

Vorlesungsplanung

Diese Woche = Woche n

Woche n+1 : 1 x VL (Do) durch Martino

Woche n+2 : 2 x VL Mi, Do

Woche n+3 : 3 x VL

Woche n+4 : 2 x VL (vermutlich)

Woche n+5 : Keine VL, Ende des Comq,

Woche n+6 : Semesterferien

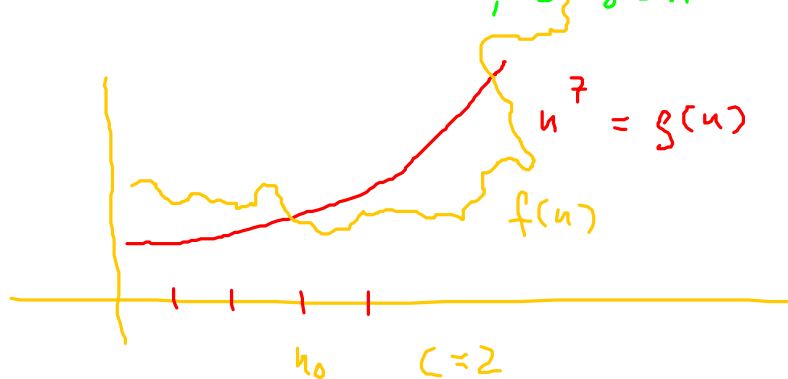
} Rücksprechen

§9 Analyse von Algorithmen

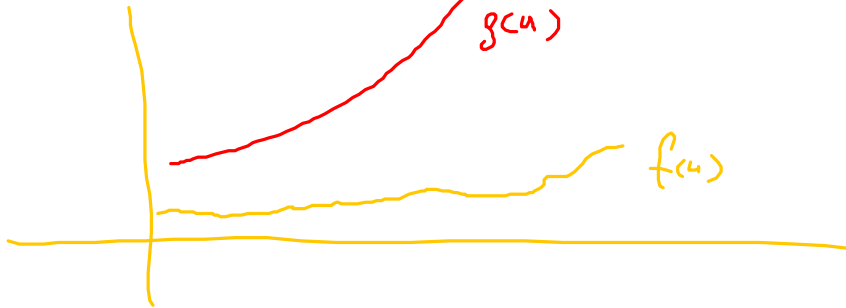
asymptotische Notation

Laufzeit fkt. $f, g: \mathbb{N} \rightarrow \mathbb{N}$

$$f \in O(g) \stackrel{\text{Def. } O(g)}{\iff} \exists c > 0, \exists n_0 \in \mathbb{N} : f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$



$$f \in O(g) \quad [f \text{ ist von echt kleinerer GröÙenordnung als } g]$$
$$\stackrel{\text{Def. } O(g)}{\iff} \forall c > 0 \exists n_0 \in \mathbb{N} \quad f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$



SATZ

$$\underline{f \in O(g)} \Leftrightarrow \lim_{u \rightarrow \infty} \frac{f(u)}{g(u)} = 0$$

gängige Größenordnungen

$$O(1) < O(\log u) < O(u) < O(u^2) < \dots < O(u^k) \quad k \text{ fest}$$

$$< O(u \log u) < \boxed{O(2^u)}$$

exponentiell

polynomial
effizient

wie dieses Kriterium überprüfen?

Regel von L'Hopital

$$\lim_{u \rightarrow \infty} \frac{f(u)}{g(u)} = \lim_{u \rightarrow \infty} \frac{\frac{d}{du} f(u)}{\frac{d}{du} g(u)} \quad \leftarrow \text{Ableitungen}$$

gilt falls der Limes der rechten Seite existiert

Beispiel 1. $\log n \in o(n)$

$$\lim_{n \rightarrow \infty} \frac{\log n}{n} = \lim_{n \rightarrow \infty} \frac{\frac{1}{\ln 2} \cdot \frac{1}{n}}{1} = \lim_{n \rightarrow \infty} \frac{1}{\ln 2 \cdot n} = 0$$

\uparrow
 $\log n = \frac{\ln n}{\ln 2}$

2. $n^k \in o(2^n)$ k fest

$$\lim_{n \rightarrow \infty} \frac{n^k}{2^n} = \lim_{n \rightarrow \infty} \frac{k \cdot n^{k-1}}{\ln 2 \cdot 2^n} = \lim_{n \rightarrow \infty} \frac{k(k-1) \cdot n^{k-2}}{(\ln 2)^2 \cdot 2^n}$$

\uparrow
 $2^n = e^{\ln 2 \cdot n}$

$$= \dots = \lim_{n \rightarrow \infty} \frac{k(k-1) \dots 2 \cdot 1}{(\ln 2)^k \cdot 2^n} \rightarrow 0$$

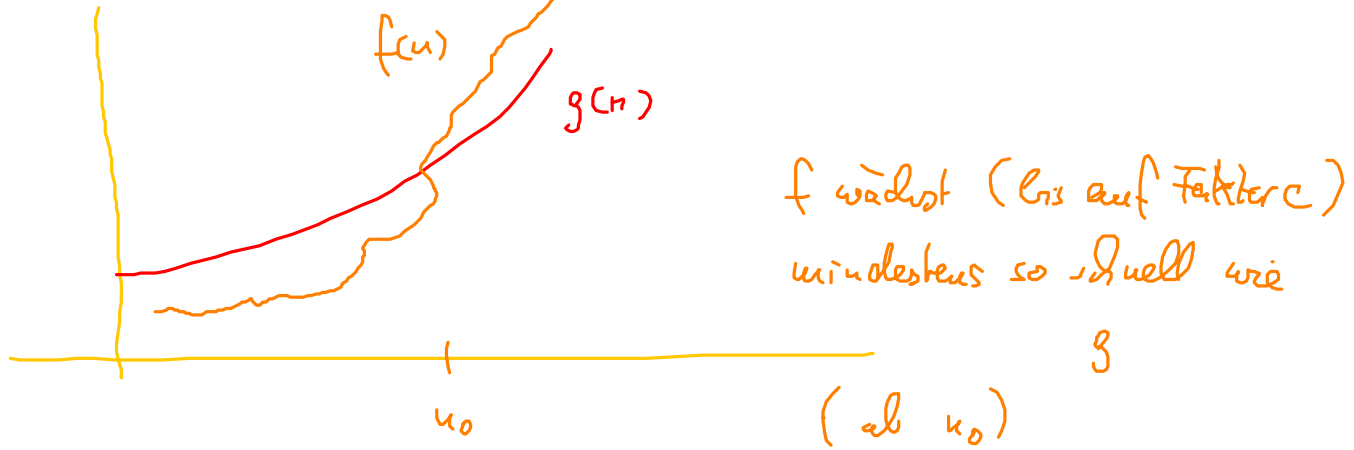
9.2.2 Untere Schranken

$f \in O(g)$ g ist obere Schranke

$f \in o(g)$ f ist unecht kleinerer Größenordnung

$\Omega(g) := \{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c > 0, \exists n_0 \in \mathbb{N} :$

$f(n) \geq c \cdot g(n) \forall n \geq n_0 \}$



$f \in O(g)$, $f \in \Omega(g)$ \leftarrow Schreibweise $f \in \Theta(g)$

$$\Theta(g) = \left\{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c_1 \exists c_2 \exists n_0 : \right. \\ \left. \underline{c_1 \cdot g(n)} \leq \underline{f(n)} \leq c_2 g(n) \quad \forall n \geq n_0 \right\}$$

Bsp 1: Sequenzielle Suche in Array



worst case Aufwand

$C(n) :=$ maximale # Vergleiche, die nötig ist
um festzustellen ob ein Wert im Array
vorkommt und ggf den Index zurückzugeben

n Vergleiche nötig $\Rightarrow C(n) \in O(n)$
weniger als n gehen nicht $\Rightarrow C(n) \in \Omega(n)$ } $C(n) \in \Theta(n)$

"Sequentielle Suche in Array mit n Komponenten erfordert im Worst Case $\Theta(n)$ Vergleiche"

Bsp 2: Matrizenmult. von $n \times n$ Matrizen A, B $A \cdot B = C$

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj} \quad \leftarrow n \text{ Mult.}, n-1 \text{ Add.}$$

$$\uparrow$$
$$n^2 \text{ Werte } c_{ij} \quad \Rightarrow n^2(2n-1) \text{ arithm. Operationen}$$

$$= 2n^3 - n^2$$

$$\stackrel{\text{SATZ}}{=} \mathcal{O}(n^3)$$

müssen n^2 Werte c_{ij} berechnen $\Omega(n^2)$ Werte Operationen

Komplexitätslücke zwischen $\Omega(n^2)$ und $\mathcal{O}(n^3)$

Es gibt andere Verfahren mit $\mathcal{O}(n^{2,376})$ arithm. Operationen

beruhen auf Aufteilung + Rekurrenz

Kap 10 Sortieren in Arrays

Klasse Item (Student)

mit Feld key \leftarrow vom Typ int
(Matrikelnummer)

Array

Item[] vec = new Item[n]

↑

aufsteigend sortieren, d.h.

$$\text{vec}[0].\text{key} \leq \text{vec}[1].\text{key} \leq \dots \leq \text{vec}[n-1].\text{key}$$

10.1 Direkte Methoden

↑ "Sortieren am Ort"

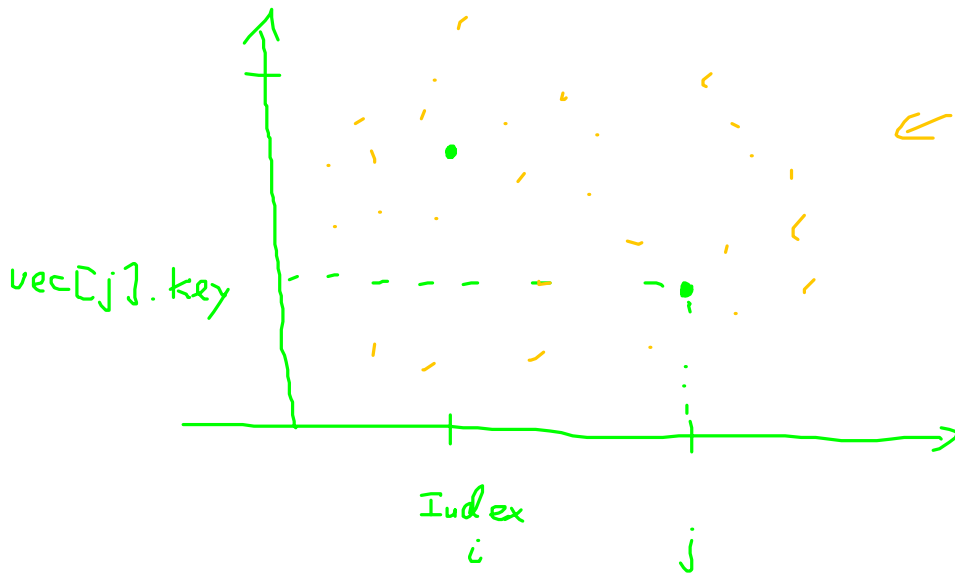
10.1.1 Bubblesort

1. Geg. ist $\text{vec}[]$ mit n Komponenten
2. vec wird $n-1$ mal durchlaufen von hinten nach vorn.
Ein Durchlauf heißt Phase.
3. Phase i läuft von Komponente $j = n-1$ bis $j = i$
und vergleicht jeweils $\text{vec}[j-1].\text{key}$ und $\text{vec}[j].\text{key}$
Ist $\text{vec}[j-1].\text{key} > \text{vec}[j].\text{key}$ so werden

vec[j-1] und vec[j] vertauscht

j	vec[j].key	Phase i=1	i=2	i=3	i=4	i=5	i=6	i=7
0	63	<u>12</u>	12	12	12	12	12	12
1	24	63	<u>18</u>	18	18	18	18	18
2	12	24	63	<u>24</u>	24	24	24	24
3	53	18	24	63	<u>35</u>	35	35	35
4	72	53	35	35	63	<u>44</u>	44	44
5	18	72	53	44	44	63	<u>53</u>	53
6	44	35	72	53	53	53	63	<u>63</u>
7	35	44	44	72	72	72	72	<u>72</u>

Visualisierung



← unsortiert

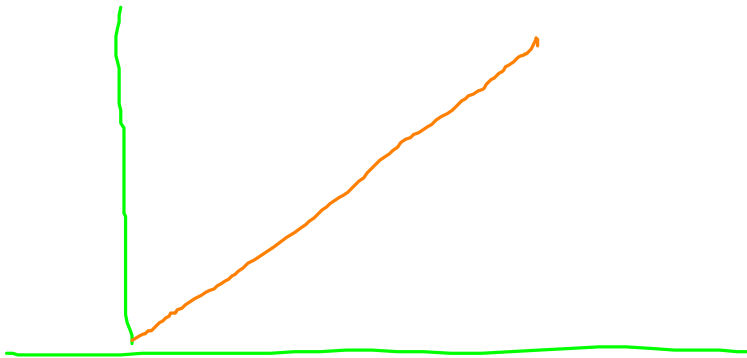
= Punktwolke

alle Werte auf x Achse

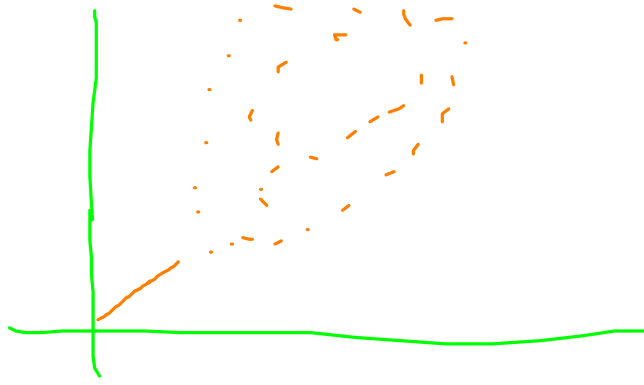
kommen genau

einmal vor

sortiertes
Array



Zurücksende mitte



Korrektheit:

Schleifeninvariante nach Ende von Phase i

$vec[0].key, \dots, vec[i-1].key$ sind richtig
sortiert, und es sind die i kleinsten
Elemente im Array

Beweis mit Induktion nach # Phasen

Phase 1 (Ind. Anfang) kleinste Element kommt nach oben

Ind. Ann: Beh. sei richtig für Phase 1, 2, ..., i

Schluss auf Phase $i+1$:

Phase i

kleinsten,
nicht
sortiert

$\left\{ \begin{array}{l} 0 \\ \vdots \\ i-1 \end{array} \right.$



} nur hier Vergleiche
in Phase $i+1$

\Rightarrow kleinste von diesen
geht nach oben



\leftarrow $\{i+1\}$ -kleinste \Rightarrow 2 Referenzvariable

Aufwand: (worst case, # Vergleiche)

Phase 1	$n-1$	Vergleiche
2	$n-2$	
\vdots		
i	$n-i$	
\vdots		
$n-1$	1	

$$\sum_{i=1}^{n-1} (n-i)$$

$$= 1 + 2 + \dots + n-1$$

$$= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

$$\in \mathcal{O}(n^2)$$

Branden immer so viele

bei jedem Lauf

$$\Rightarrow (C_n) \in \Omega(n^2)$$

$$\Rightarrow (C_n) \in \Theta(n^2)$$

Zeitschranken $A(n)$ (worst case)

A = assignment

1 Vertauschung = 3 Zuweisungen

Phase 1 $n-1$ Vertauschungen

2 $n-2$

\vdots

$A(n) \in \Theta(n^2)$

mögliche Verbesserungen:

abbrechen falls keine Vertauschung in einer Phase

Shakersort (abwechselnd von links nach rechts und von rechts nach links)

$\in O(n^2)$

Beispiele zeigen $\Omega(n^2)$ Vergleiche, Zuweisungen

$\Rightarrow C(n), A(n) \in \Theta(n^2)$ auch für diese Varianten

10.1.2 Selectran Sort

1 Gegeben ist $vec[]$ mit n Komponenten

2. vec wird $n-1$ mal von links nach rechts durchlaufen
Ein Durchlauf heißt Phase

3. Phase bottom sucht Index $minIndex$ einer Komponente

mit kleinstem Schlüssel in $vec[bottom \dots n-1]$

mittels sequenzielles Suchen, und tauscht $vec[bottom]$
und $vec[miniIndex]$

63 24 12 53 72 18 44 35
↑ ↑

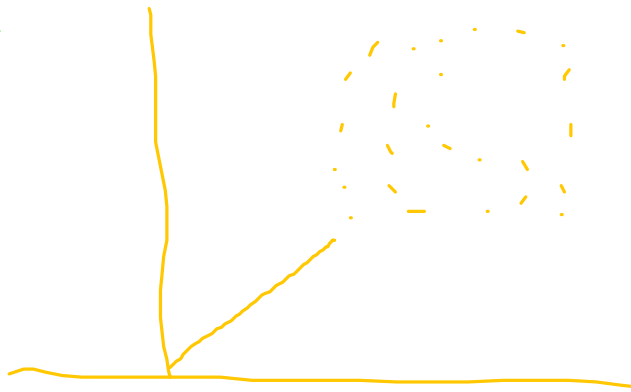
12 : 24 63 53 72 18 44 35
↑ ↑
kleinste bottom miniIndex

12 18 : 63 53 72 24 44 35

usw

Korrektheit wie bei Bubblesort mit derselben Schleifeninvariante

Visualisierung:



Aufwand: $C(n)$

Phase	1	$n-1$	=	$\Theta(n^2)$	wie Bubblesort
	2	$n-2$			
	$n-1$	1			

Zuweisungen $A(n)$ ^{von Arraykomponenten} nur 1 Verknüpfung pro Phase

$$\Rightarrow A(n) \in \Theta(n)$$

↑
besser als Bubblesort