Datenstrukturen, Algorithmen und Programmierung 2

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

1. VO SS 2009 14. April 2009

Kurzvorstellung und Forschungsinteressen

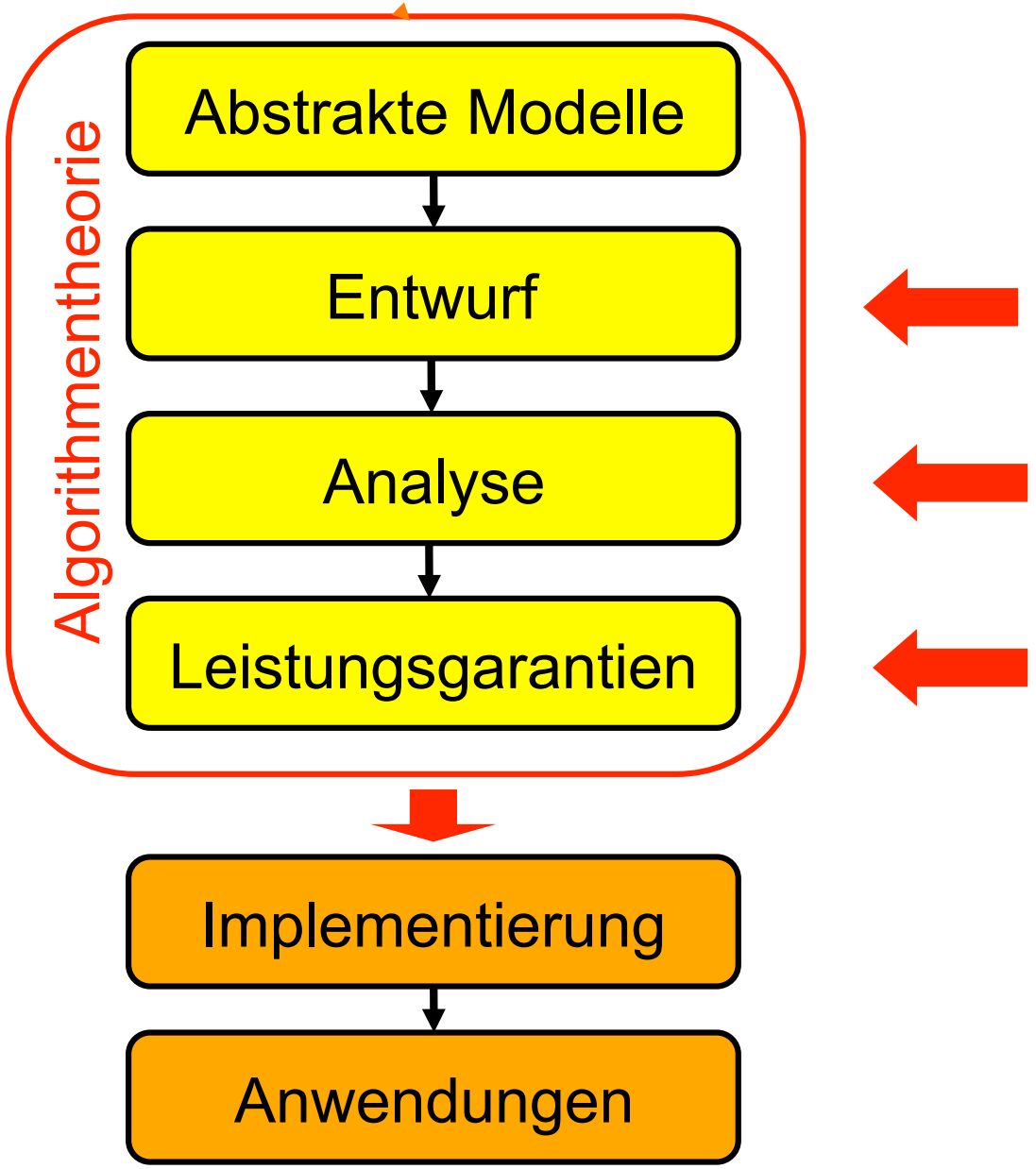
- Algorithmen und Datenstrukturen
- Graphenalgorithmen
- Kombinatorische Optimierung

- Algorithm Engineering
  - Design,
  - theoretische Analyse,
  - Implementierung, und
  - experimentelle Evaluation

  von Algorithmen und Datenstrukturen

anwendungs-orientiert

Algorithmik



Motivation

„Warum soll ich in DAP2 gehen?"

MERKE: DAP2 IST WICHTIG!!!

„Ich kann doch schon programmieren."

ABER NICHT IMMER EFFIZIENT!

Und noch ein Grund:

DAP2 IST TEIL DER ENDNOTE!

Überblick

Einführung

Organisatorisches zur

- Vorlesung
- Übung
- Praktikum

Kapitel 1: Einführung

1.1 Grundbegriffe

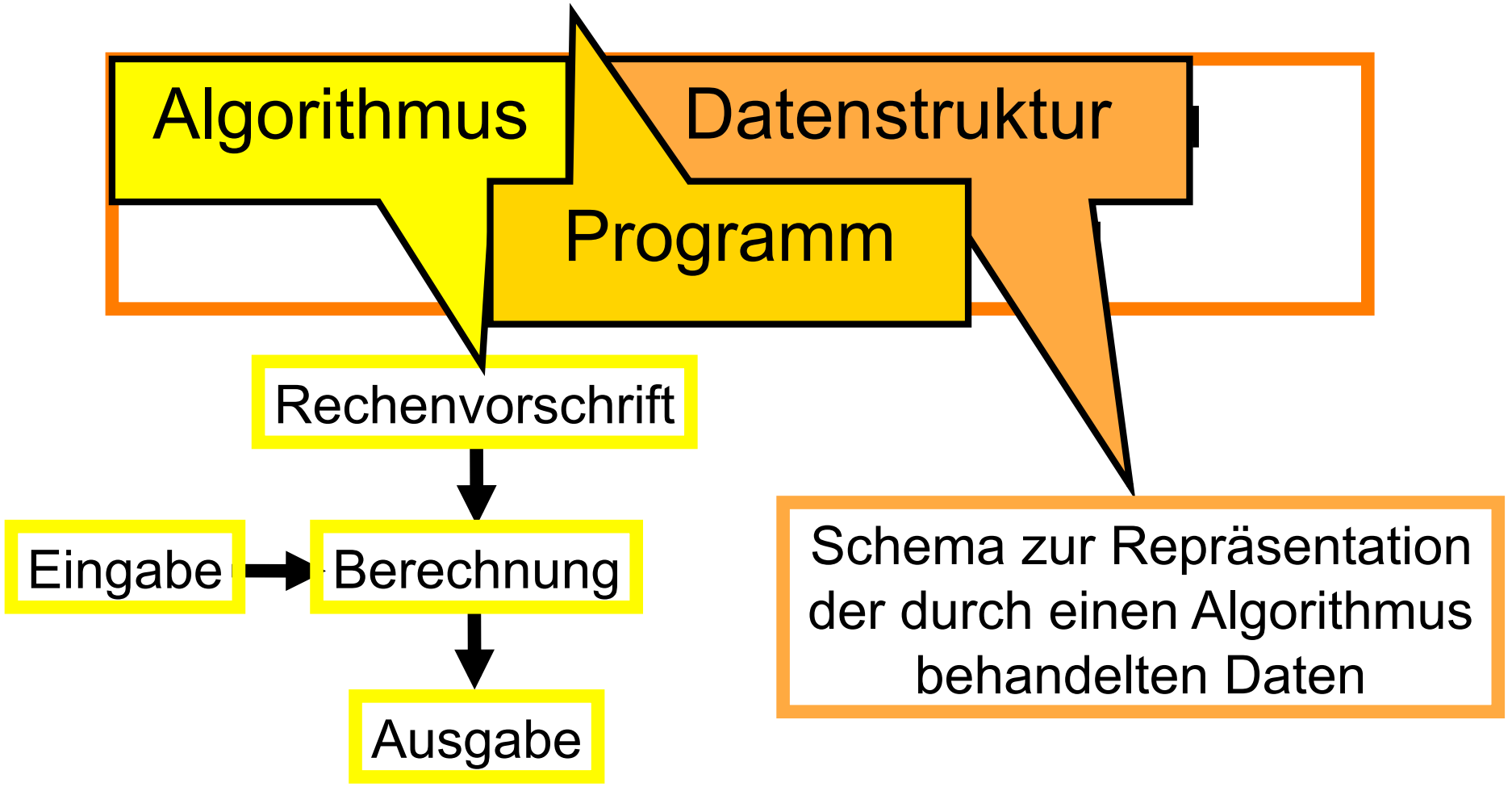
1.2 Beispiel: Sortierproblem

1.3 Analyse von Algorithmen

Konkrete Formulierung abstrakter Algorithmen, die sich auf bestimmte Darstellungen wie Datenstrukturen stützen

=

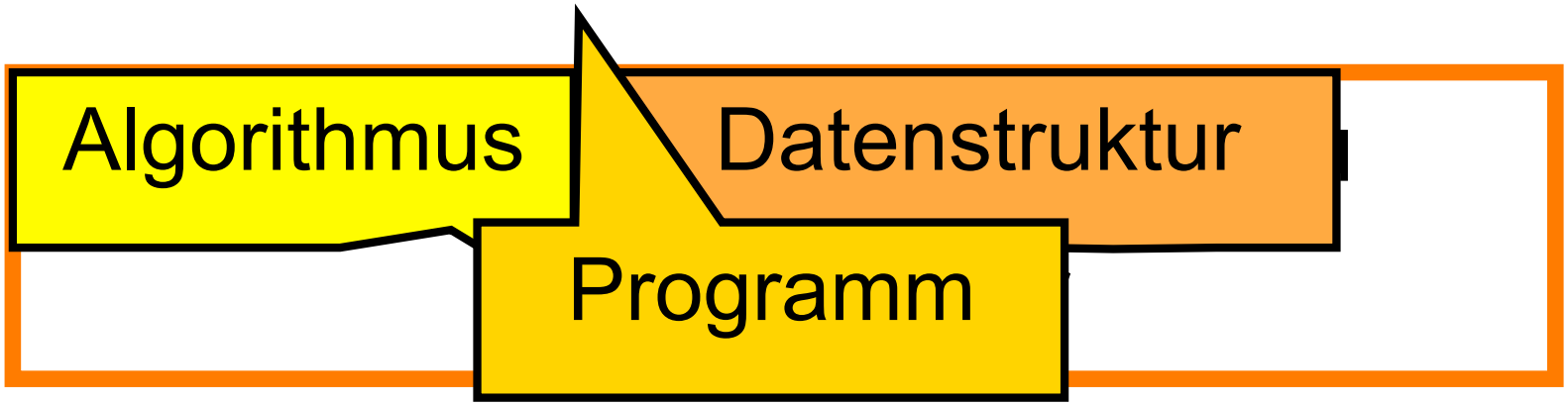
Summe aus Algorithmen und Datenstrukturen



Konkrete Formulierung abstrakter Algorithmen, die sich auf bestimmte Darstellungen wie Datenstrukturen stützen

=

Summe aus Algorithmen und Datenstrukturen



Darstellung

- als Text (Deutsch, Englisch,…)
- als Computerprogramm (Java, C++,…)
- als Hardwaredesign
- als Pseudocode

DAP2

Pseudocode

Vereinfachung real existierender Programmiersprachen

C / Java

```
int tmp = i;
i = j;
j = tmp;
```

Pseudocode

```
vertausche i mit j
```

```
for (int i=0;i<9;i++) {
        float f =A[i] …
}
```

```
for i:=0,…,8 {
    f = A[i] …
}
```

Beispiel: Sortierproblem

Eingabe: Folge von n Zahlen $\langle a_1, a_2, \ldots, a_n \rangle$

Ausgabe: Permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ der Eingabefolge, so dass $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Jede konkrete Zahlenfolge ist eine Instanz des Sortierproblems, z.B.:

$\langle 5,3,11,2,17 \rangle \rightarrow \langle 2,3,5,11,17 \rangle$

Gesucht: Korrekter Algorithmus für Problem

Sortieren durch Einfügen

Eingabe: zu sortierende Zahlenfolge (key)

Ausgabe: sortierte Zahlenfolge

Setze Index k auf 2.-tes Element

Setze Index i auf (k−1).-tes Element

// vor dem k-ten Element ist alles sortiert

// suche den richtigen Platz für k-tes Element

Solange i≠0 und (key(k) < key(i))

    Setze Index i auf (i−1).-tes Element

Platziere k-tes Element zwischen i-tes und (i+1).-tes Element

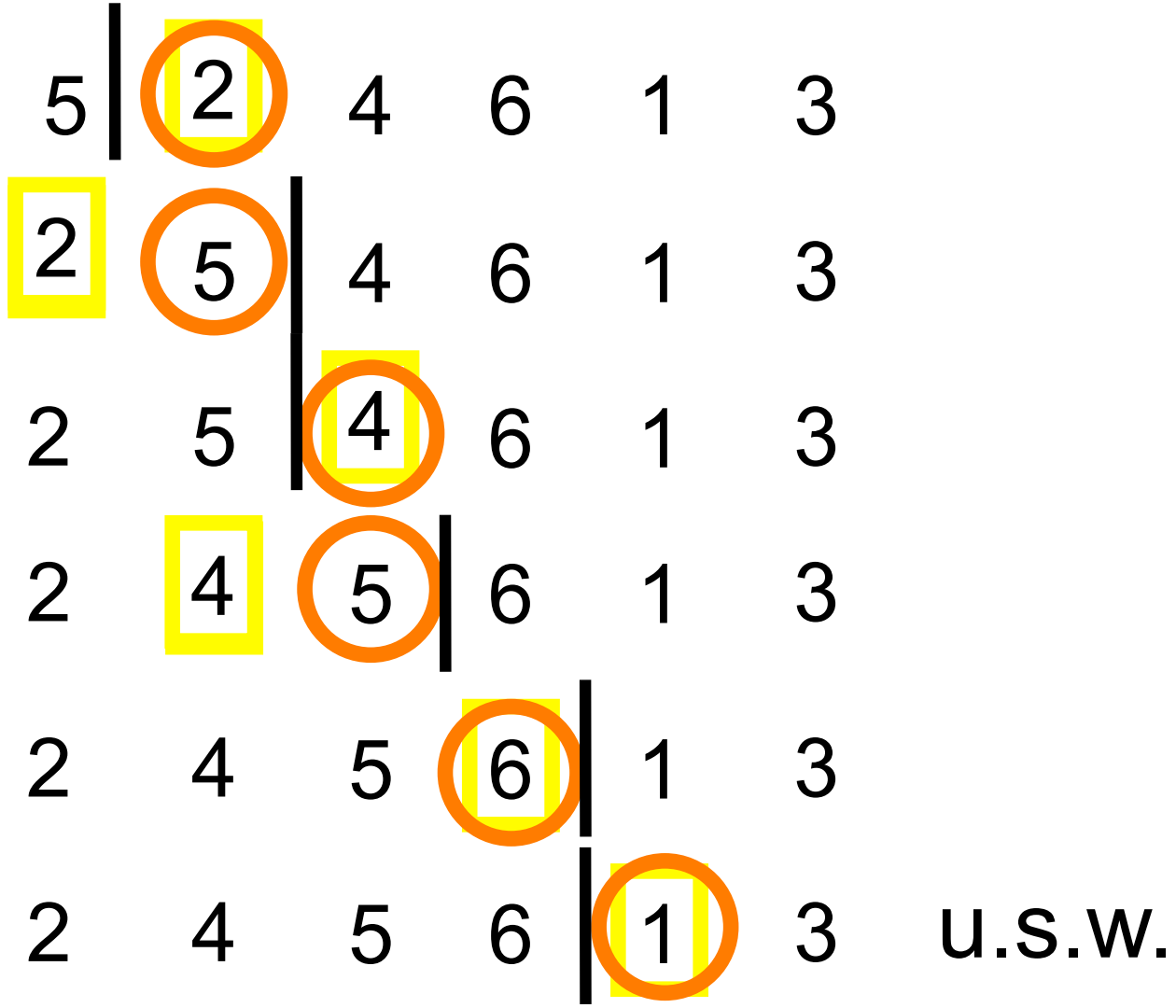
InsertionSort(ref A)

Eingabe/Ausgabe: Zahlenfolge in Feld A[1..n]

```
(1) for k:=2,…,n {
(2)   key:=A[k]
(3)   i:=k
(4)   while i>1 and A[i−1]>key {
(5)     A[i]:=A[i−1]
(6)     i:=i−1
(7)   }
(8)   A[i]:=key
(9) }
```

Ablauf von InsertionSort(ref A)

Zahlenfolge: k= i=



u.s.w.

Analyse von Algorithmen

JETZT AUFPASSEN: sehr sehr WICHTIG!

Maße für die Effizienz eines Algorithmus:

- Laufzeit
- benötigter Speicherplatz
- Anzahl der Vergleichsoperationen
- Anzahl der Datenbewegungen

RAM-Maschinenmodell

Eigenschaften der „Random Access Machine“:

- Es gibt genau einen Prozessor, der das Programm sequentiell abarbeitet
- Jede Zahl, die wir in unserem Programm benutzen paßt in eine Speichereinheit
- Alle Daten liegen in einem direkt zugreifbaren Speicher
- Alle Speicherzugriffe dauern gleich lang
- Alle primitiven Operationen benötigen konstante Zeit

Primitive Operationen

- Zuweisungen (a:=b)
- arithmetische Operationen (Addition, Multiplikation, Modulo-Op., Wurzel-Op.)
- logische Operationen (and, or, not)
- Vergleichsoperationen (≤,≥,≠)
- Befehle zur Ablaufsteuerung (if-then)

Die Laufzeit eines Algorithms ist die Anzahl der bei einer Berechnung durchgeführten primitiven Operationen.

Abmachung

Wir zählen

(1) for (i=1...n) {

(2) ...

(3) }

als

(0) i=1

(1) Ist $i \leq n$? {

(2) ...

(3) i=i+1 }

deswegen

- 0 Mal
- n+1 Mal
- ...
- n Mal

Laufzeit eines Algorithmus

Laufzeit als Funktion der Eingabegröße

– z.B. für Sortierprobleme: n

Laufzeit abhängig von spezieller Instanz,

– z.B. für Sortierprobleme: sortierte Eingabefolge eventuell schneller als für unsortierte Folge

Deswegen: Best-Case, Worst-Case und Average-Case Analyse

Analyse von InsertionSort(ref A)

s_k:Anzahl der Durchführungen von (4)

```
(1) for k:=2,…,n {
(2)    key:=A[k]
(3)    i:=k
(4)    while i>1 and A[i−1]>key {
(5)       A[i]:=A[i−1]
(6)       i:=i−1
(7)    }
(8)    A[i]:=key
(9) }
```

Zeit	Wie oft?
t_1	n
t_2	n-1
t_3	n-1
t_4	$\sum s_k$
t_5	$\sum(s_k-1)$
t_6	$\sum(s_k-1)$
t_7	$\sum(s_k-1)$
t_8	n-1
t_9	n-1

Analyse von InsertionSort(ref A)

s_k:Anzahl der Durchführungen von (4)

$$T(n) = t_1 n + t_2(n-1) + t_3(n-1) + t_4 \sum_{k=2}^{n} s_k + t_5 \sum_{k=2}^{n} (s_k - 1) + t_6 \sum_{k=2}^{n} (s_k - 1) + t_7 \sum_{k=2}^{n} (s_k - 1) + t_8(n-1) + t_9(n-1)$$

→ abhängig von s_k

Zeit	Wie oft?
t_1	n
t_2	n-1
t_3	n-1
t_4	$\sum s_k$
t_5	$\sum(s_k-1)$
t_6	$\sum(s_k-1)$
t_7	$\sum(s_k-1)$
t_8	n-1
t_9	n-1

Best-Case Analyse

Kürzeste mögliche Laufzeit über alle möglichen Eingabe-Instanzen bei vorgegebener Eingabegröße.

Sortierte Folge: $s_k=1$

$$\begin{aligned} T(n) &= (t_1+t_2+t_3+t_4+t_8+t_9)n \\ &\quad -(t_2+t_3+t_4+t_8+t_9) \\ &= an+b \end{aligned}$$

T(n)=Lineare Funktion in n mit Konstanten a und b

Zeit	Wie oft?
t_1	n
t_2	n-1
t_3	n-1
t_4	$\sum s_k$
t_5	$\sum(s_k-1)$
t_6	$\sum(s_k-1)$
t_7	$\sum(s_k-1)$
t_8	n-1
t_9	n-1

Worst-Case Analyse

Längste mögliche Laufzeit über alle möglichen Eingabe-Instanzen bei vorgegebener Eingabegröße.

Umgekehrt sortierte Folge: $s_k=k$

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

Daraus folgt $T(n) = an^2 + bn + c$

$T(n)$=Quadratische Funktion in n mit Konstanten a, b und c

Zeit	Wie oft?
t_1	n
t_2	n-1
t_3	n-1
t_4	$\sum s_k$
t_5	$\sum(s_k-1)$
t_6	$\sum(s_k-1)$
t_7	$\sum(s_k-1)$
t_8	n-1
t_9	n-1

Average-Case Analyse

Durchschnittliche Laufzeit über alle möglichen Eingabe-Instanzen bei vorgegebener Eingabegröße.

Problem: Was ist eine „durchschnittliche“ Eingabe

Hier: $s_k = k/2$

$T(n)$=Quadratische Funktion in n mit Konstanten a, b und c

Laufzeit-Analyse

Genaue Laufzeitberechnung ist sehr aufwändig, deswegen Vereinfachung

Wir betrachten nur die Ordnung der Laufzeit, wobei n als beliebig groß angenommen wird

Wir sagen: „$f(n)=an+b$ ist in $\Theta(n)$“

und „$g(n)=an^2+bn+c$ ist in $\Theta(n^2)$“

Wir sagen: „$f(n)$ wächst linear für große n“

und „$g(n)$ wächst quadratisch für große n“

Formale Definition von Θ: Donnerstag

Organisatorisches zur Vorlesung

Inhalte der Vorlesung

Literatur zur Vorlesung

Organisatorisches zur Vorlesung,
Klausur, Übung und Praktikum

Inhalte der Vorlesung

1. Einführung / Algorithmen-Analyse (O-Notation)
2. Abstrakte Datentypen und Datenstrukturen
3. Sortieralgorithmen
4. Suchalgorithmen (Binärsuche, B- und AVL-Bäume, Skiplisten)
5. Hashing
6. Graphenalgorithmen (BFS, DFS, Zshgskomp, MST, kürzeste Wege)
7. Optimierung (Heuristiken, B&B, Dyn. Prog.)
8. Geometrische Algorithmen

Literatur zur Vorlesung

- **VO-Folien auf Web:**
  - ls11-www.cs.uni-dortmund.de/people/beume/dap2-09/dap2.jsp
- **Skript auf Web (demnächst)**
- **Bücher:**
  - R. Sedgewick: Algorithmen, Pearson Studium 2002, 2. Auflage (oder: Algorithmen in C++)
  - T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein: Algorithmen – eine Einführung, Ausgabe März 2007, Oldenbourg-Verlag
  - T. Ottmann und P. Widmayr: Algorithmen und Datenstrukturen, Spektrum Akademischer Verlag 2002, 4. Auflage

Organisatorisches zu Modul DAP2:

- **Vorlesung** DAP2, 4 SWS
  - Di 12:15-14:00 im Audimax und
  - Do 14:15-16:00 im HS 1, HG II

- **Übung** zu DAP2, 2 SWS
  - 23 kleine Übungsgruppen

Übungstests

viel Zeit einplanen

- **Praktikum** zu DAP2, 2 SWS
  - 18 kleine Übungsgruppen, C++

Modulprüfung und Studienleistungen

- **Klausur:**
  - Klausurtermin: Freitag, 31. Juli 2009
  - Nebentermin: Montag, 5. Oktober 2009
  - 90 Min., Inhalte der Vorlesung, Übung hilfreich!
- **Voraussetzungen für Klausurteilnahme (Bachelor):**
  - Erfolgreiche Teilnahme am DAP1-Praktikum
  - Erfolgreiche Teilnahme an der Übung zu DAP2

- **Weitere Studienleistungen (Bachelor):**
  - Erfolgreiche Teilnahme am Praktikum zu DAP2

Übung zu DAP2

Ablauf

- Übungsblätter:
  - Abgabe Do 14 Uhr (Briefkästen Campus Süd),
  - Ausgabe Do 16 Uhr (homepage)
- Übungsgruppen, ab 20.4.
- Anmeldung **heute ab 14 Uhr bis Do 16.04.2009** über ASSES: http://ess.cs.uni-dortmund.de/ASSESS/
- 2 Übungstests während der Vorlesung (Termine werden noch angekündigt)

Übung zu DAP2

Erfolgreiche Teilnahme

- Anwesenheit (max. 2 mal Fehlen ohne Attest) und aktive Teilnahme bei Übungsgruppen und Präsenzaufgaben
- 4 Blöcke von Übungsblättern,
  mind. 40% der Punkte pro Block
- mind. 1 bestandener Übungstest (aus 2)

Übung zu DAP2

Lernraumbetreuung (ab 20.4.)

- Mo 10.15-12.15, GB4 101
- Di 10.45-11.45, OH14 340
- Mi, 11.00-13.00, GB4 101
- Fr, 12.00-13.00, OH14 340

Ansprechpartner

- Dirk Sudholt, Nicola Beume

Sprechstunde DAP2 (diese Woche)

- 16.4.2009, 16-17 Uhr, Nicola Beume, OH14 233

Praktikum zu DAP2

Ablauf

- Praktikumsblätter: Ausgabe Di 12 Uhr (Homepage), Abgabe in den Praktikumsgruppen (Präsentation)
- Kurzaufgaben (Präsenzaufgaben) und Langaufgaben
- Praktikumsgruppen, ab 21.4.
- Anmeldung **ab 14 Uhr bis Do 16.04.2009** über ASSES: http://ess.cs.uni-dortmund.de/ASSESS/

Praktikum zu DAP2

Erfolgreiche Teilnahme

- Anwesenheit an 11 Terminen (aus 14)
- Ersatztermine bei Feiertagen
- mind. 8 von 14 Punkten der Kurzaufgaben
- mind. 7 von 12 Punkten der Langaufgaben

Praktikum zu DAP2

Lernraumbetreuung (Pools) (ab 21.4.)

- Di 10:15-11:45, OH14 U04
- Mi 12:15-13:45, OH14 U04

Ansprechpartner

- Carsten Gutwenger

Weitere Informationen

- Sprechstunde von Petra Mutzel:
  - Di 14:15 Uhr in OH14, R. 231

- Aktuelle Informationen:
  - ls11-www.cs.tu-dortmund.de/people/beume/dap2-09/dap2.jsp