Aufwand

- begrenzender Schlüsselfaktor
- es macht kaum einen Unterschied, ob eine Berechnung zu lange dauert oder gar nicht endet
- gesucht sind Methoden zur Aufwandsermittlung und -analyse
- Versuch in CE-S:
  Gleichungsanwendungen brauchen Zeit

wesentliche Elemente zur Ermittlung des Aufwands von Algorithmen

- Modellierung von Algorithmen
- einschließlich ihrer Berechnung (Aufführung)
- quantitative Erfassung als Zahl der Berechnungsschritte
- Nachweisbarkeit dafür erforderlicher Eigenschaften
- geeigneter Ansatz: CE-S

CE-S-Ansatz zur Aufwandsermittlung

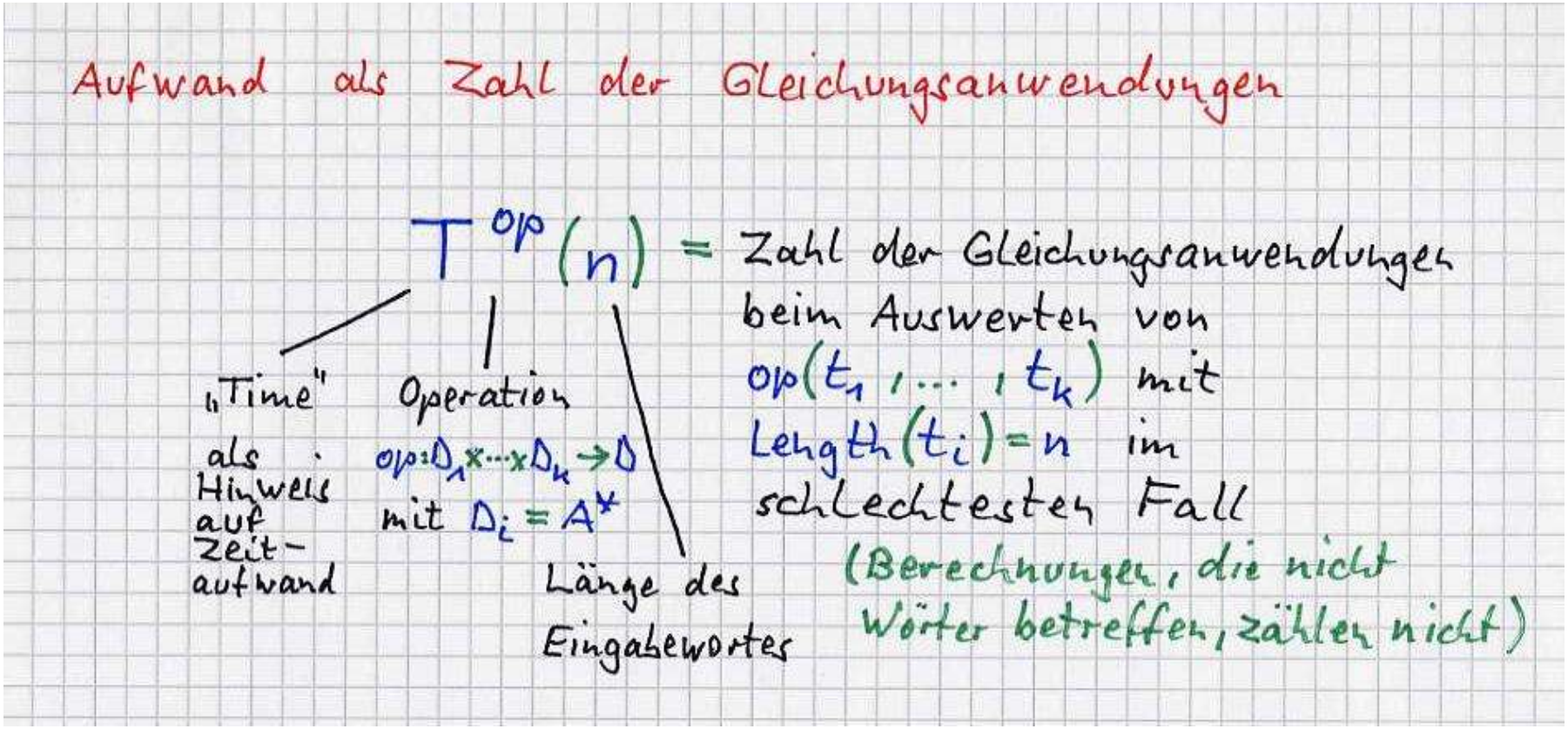
Zeitaufwand als maximale Zahl von Gleichungsanwendungen, die nötig sind, um eine Operation für bestimmte Argumente auszuwerten in Abhängigkeit von der Länge der eingegeben Zeichenketten

in Zeichen: $T^{op}(n)$

bzw. $T^{op}(n_1, \ldots, n_k)$

k: Zahl der Zeichenkettenargumente

Variante von Folie 3



Sortiertheitstest

is-sorted

opns: $\text{is-sorted} : A^* \to \text{BOOL}$

vars: $x, y \in A$, $u \in A^*$

eqns: $\text{is-sorted}(\lambda) = T$

$\text{is-sorted}(x) = T$

$\text{is-sorted}(xyu) = (x \equiv y) \wedge \text{is-sorted}(yu)$

Beobachtungen: $T^{\text{is-sorted}}(0) = T^{\text{is-sorted}}(1) = 1$ und

$T^{\text{is-sorted}}(n+2) = 1 + T^{\text{is-sorted}}(n+1)$ für $n \geq 0$

Theorem: $T^{\text{is-sorted}}(k) = k$ für alle $k \geq 1$

Beweis (mit vollständiger Induktion über k):

I.A.: $k = 1$: $T^{\text{is-sorted}}(1) = 1$ (nach Beob.)

I.V.: Beh. gelte für ein $k \geq 1$

I.S.: $k+1$: $T^{\text{is-sorted}}(k+1) \underset{n=k-1}{=} T^{\text{is-sorted}}(n+2) \underset{\text{Beob.}}{=} 1 + T^{\text{is-sorted}}(n+1)$

$\underset{\text{I.V.}}{=} 1 + n + 1 \underset{n=k-1}{=} k + 1$

Sortieren durch Einsortieren

sort

opns: $sort : A^* \to A^*$, $insort : A \times A^* \to A^*$

vars: $x, y \in A$, $u \in A^*$

eqns: $sort(\lambda) = \lambda$

$sort(x\,u) = insort(x, sort(u))$

$insort(x, \lambda) = x$

$insort(x, y\,u) = \text{if } x \leq y \text{ then } x\,y\,u \text{ else } y\,insort(x, u)$

Beobachtungen: $T^{sort}(0) = 1$ und

$T^{sort}(n+1) = 1 + T^{sort}(n) + T^{insort}(m)$

$= n+2 + T^{sort}(n)$

Voraussetzungen:

Länge von $sort(u)$

(1) $Length(sort(u)) = Length(u)$

(2) $T^{insort}(n) = n+1$

Vorgehensweise (1)

- Auswirkung der spezifizierten Gleichungen auf die Aufwandsfunktion

z.B.

$insort(x, \lambda) = x \rightsquigarrow T^{insort}(0) = 1$

$insort(x, y\,v) = \text{if } x \leq y \text{ then } x\,y\,v \text{ else } y\,insort(x, v)$

$\rightsquigarrow T^{insort}(n+1) \overset{?}{=} 1 + \begin{cases} 0 & \text{falls } x \leq y \\ T^{insort}(n) & \text{sonst} \end{cases}$

$= 1 + T^{insort}(n)$

↑ schlechteste Fall

Vorgehensweise (2)

- Aufstellen einer nichtrekursiven Aufwandshypothese (Beweis meist durch vollständige Induktion)

  z.B.

  $T^{insort}(n) = n+1$ für alle $n \in \mathbb{N}$

- statt einer exakten Aufwandsaussage kann auch nach oben abgeschätzt werden (was oft einfacher ist)

Vorgehensweise (3)

- beim Beweis von Aufwandsabschätzungen werden oft Eigenschaften der Operationen benötigt (Beweis mit Hilfe von Gleichwertigkeit oft mit vollständiger Induktion über Wörter)

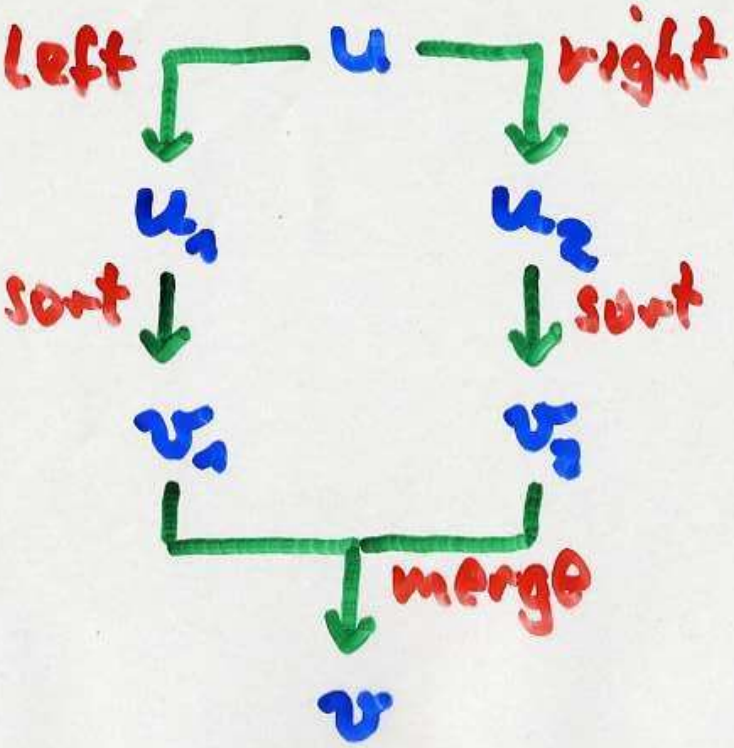
z.B. $T^{sort}(n) = \frac{(n+2)(n+1)}{2}$

falls $T^{insort}(n) = n+1$

und $length(sort(u)) = length(u)$

Beispiel: Sortieren durch Mischen

Idee: sortiere linke und rechte Hälfte eines Wortes und mische die sortierten Hälften zusammen



Sortieren durch Mischen

mergesort

ops: $sort : A^* \to A^*$; $left, right : A^* \to A^*$; $merge : A^* \times A^* \to A^*$

vars: $x, y \in A$; $u, v \in A^*$

eqns:

$$sort(\lambda) = \lambda$$

$$sort(x) = x$$

$$sort(u) = merge(sort(left(u)), sort(right(u))) \quad \text{falls } length(u) \geq 2$$

$$left(\lambda) = \lambda \qquad right(\lambda) = \lambda$$

$$left(x) = x \qquad right(x) = \lambda$$

$$left(xuy) = x\, left(u) \qquad right(xuy) = right(u)\, y$$

$$merge(\lambda, v) = v$$

$$merge(u, \lambda) = u$$

$$merge(xu, yv) = \text{if } x \leq y \text{ then } x\, merge(u, yv) \text{ else } y\, merge(xu, v)$$

Aufwandsermittlung (für Sortieren durch Mischen)

(1) was die spezifizierten Gleichungen sagen:

$$T^{sort}(0) = 1$$

$$T^{sort}(1) = 1$$

$$T^{sort}(n) = 1 + T^{merge}(\bar{n}_1, \bar{n}_2) + T^{left}(n) + T^{right}(n) + T^{sort}(n_1) + T^{sort}(n_2)$$

$\bar{n}_1$ = Length(sort(left(u)))

$\bar{n}_2$ = Length(sort(right(u)))

n_1 = Length(left(u))

n_2 = Length(right(u))

Aufwandsermittlung (für Sortieren durch Mischen)

(1) was die spezifizierten Gleichungen sagen:

$$T^{sort}(0) = 1$$
$$T^{sort}(1) = 1$$
$$T^{sort}(n) = 1 + T^{merge}(\bar{n}_1, \bar{n}_2) + T^{left}(n) + T^{right}(n) + T^{sort}(n_1) + T^{sort}(n_2)$$

$$length(sort(left(u))) = length(left(u))$$
$$length(sort(right(u))) = length(right(u))$$

(3) Eigenschaften der Operationen ausnutzen (Beob. 3 in Kap. 3)

$$T^{sort}(2m) = 1 + T^{merge}(m,m) + T^{left}(n) + T^{right}(n) + 2 \cdot T^{sort}(m)$$

1. Fall: $n = 2m$

$$= 1 + 2m + m + 1 + m + 1 + 2 \cdot T^{sort}(m)$$
$$= 3 + 4m + 2T^{sort}(m) \quad \text{(vgl. Bsp. 4.4 im Skript)}$$

$$T^{sort}(2m+1) = 1 + T^{merge}(m+1,m) + T^{left}(n) + T^{right}(n) + T^{sort}(m+1) + T^{sort}(m)$$

2. Fall: $n = 2m+1$

$$= 1 + 2m + 1 + m + 1 + m + 1 + T^{sort}(m+1) + T^{sort}(m)$$
$$= 4 + 4m + T^{sort}(m+1) + T^{sort}(m) \quad \text{(vgl. 4.4)}$$

Aufwandsermittlung (fortgesetzt)

(2) nichtrekursive Abschätzung

Vermutung: $T^{sort}(n) \leq 5 \cdot n \cdot k$ mit $2^{k-1} < n \leq 2^k$ $(k \geq 1)$

Beweis durch vollständige Induktion über k:

IA: $k=1$, d.h. $n=2$ und somit nach Fall 1:

$$T^{sort}(2) = 3 + 4 \cdot 1 + 2 \cdot 1 = 9 \leq 5 \cdot 2 \cdot 1$$

IS: $k+1$, d.h. $2^k < n \leq 2^{k+1}$

2. Fall: $n = 2m+1$, d.h. $2^{k-1} \leq m < m+1 \leq 2^k$ und somit

$$\begin{aligned} T^{sort}(2m+1) &= 4 + 4m + T^{sort}(m+1) + T^{sort}(m) \\ &\overset{IV}{\leq} 4 + 4 \cdot m + 5 \cdot (m+1) \cdot k + 5 \cdot m \cdot k \\ &= 4 + 4m + 5(2m+1)k \\ &\leq 5(2m+1) + 5(2m+1)k = 5(2m+1)(k+1) \end{aligned}$$

1. Fall analog (vgl. 4.4. im Skript)

unendlicher Aufwand

$sort_0$

opns: $sort : A^* \to A^*$

vars: $x, y \in A$; $u, v \in A^*$; $w \in A^*$

eqns: $sort(uxyv) = sort(uyxv)$

$sort(w) = w$ falls $is\text{-}sorted(w)$

Aufwand gemäß 1. Gleichung:

$$T^{sort}(n) = 1 + T^{sort}(n) \text{ für } n \geq 2$$

aber: diese Gleichung besitzt in $\mathbb{N}$ keine Lösung

das große „O" (1)

- Aufwandsklasse $O(g)$ für $g : \mathbb{N} \to \mathbb{N}$ enthält alle Funktionen $f : \mathbb{N} \to \mathbb{N}$ mit

$$f(n) \leq \underbrace{c}_{\text{konstant}} \cdot g(n) \quad \text{für alle} \quad n \geq \underbrace{n_0}_{\text{konstant}}$$

- $O(g)$ beschreibt alle Datenverarbeitungsprobleme, die eine algorithmische Lösung op besitzen mit $T^{op} \in O(g)$
- obere Schranke, bei der konstante Faktoren und kleine Eingaben nicht zählen

das große „O" (2)

- Idee: wähle g einfach & einprägsam

  z.B. $\log n$, n, $n \cdot \log n$, n^2, n^3, ..., 2^n, ...

  logarithmisch / linear / quadratisch \ kubisch / exponentiell

- $T^{mergesort} \in O(n \cdot ld\, n)$ $T^{quicksort} \in O(n^2)$
- $T^{xmpl}(n) \leq 1/3\, n\, ld\, n + 22n + 1/2\, n^2 + 7$

  d.h. $T^{xmpl} \in O(n^2)$

  bzw. xmpl hat quadratischen Aufwand

Beob.: $O(\log n) \subsetneq O(n) \subsetneq O(n^2) \subsetneq O(n^3) \subsetneq \cdots \subsetneq O(2^n)$

(vgl. Übung)

polynomieller Aufwand: die Klasse P

- Probleme in P können mit herkömmlicher Rechnertechnik in verfügbarer Zeit gelöst werden (wenn der Exponent nicht zu groß wird)