

Informatik II - SS 2018

(Algorithmen & Datenstrukturen)

Vorlesung 3 (25.4.2018)

O-Notation, Asymptotische Analyse,
Sortieren III



**UNI
FREIBURG**

Fabian Kuhn

Algorithmen und Komplexität

- Wie können wir die Laufzeit des Algorithmus analysieren?
 - Ist auf jedem Computer unterschiedlich...
 - Hängt vom Compiler, Programmiersprache, etc. Ab
- Wir benötigen ein **abstraktes Mass**, um die Laufzeit zu messen
- **Idee: Zähle Anzahl (Grund-)Operationen**
 - Anstatt direkt die Zeit zu messen
 - Ist unabhängig von Computer, Compiler
 - Ein gutes Mass für die Laufzeit, falls alle Grundoperationen etwa gleich lange brauchen:

Was ist eine Grundoperation?

- Einfache arithmetische Operationen
 - $+$, $-$, $*$, $/$, $\%$ (mod), ...
- Ein Speicherzugriff
 - Variable auslesen, Variablenzuweisung
 - Ist das wirklich eine Grundoperation?
- Ein Funktionsaufruf
 - Natürlich nur das Springen in die Funktion
- **Intuitiv:** eine Zeile Programmcode
- **Besser:** eine Zeile Maschinencode
- **Noch besser (?):** ein Prozessorzyklus

- **Wir werden sehen:** Es ist nur wichtig, dass die Anzahl Grundoperation ungefähr proportional zur Laufzeit ist.

Bisher: Anzahl Grundoperationen ist proportional zur Laufzeit

- Das können wir auch erreichen, ohne die Anzahl Grundoperationen genau zu zählen!

Vereinfachung 1: Wir berechnen nur eine **obere Schranke** (bzw. eine untere Schranke) an die Anzahl Grundoperationen

– So, dass die obere/untere Schranke immer noch proportional ist...

- Anz. Grundop. kann von div. Eigenschaften der Eingabe abhängen
 - Länge der Eingabe, aber auch z.B. bei Sortieren: zufällig, vorsortiert, ...

Vereinfachung 2: Wichtigster Parameter ist Grösse der Eingabe n

Wir betrachten daher die **Laufzeit $T(n)$ als Funktion von n .**

– Und ignorieren weitere Eigenschaften der Eingabe

Selection Sort: Analyse

$T(n)$: Anzahl Grundop. von Selection Sort bei Arrays der Länge n

Lemma: Es gibt eine **Konstante** $c_U > 0$, so dass $\underline{\underline{T(n)}} \leq \underline{c_U} \cdot n^2$

Lemma: Es gibt eine **Konstante** $c_L > 0$, so dass $T(n) \geq c_L \cdot n^2$

Landau-Symbole (“O-Notation”)

- Formalismus, um das asymptotische Wachstum von Funktionen zu beschreiben.
 - Formale Definitionen: siehe nächste Folie...

- Es gibt eine Konst. C , so dass $T(n) \leq C \cdot f(n)$ wird zu:

$$\underline{T(n) \in O(f(n))}$$

- Es gibt eine Konst. C , so dass $T(n) \geq C \cdot g(n)$ wird zu:

$$\underline{T(n) \in \Omega(g(n))}$$

- Bei Selection Sort: $T(n) \in O(n^2)$, $T(n) \in \Omega(n^2)$

Landau-Symbole : Definitionen

$$O(g(n)) := \{f(n) \mid \exists c, n_0 > 0 \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- Funktion $f(n) \in O(g(n))$, falls es Konstanten c und n_0 gibt, so dass $f(n) \leq c \cdot g(n)$ für alle $n \geq n_0$

$$\Omega(g(n)) := \{f(n) \mid \exists c, n_0 > 0 \forall n \geq n_0 : f(n) \geq c \cdot g(n)\}$$

- Funktion $f(n) \in \Omega(g(n))$, falls es Konstanten c und n_0 gibt, so dass $f(n) \geq c \cdot g(n)$ für alle $n \geq n_0$

Θ

$$\Theta(g(n)) := O(g(n)) \cap \Omega(g(n))$$

- Funktion $f(n) \in \Theta(g(n))$, falls es Konstanten c_1, c_2 und n_0 gibt, so dass $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ für alle $n \geq n_0$, resp. falls $f(n) \in O(n)$ und $f(n) \in \Omega(n)$

Landau-Symbole : Definitionen $n \in o(n^2)$

$$o(g(n)) := \{f(n) \mid \boxed{\forall c > 0} \exists n_0 > 0 \forall n \geq n_0 : \underline{f(n) \leq c \cdot g(n)}\}$$

- Funktion $f(n) \in o(g(n))$, falls für alle Konstanten $c > 0$ gilt, dass $f(n) \leq c \cdot g(n)$ (für genug grosse n , abhängig von c)

$$\omega(g(n)) := \{f(n) \mid \boxed{\forall c > 0} \exists n_0 > 0 \forall n \geq n_0 : \underline{f(n) \geq c \cdot g(n)}\}$$

- Funktion $f(n) \in \omega(g(n))$, falls für alle Konstanten $c > 0$ gilt, dass $f(n) \geq c \cdot g(n)$ (für genug grosse n , abhängig von c)

Insbesondere gilt:

$$f(n) \in o(g(n)) \implies f(n) \in O(g(n))$$

$$f(n) \in \omega(g(n)) \implies f(n) \in \Omega(g(n))$$

$f(n) \in O(g(n))$:

- $f(n) \leq g(n)$, asymptotisch gesehen...
- $f(n)$ wächst asymptotisch nicht schneller als $g(n)$

$f(n) \in \Omega(g(n))$:

- $f(n) \geq g(n)$, asymptotisch gesehen...
- $f(n)$ wächst asymptotisch mindestens so schnell, wie $g(n)$

$f(n) \in \Theta(g(n))$:

- $f(n) = g(n)$, asymptotisch gesehen...
- $f(n)$ wächst asymptotisch gleich schnell, wie $g(n)$

$f(n) \in o(g(n))$:

- $f(n) \ll g(n)$, asymptotisch gesehen...
- $f(n)$ wächst asymptotisch langsamer als $g(n)$

$f(n) \in \omega(g(n))$:

- $f(n) \gg g(n)$, asymptotisch gesehen...
- $f(n)$ wächst asymptotisch schneller als $g(n)$

Falls $f(n)$ und $g(n)$ monoton wachsen, gilt:

$$f(n) \in o(g(n)) \iff f(n) \notin \Omega(g(n))$$

$$f(n) \in \omega(g(n)) \iff f(n) \notin O(g(n))$$

Definition über Grenzwerte (vereinfacht)

Folgende Definitionen gelten für monoton wachsende Funktionen

$$f(n) \in O(g(n)), \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$f(n) \in \Omega(g(n)), \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$$

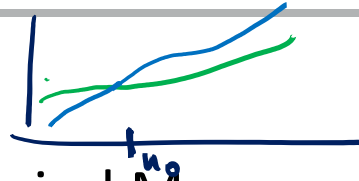
$$f(n) \in \Theta(g(n)), \quad 0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$f(n) \in o(g(n)), \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) \in \omega(g(n)), \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

Landau-Notation : Bemerkungen

Schreibweise:



- $O(g(n)), \Omega(g(n)), \dots$ sind Mengen (von Funktionen)
- Korrekte Schreibweise ist deshalb eigentlich: $f(n) \in O(g(n))$
- Sehr verbreitete Schreibweise: $f(n) = O(g(n)) + o(\dots)$

Asymptotisches Verhalten für allgemeine Grenzwerte:

- gleiche Schreibweise auch für Verhalten von z.B. $f(x)$ für $x \rightarrow 0$
- z.B. Taylor-Reihen: $e^x = 1 + x + O(x^2)$, bzw. $e^x = 1 + x + o(x)$

Alternative Definition für $\Omega(g(n))$:

$$g(n) \in \Omega(f(n))$$

$$\Omega(g(n)) := \{f(n) \mid \exists c, n_0 > 0 \forall n \geq n_0 : f(n) \geq c \cdot g(n)\}$$

$$\Omega(g(n)) := \{f(n) \mid \exists c > 0 \forall n_0 > 0 \exists n \geq n_0 : f(n) \geq c \cdot g(n)\}$$

– Wir verwenden die 1. Definition

– Macht nur bei nicht-monotonen Funktionen einen Unterschied

$$g(n) = n^2 \quad f(n) = \begin{cases} n^2 & \text{falls } n \text{ gerade} \\ 1 & \text{falls } n \text{ ungerade} \end{cases}$$

Landau-Notation : Beispiele

$$O(\log n)$$

Selection Sort:

$$\log_{10} n = \frac{\ln n}{\ln 10}$$

$$\log_2 n = \frac{\ln n}{\ln 2}$$

- Laufzeit $T(n)$, es gibt Konstanten $c_1, c_2 : c_1 n^2 \leq T(n) \leq c_2 n^2$

$$T(n) \in O(n^2), \quad T(n) \in \Omega(n^2), \quad T(n) \in \underline{\Theta(n^2)}$$

- $T(n)$ wächst schneller als linear: $T(n) \in \omega(n)$

Weitere Beispiele:

- $f(n) = 10n^3, g(n) = n^3/1000 : f(n) \in \Theta(g(n))$

- $f(n) = e^n, g(n) = n^{100} : f(n) \in \omega(g(n))$

- $f(n) = n/\log_2 n, g(n) = \sqrt{n} : f(n) \in \omega(g(n))$

- $f(n) = n^{1/256}, g(n) = 10 \ln n : f(n) \in \omega(g(n))$

- $f(n) = \log_{10} n, g(n) = \log_2 n : f(n) \in \Theta(g(n))$

- $f(n) = n^{\sqrt{n}}, g(n) = 2^n : f(n) \in o(g(n))$

$$\log_2 n^{\sqrt{n}} = \sqrt{n} \cdot \log_2 n \quad / \quad \log_2 2^n = n$$

$t = \ln n$

$t = \log_2 n$

$$\lim_{n \rightarrow \infty} \frac{e^n}{n^{100}} \rightarrow \infty$$

$$\frac{n}{\log_2 n \cdot \sqrt{n}} = \frac{\sqrt{n}}{\log_2 n} = \frac{2^{t/2}}{t}$$

$$\frac{n^{1/256}}{10 \ln n} = \frac{e^{t/256}}{10 t}$$

$$\frac{2^{\sqrt{n} \log_2 n}}{2^n} = 2^{-\frac{\sqrt{n}}{\log_2 n}}$$

Analyse Bubble Sort

BubbleSort(A):

```
1: for i=0 to n-2 do           // need to repeat n-1 times
3: (*) for j=0 to n-2-i do
4:     if (A[j] > A[j+1]) then } O(1)
5:     swap(A[j], A[j+1])
```

$x(n)$: # Schleifendurchläufe (*)

$T(n) \in \Theta(x(n))$

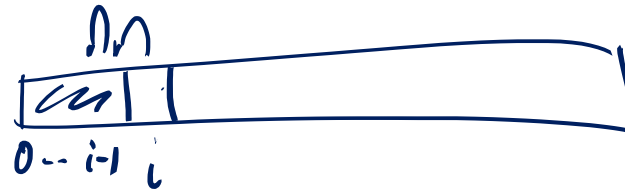
$$x(n) = \sum_{i=0}^{n-2} (n-1-i) = \sum_{k=1}^{n-1} k \in \Theta(n^2)$$

Analyse Insertion Sort

InsertionSort(A):

```
1: for i = 1 to n-1 do
2:   // prefix A[1..i] is already sorted
3:   pos = i
4:   while (pos > 0) and (A[pos] < A[pos-1]) do
5:     swap(A[pos], A[pos-1])
6:     pos = pos - 1
```

$x(n)$: # Schleifendurchl. der while-S.



$$x(n) \leq \sum_{i=1}^{n-1} i \in O(n^2)$$

$$x(n) \geq \sum_{i=1}^{n-1} 1 \in \Omega(n)$$

Worst Case Analyse

- Analysiere Laufzeit $T(n)$ für eine schlechtestmögliche Eingabe der Grösse n
- Wichtigste / Standard- Art der Algorithmenanalyse

Best Case Analyse

- Analysiere Laufzeit $T(n)$ für eine bestmögl. Eingabe der Grösse n
- Meistens uninteressant...

Average Case Analyse

- Analysiere Laufzeit $T(n)$ für eine typische Eingabe der Grösse n
- Problem: was ist eine typische Eingabe?
 - Standardansatz: zufällige Eingabe
 - nicht klar, wie nahe tatsächliche Instanzen bei uniform zufälligen sind...
 - eine mögl. Alternative: smoothed analysis (werden wir nicht anschauen)

Wie gut ist quadratische Laufzeit?

Quadratisch = 2x so grosse Eingabe → 4x so grosse Laufzeit

– das wächst für grosse n schon ziemlich schnell...

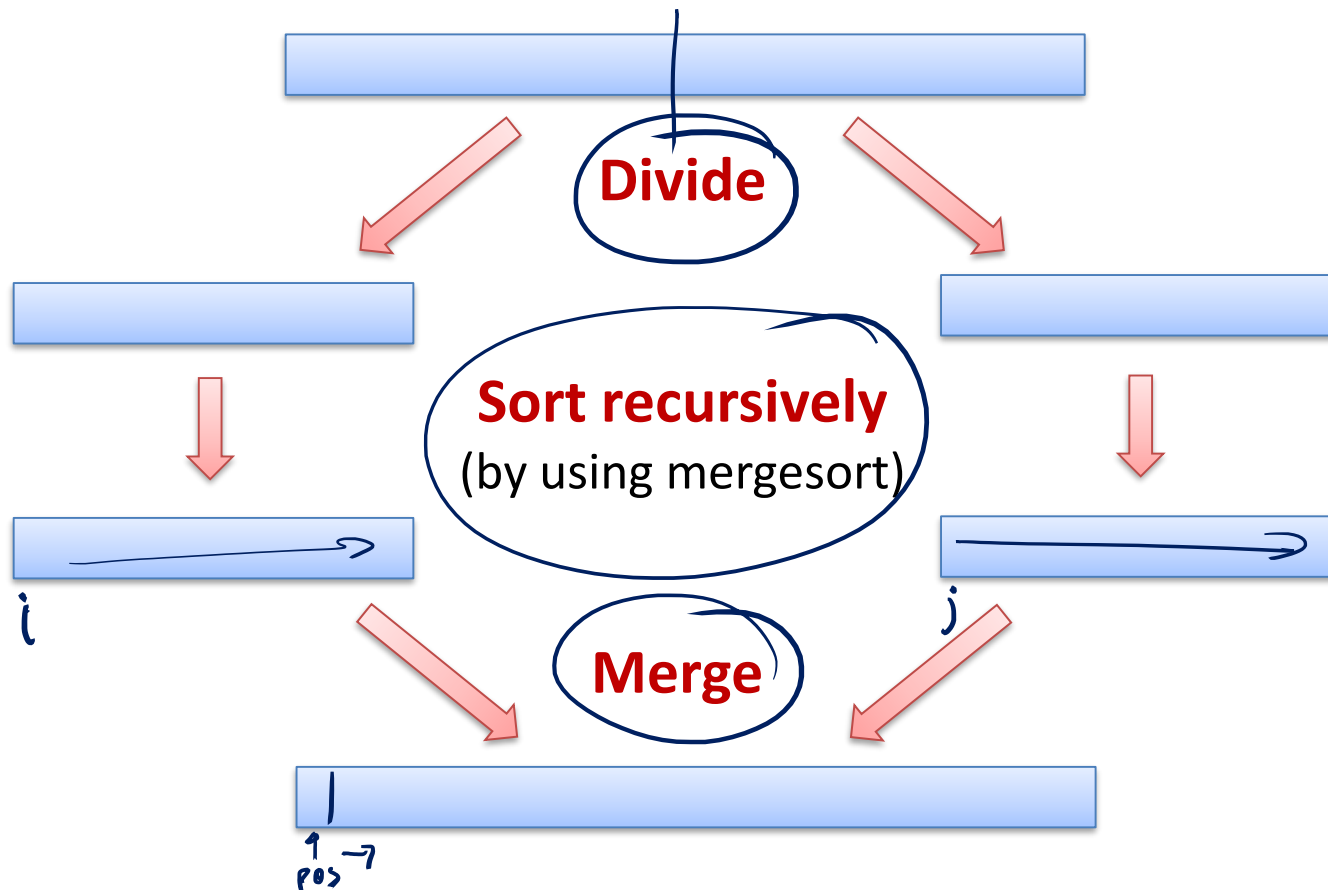
Beispielrechnung:

- Nehmen wir an, Anz. Grundop. $T(n) = n^2$
- Nehmen wir zudem an, 1 Grundop. pro Rechnerzyklus
- Bei einem 1Ghz-Rechner gibt das 1 ns pro Grundop.

Eingabegrösse n	4 Bytes pro Zahl	Laufzeit $T(n)$
<u>10^3</u> Zahlen	<u>$\approx 4\text{KB}$</u>	$10^{3 \cdot 2} \cdot 10^{-9} \text{ s} = \underline{1 \text{ ms}}$
<u>10^6</u> Zahlen	<u>$\approx 4\text{MB}$</u>	$10^{6 \cdot 2} \cdot 10^{-9} \text{ s} = \underline{16.7 \text{ min}}$
<u>10^9</u> Zahlen	<u>$\approx 4\text{GB}$</u>	$10^{9 \cdot 2} \cdot 10^{-9} \text{ s} = \underline{31.7 \text{ Jahre}}$

für grosse Probleme zu langsam!

Analyse Merge Sort



- Divide ist trivial → Kosten: $O(1)$
- Rekursives Sortieren: Werden wir gleich noch anschauen...
- Merge: Das werden wir uns zuerst anschauen...

Analyse Merge-Schritt

```
MergeSortRecursive(A, start, end, tmp)
```

```
:
```

```
5:   pos = start; i = start; j = middle
```

```
6:   while (pos < end) do j >= end or
```

```
7:     if (i < middle) and (A[i] < A[j]) then
```

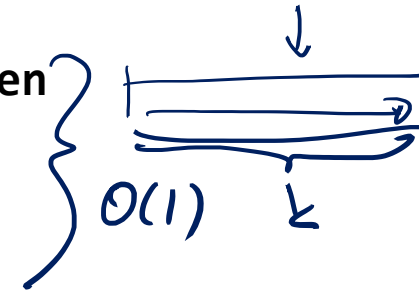
```
8:       tmp[pos] = A[i]; pos++; i++
```

```
9:     else
```

```
10:      tmp[pos] = A[j]; pos++; j++
```

```
11:   for i = start to end-1 do A[i] = tmp[i]
```

// sort A[start..end-1]



$k := \text{end} - \text{start}$

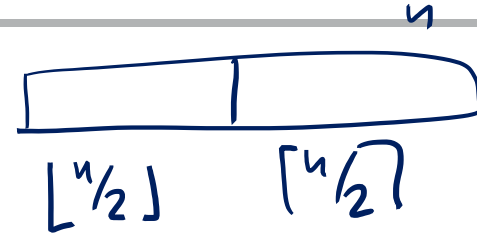
Laufzeit $\in O(\# \text{Schleifendurchl.})$

$\# \text{Schleifendurchläufe} \in O(k)$

Laufzeit: $O(k)$

Analyse Merge Sort

Laufzeit $T(n)$ setzt sich zusammen aus:



- Divide und Merge: $O(n)$
- 2 rekursive Aufrufe zum Sortieren von $\lfloor n/2 \rfloor$ und $\lceil n/2 \rceil$ Elementen

Rekursive Formulierung von $T(n)$:

- Es gibt eine Konstante $b > 0$, so dass

$$\underline{\underline{T(n)}} \leq T\left(\underline{\underline{\lfloor \frac{n}{2} \rfloor}}\right) + T\left(\underline{\underline{\lceil \frac{n}{2} \rceil}}\right) + \underline{\underline{b \cdot n}}, \quad \underline{\underline{T(1) \leq b}}$$

- Wir machen uns das Leben ein bisschen einfacher und ignorieren das Auf- und Abrunden: wähle n als Zweierpotenz

$$\underline{\underline{T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + b \cdot n, \quad T(1) \leq b}}$$

Analyse Merge Sort

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + b \cdot n, \quad T(1) \leq b$$

Setzen wir einfach mal ein, um zu sehen, was rauskommt...

$$T(n) \leq 2T(n/2) + bn \qquad T(n/2) \leq 2 \cdot T(n/4) + b \cdot \frac{n}{2}$$

$$\leq 4 \cdot T(n/4) + bn + bn$$

$$= 4T(n/4) + 2bn$$

$$\leq 4(2T(n/8) + b \cdot \frac{n}{4}) + 2bn$$

$$= 8 \cdot T(n/8) + 3bn$$

⋮

$$\leq 2^k \cdot T(n/2^k) + k \cdot bn$$

Vermutung



$$= n \cdot T(1) + b \cdot n \cdot \log_2 n \leq \underline{\underline{b \cdot n(1 + \log_2 n)}}$$

$$T(1) \leq b$$

$$T(n) \in O(n \cdot \log n)$$

Analyse Merge Sort

Rekursionsgleichung: $T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + b \cdot n$, $T(1) \leq b$

Vermutung: $T(n) \leq b \cdot n \cdot (1 + \log_2 n)$

Beweis durch vollständige Induktion: $i=0$

Verankerung: $n=1$ $T(1) \leq b \cdot 1 \cdot (1 + \log_2 1) = b$ ✓

Induktionsschritt:

Ind.voraussetzung: Vermutung gilt für Werte $< n$

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + b \cdot n$$

$$\stackrel{\text{(I.V.)}}{\leq} 2 \cdot \left(b \cdot \frac{n}{2} \cdot \underbrace{\left(1 + \log_2 \frac{n}{2} \right)}_{\log_2 n} \right) + b n$$

$$\log_2 \frac{n}{2} = \log_2 n - 1$$

$$= b \cdot n \cdot \log_2 n + b n = b n (1 + \log_2 n) \quad \checkmark$$

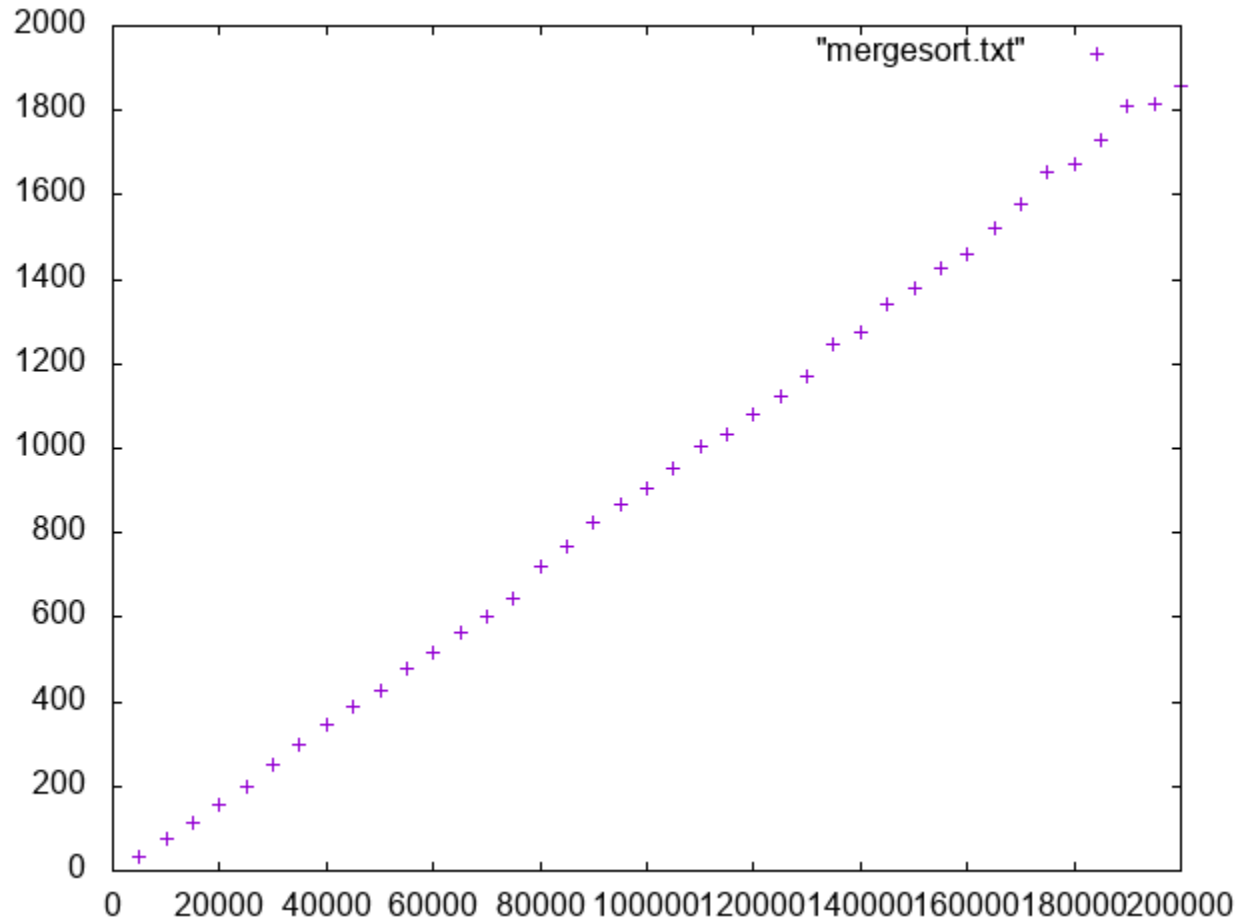
□

Alternative Analyse Merge Sort

Rekursionsgleichung: $T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + b \cdot n, T(1) \leq b$

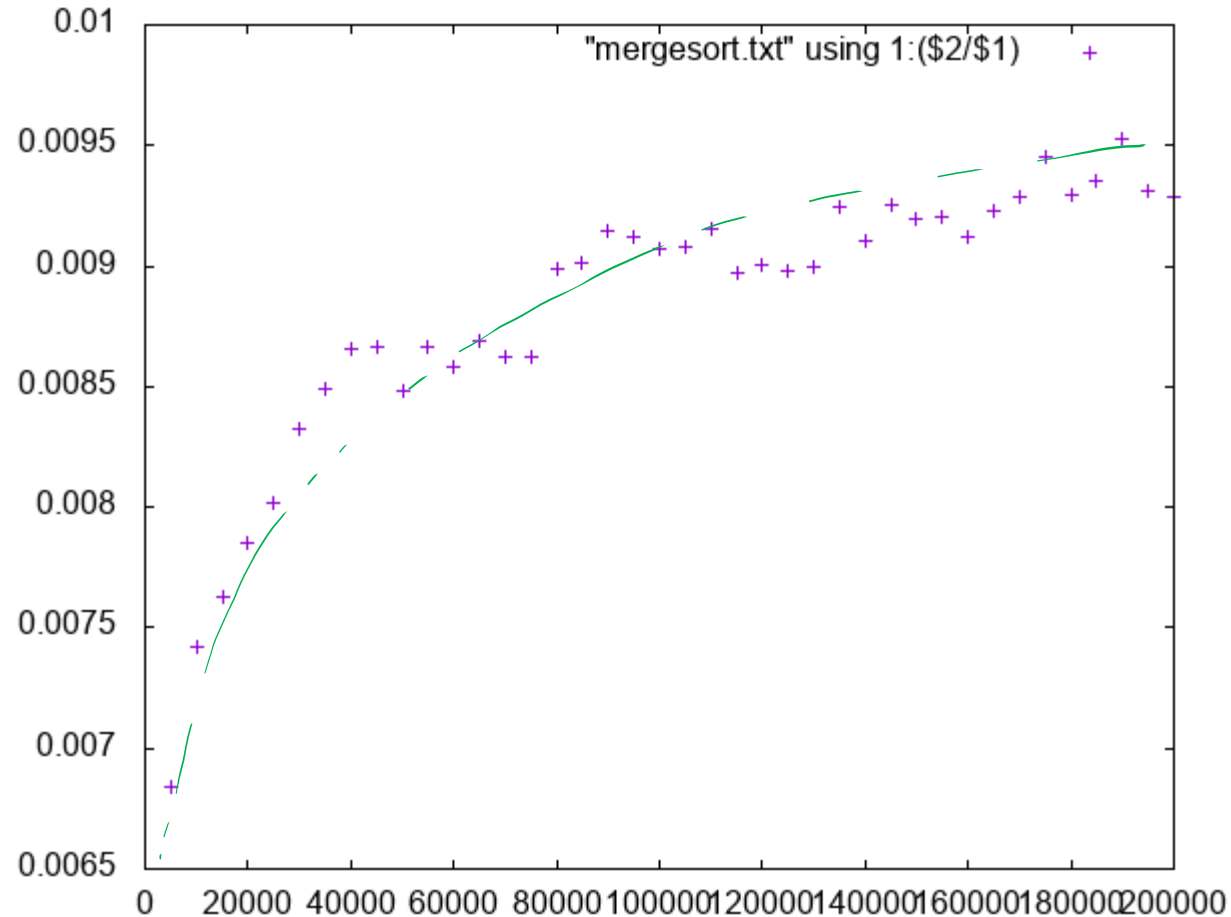
Betrachten wir den Rekursionsbaum:

Merge Sort Messungen für grössere n



```
set term png
Gnuplot: set output "mergesort_1.png"
plot "mergesortlarge.txt"
```


Merge Sort Messungen für grössere n



```
set term png
Gnuplot: set output "mergesort_2.png"
plot "mergesortlarge.txt" using 1:($2/$1)
```

Zusammenfassung Analyse Merge Sort

Die Laufzeit von Merge Sort ist $T(n) \in O(n \cdot \log n)$.

- wächst fast linear mit der Grösse der Eingabe...

Wie gut ist das?

- Beispielrechnung:
 - Nehmen wir wieder an, 1 Grundop. = 1 ns
 - Wir sind aber ein bisschen konservativer als vorher und nehmen

$$T(n) = 10 \cdot n \log_2 n$$

Eingabegrösse n	4 Bytes p. Zahl	Laufzeit $T(n) = 10 \cdot n \log n$	n^2
$2^{10} \approx 10^3$ Zahlen	$\approx 4\text{KB}$	$10 \cdot 10 \cdot 2^{10} \cdot 10^{-9} \text{ s} \approx \underline{0.1 \text{ ms}}$	<u>1 ms</u>
$2^{20} \approx 10^6$ Zahlen	$\approx 4\text{MB}$	$10 \cdot 20 \cdot 2^{20} \cdot 10^{-9} \text{ s} \approx \underline{0.2 \text{ s}}$	<u>16.7 min</u>
$2^{30} \approx 10^9$ Zahlen	$\approx 4\text{GB}$	$10 \cdot 30 \cdot 2^{30} \cdot 10^{-9} \text{ s} \approx \underline{\underline{5.4 \text{ min}}}$	<u>31.7 Jahre</u>
$2^{40} \approx 10^{12}$ Zahlen	$\approx 4\text{TB}$	$10 \cdot 40 \cdot 2^{40} \cdot 10^{-9} \text{ s} \approx \underline{\underline{122 \text{ h}}}$	<u>$> 10^7$ Jahre</u>