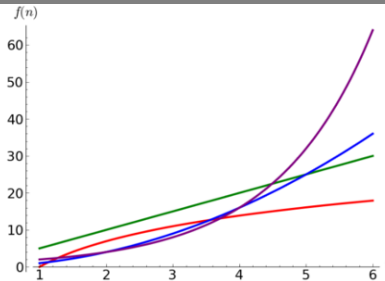


Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 23. April 2013

INSTITUT FÜR THEORETISCHE INFORMATIK, PROF. SANDERS



```

function karatsuba(num1, num2)
if (num1 < 10) or (num2 < 10)
    return num1*num2
/* calculates the size of the numbers */
m = max(size(num1), size(num2))
low1, low2 = lower half of num1, num2
high1, high2 = higher half of num1, num2
/* 3 calls made to numbers approximately half the size */
z0 = karatsuba(low1, low2)
z1 = karatsuba((low1+high1), (low2+high2))
z2 = karatsuba(high1, high2)
return (z2*10^(m)) + ((z1-z2-z0) * 10^(m/2)) + (z0)
    
```

Zu meiner Person:

- Simon Bischof
- Bachelor Informatik
- 6. Semester

Zu meiner Person:

- Simon Bischof
- Bachelor Informatik
- 6. Semester

- 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>

- 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>

- 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>

- 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>

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!

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!

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!

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!**

- 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 Punktsanspruch!**

- 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

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

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

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

Was ist ein Algorithmus?

- 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:
 $\{i \geq 2 \wedge \neg \exists a, b \geq 2 : i = a \cdot b\}$

- 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:
 $\{i \geq 2 \wedge \neg \exists a, b \geq 2 : i = a \cdot b\}$

- 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:
 $\{i \geq 2 \wedge \neg \exists a, b \geq 2 : i = a \cdot b\}$

- 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:
 $\{i \geq 2 \wedge \neg \exists a, b \geq 2 : i = a \cdot b\}$

- 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:
 $\{i \geq 2 \wedge \neg \exists a, b \geq 2 : i = a \cdot b\}$

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "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: Ω (mindestens), ω (schneller), Θ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "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: Ω (mindestens), ω (schneller), Θ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "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: Ω (mindestens), ω (schneller), Θ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "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: Ω (mindestens), ω (schneller), Θ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

- $O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- "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: Ω (mindestens), ω (schneller), Θ (genau so schnell)
- In dieser Vorlesung untersuchen wir oft den "worst case"

Beispiel: Bubble-Sort

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

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

Beispiel: Bubble-Sort

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

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

Beispiel: Bubble-Sort

```
1 function bubbleSort2(a[0..n-1])
2   repeat
3     swapped := false
4     for i:=0 to n-1
5       if a[i] > a[i+1]
6         (a[i],a[i+1]) := (a[i+1],a[i])
7         swapped := true
8   until swapped = false
```

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


```
1 function bubbleSort2(a[0..n-1])
2   repeat
3     swapped := false
4     for i:=0 to n-1
5       if a[i] > a[i+1]
6         (a[i],a[i+1]) := (a[i+1],a[i])
7         swapped := true
8   until swapped = false
```

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

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 $n! \in \omega(2^n)$

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

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

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

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

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

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

```
1 function rectMult(a, b ∈ ℕ)
2   assert a und b haben n = 2k Ziffern
3   if n = 1 then return a · b
4   Schreibe a als  $a_1 \cdot B^k + a_0$ 
5   Schreibe b als  $b_1 \cdot B^k + b_0$ 
6    $c_{11} := \text{rectMult}(a_1, b_1)$ 
7    $c_{00} := \text{rectMult}(a_0, b_0)$ 
8    $c_{0110} := \text{rectMult}(a_1 + a_0, b_1 + b_0)$ 
9    $e := c_{11} \cdot B^{2k} + (c_{0110} - c_{11} - c_{00}) \cdot B^k + c_{00}$ 
10  return e
```

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

```
1 function rectMult(a, b ∈ ℕ)
2   assert a und b haben n = 2k Ziffern
3   if n = 1 then return a · b
4   Schreibe a als  $a_1 \cdot B^k + a_0$ 
5   Schreibe b als  $b_1 \cdot B^k + b_0$ 
6    $c_{11} := \text{rectMult}(a_1, b_1)$ 
7    $c_{00} := \text{rectMult}(a_0, b_0)$ 
8    $c_{0110} := \text{rectMult}(a_1 + a_0, b_1 + b_0)$ 
9    $e := c_{11} \cdot B^{2k} + (c_{0110} - c_{11} - c_{00}) \cdot B^k + c_{00}$ 
10  return e
```

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

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

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


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

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

```
1 function binarySearch(a[0..n-1], z)
2   assert a ist sortiert
3   (l,r):=(0,n-1)
4   while l≤r
5     m:=⌊ $\frac{l+r}{2}$ ⌋
6     if a[m]=z then return true
7     if a[m]>z then r:=m-1 else l:=m+1
8   return false
```

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

```
1 function binarySearch(a[0..n-1], z)
2   assert a ist sortiert
3   (l,r):=(0,n-1)
4   while l ≤ r
5     m:=[ $\frac{l+r}{2}$ ]
6     if a[m]=z then return true
7     if a[m]>z then r:=m-1 else l:=m+1
8   return false
```

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

Hier die Grobstruktur:

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

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

Hier die Grobstruktur:

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

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