Karlsruher Institut für Technologie

Übung 9 – Algorithmen II

Florian Kurpicz *kurpicz@kit.edu*

`https://algo2.iti.kit.edu/AlgorithmenII_WS23.php`

Institut für Theoretische Informatik - Algorithm Engineering

```
        result = current_weight;
        return true;
    }

for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )
        Weight new_weight = edge.weight + current_weight;
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected.
        if( !priority_queue.isReached( edge.target ) ){
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES );
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ); )
            priority_queue.push( edge.target, new_weight );
        } else {
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){
                COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES
                priority_queue.decreaseKey( edge.target, new_weight );
            }
```

Themenübersicht

- Stringology
  - Multikey Quicksort
  - Suche mit Suffix-Arrays

Multikey Quicksort

Bentley, Sedgewick (1997)

Three-way Radix Quicksort

- sortiert Elemente mit mehreren Schlüsseln wie msd-*Radixsort*
  → z.B. Stellen einer Zahl, Zeichen eines Strings

- für einen Schlüssel wird *Quicksort* mit drei Fällen ausgeführt
  → kleiner als, gleich, größer als das Pivotelement

Multikey Quicksort

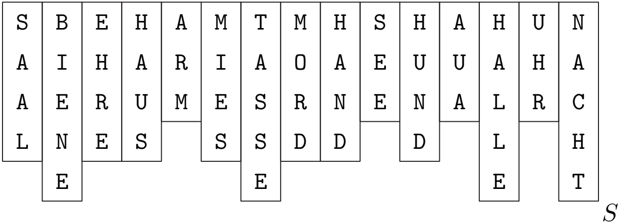
Ablauf

```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                    Basisfall
    choose $p \in S$ uniformly at random               Pivotelement
    return concatenation of                            Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```

Multikey Quicksort

Ablauf

```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                    Basisfall
    choose $p \in S$ uniformly at random                               Pivotelement
    return concatenation of                                            Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



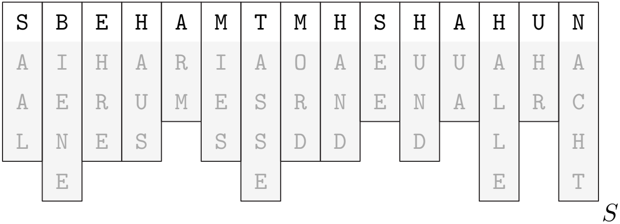
SAAL, BIENE, EHRE, HAUS, ARM, MIES, TASSE, MORD, HAND, SEE, HUND, AUA, HALLE, UHR, NACHT

S

Multikey Quicksort

Ablauf

```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D		D		L		H
	E					E						E		T

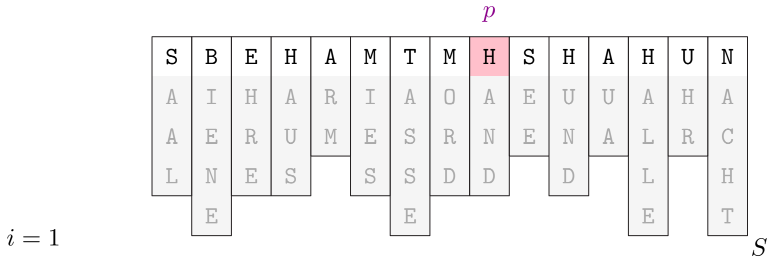
$i = 1$

S

Multikey Quicksort

Ablauf

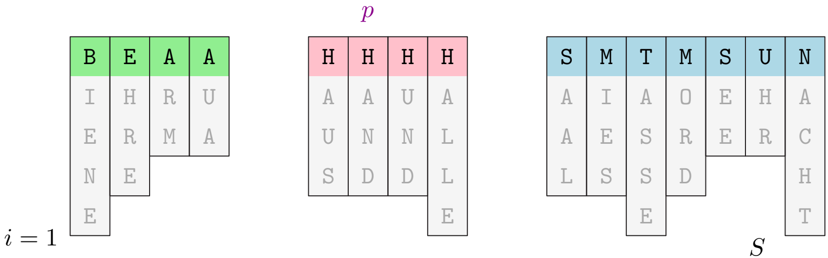
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                              Basisfall
    choose $p \in S$ uniformly at random                         Pivotelement
    return concatenation of                                      Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

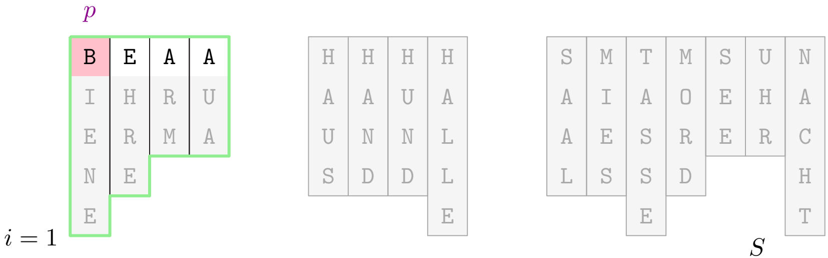
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                    Basisfall
    choose $p \in S$ uniformly at random                               Pivotelement
    return concatenation of                                            Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i+1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

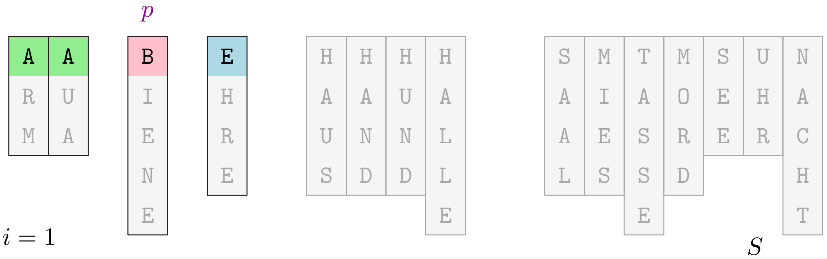
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                          Basisfall
    choose $p \in S$ uniformly at random                     Pivotelement
    return concatenation of                                  Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

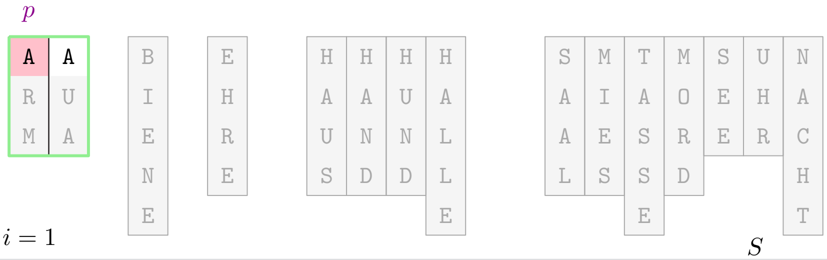
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

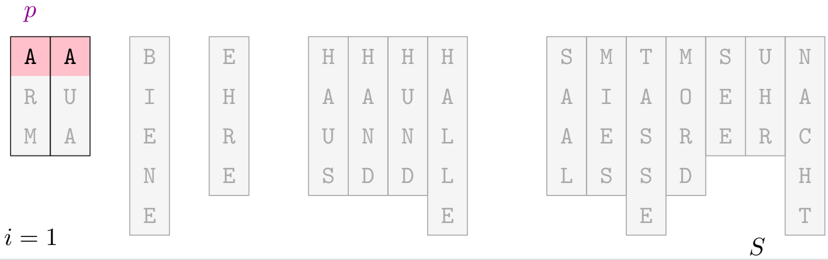
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                Basisfall
    choose $p \in S$ uniformly at random                           Pivotelement
    return concatenation of                                        Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

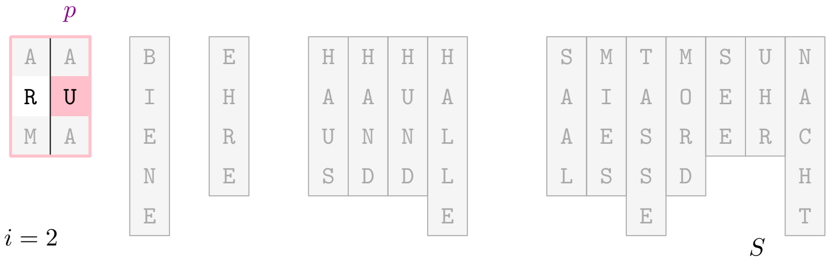
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                          Basisfall
    choose $p \in S$ uniformly at random                    Pivotelement
    return concatenation of                                  Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

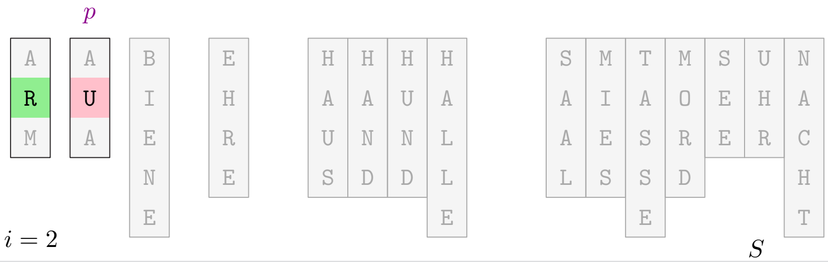
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                          Basisfall
    choose $p \in S$ uniformly at random                   Pivotelement
    return concatenation of                                 Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

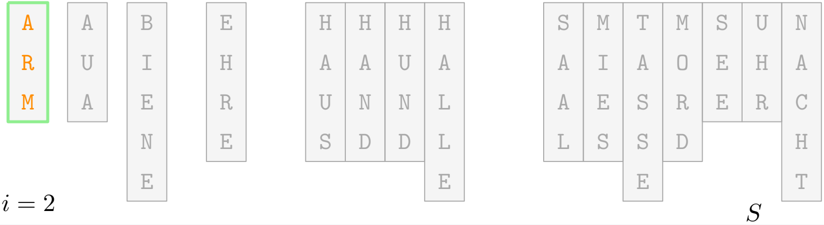
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                Basisfall
    choose $p \in S$ uniformly at random                           Pivotelement
    return concatenation of                                        Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablaut

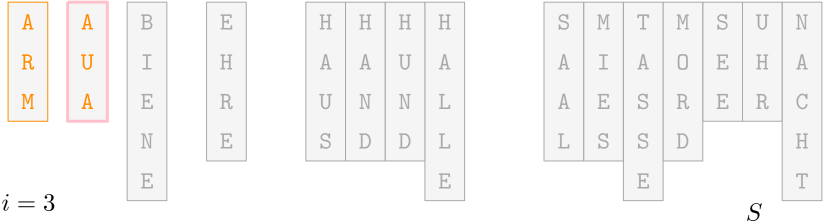
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                          Basisfall
    choose $p \in S$ uniformly at random                     Pivotelement
    return concatenation of                                  Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

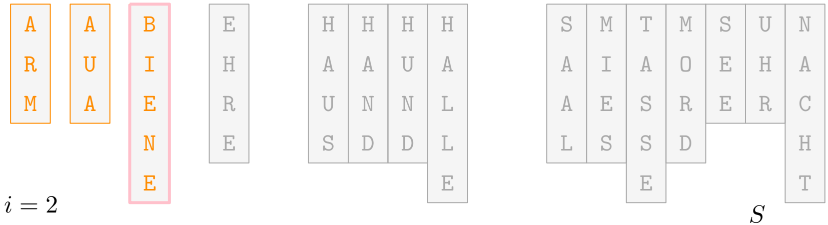
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

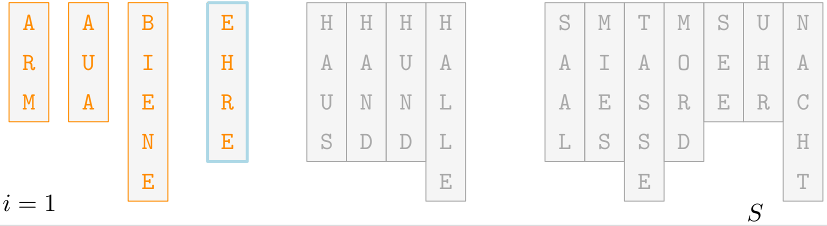
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                        Basisfall
    choose $p \in S$ uniformly at random                                   Pivotelement
    return concatenation of                                                Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

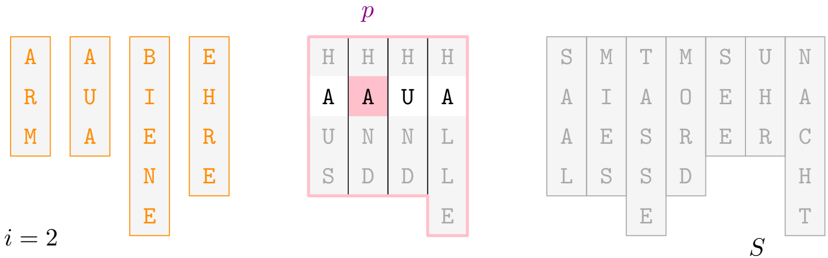
```
**Function** mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    **if** $|S| \leq 1$ **then return** $S$                                  Basisfall
    choose $p \in S$ uniformly at random                               Pivotelement
    **return** concatenation of                                        Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

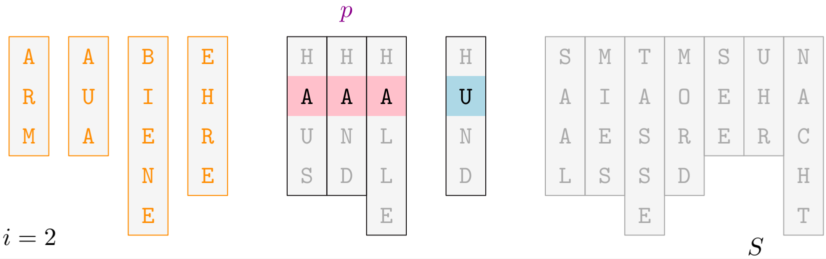
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

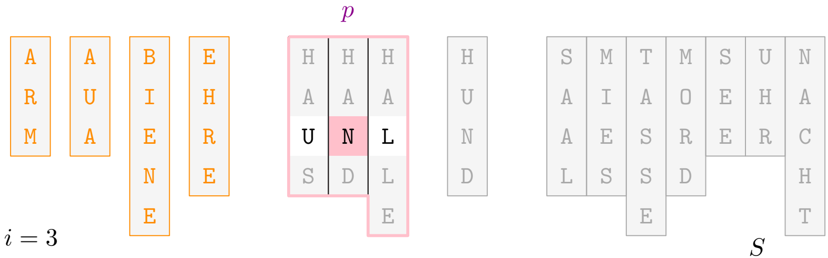
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablaut

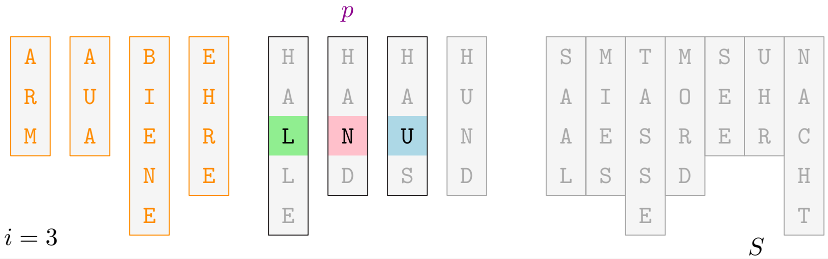
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
        mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
        mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
        mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

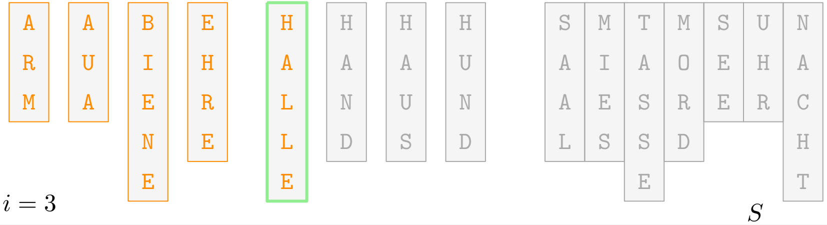
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

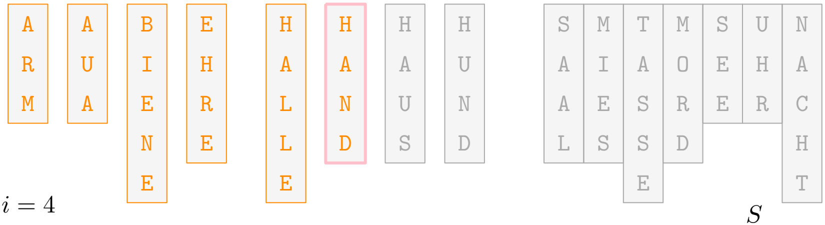
```
**Function** mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

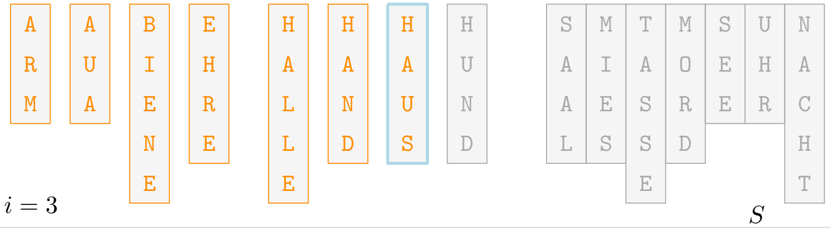
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

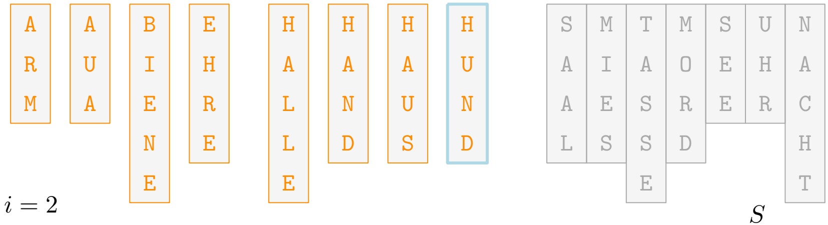
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

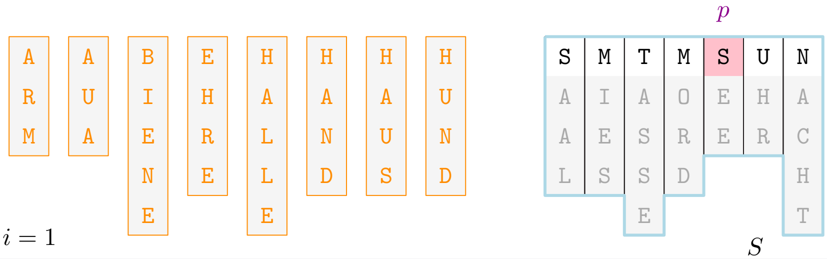
```
**Function** mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    **if** $|S| \leq 1$ **then return** $S$                                   Basisfall
    choose $p \in S$ uniformly at random                                    Pivotelement
    **return** concatenation of                                             Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

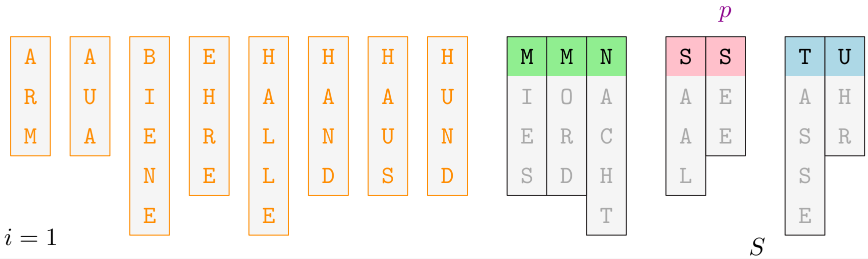
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

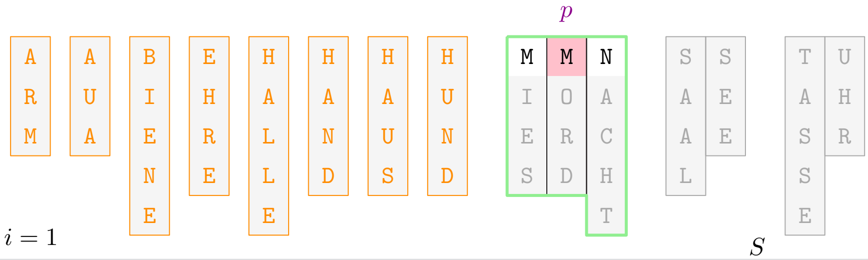
```
**Function** mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    **if** $|S| \leq 1$ **then return** $S$                              Basisfall
    choose $p \in S$ uniformly at random                        Pivotelement
    **return** concatenation of                                 Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

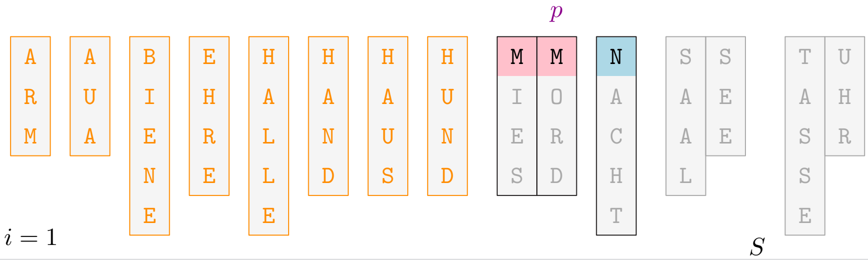
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                              Basisfall
    choose $p \in S$ uniformly at random                         Pivotelement
    return concatenation of                                      Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

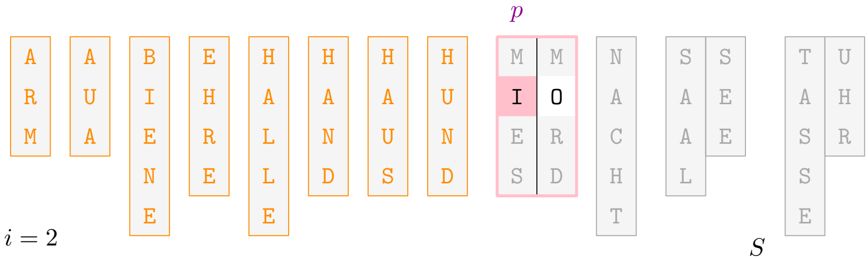
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

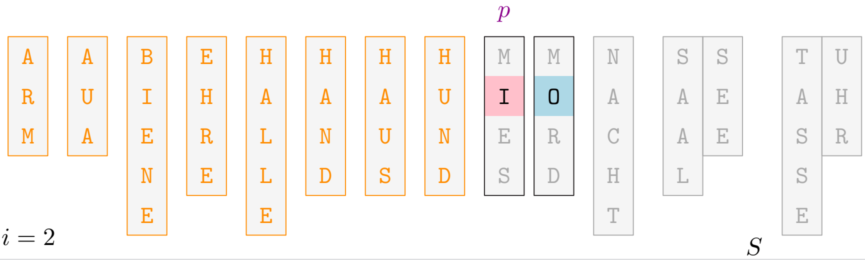
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                    Basisfall
    choose $p \in S$ uniformly at random                               Pivotelement
    return concatenation of                                            Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablaut

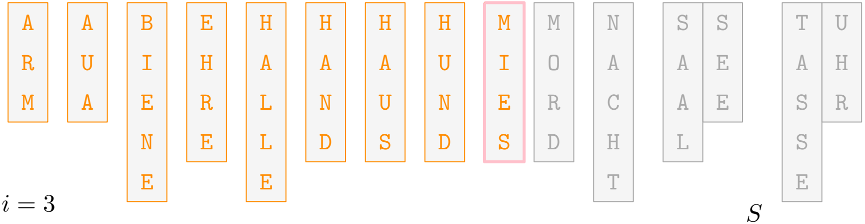
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

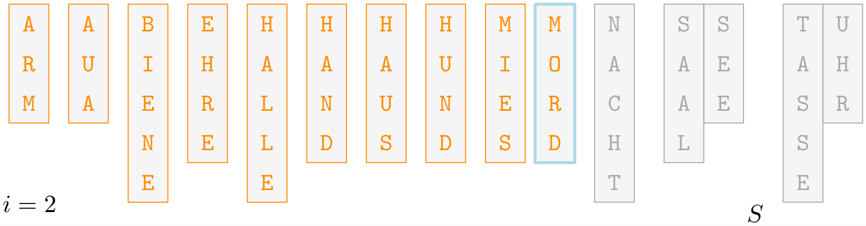
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
        mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
        mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
        mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

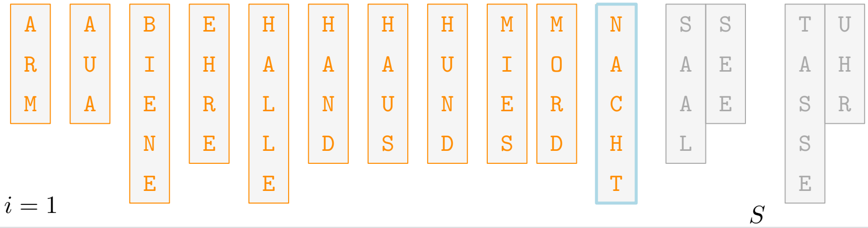
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
        mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
        mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
        mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

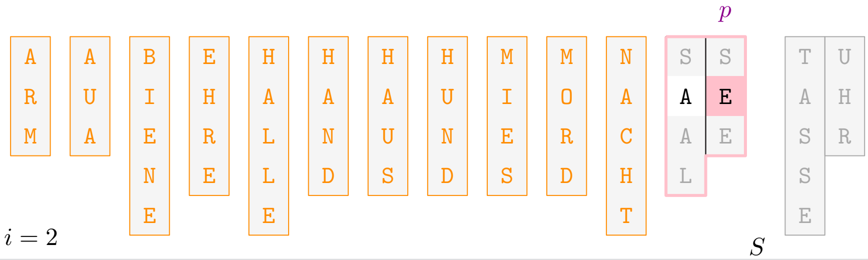
```
**Function** mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    **if** $|S| \leq 1$ **then return** $S$                              Basisfall
    choose $p \in S$ uniformly at random                                Pivotelement
    **return** concatenation of                                         Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

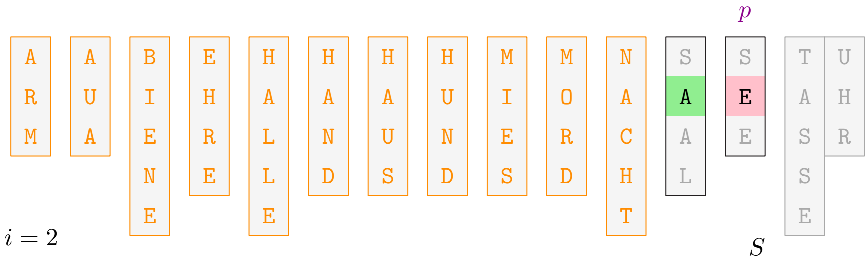
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                    Basisfall
    choose $p \in S$ uniformly at random                               Pivotelement
    return concatenation of                                            Rekursion
        mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
        mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
        mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

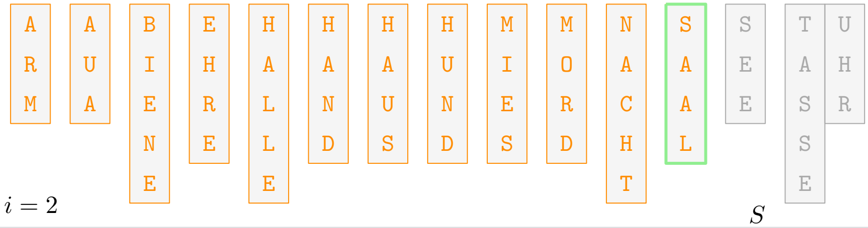
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                               Basisfall
    choose $p \in S$ uniformly at random                          Pivotelement
    return concatenation of                                       Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

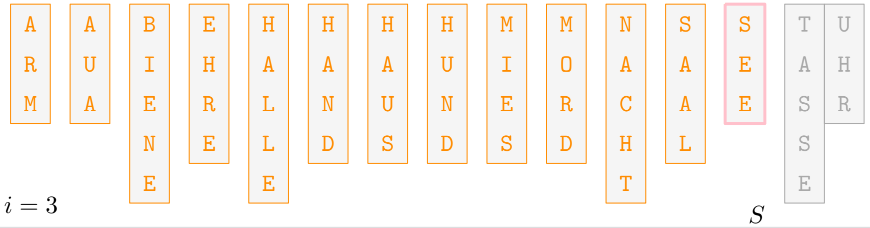
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                          Basisfall
    choose $p \in S$ uniformly at random                     Pivotelement
    return concatenation of                                  Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablaut

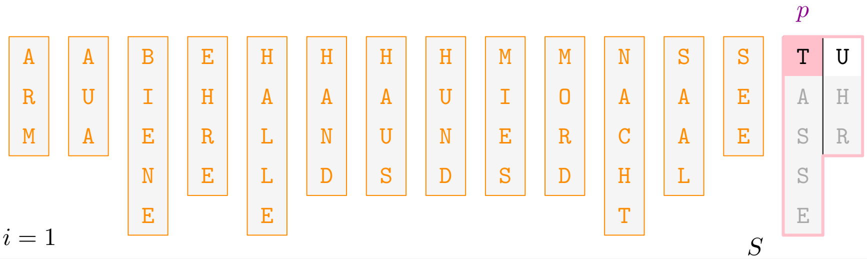
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

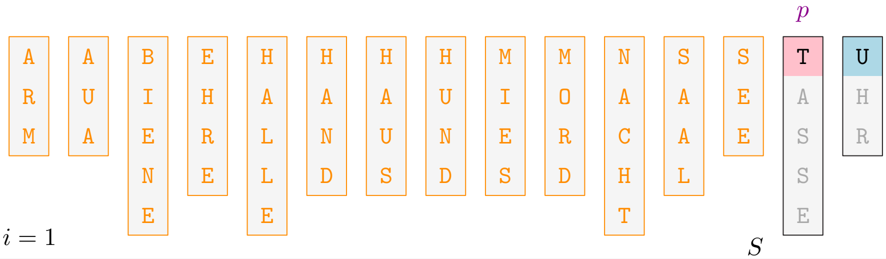
```
**Function** mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    **if** $|S| \leq 1$ **then return** $S$                                   Basisfall
    choose $p \in S$ uniformly at random                                Pivotelement
    **return** concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

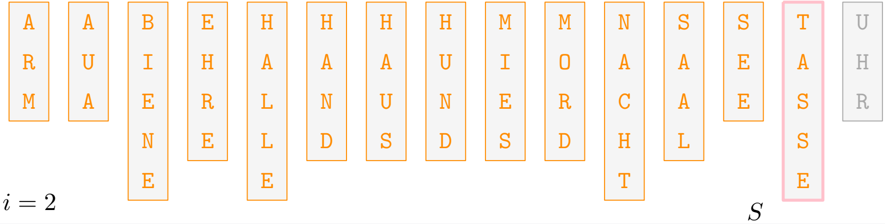
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                   Basisfall
    choose $p \in S$ uniformly at random                              Pivotelement
    return concatenation of                                           Rekursion
        mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
        mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
        mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

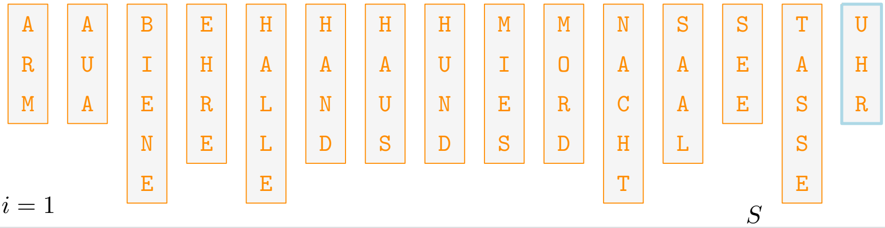
```
**Function** mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    **if** $|S| \leq 1$ **then return** $S$                         Basisfall
    choose $p \in S$ uniformly at random                         Pivotelement
    **return** concatenation of                                  Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

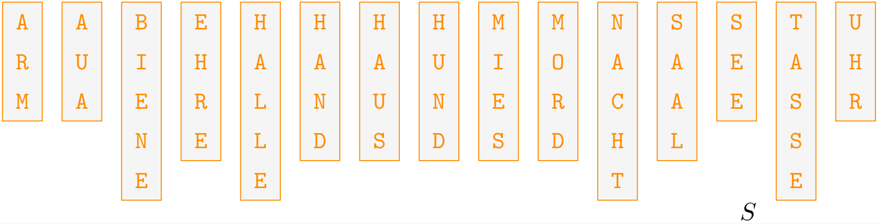
```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```



Multikey Quicksort

Ablauf

```
Function mkqSort( $S$: Array of String, $i$ : Integer) : Array of String
    if $|S| \leq 1$ then return $S$                                  Basisfall
    choose $p \in S$ uniformly at random                             Pivotelement
    return concatenation of                                          Rekursion
            mkqSort( $\langle e \in S : e[i] < p[i] \rangle, i$ ),
            mkqSort( $\langle e \in S : e[i] = p[i] \rangle, i + 1$ ),
            mkqSort( $\langle e \in S : e[i] > p[i] \rangle, i$ )
```

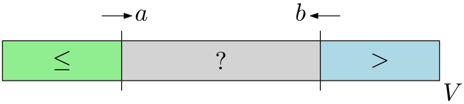


in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement *p*
- zwei Zeiger *a*, *b* wandern von außen “in die Mitte”
  → Invariante: $V[i < a] \le p$, $V[i > b] > p$

  - Wähle Pivot *p* und tausche mit erstem Element, setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \le p$, $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \le p$
  - Ende, wenn $a > b$

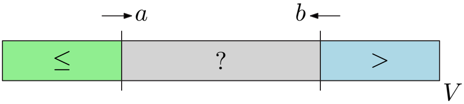


in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$

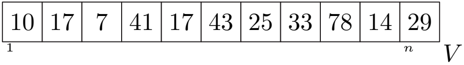
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

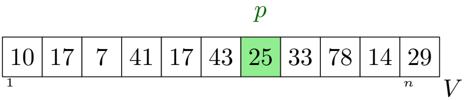


10	17	7	41	17	43	25	33	78	14	29

in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

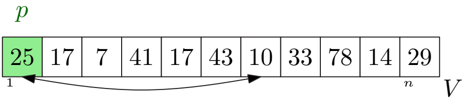


10	17	7	41	17	43	25	33	78	14	29

in-place Multikey Quicksort

in-place **bei Quicksort** für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

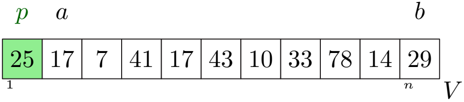


in-place Multikey Quicksort

in-place **bei Quicksort** für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$

  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$,
    $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

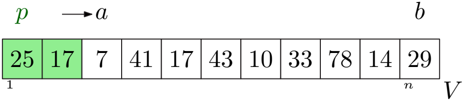


25	17	7	41	17	43	10	33	78	14	29

in-place Multikey Quicksort

in-place **bei Quicksort** für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

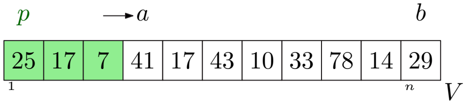


25	17	7	41	17	43	10	33	78	14	29

in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2$, $b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$,
    $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

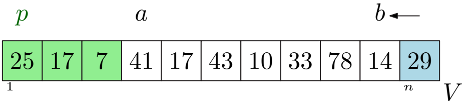


in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$

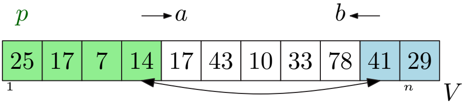
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



in-place Multikey Quicksort

in-place bei Quicksort für Integer

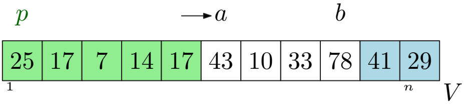
- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

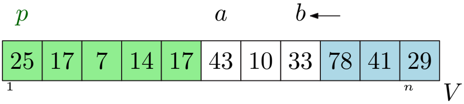


25	17	7	14	17	43	10	33	78	41	29

in-place Multikey Quicksort

in-place **bei Quicksort** für Integer

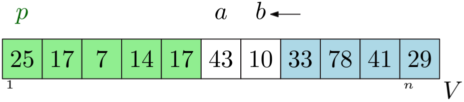
- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$,
    $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$,
    $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



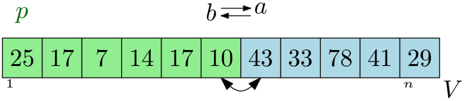
25	17	7	14	17	43	10	33	78	41	29

in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$

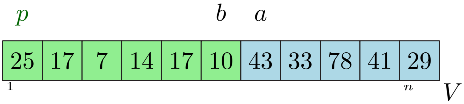
  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



in-place Multikey Quicksort

in-place bei Quicksort für Integer

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  → Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2$, $b = n$
  - $a \to a + 1$, solange $V[a] \leq p$,
    $b \to b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



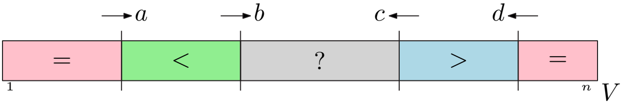
in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement *p*
- zwei Zeiger *b*, *c* wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger *a*, *d* “außen” gesammelt

→ Invariante: $V[i \in [a, b)_{a \neq b}] \leq p$, $V[i < a \lor i > d] = p$, $V[i \in (c, d]_{c \neq d}] > p$

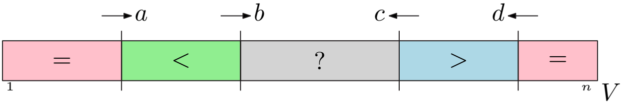


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen "in die Mitte"
- gleiche Elemente werden mit Zeiger a, d "außen" gesammelt
  → Invariante: $V[i \in [a, b)_{a \neq b}] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d]_{c \neq d}] > p$

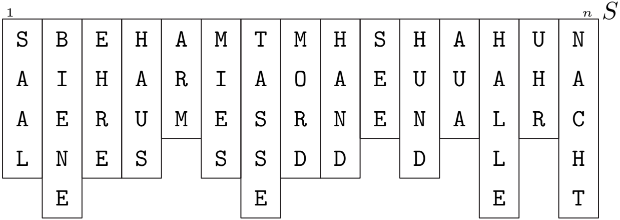


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen "in die Mitte"
- gleiche Elemente werden mit Zeiger a, d "außen" gesammelt
  → Invariante: $V[i \in [a, b)_{a \neq b}] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d]_{c \neq d}] > p$



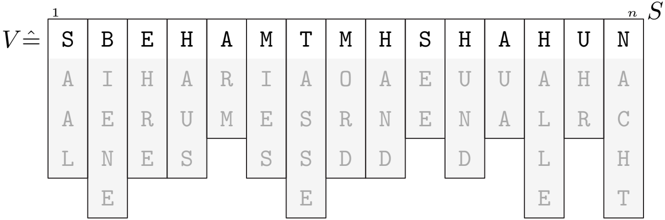
SAAL	BIENE	EHRE	HAUS	ARM	MIES	TASSE	MORD	HAND	SEE	HUND	AUA	HALLE	UHR	NACHT

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement *p*
- zwei Zeiger *b*, *c* wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger *a*, *d* “außen” gesammelt
  → Invariante: $V[i \in [a,b)_{a\neq b}] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c,d]_{c\neq d}] > p$



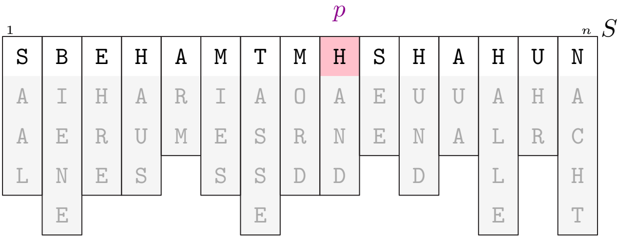
S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D		D		L		H
	E					E						E		T

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

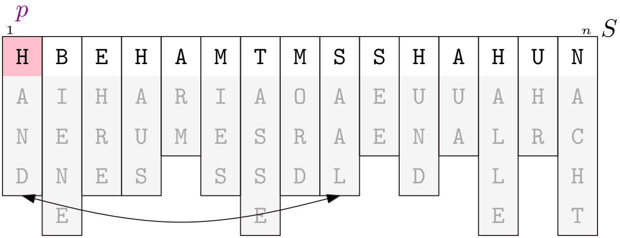


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

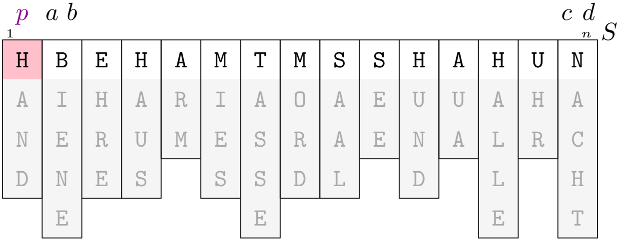


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



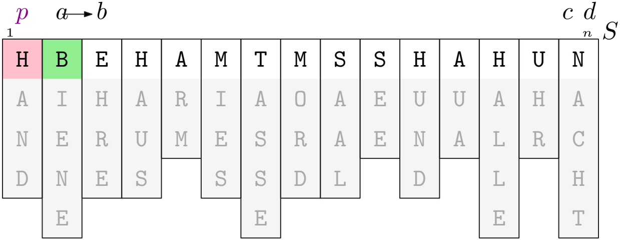
H	B	E	H	A	M	T	M	S	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
N	E	R	U	M	E	S	R	A	E	N	A	L	R	C
D	N	E	S		S	S	D	L		D		L		H
	E					E						E		T

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



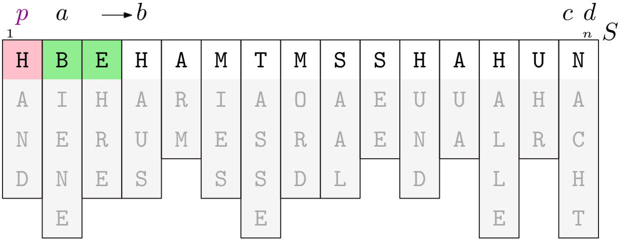
H	B	E	H	A	M	T	M	S	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
N	E	R	U	M	E	S	R	A	E	N	A	L	R	C
D	N	E	S		S	S	D	L		D		L		H
	E					E						E		T

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

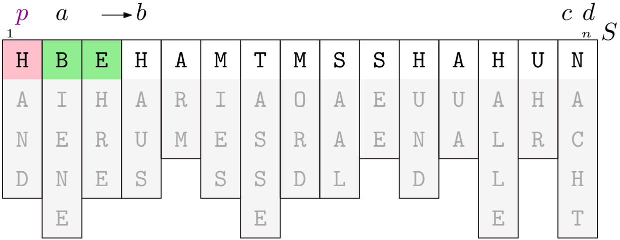


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2$, $c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

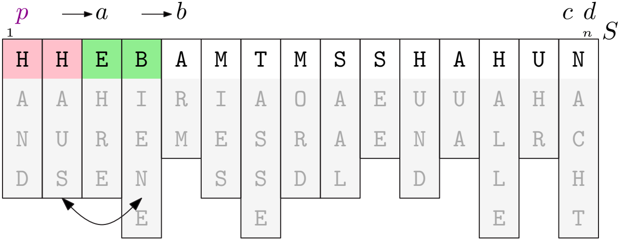


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2$, $c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

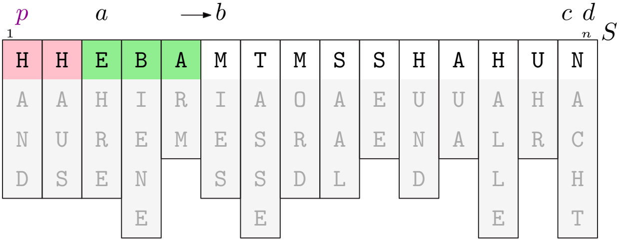


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element,
  setze $a = b = 2$, $c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$,
  $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

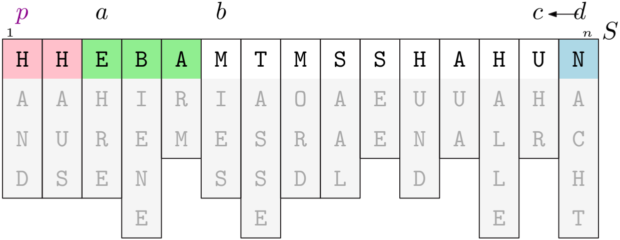


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

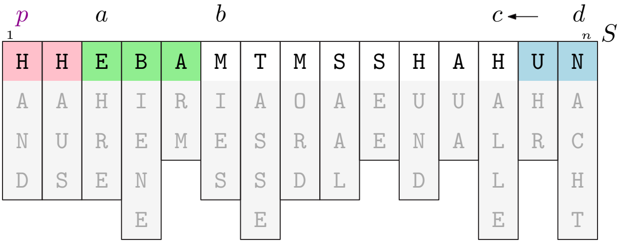


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

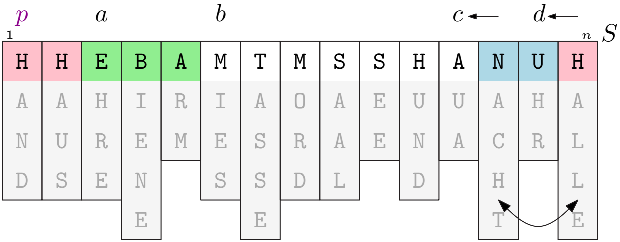


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2$, $c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

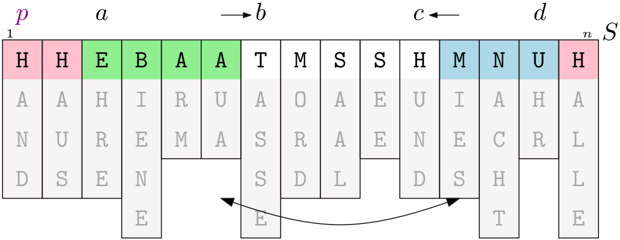


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element,
  setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$,
  $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



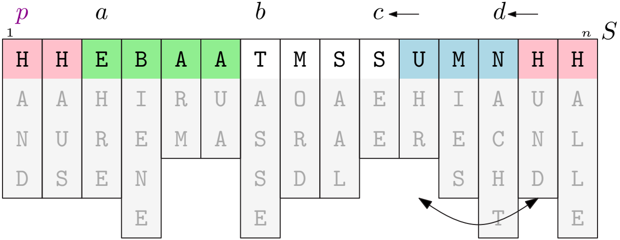
H	H	E	B	A	A	T	M	S	S	H	M	N	U	H
A	A	H	I	R	U	A	O	A	E	U	I	A	H	A
N	U	R	E	M	A	S	R	A	E	N	E	C	R	L
D	S	E	N			S	D	L		D	S	H		L
			E			E						T		E

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2$, $c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

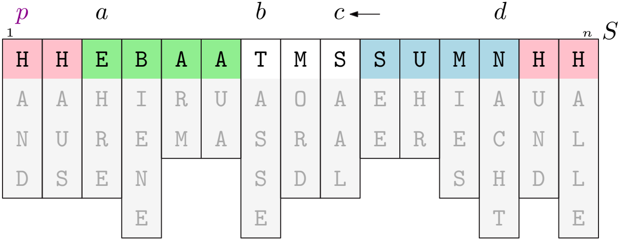


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2$, $c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



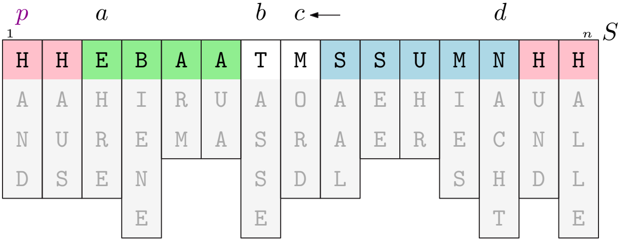
H	H	E	B	A	A	T	M	S	S	U	M	N	H	H
A	A	H	I	R	U	A	O	A	E	H	I	A	U	A
N	U	R	E	M	A	S	R	A	E	R	E	C	N	L
D	S	E	N			S	D	L			S	H	D	L
			E			E						T		E

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



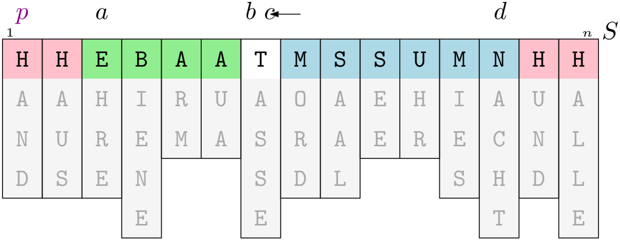
H	H	E	B	A	A	T	M	S	S	U	M	N	H	H
A	A	H	I	R	U	A	O	A	E	H	I	A	U	A
N	U	R	E	M	A	S	R	A	E	R	E	C	N	L
D	S	E	N			S	D	L			S	H	D	L
			E			E						T		E

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



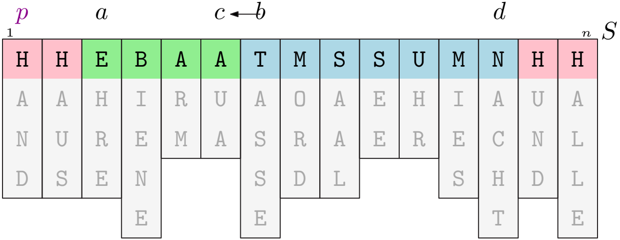
H	H	E	B	A	A	T	M	S	S	U	M	N	H	H
A	A	H	I	R	U	A	O	A	E	H	I	A	U	A
N	U	R	E	M	A	S	R	A	E	R	E	C	N	L
D	S	E	N			S	D	L			S	H	D	L
			E			E						T		E

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a + 1$, $c \to c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



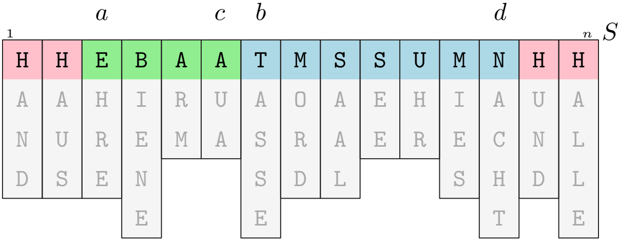
H	H	E	B	A	A	T	M	S	S	U	M	N	H	H
A	A	H	I	R	U	A	O	A	E	H	I	A	U	A
N	U	R	E	M	A	S	R	A	E	R	E	C	N	L
D	S	E	N			S	D	L			S	H	D	L
			E			E						T		E

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Umgruppierung

- $r = \min(a - 1, b - a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
  Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$

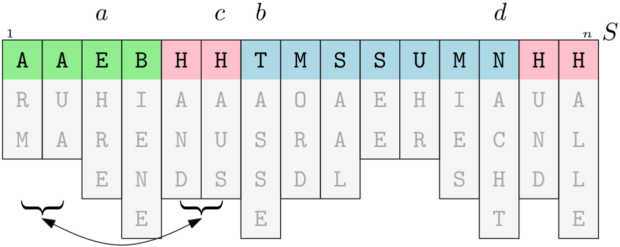


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Umgruppierung

- $r = \min(a - 1, b - a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
  Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$

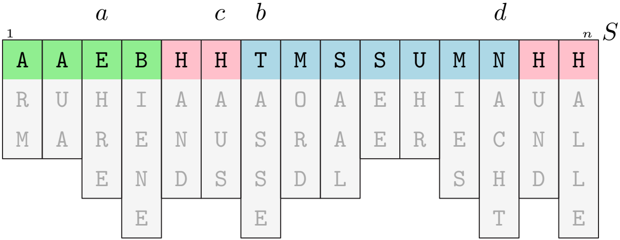


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Umgruppierung

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$

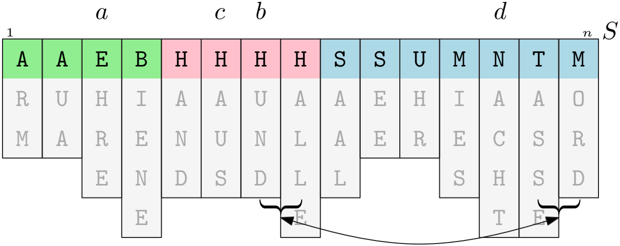


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Umgruppierung

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$

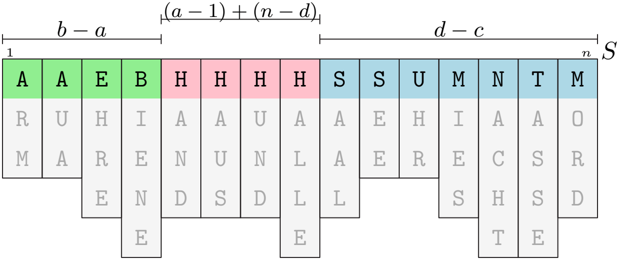


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Umgruppierung

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$



A	A	E	B	H	H	H	H	S	S	U	M	N	T	M
R	U	H	I	A	A	U	A	A	E	H	I	A	A	O
M	A	R	E	N	U	N	L	A	E	R	E	C	S	R
		E	N	D	S	D	L	L			S	H	S	D
			E				E					T	E	

in-place Multikey Quicksort

Zusammenfassung

- *Three-way Radix Quicksort*

  Partitionierung in kleiner, gleich, größer über alle Stellen analog zu msd-*Radixsort*

- effizient $\mathcal{O}(|S| \log |S| + d)$

  d ≙ Summe der Länge der unterscheidenden Präfixe

- *in-place* Partitionierung möglich

  durch geschicktes Speichern und Verschieben der gleichen Elemente

- sehr einfache Implementierung

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
4	b	a	r	h	a	b	a	r	b	e	r	$			
5	a	r	h	a	b	a	r	b	e	r	$				
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
8	a	b	a	r	b	e	r	$							
9	b	a	r	b	e	r	$								
10	a	r	b	e	r	$									
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													
15	$														

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
4	b	a	r	h	a	b	a	r	b	e	r	$			
5	a	r	h	a	b	a	r	b	e	r	$				
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
8	a	b	a	r	b	e	r	$							
9	b	a	r	b	e	r	$								
10	a	r	b	e	r	$									
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													
15	$														

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	
15	$
1	b a r b a r h a b a r b e r $
2	a r b a r h a b a r b e r $
3	r b a r h a b a r b e r $
4	b a r h a b a r b e r $
5	a r h a b a r b e r $
6	r h a b a r b e r $
7	h a b a r b e r $
8	a b a r b e r $
9	b a r b e r $
10	a r b e r $
11	r b e r $
12	b e r $
13	e r $
14	r $

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T* indiziert alle Suffixe in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
15	$														
8	a	b	a	r	b	e	r	$							
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
4	b	a	r	h	a	b	a	r	b	e	r	$			
5	a	r	h	a	b	a	r	b	e	r	$				
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
9	b	a	r	b	e	r	$								
10	a	r	b	e	r	$									
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von T
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
15	$														
8	a	b	a	r	b	e	r	$							
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
4	b	a	r	h	a	b	a	r	b	e	r	$			
5	a	r	h	a	b	a	r	b	e	r	$				
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
9	b	a	r	b	e	r	$								
10	a	r	b	e	r	$									
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
i															
15	$														
8	a	b	a	r	b	e	r	$							
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
10	a	r	b	e	r	$									
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
4	b	a	r	h	a	b	a	r	b	e	r	$			
5	a	r	h	a	b	a	r	b	e	r	$				
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
9	b	a	r	b	e	r	$								
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$

i															
15	\$														
8	a	b	a	r	b	e	r	\$							
2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$	
10	a	r	b	e	r	\$									
5	a	r	h	a	b	a	r	b	e	r	\$				
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
3	r	b	a	r	h	a	b	a	r	b	e	r	\$		
4	b	a	r	h	a	b	a	r	b	e	r	\$			
6	r	h	a	b	a	r	b	e	r	\$					
7	h	a	b	a	r	b	e	r	\$						
9	b	a	r	b	e	r	\$								
11	r	b	e	r	\$										
12	b	e	r	\$											
13	e	r	\$												
14	r	\$													

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T* indiziert alle Suffixe in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	
15	$
8	a b a r b e r $
2	a r b a r h a b a r b e r $
10	a r b e r $
5	a r h a b a r b e r $
1	b a r b a r h a b a r b e r $
3	r b a r h a b a r b e r $
4	b a r h a b a r b e r $
6	r h a b a r b e r $
7	h a b a r b e r $
9	b a r b e r $
11	r b e r $
12	b e r $
13	e r $
14	r $

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T* indiziert alle Suffixe in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
15	$														
8	a	b	a	r	b	e	r	$							
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
10	a	r	b	e	r	$									
5	a	r	h	a	b	a	r	b	e	r	$				
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
9	b	a	r	b	e	r	$								
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
4	b	a	r	h	a	b	a	r	b	e	r	$			
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
15	$														
8	a	b	a	r	b	e	r	$							
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
10	a	r	b	e	r	$									
5	a	r	h	a	b	a	r	b	e	r	$				
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
9	b	a	r	b	e	r	$								
4	b	a	r	h	a	b	a	r	b	e	r	$			
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suffix-Array *SA* von *T* indiziert alle Suffixe in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

i	
15	$
8	abarber$
2	arbarhabarber$
10	arber$
5	arhabarber$
1	barbarhabarber$
9	barber$
4	barhabarber$
12	ber$
3	rbarhabarber$
6	rhabarber$
7	habarber$
11	rber$
13	er$
14	r$

Suffix-Arrays

Wiederholung

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suffix-Array *SA* von *T* indiziert alle Suffixe in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

i	
15	$
8	abarber$
2	arbarhabarber$
10	arber$
5	arhabarber$
1	barbarhabarber$
9	barber$
4	barhabarber$
12	ber$
13	er$
3	rbarhabarber$
6	rhabarber$
7	habarber$
11	rber$
14	r$

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
15	$														
8	a	b	a	r	b	e	r	$							
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
10	a	r	b	e	r	$									
5	a	r	h	a	b	a	r	b	e	r	$				
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
9	b	a	r	b	e	r	$								
4	b	a	r	h	a	b	a	r	b	e	r	$			
12	b	e	r	$											
13	e	r	$												
7	h	a	b	a	r	b	e	r	$						
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
6	r	h	a	b	a	r	b	e	r	$					
11	r	b	e	r	$										
14	r	$													

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge
Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	
15	$
8	a b a r b e r $
2	a r b a r h a b a r b e r $
10	a r b e r $
5	a r h a b a r b e r $
1	b a r b a r h a b a r b e r $
9	b a r b e r $
4	b a r h a b a r b e r $
12	b e r $
13	e r $
7	h a b a r b e r $
14	r $
3	r b a r h a b a r b e r $
6	r h a b a r b e r $
11	r b e r $

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T* indiziert alle Suffixe in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	
15	`$`
8	`a b a r b e r $`
2	`a r b a r h a b a r b e r $`
10	`a r b e r $`
5	`a r h a b a r b e r $`
1	`b a r b a r h a b a r b e r $`
9	`b a r b e r $`
4	`b a r h a b a r b e r $`
12	`b e r $`
13	`e r $`
7	`h a b a r b e r $`
14	`r $`
3	`r b a r h a b a r b e r $`
6	`r h a b a r b e r $`
11	`r b e r $`

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von T
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	
15	$
8	a b a r b e r $
2	a r b a r h a b a r b e r $
10	a r b e r $
5	a r h a b a r b e r $
1	b a r b a r h a b a r b e r $
9	b a r b e r $
4	b a r h a b a r b e r $
12	b e r $
13	e r $
7	h a b a r b e r $
14	r $
3	r b a r h a b a r b e r $
11	r b e r $
6	r h a b a r b e r $

Suffix-Arrays

Wiederholung

Suffix-Array *SA* von *T*
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]	
1	15	$
2	8	a b a r b e r $
3	2	a r b a r h a b a r b e r $
4	10	a r b e r $
5	5	a r h a b a r b e r $
6	1	b a r b a r h a b a r b e r $
7	9	b a r b e r $
8	4	b a r h a b a r b e r $
9	12	b e r $
10	13	e r $
11	7	h a b a r b e r $
12	14	r $
13	3	r b a r h a b a r b e r $
14	11	r b e r $
15	6	r h a b a r b e r $

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

n = Textlänge, m = Pattern-Länge

- Naiv: $\mathcal{O}(n \cdot m)$
- KMP: $\mathcal{O}(n + m)$
- Mit Suffix-Arrays zunächst: $\mathcal{O}(m \cdot \log n)$
- Optimiert: $\mathcal{O}(m + \log n)$

Suche mit Suffix-Arrays

Ablauf

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]	
1	15	$
2	8	a b a r b e r $
3	2	a r b a r h a b a r b e r $
4	10	a r b e r $
5	5	a r h a b a r b e r $
6	1	b a r b a r h a b a r b e r $
7	9	b a r b e r $
8	4	b a r h a b a r b e r $
9	12	b e r $
10	13	e r $
11	7	h a b a r b e r $
12	14	r $
13	3	r b a r h a b a r b e r $
14	11	r b e r $
15	6	r h a b a r b e r $

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis

Suche mit Suffix-Arrays

Ablauf

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

Suche: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start binäre Suche

```
$l = 1, r = n$
while $(l < r)$ do
    $q = \lfloor \frac{l+r}{2} \rfloor$
    if $(P > T_{SA[q]..SA[q]+m-1})$
        then $l = q + 1$
        else $r = q$
$s = l$
if $(P \neq T_{SA[s]..SA[s]+m-1})$
    then break
```

- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start binäre Suche

```
l = 1, r = n
while (l < r) do
    $q = \lfloor \frac{l+r}{2} \rfloor$
    if $(P > T_{SA[q]..SA[q]+m-1})$
        then $l = q + 1$
        else $r = q$
s = l
if $(P \neq T_{SA[s]..SA[s]+m-1})$
    then break
```

- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

	i	SA[i]	
	1	15	$
	2	8	a b a r b e r $
	3	2	a r b a r h a b a r b e r $
	4	10	a r b e r $
	5	5	a r h a b a r b e r $
	6	1	b a r b a r h a b a r b e r $
	7	9	b a r b e r $
$q =$	8	4	b a r h a b a r b e r $
	9	12	b e r $
	10	13	e r $
	11	7	h a b a r b e r $
	12	14	r $
	13	3	r b a r h a b a r b e r $
	14	11	r b e r $
	15	6	r h a b a r b e r $

$l = 1, r = 16$

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start binäre Suche

```
$l = 1, r = n$
while $(l < r)$ do
    $q = \lfloor \frac{l+r}{2} \rfloor$
    if $(P > T_{SA[q]..SA[q]+m-1})$
        then $l = q + 1$
        else $r = q$
$s = l$
if $(P \neq T_{SA[s]..SA[s]+m-1})$
    then break
```

- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

	i	SA[i]	
	1	15	$
	2	8	abarber$
	3	2	arbarhabarber$
$q =$	4	10	arber$
	5	5	arhabarber$
	6	1	barbarhabarber$
	7	9	barber$
	8	4	barhabarber$
	9	12	ber$
	10	13	er$
	11	7	habarber$
	12	14	r$
	13	3	rbarhabarber$
	14	11	rber$
	15	6	rhabarber$

$l = 1, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start binäre Suche

```
$l = 1, r = n$
while $(l < r)$ do
    $q = \lfloor \frac{l+r}{2} \rfloor$
    if $(P > T_{SA[q]..SA[q]+m-1})$
        then $l = q + 1$
        else $r = q$
$s = l$
if $(P \neq T_{SA[s]..SA[s]+m-1})$
    then break
```

- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
$q =$ 6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

$l = 5, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start binäre Suche

```
$l = 1, r = n$
while $(l < r)$ do
    $q = \lfloor \frac{l+r}{2} \rfloor$
    if $(P > T_{SA[q]..SA[q]+m-1})$
        then $l = q + 1$
        else $r = q$
$s = l$
if $(P \neq T_{SA[s]..SA[s]+m-1})$
    then break
```

- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

	i	SA[i]	
	1	15	$
	2	8	abarber$
	3	2	arbarhabarber$
	4	10	arber$
$q =$	5	5	arhabarber$
	6	1	barbarhabarber$
	7	9	barber$
	8	4	barhabarber$
	9	12	ber$
	10	13	er$
	11	7	habarber$
	12	14	r$
	13	3	rbarhabarber$
	14	11	rber$
	15	6	rhabarber$

$l = 5, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start binäre Suche

```
$l = 1, r = n$
while ($l < r$) do
    $q = \lfloor \frac{l+r}{2} \rfloor$
    if ($P > T_{SA[q]..SA[q]+m-1}$)
        then $l = q + 1$
        else $r = q$
$s = l$
if ($P \neq T_{SA[s]..SA[s]+m-1}$)
    then break
```

- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

$l = 6, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start
- finde Ende binäre Suche

```
$l = s, r = n$
while $(l < r)$ do
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P = T_{SA[q]..SA[q]+m-1})$
        then $l = q$
        else $r = q - 1$
$t = l$
```

- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
9	12	ber$
$q =$ 10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

$l = 5, r = 15$

Suche mit Suffix-Arrays

Ablauf

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start
- finde Ende binäre Suche

```
$l = s, r = n$
while $(l < r)$ do
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P = T_{SA[q]..SA[q]+m-1})$
        then $l = q$
        else $r = q - 1$
$t = l$
```

- Ergebnis

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
$q =$ 7	9	barber$
8	4	barhabarber$
9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

$l = 5, r = 9$

Suche mit Suffix-Arrays

Ablauf

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start
- finde Ende binäre Suche

```
$l = s, r = n$
while $(l < r)$ do
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P = T_{SA[q]..SA[q]+m-1})$ $q =$
        then $l = q$
        else $r = q - 1$
$t = l$
```

- Ergebnis

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

$l = 7, r = 9$

Suche mit Suffix-Arrays

Ablauf

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start
- finde Ende binäre Suche

```
$l = s, r = n$
while $(l < r)$ do
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P = T_{SA[q]..SA[q]+m-1})$
        then $l = q$
        else $r = q - 1$
$t = l$
```

- Ergebnis

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
$q =$ 9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

$l = 8, r = 9$

Suche mit Suffix-Arrays

Ablauf

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suche: $P =$ `bar`

- (SA bestimmen)
- finde Start
- finde Ende binäre Suche

```
$l = s, r = n$
while $(l < r)$ do
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P = T_{SA[q]..SA[q]+m-1})$
        then $l = q$
        else $r = q - 1$
$t = l$
```

- Ergebnis

i	SA[i]	
1	15	$
2	8	abarber$
3	2	arbarhabarber$
4	10	arber$
5	5	arhabarber$
6	1	barbarhabarber$
7	9	barber$
8	4	barhabarber$
9	12	ber$
10	13	er$
11	7	habarber$
12	14	r$
13	3	rbarhabarber$
14	11	rber$
15	6	rhabarber$

$l = 8, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis
  - $t - s + 1$
    counting query
  - $\{SA[s], \dots, SA[t]\}$
    reporting query

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

	i	SA[i]	
	1	15	$
	2	8	abarber$
	3	2	arbarhabarber$
	4	10	arber$
	5	5	arhabarber$
	6	1	barbarhabarber$
	7	9	barber$
$q =$	8	4	barhabarber$
	9	12	ber$
	10	13	er$
	11	7	habarber$
	12	14	r$
	13	3	rbarhabarber$
	14	11	rber$
	15	6	rhabarber$

$s = 6, t = 8$

Suche mit Suffix-Arrays

Zusammenfassung

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung

  → einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  → danach Anfragen in $\mathcal{O}(m \log n)$ möglich, statt in $\mathcal{O}(m+n)$

  gut, wenn auf einem Text viele Anfragen stattfinden

- Ausnutzung der Eigenschaften des Suffix-Arrays

  → jeder Substring ist Präfix eines Suffix
  → alle Substrings liegen “sortiert” vor
  möglische Ausnahme: Substring ist Präfix von Substring

  das Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge

LCP-Array

Längster gemeinsamer Präfix

Definition:

- $\text{LCP}[i]$: Länge des längsten gemeinsamen Präfixes von je zwei lexikographisch benachbarten Suffixen $A[\text{SA}[i-1]\dots n]$ und $A[\text{SA}[i]\dots n]$

Erweiterung auf beliebige Suffixe

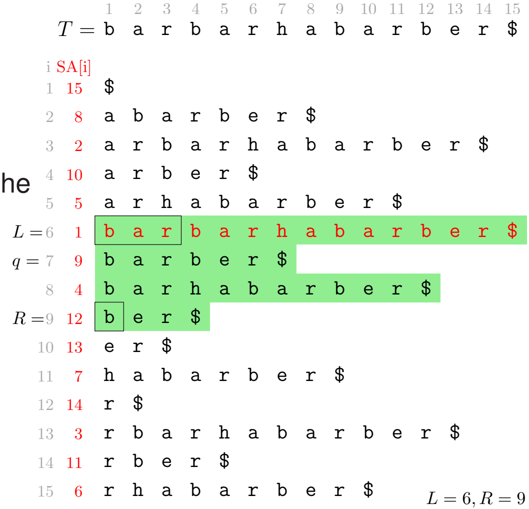
- $\text{LCP}[i][j]$: Länge des längsten gemeinsamen Präfix beliebiger lexikographischer Suffixe $A[\text{SA}[i]\dots n]$ und $A[\text{SA}[j]\dots n]$
- Konstruktion: $\mathcal{O}(n \log n)$ Zeit und Platz
- Zugriff: $\mathcal{O}(1)$

Schnelle Suche mit Suffix-Arrays

Erster Ansatz

Suche: $P = \texttt{bar}$

- Ziel: kein wiederholtes Vergleichen von Zeichen aus P
- Nutze LCP-Array um Suche zu beschleunigen
- Starte Suche bei *mlr*
  - $l := \mathrm{LCP}(L, P)$
  - $r := \mathrm{LCP}(R, P)$
  - $mlr := \min(l, r)$
  - Update von *l*, *r*, keine Neuberechnung
- Oft $\mathcal{O}(m + \log n)$
- Worst case $\mathcal{O}(m \log n)$



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

	i	SA[i]	
	1	15	$
	2	8	abarber$
	3	2	arbarhabarber$
	4	10	arber$
	5	5	arhabarber$
$L =$	6	1	barbarhabarber$
$q =$	7	9	barber$
	8	4	barhabarber$
$R =$	9	12	ber$
	10	13	er$
	11	7	habarber$
	12	14	r$
	13	3	rbarhabarber$
	14	11	rber$
	15	6	rhabarber$

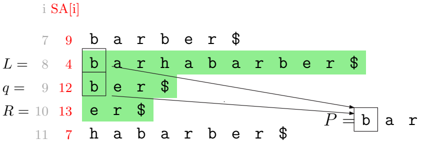
$L = 6, R = 9$
$l = 3, r = 1$
$mlr := 1 = \min(l, r)$

Schnelle Suche mit Suffix-Arrays

Redundante Vergleiche

Problem

- Falls $l \neq r \rightarrow$ wiederholtes Vergleichen



	i	SA[i]	
	7	9	b a r b e r $
$L =$	8	4	b a r h a b a r b e r $
$q =$	9	12	b e r $
$R =$	10	13	e r $
	11	7	h a b a r b e r $

$P =$ b a r

Definition

- Vergleich eines Zeichens aus P ist redundant, falls das Zeichen vorher schon einmal überprüft wurde.

Ziel

- Beschränke rendundante Vergleiche auf $\mathcal{O}(1)$ pro Iteration
- Vergleiche bei $\mathsf{max}(l, r)$ beginnen

Suche mit Suffix-Arrays

Ablauf

Ansatz

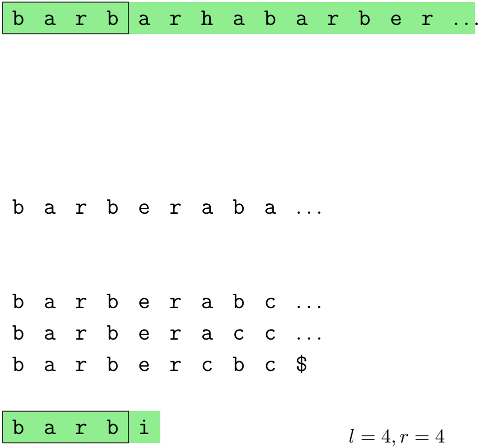
```
if ($l = r$)
    start at $mlr$
    Update $l, r, L, R$
if ($l > r \wedge \mathrm{LCP}[L, q] > l$)
    $L := q + 1$
    Update $l$
if ($l > r \wedge \mathrm{LCP}[L, q] < l$)
    $R := q$
    $r := \mathrm{LCP}[L, q]$
if ($l > r \wedge \mathrm{LCP}[L, q] = l$)
    start at $l$
```

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

- **if** $(l = r)$
  start at *mlr*
  Update l, r, L, R
- **if** $(l > r \wedge \mathrm{LCP}[L, q] > l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] < l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] = l)$



```
b a r b a r h a b a r b e r ...

b a r b e r a b a ...

b a r b e r a b c ...
b a r b e r a c c ...
b a r b e r c b c $

b a r b i
```

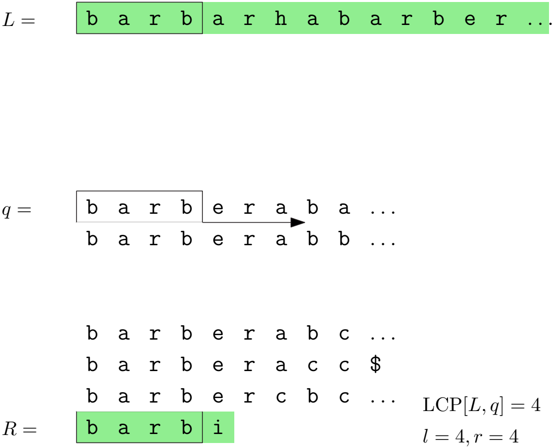
$l = 4, r = 4$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

- **if** $(l = r)$
  start at *mlr*
  Update l, r, L, R
- **if** $(l > r \wedge \mathrm{LCP}[L, q] > l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] < l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] = l)$



$L =$ `b a r b a r h a b a r b e r ...`

$q =$ `b a r b e r a b a ...`

`b a r b e r a b b ...`

`b a r b e r a b c ...`

`b a r b e r a c c $`

`b a r b e r c b c ...`

$R =$ `b a r b i`

$\mathrm{LCP}[L, q] = 4$

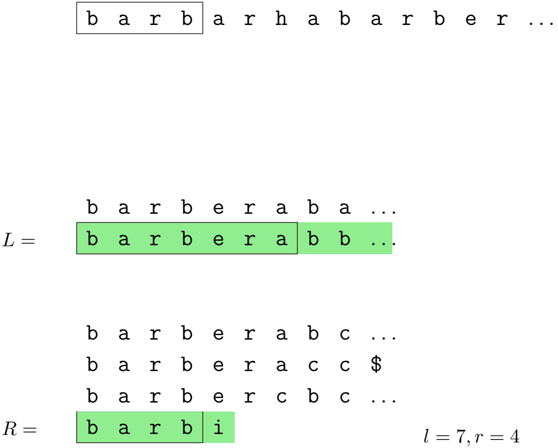
$l = 4, r = 4$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

- **if** $(l = r)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] > l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] < l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] = l)$



`barbarhabarber...`

`barberaba...`

$L =$ `barberabb...`

`barberabc...`

`barberacc$`

`barbercbc...`

$R =$ `barbi`

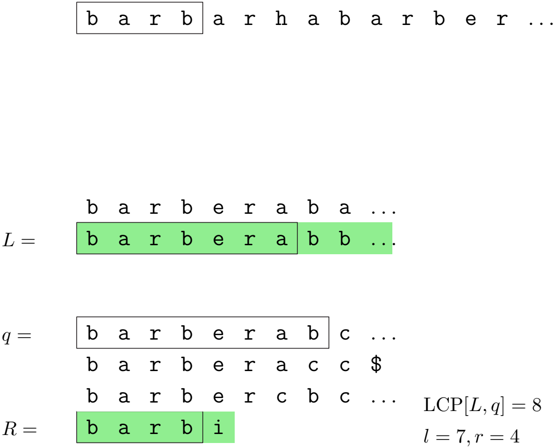
$l = 7, r = 4$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

- **if** $(l = r)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] > l)$
  $L := q + 1$
  Update l
- **if** $(l > r \wedge \mathrm{LCP}[L, q] < l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] = l)$



barbarhabarber...

barberaba...

$L =$ barberabb...

$q =$ barberabc...

barberacc$

barbercbc...

$R =$ barbi

$\mathrm{LCP}[L, q] = 8$

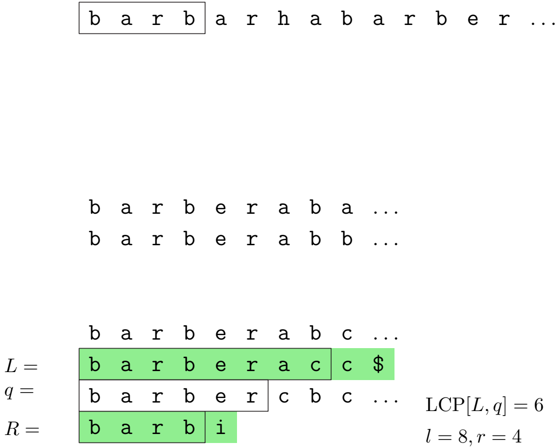
$l = 7, r = 4$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

- **if** $(l = r)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] > l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] < l)$
  $R := q$
  $r := \mathrm{LCP}[L, q]$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] = l)$

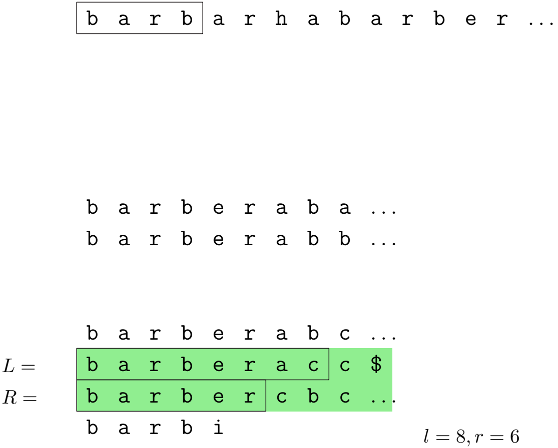


Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `barberac`

- **if** $(l = r)$
- **if** $(l > r \wedge \text{LCP}[L, q] > l)$
- **if** $(l > r \wedge \text{LCP}[L, q] < l)$
  $R := q$
  $r := \text{LCP}[L, q]$
- **if** $(l > r \wedge \text{LCP}[L, q] = l)$



barbarhabarber ...

barberaba ...

barberabb ...

barberabc ...

$L =$ barberacc$

$R =$ barbercbc ...

barbi

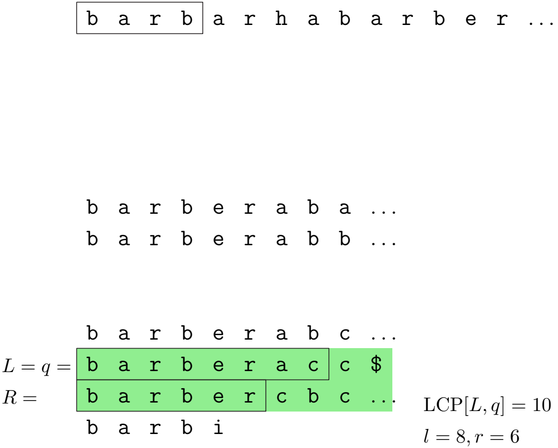
$l = 8, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `barberac`

- **if** $(l = r)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] > l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] < l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] = l)$



`barbarhabarber...`

`barberaba...`

`barberabb...`

`barberabc...`

$L = q =$ `barberacc$`

$R =$ `barbercbc...`

`barbi`

$\mathrm{LCP}[L, q] = 10$

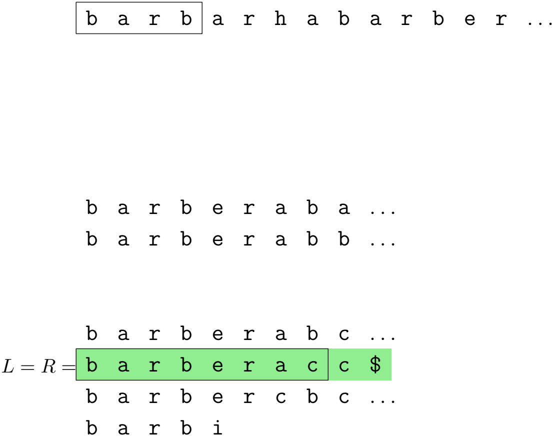
$l = 8, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

- **if** $(l = r)$
- **if** $(l > r \land \text{LCP}[L, q] > l)$
- **if** $(l > r \land \text{LCP}[L, q] < l)$
- **if** $(l > r \land \text{LCP}[L, q] = l)$
  start at l



$\boxed{\texttt{barb}}\texttt{arhabarber...}$

`barberaba...`

`barberabb...`

`barberabc...`

$L = R = \boxed{\texttt{barberac}}\texttt{c\$}$

`barbercbc...`

`barbi`

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

- **if** $(l = r)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] > l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] < l)$
- **if** $(l > r \wedge \mathrm{LCP}[L, q] = l)$

Laufzeit

- LCP + SA: $\mathcal{O}(m + \log n)$ Vergleiche
- Beweisidee
  - l, r werden nur größer
  - Anzahl an redundanten Vergleichen pro Rekursion konstant

Suche mit Suffix-Arrays

Zusammenfassung

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung
  - → einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  - → danach Anfragen in $\mathcal{O}(m \log n)$ möglich, statt in $\mathcal{O}(m + n)$

  gut, wenn auf einem Text viele Anfragen stattfinden

- Verhindern redundanter Vergleiche
  - → einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  - → einmal LCP-Array generieren in $\mathcal{O}(n)$,
  - → einmal erweitertes LCP-Array generieren in $\mathcal{O}(n \log n)$,
  - → danach Anfragen in $\mathcal{O}(m + \log n)$

Karlsruher Institut für Technologie

Algorithmen 2

Fortsetzung Übung Stringology

Florian Kurpicz

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n, dann ist die i-te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i)$$

- i-te zyklische Rotation ist die Konkatenation des i-ten Suffixes und des $(i-1)$-ten prefixes

$T = \texttt{ababcabcabba\$}$

$T^{(1)}$	$T^{(2)}$	$T^{(3)}$	$T^{(4)}$	$T^{(5)}$	$T^{(6)}$	$T^{(7)}$	$T^{(8)}$	$T^{(9)}$	$T^{(10)}$	$T^{(11)}$	$T^{(12)}$	$T^{(13)}$
a	b	a	b	c	a	b	c	a	b	b	a	\$
b	a	b	c	a	b	c	a	b	b	a	\$	a
a	b	c	a	b	c	a	b	b	a	\$	a	b
b	c	a	b	c	a	b	b	a	\$	a	b	a
c	a	b	c	a	b	b	a	\$	a	b	a	b
a	b	c	a	b	b	a	\$	a	b	a	b	c
b	c	a	b	b	a	\$	a	b	a	b	c	a
c	a	b	b	a	\$	a	b	a	b	c	a	b
a	b	b	a	\$	a	b	a	b	c	a	b	c
b	b	a	\$	a	b	a	b	c	a	b	c	a
b	a	\$	a	b	a	b	c	a	b	c	a	b
a	\$	a	b	a	b	c	a	b	c	a	b	b
\$	a	b	a	b	c	a	b	c	a	b	b	a

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n, dann ist die i-te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i)$$

- i-te zyklische Rotation ist die Konkatenation des i-ten Suffixes und des $(i-1)$-ten prefixes

Definition: Burrows-Wheeler Transform (Alt.)

Sei T ein Text und M die Matrix, welche alle zyklischen Rotationen von T in lexikographischer Reihenfolge als Spalten enthält, dann ist die **BWT** die letzte Zeile von M

$T = \texttt{ababcabcabba\$}$

$T^{(1)}$	$T^{(2)}$	$T^{(3)}$	$T^{(4)}$	$T^{(5)}$	$T^{(6)}$	$T^{(7)}$	$T^{(8)}$	$T^{(9)}$	$T^{(10)}$	$T^{(11)}$	$T^{(12)}$	$T^{(13)}$
a	b	a	b	c	a	b	c	a	b	b	a	$
b	a	b	c	a	b	c	a	b	b	a	$	a
a	b	c	a	b	c	a	b	b	a	$	a	b
b	c	a	b	c	a	b	b	a	$	a	b	a
c	a	b	c	a	b	b	a	$	a	b	a	b
a	b	c	a	b	b	a	$	a	b	a	b	c
b	c	a	b	b	a	$	a	b	a	b	c	a
c	a	b	b	a	$	a	b	a	b	c	a	b
a	b	b	a	$	a	b	a	b	c	a	b	c
b	b	a	$	a	b	a	b	c	a	b	c	a
b	a	$	a	b	a	b	c	a	b	c	a	b
a	$	a	b	a	b	c	a	b	c	a	b	b
$	a	b	a	b	c	a	b	c	a	b	b	a

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n, dann ist die i-te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i)$$

- i-te zyklische Rotation ist die Konkatenation des i-ten Suffixes und des $(i-1)$-ten prefixes

Definition: Burrows-Wheeler Transform (Alt.)

Sei T ein Text und M die Matrix, welche alle zyklischen Rotationen von T in lexikographischer Reihenfolge als Spalten enthält, dann ist die **BWT** die letzte Zeile von M

T = ababcabcabba$

$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
$	a	a	a	a	a	b	b	b	b	b	c	c
a	$	b	b	b	b	a	a	b	c	c	a	a
b	a	a	b	c	c	$	b	a	a	a	b	b
a	b	b	a	a	a	a	c	$	b	b	b	c
b	a	c	$	b	b	b	a	a	b	c	a	a
c	b	a	a	b	c	a	b	b	a	a	$	b
a	c	b	b	a	a	b	c	a	$	b	a	b
b	a	c	a	$	b	c	a	b	a	b	b	a
c	b	a	b	a	b	a	b	c	b	a	a	$
a	c	b	c	b	a	b	b	a	a	$	b	a
b	a	b	a	a	$	c	a	b	b	a	c	b
b	b	a	b	b	a	a	$	c	c	b	a	a
a	b	$	c	c	b	b	a	a	a	a	b	b

Karlsruher Institut für Technologie

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n, dann ist die i-te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i)$$

- i-te zyklische Rotation ist die Konkatenation des i-ten Suffixes und des $(i-1)$-ten prefixes

Definition: Burrows-Wheeler Transform (Alt.)

Sei T ein Text und M die Matrix, welche alle zyklischen Rotationen von T in lexikographischer Reihenfolge als Spalten enthält, dann ist die **BWT** die letzte Zeile von M

T = ababcabcabba$

$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
$	a	a	a	a	a	b	b	b	b	b	c	c
a	$	b	b	b	b	a	a	b	c	c	a	a
b	a	a	b	c	c	$	b	a	a	a	b	b
a	b	b	a	a	a	a	c	$	b	b	b	c
b	a	c	$	b	b	b	a	a	b	c	a	a
c	b	a	a	b	c	a	b	b	a	a	$	b
a	c	b	b	a	a	b	c	a	$	b	a	b
b	a	c	a	$	b	c	a	b	a	b	b	a
c	b	a	b	a	b	a	b	c	b	a	a	$
a	c	b	c	b	a	b	b	a	a	$	b	a
b	a	b	a	a	$	c	a	b	b	a	c	b
b	b	a	b	b	a	a	$	c	c	b	a	a
a	b	$	c	c	b	b	a	a	a	a	b	b

Erste und Letzte Zeile

- zwei wichtige Zeilen in der Matrix
- es gibt eine spezielle Beziehung zwischen den beiden Zeilen
- alle anderen Zeilen werden nicht benötigt

Erste Zeile **F**

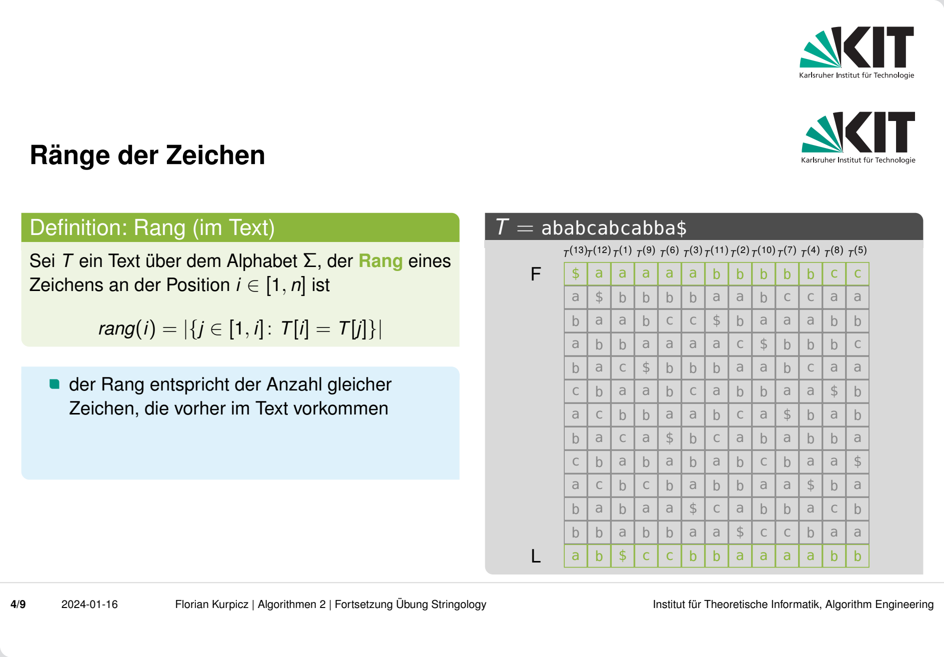
- enthält alle Zeichen von T in sortierter Reihenfolge

Letzte Zeile **L**

- ist die *BWT*

$T =$ ababcabcabba$

	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b



Ränge der Zeichen

Definition: Rang (im Text)

Sei T ein Text über dem Alphabet Σ, der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$rang(i) = |\{j \in [1, i]\colon\ T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T = ababcabcabba$

	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

Ränge der Zeichen

Definition: Rang (im Text)

Sei T ein Text über dem Alphabet Σ, der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$rang(i) = |\{j \in [1, i] \colon T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T	a	b	a	b	c	a	b	c	a	b	b	a	$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

$T = \mathtt{ababcabcabba\$}$

	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

Ränge der Zeichen

Definition: Rang (im Text)

Sei T ein Text über dem Alphabet Σ, der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$rang(i) = |\{j \in [1, i]\colon T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T	a	b	a	b	c	a	b	c	a	b	b	a	\$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

T = ababcabcabba\$

	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	\$	a	a	a	a	a	b	b	b	b	b	c	c
	1	5	1	4	3	2	5	1	4	3	2	1	2
	a	\$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	\$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	\$	b	b	b	c
	b	a	c	\$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	\$	b
	a	c	b	b	a	a	b	c	a	\$	b	a	b
	b	a	c	a	\$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	\$
	a	c	b	c	b	a	b	b	a	a	\$	b	a
	b	a	b	a	a	\$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	\$	c	c	b	a	a
L	a	b	\$	c	c	b	b	a	a	a	a	b	b

Ränge der Zeichen

Definition: Rang (im Text)

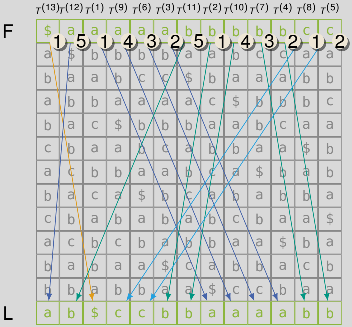
Sei T ein Text über dem Alphabet Σ, der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$rang(i) = |\{j \in [1, i] \colon T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T	a	b	a	b	c	a	b	c	a	b	b	a	\$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

$T =$ ababcabcabba\$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	\$	a	a	a	a	a	b	b	b	b	b	c	c
	1	5	1	4	3	2	5	1	4	3	2	1	2
	a	\$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	\$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	\$	b	b	b	c
	b	a	c	\$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	\$	b
	a	c	b	b	a	a	b	c	a	\$	b	a	b
	b	a	c	a	\$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	\$
	a	c	b	c	b	a	b	b	a	a	\$	b	a
	b	a	b	a	a	\$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	\$	c	c	b	a	a
L	a	b	\$	c	c	b	b	a	a	a	a	b	b

Ränge der Zeichen

Definition: Rang (im Text)

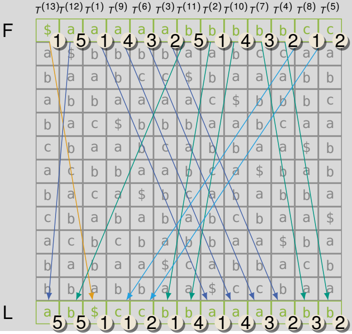
Sei T ein Text über dem Alphabet Σ, der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$rang(i) = |\{j \in [1, i] \colon T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T	a	b	a	b	c	a	b	c	a	b	b	a	\$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

$T = \texttt{ababcabcabba\$}$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	\$	a	a	a	a	a	b	b	b	b	b	c	c
	1	5	1	4	3	2	5	1	4	3	2	1	2
	a	\$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	\$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	\$	b	b	b	c
	b	a	c	\$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	\$	b
	a	c	b	b	a	a	b	c	a	\$	b	a	b
	b	a	c	a	\$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	\$
	a	c	b	c	b	a	b	b	a	a	\$	b	a
	b	a	b	a	a	\$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	\$	c	c	b	a	a
L	a	b	\$	c	c	b	b	a	a	a	a	b	b
	5	5	1	1	2	1	4	1	4	3	2	3	2

Ränge der Zeichen

Definition: Rang (im Text)

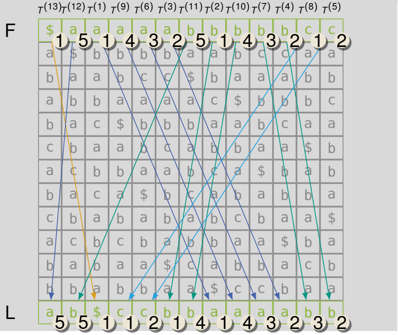
Sei T ein Text über dem Alphabet Σ, der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$rang(i) = |\{j \in [1, i] \colon T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen
- für jedes Zeichen $\alpha \in \Sigma$ ist die Reihenfolge der Ränge in der ersten und letzten Zeile gleich

T	a	b	a	b	c	a	b	c	a	b	b	a	\$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

T = ababcabcabba\$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	\$	a	a	a	a	a	b	b	b	b	b	c	c
	1	5	1	4	3	2	5	1	4	3	2	1	2
	a	\$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	\$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	\$	b	b	b	c
	b	a	c	\$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	\$	b
	a	c	b	b	a	a	b	c	a	\$	b	a	b
	b	a	c	a	\$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	\$
	a	c	b	c	b	a	b	b	a	a	\$	b	a
	b	a	b	a	a	\$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	\$	c	c	b	a	a
L	a	b	\$	c	c	b	b	a	a	a	a	b	b
	5	5	1	1	2	1	4	1	4	3	2	3	2

LF-Mapping (1/2)

- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
  - ermöglicht Mustersuche
  - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T, dann ist das LF-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba\$}$

	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

LF-Mapping (1/2)

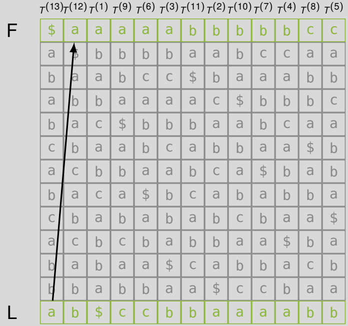
- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
  - ermöglicht Mustersuche
  - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T, dann ist das LF-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \texttt{ababcabcabba\$}$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

LF-Mapping (1/2)

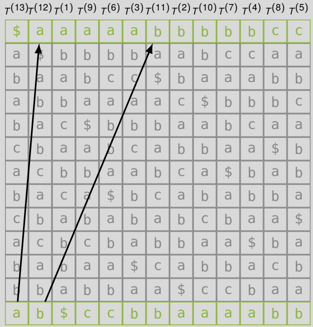
- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
    - ermöglicht Mustersuche
    - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T, dann ist das LF-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = $ ababcabcabba$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

LF-Mapping (1/2)

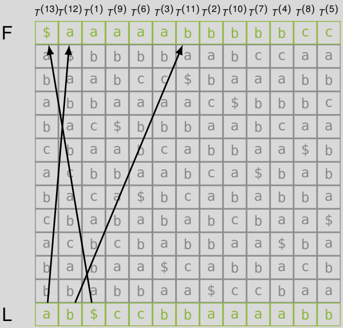
- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
  - ermöglicht Mustersuche
  - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T, dann ist das LF-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \texttt{ababcabcabba\$}$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

LF-Mapping (1/2)

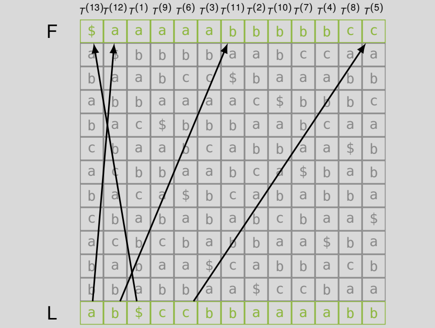
- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
  - ermöglicht Mustersuche
  - Transformation *BWT* zurück zu T

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T, dann ist das *LF*-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \mathtt{ababcabcabba\$}$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

LF-Mapping (1/2)

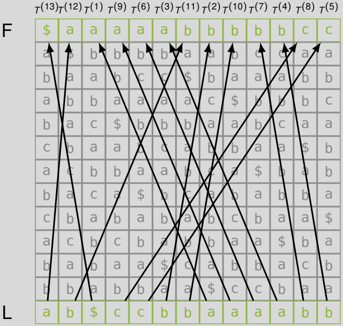
- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
  - ermöglicht Mustersuche
  - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T, dann ist das LF-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba\$}$



	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b

LF-Mapping (2/2)

Definition: *C*-Array und *Rang*-Function

Sei T ein Text der Länge n über dem Alphabet Σ, $\alpha \in \Sigma$ und $i \in [1, n]$, dann gilt

$$C[\alpha] = |i \in [1, n]\colon\ T[i] < \alpha|$$

und

$$rang_{\alpha}(i) = |\{j \in [1, i]\colon\ T[j] = \alpha\}|$$

- C enthält die absolute Anzahl kleinerer Zeichen
- $rang_{\alpha}$ enthält die Anzahl αs in Präfix $T[1..i]$

LF-Mapping (2/2)

Definition: *C*-Array und *Rang*-Function

Sei T ein Text der Länge n über dem Alphabet Σ, $\alpha \in \Sigma$ und $i \in [1, n]$, dann gilt

$$C[\alpha] = |i \in [1, n] \colon T[i] < \alpha|$$

und

$$rang_{\alpha}(i) = |\{j \in [1, i] \colon T[j] = \alpha\}|$$

- C enthält die absolute Anzahl kleinerer Zeichen
- $rang_{\alpha}$ enthält die Anzahl αs in Präfix $T[1..i]$

T	a	b	a	b	c	a	b	c	a	b	b	a	$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

- Ränge jetzt auf der *BWT*
- C ist exklusive Präfixsumme über Histogramm

LF-Mapping (2/2)

Definition: *C*-Array und *Rang*-Function

Sei T ein Text der Länge n über dem Alphabet Σ, $\alpha \in \Sigma$ und $i \in [1, n]$, dann gilt

$$C[\alpha] = |i \in [1, n] \colon T[i] < \alpha|$$

und

$$rang_{\alpha}(i) = |\{j \in [1, i] \colon T[j] = \alpha\}|$$

- C enthält die absolute Anzahl kleinerer Zeichen
- $rang_{\alpha}$ enthält die Anzahl αs in Präfix $T[1..i]$

T	a	b	a	b	c	a	b	c	a	b	b	a	$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

- Ränge jetzt auf der *BWT*
- C ist exklusive Präfixsumme über Histogramm

Definition: *LF*-Mapping (Alt.)

Gegeben die *BWT*, das C-array, und die *rang*-Function, dann gilt

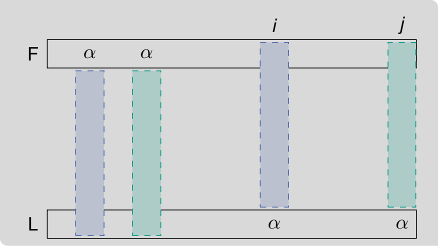
$$LF(i) = C[BWT[i]] + rang_{BWT[i]}(i)$$

Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

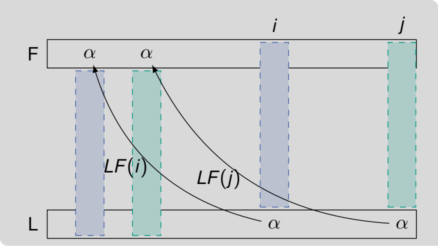
Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück



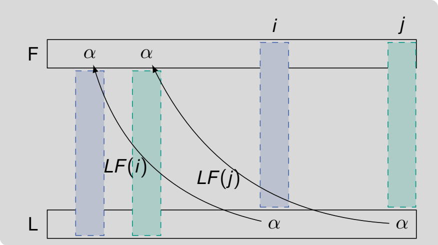
Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück



Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

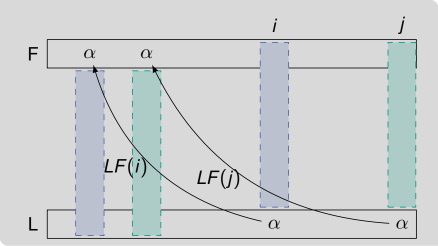


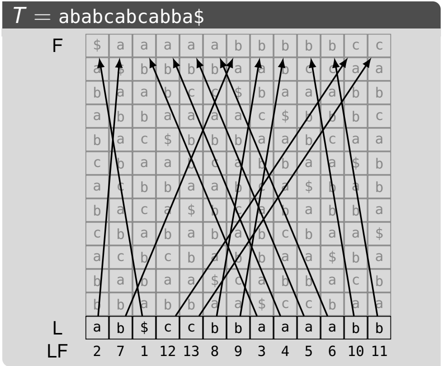
$T =$ ababcabcabba$

F	$	a	a	a	a	a	b	b	b	b	b	c	c
	a	$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	$	b	b	b	c
	b	a	c	$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	$	b
	a	c	b	b	a	a	b	c	a	$	b	a	b
	b	a	c	a	$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	$
	a	c	b	c	b	a	b	b	a	a	$	b	a
	b	a	b	a	a	$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	$	c	c	b	a	a
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück





T = ababcabcabba$

F	$	a	a	a	a	a	b	b	b	b	b	c	c
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende *LF*-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\dots(}_{i-1\text{ times}}LF(1))\dots))]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende *LF*-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\dots(}_{i-1 \text{ times}}LF(1))\dots))]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

- $T[13] = \$$ und $k = 1$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende *LF*-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\ldots(}_{i-1 \text{ times}}LF(1))\ldots))]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = \mathrm{a}$ und $k = LF(1) = 2$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\ldots(}_{i-1\text{ times}}LF(1))\ldots))]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = \mathtt{a}$ und $k = LF(1) = 2$
- $T[11] = L[2] = \mathtt{b}$ und $k = LF(2) = 7$

Umkehrung der BWT (2/2)

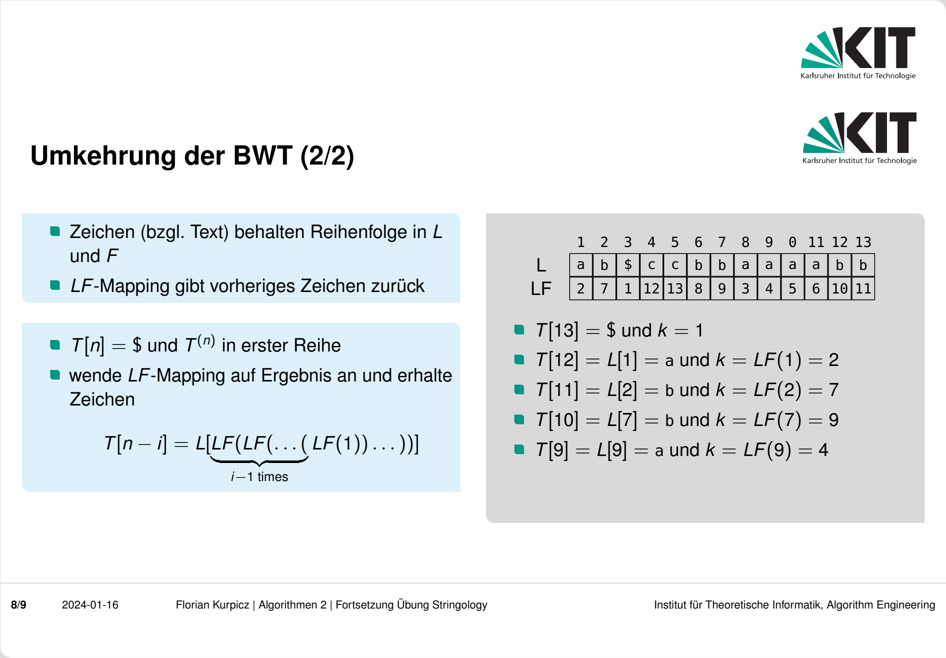
- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\ldots(}_{i-1 \text{ times}}LF(1))\ldots))]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = \mathtt{a}$ und $k = LF(1) = 2$
- $T[11] = L[2] = \mathtt{b}$ und $k = LF(2) = 7$
- $T[10] = L[7] = \mathtt{b}$ und $k = LF(7) = 9$



Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende *LF*-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\dots(}_{i-1\text{ times}}LF(1))\dots))]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = \mathrm{a}$ und $k = LF(1) = 2$
- $T[11] = L[2] = \mathrm{b}$ und $k = LF(2) = 7$
- $T[10] = L[7] = \mathrm{b}$ und $k = LF(7) = 9$
- $T[9] = L[9] = \mathrm{a}$ und $k = LF(9) = 4$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende *LF*-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\ldots(LF(1))\ldots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = \mathrm{a}$ und $k = LF(1) = 2$
- $T[11] = L[2] = \mathrm{b}$ und $k = LF(2) = 7$
- $T[10] = L[7] = \mathrm{b}$ und $k = LF(7) = 9$
- $T[9] = L[9] = \mathrm{a}$ und $k = LF(9) = 4$
- $T[9] = L[4] = \mathrm{c}$ und $k = LF(4) = 12$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in *L* und *F*
- *LF*-Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende *LF*-Mapping auf Ergebnis an und erhalte Zeichen

$$T[n-i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
L	a	b	$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = \mathrm{a}$ und $k = LF(1) = 2$
- $T[11] = L[2] = \mathrm{b}$ und $k = LF(2) = 7$
- $T[10] = L[7] = \mathrm{b}$ und $k = LF(7) = 9$
- $T[9] = L[9] = \mathrm{a}$ und $k = LF(9) = 4$
- $T[9] = L[4] = \mathrm{c}$ und $k = LF(4) = 12$
- ...

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}\$$

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}\$$

- Wie sieht die LZ78-Faktorisierung davon aus?

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}\$$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa| \dots$

Eigenschaften von LZ78

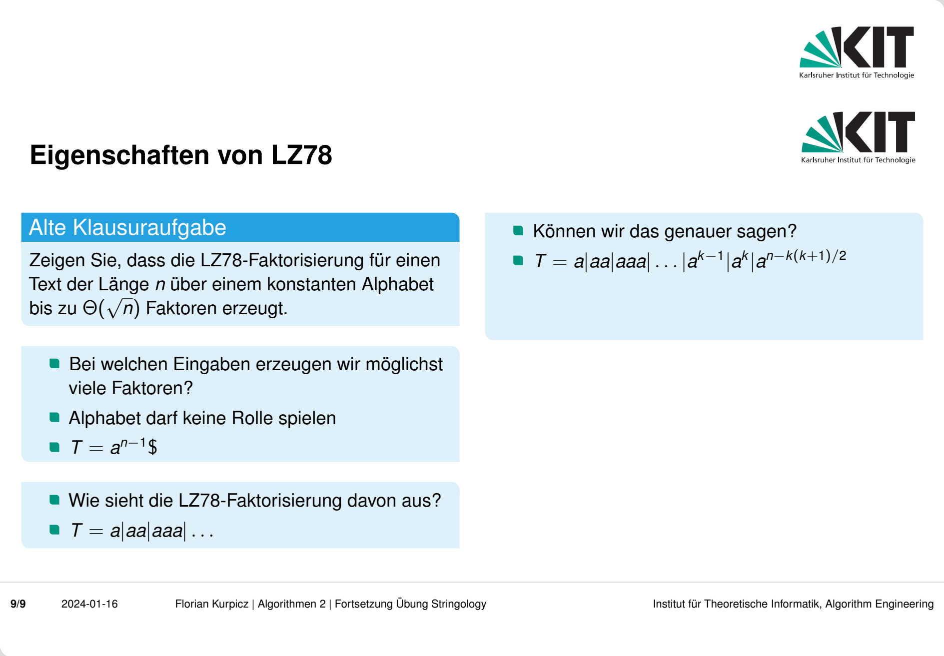
Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}\$$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa|\dots$

- Können wir das genauer sagen?



Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}\$$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa|\dots$

- Können wir das genauer sagen?
- $T = a|aa|aaa|\dots|a^{k-1}|a^{k}|a^{n-k(k+1)/2}$

Eigenschaften von LZ78

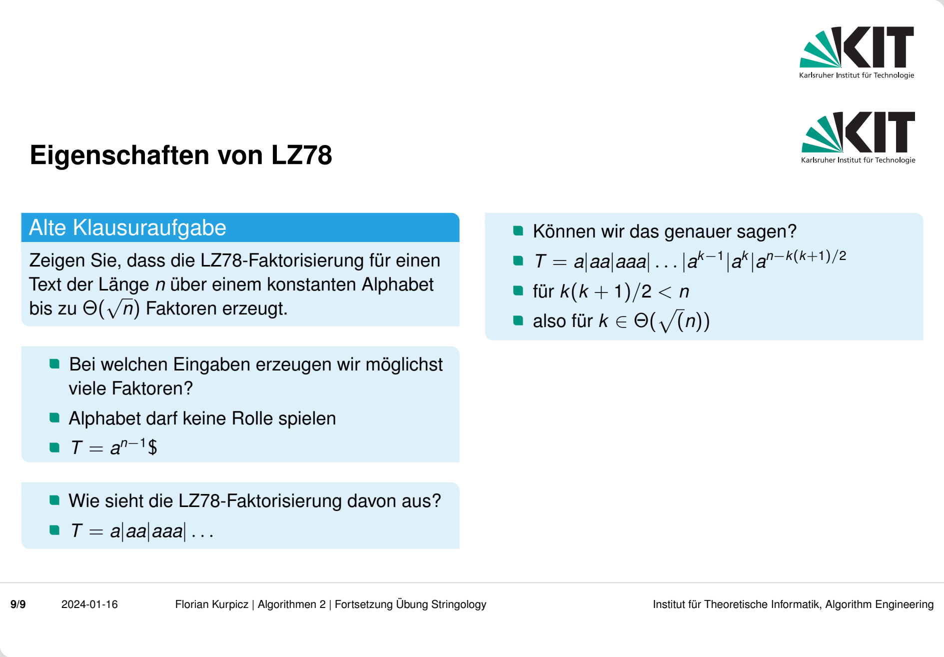
Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}\$$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa|\dots$

- Können wir das genauer sagen?
- $T = a|aa|aaa|\dots|a^{k-1}|a^{k}|a^{n-k(k+1)/2}$
- für $k(k+1)/2 < n$



Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}\$$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa|\dots$

- Können wir das genauer sagen?
- $T = a|aa|aaa|\dots|a^{k-1}|a^k|a^{n-k(k+1)/2}$
- für $k(k+1)/2 < n$
- also für $k \in \Theta(\sqrt{(n)})$

Ende!

Feierabend!