

Sortieren

- Basiert **nicht** auf Elementvergleichen
- Stattdessen:
 - Aufteilen der zu sortierenden Werte nach der Wertigkeit der einzelnen Stellen
 - Im Zehnersystem also: Einer, Zehner, Hunderter usw.

- Zwei Phasen, die für jede Stelle durchgeführt werden (beginnend mit der Einerstelle):
 1. Partitionieren
 - Anhand des jeweiligen Stellenwerts wird jede Zahl in ein passendes “Gefäß” gesteckt (im Zehnersystem: 0 bis 9)
 2. Sammeln
 - Nachdem alle Zahlen auf “Gefäße” verteilt wurden, werden sie wieder “eingesammelt”, beginnend mit dem Gefäß für die Ziffer 0

Beispiel (1)

- Zu sortieren: 321, 526, 138, 217, 923
- Partitionieren anhand der Einerstelle

Gefäß:	0	1	2	3	4	5	6	7	8	9
Inhalt:		321		923			526	217	138	

- Sammeln von links nach rechts: 321, 923, 526, 217, 138

Beispiel (2)

- Zu sortieren: 321, 923, 526, 217, 138
- Partitionieren anhand der Zehnerstelle

Gefäß:	0	1	2	3	4	5	6	7	8	9
Inhalt:		217	321	138						
			923							
			526							

- Sammeln von links nach rechts: 217, 321, 923, 526, 138
- Wichtig:
 - Bei mehreren Zahlen in einem Gefäß muss die Reihenfolge erhalten bleiben (FIFO: First In, First Out)

Beispiel (3)

- Zu sortieren: 217, 321, 923, 526, 138
- Partitionieren anhand der Hunderterstelle

Gefäß:	0	1	2	3	4	5	6	7	8	9
Inhalt:		138	217	321		526				923

- Sammeln von links nach rechts: 138, 217, 321, 526, 923

- Implementiere Radixsort
 - Verwende für die Gefäße Objekte vom Typ `LinkedList<Integer>`
 - Ermittle die jeweiligen Stellenwerte (Einer, Zehner usw.) rechnerisch
 - Verwende ggf. die Vorlage

- Implementiere Radixsort im Zweiersystem
 - Wie viele Gefäße gibt es dann?
 - Wie können die mathematischen Operationen in diesem Fall effizienter implementiert werden?

- Sei n die Anzahl der zu sortierenden Werte und x deren Stellenzahl
- Es werden also $x \cdot n$ Durchläufe benötigt
 - Falls die Größe der Werte begrenzt ist, so ist x eine Konstante!
- Laufzeit Radixsort: $O(n)$