Hochschule Offenburg
University of Applied Sciences

Algorithmen und Datenstrukturen

Prof. Dr. Klaus Dorer

Organisatorisches

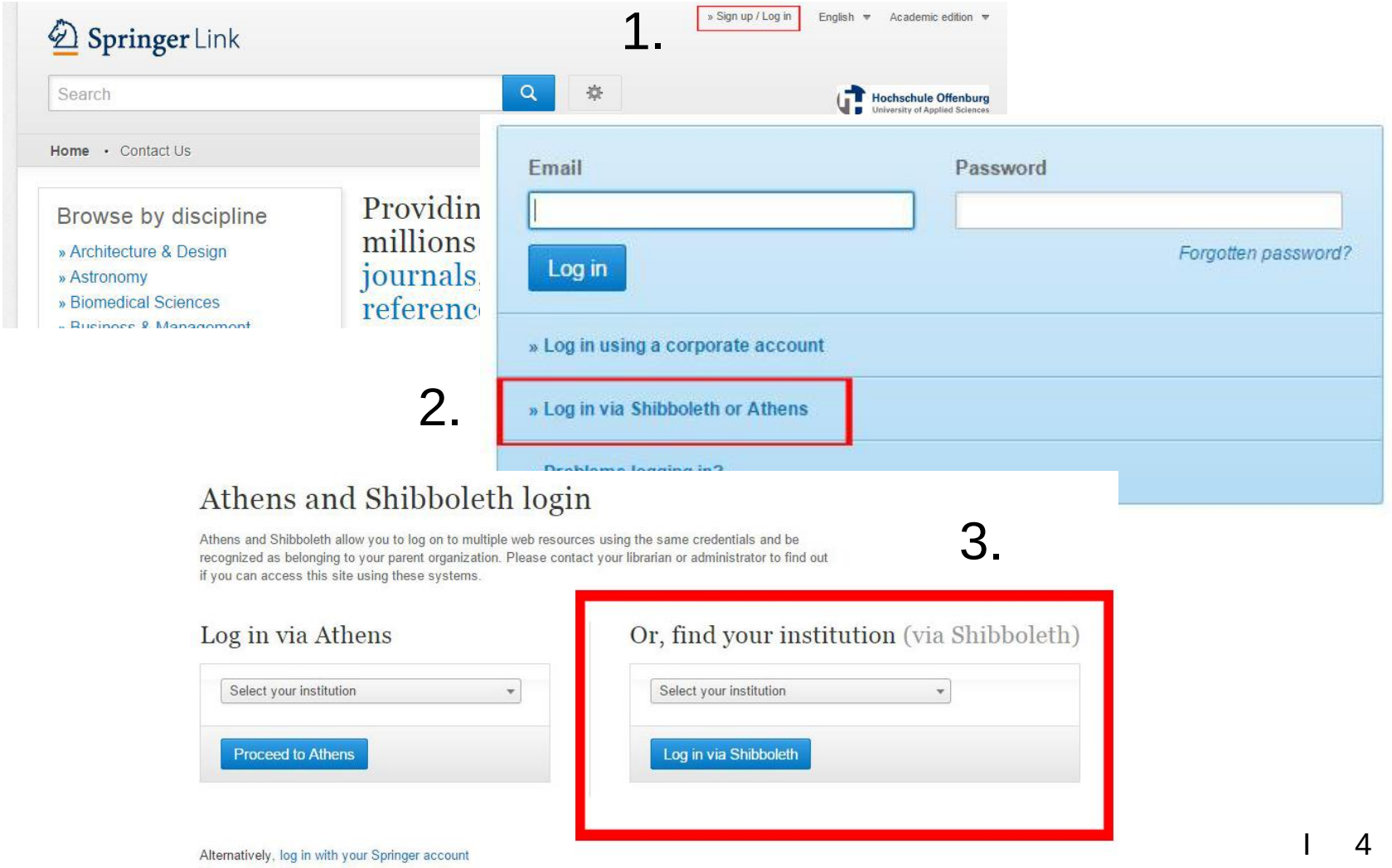
- Vorlesung
  - ((Folien Skript in der Druckerei?))
  - Und auf Moodle (nur bis zum aktuellen Kapitel)
- Labor
  - Einteilung in 2 Gruppen
  - Aufgaben und Termine im Moodle (bitte anmelden)
  - Voraussetzungen zum Bestehen
    - Anwesenheitspflicht
    - Selbständige Bearbeitung
    - 50% der Punkte
    - Bestehen des Labortests

Quellen

- Thomas Ottmann und Peter Widmayer, Algorithmen und Datenstrukturen, 4. Auflage, Spektrum, Berlin, 2002
- R. Baumann, Informatik SII, Band 1, Klett, 1992
- Gerhard Heyer, Vorlesungsfolien Algorithmen und Datenstrukturen II, Uni Leipzig, WS2000/01
- R. Dumke, Einführung Algorithmen und Datenstrukturen, Uni Magdeburg
- S. Trahasch, Vorlesungsfolien Algorithmen und Daten-strukturen, Hochschule Offenburg
- www.wikipedia.de

Buch zur Vorlesung

- Ottmann, Widmayer bei Springer Link erhältlich



Algorithmen

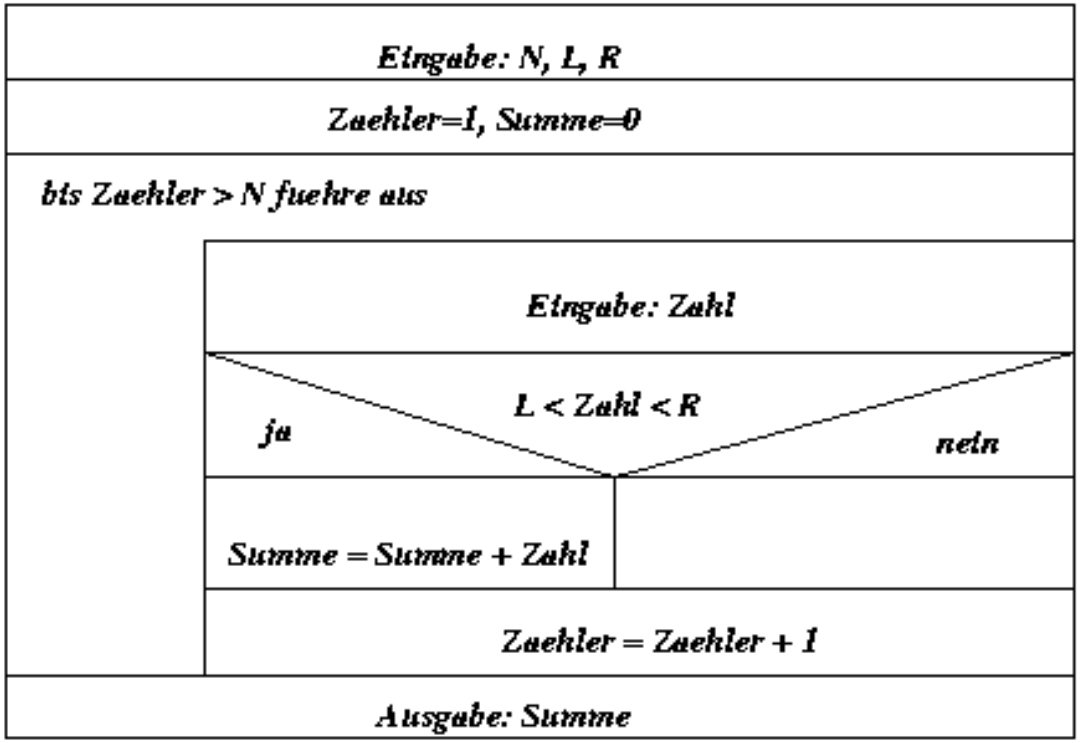
- Unter einem Algorithmus versteht man allgemein eine genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen (Wikipedia)
- Beispiele
  - Kochrezept
  - Aufbauanleitung
  - Spielregeln
  - Sieb des Erathostenes
  - Erster Algorithmus für einen Computer: 1842 von Ada Lovelace für Charles Babbages Analytical Engine
- Ein Algorithmus muss eindeutig, ausführbar und endlich sein

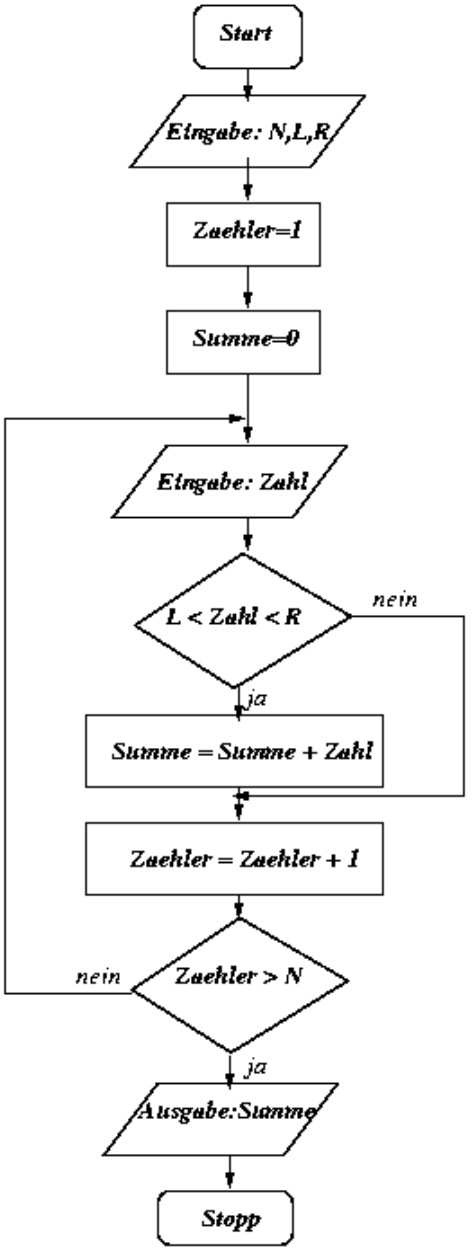
Notation

- Natürliche Sprache
  - Man nehme 125 g Zucker, 3 Eier, 250 g Margarine und rühre...
- Pseudocode
  - read N, L, R
    Zähler = 1
    Summe = 0
    while Zähler <= N
    do read Zahl
        if L < Zahl < R
            then Summe = Summe + Zahl;
        endif
        Zähler = Zähler + 1
    enddo
    write Summe

Notation

- Ablaufdiagramm
- Struktogramm

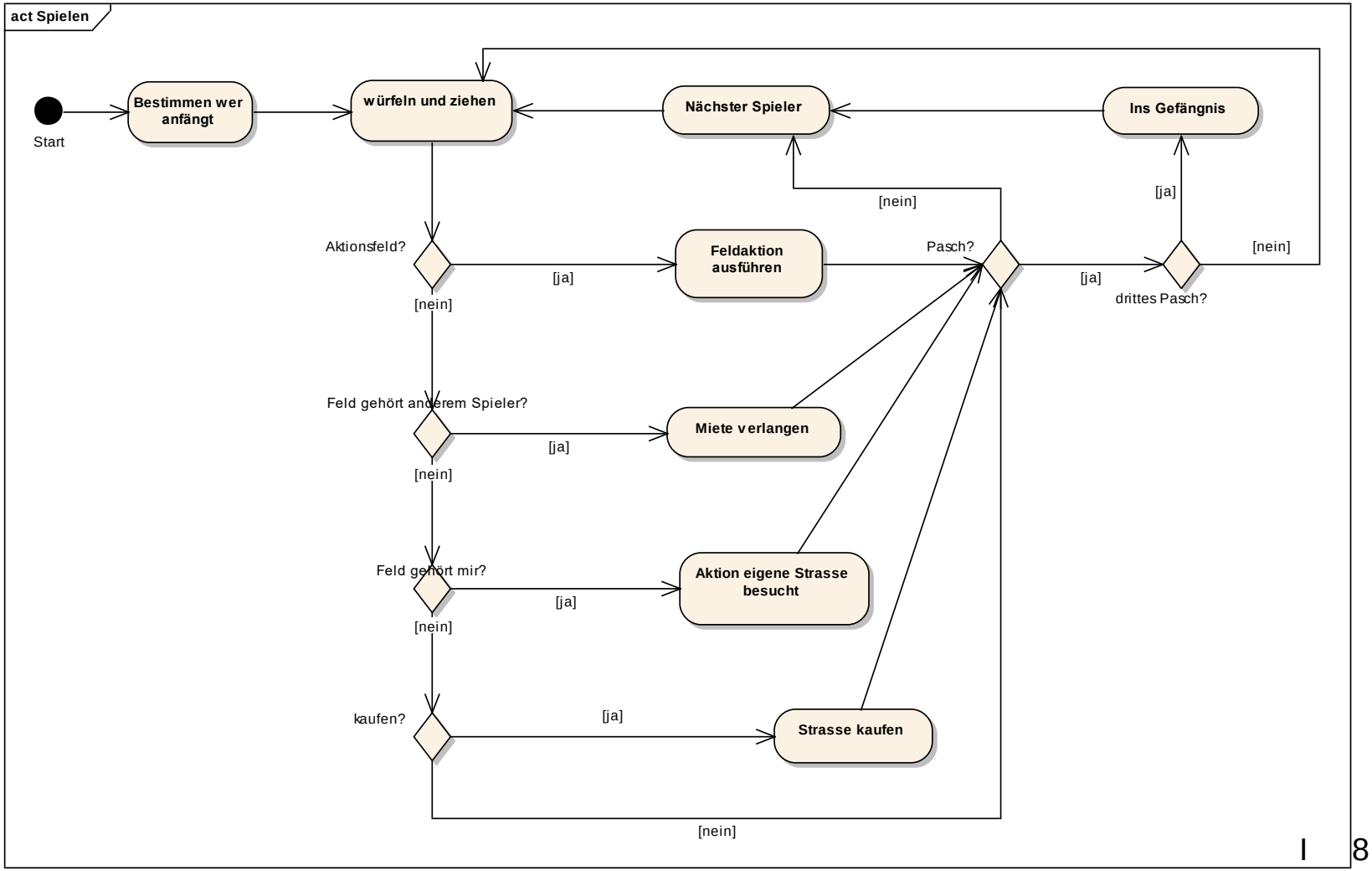




Quelle: Dumke

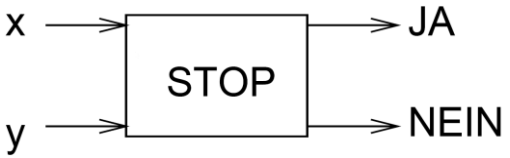
Notation

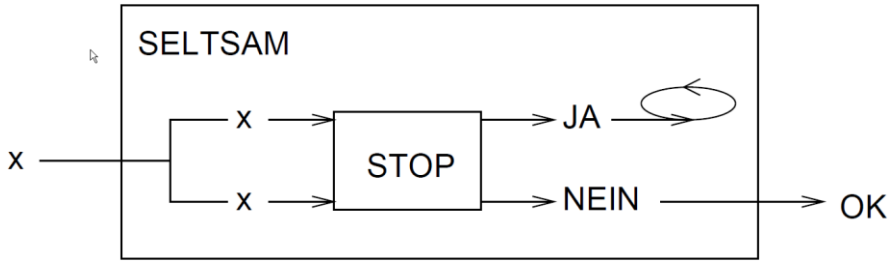
- UML Aktivitätsdiagramm



Eigenschaften

- Berechenbarkeit
  - Gibt es einen korrekten Algorithmus für das Problem?
  - Beispiel Halteproblem: gibt es einen Algorithmus „STOP“, der bei Eingabe von Programm x und Daten y entscheiden kann, ob das Programm x bei Eingabe von y anhält?
  - Nein! Beweisidee:
    - Algorithmus „SELTSAM“ wendet x auf sich selbst an und geht bei Ja in eine Endlosschleife, bei Nein in OK
    - Endet SELTSAM bei der Eingabe von SELTSAM?





Eigenschaften

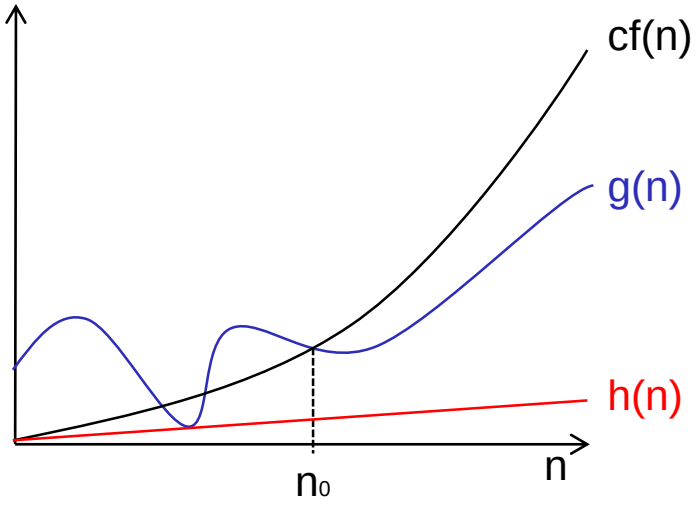
- Korrektheit
  - Ergebnis entspricht der Spezifikation
  - Voraussetzung: Genaue Spezifikation in einem festen Formalismus
    - formale Sprache: eindeutige Syntax, mathematische Semantik (d.h. kein Spielraum für Interpretationen)
    - formale Methode = formale Sprache plus Verfahren (Kalkül) für Beweise, Ableitungen, Transformationen, Ausführungen,…
- Prüfen auf Korrektheit
  - Testen
    - Algorithmus mit Testdaten durchführen
    - “Program testing can be used to show the presence of bugs, but never to show their absence!” (Edsger W. Dijkstra)
  - Verifikation
    - Formaler Beweis der Korrektheit

Eigenschaften

- Vollständigkeit
  - Es wird eine Lösung gefunden, wenn es eine gibt (nicht unbedingt alle)
- Optimalität
  - Wenn eine Lösung gefunden wird ist es die optimale Lösung
  - Gütekriterium für Lösungen notwendig
- Terminierung
  - Algorithmus bricht nach endlich vielen Schritten ab
- Deterministisch
  - Wenn es eine eindeutige Schrittreihenfolge gibt
  - Determiniertes Ergebnis: bei gleicher Eingabe dasselbe Ergebnis
- Komplexität
  - Anzahl der Operationen oder gespeicherten Daten in Abhängigkeit von der Anzahl Eingabedaten
  - Die Optimierung der einen geht häufig auf Kosten der anderen Ressource
  - Interessant ist häufig nur das asymptotische Wachstum für große n

Komplexität

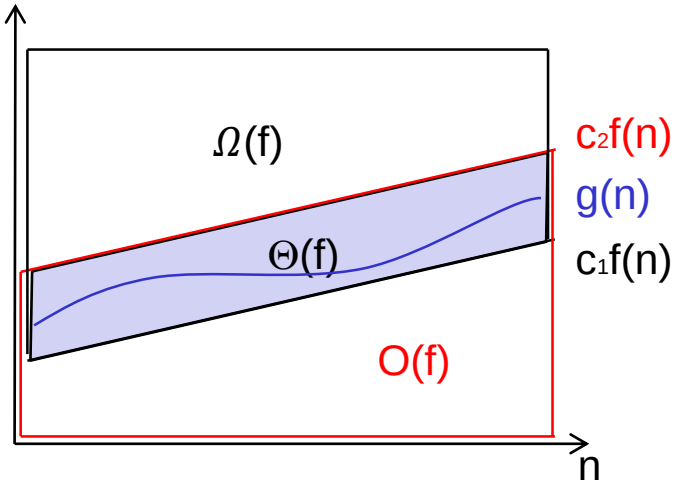
- Obere Schranke wird mithilfe der Groß-Oh Notation angegeben
  - $O(f) = \{g: \mathbb{N} \to \mathbb{N} | \exists c > 0, \exists n_0 > 0, g(n) \leq cf(n) \forall n > n_0\}$



  - Menge aller Funktionen, die nicht stärker als f wachsen
  - Man sagt: g(n) ist in O(f) und schreibt g(n) = O(f)
  - Auch h(n) ist in O(f), wird aber von f schlecht abgeschätzt
  - Beispiel: $5n^2 + 15 = O(n^2)$ denn $5n^2 + 15 < 6n^2$ für $n > 3$
  - Konstante Faktoren und Summanden werden ignoriert

Komplexität

- Untere Schranke analog mit der Groß-Omega Notation
  - $\Omega(f) = \{g: \mathbb{N} \to \mathbb{N} | \exists c > 0, \exists n_0 > 0, g(n) \geq cf(n)\ \forall n > n_0\}$
  - Zeit/Speicher, die/den ein Algorithmus mindestens braucht
  - Beispiel: f: ‚alle n CDs durchsehen' = Ω(n)
- Ist g∈O(f) und g∈Ω(f)
  - dann schreibt man g=Θ(f)
  - „g wächst im Wesentlichen so schnell wie f““

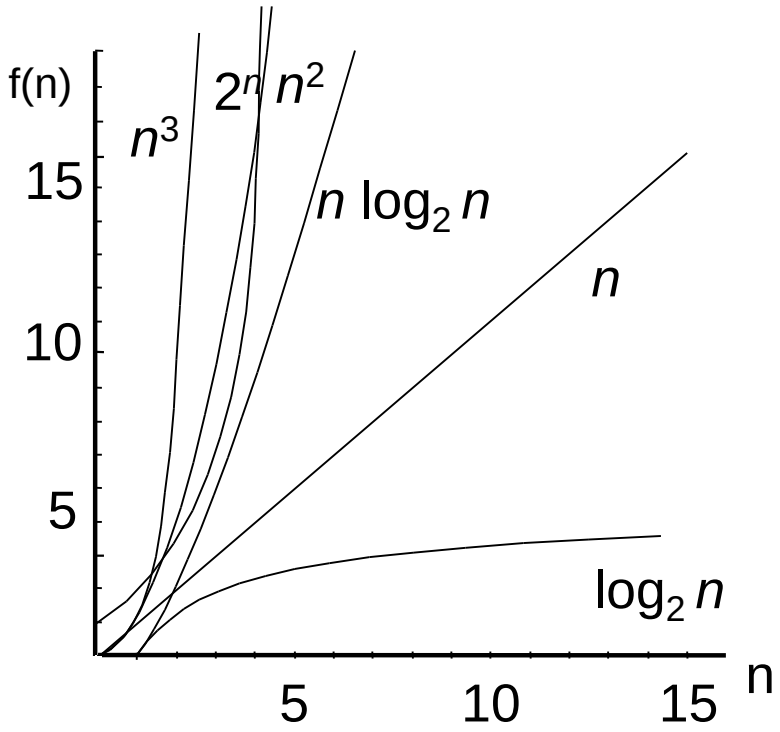


Komplexität

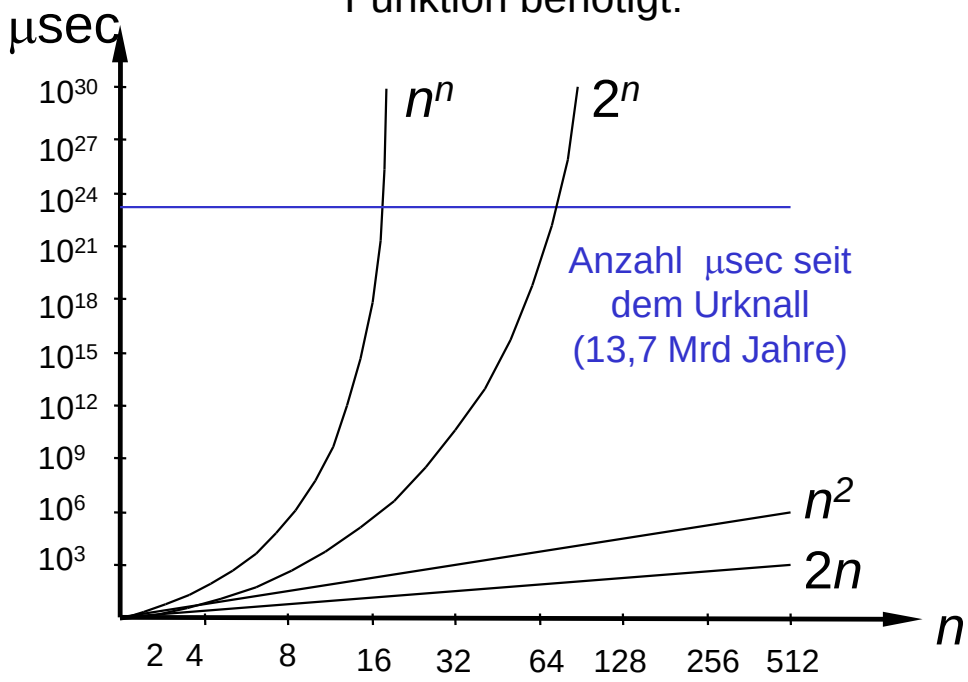
O(1)	Holen des ersten Elements aus einer Menge von Daten
O(log n)	Halbieren einer Datenmenge, wieder halbieren der Hälfte usw.
O(n)	Durchlaufen einer Datenmenge
O(n log n)	Wiederholtes Halbieren einer Datenmenge und Durchlaufen der Hälfte
$O(n^2)$	Einmaliges Durchlaufen einer Datenmenge für jedes Element einer anderen Menge gleicher Mächtigkeit
$O(2^n)$	Generieren aller Teilmengen einer Datenmenge
O(n!)	Generieren aller Permutationen einer Datenmenge

Komplexität

- Wachstum dieser Funktionen
  - Für kleine Werte n



Für große Werte von n und einem Prozessor, der 1 µs = 10^{-6} sec pro elementarer Funktion benötigt:



Komplexität

- Exemplarische Werte

log n	n	n log n	n^2	2^n	n!
1	2	2	4	4	2
2	4	8	16	16	24
3	8	24	64	256	40.320
4	16	64	256	65.536	20.922.789.888.000
5	32	160	1.024	4.294.967.296	2,613 e 35

Komplexität

- Einfache Regeln
  - f = O(f)
  - O(O(f)) = O(f)
  - k * O(f) = O(f) für konstantes k
  - O(f + k) = O(f) für konstantes k
- Additionsregel
  - O(f) + O(g) = O(f + g) = O(max{f; g})
  - Nacheinander Ausführung von zwei Algorithmen
- Multiplikationsregel
  - O(f) * O(g) = O(f * g)
  - Geschachtelte Ausführung zweier Algorithmen

Komplexität

- Zeitaufwand für Programmstück A
- A ist einfache Zuweisung (read, write, . . .):
  - cost(A) = const $\in$ O(1)
- A ist Folge von Anweisungen
  - Additionsregel anwenden
- A ist if-Anweisung
  - if (cond) B;
    - cost(A) = cost(cond) + cost(B)
  - if (cond) B; else C;
    - cost(A) = cost(cond) + max{cost(B); cost(C)}
- A ist eine Schleife
  - while (Bedingung) { Anweisung }
    - cost(A) = #Umläufe * {cost(Anweisung) + cost(Bedingung)}

Beispiel

- Von welcher Größenordnung ist die Laufzeitkomplexität von folgendem Algorithmus?

```
public int doSth(int[] a)
{
  int result = 0;
  for (int i = 0; i < 10000; i++) {
    result += a[i];
  }
  return result;
}
```

Beispiel

- Komplexität?

```
public int doSth(int[] a)
{
  int result = 0;
  int n = a.length;
  for (int i = 0; i < 0.5*n + 1; i++) {
    result += a[i];
  }
  return result;
}
```

Beispiel

- Von welcher Größenordnung ist die Laufzeitkomplexität von folgendem Algorithmus?

```
for (int i = 0; i < a.length - 1; i++) {
    for (int j = 1; j < a.length - i; j++) {
        if (isLessThan(a[j], a[j - 1])) {
            swap(a, j - 1, j);
        }
    }
}
```

- Im besten Fall?
- Im schlechtesten Fall?

Beispiel

- Optimierung von LKWs
  - bilde Vereinigung zweier Mengen ohne Duplikate

```
List allOptimizedTrucks = transportations.getAllTrucks();
List allSpotTrucks = spotMarkt.getAllSpotMarktTrucks();
// avoid duplicates
allOptimizedTrucks.removeAll(allSpotTrucks);
allOptimizedTrucks.addAll(allSpotTrucks);
return allOptimizedTrucks;
```

  - Laufzeit?
  - Verbesserung?

Beispiel

- Komplexität?

```
int doSth(int[] a, int calls, int depth)
{
  int n = a.length;
  calls++;
  if (depth >= n) {
    return calls;
  }
  depth++;
  calls = doSth(a, calls, depth);
  calls = doSth(a, calls, depth);
  return calls;
}
```

Beispiel

- Das Maximum-Subarray-Problem
  - Berechne die maximale Summe einer Teilfolge
  - Bsp.: 31, -41, 59, 26, -53, 58, 97, -93, -23, 84 Lösung?
- Naives Verfahren
  - Größenordnung Laufzeit?

```
int result = 0;
for (int left = 0; left < a.length; left++) {
   for (int right = left; right < a.length; right++) {
      int sum = 0;
      for (int i = left; i <= right; i++){
         sum += a[i];
      }
      if (result < sum) {
         result = sum;
      }
   }
}
return result;
```

Beispiel

- Speichern der aktuellen Summe
  - Größenordnung Laufzeit?
  - Größenordnung Speicher?

```
int result = 0;
for (int left = 0; left < a.length; left++) {
   int oldSum = 0;
   for (int right = left; right < a.length; right++) {
      int sum = oldSum + a[right];
      if (result < sum) {
         result = sum;
      }
      oldSum = sum;
   }
}
return result;
```

Beispiel

- Divide and Conquer
  - Teilt man die Folge in der Mitte, liegt die maximale Teilfolge
    - Ganz links rekursiv lösen
    - An der Trennstelle in $\Theta(r - l)$ Schritten rmax und lmax bestimmen
    - Ganz rechts rekursiv lösen
  - Laufzeit: $\Theta(n \log n)$

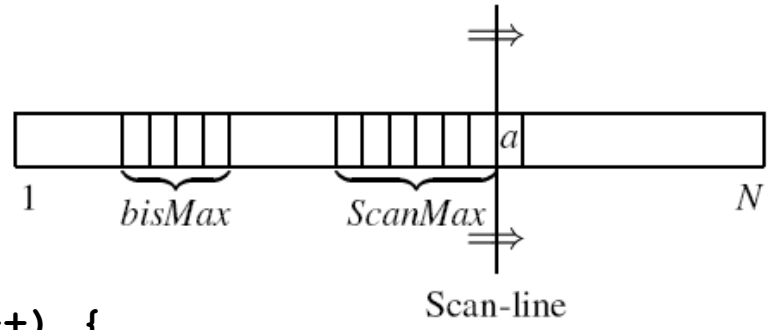
```
private int divideAndConquer(int[] a, int left, int right)
{
   if (left == right) {
      if (a[right] <= 0) { return 0; } else { return a[right]; }
   }

   int middle = (left + right) / 2;
   int sumLeft = divideAndConquer(a, left, middle);
   int sumRight = divideAndConquer(a, middle + 1, right);
   int sumMiddle = rmax(a, left, middle) + lmax(a, middle + 1, right);

   return Math.max(sumLeft, Math.max(sumMiddle, sumRight));
}
```

Beispiel

- Scanline-Prinzip
  - Durchlaufe die Folge von links nach rechts
  - Speichere bisher höchste Teilsumme bisMax und aktuelle Summe ScanMax
  - Größenordnung Laufzeit?




```
int bisMax = 0;
int scanMax = 0;
for (int i = 0; i < a.length; i++) {
    scanMax += a[i];
    if (scanMax <= 0) {
        scanMax = 0;
    }
    if (scanMax > bisMax) {
        bisMax = scanMax;
    }
}
return bisMax;
```

Beispiel

n	Naiv	Divide and Conquer	Scanline
2^2	19	13	9
2^4	815	113	45
2^8	2.829.055	3.841	765
2^{10}	179.481.599	19.457	3.069
2^{15}	$> 5*10^{12}$	950.273	98.301

Übersicht

1.	Einführung	1
2.	Listen	1
3.	Sortieren	2
4.	Suchen	1
5.	Lokale Suche (Optimierung)	2
6.	Bäume	2
7.	Baumsuche	1
8.	Graphen	2
9.	Hashing	1
10.	Klausurvorbereitung	1

Ziele

- Eigenschaften von Algorithmen kennen und einordnen können, insbesondere Komplexität
- Detailkenntnis von
  - Klassischen Algorithmen der Informatik
  - Klassischen Datenstrukturen
  - Modernen Algorithmen
- Zusammenhang von Algorithmus und Datenstruktur verstehen
- In der Lage sein, selbst effiziente Algorithmen und Datenstrukturen zu entwickeln