Informatik II - SS 2014

(Algorithmen & Datenstrukturen)

Vorlesung 3 (6.5.2014)

O-Notation, Asymptotische Analyse, Sortieren III

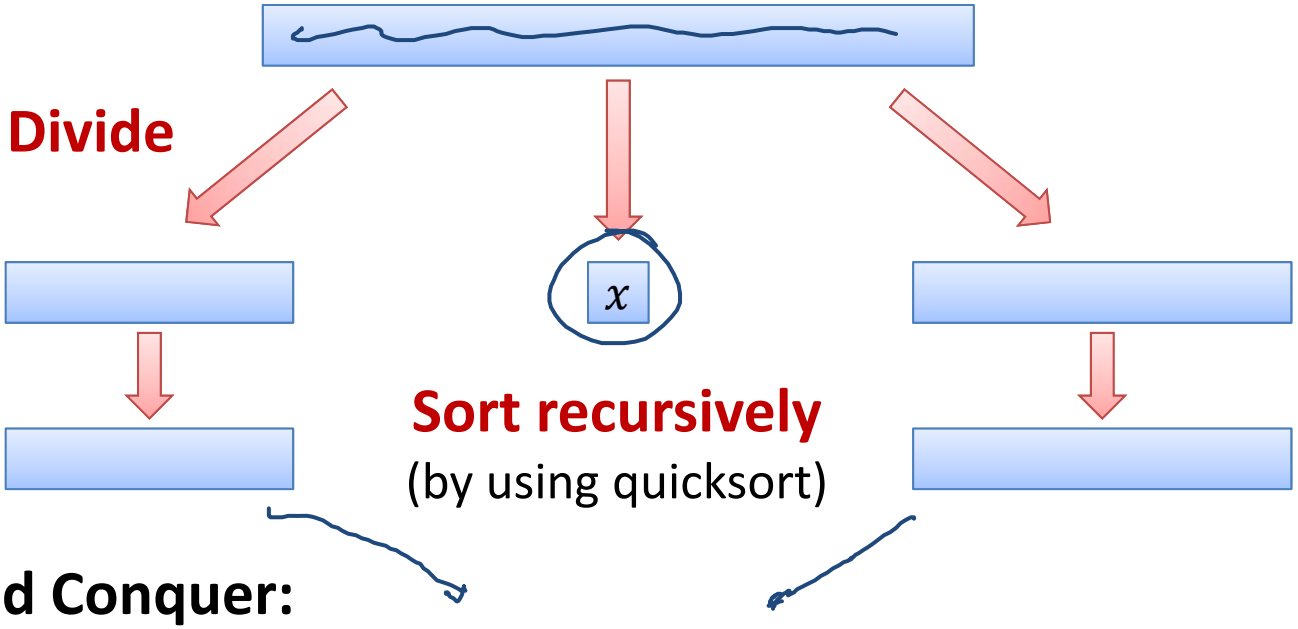
UNI FREIBURG

Fabian Kuhn

Algorithmen und Komplexität

Divide and Conquer

Übersicht QuickSort:



Divide and Conquer:

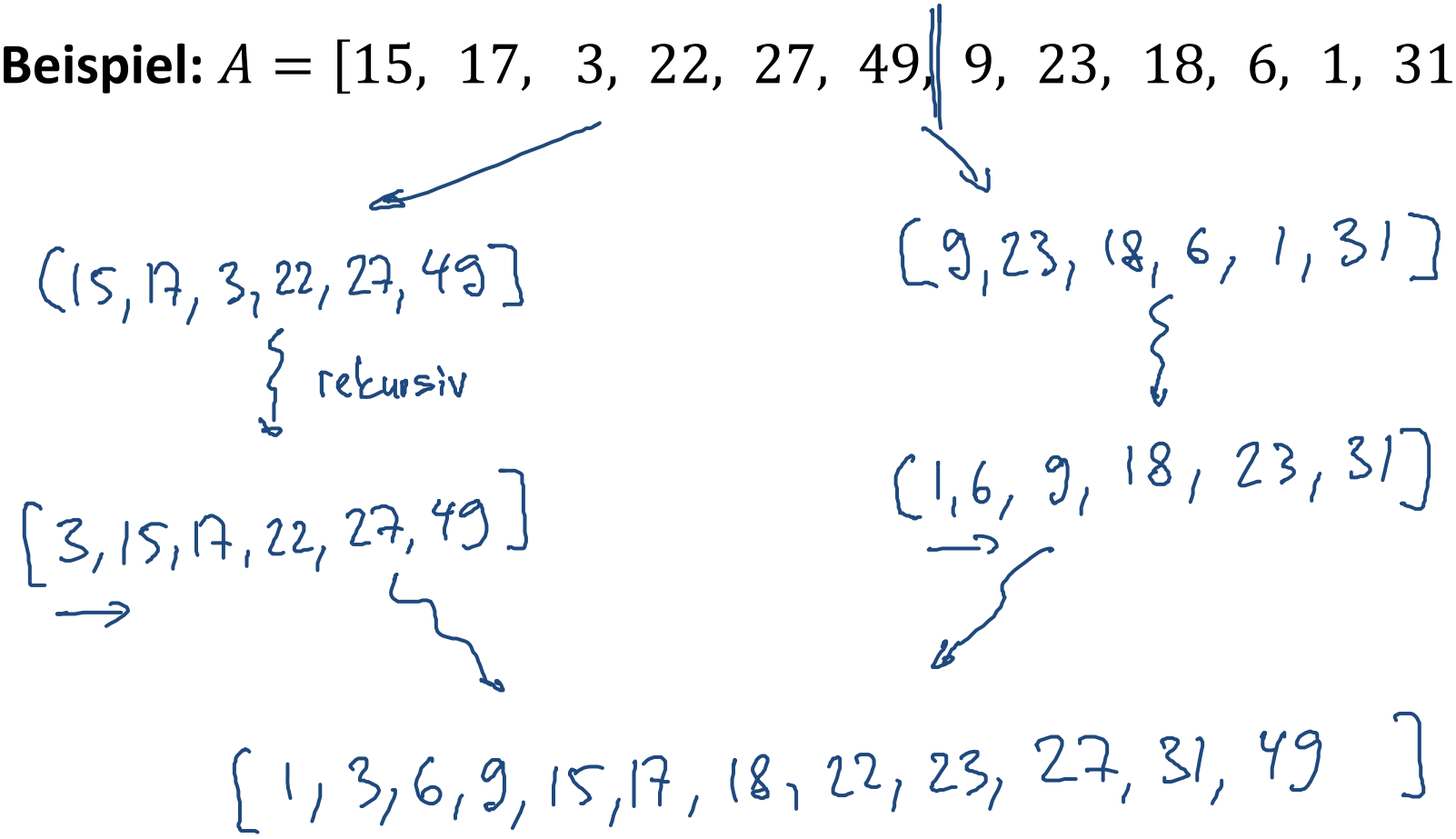
- Verbreitetes Prinzip für den Algorithmenentwurf

1. Teile Eingabe in 2 oder mehrere kleinere Teilprobleme
2. Löse die Teilprobleme rekursiv
3. Kombiniere die Teillösungen zur Gesamtlösung

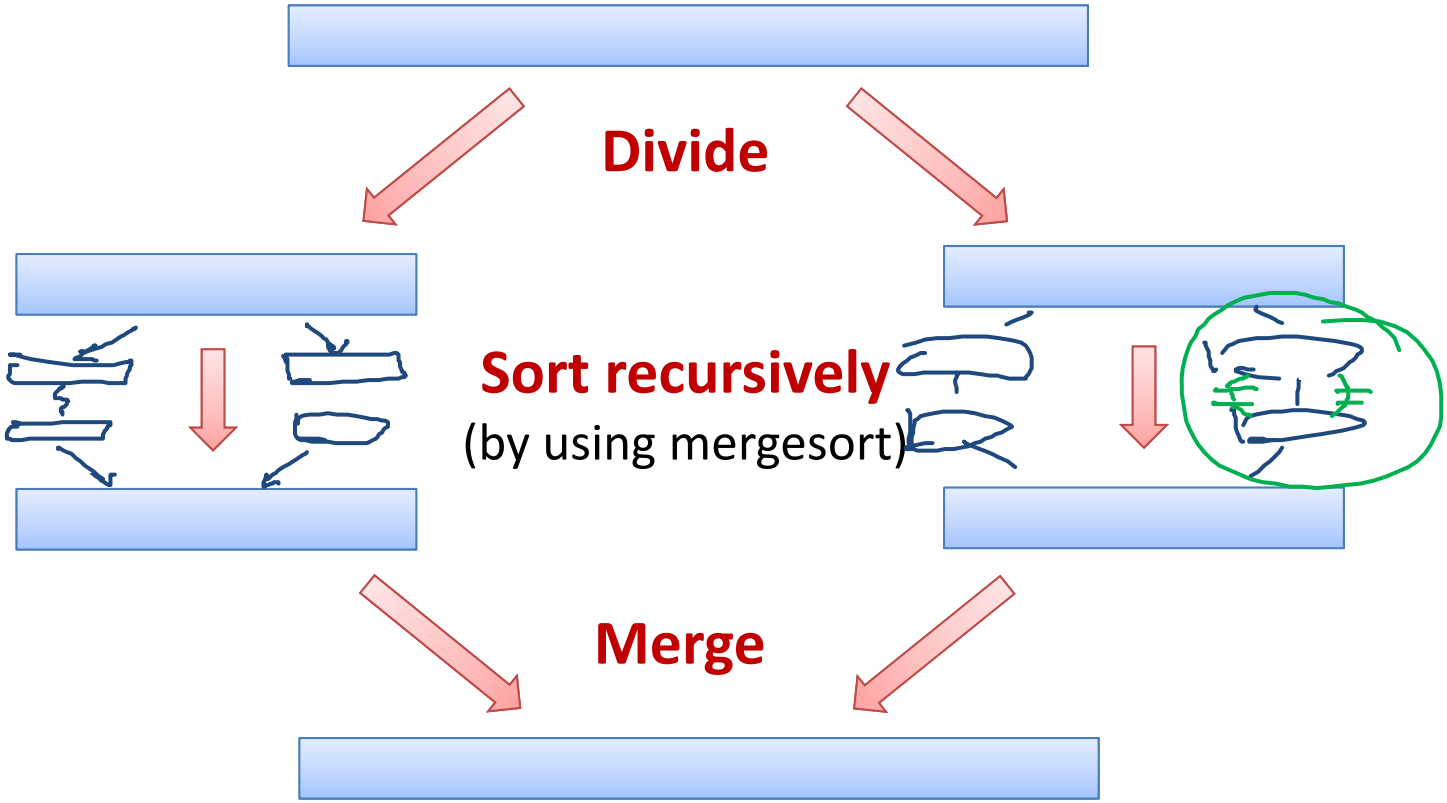
MergeSort

- Ein weiterer Sortieralgorithmus, welcher auf dem Divide-and-Conquer Prinzip basiert

Beispiel: $A = [15,\ 17,\ 3,\ 22,\ 27,\ 49,\ 9,\ 23,\ 18,\ 6,\ 1,\ 31]$

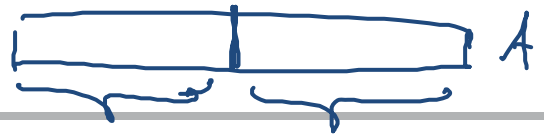


Übersicht MergeSort



- Divide ist bei MergeSort trivial
- Merge (kombinieren der Lösungen) benötigt dafür Arbeit...

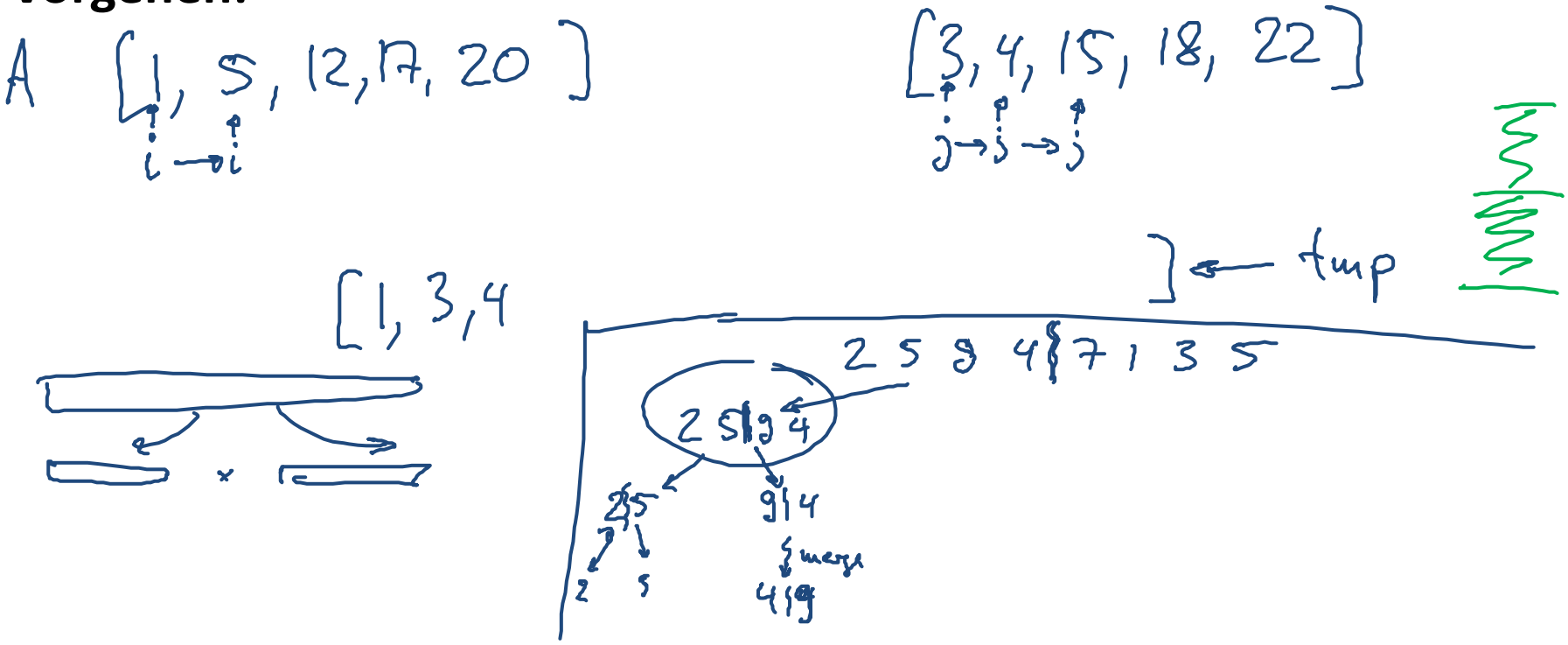
MergeSort: Merge-Schritt



Verschmelzen (merge) von zwei sortierten Arrays:

- Gegeben: sortierte Arrays A und B der Länge n und m
- Ausgabe: sortiertes Array C mit den Elementen von A und B

Vorgehen:

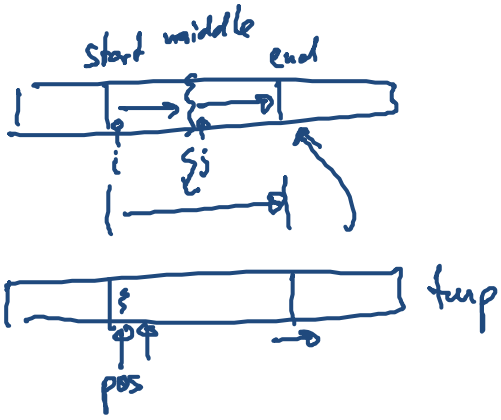


MergeSort: Pseudocode

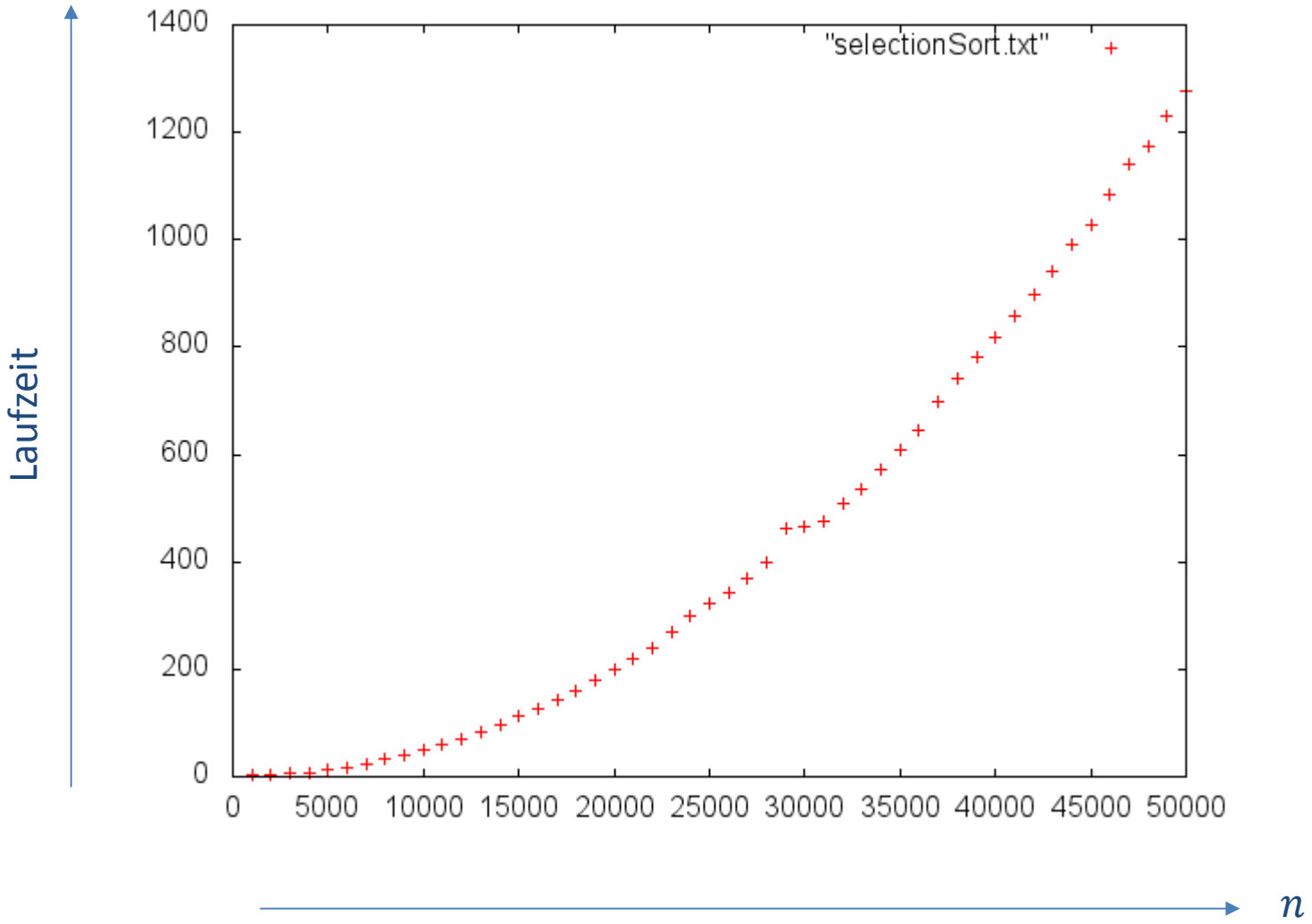
Eingabe: Array A der Grösse n

start + $\left\lfloor \frac{\text{end} - \text{start}}{2} \right\rfloor$

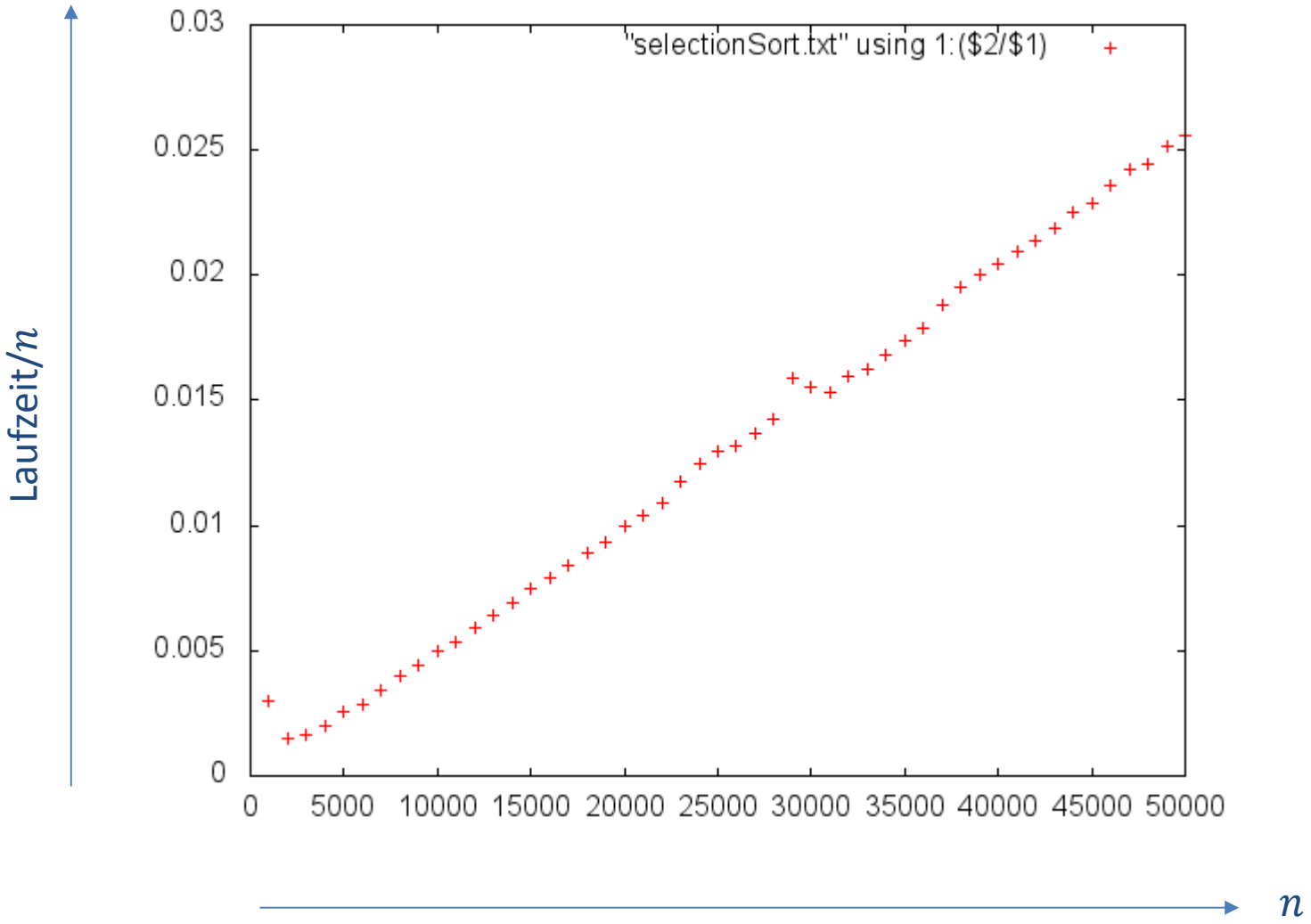
```
MergeSort(A):
 1: allocate array tmp to store intermediate results
 2: MergeSortRecursive(A, 0, n, tmp)

MergeSortRecursive(A, start, end, tmp)          // sort A[start..end-1]
 1: if (end – start > 1) then
 2:   middle = start + (end – start) / 2        // int. division
 3:   MergeSortRecursive(A, start, middle, tmp)
 4:   MergeSortRecursive(A, middle, end, tmp)
 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]
```



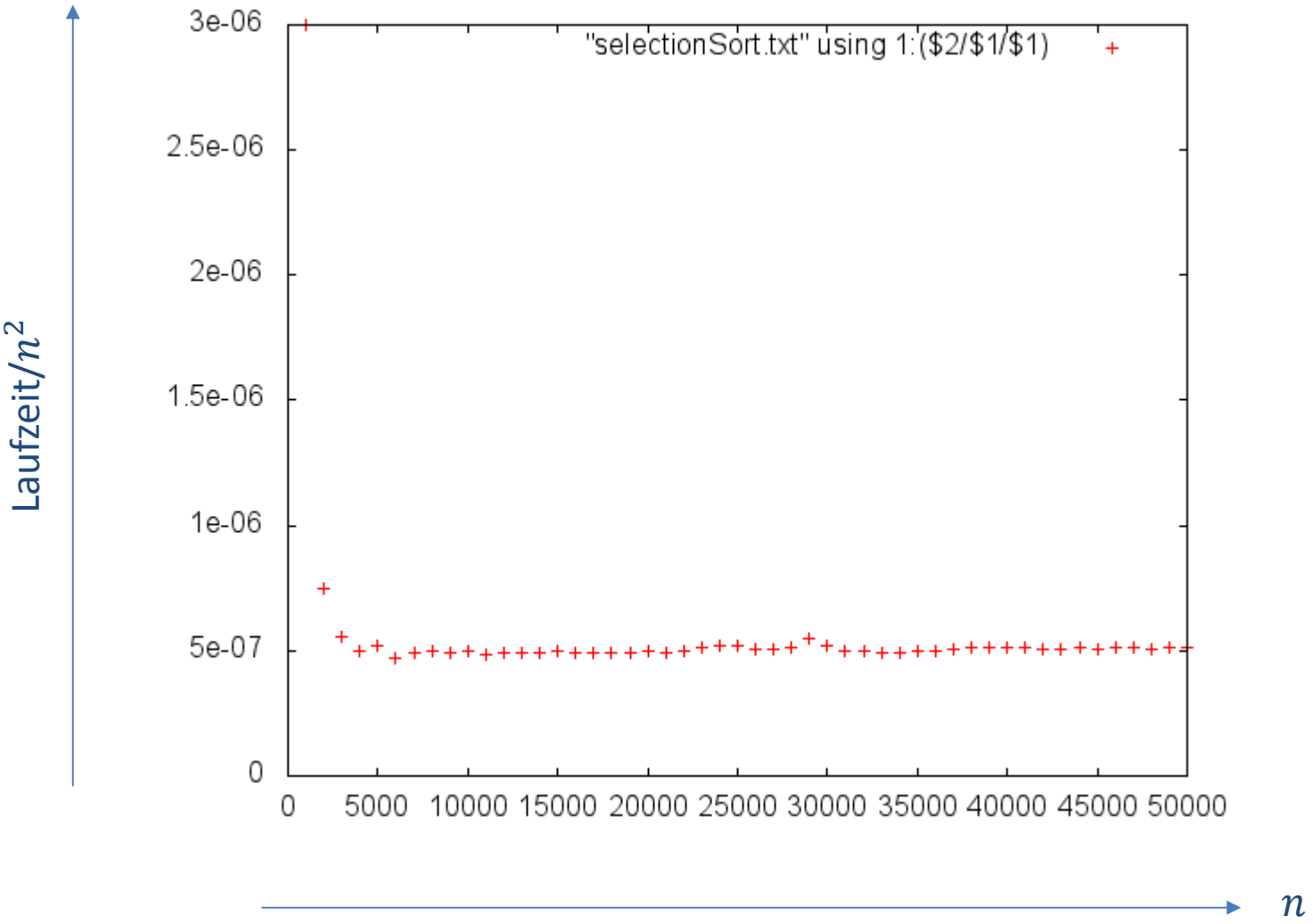
Zeitmessung SelectionSort



Zeitmessung SelectionSort



Zeitmessung SelectionSort



Laufzeitanalyse I

UNI FREIBURG

- 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

Grundoperation

UNI FREIBURG

Was ist eine Grundoperation?

- Einfache arithmetische Operationen (auf Integers)
  - +, -, *, /, % (mod), …
- Ein Speicherzugriff
  - Variable auslesen, Variablenzuweisung
  - Ist das wirklich eine Grundoperation?

$x = 7$

$y = 2 \cdot x$

- 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.

RAM Modell

UNI FREIBURG

RAM = Random Access Machine

- Standardmodell(e), um Algorithmen zu analysieren!
- Grundoperationen (wie “definiert”) benötigen alle eine Zeiteinheit
- Insbesondere sind alle Speicherzugriffe gleich teuer:

  Jede Speicherzelle (1 Maschinenwort) kann in 1 Zeiteinheit gelesen, bzw. beschrieben werden
  - ignoriert insbesondere Speicherhierarchien
  - Ist aber in den meisten Fällen eine vernünftige Annahme

- Alternative abstrakte Modelle existieren:
  - um Speicherhierarchien explizit abzubilden
  - bei riesigen Datenmengen (vgl. «Buzzword» Big Data)
    - z.B.: Streaming-Modelle: Speicher muss sequentiell gelesen werden
  - für verteilte/parallele Architekturen
    - Speicherzugriff kann lokal oder über’s Netzwerk sein…

Laufzeitanalyse II

UNI FREIBURG

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 ob/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

```
SelectionSort(A):
 1: for i=0 to n-2 do
 2:    minIdx = i
 3:    for j=i to n-1 do
 4:       if A[j] < A[minIdx] then
 5:          minIdx = j
 6:    swap(A[i], A[minIdx])
```

$\leq c_1$

Konstante $c > 0$

$\#op. \leq c \cdot \#(\text{Schleifendurchläufe } *) = x(n)$

$\#op \leq c \cdot n^2$

$$x(n) = \sum_{i=0}^{n-2} (n-i) = \sum_{j=2}^{n} j \leq \sum_{j=1}^{n} j = \frac{n(n+1)}{2} \leq n^2$$

Selection Sort: Obere Schranke

UNI FREIBURG

$\boldsymbol{T(n)}$**:** Anzahl Grundop. von Selection Sort bei Arrays der Länge n

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

Beweis:

$$T(n) = \sum_{j=2}^{n} j = \left(\sum_{j=1}^{n} j\right) - 1 = \frac{n(n+1)}{2} - 1 \geq \frac{n^2}{2}$$

Selection Sort: Untere Schranke

UNI FREIBURG

$\boldsymbol{T(n)}$**:** Anzahl Grundop. von Selection Sort bei Arrays der Länge n

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

Beweis:

$T(n) = \#op \geq \#(\text{Schleifendurchläufe})$, $\#(\text{Schleifendurchläufe}) = x(n)$

$x(n) \geq \frac{n^2}{2}$

$c_L \cdot n^2 \leq T(n) \leq c_U \cdot n^2$

Laufzeitanalyse

UNI FREIBURG

Zusammenfassung

- Wir können nur eine Grösse berechnen, welche proportional zur Laufzeit ist
- Wir wollen auch gar nichts anderes berechnen:
  - Analyse sollte unabhängig von Computer / Compiler / etc. sein
  - Wir wollen Aussagen, welche auch in 10/100/… Jahren noch Gültigkeit haben

- Wir werden immer Aussagen der folgenden Art haben:

  Es gibt eine Konstante C, so dass

$$T(n) \leq C \cdot f(n) \quad \text{oder} \quad T(n) \geq C \cdot f(n)$$

- Um dies zu vereinfachen / verallgemeinern gibt's die O-Notation…

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:

$$T(n) \in O(f(n))$$

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

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

- Bei Selection Sort:

$T(n) \in O(n^2)$, $T(n) \in \Omega(n^2)$ $\Longrightarrow$ $T(n) = \Theta(n^2)$

Landau-Symbole : Definitionen $g: \mathbb{N} \to \mathbb{R}^+$

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

- 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 $n \geq n_0$

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

- 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 $n \geq n_0$

$$\boldsymbol{\Theta(g(n))} \coloneqq \boldsymbol{O(g(n))} \cap \boldsymbol{\Omega(g(n))}$$

- 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

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

- 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)

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

- 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:

$$f(n) \in o\big(g(n)\big) \quad \Longrightarrow \quad f(n) \in O\big(g(n)\big)$$

$$f(n) \in \omega\big(g(n)\big) \quad \Longrightarrow \quad f(n) \in \Omega\big(g(n)\big)$$

Landau-Symbole : Intuitiv

UNI FREIBURG

$\boldsymbol{f(n) \in O(g(n))}$:

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

$\boldsymbol{f(n) \in \Omega(g(n))}$:

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

$\boldsymbol{f(n) \in \Theta(g(n))}$:

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

Landau-Symbole : Intuitiv

UNI FREIBURG

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

- $f(n)$ " $\leq$ " $g(n)$, asymptotisch gesehen…
- $f(n)$ wächst asymptotisch langsamer als $g(n)$

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

- $f(n)$ " $\geq$ " $g(n)$, asymptotisch gesehen…
- $f(n)$ wächst asymptotisch schneller als $g(n)$

$f(n+1) \geq f(n)$

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

$T(n) \in O(n^2)$

$$f(n) \in O\big(g(n)\big), \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

$$f(n) \in \Omega\big(g(n)\big), \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

$$f(n) \in \Theta\big(g(n)\big), \qquad 0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

$$f(n) \in o\big(g(n)\big), \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) \in \omega\big(g(n)\big), \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Landau-Notation : Bemerkungen

Schreibweise:

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

Asymptotisches Verhalten für allgemeine Grenzwerte:

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

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

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

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

- Wir verwenden die 1. Definition
- Macht nur bei nicht-monotonen Funktionen einen Unterschied

Landau-Notation : Beispiele

$O(\log n)$

Selection Sort:

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

$$T(n) \in O(n^2), \qquad T(n) \in \Omega(n^2), \qquad T(n) \in \Theta(n^2)$$

- $T(n)$ wächst schneller als linear: $T(n) \in \omega(n)$ $\quad T(n) = o(n^3)$

Weitere Beispiele:

$\log_a b = \frac{\log_c b}{\log_c a}$

- $f(n) = 10n^3,\ g(n) = n^3/1000$: $10n^3 \in \Theta(n^3/1000)$
- $f(n) = e^n,\ g(n) = n^{100}$: $f(n) \in \omega(g(n))$
- $f(n) = n/\log_2 n,\ g(n) = \sqrt{n}$: $\frac{f(n)}{g(n)} = \frac{\sqrt{n}}{\log_2 n} = \frac{2^{x/2}}{x}$ $\quad f(n) \in \omega(g(n))$
- $f(n) = n^{1/256},\ g(n) = 10 \ln n$: $f(n) \in \omega(g(n))$
- $f(n) = \log_{10} n, g(n) = \log_2 n$: $f(n) \in \Theta(g(n))$ $\quad \log_{10} n = \frac{\ln n}{\ln 10}$
- $f(n) = n^{\sqrt{n}}, g(n) = 2^n$:

$\sqrt{n} \cdot \log_2 n$ $\qquad$ n