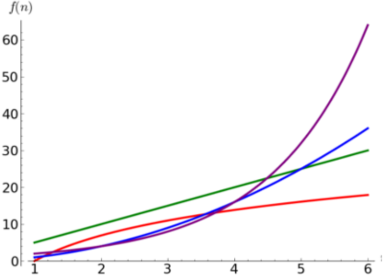
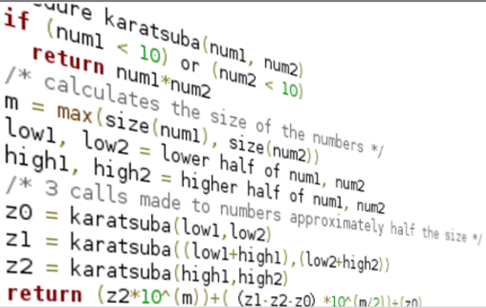
Karlsruher Institut für Technologie

# Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 26. April 2013

Institut für theoretische Informatik, Prof. Sanders



# Vorstellung

Zu meiner Person:

- Simon Bischof
- Bachelor Informatik
- 6. Semester

# Vorstellung

Zu meiner Person:

- Simon Bischof
- Bachelor Informatik
- 6. Semester

# Kontakt

- E-Mail: simon.bischof2@student.kit.edu
- bei inhaltlichen und sonstigen Fragen zu Algorithmen 1
- Liste zum Eintragen der Mailadresse
- für kurzfristige Informationen
- Folien sind auf https://github.com/ratefuchs/Algo1-Tut-SS13

# Kontakt

- E-Mail: simon.bischof2@student.kit.edu
- bei inhaltlichen und sonstigen Fragen zu Algorithmen 1
- Liste zum Eintragen der Mailadresse
- für kurzfristige Informationen
- Folien sind auf
  `https://github.com/ratefuchs/Algo1-Tut-SS13`

# Kontakt

- E-Mail: simon.bischof2@student.kit.edu
- bei inhaltlichen und sonstigen Fragen zu Algorithmen 1
- Liste zum Eintragen der Mailadresse
- für kurzfristige Informationen
- Folien sind auf `https://github.com/ratefuchs/Algo1-Tut-SS13`

# Kontakt

- E-Mail: simon.bischof2@student.kit.edu
- bei inhaltlichen und sonstigen Fragen zu Algorithmen 1
- Liste zum Eintragen der Mailadresse
- für kurzfristige Informationen
- Folien sind auf https://github.com/ratefuchs/Algo1-Tut-SS13

# Übungsbetrieb

Es gibt Übungsblätter:

- Übungsblattabgabe in Zweiergruppen
- Nur falls im gleichen Tutorium!
- Ausgabe Mittwochs, Abgabe Freitag 9 Tage später
- **Immer mit dem gleichen Partner abgeben!**
- **Plagiate werden mit 0 Punkten bewertet!**

# Übungsbetrieb

Es gibt Übungsblätter:

- Übungsblattabgabe in Zweiergruppen
- Nur falls im gleichen Tutorium!
- Ausgabe Mittwochs, Abgabe Freitag 9 Tage später
- **Immer mit dem gleichen Partner abgeben!**
- **Plagiate werden mit 0 Punkten bewertet!**

# Übungsbetrieb

Es gibt Übungsblätter:

- Übungsblattabgabe in Zweiergruppen
- Nur falls im gleichen Tutorium!
- Ausgabe Mittwochs, Abgabe Freitag 9 Tage später
- **Immer mit dem gleichen Partner abgeben!**
- **Plagiate werden mit 0 Punkten bewertet!**

# Übungsbetrieb

Es gibt Übungsblätter:

- Übungsblattabgabe in Zweiergruppen
- Nur falls im gleichen Tutorium!
- Ausgabe Mittwochs, Abgabe Freitag 9 Tage später
- **Immer mit dem gleichen Partner abgeben!**
- **Plagiate werden mit 0 Punkten bewertet!**

# Beschriftung

- Namen und Matrikelnummern (bei Partnerabgabe für beide)
- Nummer des Übungsblatts, Name des Tutors
- rechts oben groß: Nummer des Tutoriums! (mein Tut: 9)
- gut zusammenheften
- **Zuwiderhandelnde haben keinen Punktanspruch!**

# Vorrechnen an der Tafel

- Vorrechnen von Aufgaben des letzten Übungsblatts
- Im Mittel zwei Studenten pro Woche
- Jeder darf maximal einmal vorrechnen (erstes Mal zählt)
- Gibt maximal 6 Übungspunkte

# Was bringt das euch?

- Punkte aus Übungsblättern
- \+ Punkte aus Mittsemesterklausur (Umfang etwa 2 ÜB)
- \+ Punkte aus dem Vorrechnen
- \+ **Zusatz-**Punkte aus Zusatzaufgaben
- $\Rightarrow$ Bonuspunkte auf bestandene Klausur
- (25%$\rightarrow$1 Punkt, 50%$\rightarrow$2, 75%$\rightarrow$3)
- Übungsblätter sind gute Übung für Klausur
- $\Rightarrow$ Blätter auch unabhängig vom Bonus machen

# Was bringt das euch?

- Punkte aus Übungsblättern
- + Punkte aus Mittsemesterklausur (Umfang etwa 2 ÜB)
- + Punkte aus dem Vorrechnen
- + **Zusatz-**Punkte aus Zusatzaufgaben
- $\Rightarrow$ Bonuspunkte auf bestandene Klausur
- (25%$\rightarrow$1 Punkt, 50%$\rightarrow$2, 75%$\rightarrow$3)
- Übungsblätter sind gute Übung für Klausur
- $\Rightarrow$ Blätter auch unabhängig vom Bonus machen

# zum Tutorium

- Hilfe bei Verständnisproblemen
- angeleitetes Aufgabenlösen
- Fragen/Vorschläge/Anmerkungen willkommen!

# zum Tutorium

- Hilfe bei Verständnisproblemen
- angeleitetes Aufgabenlösen
- Fragen/Vorschläge/Anmerkungen willkommen!

# zum Tutorium

- Hilfe bei Verständnisproblemen
- angeleitetes Aufgabenlösen
- Fragen/Vorschläge/Anmerkungen willkommen!

Was ist ein Algorithmus?

# Pseudocode

- Vereinfachte Programmiersprache
- if, else, while, repeat . . . until, for, . . .
- Blöcke durch Einrückung (vgl. Java: "{" und "}")
- Zuweisung mit ":=", Kommentar "//"
- Tupelschreibweise: (c, s) = a+b+c
- in Assertions beliebige Mathe-Ausdrücke:
  $\{i \geq 2 \wedge \neg\exists a, b \geq 2 : i = a \cdot b\}$

# Pseudocode

- Vereinfachte Programmiersprache
- if, else, while, repeat . . . until, for, . . .
- Blöcke durch Einrückung (vgl. Java: "{" und "}")
- Zuweisung mit ":=", Kommentar "//"
- Tupelschreibweise: (c, s) = a+b+c
- in Assertions beliebige Mathe-Ausdrücke:
  $\{i \geq 2 \land \neg\exists a, b \geq 2 : i = a \cdot b\}$

# Pseudocode

- Vereinfachte Programmiersprache
- if, else, while, repeat . . . until, for, . . .
- Blöcke durch Einrückung (vgl. Java: "{" und "}")
- Zuweisung mit ":=", Kommentar "//"
- Tupelschreibweise: (c, s) = a+b+c
- in Assertions beliebige Mathe-Ausdrücke:
  $\{i \geq 2 \wedge \neg\exists a, b \geq 2 : i = a \cdot b\}$

# Pseudocode

- Vereinfachte Programmiersprache
- if, else, while, repeat ... until, for, ...
- Blöcke durch Einrückung (vgl. Java: "{" und "}")
- Zuweisung mit ":=", Kommentar "//"
- Tupelschreibweise: (c, s) = a+b+c
- in Assertions beliebige Mathe-Ausdrücke: $\{i \geq 2 \land \neg\exists a, b \geq 2 : i = a \cdot b\}$

# Pseudocode

- Vereinfachte Programmiersprache
- if, else, while, repeat . . . until, for, . . .
- Blöcke durch Einrückung (vgl. Java: "{" und "}")
- Zuweisung mit ":=", Kommentar "//"
- Tupelschreibweise: (c, s) = a+b+c
- in Assertions beliebige Mathe-Ausdrücke:
  $\{i \geq 2 \wedge \neg\exists a, b \geq 2 : i = a \cdot b\}$

# Wiederholung: Die O-Notation

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "$g$ wächst höchstens so schnell wie $f$"
- Oft schreibt man statt z.B. $2n \in O(n)$ auch $2n = O(n)$ (ist allerdings eher unschön)
- $o(f(n)) = \{g(n) : \forall c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) < c \cdot f(n)\}$
- "$g$ wächst langsamer als $f$"
- Außerdem: $\Omega$ (mindestens), $\omega$ (schneller), $\Theta$ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

# Wiederholung: Die O-Notation

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "$g$ wächst höchstens so schnell wie $f$"
- Oft schreibt man statt z.B. $2n \in O(n)$ auch $2n = O(n)$ (ist allerdings eher unschön)
- $o(f(n)) = \{g(n) : \forall c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) < c \cdot f(n)\}$
- "$g$ wächst langsamer als $f$"
- Außerdem: $\Omega$ (mindestens), $\omega$ (schneller), $\Theta$ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

# Wiederholung: Die O-Notation

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "*g* wächst höchstens so schnell wie *f*"
- Oft schreibt man statt z.B. $2n \in O(n)$ auch $2n = O(n)$ (ist allerdings eher unschön)
- $o(f(n)) = \{g(n) : \forall c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) < c \cdot f(n)\}$
- "*g* wächst langsamer als *f*"
- Außerdem: $\Omega$ (mindestens), $\omega$ (schneller), $\Theta$ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

# Wiederholung: Die O-Notation

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "$g$ wächst höchstens so schnell wie $f$"
- Oft schreibt man statt z.B. $2n \in O(n)$ auch $2n = O(n)$ (ist allerdings eher unschön)
- $o(f(n)) = \{g(n) : \forall c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) < c \cdot f(n)\}$
- "$g$ wächst langsamer als $f$"
- Außerdem: $\Omega$ (mindestens), $\omega$ (schneller), $\Theta$ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

# Wiederholung: Die O-Notation

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "$g$ wächst höchstens so schnell wie $f$"
- Oft schreibt man statt z.B. $2n \in O(n)$ auch $2n = O(n)$ (ist allerdings eher unschön)
- $o(f(n)) = \{g(n) : \forall c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) < c \cdot f(n)\}$
- "$g$ wächst langsamer als $f$"
- Außerdem: $\Omega$ (mindestens), $\omega$ (schneller), $\Theta$ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

# Beispiel: Bubble-Sort

```
function bubbleSort(a[0..n-1])
for j:=0 to n-1
  for i:=0 to n-2
    if a[i] > a[i+1]
      (a[i],a[i+1]) := (a[i+1],a[i])
```

Laufzeit ist $\Theta(n^2)$. Geht das noch besser?

# Beispiel: Bubble-Sort

```
function bubbleSort(a[0..n-1])
for j:=0 to n-1
  for i:=0 to n-2
    if a[i] > a[i+1]
      (a[i],a[i+1]) := (a[i+1],a[i])
```

Laufzeit ist $\Theta(n^2)$. Geht das noch besser?

# Beispiel: Bubble-Sort

```
function bubbleSort2(a[0..n-1])
  repeat
    swapped := false
    for i:=0 to n-1
      if a[i] > a[i+1]
        (a[i],a[i+1]) := (a[i+1],a[i])
        swapped := true
  until swapped = false
```

Laufzeit ist im worst (und average) case $\Theta(n^2)$, im best case aber $\Theta(n)$.

# Beispiel: Bubble-Sort

```
function bubbleSort2(a[0..n-1])
  repeat
    swapped := false
    for i:=0 to n-1
      if a[i] > a[i+1]
        (a[i],a[i+1]) := (a[i+1],a[i])
        swapped := true
  until swapped = false
```

Laufzeit ist im worst (und average) case $\Theta(n^2)$, im best case aber $\Theta(n)$.

# Aufgaben zum O-Kalkül

Zeige oder widerlege:

1. $10n^2 + 5n^2 \in \omega(n^2)$
2. $\log_a(n) \in \Theta(\log_b(n))$ $(a, b > 1)$
3. $n^2 \in o(n^2 \log(n))$
4. $(\log(n))^5 \in o(n)$
5. $2^n \in o(n^2)$
6. $n! \in \omega(2^n)$

# Das Master-Theorem

Gegeben sei eine Rekurrenz-Gleichung der Form $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.
Es gibt nun drei Fälle:

1. $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$ für ein $\varepsilon > 0$
   $\Rightarrow T(n) \in \Theta\left(n^{\log_b a}\right)$
2. $f(n) \in \Theta\left(n^{\log_b a}\right)$
   $\Rightarrow T(n) \in \Theta\left(n^{\log_b a} \log(n)\right)$
3. $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ für ein $\varepsilon > 0$ und ebenfalls für ein $c$ mit $0 < c < 1$ und alle hinreichend großen $n$ gilt: $a \cdot f(\frac{n}{b}) \leq cf(n)$
   $\Rightarrow T(n) \in \Theta(f(n))$

# Das Master-Theorem

Gegeben sei eine Rekurrenz-Gleichung der Form $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.
Es gibt nun drei Fälle:

1. $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$ für ein $\varepsilon > 0$
   $\Rightarrow T(n) \in \Theta\left(n^{\log_b a}\right)$
2. $f(n) \in \Theta\left(n^{\log_b a}\right)$
   $\Rightarrow T(n) \in \Theta\left(n^{\log_b a} \log(n)\right)$
3. $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ für ein $\varepsilon > 0$ und ebenfalls für ein $c$ mit $0 < c < 1$ und alle hinreichend großen $n$ gilt: $a \cdot f(\frac{n}{b}) \leq cf(n)$
   $\Rightarrow T(n) \in \Theta(f(n))$

# Das Master-Theorem

Gegeben sei eine Rekurrenz-Gleichung der Form $T(n) = a \cdot T(\frac{n}{b}) + f(n)$.
Es gibt nun drei Fälle:

1. $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$ für ein $\varepsilon > 0$
   $\Rightarrow T(n) \in \Theta\left(n^{\log_b a}\right)$
2. $f(n) \in \Theta\left(n^{\log_b a}\right)$
   $\Rightarrow T(n) \in \Theta\left(n^{\log_b a} \log(n)\right)$
3. $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ für ein $\varepsilon > 0$ und ebenfalls für ein $c$ mit $0 < c < 1$ und alle hinreichend großen $n$ gilt: $a \cdot f(\frac{n}{b}) \leq cf(n)$
   $\Rightarrow T(n) \in \Theta(f(n))$

# Algorithmus von Karatsuba-Ofman

function rectMult(a, b $\in \mathbb{N}$)
  **assert** a und b haben n = 2k Ziffern
  **if** n = 1 **then** **return** a $\cdot$ b
  Schreibe a als $a_1 \cdot B^k + a_0$
  Schreibe b als $b_1 \cdot B^k + b_0$
  $c_{11}$ := recMult($a_1$, $b_1$)
  $c_{00}$ := recMult($a_0$, $b_0$)
  $c_{0110}$ := recMult($a_1 + a_0$, $b_1 + b_0$)
  e := $c_{11} \cdot B^{2k} + (c_{0110} - c_{11} - c_{00}) \cdot B^k + c_{00}$
  **return** e

Mit Mastertheorem folgt: Die Laufzeit ist in $\Theta(n^{\log_2(3)}) \approx \Theta(n^{1.58})$

# Algorithmus von Karatsuba-Ofman

```
function rectMult(a, b ∈ ℕ)
  assert a und b haben n = 2k Ziffern
  if n = 1 then return a · b
  Schreibe a als a_1 · B^k + a_0
  Schreibe b als b_1 · B^k + b_0
  c_11 := recMult(a_1, b_1)
  c_00 := recMult(a_0, b_0)
  c_0110 := recMult(a_1 + a_0, b_1 + b_0)
  e := c_11 · B^(2k) + (c_0110 - c_11 - c_00) · B^k + c_00
  return e
```

Mit Mastertheorem folgt: Die Laufzeit ist in $\Theta(n^{\log_2(3)}) \approx \Theta(n^{1.58})$

# Beispiel: Lineare Suche

```
function linearSearch(a[0..n-1], z)
  for i:=0 to n-1
    if a[i]=z
     return true
  return false
```

worst/average: $\Theta(n)$, best: $\Theta(1)$

# Beispiel: Lineare Suche

```
function linearSearch(a[0..n-1], z)
  for i:=0 to n-1
    if a[i]=z
     return true
  return false
```

worst/average: $\Theta(n)$, best: $\Theta(1)$

# Beispiel: Binäre Suche

```
function binarySearch(a[0..n-1], z)
  assert a ist sortiert
  (l,r):=(0,n-1)
  while l≤r
    m:=⌊(l+r)/2⌋
    if a[m]=z then return true
    if a[m]>z then r:=m-1 else l:=m+1
  return false
```

worst/average: $\Theta(\log(n))$, best: $\Theta(1)$

# Beispiel: Binäre Suche

```
function binarySearch(a[0..n-1], z)
  assert a ist sortiert
  (l,r):=(0,n-1)
  while l≤r
    m:=⌊(l+r)/2⌋
    if a[m]=z then return true
    if a[m]>z then r:=m-1 else l:=m+1
  return false
```

worst/average: $\Theta(\log(n))$, best: $\Theta(1)$

# Beispiel: Bogosort

Hier die Grobstruktur:

```
function bogosort(a[0..n-1])
  while a ist nicht sortiert
    ordne Elemente von a per Zufall an
```

best case: $\Theta(n)$, average case (laut Wikipedia): $Theta(n \cdot n!)$, und der worst case ist $\infty$!

# Beispiel: Bogosort

Hier die Grobstruktur:

```
function bogosort(a[0..n-1])
  while a ist nicht sortiert
    ordne Elemente von a per Zufall an
```

best case: $\Theta(n)$, average case (laut Wikipedia): $Theta(n \cdot n!)$, und der worst case ist $\infty$!