

Informatik II - SS 2014

(Algorithmen & Datenstrukturen)

Vorlesung 2 (22.4.2016)

Sortieren II



**UNI
FREIBURG**

Fabian Kuhn

Algorithmen und Komplexität

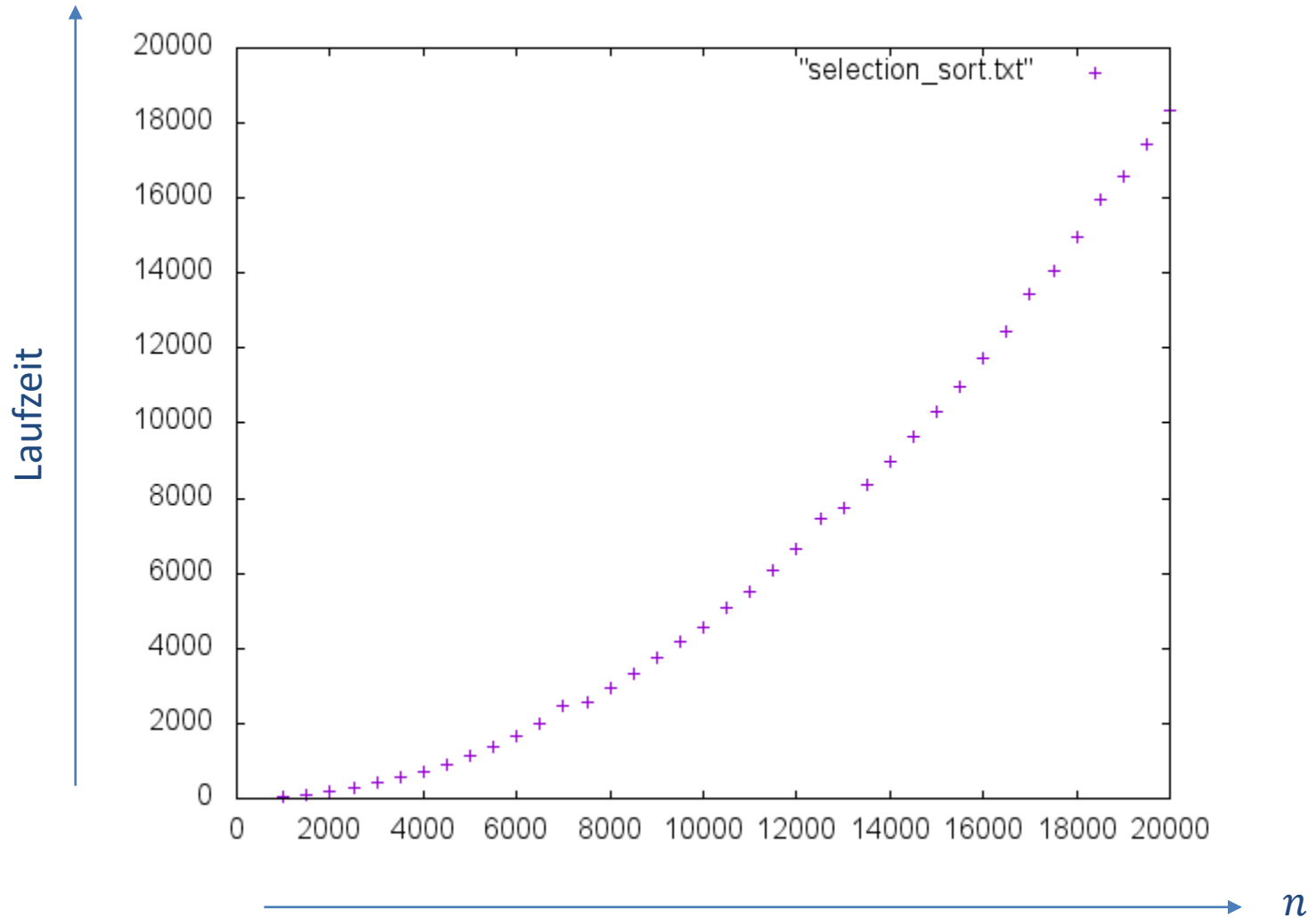
SelectionSort: Programm

Schreiben wir doch das gleich mal als Java/C++ - Programm um...

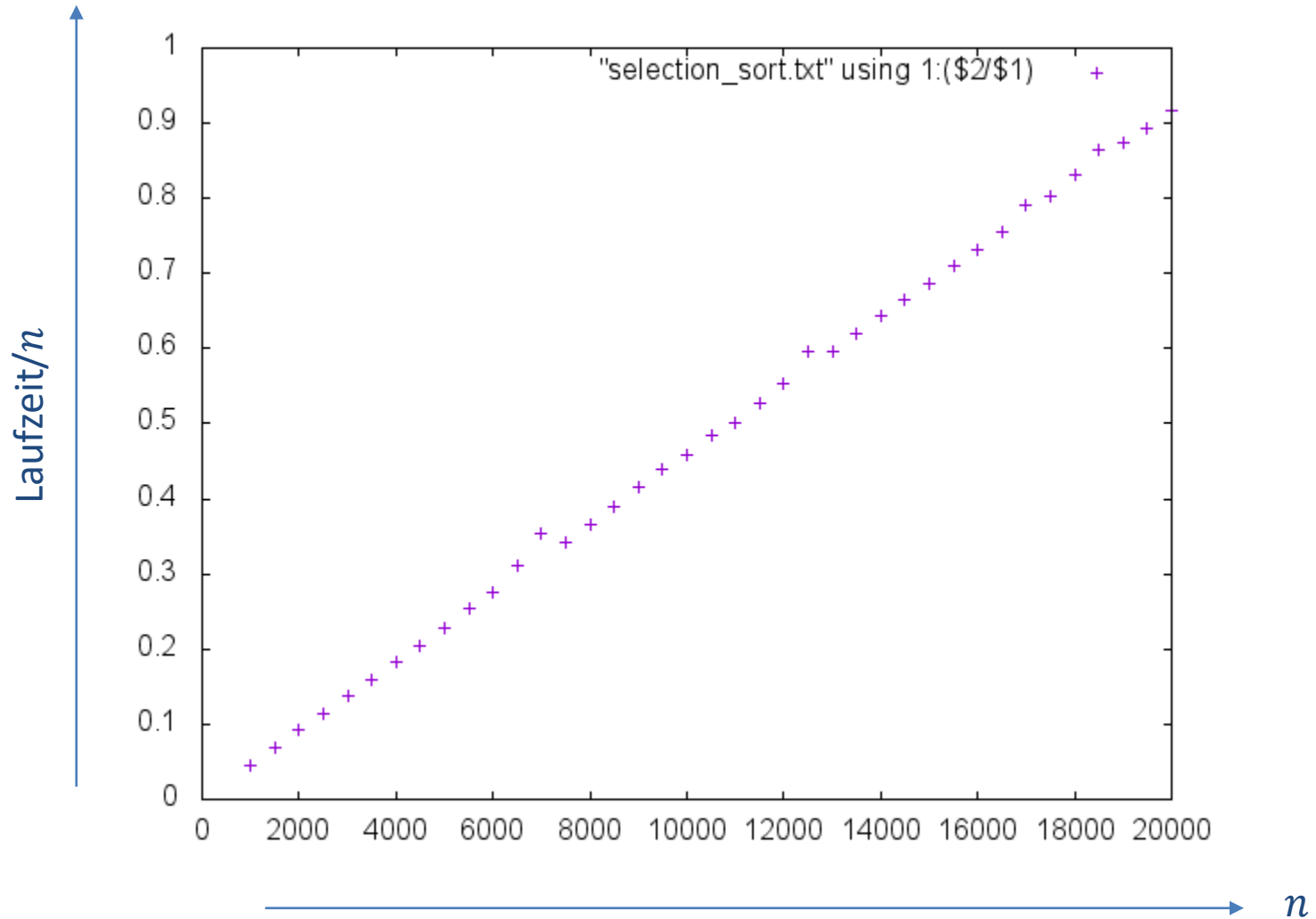
Dabei kann ich auch die Umgebung für die Übungen demonstrieren:

- **Unit Tests** : [junit](#) / [gtest](#)
- **Coding Standards** : [checkstyle](#) / [cpplint](#)
- **Build-Framework** : [ant](#) / [make](#)
- **Build-System** : [jenkins](#)
- **File-Repository** : [SVN](#)

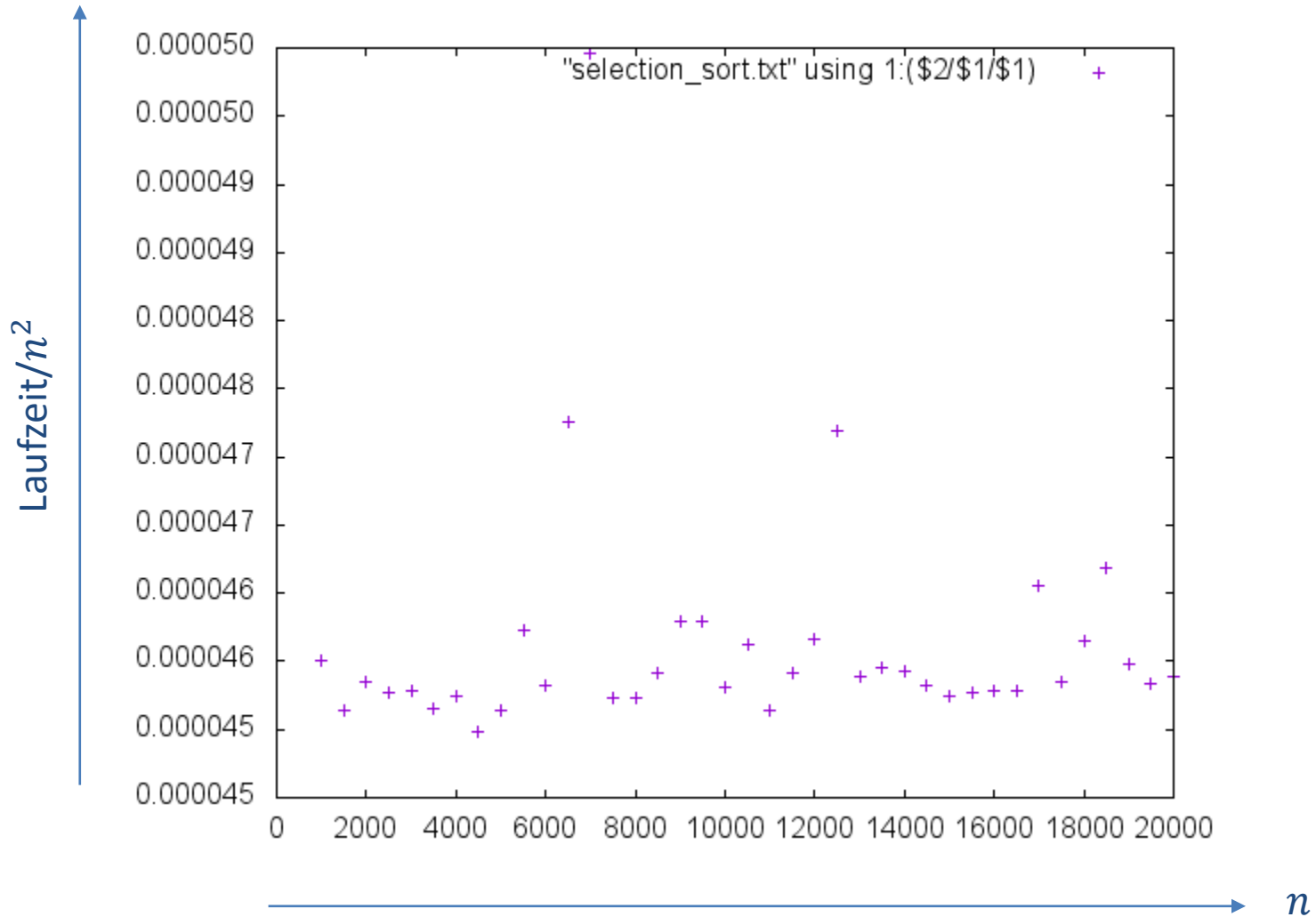
Zeitmessung SelectionSort



Zeitmessung SelectionSort



Zeitmessung SelectionSort



Zeitmessung Selection Sort:

- Scheint mit wachsender Grösse des Arrays unverhältnismässig langsamer zu werden
- Die Zeit scheint etwa quadratisch mit der Grösse des Arrays zu wachsen
 - doppelt so grosses Array \rightarrow 4 x so lange Laufzeit
 - dreimal so grosses Array \rightarrow 9 x so lange Laufzeit
 - ...
- Wir werden sehen, dass die Laufzeit tatsächlich quadratisch ist
 - und wie man das formal korrekt analysiert und ausdrückt
- Zuerst überlegen wir, wie man sonst noch sortieren kann...

- Anfang (Präfix) des Arrays ist sortiert
 - Am Anfang nur das erste Element, mit der Zeit mehr...
- Füge schrittweise immer das nächste Element in den bereits sortierten Teil ein.

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

Insertion Sort: Pseudocode

Eingabe: Array A der Grösse n

InsertionSort(A):

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


Bubble Sort

- Gehe durch's Array, vertausche benachbarte Elemente, falls sie in der falschen Reihenfolge sind
- Wiederhole bis das Array sortiert ist...

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

Bubble Sort: Pseudocode

Eingabe: Array A der Grösse n

BubbleSort(A):

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

- Wir werden sehen: Insertion Sort und Bubble Sort sind auch nicht besser als Selection Sort ...

QuickSort : Idee

1. Teile Array in zwei Teile auf
 - linker Teil: kleine Elemente, rechter Teil: grosse Elemente
2. Sortiere die beiden Teile rekursiv!

Informelle Beschreibung:

1. Teile Array in linken und rechten Teil, so dass

Elemente links \leq Elemente rechts

- Bemerkung: Die Elemente in beiden Teilen müssen noch nicht sortiert sein
2. a) **Sortiere die Elemente im linken Teil rekursiv**
b) **Sortiere die Elemente im rechten Teil rekursiv**
 - Rekursion: “löse ein kleineres Teilproblem der gleichen Art mit der gleichen Methode wie das Hauptproblem”
- Sobald die Teilprobleme so klein werden, dass Sortieren trivial wird, endet die Rekursion
 - spätestens wenn die Teile nur noch aus einem Element bestehen

QuickSort : Beispiel

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

- Wir müssen so aufteilen, dass
Elemente im linken Teil \leq Elemente im rechten Teil
- **Idee:** Wähle ein **Pivot x** , welches bestimmt wo die Mitte ist
 - Elemente $< x$ müssen nach links
 - Elemente $> x$ müssen nach rechts
 - Bei Elementen $= x$ ist's egal...
- **Algorithmus für Divide** (oft auch Partition genannt):
 - Idee: Iteriere von links und von rechts über's Array
 - Wenn man ein Element trifft, das auf der richtigen Seite ist, muss man nichts tun und kann weiter gehen.
 - Bei einem Element, welches die Seite wechseln muss, kann man mit einem Element auf der anderen Seite vertauschen, welches auch die Seite wechseln muss.

Algorithmus für Divide (etwas formaler):

- Aufgabe: Divide array A der Länge n anhand von Pivot x
 - Annahme: Elem. $\leq x$ gehen nach links, elem. $> x$ gehen nach rechts
- Generelles Vorgehen:
 - Zwei Variablen l und r , um von links und rechts durch's Array zu gehen
 - Inkrementiere l bis $A[l] > x$ (Element muss nach rechts)
 - Dekrementiere r bis $A[r] \leq x$ (Element muss nach links)
 - Vertausche, und stelle l und r eins vor ($l += 1, r -= 1$)
 - Divide fertig, sobald sich l und r treffen
- Die Details müssen Sie in der Übung selbst ausarbeiten...

QuickSort Divide : Beispiel

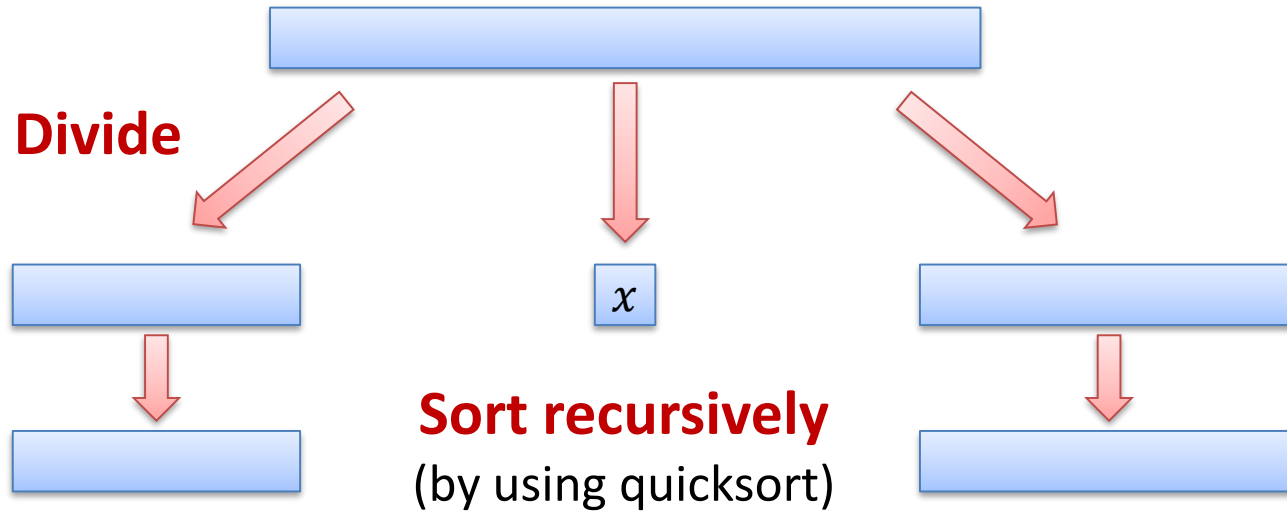
Beispiel: $A = [15, 17, 3, 22, 27, 49, 9, 23, 18, 6, 1, 31]$

- Wie gross die zwei Teile beim Divide werden, hängt von der Wahl des Pivots ab...
- Wir werden sehen: Der Algorithmus ist am schnellsten, wenn die zwei Teile möglichst gleich gross sind.

Strategien zur Bestimmung des Pivots:

- Ideal wäre der **Median** → kann nicht einfach gefunden werden...
 - werden wir noch anschauen...
- Ein **fixes Element** des Arrays (z.B. immer das erste des Bereichs)
 - kann zu sehr ungleichen Teilen führen...
- Ein Element an einer **zufälligen Position** (innerhalb des Bereichs)
 - Randomized QuickSort meint meistens genau das
 - Wird meistens vernünftig grosse Teile liefern
- **Median von drei** (oder mehr) zufälligen Elementen
 - etwas “teurer”, dafür werden die Teile “gleicher”

Übersicht QuickSort:

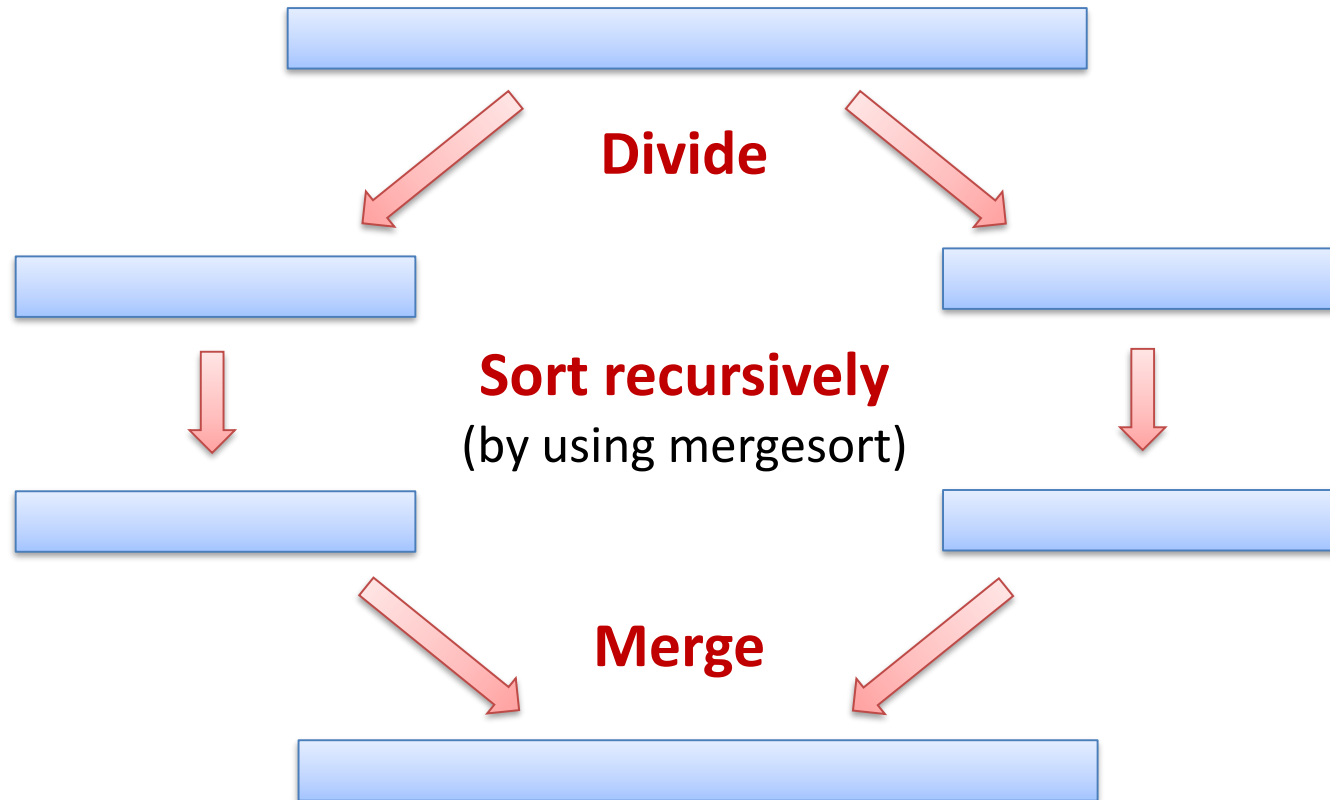


Divide and Conquer:

- 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

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

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



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

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

MergeSort(A):

- 1: allocate array tmp to store intermediate results
- 2: MergeSortRecursive(A , θ , 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** $A[i] < A[j]$ **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** $A[i] = tmp[i]$