

Übung 1

Institutsleitung
Prof. Dr.-Ing. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Übung zu Informationstechnik II und Automatisierungstechnik – Nathalie Brenner

Prof. Dr.-Ing. Eric Sax



ORGANISATORISCHES



Ansprechpartner

■ Vorlesung

■ Prof. Dr.-Ing. Eric Sax

- Gebäude 30.10 - Raum 336
- Telefon: 608-42500
- eric.sax@kit.edu
- <http://www.itiv.kit.edu>
- Sprechstunden nach Vereinbarung



■ Übungen

■ M.Sc. Nathalie Brenner

- Wissenschaftliche Mitarbeiterin
- Gebäude 30.10 - Raum: 126
- Telefon 608-43093
- Nathalie.brenner@kit.edu
- Sprechstunden nach Vereinbarung

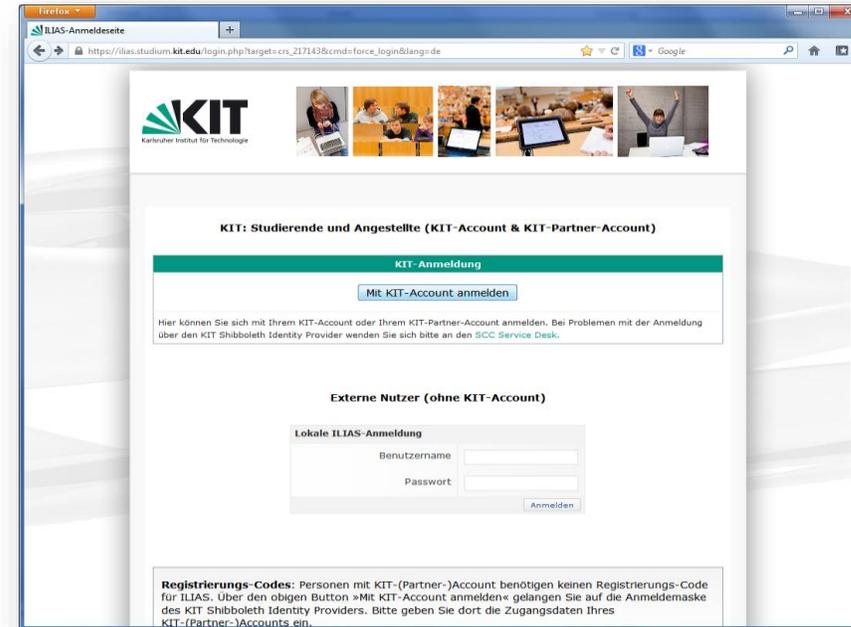


■ Nutzung der Lernplattform ILIAS

- <https://ilias.studium.kit.edu>
- Registrierung notwendig (falls noch nicht erfolgt)
- Kurs: Informationstechnik II und Automatisierungstechnik (SS 2020)
 - Passwort: ITII2020_AutoEncoder (Groß-/Kleinschreibung beachten!)

■ Inhalt

- Links zu Webinaren, Vorlesungsfolien, Übungsaufgaben, Übungsfolien, ...



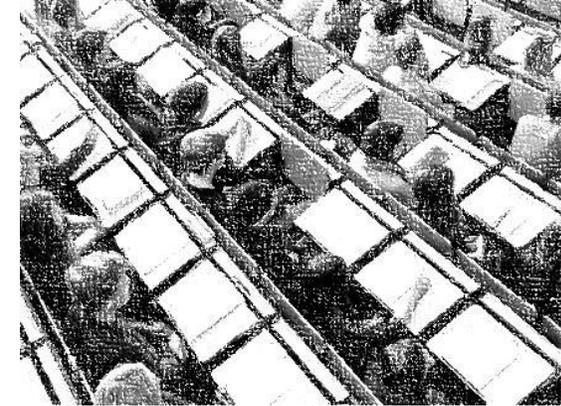
Magazin » Organisationseinheiten » Fakultät für Elektrotechnik und Informationstechnik » SS 2020 » Informationstechnik II und Automatisierungstechnik (SS 2020)



Informationstechnik II und Automatisierungstechnik (SS 2020)

Klausur (schriftlich)

- Inhalt: Vorlesung und Übung
- Termin: Mittwoch, **05.08.2020, 16:00 bis 18:00 Uhr**
- Ort:
 - Voraussichtlich Carl-Benz-Hörsaal, Gottlieb-Daimler-Hörsaal, Frit-Haller-Hörsaal, Gerthsen-Hörsaal und Gaede-Hörsaal
- Dauer: 2 Stunden
- Mitzubringen: Fricard oder Lichtbildausweis und Schreibzeug
 - Nicht Bachelorstudenten zusätzlich: Studienzeitbescheinigung
- Hilfsmittel: 1x A4 Blatt (2x A4 Seiten, handschriftlich, lesbar ohne Hilfsmittel)
- Hörsaalverteilung (nach Nachnamen, bei Doppelnamen ist der erste Nachname maßgebend)
 - Details werden vor der Prüfung bekannt gegeben.



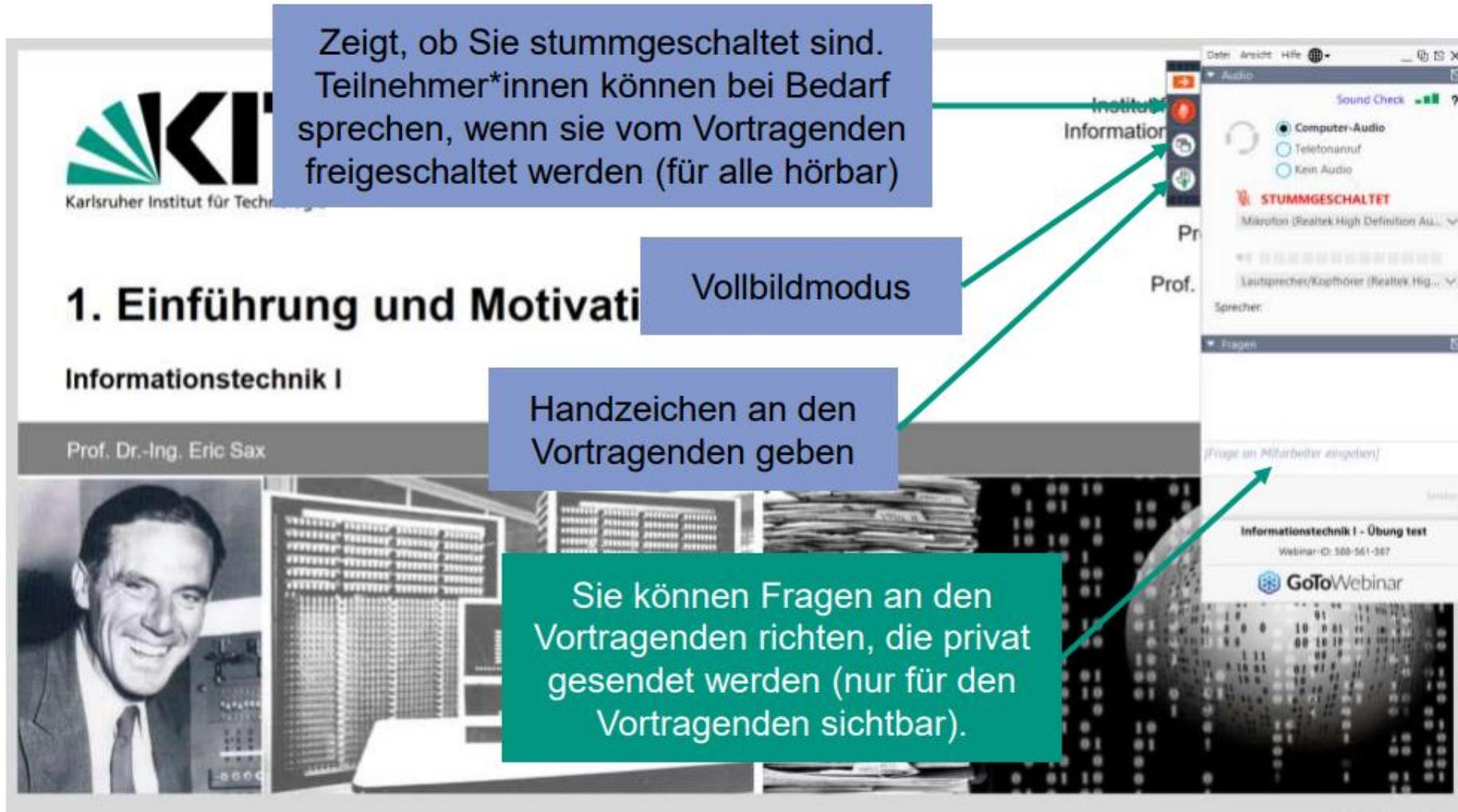
Anmeldung zur Klausur

- Bachelor:
 - Verwendung der Selbstbedienungsfunktion des Studienbüros <https://campus.studium.kit.edu/>
- Andere oder nicht online freigeschaltete Studiengänge:
 - Direkte Anmeldung durch Abgabe des blauen Zettels oder Entsprechendem bei einem Betreuer (spätestens eine Woche vor der Prüfung)

- Anmeldeschluss: Mittwoch, 29.07.2020
- Abmeldeschluss: Dienstag, 04.08.2020

ITII-Veranstaltungsplan SS2020

Veranstaltungsplan Informationstechnik 2 SS2020						
Vorlesung/Übung:		Mo, 8:00 - 9:30 Uhr	Online			
Vorlesung/Übung:		Fr, 9:45 - 11:15 Uhr	Online			
Woche	Datum Mo.	Vorlesung	Datum Fr.	Übung	Inhalt Mo.	Inhalt Fr.
1.	20. Apr		24. Apr	VL		VL0: Einleitung und Motivation
2.	27. Apr	VL	01. Mai	frei	VL1: Algorithmische Grundlagen	Tag der Arbeit
3.	04. Mai	VL	08. Mai	VL	VL2: Sortieralgorithmen	VL3: Algorithmen auf Graphen
4.	11. Mai	ÜB	15. Mai	VL	Organisatorisches; Übung 1	VL4: Optimierungsalgorithmen
5.	18. Mai		22. Mai	VL		VL5: Big Data Einführung
6.	25. Mai	ÜB	29. Mai	VL	Übung 2	VL6: Big Data - Prozesse
7.	01. Jun	frei	05. Jun	frei	Pfingsten	Pfingsten
8.	08. Jun	ÜB	12. Jun	VL	Übung 3	VL7: Big Data - Datenvorverarbeitung
9.	15. Jun		19. Jun	VL		VL8: Big Data - Modelling 1
10.	22. Jun	ÜB	26. Jun	VL	Übung 4	VL9: Big Data - Modellin 2
11.	29. Jun	ÜB	03. Jul	VL	Übung 5	VL10: Big Data - Evaluation
12.	06. Jul		10. Jul	VL		VL11: Big Data - Infrastruktur
13.	13. Jul	ÜB	17. Jul	VL	Übung 6	VL12: Anwendungen Institut
14.	20. Jul	ÜB	24. Jul	VL	Übung 7	VL13: Puffer



The image shows a screenshot of a GoTo Webinar interface. The main content area displays the KIT logo, the title "1. Einführung und Motivation", and the course name "Informationstechnik I" by Prof. Dr.-Ing. Eric Sax. A photo of Prof. Sax is visible in the bottom left. On the right, a control panel is open, showing audio settings (muted), a microphone selection dropdown, and a "Fragen" (Questions) section. A "Handzeichen" (hand icon) is visible in the top right of the main area. Four callout boxes provide instructions: a blue box explains the mute status indicator; a blue box points to the "Vollbildmodus" (Full Screen) button; a blue box points to the hand icon; and a green box explains the private question feature.

Zeigt, ob Sie stummgeschaltet sind. Teilnehmer*innen können bei Bedarf sprechen, wenn sie vom Vortragenden freigeschaltet werden (für alle hörbar)

Vollbildmodus

Handzeichen an den Vortragenden geben

Sie können Fragen an den Vortragenden richten, die privat gesendet werden (nur für den Vortragenden sichtbar).

- Aufteilung
- Klausur
- Organisatorisches

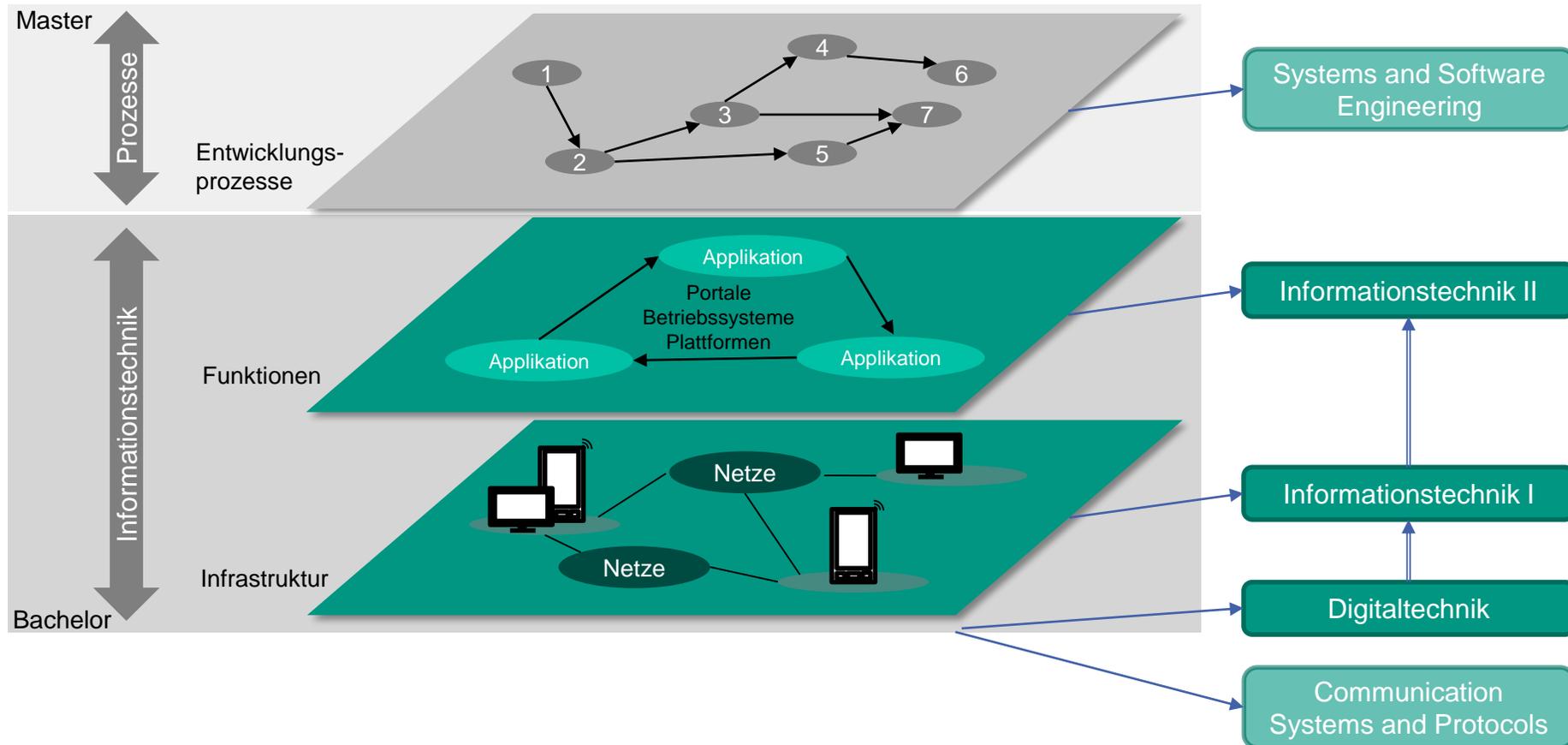


VERANSTALTUNGSINHALT



Veranstaltungsinhalt

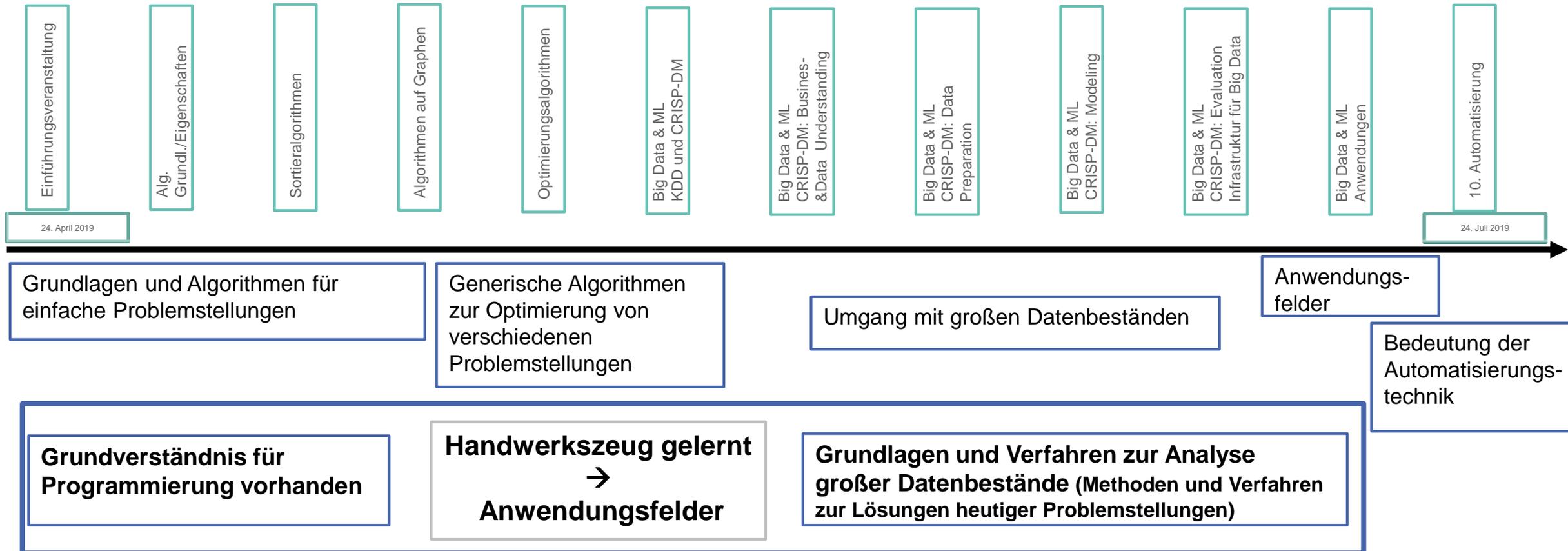
Einordnung in das konsekutive Studium



Veranstaltungsübersicht

Warum Informationstechnik II?

- Problemstellungen wie sie in der Realität vorkommen
- Programmierfähigkeiten erweitern mit der Vertiefung auf algorithmisches Denken
- Anwendungsfelder des erlernten Wissen für heutige Problemstellungen



Veranstaltungsinhalt

Aufteilung der einzelnen Übungseinheiten

1

- Organisatorisches; Merkmale und Eigenschaften von Algorithmen; Sortieralgorithmen

2

- Sortieralgorithmen; Algorithmen auf Graphen

3

- Optimierungsalgorithmen; Einführung in Big Data Prozesse

4

- (Big Data) Geschäfts- und Datenverständnis

5

- (Big Data) Datenaufbereitung

6

- (Big Data) Modelling: Überwachte Lernverfahren

7

- (Big Data) Modelling: Unüberwachte Lernverfahren

INHALT ÜBUNG 1



Ziele der heutigen Übung



■ Nach der heutigen Übung können Sie....

- ... Merkmale, Eigenschaften und Klassen von Algorithmen benennen, einordnen und bestimmen

1

- ... Merkmale von Algorithmen benennen

2

- ... Algorithmen anhand von ihren Merkmalen voneinander abgrenzen

3

- ... Eigenschaften von Algorithmen benennen und bestimmen

4

- ... Algorithmen nach ihren Merkmalen und Eigenschaften in Klassen einordnen

5

- ... Merkmale von verschiedenen Sortieralgorithmen nennen

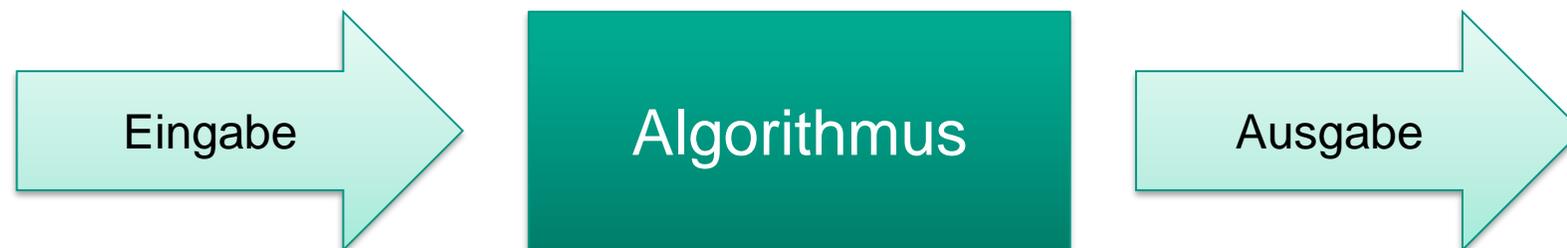
MERKMALE UND EIGENSCHAFTEN VON ALGORITHMEN



Beschreibung von Algorithmen

Was ist ein Algorithmus?

- Algorithmen...
 - ... sind genau definierte Handlungsvorschriften zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten
 - ... sind eine Folge von Anweisungen zur Lösung eines bestimmten Problems
 - ... sind eine Folge von wohldefinierten Anweisungen
 - ... werden nach einer strikten Syntax geschrieben
- Algorithmen im engeren Sinne der Informatik
 - Wohldefinierte Rechenvorschrift, die eine Größe oder eine Menge als Eingabe verwendet und eine Größe oder eine Menge von Größen als Ausgabe erzeugt



Beschreibung von Algorithmen

Merkmale und Eigenschaften

■ Was fallen Ihnen für mögliche Eigenschaften von Algorithmen ein?



abbrechbar	Alg. Kann zu jedem beliebigen Zeitpunkt abgebrochen werden, liefert nicht optimales Ergebnis
Heuristisch	Mit geringem Rechenaufwand und kurzer Laufzeit zulässige Lösungen für bestimmtes Problem erhalten
Stabil	Der Alg. produziert für alle erlaubten und in der Größenordnung der Rechengenauigkeit gestörten Eingabedaten akzeptable Resultate
In place	Benötigt konstante Menge von Speicher, Eingabedaten werden von Ausgabedaten überschrieben
Inkrementell	Während der Laufzeit werden Ergebnisse hinzugefügt
Finitheit	Der Algorithmus benötigt zu jedem Schritt nur endlich viel Speicherplatz
Determiniertheit	Der Algorithmus liefert bei gleichen Voraussetzungen stets das gleiche Ergebnis
Ausführbarkeit	Jeder Einzelschritt muss ausführbar sein
Rekursiv	Algorithmus Ruft sich selber auf, aber es kommen keine Schleifen vor

Beschreibung von Algorithmen

Merkmale und Eigenschaften: rekursiv oder iterativ?

- Was fallen Ihnen für mögliche Eigenschaften von Algorithmen ein?



Iterative Algorithmen

- Ein Algorithmus ist iterativ, wenn in seiner Realisierung keine Rekursionen vorkommen.

Rekursive Algorithmen

- Ein Algorithmus ist rekursiv, wenn dieser sich in seiner Realisierung nur selbst aufruft, jedoch keine Schleifen vorkommen.

Halbiterative/ Halbrekursive Algorithmen

- Ein Algorithmus ist halbiterativ/ halbrekursiv, wenn er weder iterativ noch rekursiv ist, jedoch hauptsächlich solche iterative/ rekursive Merkmale besitzt.

Parallele Algorithmen

- Ein Algorithmus heißt parallel, wenn eine Laufzeitoptimierung möglich ist, indem er auf einem Mehrprozessorsystem implementiert wird.

Beschreibung von Algorithmen

Merkmale und Eigenschaften: Beispiele (II)

■ Fakultätsberechnung

$$f(n) = n! = n * (n - 1) * .. * 1$$

■ Eigenschaften?

Abbrechbar	✗
Heuristisch	✗
Stabil	✓
In place	✗
Inkrementell	✗
Rekursiv	✓
Iterativ	✗

```
int fakultaet (int zahl)
  if (zahl <=1){
    return 1;
  }
  else (zahl <=-1){
    return zahl * fakultaet(zahl-1);
  }
}
```

```
Int main() {
  int zahl;
  std::cin >> zahl;
  ergebnis = fakultaet(zahl);
  std::cout << ergebnis;
}
```

Zwischenübung



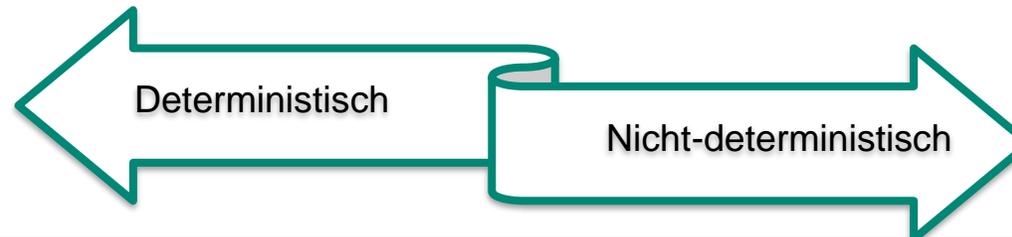
EIGENSCHAFTEN VON ALGORITHMEN - LAUFZEITKOMPLEXITÄT



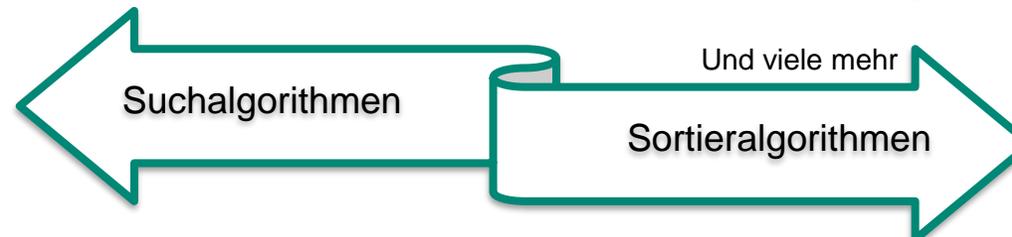
Beschreibung von Algorithmen

Einteilung von Algorithmen in Klassen

- Algorithmen können nach Eigenschaften in Klassen eingeteilt werden
 - Maschinenfähigkeit

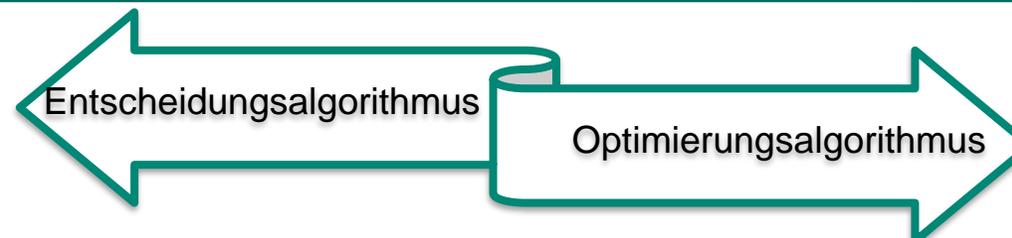


- Anwendung



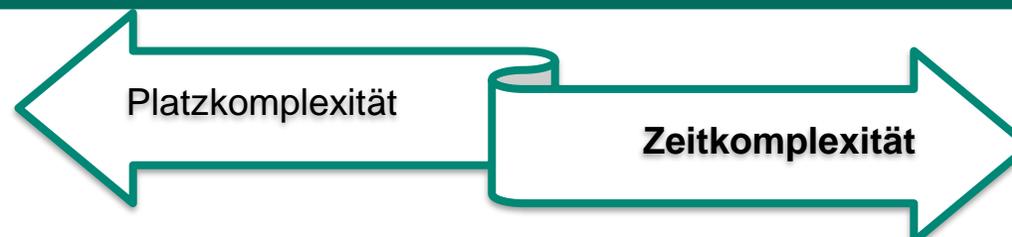
IT2 – Übung 1+2

- Problemstellung



IT2 – Übung 2

- Komplexität



IT2 – Übung 1

Eigenschaften von Algorithmen

Laufzeit und Laufzeitkomplexität

■ Zeitkomplexität:

- „die Anzahl der Rechenschritte, die ein optimaler Algorithmus zur Lösung des Problems benötigt, in Abhängigkeit der Länge der Eingabe“
- Wie verhält/steigt der Zeitbedarf, wenn die Eingabemenge wächst?
 $O(n)$ beschreibt das Wachstumsverhalten eines Algorithmus (nicht das eines Programms!)
- Laufzeit wird in Abhängigkeit der Eingabeelemente n angegeben

■ Problem für Laufzeitanalyse:

- Abschätzung unter Verzicht auf Einzelheiten.
 - 90/10-Regel:
90% der Laufzeit bewirkt durch nur 10% des Programm-Codes
 - 1:50 Regel:
1% des Programm-Codes bewirkt 50% der Laufzeit

Teilgebiete der Algorithmenanalyse

- Komplexitätstheorie:
 - Verhalten von Algorithmen bezüglich Ressourcenbedarf wie Rechenzeit und Speicherbedarf
 - Ressourcenbedarf wird dabei in Abhängigkeit von der Länge der Eingabe ermittelt, z.B.:
 - Anzahl Elemente eines Feldes, einer Matrix
 - Länge einer Zeichenkette
 - Grad eines Polynoms
 - Anzahl der Knoten in einer dynamischen Datenstruktur (Liste, Baum)
 - → Die meisten Algorithmen besitzen einen **Hauptparameter n** , der die Anzahl der zu verarbeitenden Datenelemente angibt.
- Ziel: Bestimmung der Laufzeit unabhängig von
 - Hardware
 - Betriebssystem
 - Programmiersprache
 - Verwendete Bibliotheken
 - Compiler-Optimierungen

Relativer Laufzeitaufwand

Feinanalyse

Ziel: Bestimmung der Laufzeit unabhängig von

- Hardware
- Betriebssystem
- Programmiersprache
- Verwendete Bibliotheken
- Compiler-Optimierungen

nur bei sicherheitsrelevanten Systemen mit harten Realzeitanforderungen durchgeführt

In den meisten praktischen Anwendungen stattdessen nur Grobanalyse

Analysiere prinzipielles Laufzeitverhalten in Abhängigkeit der Problemgröße n

- Günstigster Fall (best case) → minimale Zahl an Arbeitsschritten
- Ungünstigster Fall (worst case) → maximale Zahl an Arbeitsschritten
- Durchschnittlicher Fall (average case)

Elementare Aktionen und Operationen mit *integer*-Datenobjekten und ihr relativer Zeitaufwand bezogen auf die Zuweisung

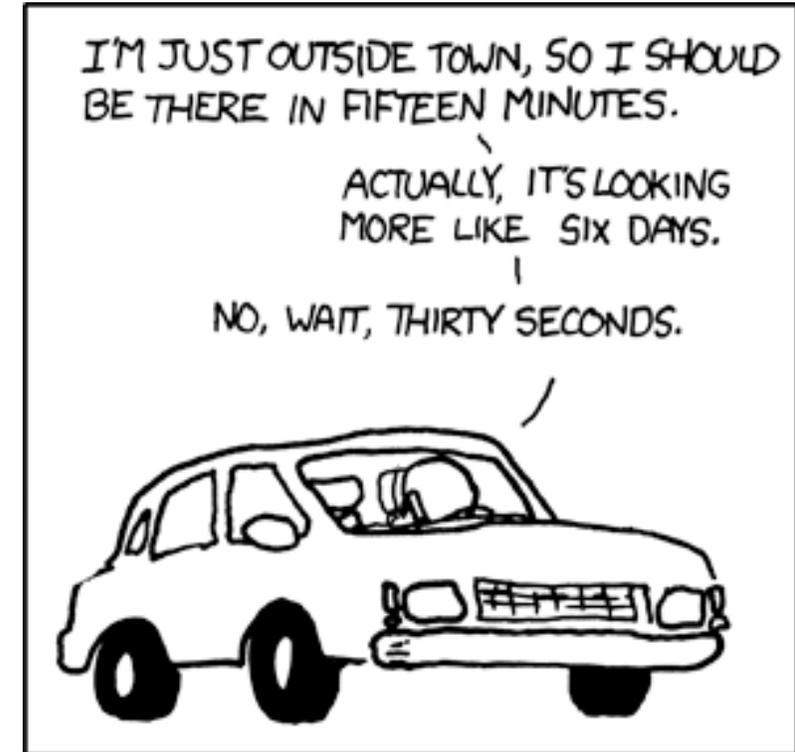
Elementare Aktion/Operation	Relativer Zeitaufwand
Zuweisung (= Basis)	1.0
Addition oder Subtraktion	1.4
Multiplikation	2.3
Division	8.0
Vergleich (inkl. Sprung bei <i>if</i> oder <i>while</i>)	1.5
Indizierung einer Matrix	4.2

Werte wurden ermittelt mit einem PASCAL-Programm,
 Borland PASCAL-Compiler Version 7.0,
 auf PC unter Betriebssystem Microsoft XP
 mit Prozessor Intel Pentium 4 mit 2GHz Taktfrequenz

Laufzeitanalyse

Wenn's schief geht...

- Günstigster Fall (best case)
- Ungünstigster Fall (worst case)
- Durchschnittlicher Fall (average case)



THE AUTHOR OF THE WINDOWS FILE COPY DIALOG VISITS SOME FRIENDS.

Relativer Laufzeitaufwand:

Grobanalyse (Laufzeitkomplexitätsklassen): O-Notation

■ Laufzeit von elementaren Operationen

<code>i1 := 0;</code>	$O(1)$
-----------------------	--------------------------

■ Laufzeit einer Sequenz von elementaren Operationen

<code>i1 := 0;</code>	$O(1)$	$42 * O(1) = O(1)$ Wenn 42 unabhängig von n (Eingabelemente) ist
<code>i2 := 0;</code>	$O(1)$	
...	...	
<code>i42 := 0;</code>	$O(1)$	

Asymptotische Laufzeitkomplexität	Bezeichnung	Erläuterung am Beispiel der Verdoppelung der Problemgröße und typische Algorithmen mit dieser Ordnung
$O(1)$	konstant	Laufzeit ist unabhängig von der Problemgröße, optimaler Fall, der praktisch nicht auftritt
$O(\log n)$	logarithmisch	Verdoppelung der Problemgröße bewirkt Anstieg der Laufzeit um $\log 2$, also um eine Konstante (um 1 für <i>logarithmus dualis</i>), sehr günstig und daher erstrebenswert, z.B. <i>binäre Suche</i> (siehe Kapitel 6)
$O(n)$	linear	Verdoppelung der Problemgröße bewirkt Verdoppelung der Laufzeit, immer noch zufrieden stellend, z.B. <i>sequenzielle Suche</i> (siehe Kapitel 6)
$O(n \log n)$	–	Fast so gut wie linear, weil $\log n$ im Verhältnis zu n klein ist, z.B. gute Sortierverfahren wie <i>Quicksort</i> (siehe Kapitel 7)
$O(n^2)$	quadratisch	Verdoppelung der Problemgröße bewirkt Vervierfachung der Laufzeit, ungünstig, z.B. schlechte Sortierverfahren wie <i>Bubblesort</i> (siehe Kapitel 7)
$O(n^3)$	kubisch	Verdoppelung der Problemgröße bewirkt Veracht-fachung der Laufzeit, sehr unbefriedigend, z.B. einfache <i>Matrizenmultiplikation</i>
$O(k^n)$	exponentiell	Verdoppelung der Problemgröße bedeutet Quadrierung (weil $k^{2n} = (k^n)^2$) der Laufzeit, katastrophal, z.B. <i>Backtracking</i> oder <i>Exhaustionsalgorithmen</i> (siehe Kapitel 9)

Relativer Laufzeitaufwand

O-Notation

■ Laufzeit einer Schleife

for i:= 1 to n do	O(n)	O(1)*O(n) =O(n)
begin		
a[i] :=0;	O(1)	
end;		

for i:= 1 to n do	O(n)	O(1)*O(n) =O(n)
begin		
a1[i] :=0; a2[i] :=0; ... a42[i] :=0;	42*O(1) =O(1)	
end;		

Asymptotische Laufzeitkomplexität	Bezeichnung	Erläuterung am Beispiel der Verdoppelung der Problemgröße und typische Algorithmen mit dieser Ordnung
O(1)	konstant	Laufzeit ist unabhängig von der Problemgröße, optimaler Fall, der praktisch nicht auftritt
O(log n)	logarithmisch	Verdoppelung der Problemgröße bewirkt Anstieg der Laufzeit um log 2, also um eine Konstante (um 1 für <i>logarithmus dualis</i>), sehr günstig und daher erstrebenswert, z.B. <i>binäre Suche</i> (siehe Kapitel 6)
O(n)	linear	Verdoppelung der Problemgröße bewirkt Verdoppelung der Laufzeit, immer noch zufrieden stellend, z.B. <i>sequenzielle Suche</i> (siehe Kapitel 6)
O(n log n)	-	Fast so gut wie linear, weil log n im Verhältnis zu n klein ist, z.B. gute Sortierverfahren wie <i>Quicksort</i> (siehe Kapitel 7)
O(n ²)	quadratisch	Verdoppelung der Problemgröße bewirkt Vervielfachung der Laufzeit, ungünstig, z.B. schlechte Sortierverfahren wie <i>Bubblesort</i> (siehe Kapitel 7)
O(n ³)	kubisch	Verdoppelung der Problemgröße bewirkt Veracht-fachung der Laufzeit, sehr unbefriedigend, z.B. einfache <i>Matrizenmultiplikation</i>
O(k ⁿ)	exponentiell	Verdoppelung der Problemgröße bedeutet Quadrierung (weil k ²ⁿ = (k ⁿ) ²) der Laufzeit, katastrophal, z.B. <i>Backtracking</i> oder <i>Exhaustionsalgorithmen</i> (siehe Kapitel 9)

Relativer Laufzeitaufwand:

Grobanalyse (Laufzeitkomplexitätsklassen): O-Notation

Zuweisung	$O(1)$
Binäre Suche	$O(\log n)$
Lineare Suche	$O(n)$
Schlaues Sortieren	$O(n \log n)$
Dummes Sortieren	$O(n^2)$
Gleichungssystem lösen	$O(n^3)$

Asymptotische Laufzeitkomplexität	Bezeichnung	Erläuterung am Beispiel der Verdoppelung der Problemgröße und typische Algorithmen mit dieser Ordnung
$O(1)$	konstant	Laufzeit ist unabhängig von der Problemgröße, optimaler Fall, der praktisch nicht auftritt
$O(\log n)$	logarithmisch	Verdoppelung der Problemgröße bewirkt Anstieg der Laufzeit um $\log 2$, also um eine Konstante (um 1 für <i>logarithmus dualis</i>), sehr günstig und daher erstrebenswert, z.B. <i>binäre Suche</i> (siehe Kapitel 6)
$O(n)$	linear	Verdoppelung der Problemgröße bewirkt Verdoppelung der Laufzeit, immer noch zufrieden stellend, z.B. <i>sequenzielle Suche</i> (siehe Kapitel 6)
$O(n \log n)$	–	Fast so gut wie linear, weil $\log n$ im Verhältnis zu n klein ist, z.B. gute Sortierverfahren wie <i>Quicksort</i> (siehe Kapitel 7)
$O(n^2)$	quadratisch	Verdoppelung der Problemgröße bewirkt Vervierfachung der Laufzeit, ungünstig, z.B. schlechte Sortierverfahren wie <i>Bubblesort</i> (siehe Kapitel 7)
$O(n^3)$	kubisch	Verdoppelung der Problemgröße bewirkt Veracht-fachung der Laufzeit, sehr unbefriedigend, z.B. einfache <i>Matrizenmultiplikation</i>
$O(k^n)$	exponentiell	Verdoppelung der Problemgröße bedeutet Quadrierung (weil $k^{2n} = (k^n)^2$) der Laufzeit, katastrophal, z.B. <i>Backtracking</i> oder <i>Exhaustionsalgorithmen</i> (siehe Kapitel 9)

Relativer Laufzeitaufwand:

Grobanalyse (Laufzeitkomplexitätsklassen): O-Notation

■ Rechengesetze zur O-Notation:

- $c * n \rightarrow O(n)$
- $O(c * n) \rightarrow O(n)$
- $c * O(n) \rightarrow O(n)$
- $O(O(n)) \rightarrow O(n)$
- $O(n_1) + O(n_2) + \dots + O(n_x) \rightarrow O(\max(n_i))$
- $O(n) * O(m) \rightarrow O(n * m)$
- $m = O(n) \rightarrow O(n + m) = O(n)$

- $n^a \rightarrow O(n^b)$, wenn $a < b$
- $\log_a n \rightarrow O(\log_b n)$, $\forall a, b > 1$

Asymptotische Laufzeitkomplexität	Bezeichnung	Erläuterung am Beispiel der Verdoppelung der Problemgröße und typische Algorithmen mit dieser Ordnung
$O(1)$	konstant	Laufzeit ist unabhängig von der Problemgröße, optimaler Fall, der praktisch nicht auftritt
$O(\log n)$	logarithmisch	Verdoppelung der Problemgröße bewirkt Anstieg der Laufzeit um $\log 2$, also um eine Konstante (um 1 für <i>logarithmus dualis</i>), sehr günstig und daher erstrebenswert, z.B. <i>binäre Suche</i> (siehe Kapitel 6)
$O(n)$	linear	Verdoppelung der Problemgröße bewirkt Verdoppelung der Laufzeit, immer noch zufrieden stellend, z.B. <i>sequenzielle Suche</i> (siehe Kapitel 6)
$O(n \log n)$	–	Fast so gut wie linear, weil $\log n$ im Verhältnis zu n klein ist, z.B. gute Sortierverfahren wie <i>Quicksort</i> (siehe Kapitel 7)
$O(n^2)$	quadratisch	Verdoppelung der Problemgröße bewirkt Vervierfachung der Laufzeit, ungünstig, z.B. schlechte Sortierverfahren wie <i>Bubblesort</i> (siehe Kapitel 7)
$O(n^3)$	kubisch	Verdoppelung der Problemgröße bewirkt Veracht-fachung der Laufzeit, sehr unbefriedigend, z.B. einfache <i>Matrizenmultiplikation</i>
$O(k^n)$	exponentiell	Verdoppelung der Problemgröße bedeutet Quadrierung (weil $k^{2n} = (k^n)^2$) der Laufzeit, katastrophal, z.B. <i>Backtracking</i> oder <i>Exhaustionsalgorithmen</i> (siehe Kapitel 9)

Relativer Laufzeitaufwand

O-Notation, Laufzeitberechnung, Beispiel

- Laufzeitberechnung rekursiver Algorithmus zur Berechnung der Fakultät $f(n) = n!$

<code>int fakultaet (int zahl)</code>			
<code>if (zahl <=1){</code>	\sum^1	$O(1)$	$4 \cdot O(1)$ $= O(1)$
<code> return 1;</code>	\sum^1	$O(1)$	
<code>}</code>	\sum^1	$O(1)$	
<code>else {</code>	\sum^1	$O(1)$	
<code> return</code>			
<code> zahl*fakultaet(zahl-1);</code>	\sum^1	$O(1)$	
<code>}}</code>			

<code>Int main() {</code>			
<code>int zahl;</code>	\sum^1		
<code>std::cin >> zahl;</code>			
<code>ergebnis = fakultaet(zahl);</code>	$\sum_{i=1}^{zahl}$	$O(n)$	$O(1) \cdot O(n)$ $= O(n)$
<code>std::cout << ergebnis;</code>			
<code>}</code>			

$\sum_{i=1}^{zahl} 1 = zahl = n \xrightarrow{O\text{-Notation}} = O(n)$

Laufzeit: $O(1) \cdot O(n) = O(n)$

O-Notation

Zwischenübung



- Bestimmen Sie die Laufzeit des iterativen Algorithmus zur Berechnung der Fakultät $f(n) = n!$

<code>int fakultaet (int zahl)</code>			
<code>if (zahl <=1){</code>	\sum^1	$O(1)$	$4 \cdot O(1)$ $=O(1)$
<code>return 1;</code>	\sum^1	$O(1)$	
<code>}</code>	\sum^1	$O(1)$	
<code>else {</code>	\sum^1	$O(1)$	
<code>return</code>		$O(1)$	
<code>zahl*fakultaet(zahl-1);</code>	\sum^1		
<code>}}</code>			

<code>int fakultaet (int zahl){</code>		
<code>int ergebnis = 1;</code>		
<code>for (int i=2;i<=zahl;i++){</code>		
<code>ergebnis *=i;</code>		
<code>}</code>		
<code>return ergebnis;</code>		
<code>}</code>		

<code>Int main() {</code>			
<code>int zahl;</code>	\sum^1	$O(1)$	$O(1) \cdot O(n)$ $=O(n)$
<code>std::cin >> zahl;</code>			
<code>ergebnis = fakultaet(zahl);</code>	$\sum_{i=1}^{zahl}$	$O(n)$	
<code>std::cout << ergebnis;</code>			
<code>}</code>			

Laufzeit: $O(1) \cdot O(n) = O(n)$

O-Notation

Zwischenübung



- Bestimmen Sie die Laufzeit des iterativen Algorithmus zur Berechnung der Fakultät $f(n) = n!$

<code>int fakultaet (int zahl)</code>			
<code>if (zahl <=1){</code>	\sum^1	$O(1)$	$4 \cdot O(1)$ $=O(1)$
<code>return 1;</code>	\sum^1	$O(1)$	
<code>}</code>	\sum^1	$O(1)$	
<code>Else {</code>	\sum^1	$O(1)$	
<code>return</code>			
<code>zahl*fakultaet(zahl-1);</code>	\sum^1	$O(1)$	
<code>}}</code>			

<code>int fakultaet (int zahl){</code>			
<code>int ergebnis = 1;</code>	\sum^1	$O(1)$	$3 \cdot O(1) + O(n)$ $=O(n)$
<code>for (int i=2;i<=zahl;i++){</code>	$\sum_{i=2}^{zahl-1}$	$O(n)$	
<code>ergebnis *=i;</code>	\sum^1	$O(1)$	
<code>}</code>			
<code>return ergebnis;</code>	\sum^1	$O(1)$	
<code>}</code>			

$$\sum_{i=2}^{zahl-1} 1_i \approx \sum_{i=1}^{zahl-1} 1_{i-1} \xrightarrow{O\text{-Notation}} \approx n - 1 = O(n)$$

Laufzeit: $O(n)$

<code>Int main() {</code>			
<code>int zahl;</code>	\sum^1	$O(1)$	$O(1) \cdot O(n)$ $=O(n)$
<code>std::cin >> zahl;</code>			
<code>ergebnis = fakultaet(zahl);</code>	$\sum_{i=1}^{zahl}$	$O(n)$	
<code>std::cout << ergebnis;</code>			
<code>}</code>			

Laufzeit: $O(1) \cdot O(n) = O(n)$

Merkmale und Eigenschaften von Algorithmen

Zusammenfassung

- Ein Algorithmus ist eine genau definierte Handlungsvorschriften zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten
- Ein Algorithmus wird durch seine Merkmale und Eigenschaften beschrieben
 - Stabilität
 - Abbrechbarkeit
 - ...
 - Laufzeit
(Eine grobe Laufzeitanalyse kann durch die „O-Notation“ beschrieben werden und gibt die asymptotische Laufzeitanalyse an)
- Algorithmen können durch ihre Merkmale in Klassen eingeordnet werden

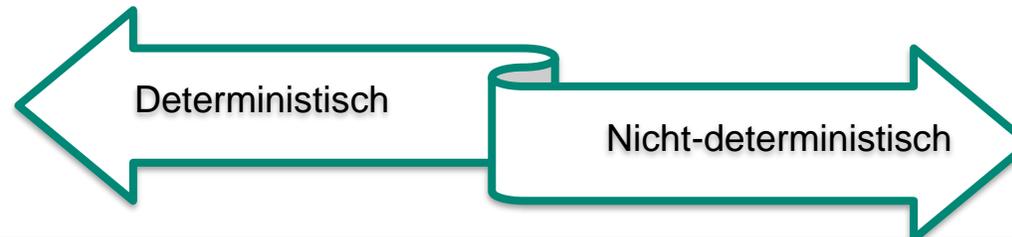
KLASSEN VON ALGORITHMEN - SORTIERALGORITHMEN



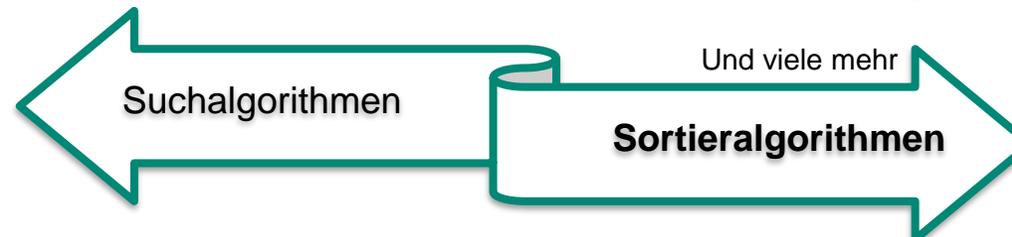
Beschreibung von Algorithmen

Einteilung von Algorithmen in Klassen

- Algorithmen können nach Eigenschaften in Klassen eingeteilt werden
 - Maschinenfähigkeit

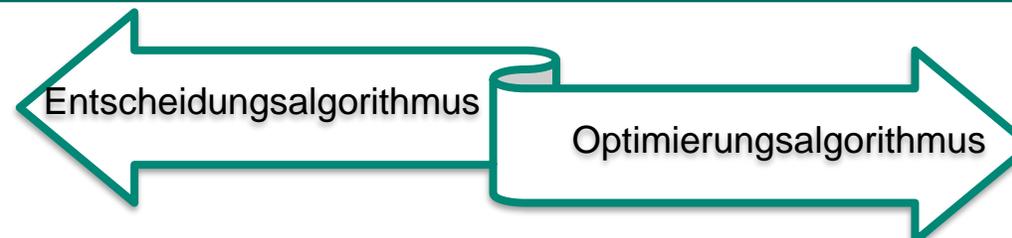


- Anwendung



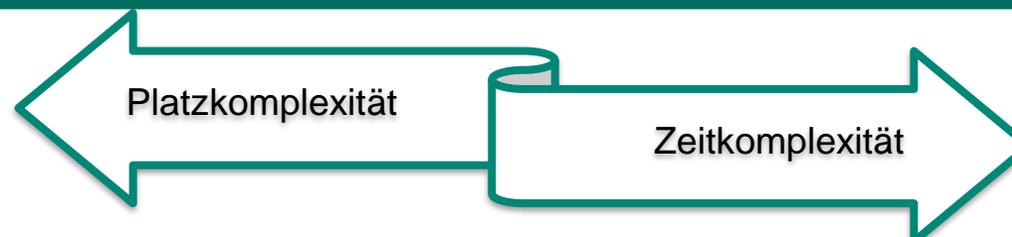
IT2 – Übung 1+2

- Problemstellung



IT2 – Übung 2

- Komplexität

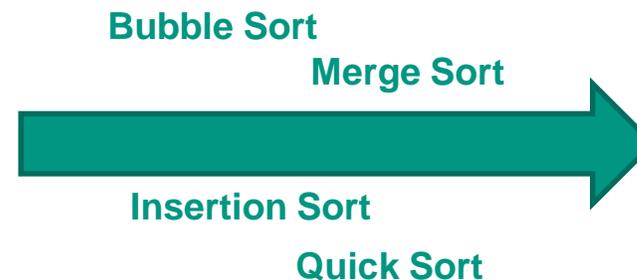


IT2 – Übung 1

Sortieralgorithmen

Wozu braucht man diese Algorithmen?

- Durch einen sortierten Datenbestand kann eine Abfrage deutlich schneller bearbeitet werden!
 - Beim Einfügen der Daten
 - „Aufräumen“ der bestehenden Daten
- Problemstellung:
 - Wie kann ich Datensätze sortieren / sortiert ausgeben?
 - Wie kann ich Personen nach Einkommen sortieren?
 - Wie kann ich Daten sortieren, um diese schneller zu verarbeiten?
- Lösung: Sortieralgorithmen
 - Sortieren eines Arrays oder einer Liste
 - Verschiedene Algorithmen
 - Unterschiedliche Geschwindigkeit, Rechen- und Speicherbedarf



Sortieralgorithmen

Bubble Sort

- Einfacher Algorithmus – langsam, aber kaum zusätzlicher Speicherplatz

- Von Pseudocode zu C++ Quellcode

```
A = [5, 3, 1, 4, 2]
for i = 1 to länge[A] - 1
  do for j = 1 to länge[A] - i
    do if A[j] > A[j+1]
      then vertausche A[j] mit A[j+1]
```

j=1	j=2	j=3	j=4	j=5
5	3	1	4	2



Sortieralgorithmen

Bubble Sort

- Einfacher Algorithmus – langsam, aber kaum zusätzlicher Speicherplatz

- Von Pseudocode zu C++ Quellcode

```
A = [5, 3, 1, 4, 2]
for i = 1 to länge[A] - 1
  do for j = 1 to länge[A] - i
    do if A[j] > A[j+1]
      then vertausche A[j] mit A[j+1]
```



```
int A[] = {5, 3, 1, 4, 2};
for( int i = 0; i < 4; i++ ) {
  for( int j = 0; j < 4 - i; j++ ) {
    if( A[j] > A[j+1] ) {
      int temp = A[j];
      A[j] = A[j+1];
      A[j+1] = temp;
    }
  }
}
```

j=1	j=2	j=3	j=4	j=5
5	3	1	4	2
3	5	1	4	2
3	1	5	4	2
3	1	4	5	2
3	1	4	2	5



...

1	2	3	4	5
---	---	---	---	---

```
C:\WINDOWS\system32\cmd.exe
Array vorher: 5 3 1 4 2
Array nachher: 1 2 3 4 5
Drücken Sie eine beliebige Taste . . .
```

Sortieralgorithmen

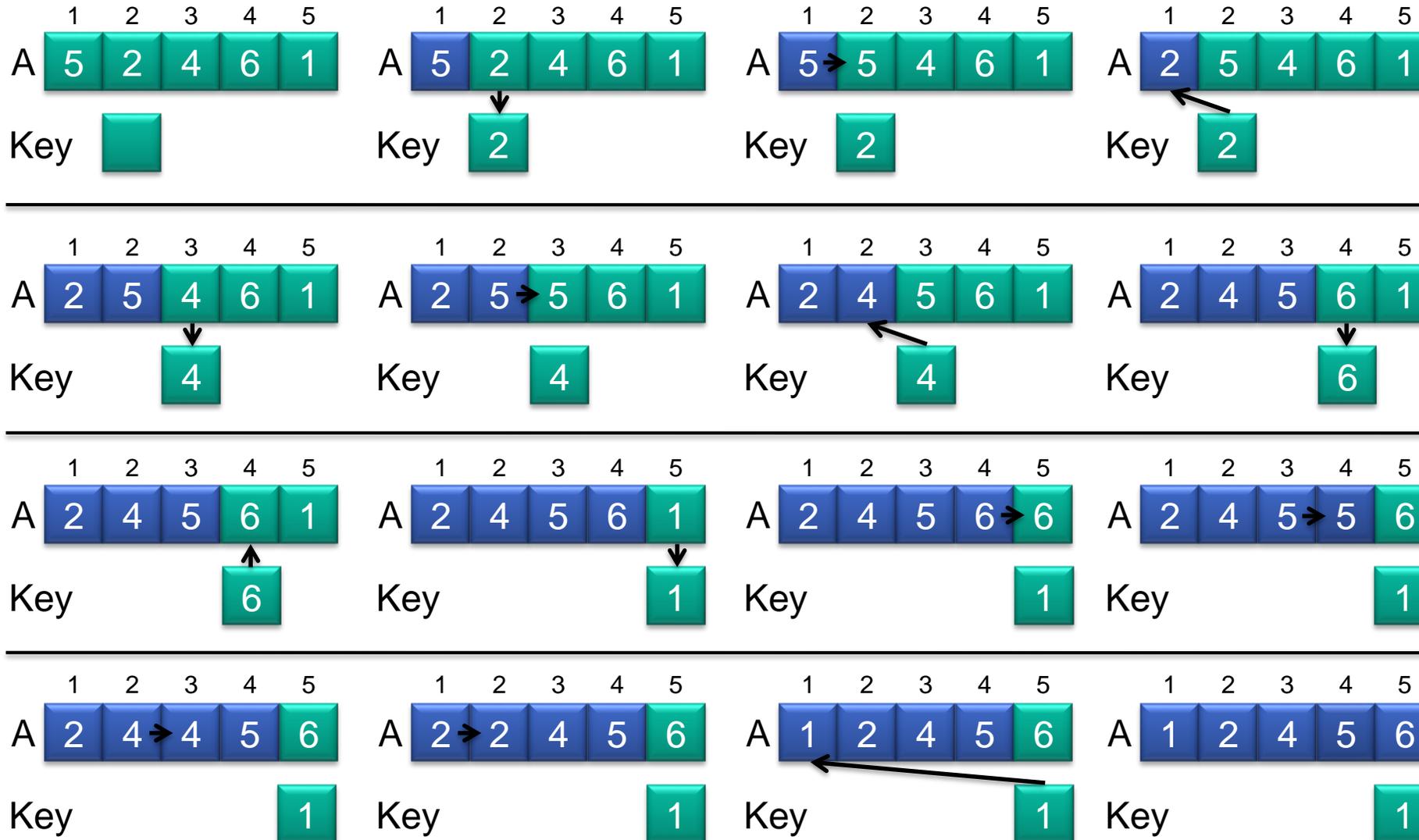
Insertion Sort

- Sortieren durch Einfügen
 - Relativ einfacher Algorithmus
- Vorteile
 - Effizient bei kleinen & bei schon vorsortierten Eingabemengen
 - Einfache Implementierung
 - Stabil
 - Minimal im Speicherverbrauch
- Nachteile
 - Weniger effizient wie komplizierte Sortierverfahren

```
InsertionSort
for (j = 2 to length(A) ) do
  key = A[j]
  i = j - 1
  while ( i > 0 and A[i] > key ) do
    A[i+1] = A[i]
    i = i - 1
  A[i+1] = key
```

Sortieralgorithmus

Insertionsort - Ablauf



Sortieralgorithmen

Zwischenübung - Lsg

- Gegeben sind folgende 6 Buchstaben

E	C	F	A	D	B
---	---	---	---	---	---

- Aufgabe:
Buchstaben mit dem Bubble Sort Algorithmus alphabetisch sortieren.
Dabei jeden Zwischenschritt/Iterationsschritt angeben

- Lösungsziel:

A	B	C	D	E	F
---	---	---	---	---	---



Sortieralgorithmen und Laufzeit

Zwischenübung

- Führen Sie eine Laufzeitanalyse des Insertion Sort Algorithmus durch

```
for j=2 to length(A){  
  do key= A[j];  
  I = j-1;  
  while i>0 and A[i]>key{  
    do A[i+1] = A[i];  
    I = i-1;  
  }  
  A[i+1] = key  
}
```




Sortieralgorithmen und Laufzeit

Zwischenübung

- Führen Sie eine Laufzeitanalyse des Insertion Sort Algorithmus durch



<code>for j=2 to length(A) {</code>	$\sum_{i=1}^n 1 = n$	$O(n)$	$O(n^2)$
<code> do key= A[j];</code>	$\sum_{i=1}^n (1) - 1 = n - 1$	$O(n)$	
<code> I = j-1;</code>	$\sum_{i=1}^n (1) - 1 = n - 1$	$O(n)$	
<code> while i>0 and A[i]>key{</code>	$\sum_{i=1}^n n$	$O(n^2)$	
<code> do A[i+1] = A[i];</code>	$\sum_{i=1}^n n - 1$	$O(n^2)$	
<code> I = i-1;</code>	$\sum_{i=1}^n n - 1$	$O(n^2)$	
<code> }</code>			
<code> A[i+1] = key</code>	$\sum_{i=1}^n (1) - 1 = n - 1$	$O(n)$	
<code>}</code>			

$$\sum_{i=1}^n n = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} \xrightarrow{O\text{-Notation}} = O(n^2)$$

- Laufzeitanalyse
- Sortieralgorithmen
 - Bubble Sort
 - Insertion Sort



Fragen?

Ziele der heutigen Übung



■ Nach der heutigen Übung können Sie....

- ... Merkmale, Eigenschaften und Klassen von Algorithmen benennen, einordnen und bestimmen

1

- ... Merkmale von Algorithmen benennen

2

- ... Algorithmen anhand von ihren Merkmalen voneinander abgrenzen

3

- ... Eigenschaften von Algorithmen benennen und bestimmen

4

- ... Algorithmen nach ihren Merkmalen und Eigenschaften in Klassen einordnen

5

- ... Merkmale von verschiedenen Sortieralgorithmen nennen