Karlsruher Institut für Technologie

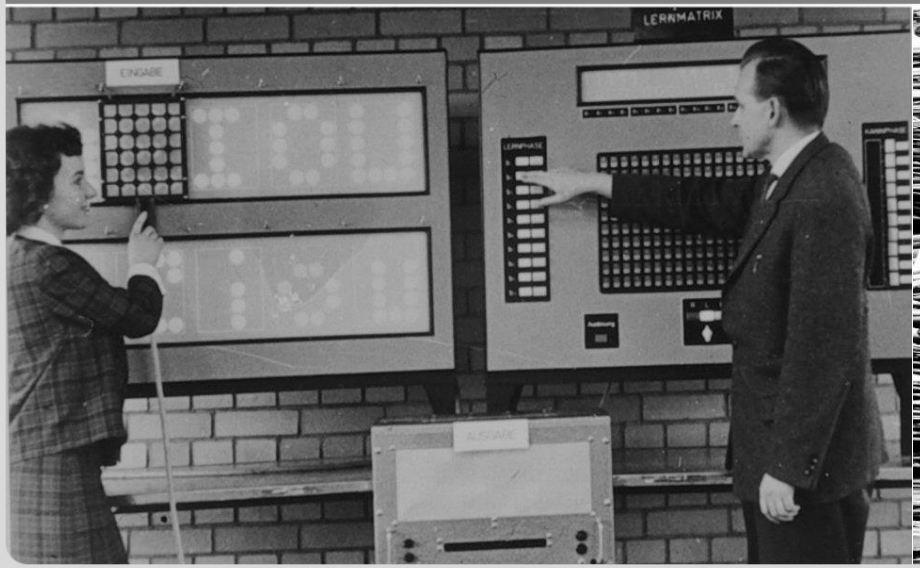
Institut für Technik der Informationsverarbeitung

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

Übung 1

Übung zu Informationstechnik II und Automatisierungstechnik – Nathalie Brenner

Prof. Dr.-Ing. Eric Sax



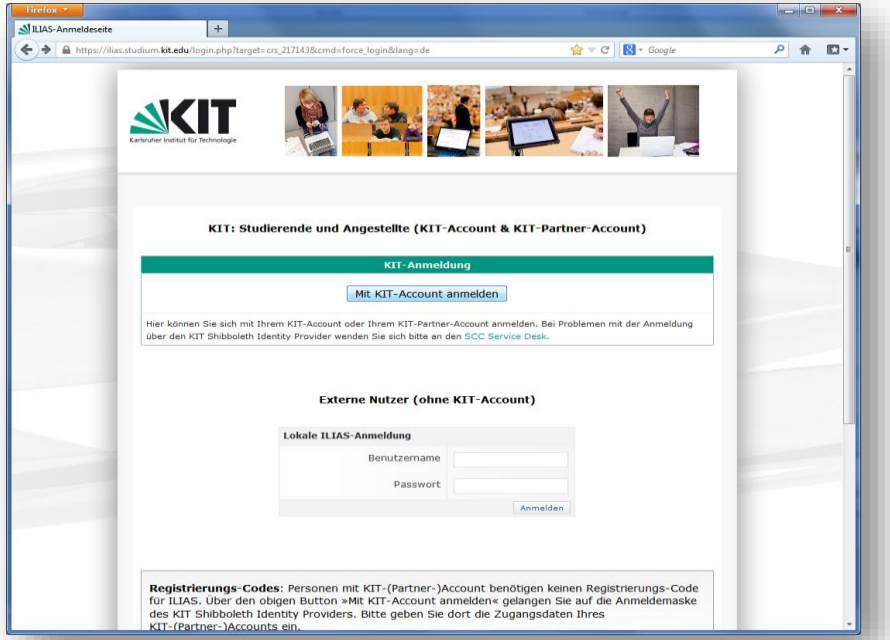
ORGANISATORISCHES

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

Unterlagen zu Informationstechnik

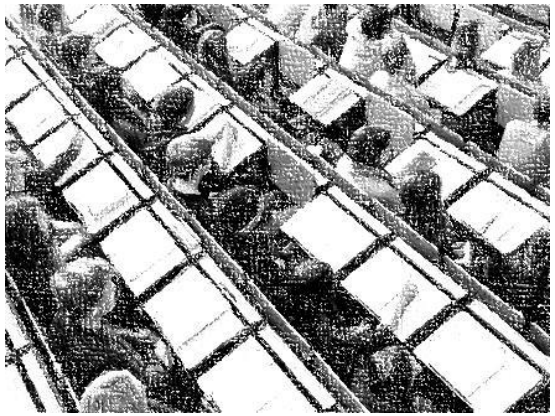
- Nutzung der Lernplattform ILIAS
  - https://ilias.studium.kit.edu
  - Registrierung notwendig
    (falls noch nicht erfolgt)
  - Kurs: [2311654] Informationstechnik II und Automatisierungstechnik
    (SS 2019)
    - Passwort: ITII2019_DecisionTree
      (Groß-/Kleinschreibung beachten!)
- Inhalt
  - Vorlesungsfolien, Übungsaufgaben, Übungsfolien, …



Magazin » Organisationseinheiten » Fakultät für Elektrotechnik und Informationstechnik » SS 2019 » [2311654] Informationstechnik II und Automatisierungstechnik (SS 2019)

ITIV [2311654] Informationstechnik II und Automatisierungstechnik (SS 2019)

Klausur (schriftlich)



- Inhalt: Vorlesung und Übung
- Termin: Donnerstag, 26.09.2019, 11:00 bis 13:00 Uhr
- Ort:
  - Voraussichtlich Fritz-Haller 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: Donnerstag, 19.09.2019
- Abmeldeschluss: Mittwoch, 25.09.2019

ITII-Veranstaltungsplan SS2019

Veranstaltungsplan Informationstechnik 2 SS2019						
Vorlesung/Übung:		Mi, 8:00 - 9:30 Uhr		Gebäude 30.22, Gaede-Hörsaal		
Vorlesung/Übung:		Fr, 9:45 - 11:15 Uhr		Gebäude 30.22, Gaede-Hörsaal		
Woche	**Datum Mi.**	**Vorlesung**	**Datum Fr.**	**Übung**	**Inhalt Mi.**	**Inhalt Fr.**
1.	24. Apr		26. Apr	VL		VL0: Einleitung und Motivation
2.	01. Mai	frei	03. Mai	VL	Tag der Arbeit	VL1: Algorithmische Grundlagen
3.	08. Mai	ÜB	10. Mai	VL	ÜB1: Organisatorisches; Besprechung Aufg. 1;	VL2: Sortieralgorithmen
4.	15. Mai		17. Mai	VL		VL3: Algorithmen auf Graphen
5.	22. Mai	ÜB	24. Mai	VL	ÜB2: Besprechung Aufg. 2;	VL4: Optimierungsalgorithmen
6.	29. Mai		31. Mai	VL		VL5: Big Data Einführung
7.	05. Jun	ÜB	07. Jun	VL	ÜB3: Besprechung Aufg. 3;	VL6: Big Data - Prozesse
8.	12. Jun	frei	14. Jun	frei	Pfingsten	Pfingsten
9.	19. Jun	ÜB	21. Jun	VL	ÜB4: Besprechung Aufg. 4;	VL7: Big Data - Modelling 1
10.	26. Jun	ÜB	28. Jun	VL	ÜB5: Besprechung Aufg. 5;	VL8: Big Data - Modellin 2
11.	03. Jul		05. Jul	VL		VL9: Big Data - Evaluation
12.	10. Jul	ÜB	12. Jul	VL	ÜB6: Besprechung Aufg. 6;	VL10: Big Data - Infrastruktur
13.	17. Jul	ÜB	19. Jul	VL	ÜB7: Besprechung Aufg. 7;	VL11: Anwendungen Institut
14.	24. Jul	ÜB	26. Jul	VL	ÜB8: Besprechung Aufg. 8;	VL12: Cyber Security

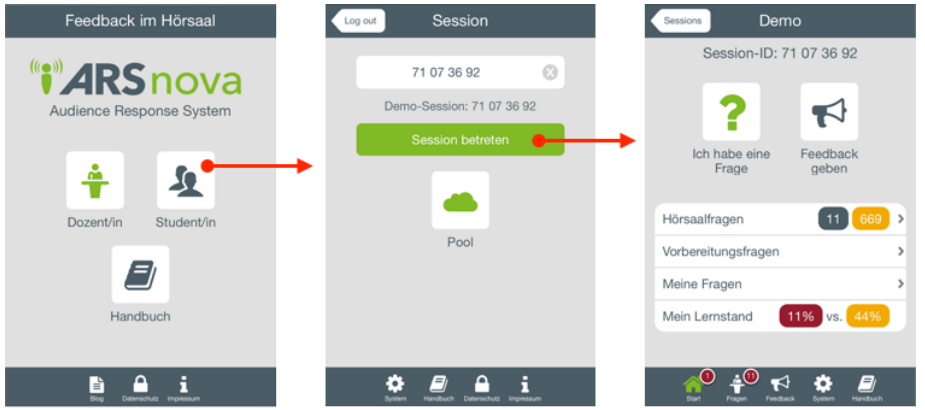
KW= Kalenderwoche; SW = Semesterwoche; VL = Vorlesung; ÜB = Übung; T= Tutorium; P= Praktikum; T4 u. T5 wertvoll für P

Live Feedback und Interaktive Gestaltung

- Keine Übungsblätter, aber zur Prüfungsvorbereitung Übungsaufgaben

- Wir wollen, dass Sie aktiv werden!
- Beantworten Sie Fragen während der Übung mittels Smartphone oder Laptop um:
  - Uns Feedback zu geben
  - Einen Wissenscheck bei den Verständnisfragen durchzuführen
  - Etc.

- Open-Source Online Tool ARSnova
  - https://arsnova.thm.de/blog/en/homepage/



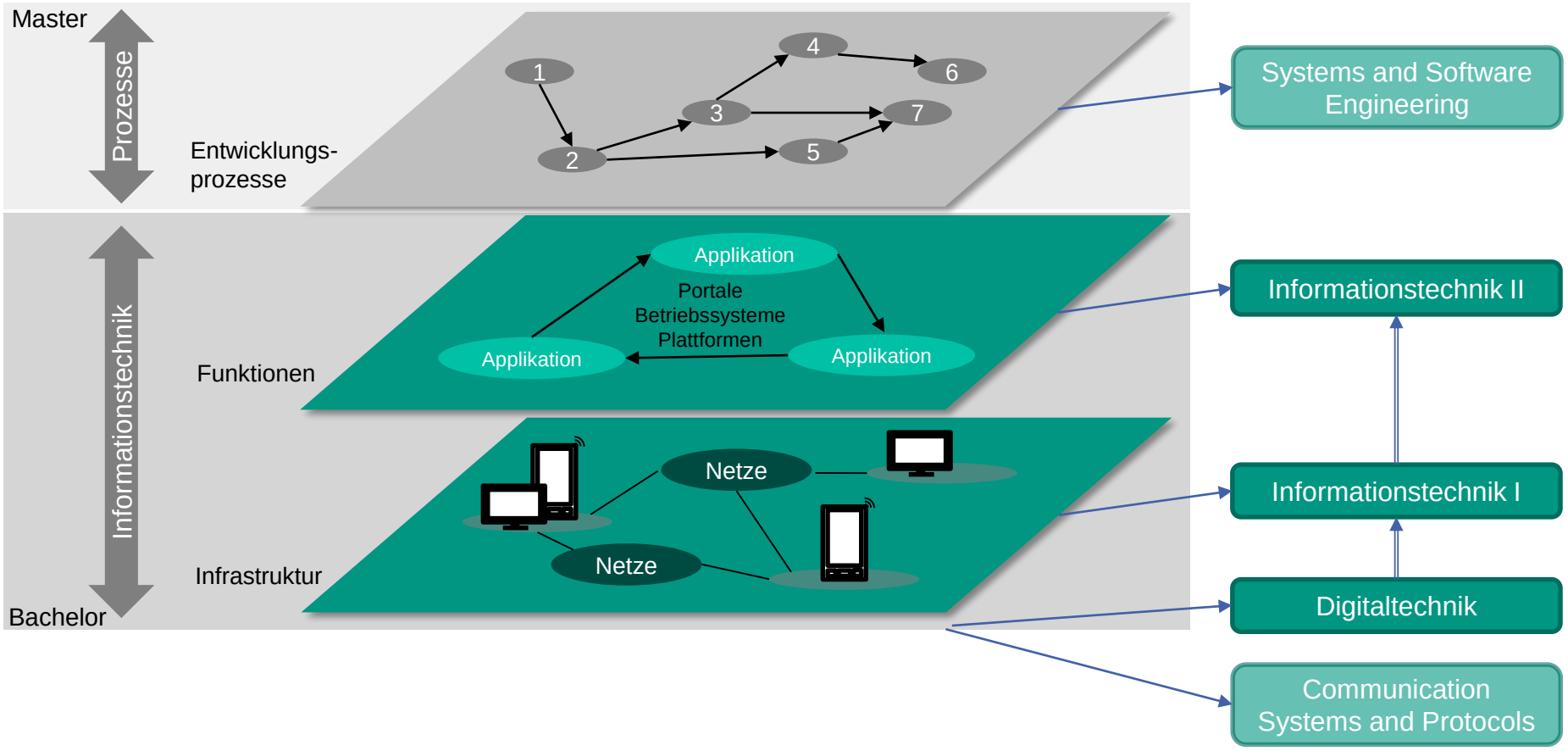
arsnova.eu | Session-ID | Startseite der Session

- **Aufteilung**
- **Klausur**
- **Organisatorisches**

VERANSTALTUNGSINHALT

Veranstaltungsinhalt

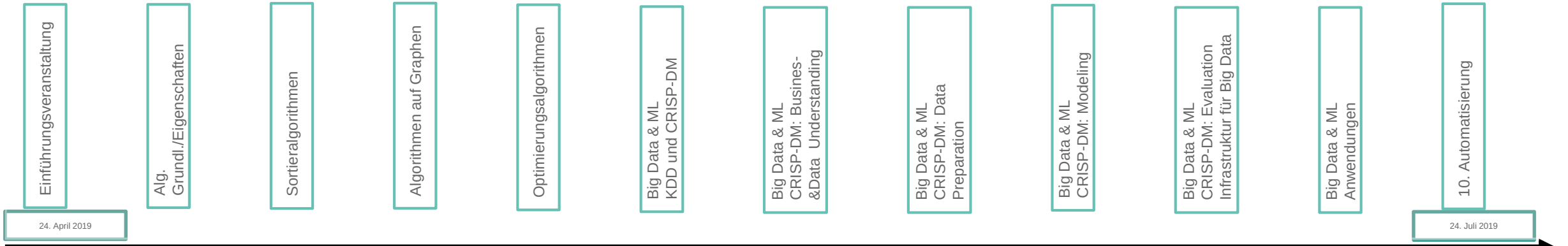
Einordnung in das konsekutive Studium



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



Grundlagen und Algorithmen für einfache Problemstellungen

Generische Algorithmen zur Optimierung von verschiedenen Problemstellungen

Umgang mit großen Datenbeständen

Anwendungs-felder

Bedeutung der Automatisierungs-technik

Grundverständnis für Programmierung vorhanden

Handwerkszeug gelernt → Anwendungsfelder

Grundlagen und Verfahren zur Analyse großer Datenbestände (Methoden und Verfahren zur Lösungen heutiger Problemstellungen)

Veranstaltungsinhalt

Aufteilung der einzelnen Übungseinheiten

1. • Algorithmen: Merkmale, Eigenschaften und Klassen (Sortieralgorithmen)
2. • Sortieralgorithmen, Algorithmen auf Graphen und Optimierungsalgorithmen
3. • Einführung in Big Data Charakteristika
4. • CRISP-DM: Business- und Dataunderstanding
5. • CRISP-DM: Machine Learning Problemstellung; Evaluation
6. • Infrastruktur für Big Data und Data Mining
7. • Anomalieerkennung mit CRISP-DM
8. • Begriffe der IT-Sicherheit und Schutzmechanismen

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 (I)

- BubbleSort Algorithmus:
  sortiert den Eingang nach definierter Vorschrift
  einfacher Algorithmus um Sortierproblem zu lösen
  (später mehr)

- Eigenschaften?

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

```
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;
    }
  }
}
```

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

Verständnisfragen zu Merkmalen und Eigenschaften von Algorithmen

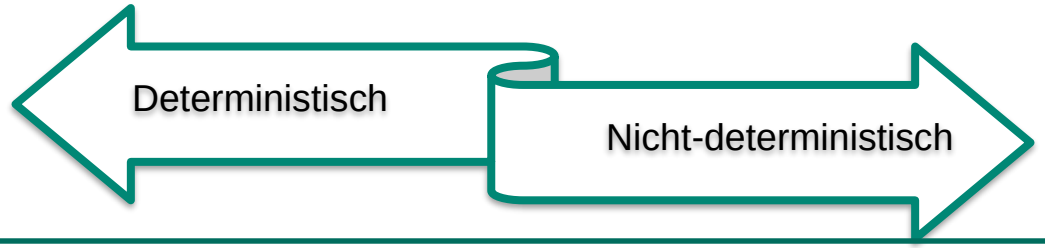
- https://arsnova.eu/mobile/#id/15378598

EIGENSCHAFTEN VON ALGORITHMEN
- LAUFZEITKOMPLEXITÄT

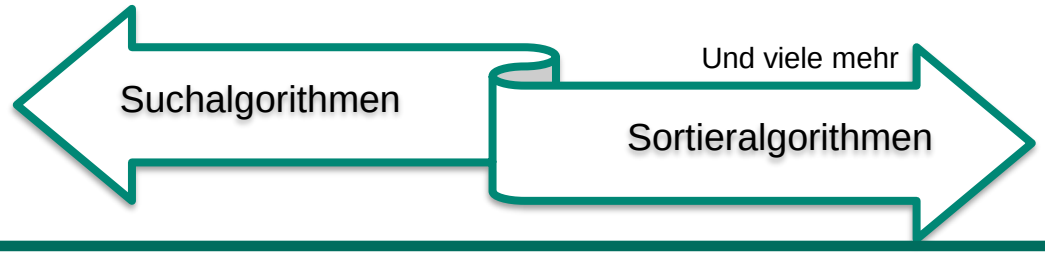
Beschreibung von Algorithmen

Einteilung von Algorithmen in Klassen

- Algorithmen können nach Eigenschaften in Klassen eingeteilt werden
  - Maschinenfähigkeit

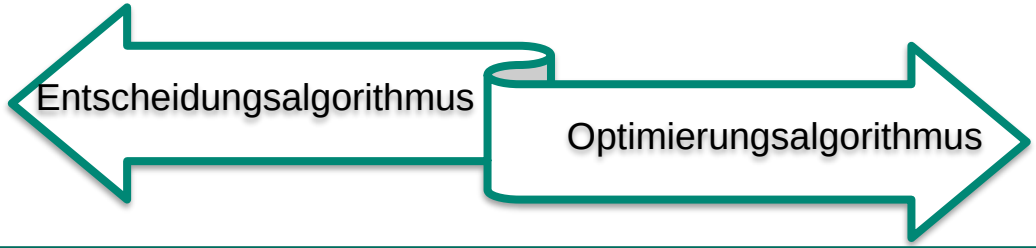


- Anwendung



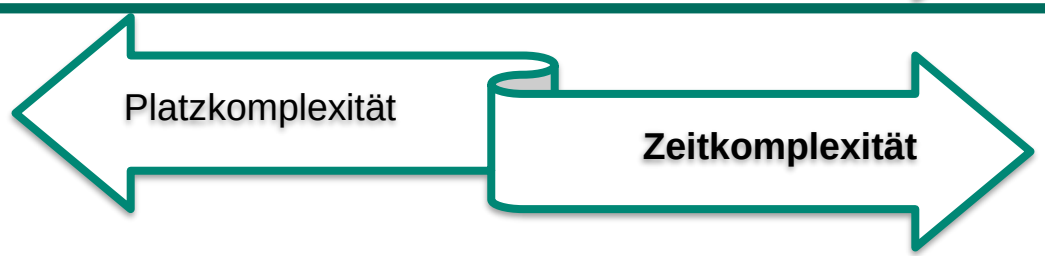
IT2 – Übung 1+2

- Problemstellung



IT2 – Übung 2

- **Komplexität**

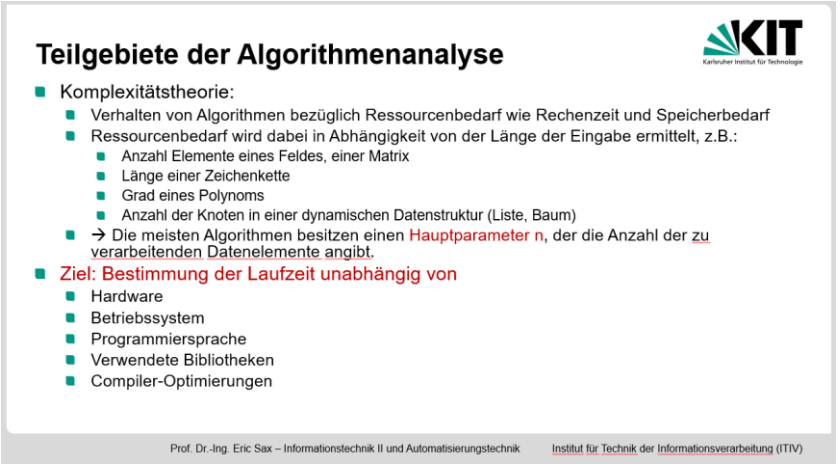


IT2 – Übung 1

Eigenschaften von Algorithmen

Laufzeit und Laufzeitkomplexität

- Zeitkomplexität:
  - „die Anzahl der Rechnerschritte, 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

Relativer Laufzeitaufwand

Feinanalyse

- Ziel: Bestimmung der Laufzeit unabhängig von
  - Hardware
  - Betriebssystem
  - Programmiersprache
  - Verwendete Bibliotheken
  - Compiler-Optimierungen

- nur bei sicherheitsrelevanten Systemen mit harten Realzeitanforderungen durchgeführt

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 *if* oder *while*)	1.5
Indizierung einer Matrix	4.2

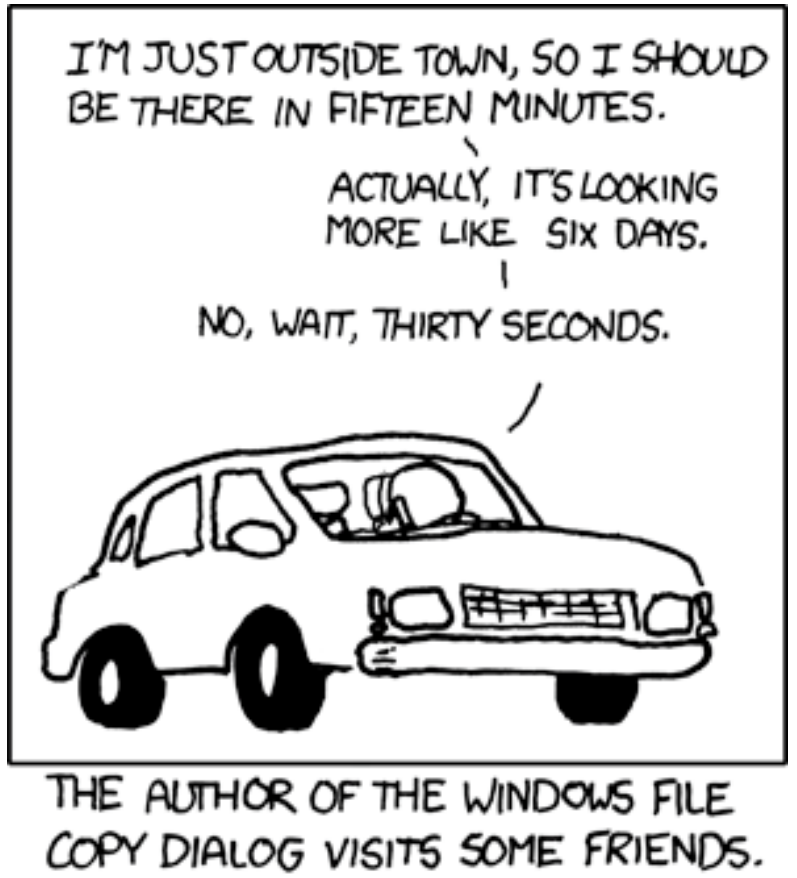
Werte wurden ermittelt mit einem PASCAL-Programm,
Borland PASCAL-Compiler Version 7.0,
auf PC unter Betriebssystem Microsoft XP
mit Prozessor Intel Pentium 4 mit 2GHz Taktfrequenz

- In den meißten 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)

Laufzeitanalyse

Wenn‘s schief geht…

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



Relativer Laufzeitaufwand:

Grobanalyse (Laufzeitkomplexitätsklassen): O-Notation

- Laufzeit von elementaren Operationen

i1 := 0;	**O(1)**

- Laufzeit einer Sequenz von elementaren Operationen

i1 := 0;	**O(1)**	**42∗O(1)=O(1)** Wenn 42 unabhängig von n (Eingabelemente) ist
i2 := 0;	**O(1)**	
…	**…**	
i42 := 0;	**O(1)**	

Asymptotische Laufzeit-komplexität	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 *logarithmus dualis*), sehr günstig und daher erstrebenswert, z.B. *binäre Suche* (siehe Kapitel 6)
$O(n)$	linear	Verdoppelung der Problemgröße bewirkt Verdoppelung der Laufzeit, immer noch zufrieden stellend, z.B. *sequenzielle Suche* (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 *Quicksort* (siehe Kapitel 7)
$O(n^2)$	quadratisch	Verdoppelung der Problemgröße bewirkt Vervierfachung der Laufzeit, ungünstig, z.B. schlechte Sortierverfahren wie *Bubblesort* (siehe Kapitel 7)
$O(n^3)$	kubisch	Verdoppelung der Problemgröße bewirkt Verachtfachung der Laufzeit, sehr unbefriedigend, z.B. einfache *Matrizenmultiplikation*
$O(k^n)$	exponentiell	Verdoppelung der Problemgröße bedeutet Quadrierung (weil $k^{2n} = (k^n)^2$) der Laufzeit, katastrophal, z.B. *Backtracking-* oder *Exhaustionsalgorithmen* (siehe Kapitel 9)

Relativer Laufzeitaufwand

O-Notation

- Laufzeit einer Schleife

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

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

Asymptotische Laufzeit-komplexität	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 *logarithmus dualis*), sehr günstig und daher erstrebenswert, z.B. *binäre Suche* (siehe Kapitel 6)
$O(n)$	linear	Verdoppelung der Problemgröße bewirkt Verdoppelung der Laufzeit, immer noch zufrieden stellend, z.B. *sequenzielle Suche* (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 *Quicksort* (siehe Kapitel 7)
$O(n^2)$	quadratisch	Verdoppelung der Problemgröße bewirkt Vervierfachung der Laufzeit, ungünstig, z.B. schlechte Sortierverfahren wie *Bubblesort* (siehe Kapitel 7)
$O(n^3)$	kubisch	Verdoppelung der Problemgröße bewirkt Verachtfachung der Laufzeit, sehr unbefriedigend, z.B. einfache *Matrizenmultiplikation*
$O(k^n)$	exponentiell	Verdoppelung der Problemgröße bedeutet Quadrierung (weil $k^{2n} = (k^n)^2$) der Laufzeit, katastrophal, z.B. *Backtracking-* oder *Exhaustionsalgorithmen* (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(n²)**
Gleichungssystem lösen	**O(n³)**

Asymptotische Laufzeit-komplexität	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 *logarithmus dualis*), sehr günstig und daher erstrebenswert, z.B. *binäre Suche* (siehe Kapitel 6)
$O(n)$	linear	Verdoppelung der Problemgröße bewirkt Verdoppelung der Laufzeit, immer noch zufrieden stellend, z.B. *sequenzielle Suche* (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 *Quicksort* (siehe Kapitel 7)
$O(n^2)$	quadratisch	Verdoppelung der Problemgröße bewirkt Vervierfachung der Laufzeit, ungünstig, z.B. schlechte Sortierverfahren wie *Bubblesort* (siehe Kapitel 7)
$O(n^3)$	kubisch	Verdoppelung der Problemgröße bewirkt Verachtfachung der Laufzeit, sehr unbefriedigend, z.B. einfache *Matrizenmultiplikation*
$O(k^n)$	exponentiell	Verdoppelung der Problemgröße bedeutet Quadrierung (weil $k^{2n} = (k^n)^2$) der Laufzeit, katastrophal, z.B. *Backtracking-* oder *Exhaustionsalgorithmen* (siehe Kapitel 9)

Relativer Laufzeitaufwand

O-Notation, Laufzeitberechnung, Beispiel

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

`int fakultaet (int zahl)`			
`if (zahl <=1){`	$\sum 1$	O(1)	
`  return 1;`	$\sum 1$	O(1)	4*O(1) **=O(1)**
`}`			
`else {`	$\sum 1$	O(1)	
`  return zahl*fakultaet(zahl-1);`	$\sum 1$	O(1)	
`}}`			

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

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

Laufzeit: **O(1)*O(n)=O(n)**

O-Notation

Zwischenübung

- Bestimmen Sie die Laufzeit des iterativen Algorithmus zur Berechnung der Fakultät

$$f(n) = n!$$

`int fakultaet (int zahl)`			
`if (zahl <=1){`	$\sum 1$	O(1)	4*O(1) **=O(1)**
`  return 1;`	$\sum 1$	O(1)	
`}`			
`else {`	$\sum 1$	O(1)	
`  return zahl*fakultaet(zahl-1);`	$\sum 1$	O(1)	
`}}`			

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

Laufzeit: **O(1)*O(n)=O(n)**

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

O-Notation

Zwischenübung

- Bestimmen Sie die Laufzeit des iterativen Algorithmus zur Berechnung der Fakultät

$$f(n) = n!$$

`int fakultaet (int zahl)`			
`if (zahl <=1){`	$\sum 1$	O(1)	4*O(1) **=O(1)**
`  return 1;`	$\sum 1$	O(1)	
`}`			
`Else {`	$\sum 1$	O(1)	
`  return zahl*fakultaet(zahl-1);`	$\sum 1$	O(1)	
`}}`			

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

Laufzeit: **O(1)*O(n)=O(n)**

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

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

Laufzeit: **O(n)**

Merkmale und Eigenschaften von Algorithmen

Zusammenfassung

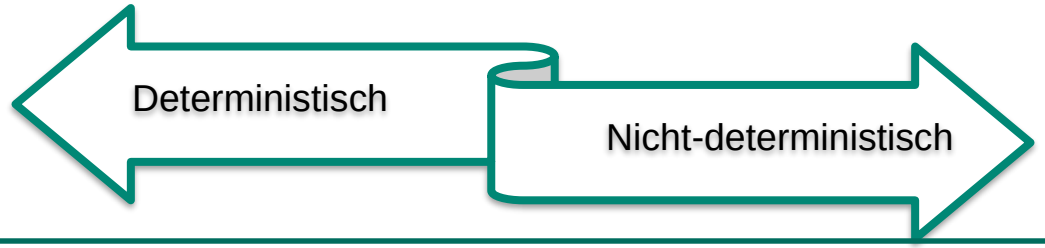
- Ein Algorithmus ist eine genau definierte Handlungsvorschriften zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten
- Ein Algorithmus wird durch seine Merkmale und Eigenschaften beschrieben
  - Stabilität
  - Abbrechbarkeit
  - …
  - Laufzeit
    (Eine grobe 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

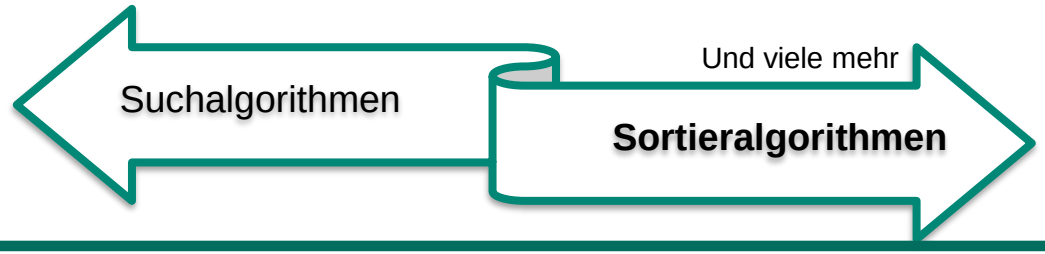
Beschreibung von Algorithmen

Einteilung von Algorithmen in Klassen

- Algorithmen können nach Eigenschaften in Klassen eingeteilt werden
  - Maschinenfähigkeit

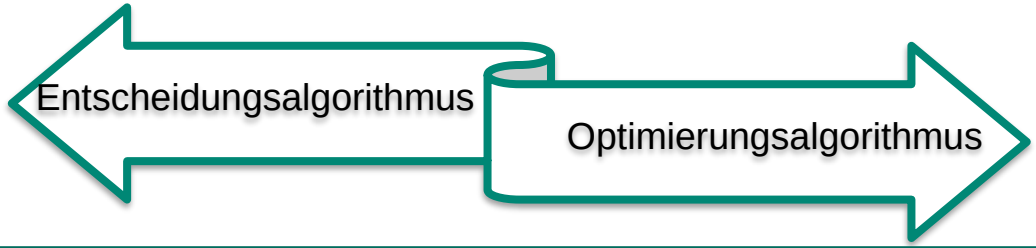


- Anwendung



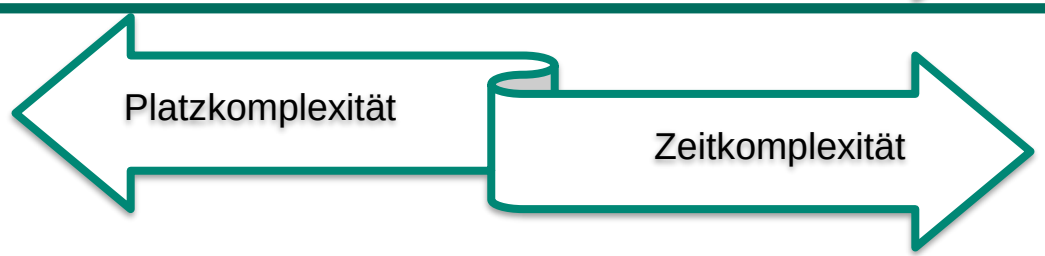
IT2 – Übung 1+2

- Problemstellung



IT2 – Übung 2

- **Komplexität**

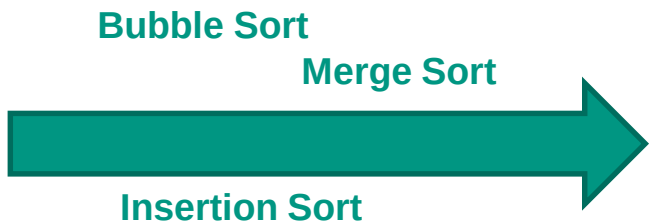


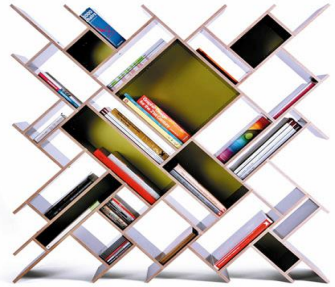
IT2 – Übung 1

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





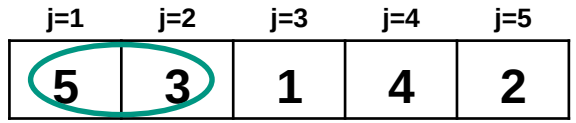
Sortieralgorithmen

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]
```



...

Sortieralgorithmen

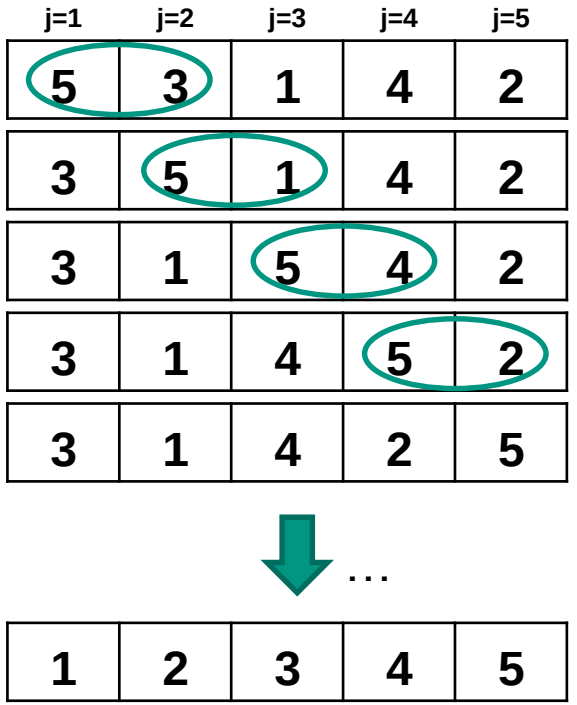
Bubble Sort

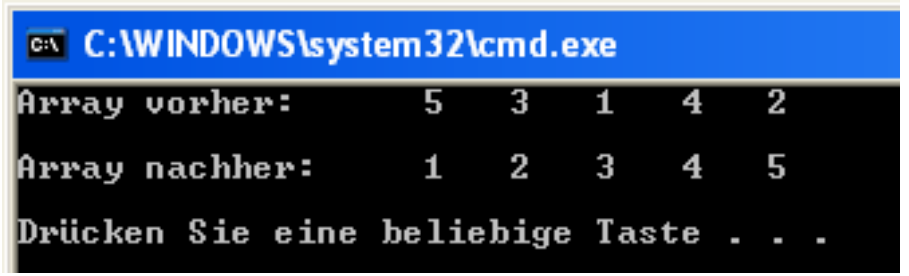
- Einfacher Algorithmus – langsam, aber kaum zusätzlicher Speicherplatz

- 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;
    }
  }
}
```





Sortieralgorithmen

Insertion Sort

- Sortieren durch Einfügen
  - Relativ einfacher Algorithmus

- Vorteile
  - Effizient bei kleinen & bei schon vorsortierten Eingabemengen
  - Einfache Implementierung
  - Stabil
  - Minimal im Speicherverbrauch

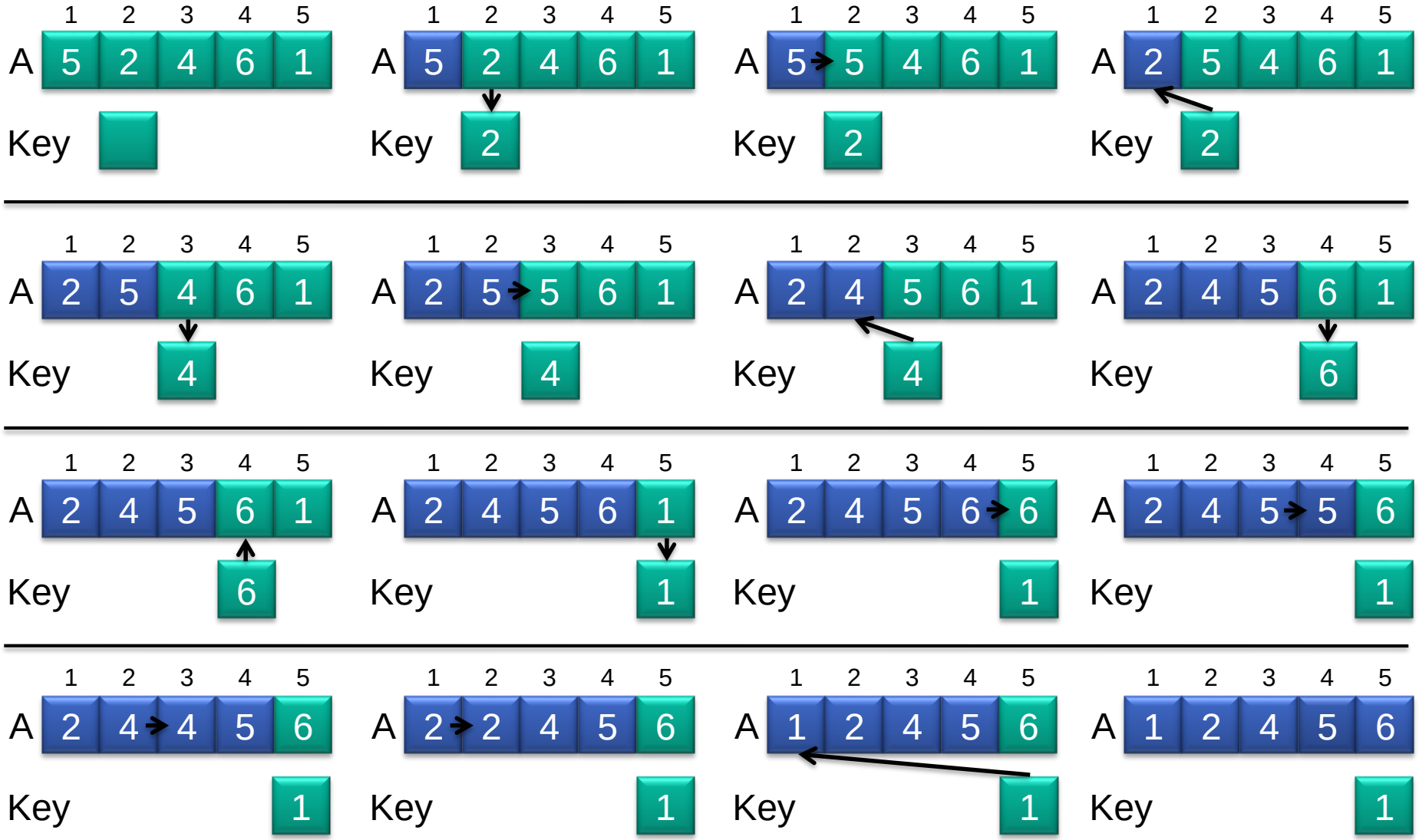
- Nachteile
  - Weniger effizient wie komplizierte Sortierverfahren

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 Laufzeig

Zwischenübung

- Führen Sie eine Laufzeitanalyse des Insertion Sort Algorithmus durch

`for j=2 to length(A){`	$\sum_{i=1}^{n} 1 = n$	O(n)	
`do key= A[j];`	$\sum_{i=1}^{n} (1) - 1 = n - 1$	O(n)	
`I = j-1;`	$\sum_{i=1}^{n} (1) - 1 = n - 1$	O(n)	
`while i>0 and A[i]>key{`	$\sum_{i=1}^{n} n$	O(n²)	
`do A[i+1] = A[i];`	$\sum_{i=1}^{n} n - 1$	O(n²)	**O(n²)**
`I = i-1;`	$\sum_{i=1}^{n} n - 1$	O(n²)	
`}`			
`A[i+1] = key`	$\sum_{i=1}^{n} (1) - 1 = n - 1$	O(n)	
`}`			

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

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