Informatik II - SS 2014

(Algorithmen & Datenstrukturen)

Vorlesung 4 (7.5.2014)

Asymptotische Analyse, Sortieren IV

UNI FREIBURG

Fabian Kuhn

Algorithmen und Komplexität

Erfahrungen 1. Übung

UNI FREIBURG

- C++ / Java sind komplett ungewohnt
  - Struktur der Dateien / Klassen, was man genau braucht
  - Nicht zum Laufen bekommen...
- Wöchentliche Fragestunde zusätzlich zum Forum?
  - Werden wir ganz sicher einmal machen und zwar am Dienstag, 20. Mai (anstatt Vorlesung)
  - Im 101-00-026 oder im Kinohörsaal (Bekanntgabe über Webseite & Forum)
- Nach 6 Stunden aufgegeben…
  - ca 9-10 Stunden pro Übungsblatt sind normal
  - **Aber:** Wenden Sie sich bei Problemen frühzeitig ans Forum!
- Gnuplot wurde nicht erklärt
  - Sollte man selbst rausfinden können / Sie müssen nicht Gnuplot verwenden
  - Alternativen: excel, andere Office-Pakete, matlab, R, ...

Bemerkungen 1. Übung

UNI FREIBURG

Hinweise zu Daphne, etc. auf Webseite (seit Freitag)

http://ac.informatik.uni-freiburg.de/teaching/ss_14/info2/InfoUebungen.php

- Einfachste Lösung: Vorlage aus der Vorlesung übernehmen und anpassen
  - Makefile / build.xml einfach übernehmen und jedesmal verwenden
  - build.properties jeweils kurz anpassen
  - Für Übung 1 dann einfach Quicksort-Code hinzufügen und Main-File anpassen
  - Wir werden bis Ende Woche eine Musterlösung bereitstellen:
    Auch wenn Sie die Aufgabe nicht gelöst haben, versuchen Sie die Vorlesungsbeispiele und die Musterlösung aus dem SVN (public-Folder) auszuchecken und bei Ihnen zu kompilieren

Nächste Übungen

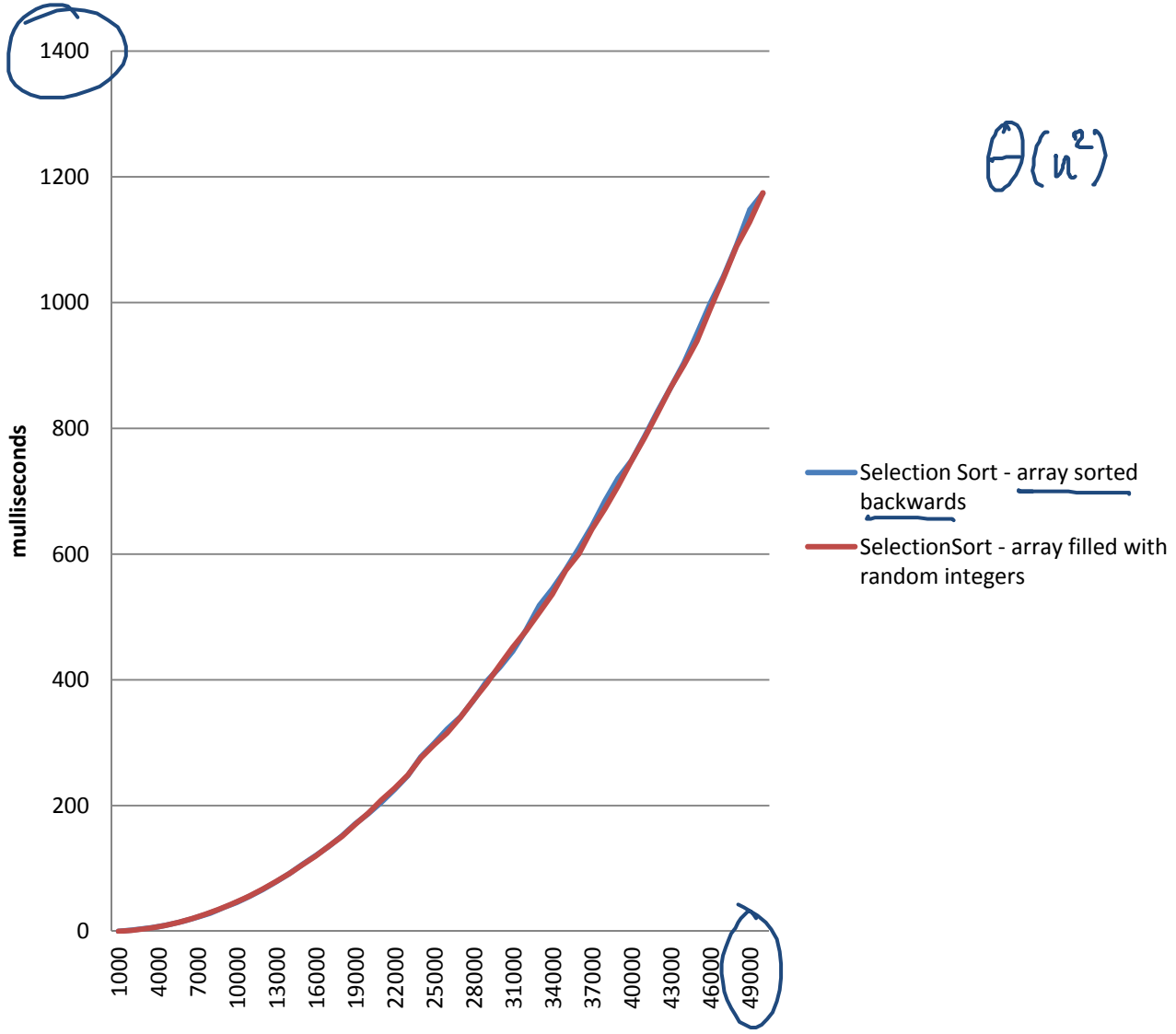
UNI FREIBURG

- Übung 2 (online) ist rein theoretisch...
- Übung 3 wird mind. zur Hälfte theoretisch sein...

Weitere Programmierübungen:

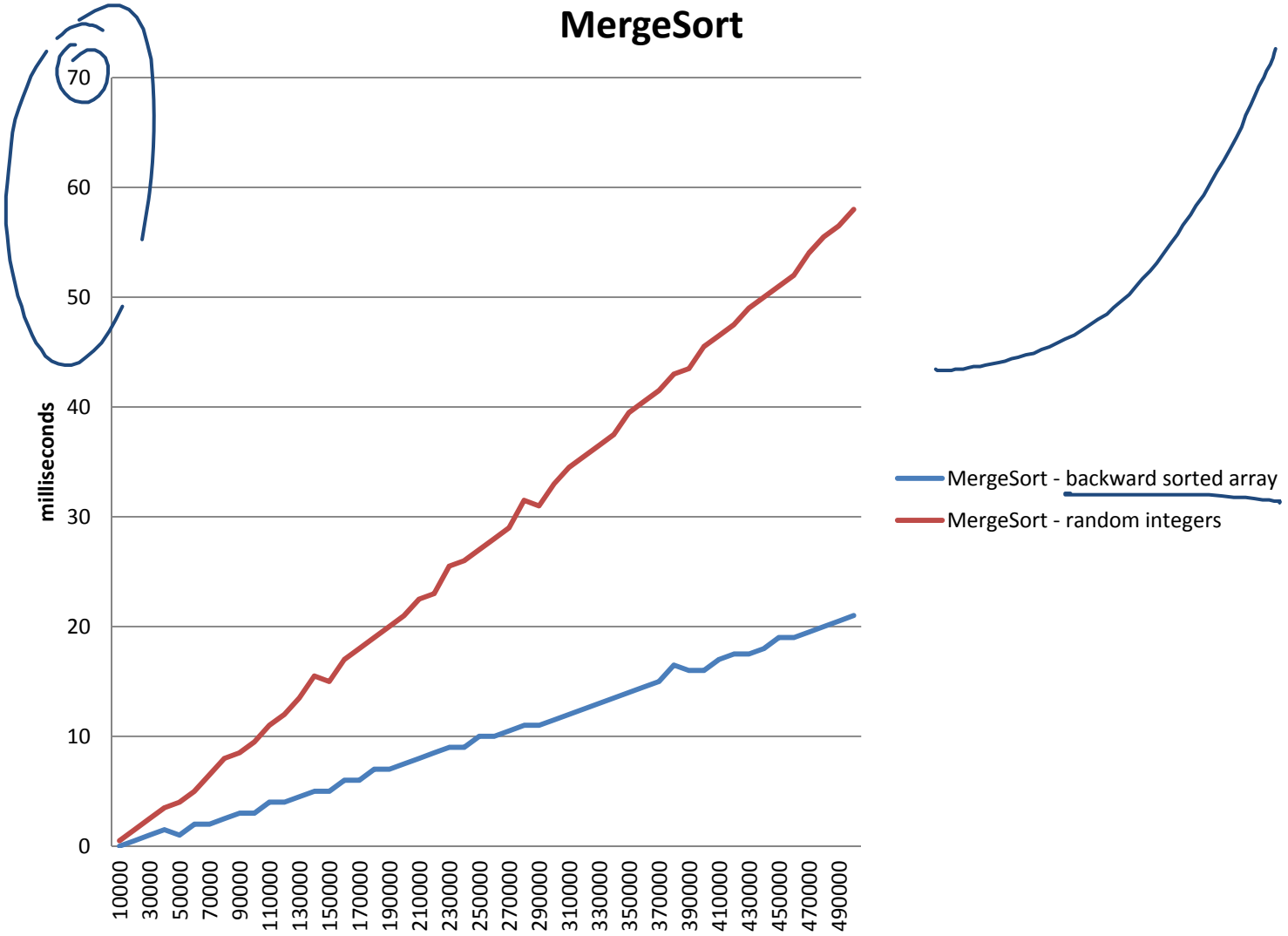
- Wir werden versuchen, jeweils ein Java/C++ Grundgerüst zu geben, welches Sie dann entsprechend ergänzen können
- Wir werden versuchen, in der Übungsaufgabe genauer zu beschreiben, was Sie tun müssen
  - wenigstens in den ersten paar praktischen Übungen...
- Es ist normal, dass die Dinge nicht immer auf Anhieb funktionieren...
  - Verschwenden Sie allerdings nicht zu viel Zeit, ohne Fortschritt zu machen, sondern wenden Sie sich **frühzeitig** mit Ihren Fragen ans Forum
  - Wenn möglich nicht erst am Montag Abend / Nacht!

Messungen Selection Sort

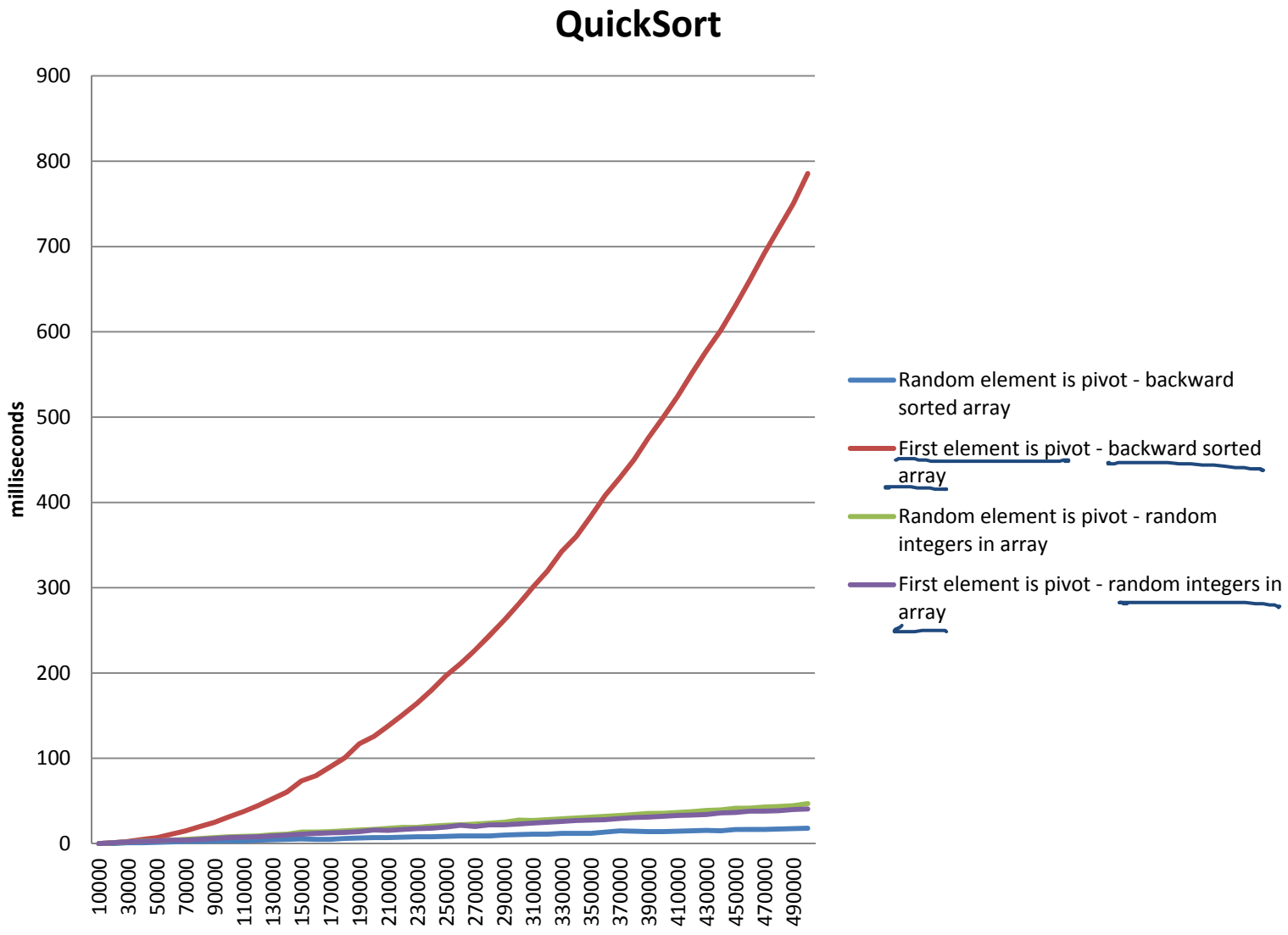


$\Theta(n^2)$

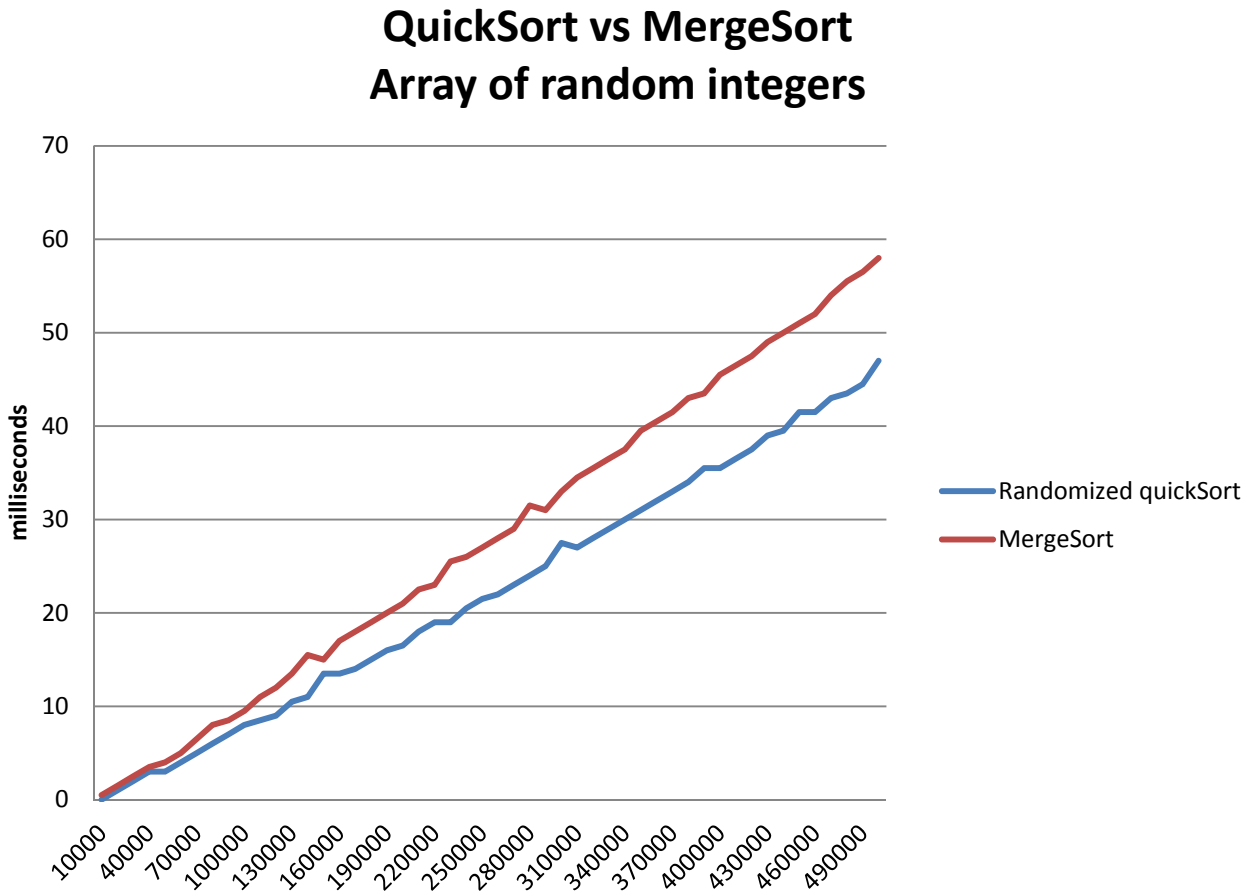
Messungen Merge Sort



Messungen Quick Sort



Messungen – Merge Sort vs. Quick Sort



Analyse Bubble Sort

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

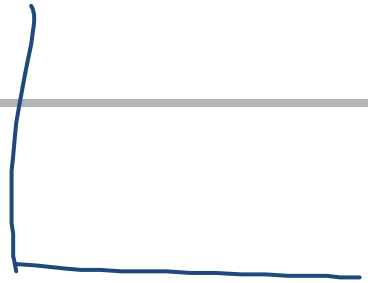
$O(1)$

$T(n) = \Theta(x(n)) = \Theta(n^2)$

$x(n) =$ "Schleifend."

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

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
```

$O(1)$

$X(n) = \text{"} \ldots \text{"}$

$X(n) \leq \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \leq \frac{n^2}{2}$

$T(n) = O(n^2)$

$X(n) \geq \sum_{i=1}^{n-1} 1 = n-1$

$T(n) = \Omega(n)$

Worst case, best case, average case

UNI FREIBURG

Worst Case Analyse

$T(n) \leq \ldots$

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

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?

UNI FREIBURG

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

- das wächst für grosse n schon ziemlich schnell…

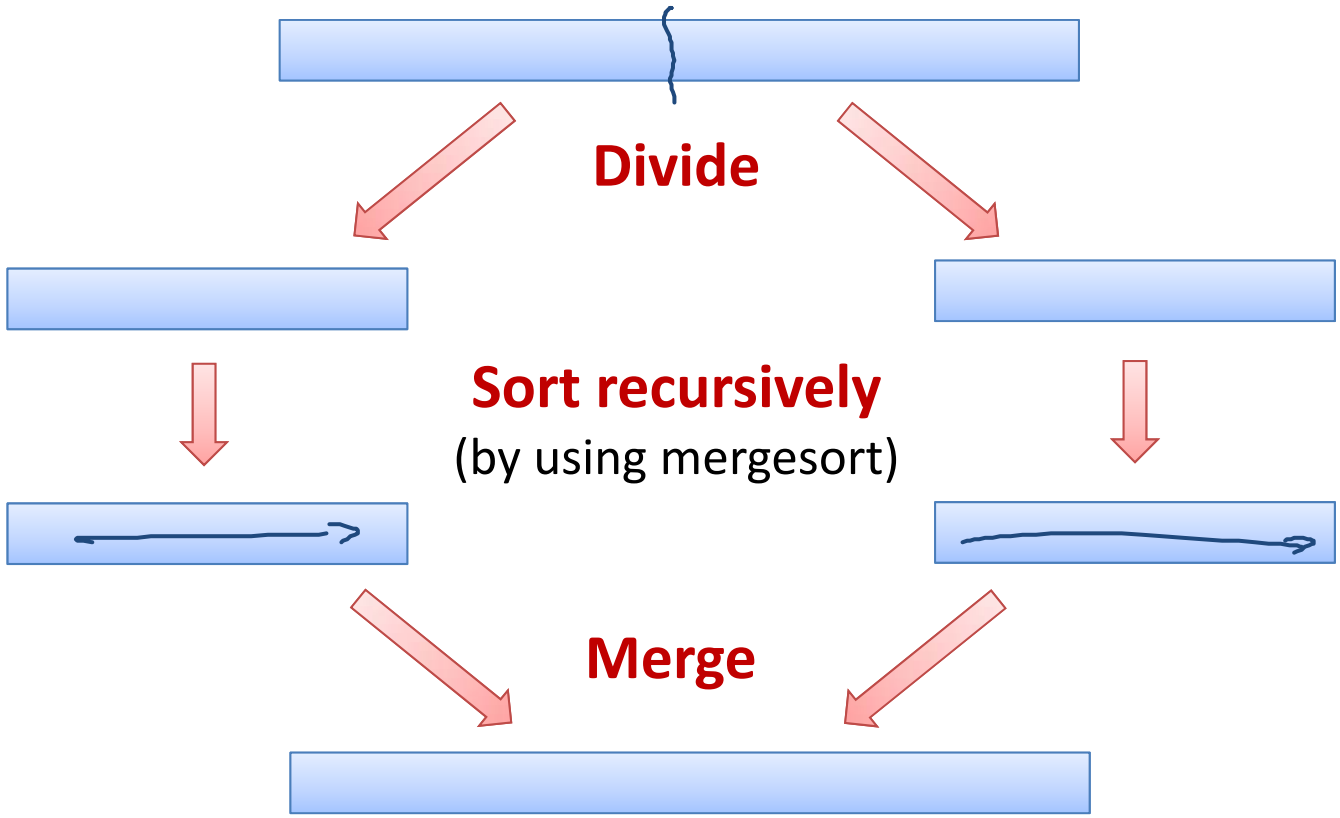
Beispielrechnung:

- 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 $T(n)$
10^3 Zahlen	$\approx 4\text{KB}$	$10^{3\cdot 2} \cdot 10^{-9}\ \text{s} = 1\ \text{ms}$
10^6 Zahlen	$\approx 4\text{MB}$	$10^{6\cdot 2} \cdot 10^{-9}\ \text{s} = 16.7\ \text{min}$
10^9 Zahlen	$\approx 4\text{GB}$	$10^{9\cdot 2} \cdot 10^{-9}\ \text{s} = 31.7\ \text{Jahre}$

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...

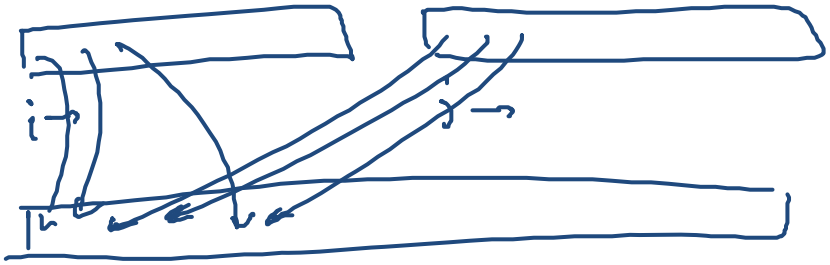
Analyse Merge-Schritt

```
MergeSortRecursive(A, start, end, tmp)          // sort A[start..end-1]
    ⋮
 5:   pos = start; i = start; j = middle
 6:   while (pos < end) do
 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]
```

$m := end - start$

$O(1)$

$O(end - start) = O(m)$



Anz. while-Durchläufe: $O(m)$

tmp → A

Merge-Schritt für Teil-Array der Länge m

Laufzeit $O(m)$

Analyse Merge Sort

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

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

Rekursive Formulierung von $T(n)$:

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

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

- Wir machen uns das Leben ein bisschen einfacher und ignorieren das Auf- und Abrunden:

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

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…

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

$$\leq 2\left(2T\left(\frac{n}{4}\right) + b \cdot \frac{n}{2}\right) + b \cdot n$$

$$= 4 \cdot T\left(\frac{n}{4}\right) + 2b \cdot n$$

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

$$= 8 \cdot T\left(\frac{n}{8}\right) + 3bn$$

$$\vdots$$

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

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

$k = \log_2 n$

$$T\left(\frac{n}{2}\right) \leq 2 \cdot T\left(\frac{n}{4}\right) + b \cdot \frac{n}{2}$$

$$T(n) \leq bn(1 + \log_2 n)$$

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:

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

Induktionsschritt:

Ind.-voraussetzung: Vermutung gilt für alle Werte $\leq n-1$

$$T(n) \leq 2 \cdot T\left(\tfrac{n}{2}\right) + bn$$

$$\overset{\text{I.V.}}{\leq} 2\left(b \cdot \tfrac{n}{2} \cdot \left(1 + \log_2 \tfrac{n}{2}\right)\right) + bn$$

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

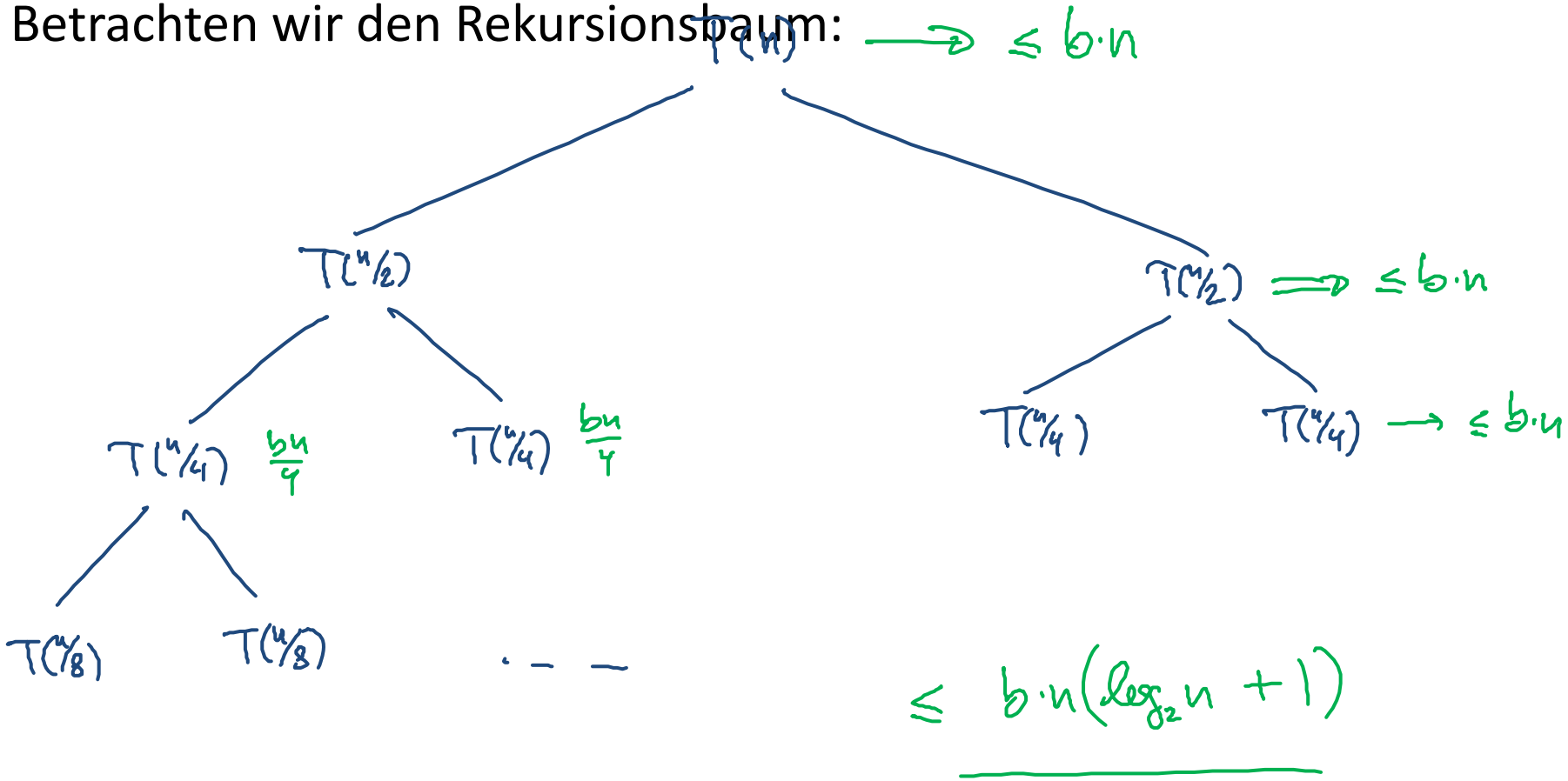
$$= b \cdot n \cdot \log_2 n + b \cdot n = bn(1 + \log_2 n)$$ ✓ □

$$\Rightarrow T(n) = O(n \cdot \log n) \leq o(n^2)$$

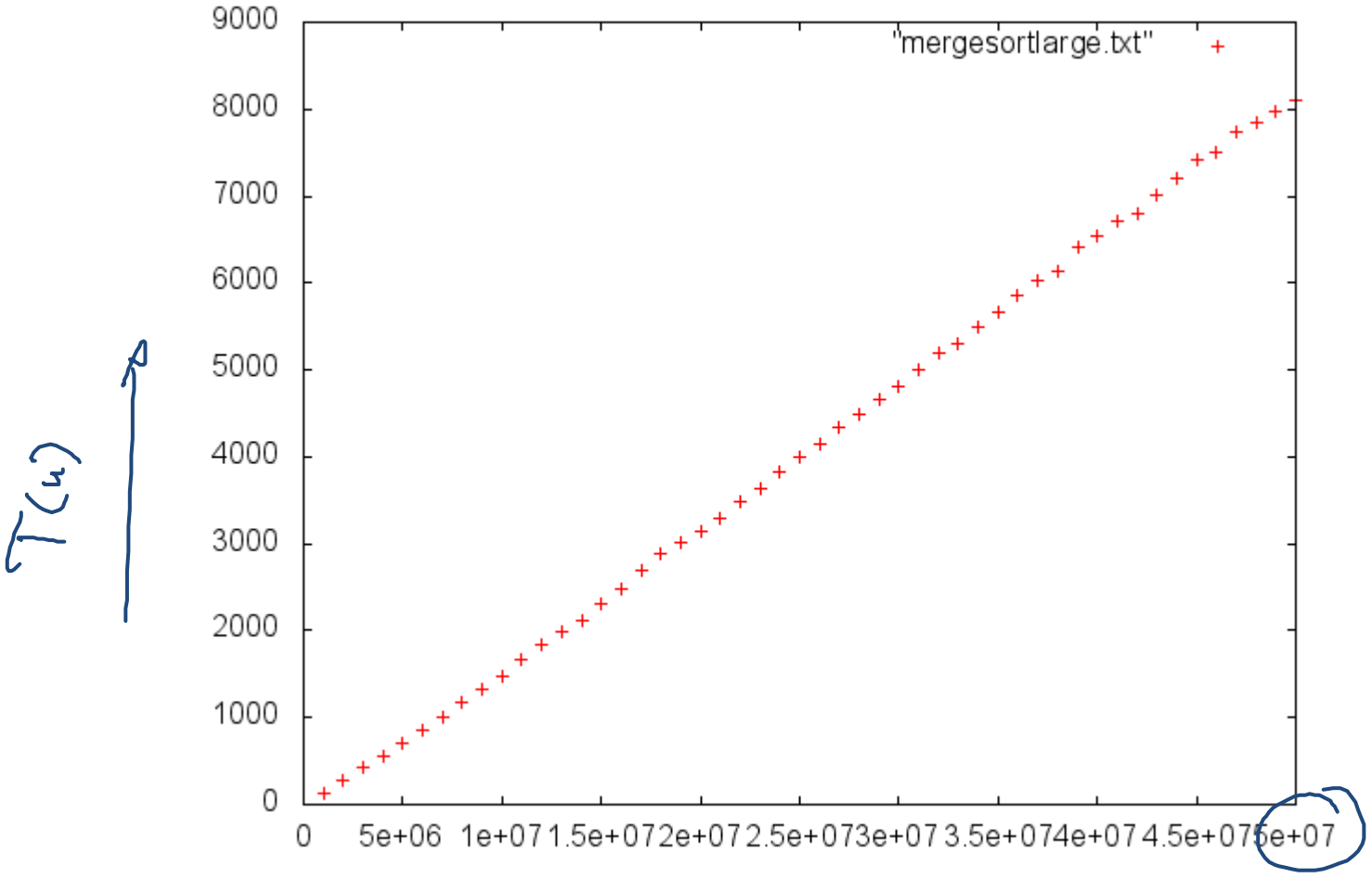
Alternative Analyse Merge Sort

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

Betrachten wir den Rekursionsbaum:



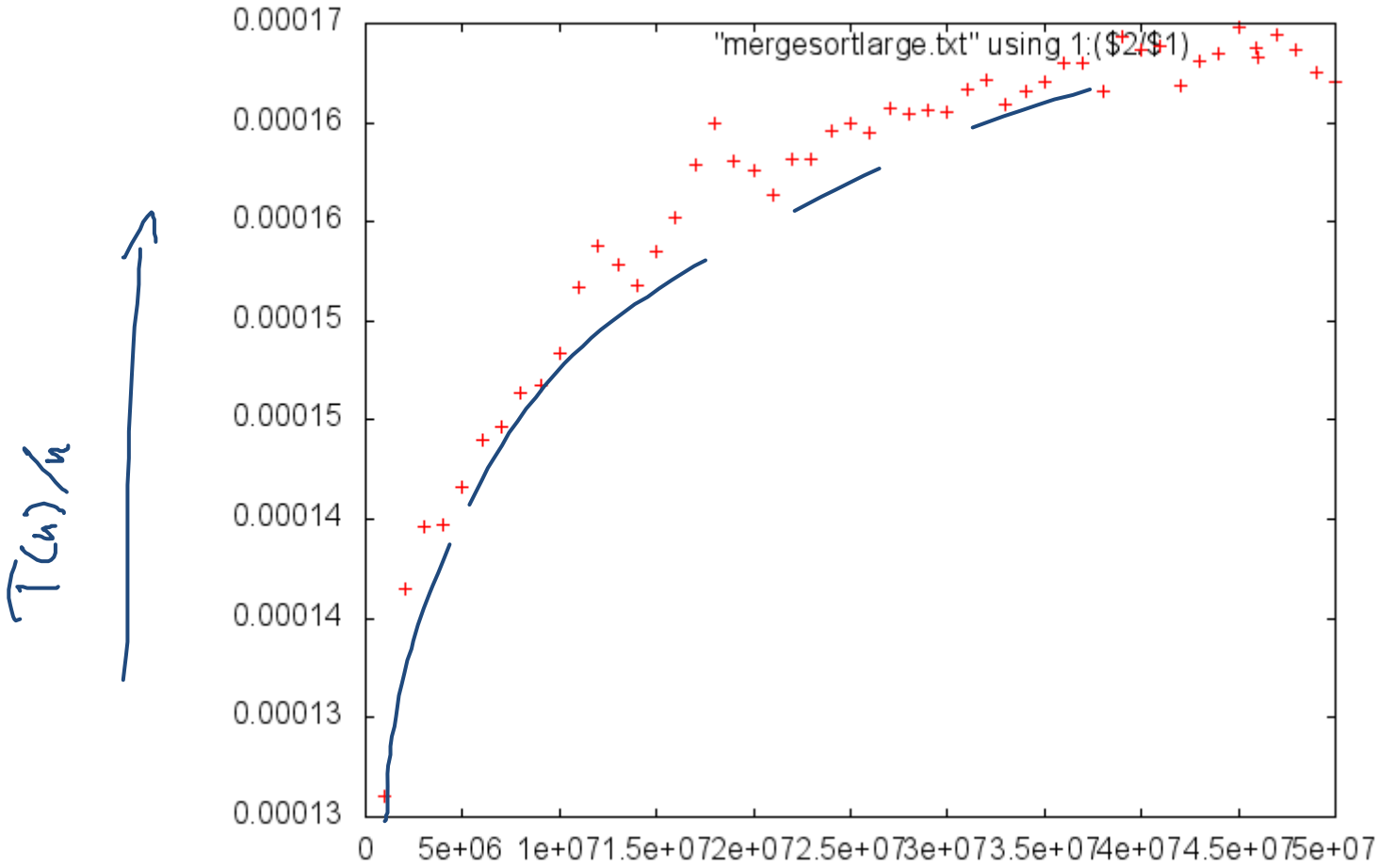
Merge Sort Messungen für grössere n



Gnuplot:

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

Merge Sort Messungen für grössere n



Gnuplot:

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

Zusammenfassung Analyse Merge Sort

UNI FREIBURG

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

- wächst fast linear mit der Grösse der Eingabe…

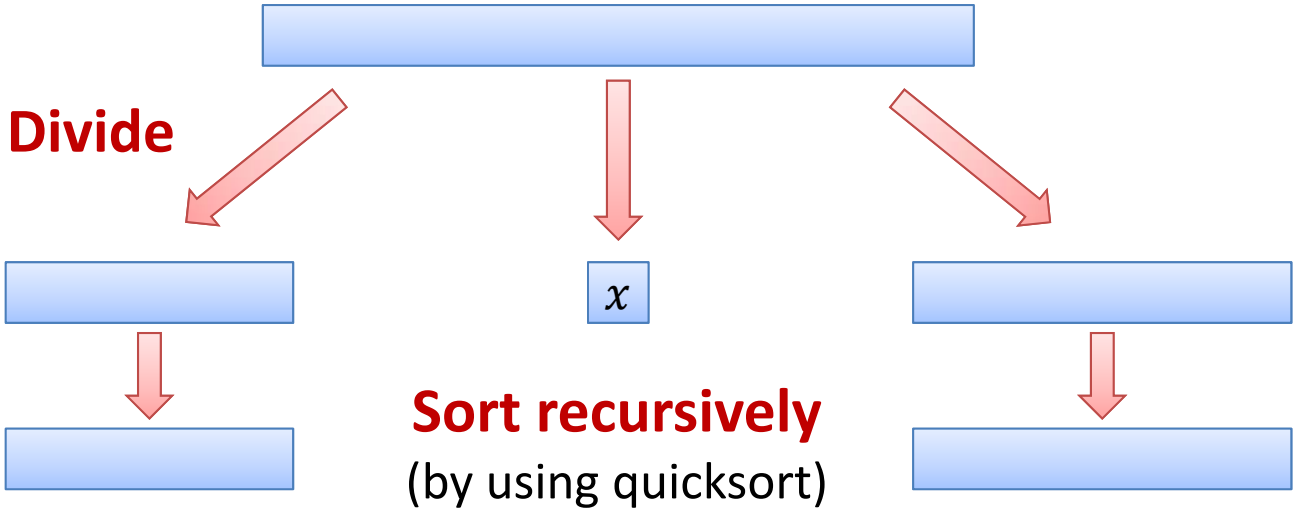
Wie gut ist das?

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

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

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

Quick Sort : Analyse



- Laufzeit hängt davon ab, wie gut die Pivots sind
- Laufzeit, um Array der Länge n zu sortieren, falls das Pivot in Teile der Grösse λn und $(1-\lambda)n$ partitioniert:

$$\boldsymbol{T(n) = T(\lambda n) + T\big((1-\lambda)n\big) + \text{"Pivotsuche + Divide"}}$$

$\text{"Pivotsuche + Divide"} = O(n)$

- **Divide:**
  - Wir gehen einmal von beiden Seiten über's Array mit konstanten Kosten pro Schritt → Zeit, um Array der Länge n zu partitionieren: $O(n)$

Quick Sort : Analyse

Falls wir in $O(n)$ Zeit ein Pivot finden können, welches das Array in Teile der Grösse λn und $(1-\lambda)n$ unterteilt:

- Es gibt eine Konstante $b > 0$, so dass $\lambda \in [\frac{1}{4}, \frac{1}{2}]$

$$T(n) \leq T(\lambda n) + T\big((1-\lambda)n\big) + b \cdot n, \qquad T(1) \leq b$$

Extremfall I) $\lambda = 1/2$ (best case):

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

- Wie bei Merge Sort: $T(n) \in O(n \log n)$

Extremfall II) $\lambda n = 1$, $(1-\lambda)n = n-1$ (worst case):

$$T(n) = T(n-1) + bn, \qquad T(1) \leq b$$

Quick Sort : Worst Case Analyse

Extremfall II) $\lambda n = 1, (1-\lambda)n = n-1$ (worst case):

$$T(n) \leq T(n-1) + bn, \qquad T(1) \leq b$$

In dem Fall, ergibt sich $T(n) \in \Theta(n^2)$:

Quick Sort worst case $T(n) \in O(n^2)$

$T(n) \leq T(n-1) + bn$

$\leq T(n-2) + b(n-1) + bn$

$\leq T(n-3) + b(n-2) + b(n-1) + bn$

$\vdots$

$\leq T(1) + b(n + (n-1) + (n-2) + \dots + 2)$

$\leq \frac{n(n+1)}{2} - 1$

$\leq \frac{n(n+1)}{2} \cdot b$

Verankerung:

$T(1) \leq \frac{1 \cdot 2}{2} \cdot b = b$ ✓

Schritt:

$T(n) \leq T(n-1) + b \cdot n$

I.V. $\leq \frac{n(n-1)}{2} \cdot b + b \cdot n$

$= \left(\frac{n(n+1)}{2} - n\right) \cdot b + bn$ ✓

$\frac{n(n-1)}{2}$

□

Quick Sort mit zufälligem Pivot

Aufteilung bei zufälligem Pivot:

- Laufzeit $\boldsymbol{T(n) = O(n \log n)}$ für alle Eingaben
  - allerdings nur im Erwartungswert, bzw. mit sehr grosser Wahrscheinlichkeit

Intuition:

- Mit Wahrscheinlichkeit $1/2$, haben die Teile Grösse $\geq n/4$, so dass

$$T(n) \leq T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + bn$$

in-place