Algorithms and Data Structures

A14. Sorting: Counting Sort & Radix Sort

Gabriele Röger

University of Basel

March 21, 2024

Algorithms and Data Structures

March 21, 2024 — A14. Sorting: Counting Sort & Radix Sort

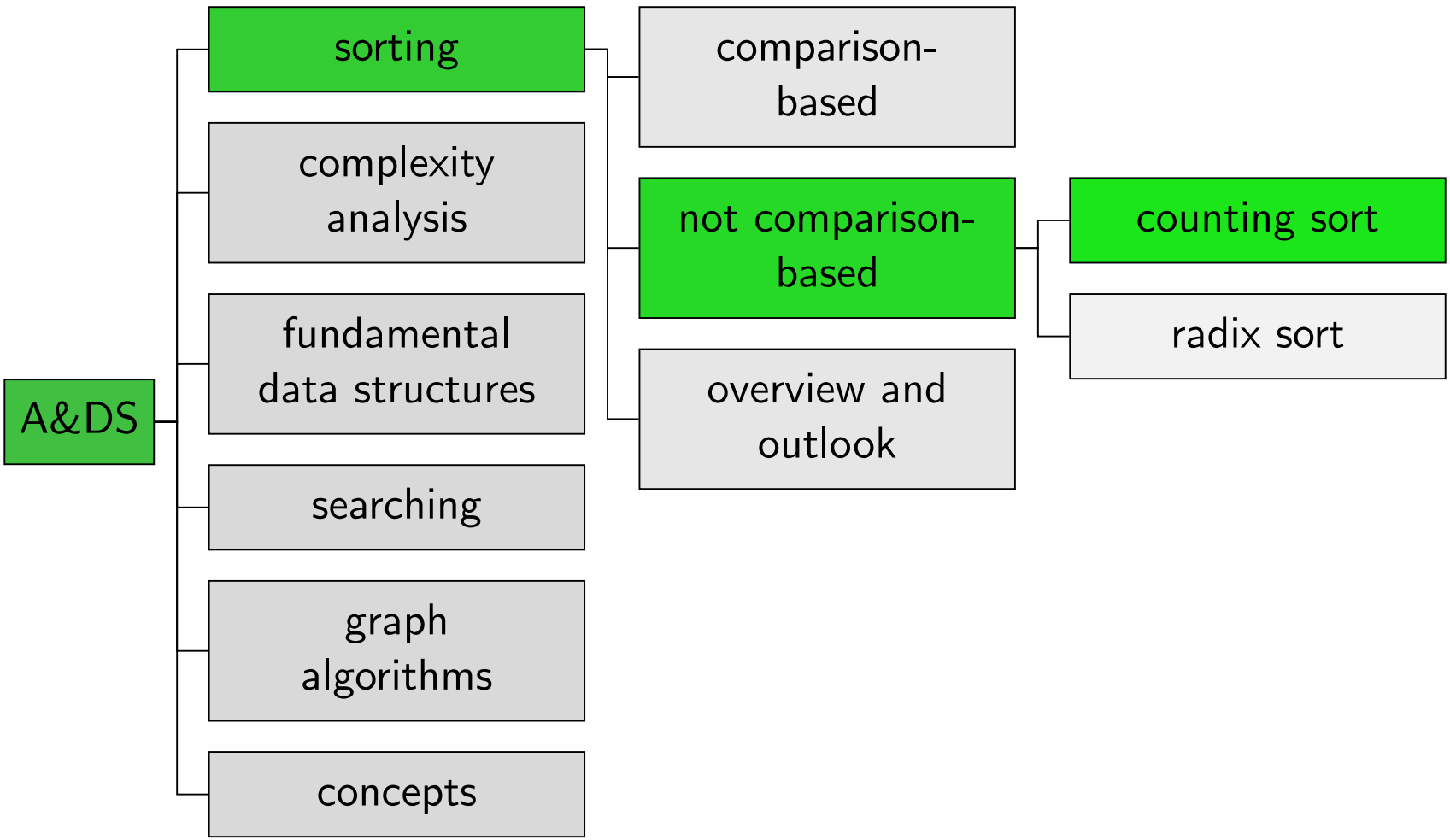
A14.1 Counting Sort

A14.2 Radix Sort

A14.3 Summary

A14.1 Counting Sort

Content of the Course



Counting Sort: Idea

„Sort by counting“

- Assumption: Elements are from the range $0, \ldots, k-1$.
- Iterate once over the input array and count the number of occurrences of each element.
- Let $\#i$ be the number of occurrences of element i.
- For $i = 0, \ldots, k-1$ write $\#i$ times element i into the sequence.

Counting Sort: Algorithm

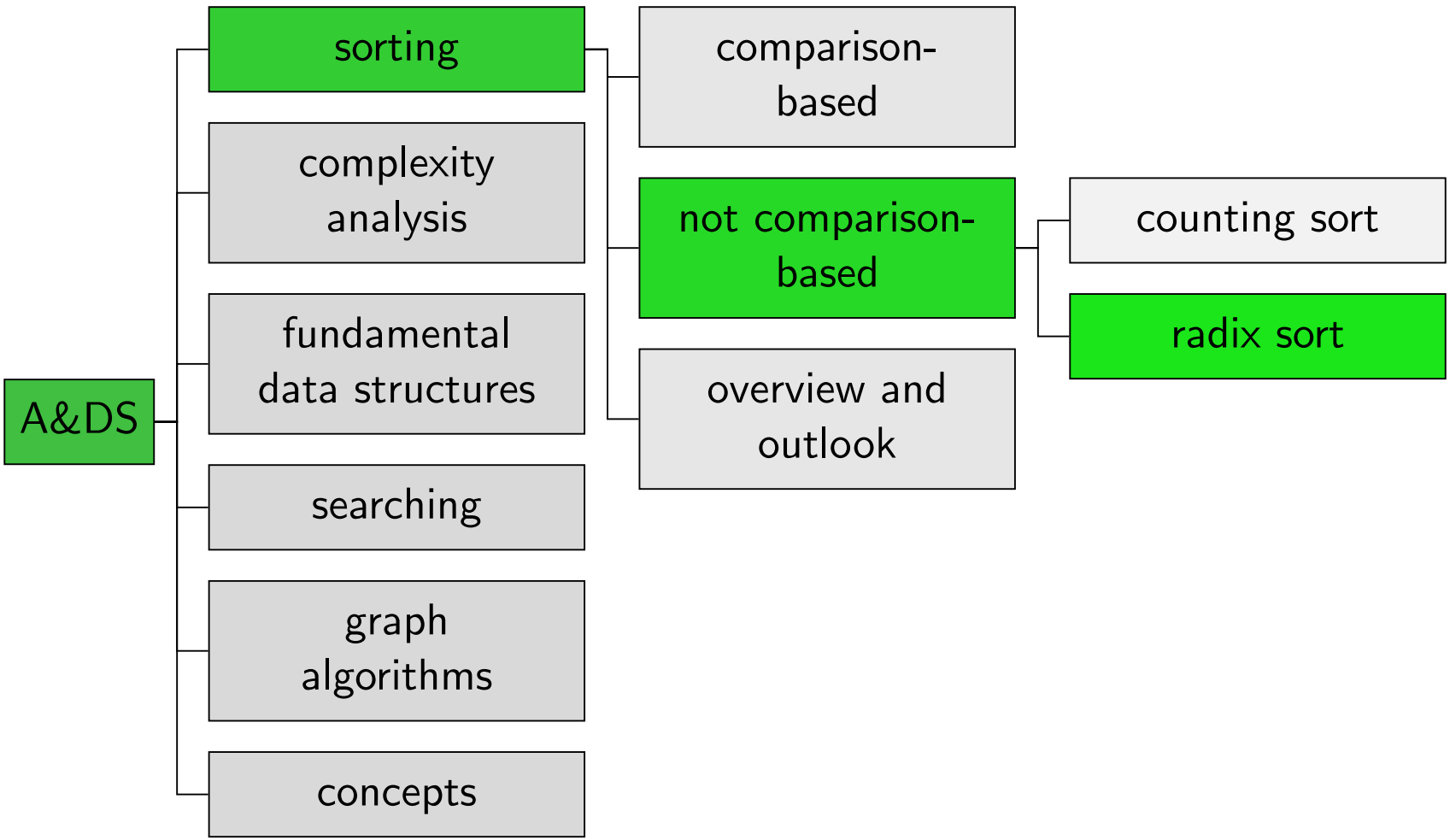
```
1  def sort(array, k):
2      counts = [0] * k  # list of k zeros
3      for elem in array:
4          counts[elem] += 1
5
6      pos = 0
7      for i in range(k):
8          occurrences_of_i = counts[i]
9          for j in range(occurrences_of_i):
10             array[pos + j] = i
11         pos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

Running time: $O(n + k)$ (n size of input sequence)

$\rightarrow$ For fixed k or $k \in O(n)$ linear.

A14.2 Radix Sort

Content of the Course



Radix Sort: Idea

- Assumption: Keys are decimal numbers
  z.B. 763, 983, 96, 286, 462
- Separate items by the least significant (= last) digit:

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

- Collect items from left to right/top to bottom:
  462, 763, 983, 96, 286
- Separate items by the second last digit and collect them.
- Separate items by the third last digit and collect them.
- ... until you considered all positions of digits.

Radix Sort: Example

- Input: 263, 983, 96, 462, 286
- Separation by last digit:

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

  After collection: 462, 263, 983, 96, 286

- Separation by second last digit:

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

  After collection: 462, 263, 983, 286, 96

- Separation by third last digit:

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

  After collection: 96, 263, 286, 462, 983

Jupyter Notebook

Jupyter notebook: `radix_sort.ipynb`

Radix Sort: Algorithm (for arbitrary base)

```
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
```

Radix Sort: Running Time

- m: Maximal number of digits in representation with given base b.
- n: length of input sequence
- Running time $O(m \cdot (n + b))$

For fixed m and b, radix sort has linear running time.

Radix Sort: High-level Perspective

All entries in the array have d digits, where the lowest-order digit is at position 0 and the highest-order digit at position d-1.

```
1  def radix_sort(array, d)
2    for i in range(d):
3      # use a stable sort to sort array on the digit at position i
```

A14.3 Summary

Summary

- Counting sort and radix sort are not comparison-based and allow us (under certain restrictions) to sort in linear time.
- However, they place additional restrictions on the keys used.