länge n höchstens $f(n)$ Schritte.
- Falls wc-zeit$_A(n) \geq g(n)$ dann gibt es Eingaben x der Länge n für die der Algorithmus mindestens $g(n)$ Schritte benötigt.

Warum Gleichverteilung ?

- Tatsächliche Verteilung ist unbekannt.
- Gleichverteilung besitzt maximale Entropie.
  (Maximale Ungewissheit, Algorithmus kann sich nicht auf einige typische Eingaben spezialisieren.)

Beispiel: Maximum bestimmen

Eingabe: Array $a[1..n]$ von Zahlen.

(1) max := $a[1]$
(2) FOR $i := 2$ TO n DO
(3) IF $a[i] >$ max THEN $\underbrace{\text{max} := a[i]}_{(4)}$

- die Zuweisung (Schritt 1) habe Aufwand c_1
- ein Schleifendurchlauf (Schritt 2 und 3) habe Aufwand c_2 und
- die Zuweisung (Schritt 4) habe Aufwand c_4.

Worst-case Komplexität:

$$\text{wc-zeit}_A(n) = c_1 + (n-1) \cdot (c_2 + c_4)$$

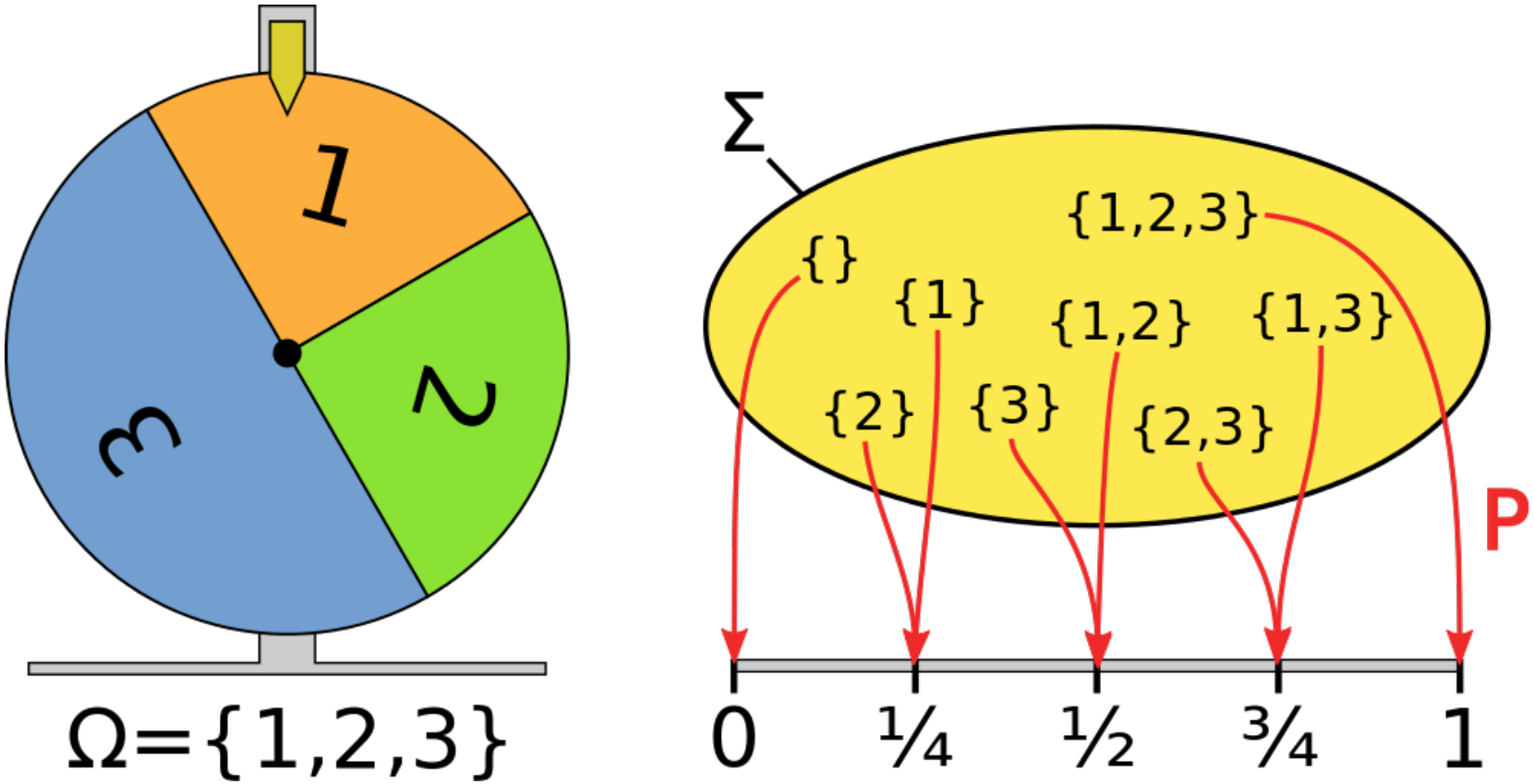
Tritt bei einer aufsteigend sortierten Folge von Zahlen ein.

Elementare Wahrscheinlichkeitsrechnung

Zur Analyse der average-case Komplexität wiederholen wir elementare Begriffe aus der Wahrscheinlichkeitsrechnung.

- Wir betrachten ein Zufallsexperiment, das endlich viele Elementarereignisse $\omega_1, \omega_2, \ldots, \omega_n$ haben kann.
- Endliche Grundmenge $\Omega := \{\omega_1, \omega_2, \ldots, \omega_n\}$
- Wahrscheinlichkeitsmaß $P : \Omega \to [0, 1]$ mit $\sum_{i=1}^{n} P(\omega_i) = 1$.
- Wahrscheinlichkeitsraum besteht aus der Grundmenge Ω und dem Wahrscheinlichkeitsmaß P.
- $E \subseteq \Omega$ heißt Ereignis.
- $P(E) := \sum_{\omega_i \in E} P(\omega_i)$.
- Hat jedes Elementarereignis die Wahrscheinlichkeit $1/n$, so spricht man von Gleichverteilung. In diesem Fall gilt $P(E) = |E|/|\Omega|$.

Wahrscheinlichkeitsmaß: Beispiel aus Wikipedia



Einmaliges Drehen eines Glücksrades

Eigenschaften eines Wahrscheinlichkeitsmaßes

Es gilt: $P(\emptyset) = 0$ und $P(\Omega) = 1$

Für das Komplementärereignis $\overline{E} = \Omega - E$ eines Ereignisses E gilt: $P(\overline{E}) = 1 - P(E)$

- Glücksrad: $P(\overline{\{2\}}) = P(\{1,3\}) = 1 - P(\{2\}) = 3/4$

Es gilt: $P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$

- Glücksrad: $P(\{1,2\} \cup \{1,3\}) =$
  $P(\{1,2\}) + P(\{1,3\}) - P(\{1\}) = 1/2 + 3/4 - 1/4 = 1$

Zwei Ereignisse E_1 und E_2 sind unabhängig, falls gilt:

$$P(E_1 \cap E_2) = P(E_1) \cdot P(E_2)$$

- Glücksrad: $0 = P(\{1\} \cap \{2\}) \neq P(\{1\}) \cdot P(\{2\}) = 1/16$

Zweimaliges Drehen des Glücksrades

$$\Omega' := \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\}$$

$$P'((x,y)) = P(x) \cdot P(y)$$

$$A = \{1 \text{ beim ersten Drehen des Rades}\} = \{(1,1),(1,2),(1,3)\}$$

$$B = \{2 \text{ beim zweiten Drehen des Rades}\} = \{(1,2),(2,2),(3,2)\}$$

$$P'(A \cap B) = P'(\{(1,2)\}) = P'((1,2)) = P(1) \cdot P(2) = 1/16$$

$$P'(A) \cdot P'(B) = \sum_{(x,y) \in A} P'((x,y)) \cdot \sum_{(x,y) \in B} P'((x,y)) = 1/16$$

D.h. A und B sind stochastisch unabhängige Ereignisse.

Zufallsvariablen

Eine Zufallsvariable ist eine Funktion $X : \Omega \to \mathbb{R}$. Der Erwartungswert $E(X)$ einer Zufallsvariablen X gibt an, welchen Wert X "im Mittel" annimmt. Formal:

$$E(X) = \sum_{x} x \cdot P(X = x)$$

Hierbei durchläuft die Summe die endlich vielen Werte x, die die Zufallsvariable X annehmen kann. Beispiel Glücksrad:

$$E(X) = \sum_{x \in \{1,2,3\}} x \cdot P(X = x) =$$

Seien $X_1, X_2, \ldots, X_k$ Zufallsvariablen, dann gilt:

$$E(\sum_{i=1}^{k} X_i) = \sum_{i=1}^{k} E(X_i)$$

Maximum bestimmen: average-case Komplexität

Eingabe: Array $a[1..n]$ von Zahlen.

(1) max $:= a[1]$

(2) FOR $i := 2$ TO n DO

(3) IF $a[i] >$ max THEN $\underbrace{\text{max} := a[i]}_{(4)}$

Annahme: Die n Eingabezahlen sind paarweise verschieden und alle $n!$ Permutationen der Eingabezahlen sind gleich wahrscheinlich. Dann gilt

$$\begin{aligned} \Pr[a[2] > a[1]] &= 1/2 \\ \Pr[a[3] > \max\{a[1], a[2]\}] &= 1/3 \\ &\vdots \\ \Pr[a[j] > \max\{a[1], \ldots, a[j-1]\}] &= 1/j \end{aligned}$$

Sei X_j die Zufallsvariable mit

$$X_j = \begin{cases} 1, & a[j] \text{ ist Maximum in } a[1..j] \\ 0, & \text{sonst.} \end{cases}$$

Es gilt: $E[X_j] = 1 \cdot \Pr[X_j = 1] + 0 \cdot \Pr[X_j = 0] = 1/j$

$$\begin{aligned} \text{av-zeit}_A(n) &= c_1 + (n-1)c_2 + c_4 \cdot E\left[\sum_{j=2}^{n} X_j\right] \\ &= c_1 + (n-1)c_2 + c_4 \cdot \sum_{j=2}^{n} E[X_j] \\ &= c_1 + (n-1)c_2 + c_4 \cdot \sum_{j=2}^{n} 1/j \\ &= c_1 + (n-1)c_2 + c_4 \cdot (H_n - 1) \end{aligned}$$

Dabei ist $H_n = \sum_{j=1}^{n} 1/j$ die n-te Harmonische Summe.

Vergleich worst-case und average-case Komplexität

Eingabe: Array $a[1..n]$ von Zahlen.

(1) max := $a[1]$

(2) FOR $i := 2$ TO n DO

(3) IF $a[i] >$ max THEN $\underbrace{\text{max} := a[i]}_{(4)}$

- wc-zeit$_A(n) = c_1 + (n-1)(c_2 + c_4)$
- av-zeit$_A(n) = c_1 + (n-1)c_2 + c_4 \cdot (H_n - 1)$
- Es gilt $H_n \leq 1 + \ln n$ (Beweis folgt später)
- av-zeit$_A(n) \leq c_1 + (n-1)c_2 + c_4 \cdot \ln n$

Im Mittel (Durchschnitt, average-case) wird die Anweisung (Schritt 4) nur logarithmisch oft ausgeführt.

Abschätzung Harmonische Summe

- Die n-te Harmonische Summe

$$H_n := \sum_{i=1}^{n} \frac{1}{i}$$

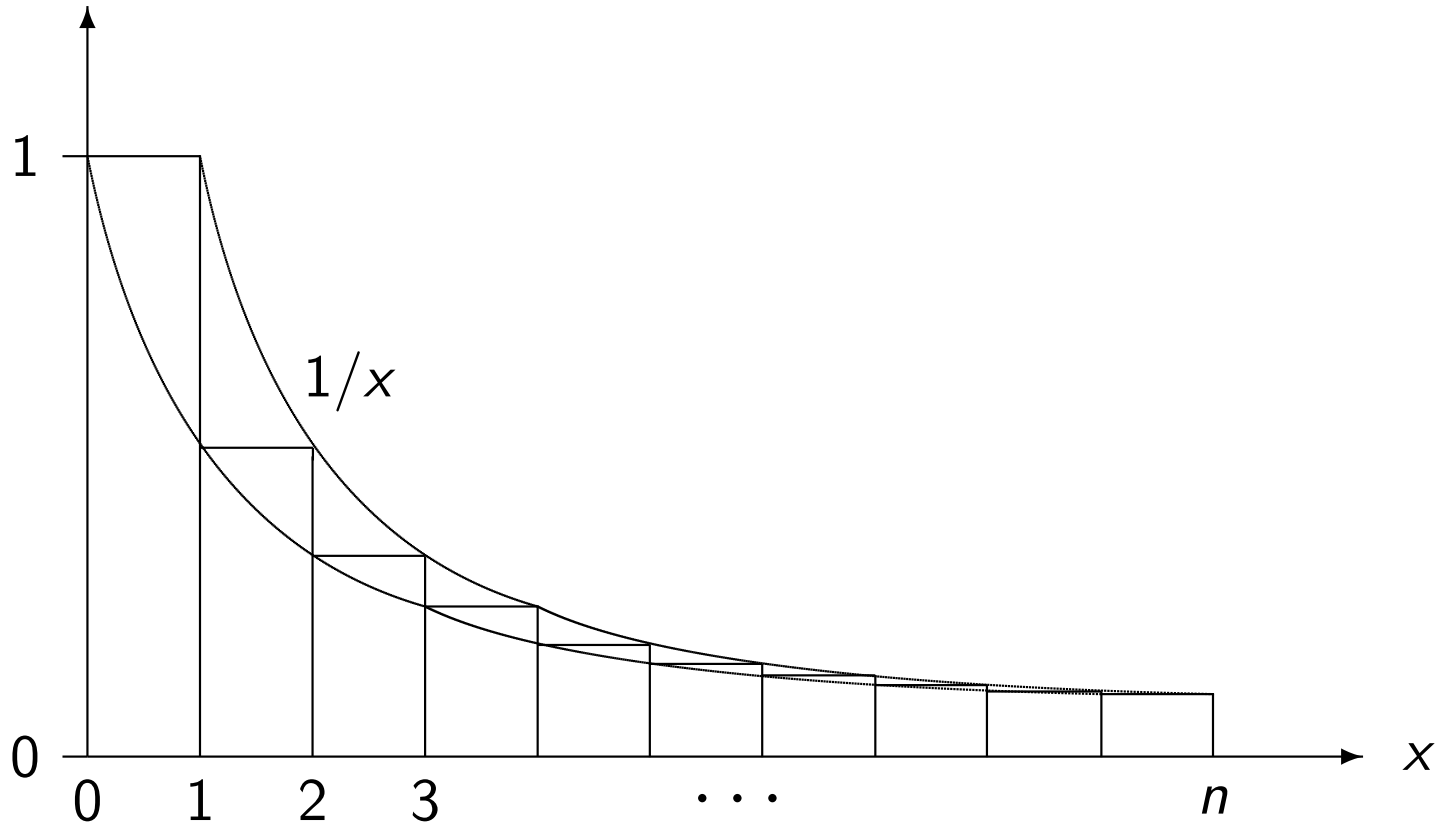
lässt sich von oben und unten durch ein Integral abschätzen:

$$\ln(n+1) = \int_1^{n+1} \frac{1}{x} dx \leq H_n \leq 1 + \int_1^n \frac{1}{x} dx = 1 + \ln n.$$

damit ergibt sich:

$$\ln(n+1) \leq H_n \leq 1 + \ln n$$

Abschätzung Harmonische Summe



Weitere nützliche Abschätzung

Es gilt

$$1 < \frac{n!}{\sqrt{2\pi n} \cdot (\frac{n}{e})^n} < 1 + \frac{1}{11n}$$

Wir schreiben $f(n) \sim g(n)$ gdw. $\lim_{n\to\infty} f(n)/g(n) = 1$.

Stirlingsche Formel:

$$n! \sim \sqrt{2\pi n} \cdot (\frac{n}{e})^n$$

Mit $\log(n^n/e^n) = \log n^n - \log e^n = n \log n - n \log e$ folgt

$$\log(n!) \sim n \log n - n \log e + \log \sqrt{2\pi n}$$

Asymptotische Notationen

Die genaue Angabe der Komplexitätsfunktionen eines Algorithmus ist oft schwierig. Die konkrete Laufzeit hängt ab von z.B.

- dem Rechner
- der Programmiersprache
- dem Compiler

Die Komplexität der Elementaroperationen wird hier als konstant angenommen (uniformes Maß).

Beispiel Komplexitätsfunktion

Eingabe: Array $a[1..n]$ von Zahlen
Ausgabe: kleinster Index m mit $\sum_{i=1}^{m} a[i] \geq \sum_{i=1}^{n} a[i]/2$

```
sum[1] := a[1]
FOR i := 2 TO n DO
    sum[i] := sum[i − 1] + a[i]
s := ⌈sum[n]/2⌉, l := 1, r := n
REPEAT
    m := ⌊(l + r)/2⌋
    IF sum[m] ≥ s THEN r := m
    ELSE l := m + 1
UNTIL l ≥ r
RETURN m
```

Die for-Schleife wird n mal Durchlaufen.
(Komplexität $cn + d$).
Die repeat-Schleife wird $\log n$ mal Durchlaufen.
(Komplexität $c \log n + d$).
Die Komplexitätsfunktion wird von der for-Schleife dominiert.

Komplexitätsfunktionen

Also

- Die Komplexitätsfunktion wird oft von einem Programmteil dominiert.
- Die Komplexität der Elementaroperationen ist konstant (uniformes Maß).

Für die Beschreibung der Komplexität ignorieren wir daher

- Konstante Faktoren
  Wir identifizieren z.B. $g(n) = 2 \cdot n$ mit $f(n) = n$.
- Terme niedriger Ordnung
  Wir identifizieren z.B. $g(n) = n^2 + 3n$ mit $f(n) = n^2$.

O-Notation

Im Folgenden sind f und g Funktionen von $\mathbb{N}$ nach $\mathbb{R}_+$.

Definition: Mit $O(f(n))$ bezeichnen wir die Klasse der Funktionen g für die es Zahlen c und n_0 gibt, so dass für alle $n \geq n_0$ gilt:

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

In Zeichen:

$$O(f(n)) = \{g(n) \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$$

Beispiel:	$7n + 64 \in O(n)$	mit $c = 8, n_0 = 64$
	$8n^3 \log n \in O(n^4)$	mit $c = 8, n_0 = 2$
	$e^n/2 \notin O(2^n)$	$\forall c \forall n_0 \exists n \geq n_0$ mit $e^n > 2c2^n$

Ω-Notation und Θ-Notation

Definition: Mit $\Omega(f(n))$ bezeichnen wir die Klasse der Funktionen g mit der Eigenschaft:

$$\exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

Beispiel:

$7n + 64 \in \Omega(n)$	mit $c = 1, n_0 = 1$
$8n^3 \log n \notin \Omega(n^4)$	$\forall c \forall n_0 \exists n \geq n_0 : n^4 > c(8n^3 \log n)$
$e^n/2 \in \Omega(2^n)$	mit $c = ?, n_0 = ?$

Definition: Mit $\Theta(f(n))$ bezeichnen wir die Klasse $O(f(n)) \cap \Omega(f(n))$.

Beispiel:

$7n + 64 \in \Theta(n)$

$2 + \sin n \in \Theta(1)$

Vereinfachte Notation

Üblicherweise wird bei asymptotischen Abschätzungen das Gleichheitszeichen $=$ verwendet, obwohl $\in$ oder $\subseteq$ richtig wäre.

$$4n^2 - 2n = 4n^2 - \Theta(n) \;=\; \Theta(n^2)$$

anstelle von

$$4n^2 - 2n \in 4n^2 - \Theta(n) \;\subseteq\; \Theta(n^2)$$

Gleichheitszeichen werden von links nach rechts gelesen.

Erinnerung:
$f(n) \sim g(n)$, genau dann, wenn $\lim_{n\to\infty} f(n)/g(n) = 1$.
Der konstante Faktor, der beiden am stärksten wachsenden Terme muß gleich sein, z.B. $4n^3 + 27n \sim 4n^3 + 3n^2$.

Analyse von Algorithmen: Rekursionsgleichungen

- Wird zur Berechnung einer Funktion f für den Wert n die Funktion f für andere, kleinere Werte verwendet, so entsteht eine Rekursionsgleichung.
- Allgemein hat eine Rekursionsgleichung die Form $f(n) = G(f(1), f(2), \cdots, f(n-1))$.
- Eine Rekursionsgleichung definiert eine Funktionenschar (alle Funktionen, die die Gleichung erfüllen).
- Aus der Funktionenschar wird eine Funktion durch Festlegen weiterer Parameter spezifiziert, z.B. durch Lösen der Anfangswertbedingungen

$$\begin{aligned} f(1) &= a_1 \\ f(2) &= a_2 \\ &\vdots \\ f(k) &= a_k \end{aligned}$$

Rekursionsgleichungen: Beispiel Fibonacci Folge

Die Fibonacci Folge wird definiert durch die Rekursionsgleichung

$$f(n) = f(n-1) + f(n-2), \text{ und } f(1) = f(2) = 1$$

- Offensichtlich ist $f(n)$ größer als $2 \cdot f(n-2)$, d.h., $f(n)$ wächst asymptotisch mindestens mit $2^{n/2} = \sqrt{2}^n$.
- Wir vermuten daher, dass die Funktion exponentielles Wachstum hat.

Behauptung: $f(n) \geq ac^n$ für gewisse Konstanten $a, c > 0$.

Rekursionsgleichungen: Induktive Einsetzungsmethode

Induktives Einsetzen in die Rekursionsgleichung ergibt

$$\begin{aligned} f(n) &= f(n-1) + f(n-2) \\ &\geq ac^{n-1} + ac^{n-2} \\ &= ac^n(\frac{1}{c} + \frac{1}{c^2}) = ac^n \frac{c+1}{c^2} \end{aligned}$$

- Damit der Induktionsschritt korrekt ist, muss $ac^n \frac{c+1}{c^2} \geq ac^n$ gelten, d.h., $c^2 - c - 1 \leq 0$.
- Lösen der quadratischen Gleichung $c^2 - c - 1 = 0$ liefert $c = \frac{\sqrt{5}+1}{2}$ (goldener Schnitt).
- Also gilt der Induktionsschritt für $c \leq \frac{\sqrt{5}+1}{2} \approx 1,6182$.

Rekursionsgleichungen: Beispiel Fibonacci Folge

- Induktionsanfang: Da $c \leq 2$, gilt $f(1) \geq ac^1$ und $f(2) \geq ac^2$ für $a = 1/4$.
- Insgesamt gilt für alle n: $f(n) \geq ac^n$, wobei $a = 1/4$ und $c \leq \frac{\sqrt{5}+1}{2}$.
- Durch Vertauschen von $\leq$ und $\geq$ zeigt man analog, dass $f(n) \leq bd^n$ für eine geeignete Konstanten b und $d \geq \frac{\sqrt{5}+1}{2}$.
- Daher gilt $f(n) = \Theta((\frac{\sqrt{5}+1}{2})^n)$.

Rekursionsgleichungen: Divide and Conquer

Viele Algorithmen zerlegen das zu lösende Problem in 2 oder mehrere Teilprobleme mit einer geringeren Größe.

Die Teilprobleme werden (rekursiv) berechnet und die Lösung aus den Ergebnissen zusammengesetzt.

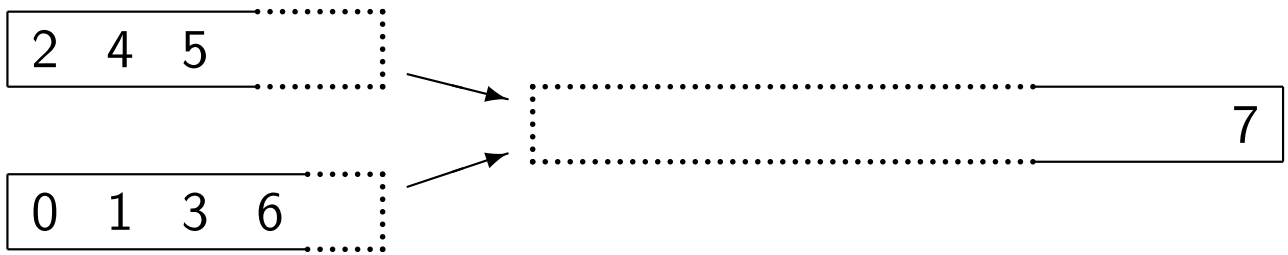
Die Laufzeit für eine Eingabe der Länge n setzt sich also zusammen aus der Laufzeit für die kleineren Eingaben plus der Laufzeit, die benötigt wird, um die Eingabe zu zerlegen und die Lösung zusammenzufügen.

Rekursionsgleichungen: Beispiel MergeSort

- In der rekursiven Formulierung ist MergeSort ein klassisches divide-and-conquer Verfahren.
- Gegeben sei eine Folge von Zahlen $a[1], \ldots, a[n]$. Ein *Sortieralgorithmus* soll diese Folge sortieren; das heißt, dieser soll durch entsprechende Vergleichsoperationen am Ende eine Permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ bestimmen mit $a[\pi(1)] \leq a[\pi(2)] \leq \cdots \leq a[\pi(n)]$.
- MergeSort teilt das zu sortierende Array $a[1..n]$ zunächst in zwei gleich große Hälften auf; diese werden separat durch zwei rekursive Aufrufe sortiert.
- Die beiden sortierten Teilarrays werden dann zu einer sortierten Gesamtfolge gemischt.

Rekursionsgleichungen: MergeSort

- Den Mischvorgang kann man sich (wie bei einem Reißverschluss) wie folgt veranschaulichen.
- Die beiden sortierten Teilfolgen $(2, 4, 5, 7)$ und $(0, 1, 3, 6)$ der Länge $n/2$ wurden bereits zum Teil in ein weiteres Array in sortierter Reihenfolge hineingeschrieben.
- Im nächsten Schritt müsste die Zahl 5 mit der Zahl 6 verglichen werden, und die größere der beiden, die 6, wandert dann in das Ergebnis-Array, usw.



Rekursionsgleichungen: MergeSort

PROCEDURE MergeSort(a)
    IF $|a| = 1$ THEN RETURN a
    ELSE zerlege a in zwei (gleichgroße) Teile a_1 und a_2
        b_1:=MergeSort(a_1)
        b_2:=MergeSort(a_2)
        Mische b_1 und b_2 zur Lösung b
    RETURN b

Der Mischvorgang benötigt im worst case $n-1$ Vergleiche zwischen Elementen. Wir erhalten für die Anzahl der *Vergleiche* die Rekursionsgleichung

$$f(n) = 2f(n/2) + n - 1, \qquad f(1) = 0$$

Statt f wird häufig T (Time) oder V (Vergleiche) verwendet.

Rekursionsgleichungen: Visualisierung der Baumstruktur

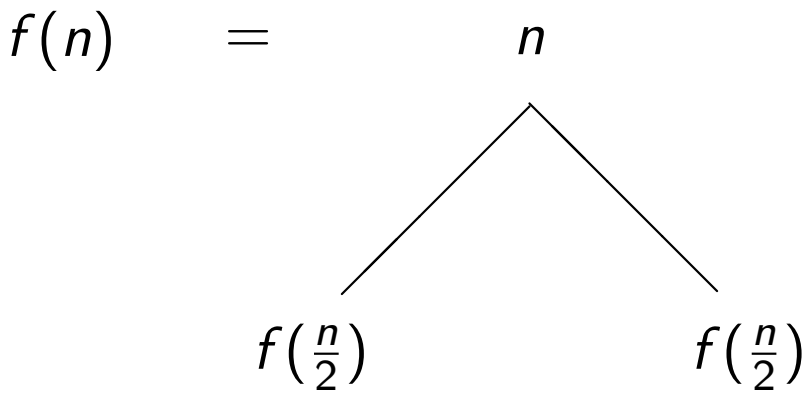
- Die Rekursionsgleichung wird Schritt für Schritt in eine Baumstruktur überführt.
- Die Rekursionsgleichung selbst entspricht einem Baum, der aus einem Wurzelknoten mit einem oder mehreren Kindknoten besteht.
- Die linke Seite der Rekursionsgleichung ist die Wurzel, die Vorkommen der Funktion auf der rechten Seite ergeben die Kindknoten.

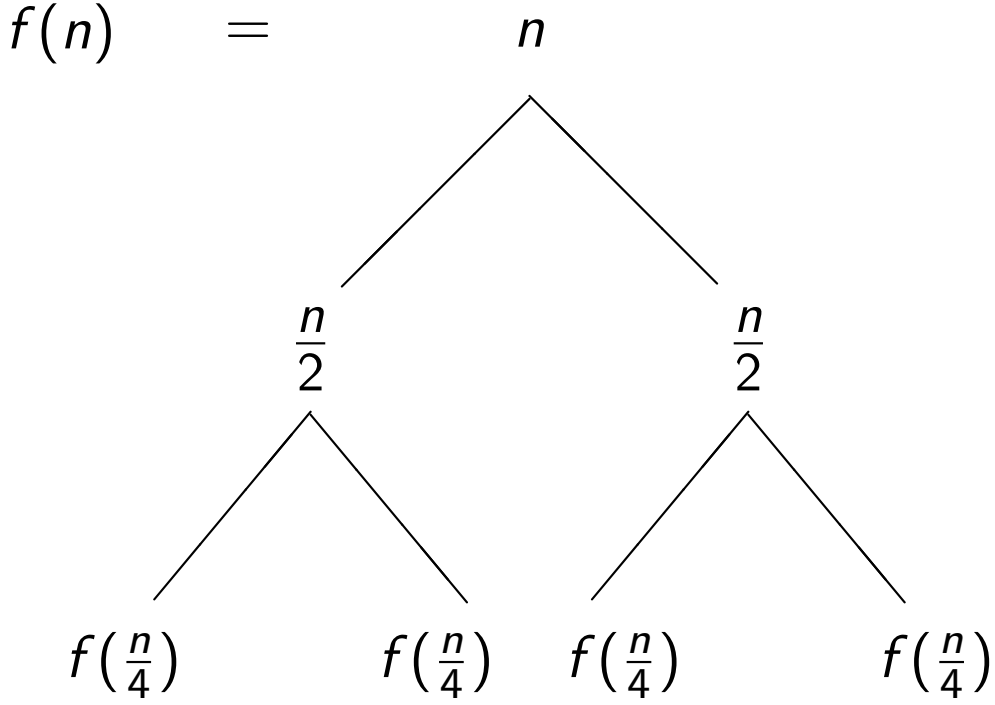
Analyse:

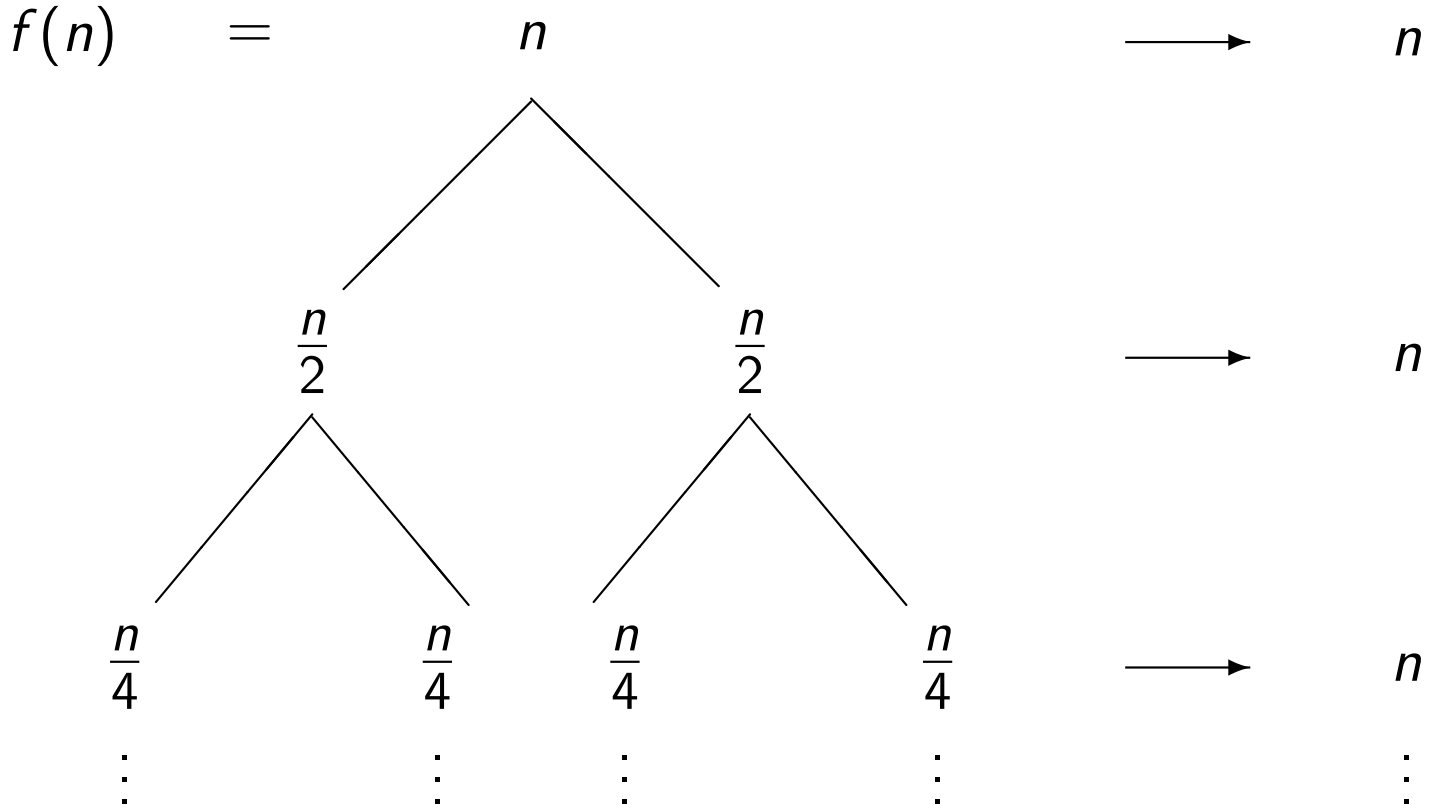
- Die maximale Tiefe des Baumes (Rekursionstiefe) wird abgeschätzt.
- Die Komplexität wird für jede Schicht aufsummiert.
- Die Gesamtkomplexität ergibt sich durch Aufsummieren der Teilsummen.

Rekursionsgleichungen: Visualisierung der Baumstruktur

Beispiel: $f(n) = n + f(n/2) + f(n/2)$




$f(n)$ $=$ n
$\frac{n}{2}$ $\frac{n}{2}$
$f(\frac{n}{4})$ $f(\frac{n}{4})$ $f(\frac{n}{4})$ $f(\frac{n}{4})$



- Komplexität in der j-ten Schicht: n
- Die Baumtiefe ist $O(\log n)$.
- Daher ist $f(n) = O(n \log n)$.

Rekursionsgleichungen: Divide and Conquer

PROCEDURE DivideAndConquer(x:Eingabe)
    IF $|x| \leq 1$ THEN
        berechne Lösung l für x direkt
    ELSE zerlege x in zwei (gleichgroße) Teile x_1 und x_2
        l_1:=DivideAndConquer(x_1)
        l_2:=DivideAndConquer(x_2)
        berechne Lösung l für x aus l_1 und l_2
RETURN l

Der Aufwand für das Zerlegen und Zusammenfügen sei $cn + d$. Die Komplexität ergibt sich damit zu:

$$f(n) = 2f(n/2) + cn + d.$$

Rekursionsgleichungen: Induktive Einsetzungsmethode

- Wir versuchen wieder eine Lösung zu der Rekursionsgleichung zu finden und diese zu verifizieren.
- Bei jedem Aufruf halbiert sich die Problemgröße.
- Nach $\log n$ vielen Schritten wird die Größe 1 und damit das Ende der Rekursion erreicht.
- Wir vermuten daher eine Lösung der Form

$$f(n) = \alpha + \beta n + \delta n \log n$$

Rekursionsgleichungen: Induktive Einsetzungsmethode

Die vermutete Lösung $\alpha + \beta n + \delta n \log n$ setzen wir induktiv ein:

$$\begin{aligned} f(n) &= cn + d + 2f(n/2) \\ &= cn + d + 2(\alpha + \beta(n/2) + \delta(n/2)\log(n/2)) \\ &= cn + d + 2\alpha + \beta n + \delta n \log n - \delta n \end{aligned}$$

Durch Koeffizientenvergleich mit der vermuteten Lösung ergibt sich $\alpha = -d$ und $\delta = c$. Dies ergibt Funktionen der Form

$$\{-d + \beta n + cn \log n \mid \beta \in \mathbb{R}\}$$

Der Wert von β ergibt sich aus der Anfangswertbedingung, z.B. für MergeSort: $f(1) = 0$. Aus $-d + \beta + c \log 1 = 0$ folgt $\beta = d$.
Für MergeSort gilt $c = 1$, $d = -1$ und $\beta = -1$.
Die Lösungsfunktion für MergeSort lautet $f(n) = n \log n - n + 1$.

Iterationsmethode

Bei der Iterationsmethode wird die Funktion auf der rechten Seite immer wieder durch die Rekursionsgleichung ersetzt. Dann wird versucht, die rechte Seite in eine geschlossene Form zu bringen.

Beispiel Rekursionsgleichung MergeSort:

$$f(n) = 2f(n/2) + n - 1, \qquad f(1) = 0$$

Erinnerung: k-te Partialsumme für $q \neq 1$ ist

$$\sum_{i=0}^{k} q^i = \frac{1 - q^{k+1}}{1 - q}$$

Iterationsmethode: Beispiel MergeSort

Wir setzen iterativ ein:

$$
\begin{aligned}
f(n) &= 2f(n/2) + n - 1 \\
&= 2(2f(n/4) + n/2 - 1) + n - 1 \\
&= 4f(n/4) + n - 2 + n - 1 \\
&= 8f(n/8) + n - 4 + n - 2 + n - 1 \\
&= \ldots \\
&= n\log_2 n - \sum_{i=0}^{(\log_2 n)-1} 2^i \\
&= n\log_2 n - \frac{1 - 2^{(\log_2 n)-1+1}}{1-2} \\
&= n\log_2 n + 1 - 2^{\log_2 n} \\
&= n\log_2 n + 1 - n
\end{aligned}
$$

Iterationsmethode: Beispiel $f(n) = n + 3f(n/4)$

Iteriertes Einsetzen und Ersetzen der Summe ergibt:

$$\begin{aligned} f(n) &= n + 3f(n/4) \\ &= n + 3(n/4 + 3f(n/16)) \\ &= n + 3n/4 + 9(n/16 + 3f(n/64)) \\ &\leq n + 3n/4 + 9n/16 + \cdots \\ &= n \cdot \sum_{i=0}^{\infty} (3/4)^i \quad = \quad n \cdot \frac{1}{1 - 3/4} \quad = \quad 4n = O(n) \end{aligned}$$

(Geometrische Reihe: $\sum_{i=0}^{\infty} c^i = \frac{1}{(1-c)}$ für $|c| < 1$.)

Daher ist $f(n) = \Theta(n)$. Die Anfangswertbedingung $f(1) = a$ wirkt sich nur auf die in der Θ-Notation verborgenen Konstanten aus.

Rekursionsgleichungen: Variablensubstitution

Einige Rekursionsgleichungen lassen sich mit den bisherigen Methoden nicht (einfach) lösen.

Dies tritt zum Beispiel ein, wenn die Argumente der Funktion f (auf der rechten Seite) nicht die Form $n/2$ bzw. n/c für eine Konstante c haben.

Beispiel: $f(n) = 2f(\sqrt{n}) + \log n$.

Diese Gleichungen lassen sich oft durch Variablensubstitution auf die vertraute Form bringen.

Variablensubstitution für $f(n) = 2f(\sqrt{n}) + \log n$

Führen wir eine neue Variable m mit $m = \log n$ bzw. $n = 2^m$ ein, so ergibt sich:

$$f(2^m) = 2f(\sqrt{2^m}) + \log 2^m = 2f(2^{m/2}) + m.$$

Ersetzen von $f(2^m)$ durch $g(m)$ ergibt die Rekursionsgleichung:

$$g(m) = 2g(m/2) + m.$$

Diese Gleichung können wir z.B. mit der Iterationsmethode lösen und erhalten $g(m) = \Theta(m \log m)$.

Ersetzen von g durch f und m durch $\log n$ ergibt $f(2^m) = \Theta(m \log m)$ und damit $f(n) = \Theta(\log n \cdot \log \log n)$.

Rekursionsgleichungen: Vereinfachungen

Die Analyse von Algorithmen wird durch Vereinfachungen und Vernachlässigungen bei den Rekursionsgleichungen erleichtert.

- Nicht-Ganzzahligkeit
  - Die Rekursionsgleichung lautet zum Beispiel

$$f(n) = f(\lceil n/2 \rceil) + f(\lfloor n/2 \rfloor) + n$$

  - Das Erzwingen der Ganzzahligkeit des Argumentes wird ignoriert.
  - Die Gleichung $f(n) = 2f(n/2) + n$ wird analysiert.
  - Die Änderung wirkt sich asymptotisch nicht aus.

Rekursionsgleichungen: Vereinfachungen

- Ignorieren der Anfangswertbedingung
  Diese legen meist nur die konstanten Faktoren fest, die durch die asymptotischen Notationen ignoriert (versteckt) werden.
- Obere und untere Schranken
  Oft interessiert nur eine Abschätzung (nach oben oder unten). Daher kann es einfacher sein zu zeigen, dass die Komplexität $\leq f(n)$, als die exakte Funktion $g(n)$ zu bestimmen.
- Abschätzungen gelten nur für große n
  Zum Beispiel wird $7 + 2\lceil \ln n \rceil < n$ verwendet, ohne darauf hinzuweisen, dass die Abschätzung erst für $n \geq 14$ gilt.

Rekursionsgleichungen: Vereinfachungen

- Asymptotische Notation bereits in der Rekursionsgleichung
  Hier ist Vorsicht angebracht. Verwenden wir zum Beispiel die Iterationsmethode zur Abschätzung der Funktion
  $f(n) = 7n + 23 + f(n/2) = \Theta(n) + f(n/2)$, so ergibt sich:

$$\begin{aligned} f(n) &= \Theta(n) + f(n/2) \\ &= \Theta(n) + \Theta(n/2) + f(n/4) \\ &\vdots \\ &= \Theta(n) + \Theta(n/2) + \Theta(n/4) + \cdots \end{aligned}$$

  Hier muß man beachten, dass die in den Θ-Notationen versteckten Konstanten gleich sind.
  (Die Komplexität von $f(n)$ ist $\Theta(n)$ und nicht $\Theta(n \log n)$).

Master Theorem

Satz: Gegeben sei eine Rekursionsgleichung der Form

$$T(n) = \sum_{i=1}^{m} T(\alpha_i \cdot n) + \Theta(n^k)$$

wobei $0 \leq \alpha_i \leq 1, m \geq 1, k \geq 0$.
Dann kann $T(n)$ asymptotisch abgeschätzt werden durch:

$$T(n) = \begin{cases} \Theta(n^k), & \text{falls } \sum_{i=1}^{m} \alpha_i^k < 1 \\ \Theta(n^k \log n), & \text{falls } \sum_{i=1}^{m} \alpha_i^k = 1 \\ \Theta(n^c), & \text{falls } \sum_{i=1}^{m} \alpha_i^k > 1, \end{cases}$$

hierbei ergibt sich c durch Lösen der Gleichung $\sum_{i=1}^{m} \alpha_i^c = 1$. Für den Spezialfall, dass alle $\alpha_i = \alpha$ für ein α, gilt $c = -\ln m / \ln \alpha$.

Beweis z.B. im Buch Algorithmik von Uwe Schöning (2001).

Beispiele Master Theorem

$$U(n) = 8U(n/3) + n^2 \qquad \alpha = 1/3, m = 8, k = 2$$
$$V(n) = 9V(n/3) + n^2 \qquad \alpha = 1/3, m = 9, k = 2$$
$$W(n) = 10W(n/3) + n^2 \qquad \alpha = 1/3, m = 10, k = 2$$

Mit dem Master Theorem ($\sum_{i=1}^{m} \alpha_i^k$ betrachten) ergibt sich

$$U(n) = \Theta(n^2) \qquad \sum_{i=1}^{8} (1/3)^2 < 1$$

$$V(n) = \Theta(n^2 \log n) \qquad \sum_{i=1}^{9} (1/3)^2 = 1$$

$$W(n) = \Theta(n^c) \qquad \text{mit } c = -\frac{\ln 10}{\ln(1/3)} = \log_3(10) \approx 2.096.$$