Algorithms and Data Structures

A14. Sorting: Counting Sort & Radix Sort

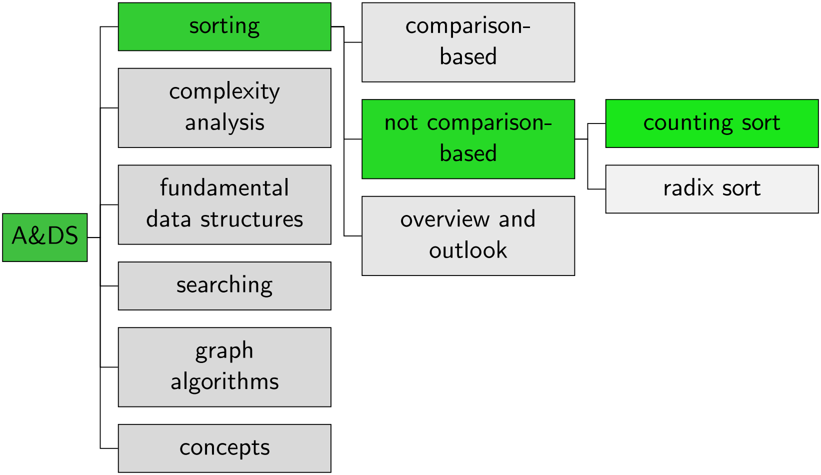
Gabriele Röger and Patrick Schnider

University of Basel

March 20, 2025

Counting Sort

Content of the Course



Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.
- From these counts we can determine the positions that the elements for each key should occupy in the sorted output.

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.
- From these counts we can determine the positions that the elements for each key should occupy in the sorted output.
  - elements with key 0 fill positions 0 to $\#0 - 1$.

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.
- From these counts we can determine the positions that the elements for each key should occupy in the sorted output.
  - elements with key 0 fill positions 0 to $\#0 - 1$.
  - elements with key 1 fill positions $\#0$ to $\#0 + \#1 - 1$.

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.
- From these counts we can determine the positions that the elements for each key should occupy in the sorted output.
  - elements with key 0 fill positions 0 to $\#0 - 1$.
  - elements with key 1 fill positions $\#0$ to $\#0 + \#1 - 1$.
  - elements with key 2 fill positions $\#0 + \#1$ to $\#0 + \#1 + \#2 - 1$.

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.
- From these counts we can determine the positions that the elements for each key should occupy in the sorted output.
  - elements with key 0 fill positions 0 to $\#0 - 1$.
  - elements with key 1 fill positions $\#0$ to $\#0 + \#1 - 1$.
  - elements with key 2 fill positions $\#0 + \#1$ to $\#0 + \#1 + \#2 - 1$.
  - ...

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \dots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.
- From these counts we can determine the positions that the elements for each key should occupy in the sorted output.
  - elements with key 0 fill positions 0 to $\#0 - 1$.
  - elements with key 1 fill positions $\#0$ to $\#0 + \#1 - 1$.
  - elements with key 2 fill positions $\#0 + \#1$ to $\#0 + \#1 + \#2 - 1$.
  - ...
  - elements with key i fill positions $\sum_{j=0}^{i-1} \#j$ to $\left(\sum_{j=0}^{i-1} \#j\right) + \#i - 1$.

Counting Sort: Idea

"Sort by counting"

- Assumption: Keys are from the range $0, \ldots, k-1$.
- Iterate once over the input array and determine the number $\#i$ of elements for each key i.
- From these counts we can determine the positions that the elements for each key should occupy in the sorted output.
  - elements with key 0 fill positions 0 to $\#0 - 1$.
  - elements with key 1 fill positions $\#0$ to $\#0 + \#1 - 1$.
  - elements with key 2 fill positions $\#0 + \#1$ to $\#0 + \#1 + \#2 - 1$.
  - ...
  - elements with key i fill positions $\sum_{j=0}^{i-1} \#j$ to $\left(\sum_{j=0}^{i-1} \#j\right) + \#i - 1$.
- (Backwards) iterate over the input array and copy the entries to the corresponding positions in the output array.

Counting Sort: Algorithm

```python
def sort(array, k):
    counts = [0] * (k + 1)  # list of k + 1 zeros
    result = [0] * len(array) # list of same size as array

    for elem in array:
        counts[elem] += 1
    # counts[j] contains number of occurrences of j

    for i in range(1, k+1): # i = 1,2, ... , k
        counts[i] += counts[i-1]
    # counts[j] now contains number of occurrences of elements <= j

    # copy elements from array to result, starting from the end
    for elem in reversed(array):
        result[counts[elem]-1] = elem
        counts[elem] -= 1

    return result
```

Jupyter Notebook

Jupyter notebook: `counting_sort.ipynb`

Counting Sort: Properties

- Counting sort is not adaptive.

Counting Sort: Properties

- Counting sort is not adaptive.
- Running time: $\Theta(n + k)$ (n size of input sequence)

Counting Sort: Properties

- Counting sort is not adaptive.
- Running time: $\Theta(n + k)$ (n size of input sequence)
  $\rightarrow$ For fixed k or $k \in O(n)$ linear.

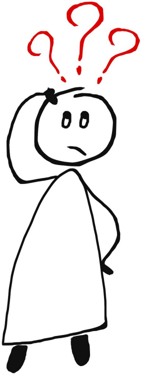
Counting Sort: Properties

- Counting sort is not adaptive.
- Running time: $\Theta(n + k)$ (n size of input sequence)
  $\rightarrow$ For fixed k or $k \in O(n)$ linear.
- Memory: $\Theta(n + k)$ (not in-place)

Counting Sort: Properties

- Counting sort is not adaptive.
- Running time: $\Theta(n + k)$ (n size of input sequence)
  $\rightarrow$ For fixed k or $k \in O(n)$ linear.
- Memory: $\Theta(n + k)$ (not in-place)
- Counting sort is stable. Why?

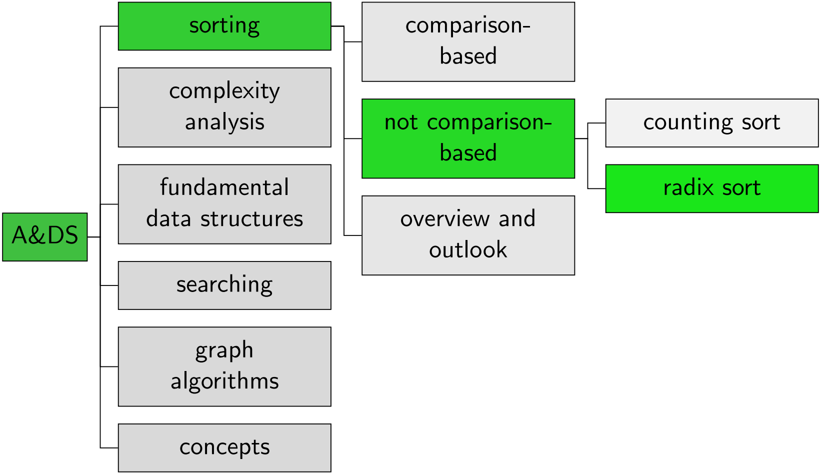
Questions



Questions?

Radix Sort

Content of the Course



Radix Sort: Idea

- Assumption: Keys are decimal numbers
  z.B. 763, 983, 96, 286, 462

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			

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

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

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

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

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)

```python
def sort(array, base=10):
    if not array:  # array is empty
        return
    iteration = 0
    max_val = max(array) # identify largest element
    while base ** iteration <= max_val:
        buckets = [[] for num in range(base)]
        for elem in array:
            digit = (elem // (base ** iteration)) % base
            buckets[digit].append(elem)
        pos = 0
        for bucket in buckets:
            for elem in bucket:
                array[pos] = elem
                pos += 1
        iteration += 1
```

Radix Sort: Running Time

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

Radix Sort: Running Time

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

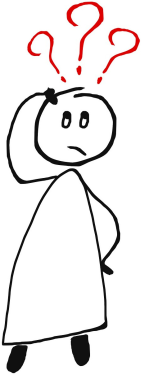
For fixed m and b, radix sort has linear running time.

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.

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

Questions



Questions?

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.