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 der Comp, } Rücksprachen

Woche n+6 : Semesterferien

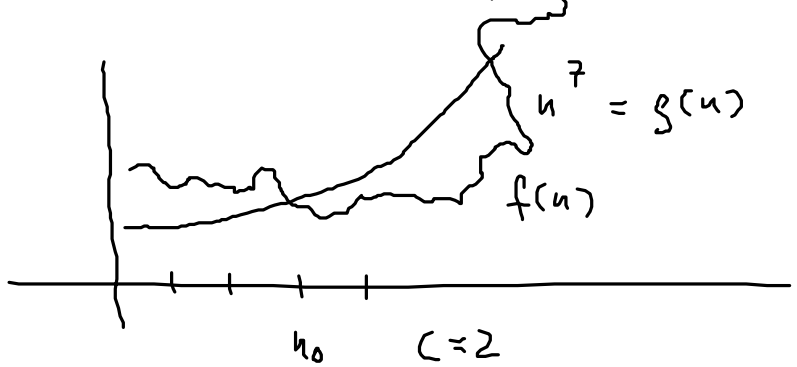
---

§9 Analyse von Algorithmen

asymptotische Notation

Laufzeitfkt. $f, g : \mathbb{N} \to \mathbb{N}$

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



$f \in o(g)$ [f ist von echt kleinerer Größenordnung als g]

$\overset{\text{Def } o(g)}{\iff} \forall c > 0 \quad \exists n_0 \in \mathbb{N} \quad f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$

SATZ

$$f \in O(g) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

gängige Größenordnungen

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

$$< O(n \log n) < O(2^n)$$

polynomial effizient

exponentiell

wie dieses Kriterium überprüfen?

Regel von L'Hopital

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{\frac{d}{dn} f(n)}{\frac{d}{dn} g(n)}$$ ← Ableitungen

gilt falls der Limes der rechten Seite existiert

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

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

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

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

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

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

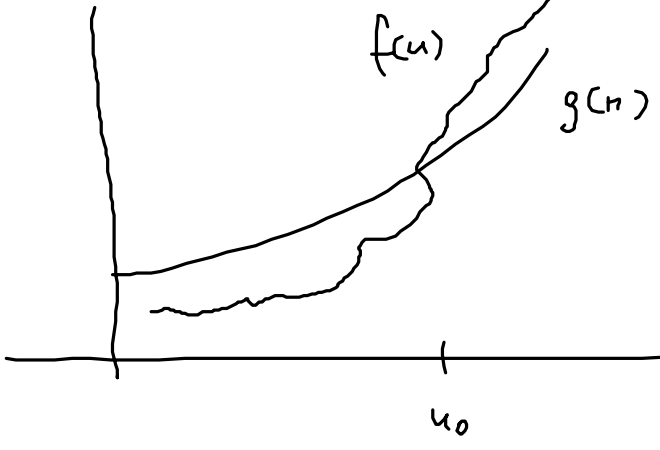
$$= \ldots = \lim_{n\to\infty} \frac{k(k-1)\cdots 2\cdot 1}{(\ln 2)^k \cdot 2^n} \to 0$$

9.2.2 Untere Schranken

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

$f \in o(g)$ f ist von echt kleinerer Größenordnung

$$\Omega(g) := \{ f: \mathbb{N} \to \mathbb{N} \mid \exists\, c > 0,\ \exists\, n_0 \in \mathbb{N}: \ f(n) \geq c \cdot g(n) \ \forall\, n \geq n_0 \}$$



f wächst (bis auf Faktor c) mindestens so schnell wie g (ab n_0)

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

$$\Theta(g) = \{ f: \mathbb{N} \to \mathbb{N} \mid \exists c_1 \exists c_2 \exists n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \, g(n) \quad \forall n \geq n_0 \}$$

Bsp 1: Sequentielle 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 reichen $\Rightarrow C(n) \in O(n)$

weniger als n gehen nicht $\Rightarrow C(n) \in \Omega(n)$

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

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

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

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

↑

n^2 viele c_{ij} $= n^2(2n-1)$ arithm. Operationen

$= 2n^3 - n^2$

$\overset{\text{STZ}}{=} \mathcal{O}(n^3)$

müssen n^2 viele c_{ij} berechnen $\Omega(n^2)$ viele 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 + Beschleunigung

Kap 10 Sortieren in Arrays

Klasse Item (Student)

mit Feld key ⇐ vom Typ int

(Matrikelnummer)

Array

```
Item [ ]  vec  =  new  Item [n]
```

↑

aufsteigend sortieren, d.h.

$vec[0].key \leq vec.[1].key \leq \ldots \leq vec[n-1].key$

10.1 Direkte Methoden

↑ "Sortieren am Ort"

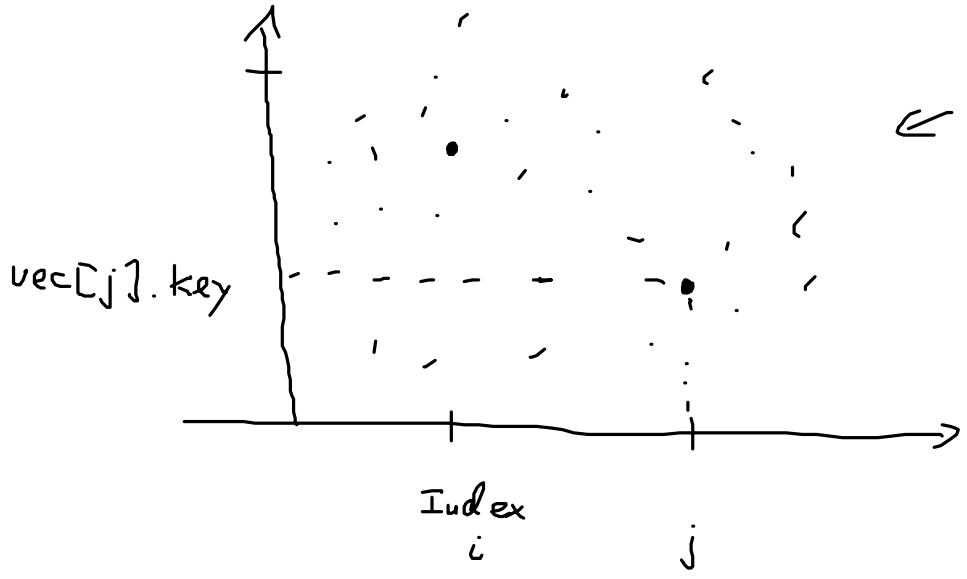
10.1.1 Bubblesort

1. Ges. ist 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 vec[j-1].key und vec[j].key
   Ist vec[j-1].key > vec[j].key so werden

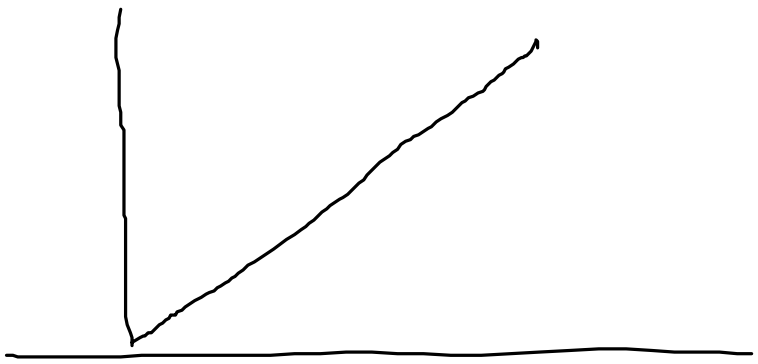
vec[j-1] und vec[j] vertauscht

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

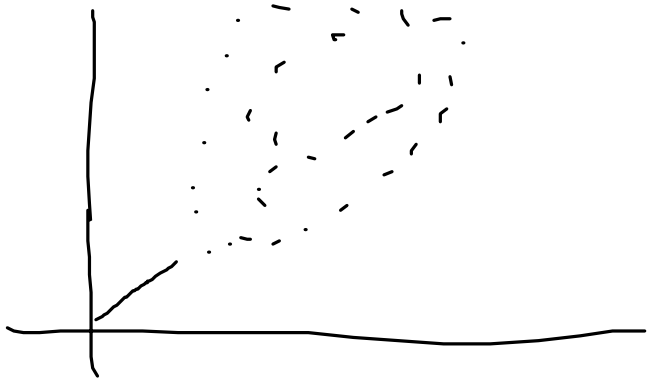
Visualisierung



← unsortiert
≙ Punktwolke
alle Werte auf y Achse
kommen genau
ein mal vor



Zwischenstände



Korrektheit:

Schleifeninvariante nach Ende von Phase i

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

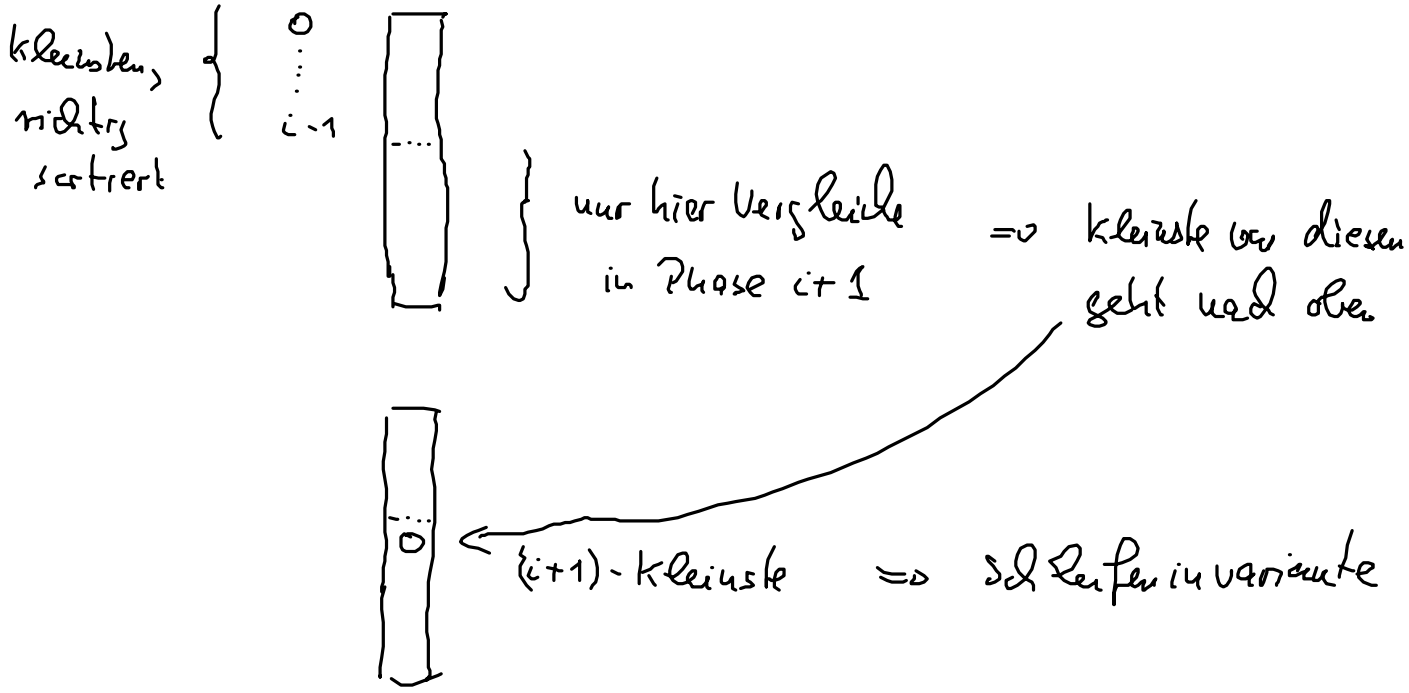
Beweis mit Induktion nach # Phasen

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

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

Schluss auf Phase i+1:

Phase i



Aufwand: (worst case, # Vergleiche)

Phase		
1	$n-1$	Vergleiche
2	$n-2$	
⋮		
i	$n-i$	
⋮		
$n-1$	1	

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

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

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

$$\in O(n^2)$$

Brauchen immer so viele bei jedem Lauf

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

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

Zuweisungen $A(n)$ (worst case)

A ≙ assignment

1 Vertauschung = 3 Zuweisungen

Phase 1 $n-1$ Vertauschungen
2 $n-2$
⋮

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

mögliche Verbesserungen:

abbrechen falls keine Vertauschung in einer Phase

Shakersort (abwechselnd von hinten nach vorn und von vorn nach hinten)

$\in O(n^2)$

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

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

10.1.2 Selection Sort

1 Gegeben ist vec[] mit n Komponenten

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

3. Phase bottom sucht Index minIndx einer kompatible

mit kleinstem Schlüssel in vec[bottom ... n-1]

mittels sequentieller Suche, und findet vec[bottom] und vec[minIndx]

63 24 12 53 72 18 44 35
↑ ↑

12 | 24 63 53 72 18 44 35
↑ ↑

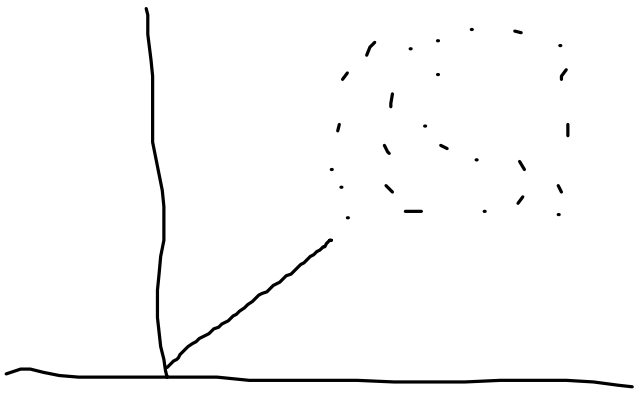
kleinste bottom minIndx

12 18 | 63 53 72 24 44 35

usw

Korrektheit wie bei Bubblesort mit derselben Schleifeninvariante

Visualisierung:



Aufwand: $C(n)$

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

$n-1$ 1

Zuweisungen $A(n)$ — von Arraykomponenten

nur 1 Vertauschung pro Phase

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

↑

besser als Bubblesort