

Sortieren

- Die bislang betrachteten Sortierverfahren (Bubblesort, Insertionsort) lagen in $O(n^2)$
- Geht es besser?

- Gegeben:
 - Ein Array von unsortierten Daten
 - Zur Vereinfachung: Keine Duplikate
- Ablauf
 - Wähle ein beliebiges Element aus dem Array aus, das sogenannte **Pivotelement**
 - Teile das Array in zwei Teile, so dass gilt:
 - links vom Pivotelement befinden sich nur Elemente, die kleiner sind als das Pivotelement
 - rechts vom Pivotelement befinden sich nur Elemente, die größer sind als das Pivotelement
 - Wende dieses Vorgehen auf das linke und rechte Teilarray an, solange diese aus mehr als einem Element bestehen

Quicksort - Beispiel (1)

Sortieren der Zahlenfolge 17, 12, 3, 9, 19, 8, 21 in aufsteigender Reihenfolge:

17	12	3	[9]	19	8	21
----	----	---	-----	----	---	----

i 17	12	3	[9]	19	8	21
-----------	----	---	-----	----	---	----

i 17	12	3	[9]	19	8	j 21
-----------	----	---	-----	----	---	-----------

i 17	12	3	[9]	19	j 8	21
-----------	----	---	-----	----	----------	----

i 8	12	3	[9]	19	j 17	21
----------	----	---	-----	----	-----------	----

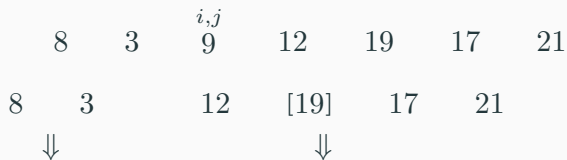
8	i 12	3	[9]	19	j 17	21
---	-----------	---	-----	----	-----------	----

8	i 12	3	[9]	j 19	17	21
---	-----------	---	-----	-----------	----	----

8	i 12	3	j [9]	19	17	21
---	-----------	---	------------	----	----	----

Quicksort - Beispiel (2)

8	$\overset{i}{12}$	3	$\overset{j}{[9]}$	19	17	21
8	$\overset{i}{[9]}$	3	$\overset{j}{12}$	19	17	21
8	$\overset{i}{[9]}$	$\overset{j}{3}$	12	19	17	21
8	3	$\overset{i,j}{[9]}$	12	19	17	21

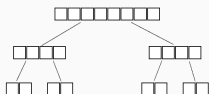


Rekursion

- Implementiere Quicksort
 - Vervollständige die Methoden **sortiere** und **teile**
 - Die Parameter **links** und **rechts** definieren die linke bzw. rechte Grenze des zu sortierenden Teilarrays

Laufzeit Quicksort (1)

- Bei “guter” Aufteilung des Arrays ergibt sich die folgende Struktur:



- Der Aufwand pro Ebene liegt in $O(n)$
- Die Anzahl an Ebenen beträgt $\log_2 n$
- Laufzeit Quicksort: $O(n \cdot \log n)$
 - Aaaaaaaaaber...

- Wir wählen das Pivotelement zufällig
- Ist die Aufteilung des Arrays also immer “gut”?
- Worst Case:
 - Das Pivotelement ist immer das kleinste (oder größte) Element des aktuellen Teilarrays
 - In diesem Fall fällt mit jedem rekursiven Aufruf nur ein Element weg
- Laufzeit Quicksort im Worst Case: $O(n^2)$

- Quicksort arbeitet nach dem Prinzip “Divide and Conquer” (Teile und Herrsche)
 - Rekursive Zerlegung in kleinere Teilprobleme, bis die Problemgröße “beherrschbar” ist
 - Rekonstruktion der Gesamtlösung aus den Teillösungen