

# Algorithmen und Datenstrukturen

## A13. Sortieren: Countingsort & Radixsort

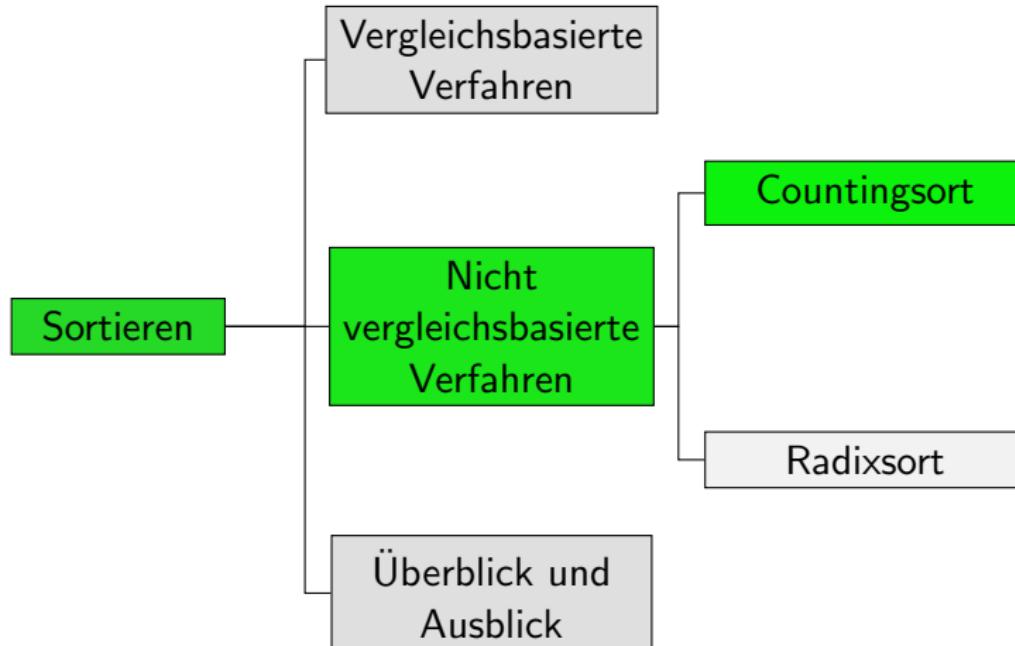
Marcel Lüthi and Gabriele Röger

Universität Basel

30. März 2022

# Nicht vergleichsbasierte Verfahren

# Sortierverfahren



## Countingsort: Idee

„Sortieren durch Zählen“

- **Annahme:** Elemente sind aus Bereich  $0, \dots, k - 1$ .
- Laufe einmal über die Eingabesequenz und zähle dabei, wie oft jedes Element vorkommt.
- Sei  $\#i$  die Anzahl der Vorkommen von Element  $i$ .
- Iteriere  $i = 0, \dots, k - 1$  und schreibe jeweils  $\#i$ -mal Element  $i$  in die Sequenz.

# Countingsort: Algorithmus

---

```
1 def sort(array, k):
2     counts = [0] * k  # list of k zeros
3     for elem in array:
4         counts[elem] += 1
5
6     pos = 0
7     for i in range(k):
8         occurrences_of_i = counts[i]
9         for j in range(occurrences_of_i):
10            array[pos + j] = i
11            pos += occurrences_of_i
```

---

# Countingsort: Algorithmus

---

```
1 def sort(array, k):
2     counts = [0] * k  # list of k zeros
3     for elem in array:
4         counts[elem] += 1
5
6     pos = 0
7     for i in range(k):
8         occurrences_of_i = counts[i]
9         for j in range(occurrences_of_i):
10            array[pos + j] = i
11            pos += occurrences_of_i
```

---

Laufzeit:  $O(n + k)$  ( $n$  Grösse der Eingabesequenz)

# Countingsort: Algorithmus

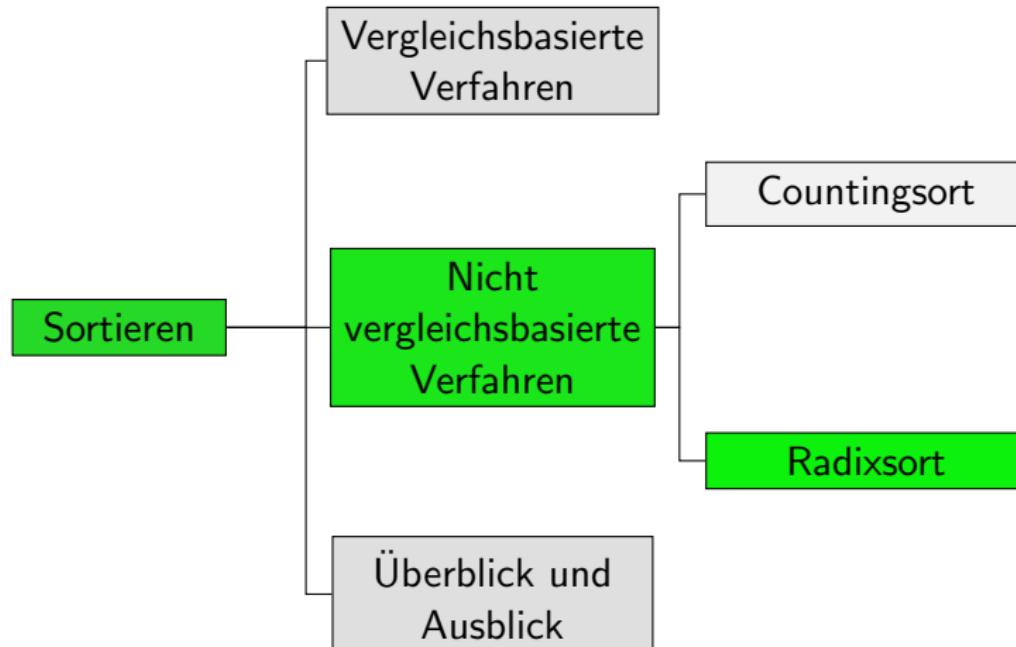
---

```
1 def sort(array, k):
2     counts = [0] * k  # list of k zeros
3     for elem in array:
4         counts[elem] += 1
5
6     pos = 0
7     for i in range(k):
8         occurrences_of_i = counts[i]
9         for j in range(occurrences_of_i):
10            array[pos + j] = i
11            pos += occurrences_of_i
```

---

Laufzeit:  $O(n + k)$  ( $n$  Grösse der Eingabesequenz)  
→ Für festes  $k$  linear

# Sortierverfahren



## Radixsort: Idee

„Sortieren durch Fachverteilen“

- Annahme: Schlüssel sind Zahlen im Dezimalsystem  
z.B. 763, 983, 96, 286, 462

## Radixsort: Idee

„Sortieren durch Fachverteilen“

- Annahme: Schlüssel sind Zahlen im Dezimalsystem  
z.B. 763, 983, 96, 286, 462
- Teile Zahlen nach **letzter** Stelle auf:

0	1	2	3	4	5	6	7	8	9
			462	763		96			
			983			286			

## Radixsort: Idee

„Sortieren durch Fachverteilen“

- Annahme: Schlüssel sind Zahlen im Dezimalsystem  
z.B. 763, 983, 96, 286, 462
- Teile Zahlen nach **letzter** Stelle auf:

0	1	2	3	4	5	6	7	8	9
		462	763			96			
			983			286			

- Sammle Zahlen von vorne nach hinten/oben nach unten auf  
462, 763, 983, 96, 286

## Radixsort: Idee

„Sortieren durch Fachverteilen“

- Annahme: Schlüssel sind Zahlen im Dezimalsystem  
z.B. 763, 983, 96, 286, 462
- Teile Zahlen nach **letzter** Stelle auf:

0	1	2	3	4	5	6	7	8	9
		462	763			96			
			983			286			

- Sammle Zahlen von vorne nach hinten/oben nach unten auf  
462, 763, 983, 96, 286
- Teile Zahlen nach **vorletzter** Stelle auf, sammle sie auf.
- Teile Zahlen nach **drittletzter** Stelle auf, sammle sie auf.
- usw. bis alle Stellen betrachtet wurden.

## Radixsort: Beispiel

- Eingabe: 263, 983, 96, 462, 286

## Radixsort: Beispiel

- Eingabe: 263, 983, 96, 462, 286
- Aufteilung nach letzter Stelle:

0	1	2	3	4	5	6	7	8	9
		462	263			96			
			983			286			

Aufsammeln ergibt: 462, 263, 983, 96, 286

## Radixsort: Beispiel

- Eingabe: 263, 983, 96, 462, 286
- Aufteilung nach letzter Stelle:

0	1	2	3	4	5	6	7	8	9
		462	263			96			
			983			286			

Aufsammeln ergibt: 462, 263, 983, 96, 286

- Aufteilung nach vorletzter Stelle:

0	1	2	3	4	5	6	7	8	9
				462			983		96
				263			286		

Aufsammeln ergibt: 462, 263, 983, 286, 96

## Radixsort: Beispiel

- Eingabe: 263, 983, 96, 462, 286
- Aufteilung nach letzter Stelle:

0	1	2	3	4	5	6	7	8	9
		462	263			96			
			983			286			

Aufsammeln ergibt: 462, 263, 983, 96, 286

- Aufteilung nach vorletzter Stelle:

0	1	2	3	4	5	6	7	8	9
				462			983		96
				263			286		

Aufsammeln ergibt: 462, 263, 983, 286, 96

- Aufteilung nach drittletzter Stelle:

0	1	2	3	4	5	6	7	8	9
096		263		462					983
			286						

Aufsammeln ergibt: 96, 263, 286, 462, 983

# Jupyter-Notebook



Jupyter-Notebook: radix\_sort.ipynb

# Radixsort: Algorithmus (für beliebige Basis)

---

```
1 def sort(array, base=10):  
2     if not array: # array is empty  
3         return  
4     iteration = 0  
5     max_val = max(array) # identify largest element  
6     while base ** iteration <= max_val:  
7         buckets = [[] for num in range(base)]  
8         for elem in array:  
9             digit = (elem // (base ** iteration)) % base  
10            buckets[digit].append(elem)  
11            pos = 0  
12            for bucket in buckets:  
13                for elem in bucket:  
14                    array[pos] = elem  
15                    pos += 1  
16            iteration += 1
```

---

## Radixsort: Laufzeit

- $m$ : Maximale Anzahl Stellen in Repräsentation mit gegebener Basis  $b$ .
- $n$ : Länge der Eingabesequenz
- Laufzeit in  $O(m \cdot (n + b))$

## Radixsort: Laufzeit

- $m$ : Maximale Anzahl Stellen in Repräsentation mit gegebener Basis  $b$ .
- $n$ : Länge der Eingabesequenz
- Laufzeit in  $O(m \cdot (n + b))$

Für festes  $m$  und  $b$  hat Radixsort lineare Laufzeit.

# Zusammenfassung

## Zusammenfassung

- Countingsort und Radixsort sind nicht vergleichsbasiert und erlauben (unter bestimmten Restriktionen) ein Sortieren in linearer Zeit.
- Sie machen jedoch zusätzliche Einschränkungen an die verwendeten Schlüssel.