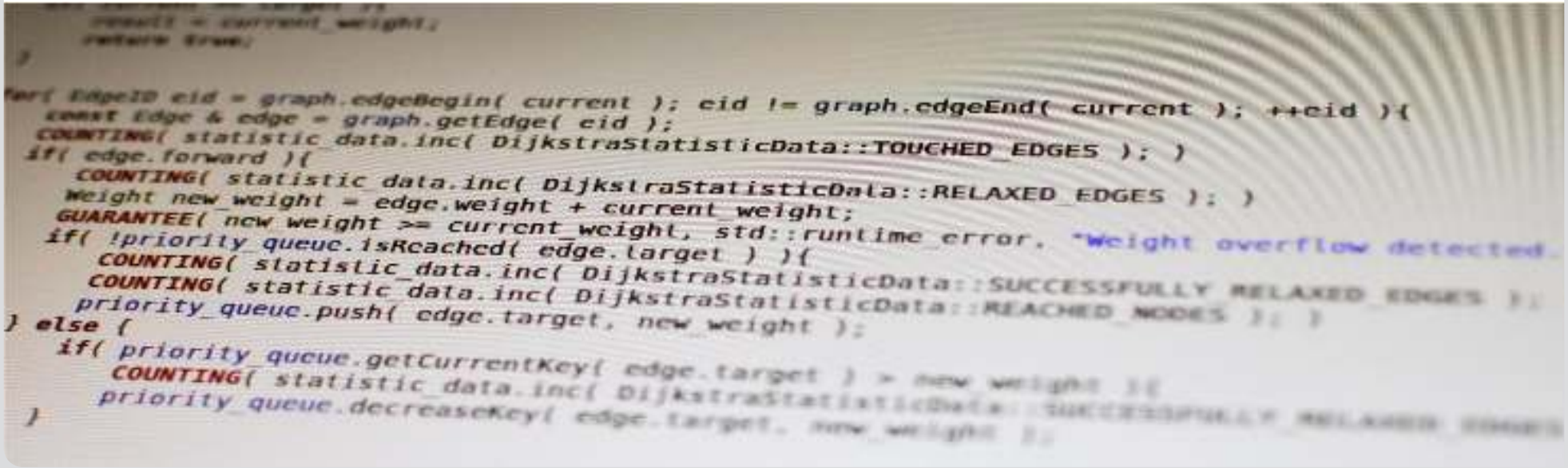
Karlsruher Institut für Technologie

Übung 3 – Algorithmen II

Hans-Peter Lehmann, Daniel Seemaier – *{hans-peter.lehmann, daniel.seemaier}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS21.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

- *in-place Multikey Quicksort*
  Sortierung von Zeichenketten *in-place*

- Suche mit Hilfe von Suffix-Arrays
  - Beschleunigung mittels LCP-Array

Multikey Quicksort

Wiederholung

Bentley, Sedgewick (1997)

Three-way Radix Quicksort

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

- für einen Schlüssel wird *Quicksort* mit drei Fällen ausgeführt
  $\rightarrow$ 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($\langle e \in S : e[i] > p[i] \rangle, i$)

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

S

Multikey Quicksort

Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S — Basisfall

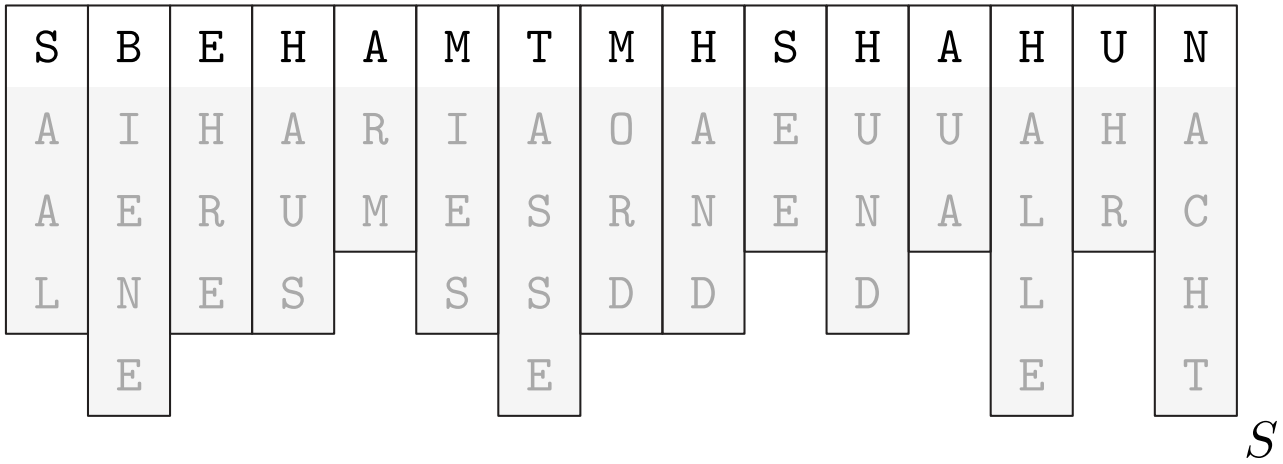
choose $p \in S$ uniformly at random — Pivotelement

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

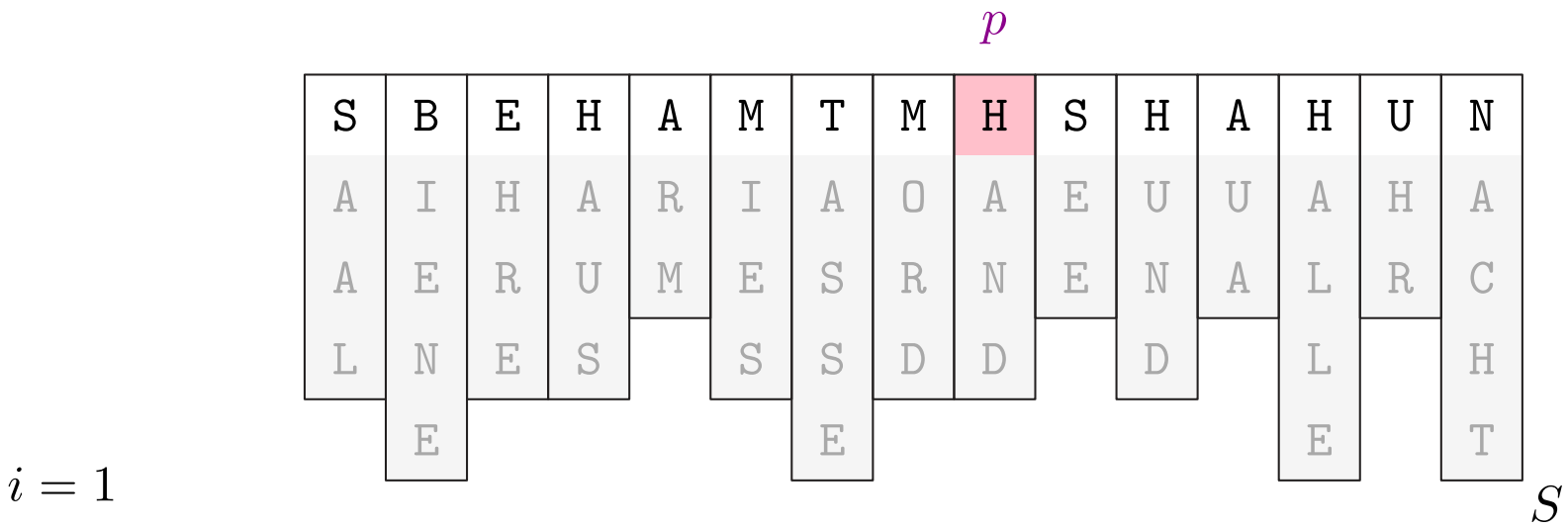
choose $p \in S$ uniformly at random — Pivotelement

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($\langle e \in S : e[i] > p[i] \rangle, i$)



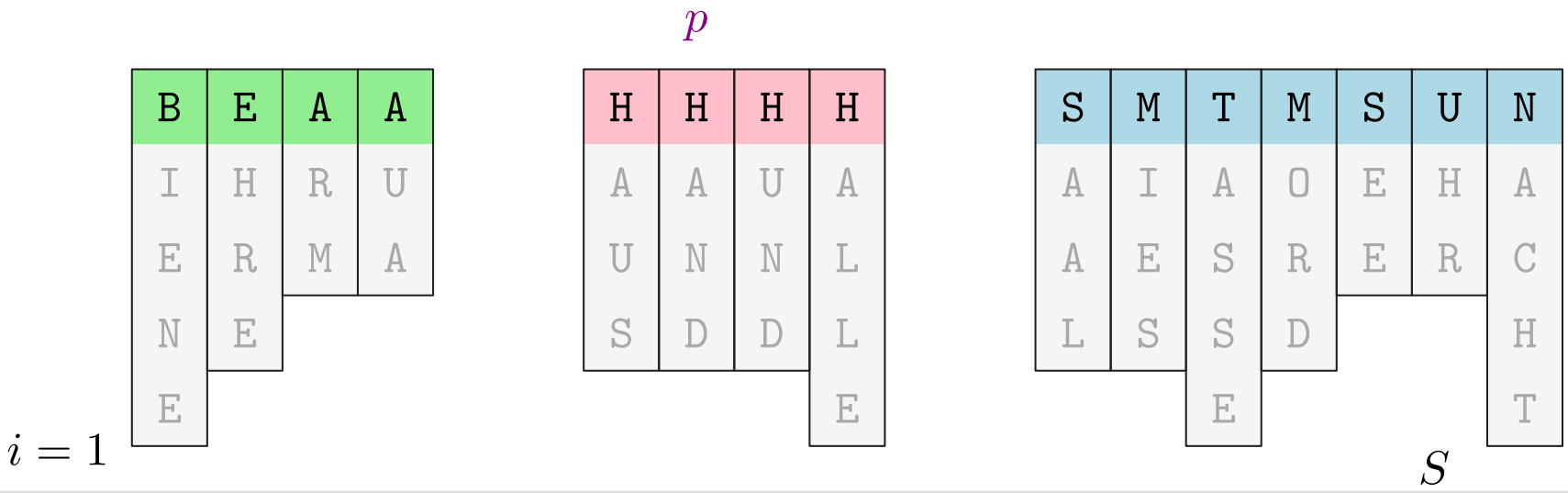
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



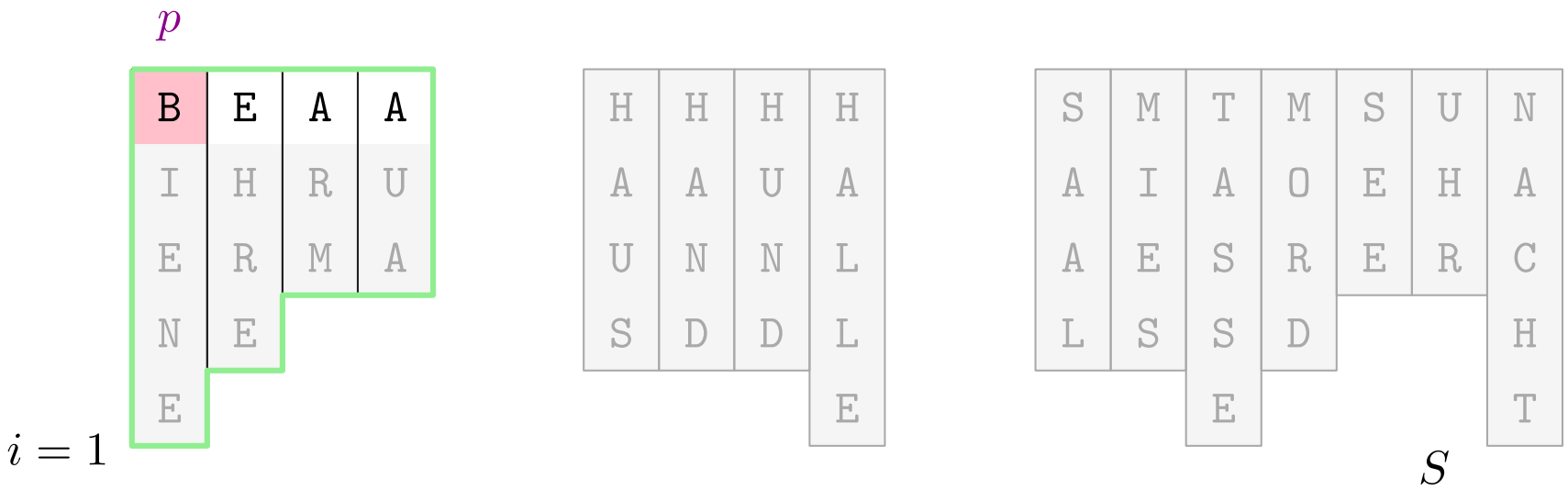
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



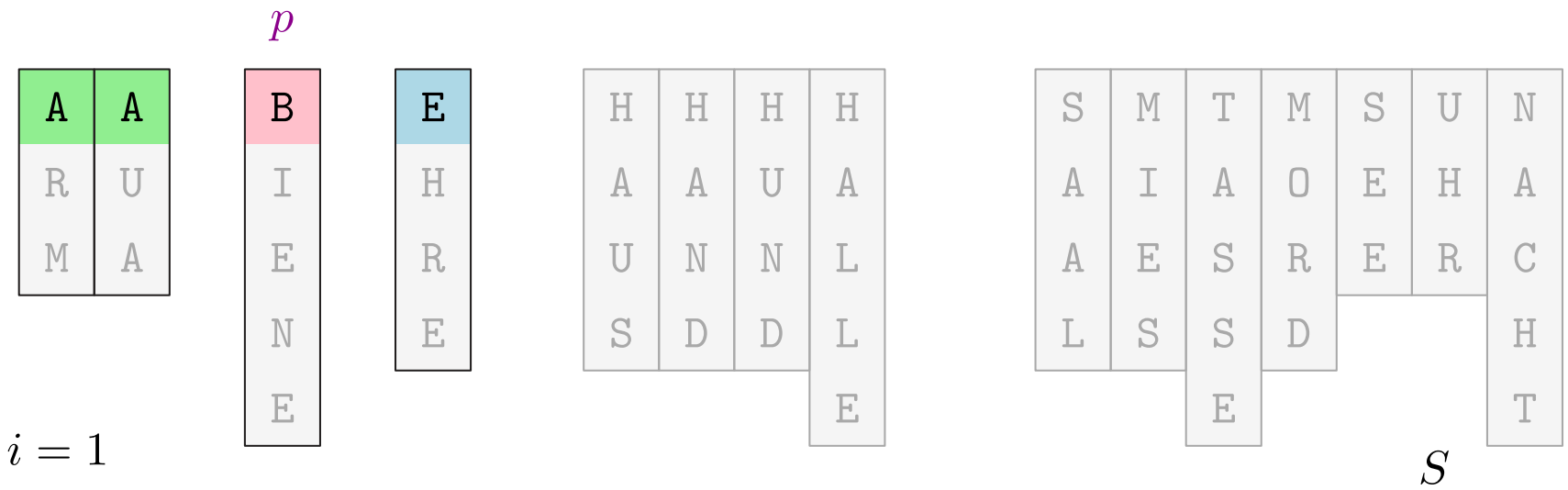
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



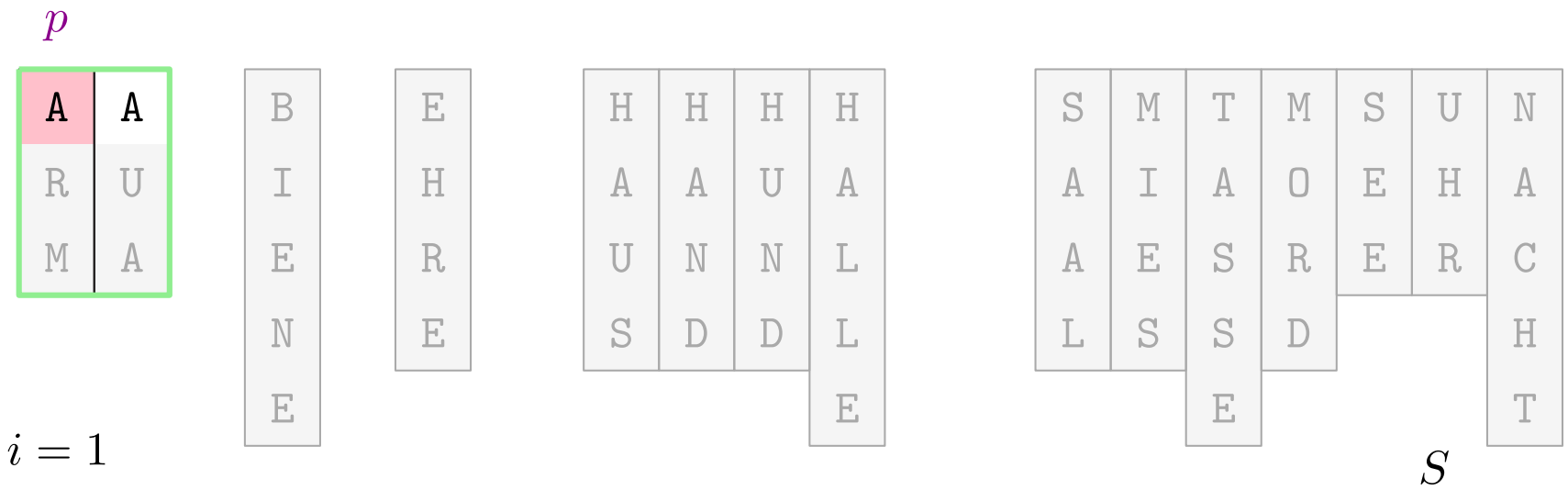
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



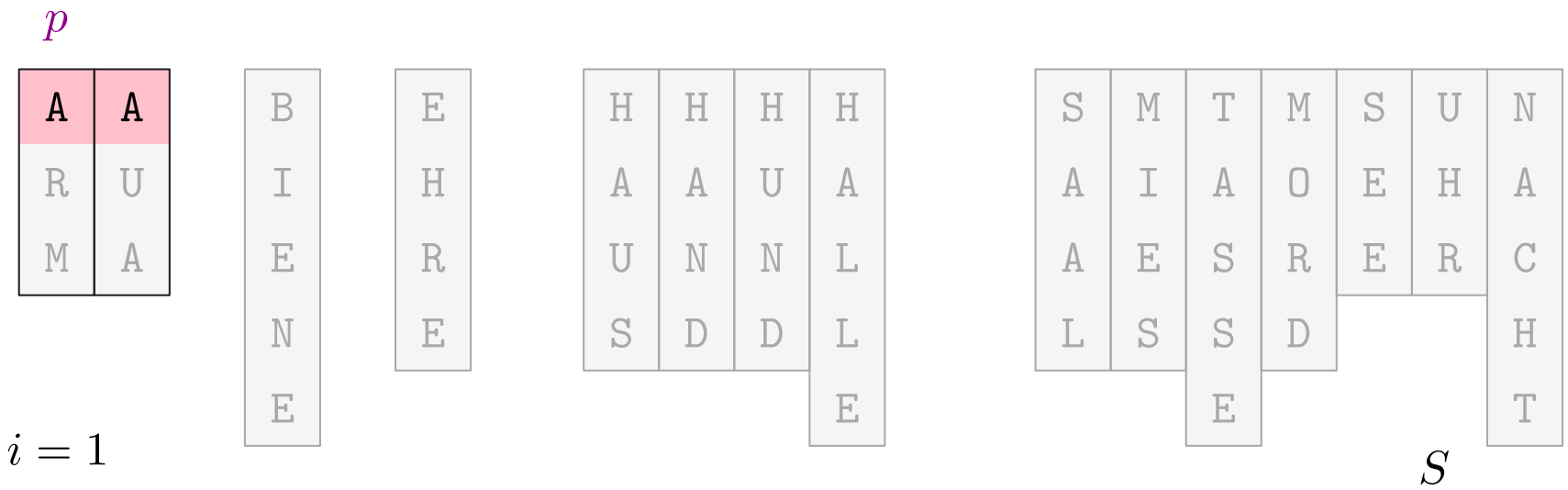
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



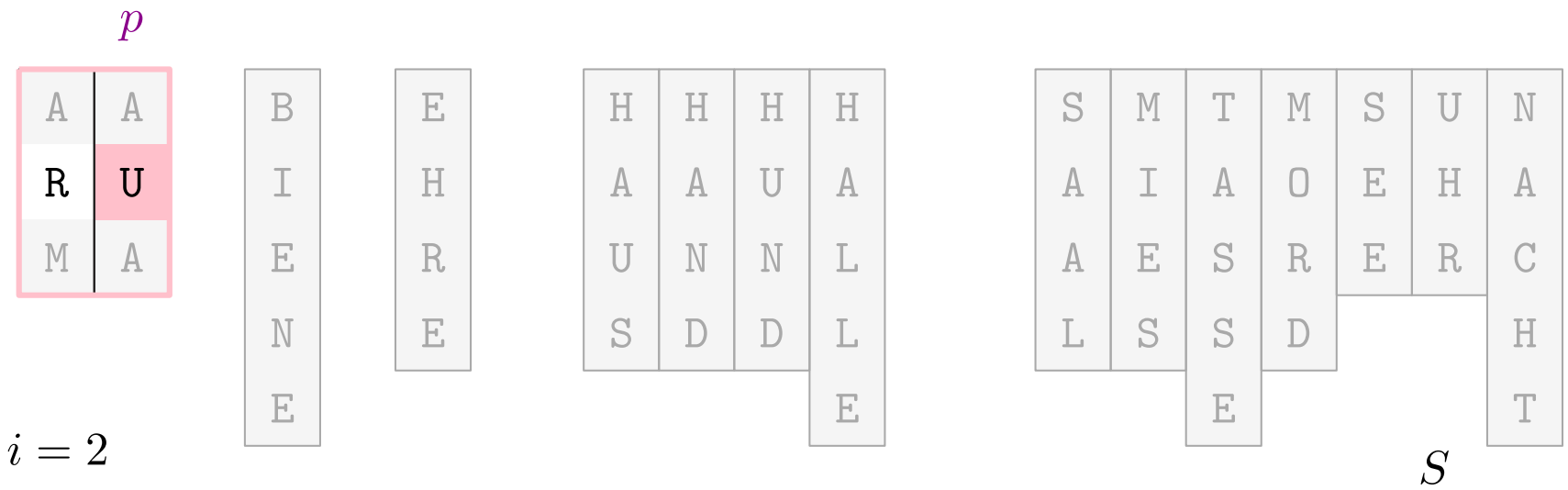
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



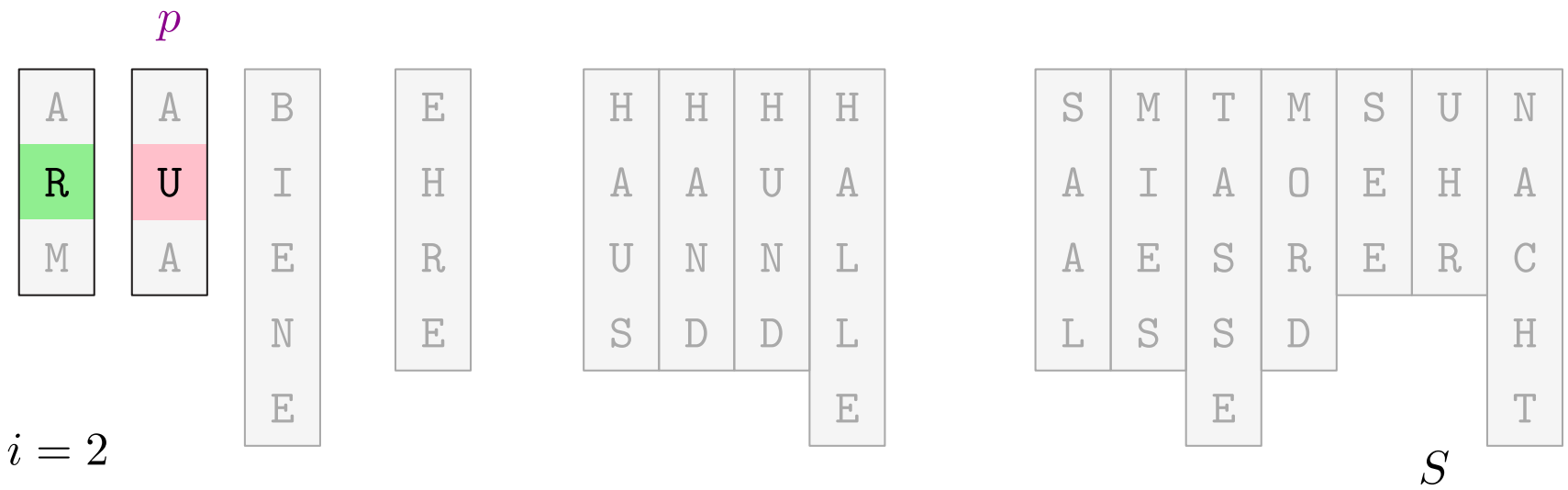
Multikey Quicksort

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

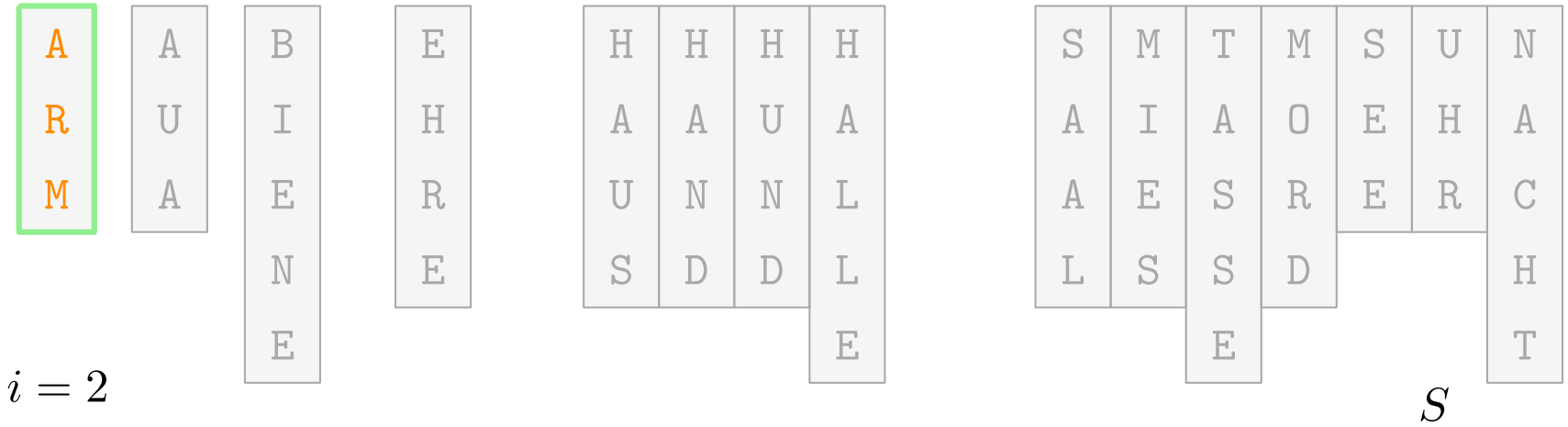
choose $p \in S$ uniformly at random — Pivotelement

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($\langle e \in S : e[i] > p[i] \rangle, i$)



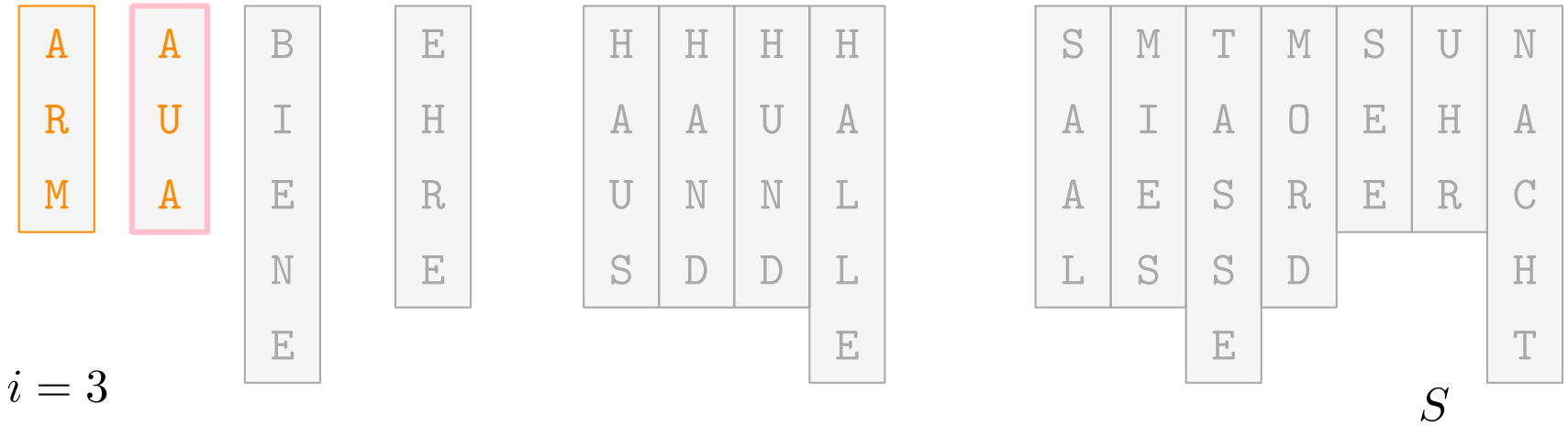
Multikey Quicksort

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

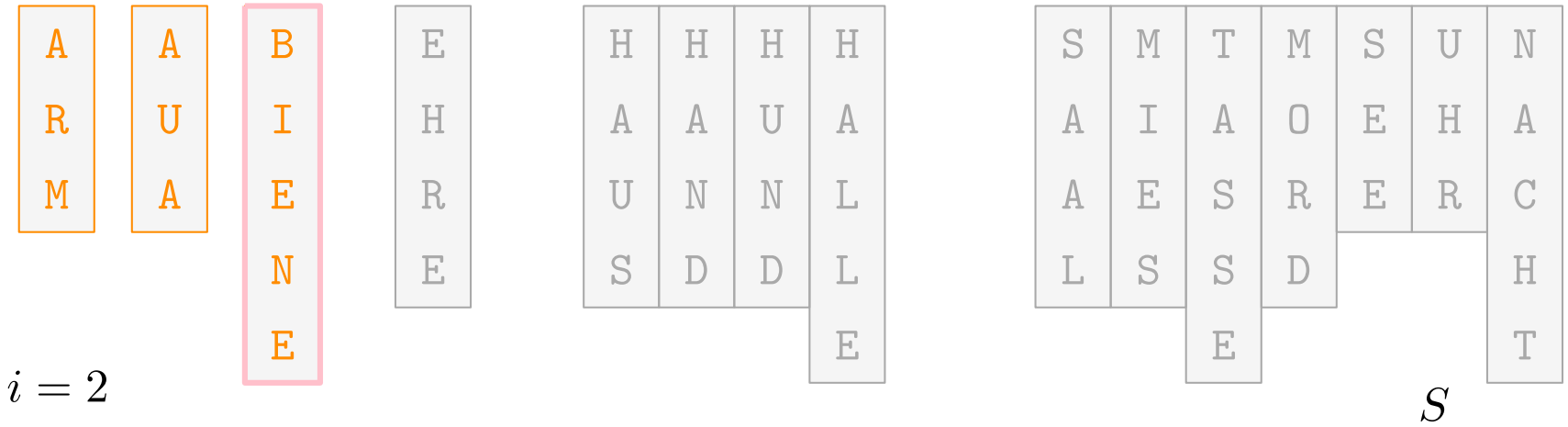
choose $p \in S$ uniformly at random — Pivotelement

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($\langle e \in S : e[i] > p[i] \rangle, i$)



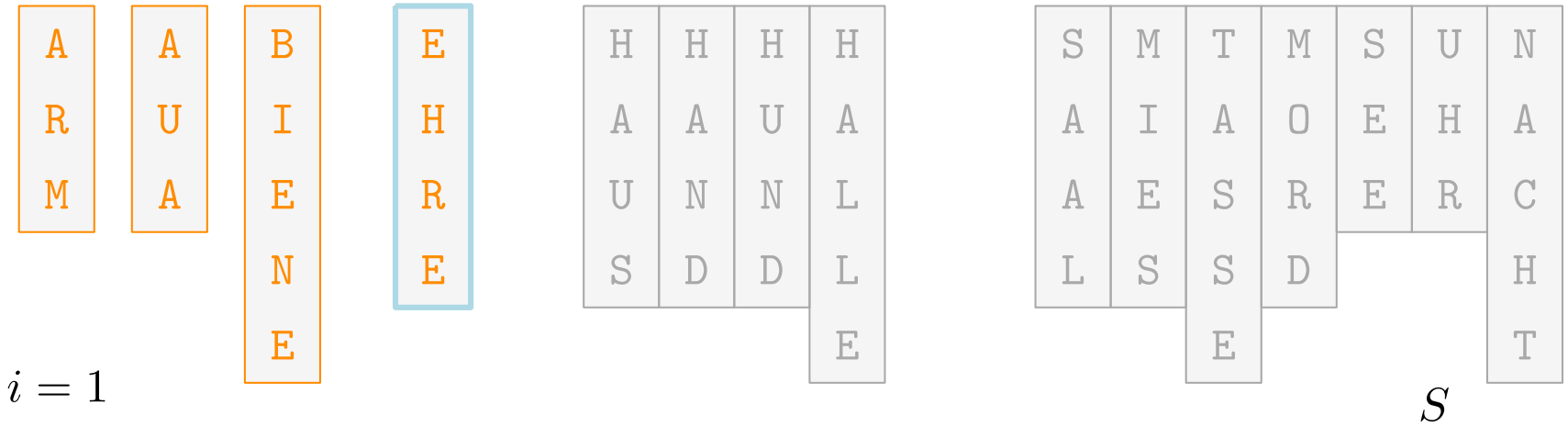
Multikey Quicksort

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

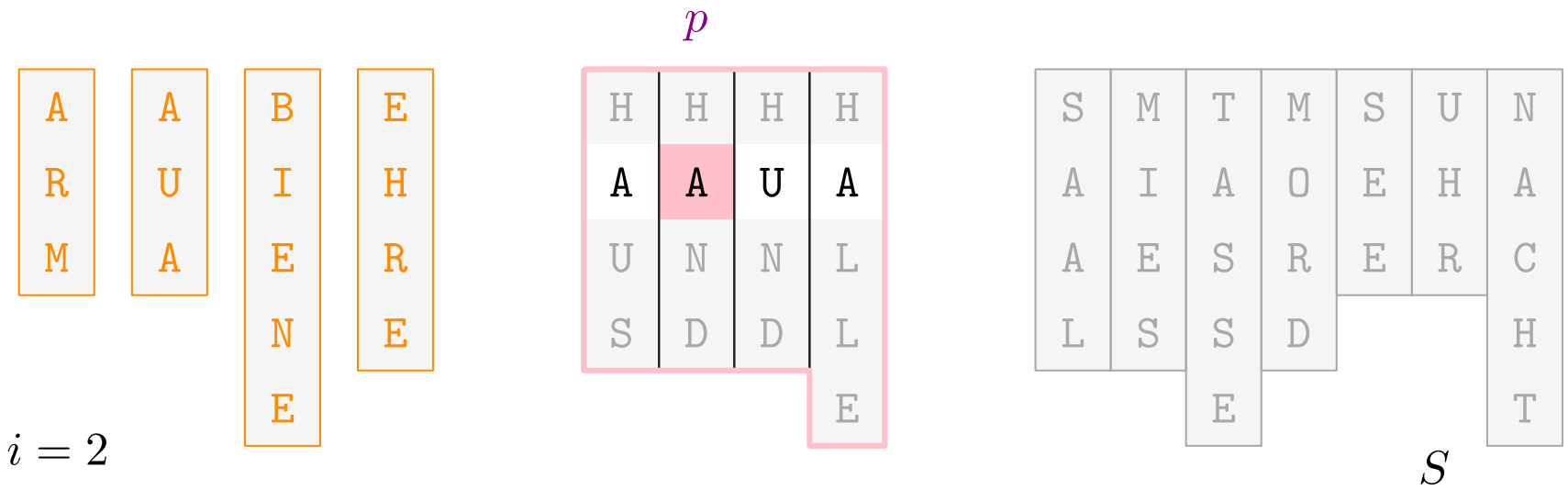
choose $p \in S$ uniformly at random — Pivotelement

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

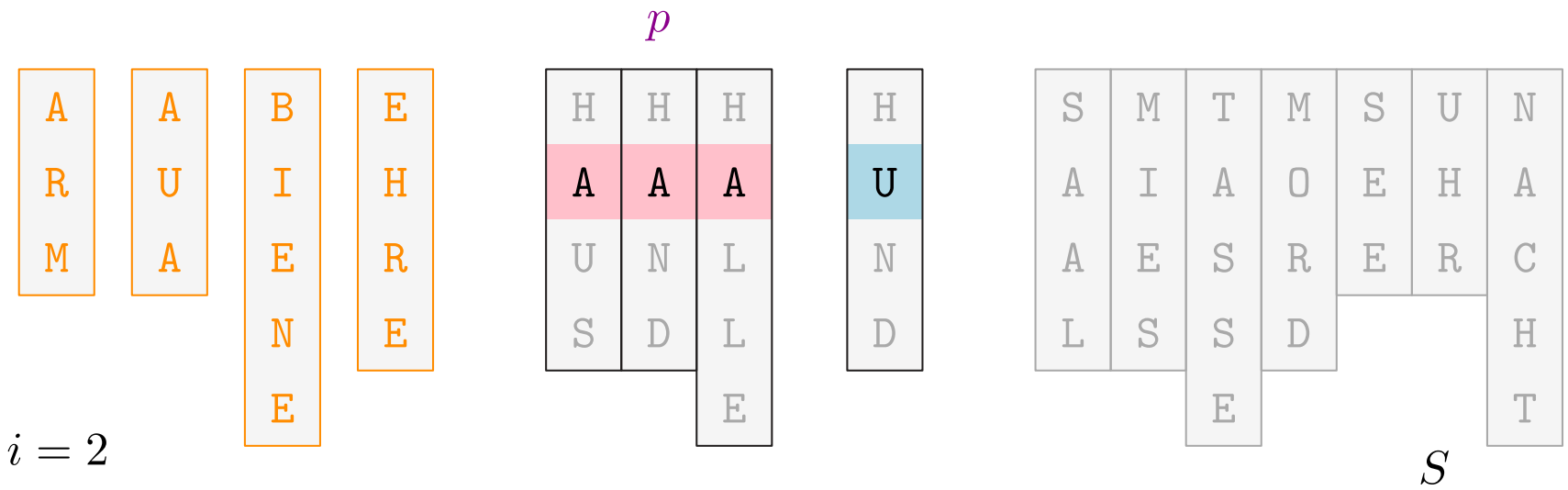
choose $p \in S$ uniformly at random — Pivotelement

return concatenation of — Rekursion

$\text{mkqSort}(\ \langle e \in S : e[i] < p[i] \rangle, i\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] = p[i] \rangle, i+1\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] > p[i] \rangle, i\ )$



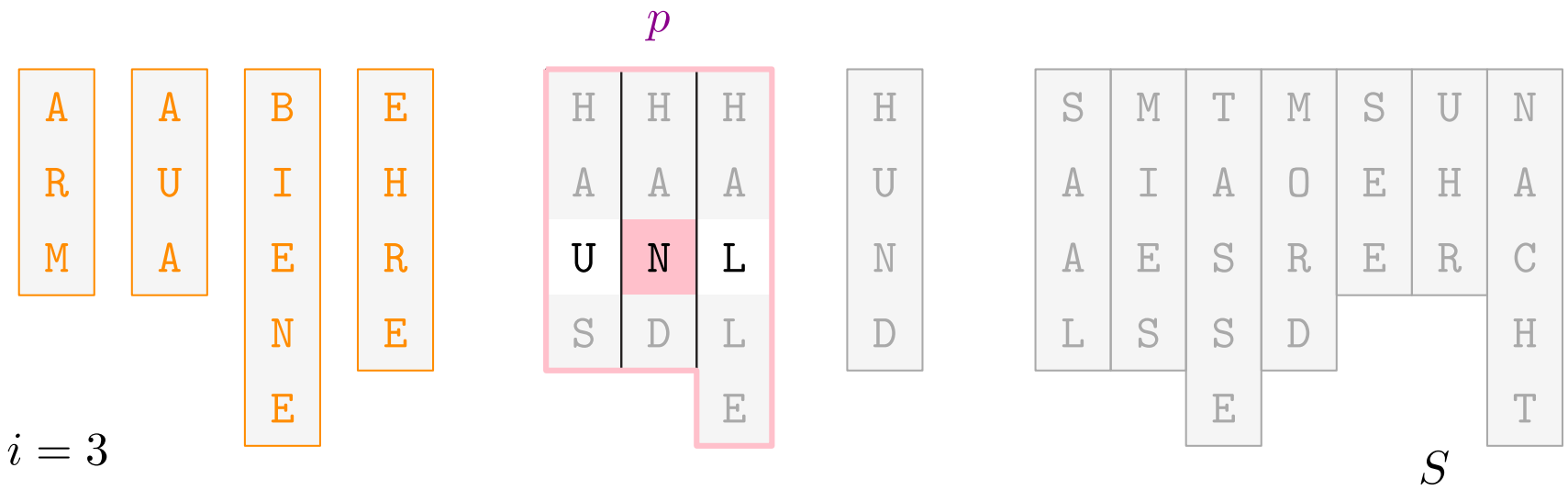
Multikey Quicksort

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

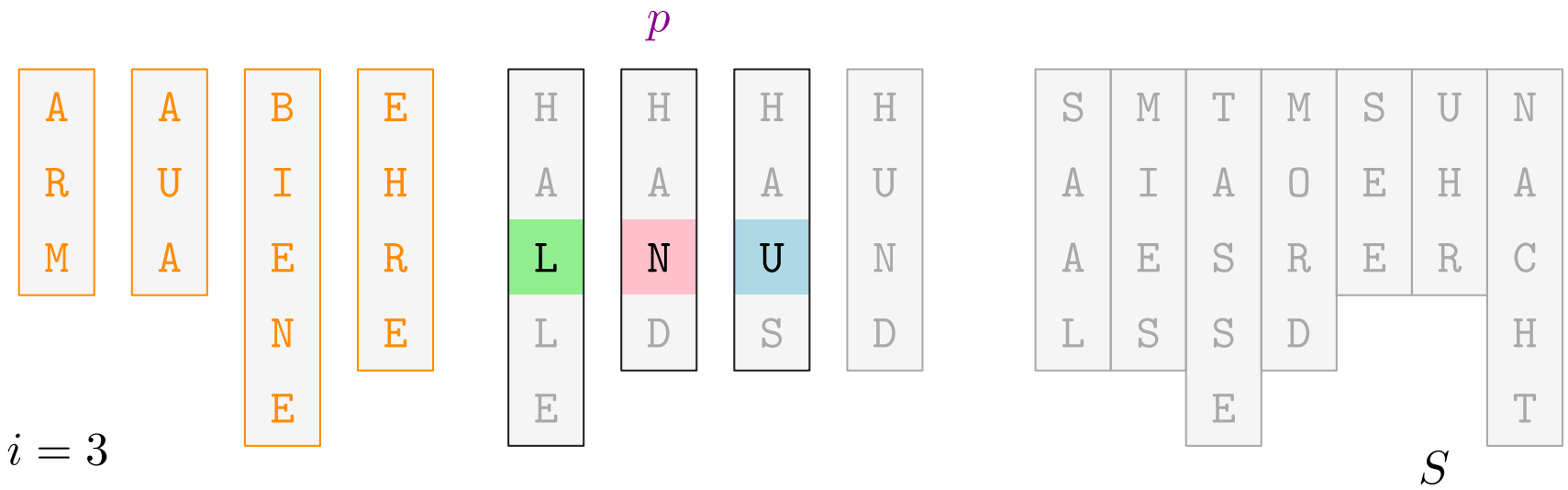
  choose $p \in S$ uniformly at random — Pivotelement

  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

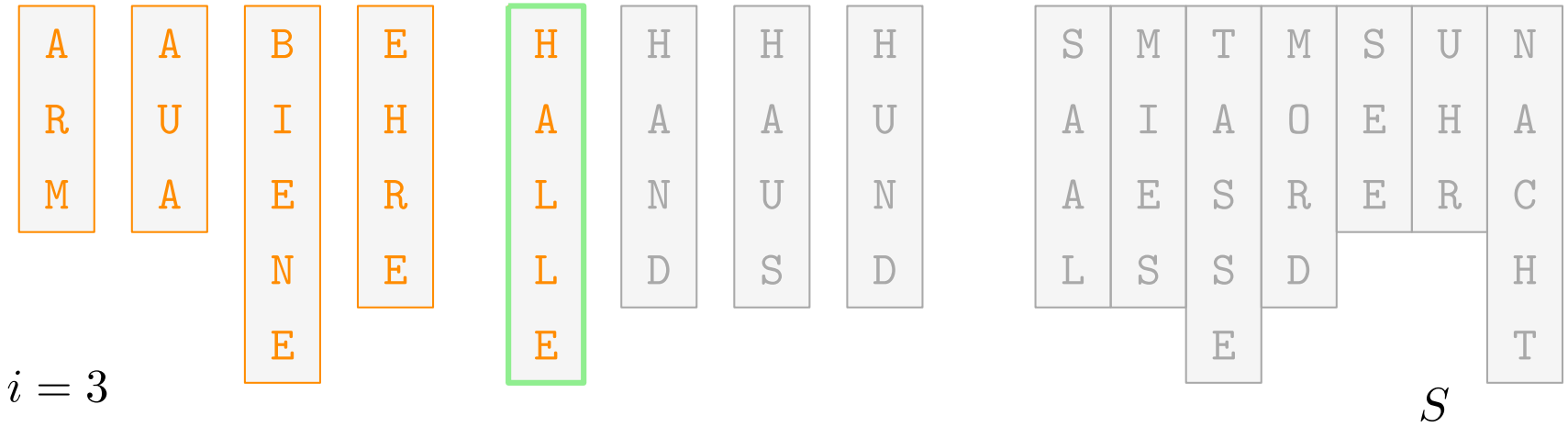
choose $p \in S$ uniformly at random — Pivotelement

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

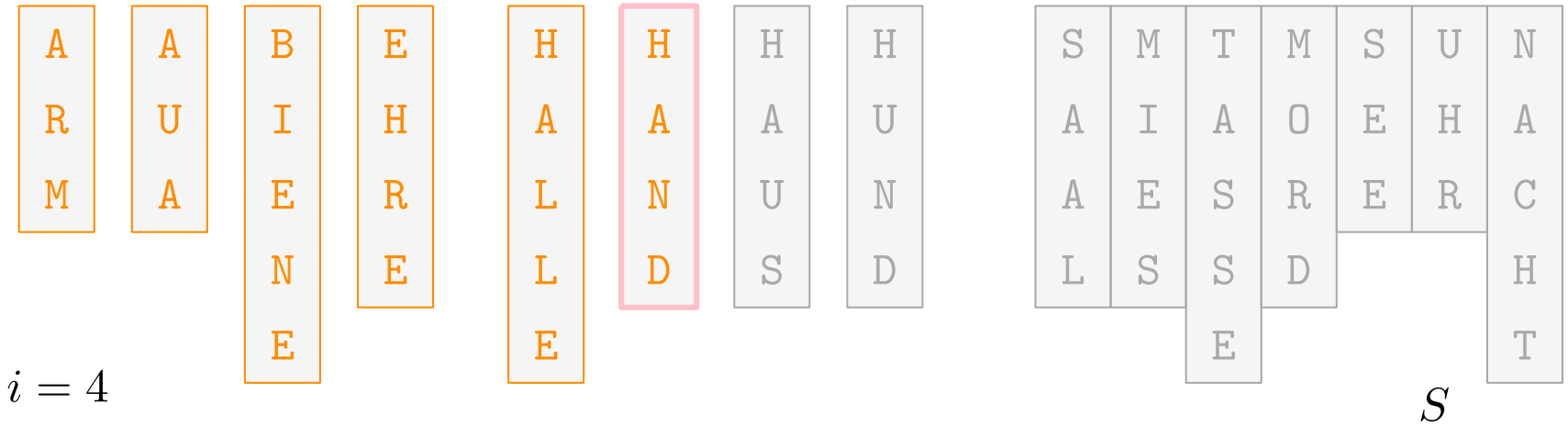
choose $p \in S$ uniformly at random — Pivotelement

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

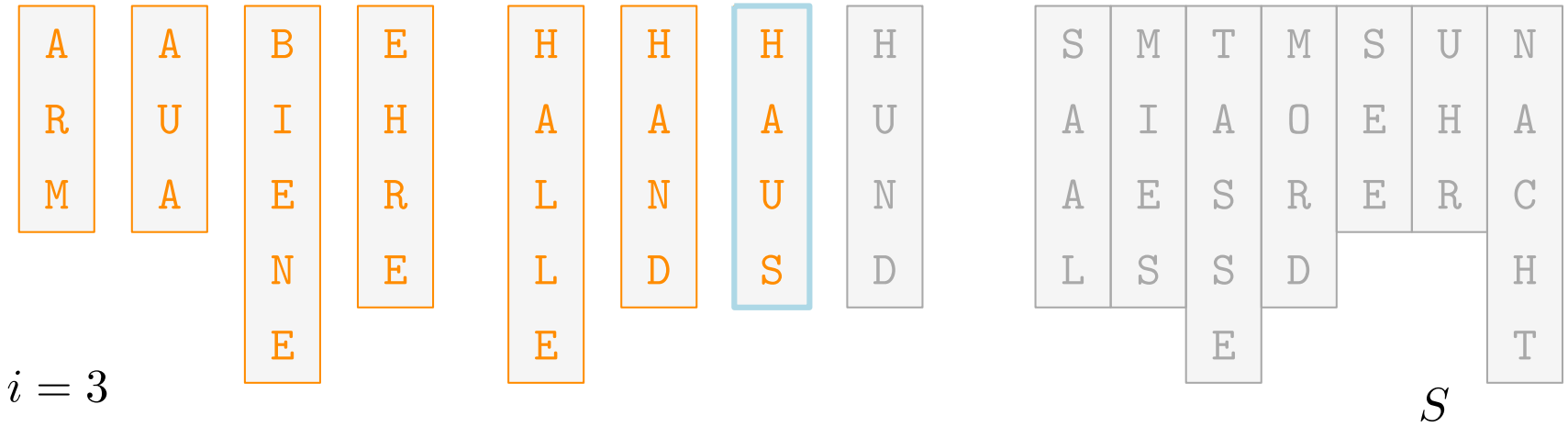
choose $p \in S$ uniformly at random — Pivotelement

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($\langle e \in S : e[i] > p[i] \rangle, i$)



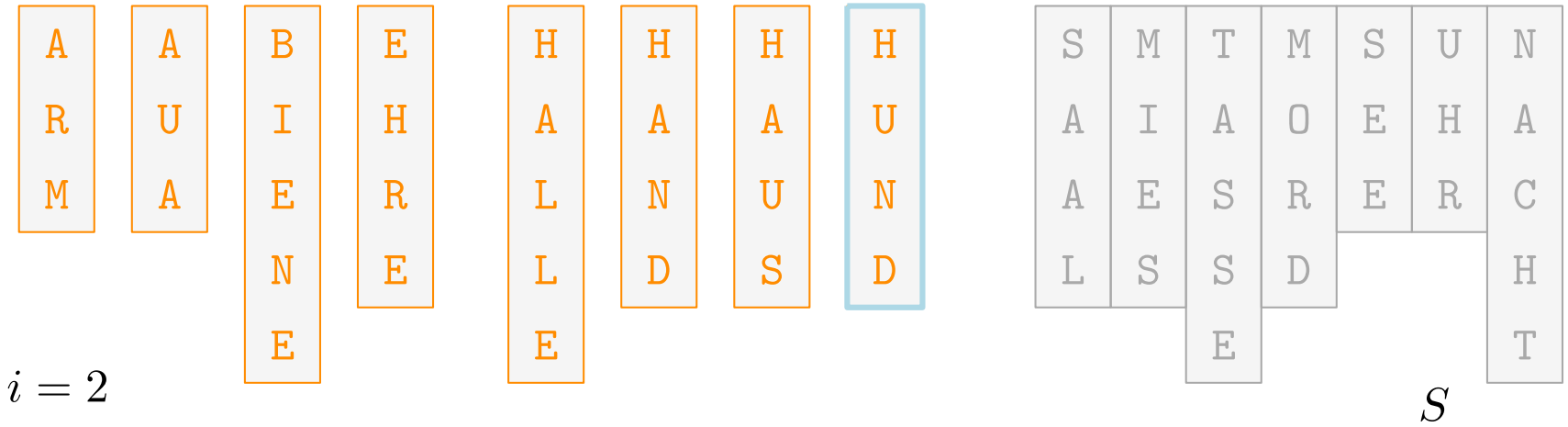
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



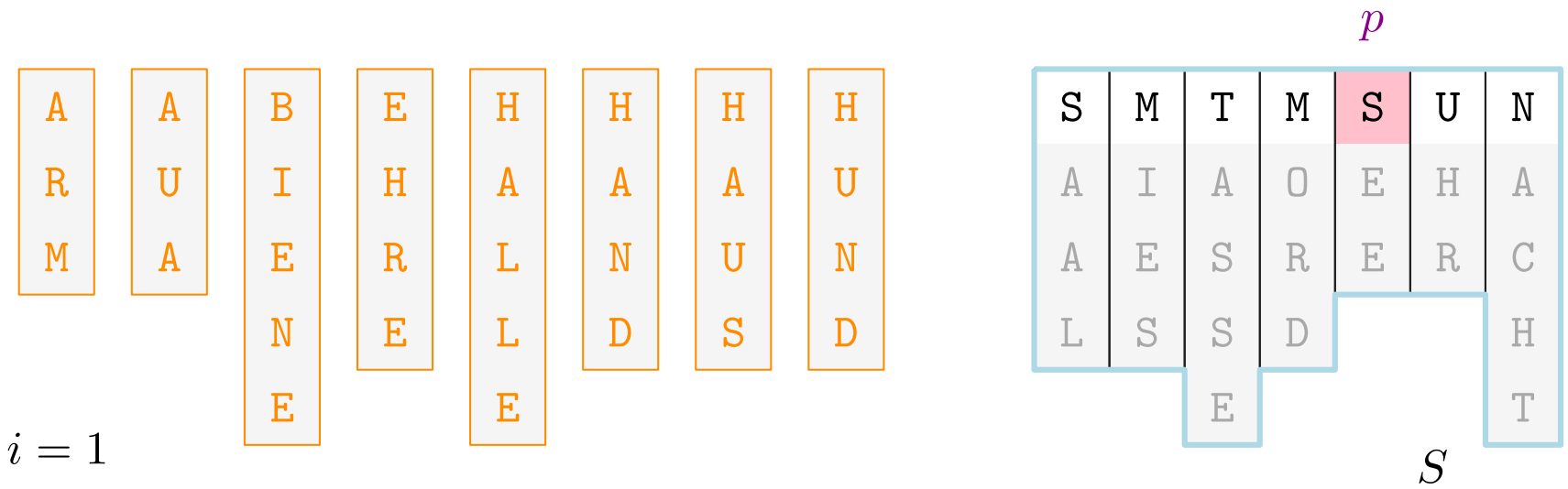
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



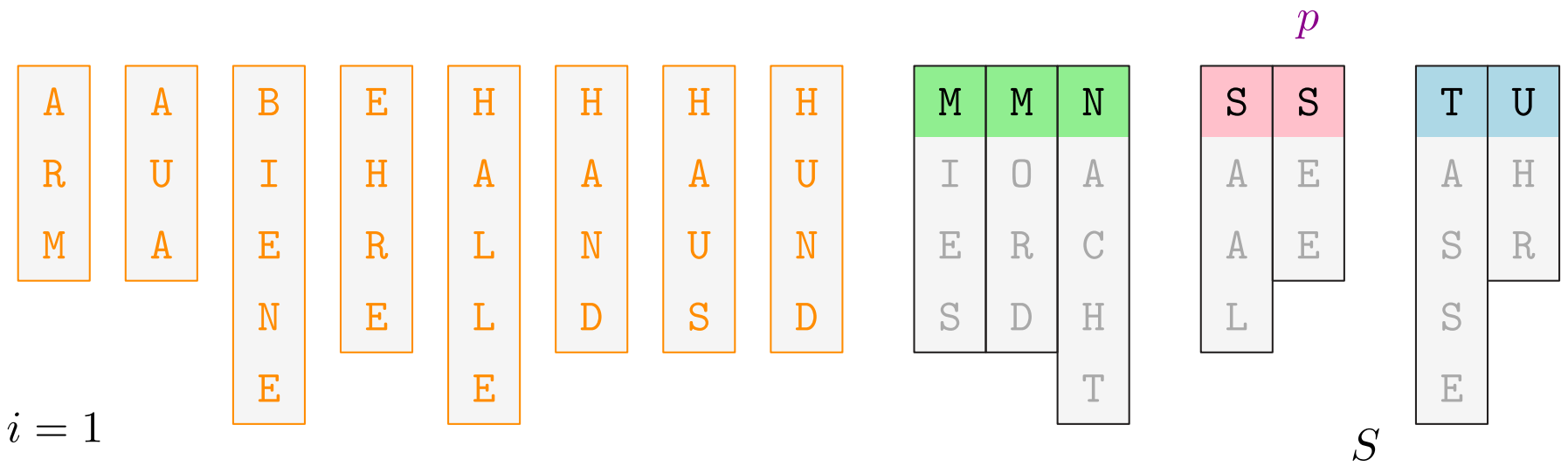
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



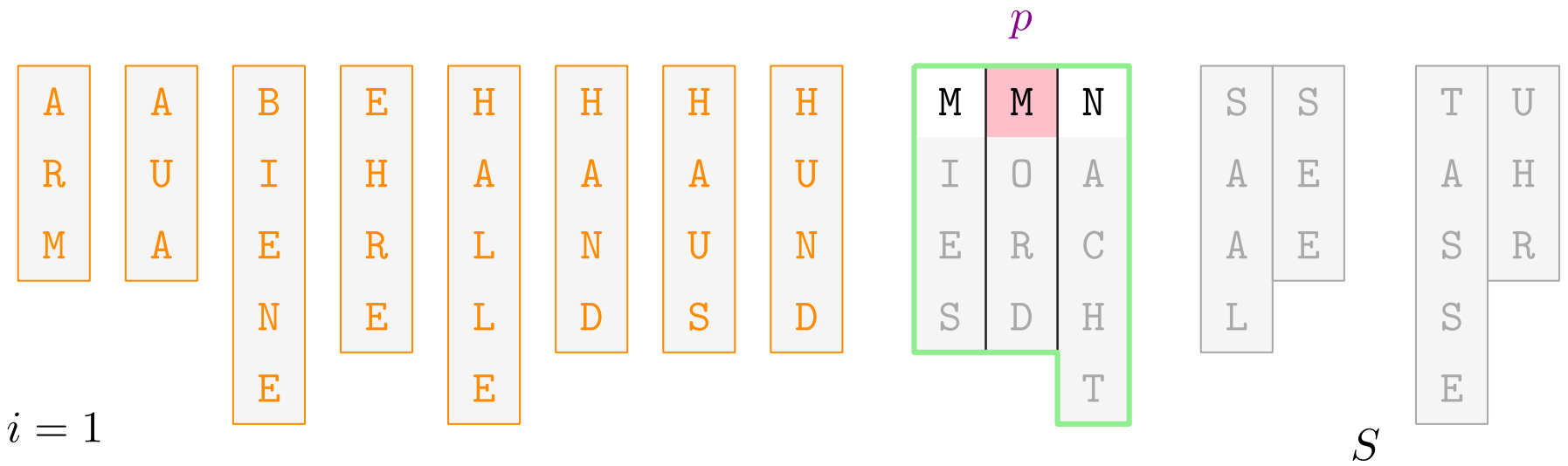
Multikey Quicksort

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

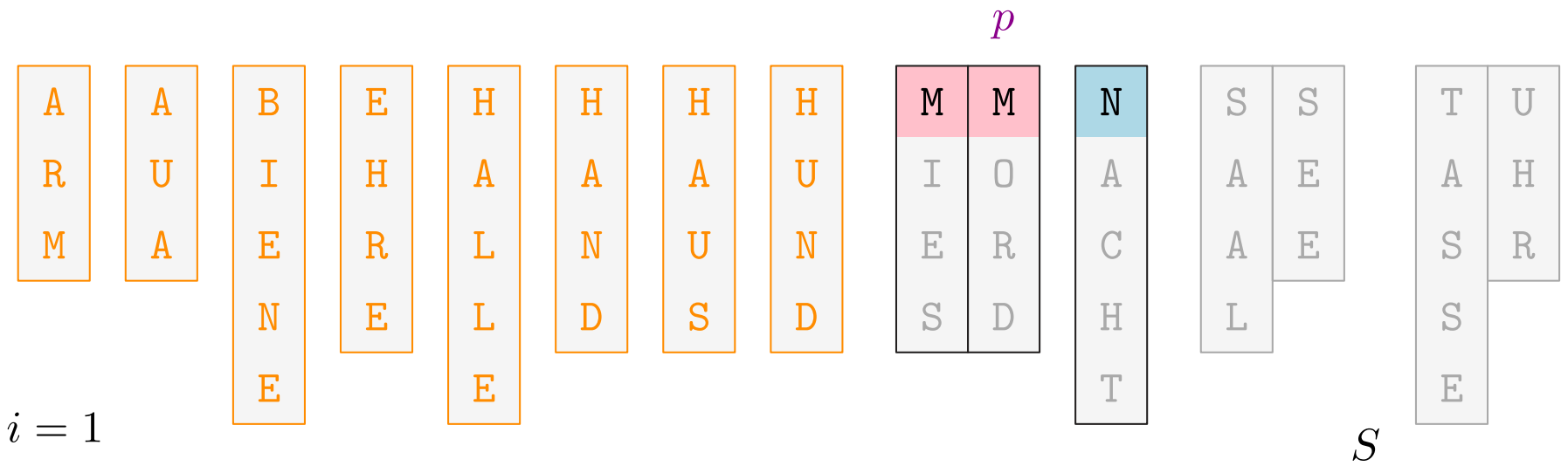
choose $p \in S$ uniformly at random — Pivotelement

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

Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S — Basisfall

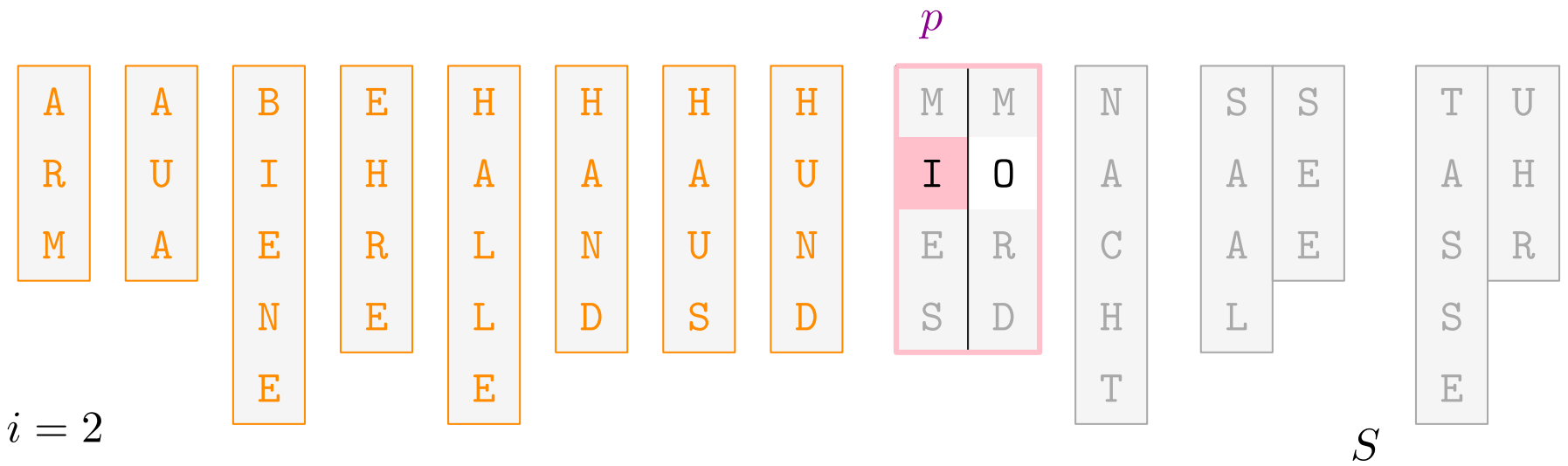
choose $p \in S$ uniformly at random — Pivotelement

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

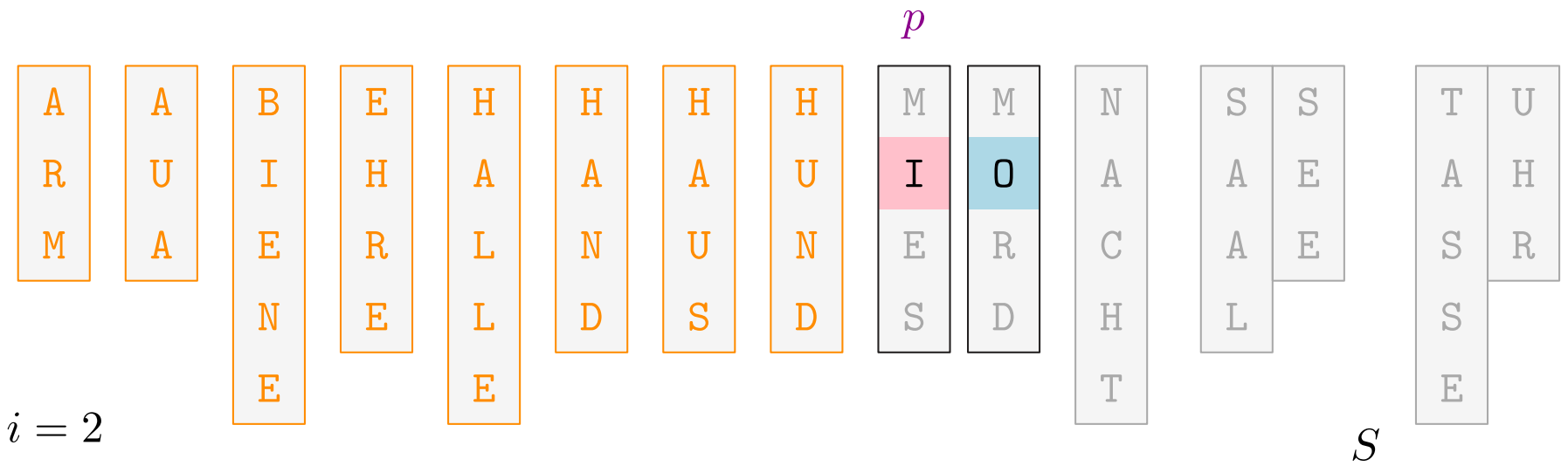
choose $p \in S$ uniformly at random — Pivotelement

return concatenation of — Rekursion

$\text{mkqSort}(\ \langle e \in S : e[i] < p[i] \rangle, i\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] = p[i] \rangle, i+1\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] > p[i] \rangle, i\ )$



Multikey Quicksort

Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S — Basisfall

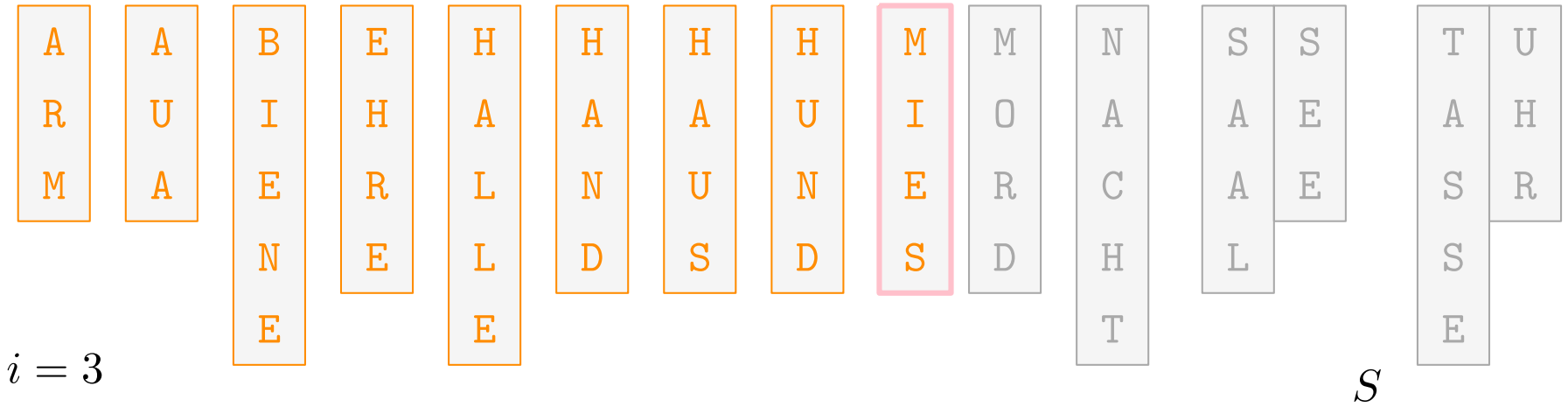
choose $p \in S$ uniformly at random — Pivotelement

return concatenation of — Rekursion

$\text{mkqSort}(\ \langle e \in S : e[i] < p[i] \rangle, i\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] = p[i] \rangle, i+1\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] > p[i] \rangle, i\ )$



Multikey Quicksort

Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S — Basisfall

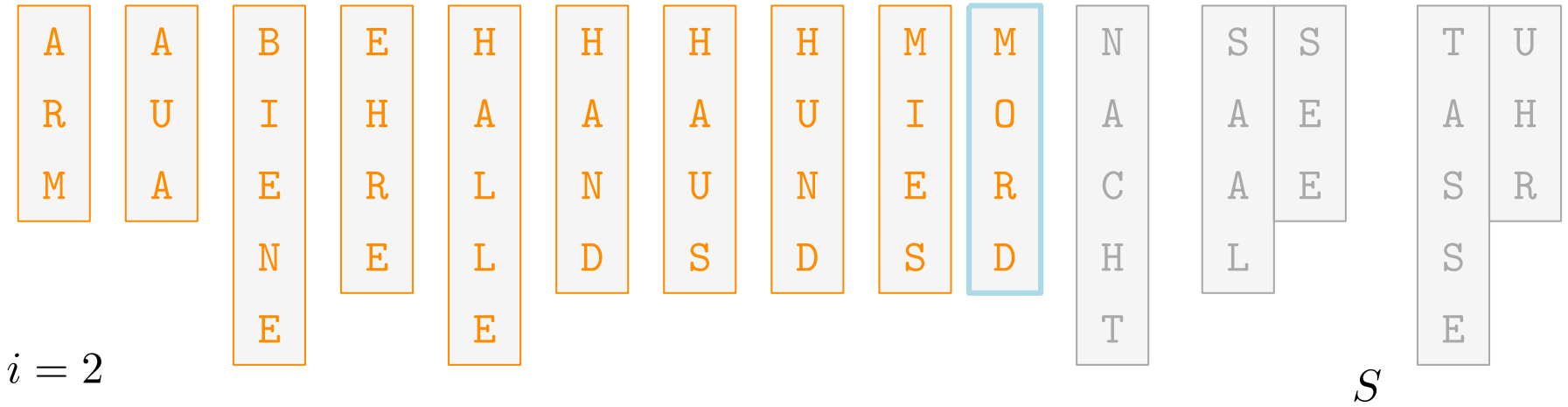
choose $p \in S$ uniformly at random — Pivotelement

return concatenation of — Rekursion

$\text{mkqSort}(\ \langle e \in S : e[i] < p[i] \rangle, i\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] = p[i] \rangle, i+1\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] > p[i] \rangle, i\ )$



Multikey Quicksort

Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S — Basisfall

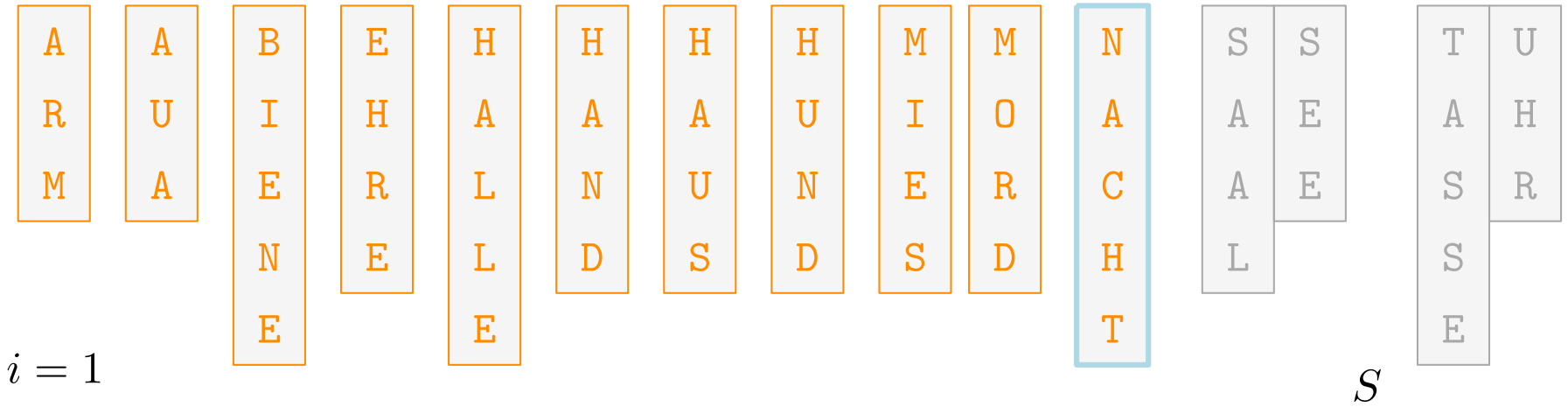
choose $p \in S$ uniformly at random — Pivotelement

return concatenation of — Rekursion

$\text{mkqSort}(\ \langle e \in S : e[i] < p[i] \rangle, i\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] = p[i] \rangle, i+1\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] > p[i] \rangle, i\ )$



Multikey Quicksort

Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S — Basisfall

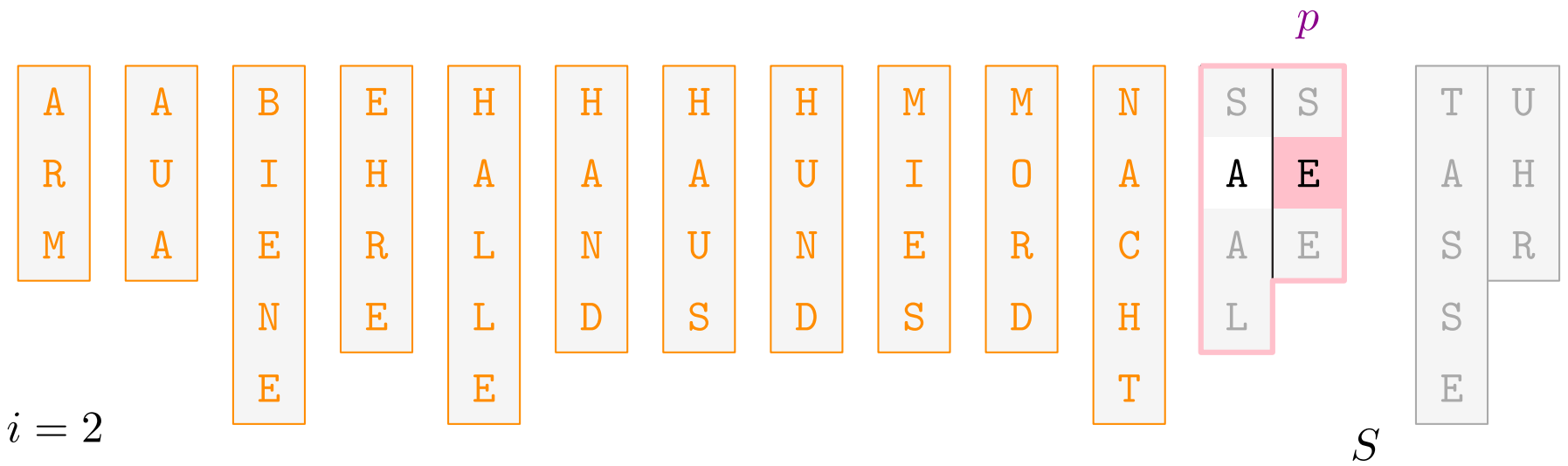
choose $p \in S$ uniformly at random — Pivotelement

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

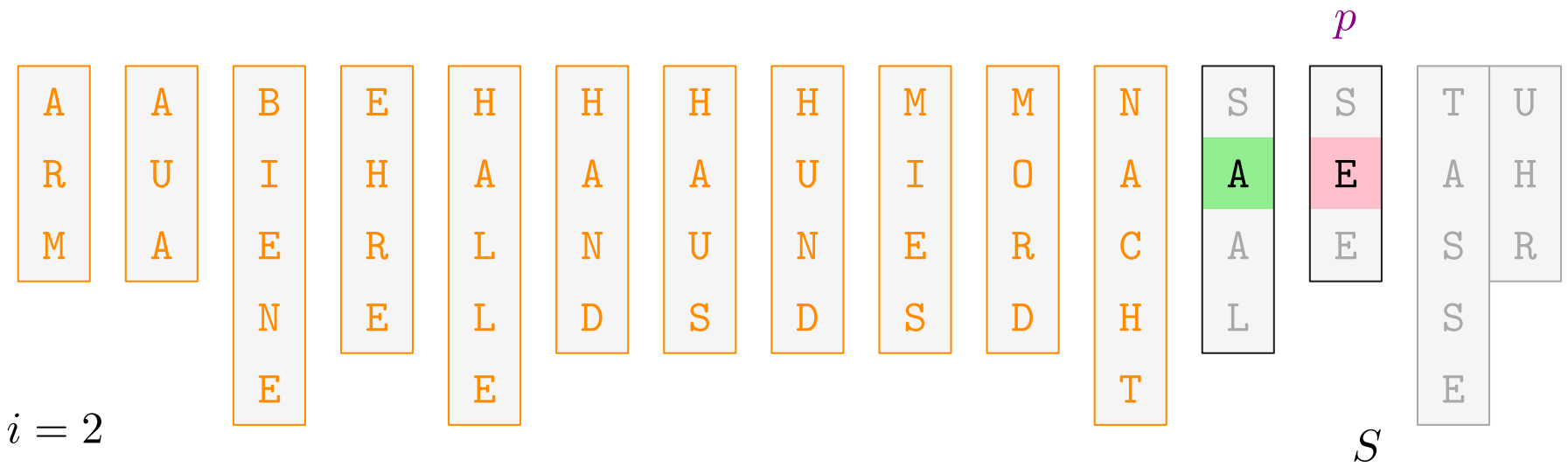
choose $p \in S$ uniformly at random — Pivotelement

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

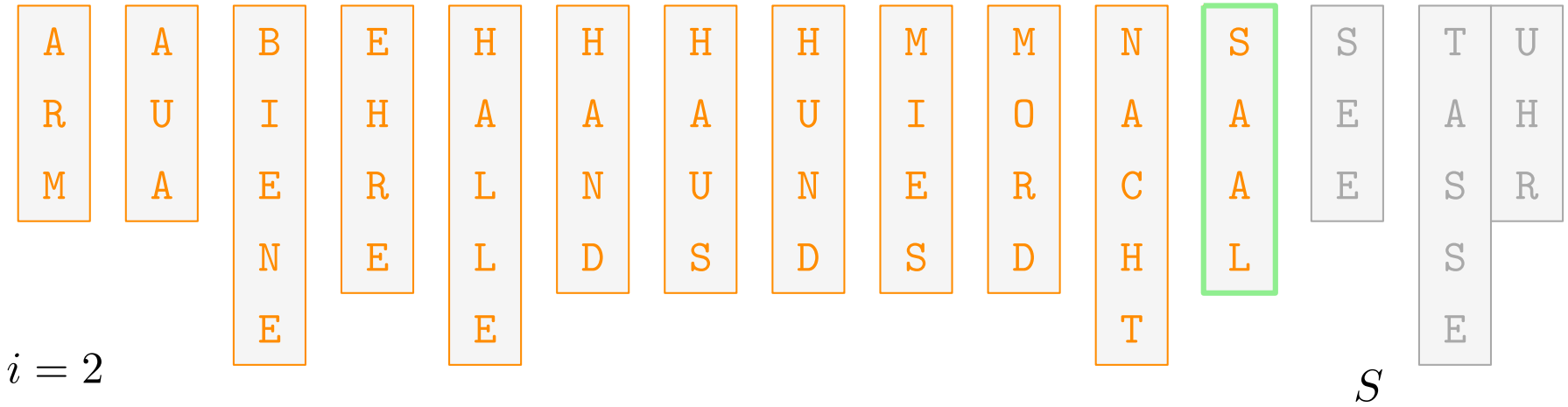
choose $p \in S$ uniformly at random — Pivotelement

return concatenation of — Rekursion

$\text{mkqSort}(\ \langle e \in S : e[i] < p[i] \rangle, i\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] = p[i] \rangle, i+1\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] > p[i] \rangle, i\ )$



Multikey Quicksort

Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String

  if $|S| \leq 1$ **then return** S — Basisfall

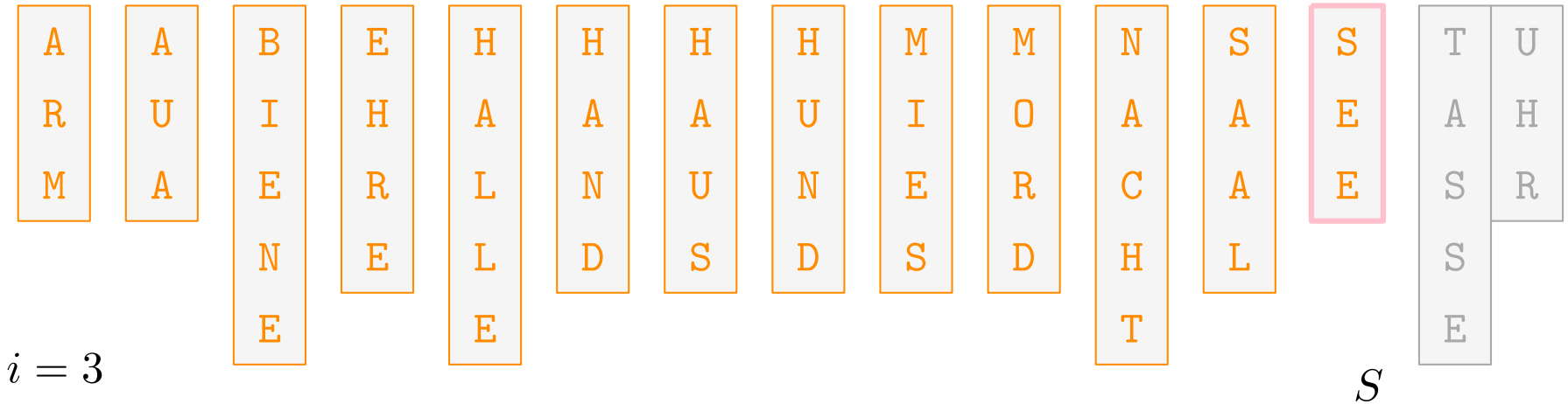
  choose $p \in S$ uniformly at random — Pivotelement

  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

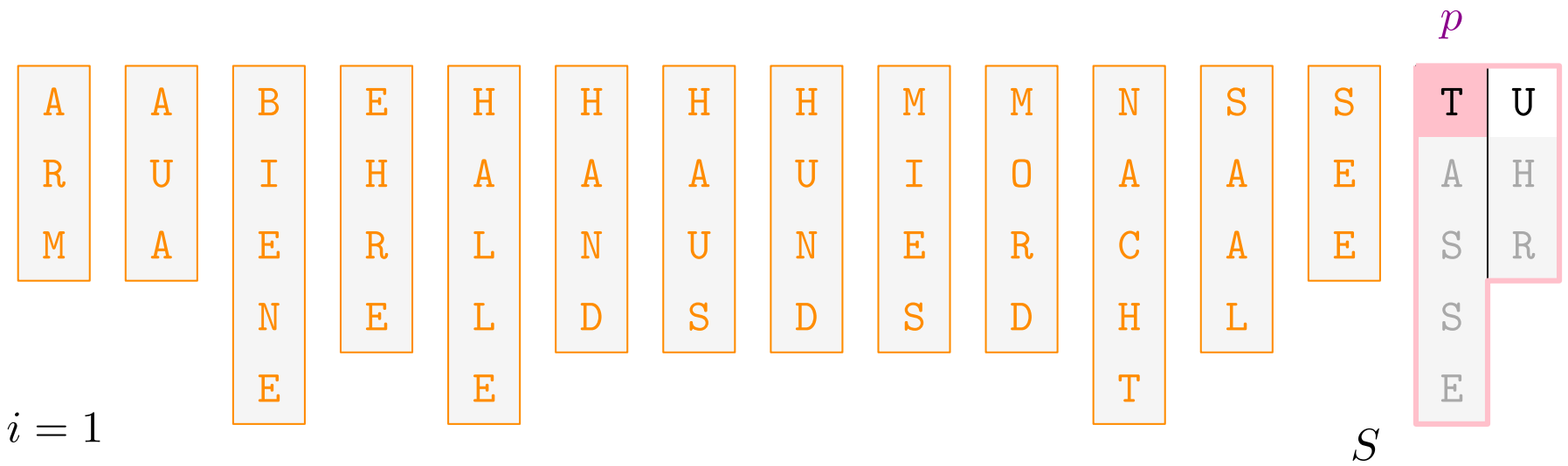
choose $p \in S$ uniformly at random — Pivotelement

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($\langle e \in S : e[i] > p[i] \rangle, i$)



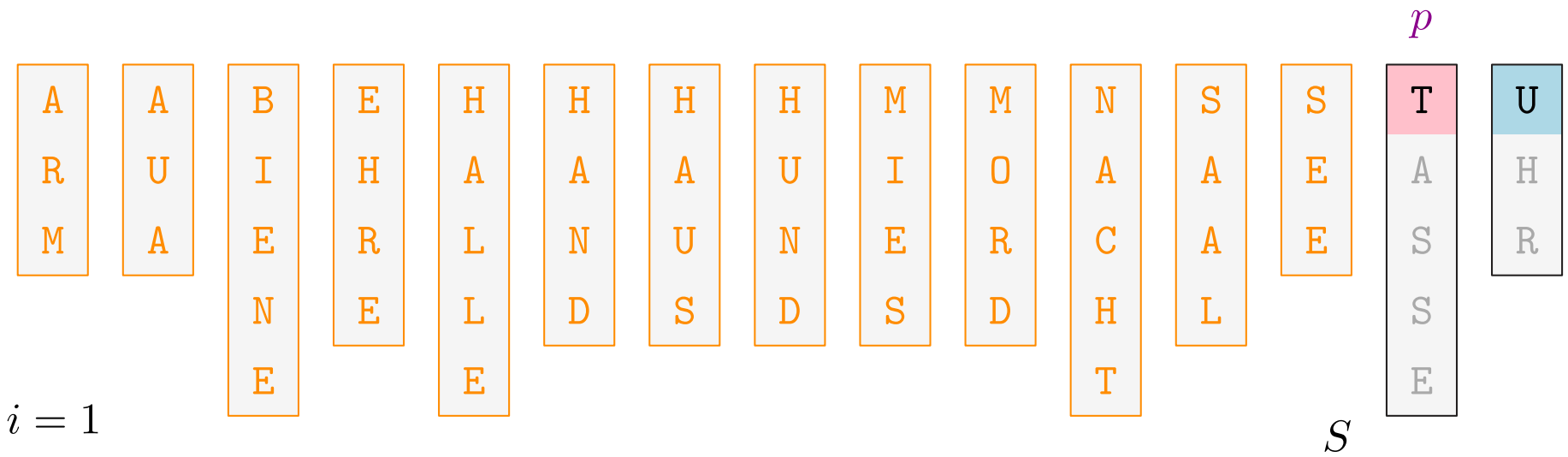
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



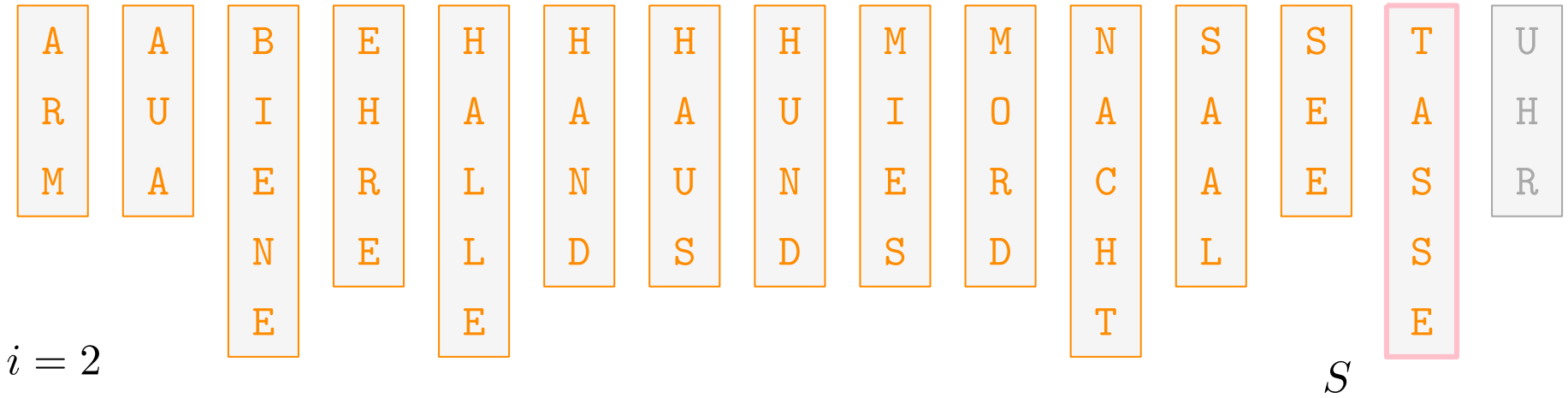
Multikey Quicksort

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($\langle e \in S : e[i] > p[i] \rangle, i$)



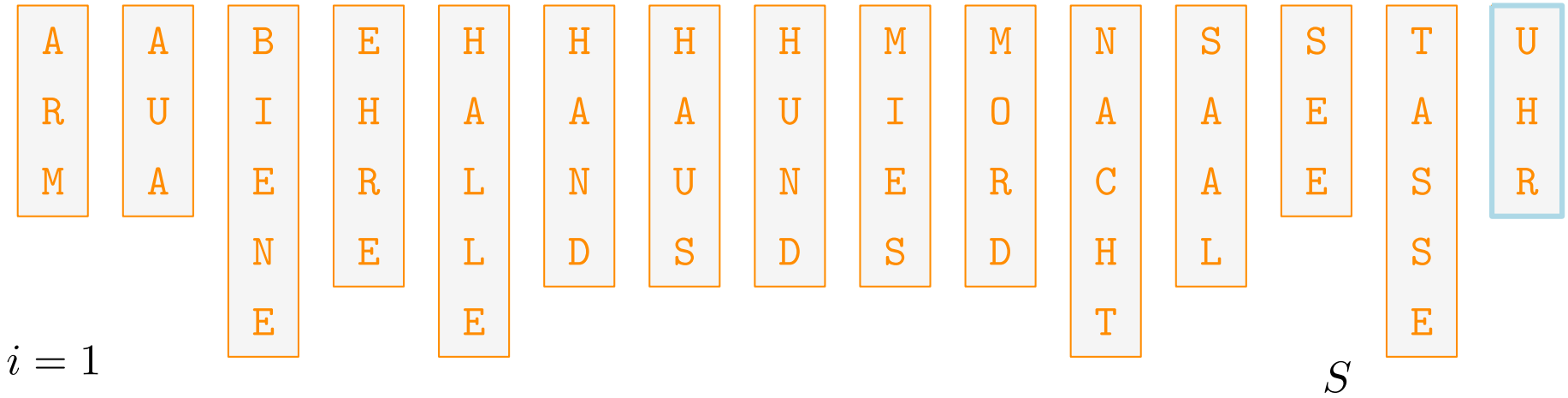
Multikey Quicksort

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

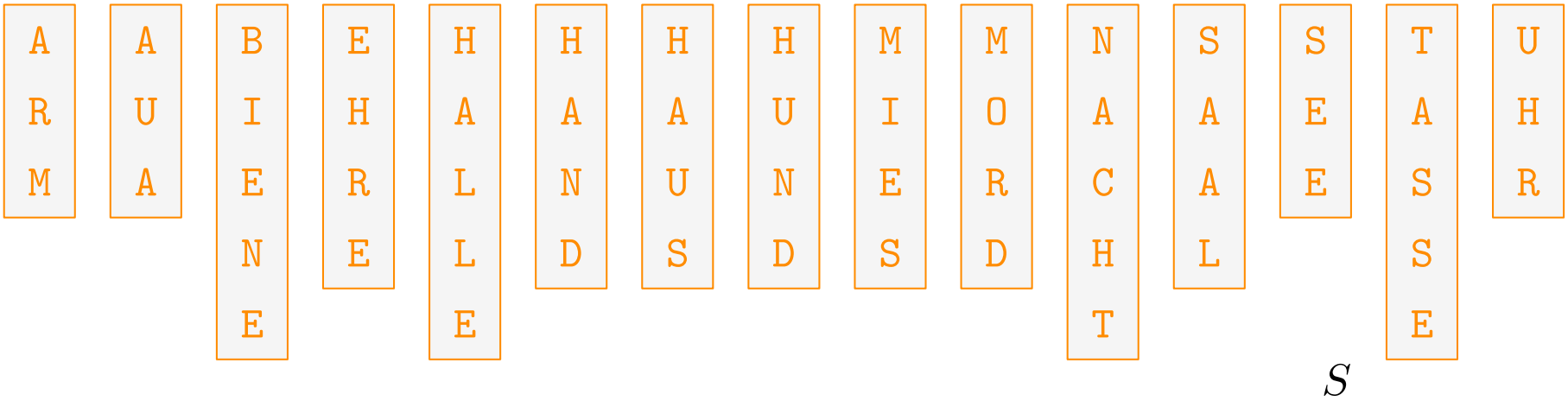
choose $p \in S$ uniformly at random — Pivotelement

return concatenation of — Rekursion

$\text{mkqSort}(\ \langle e \in S : e[i] < p[i] \rangle, i\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] = p[i] \rangle, i+1\ ),$

$\text{mkqSort}(\ \langle e \in S : e[i] > p[i] \rangle, i\ )$



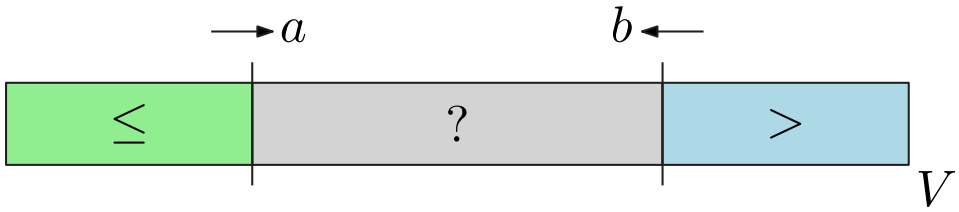
in-place Multikey Quicksort

Wiederholung: Partitionierung

***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"
  $\rightarrow$ 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 $a > b$



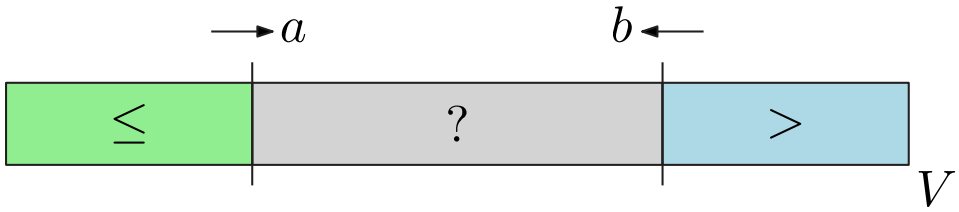
in-place Multikey Quicksort

Wiederholung: Partitionierung

***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"
  $\rightarrow$ 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

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

10	17	7	41	17	43	25	33	78	14	29

1 n V

in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

p

10	17	7	41	17	43	25	33	78	14	29

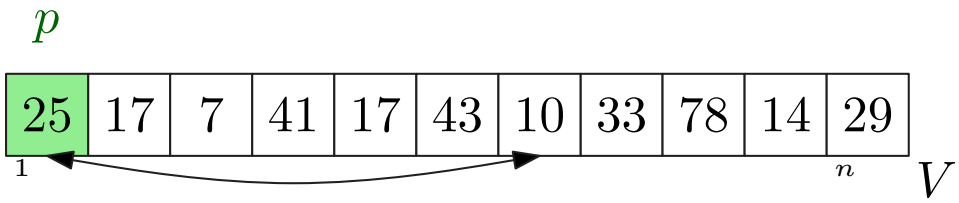
1 n V

in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

p	a									b
25	17	7	41	17	43	10	33	78	14	29
1										n

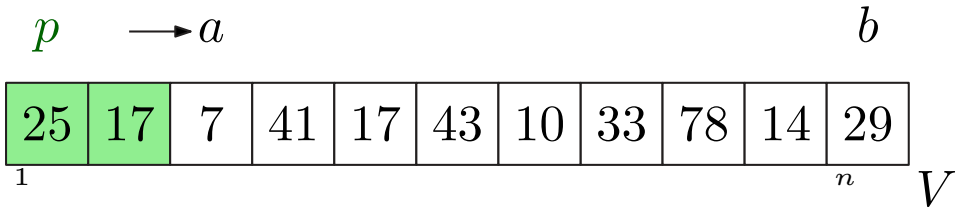
V

in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

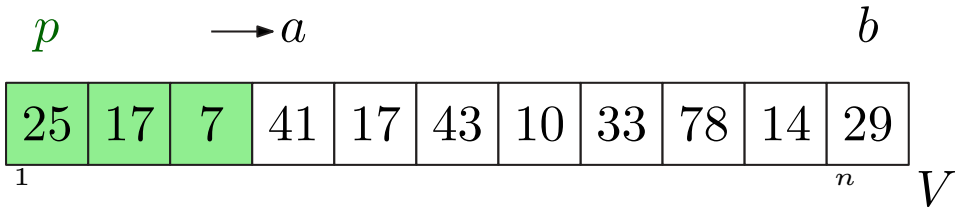


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

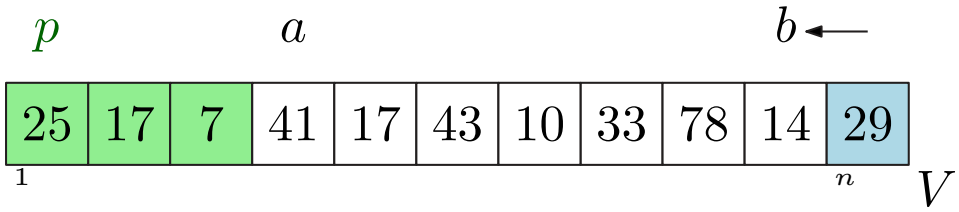


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

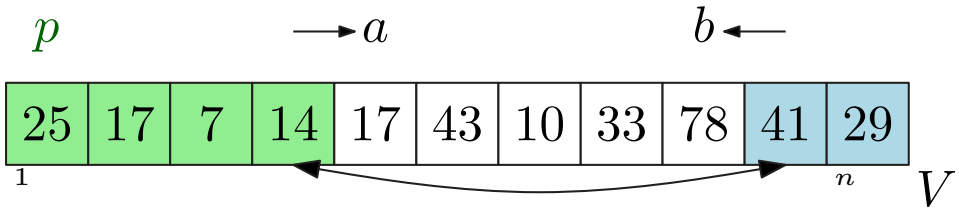


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

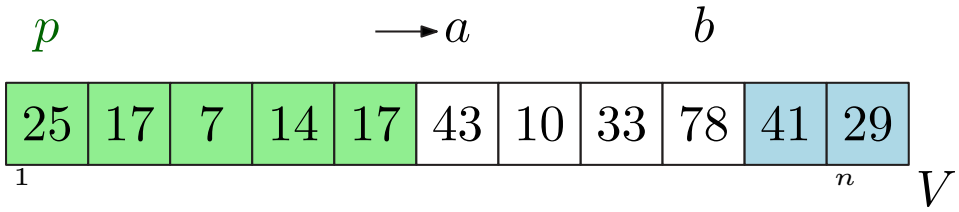


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

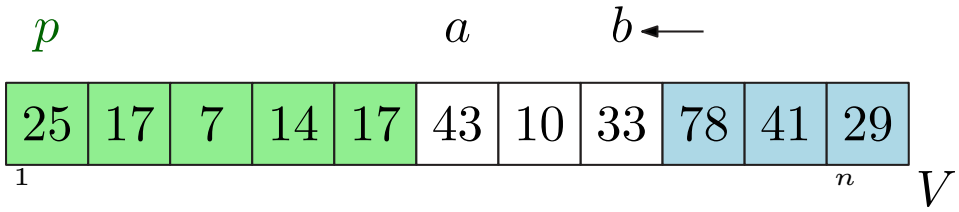


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

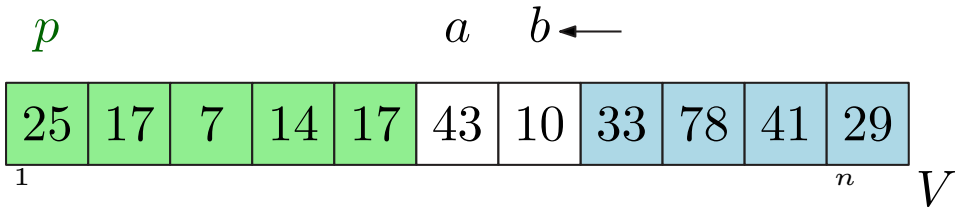


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

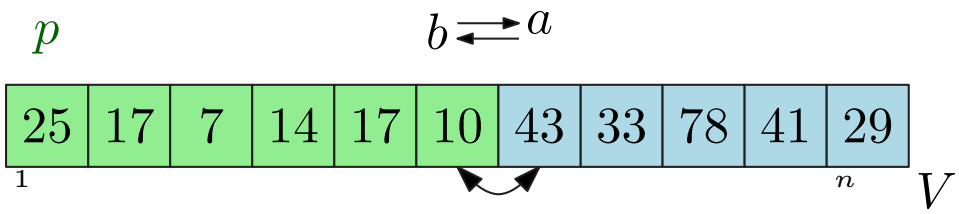


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$

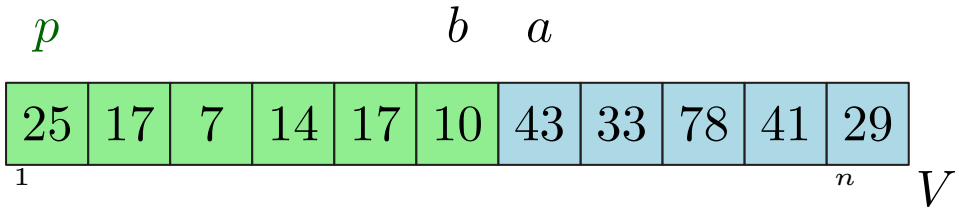


in-place Multikey Quicksort

Wiederholung: Partitionierung

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"
  $\rightarrow$ 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 $a > b$



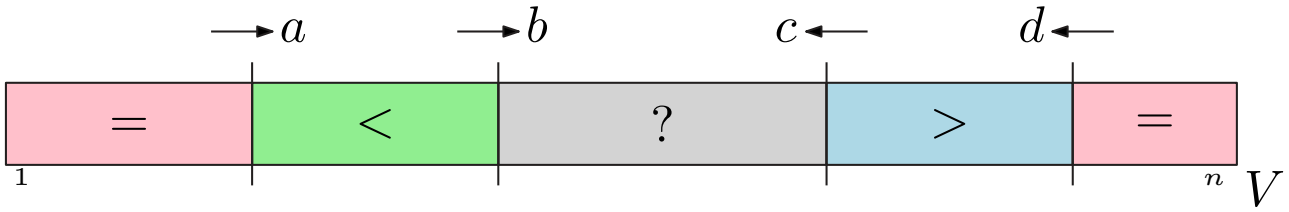
in-place Multikey Quicksort

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

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

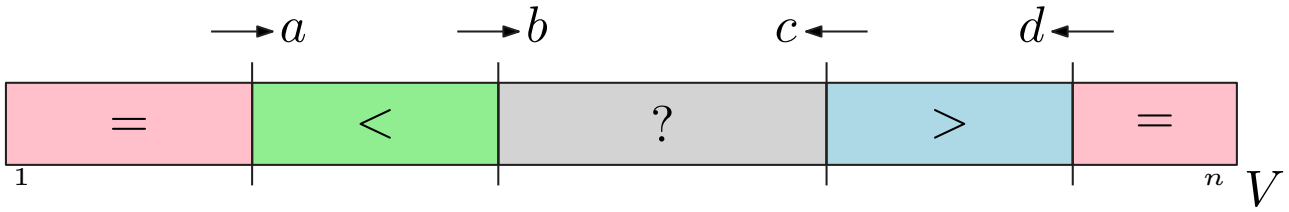


in-place Multikey Quicksort

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

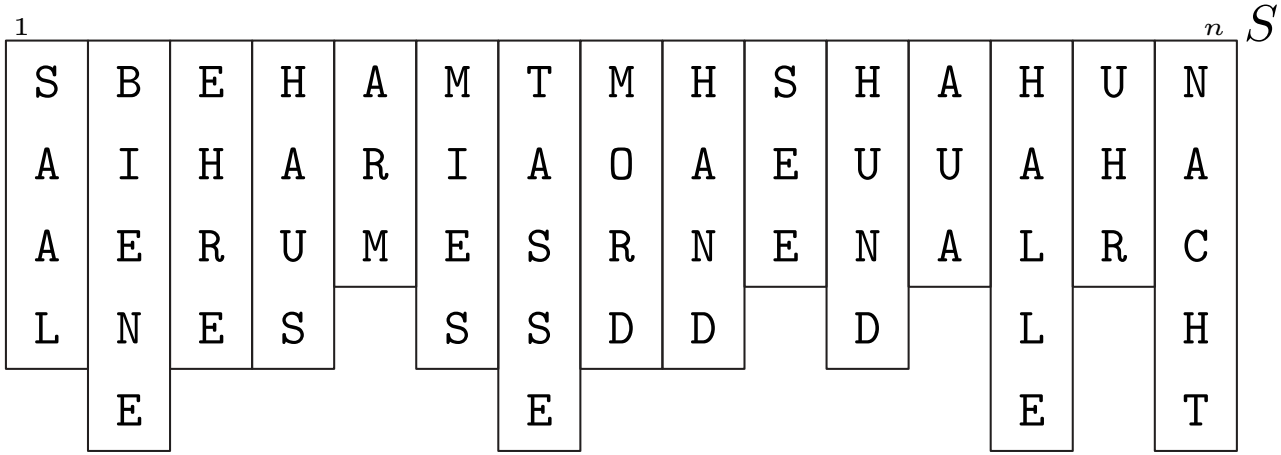


in-place Multikey Quicksort

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

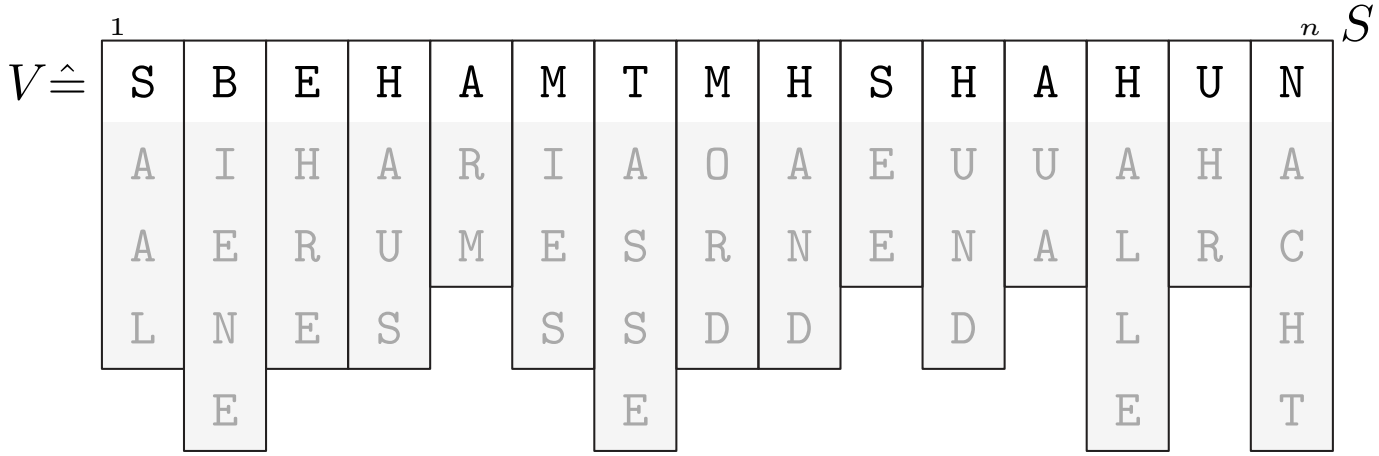


in-place Multikey Quicksort

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

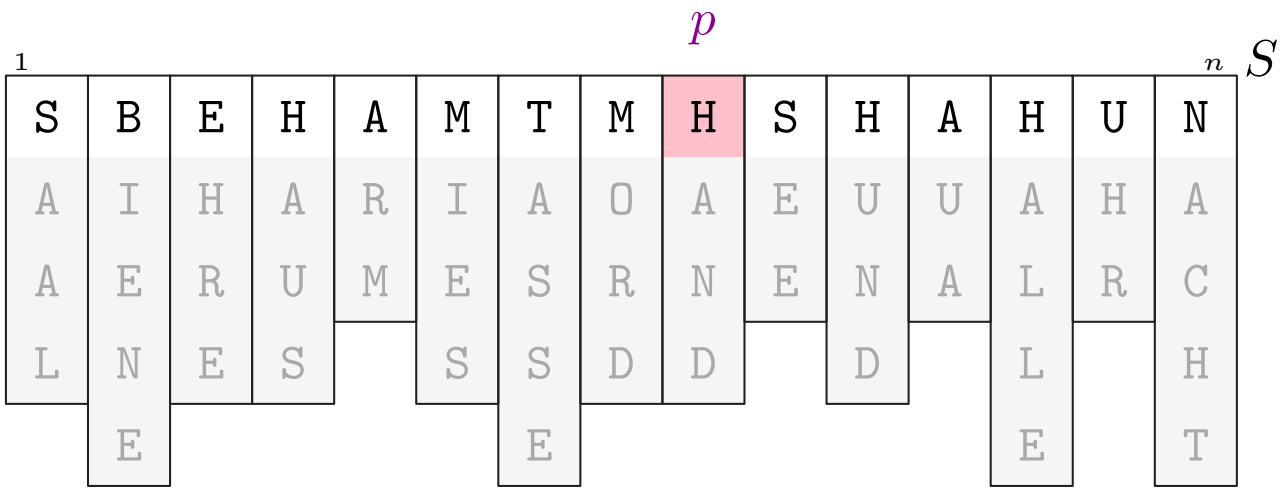


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort 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 $b > c$

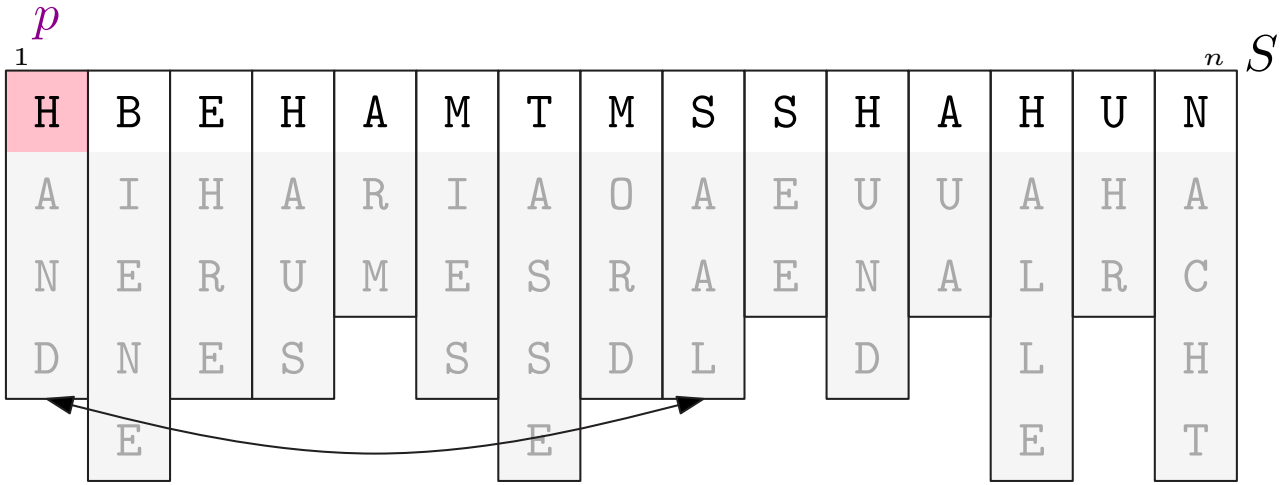


in-place Multikey Quicksort

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 $b > c$

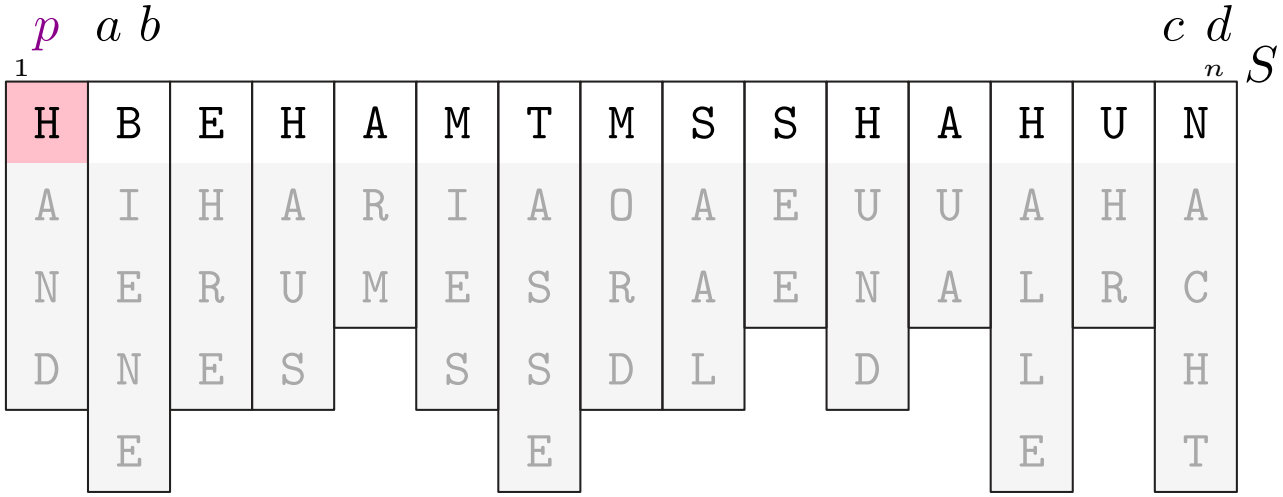


in-place Multikey Quicksort

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 $b > c$

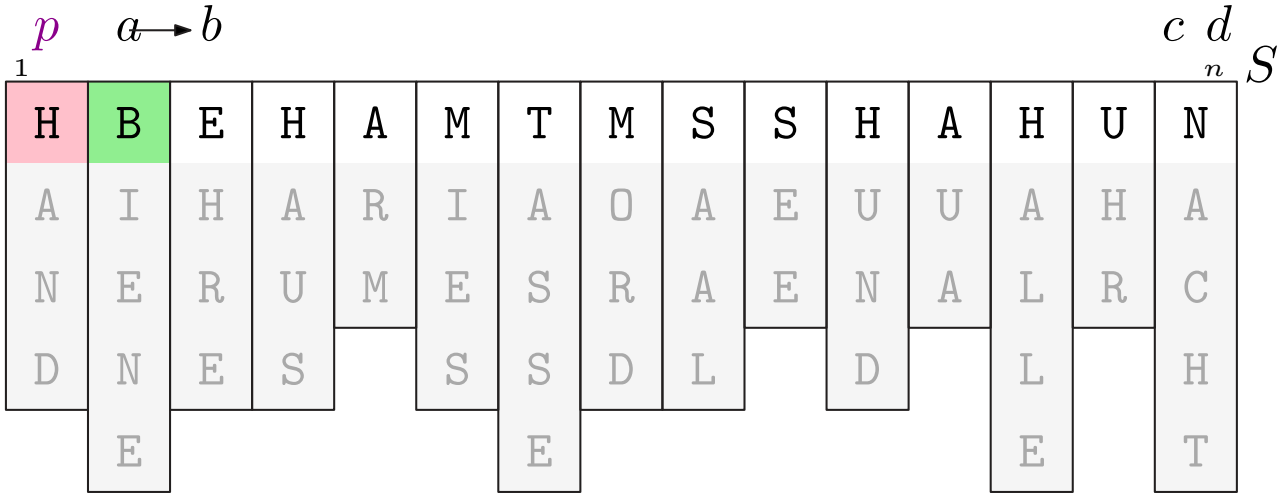


in-place Multikey Quicksort

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 $b > c$

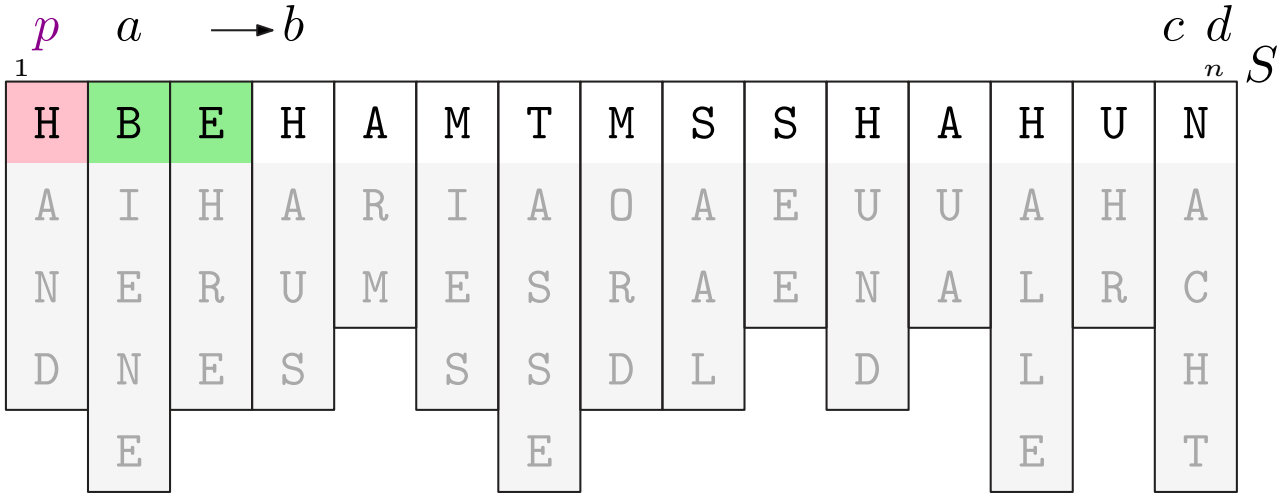


in-place Multikey Quicksort

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 $b > c$

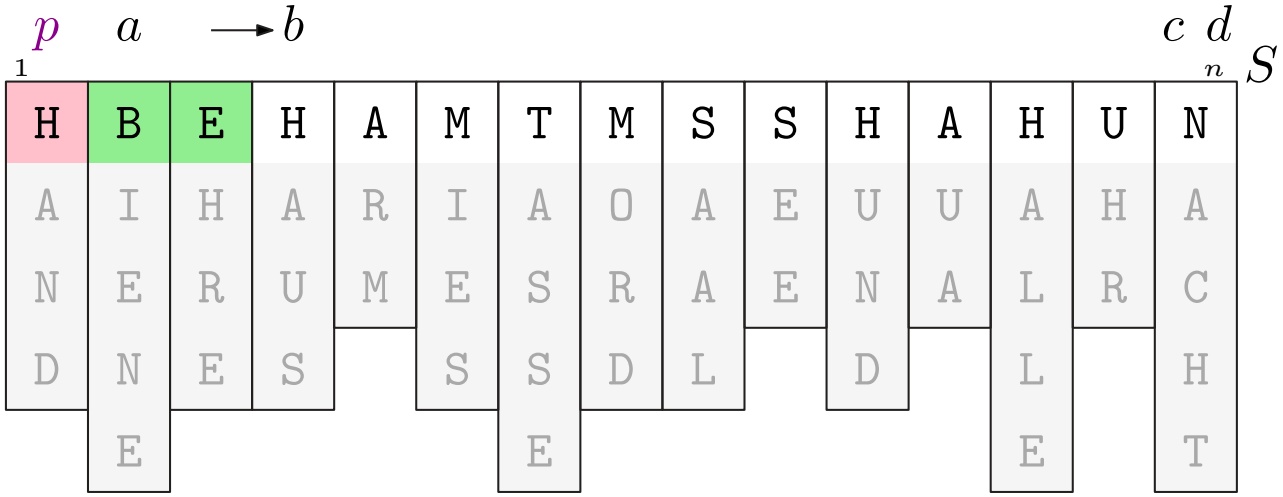


in-place Multikey Quicksort

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 $b > c$

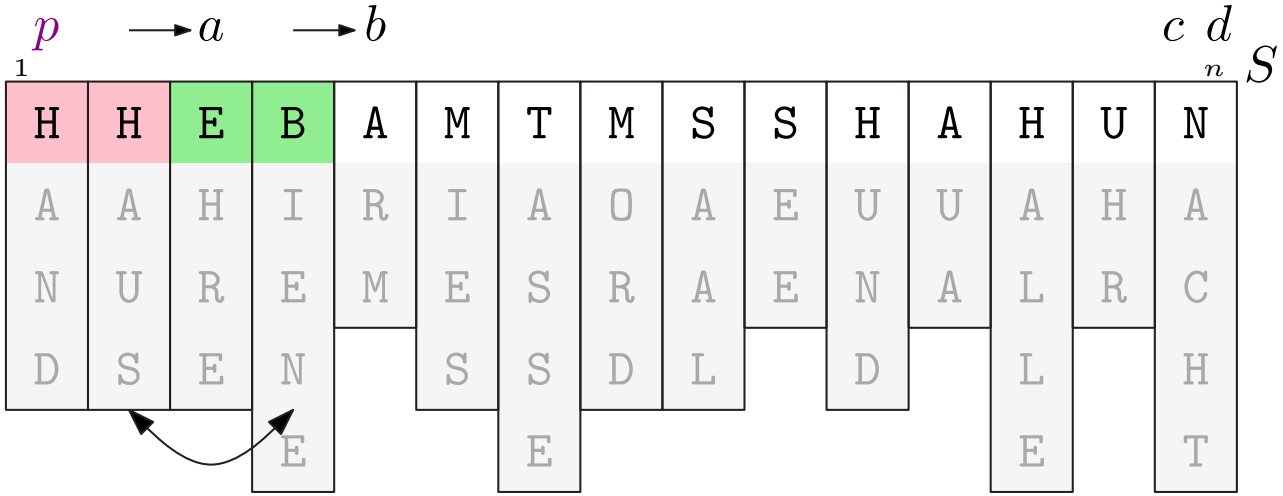


in-place Multikey Quicksort

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 $b > c$

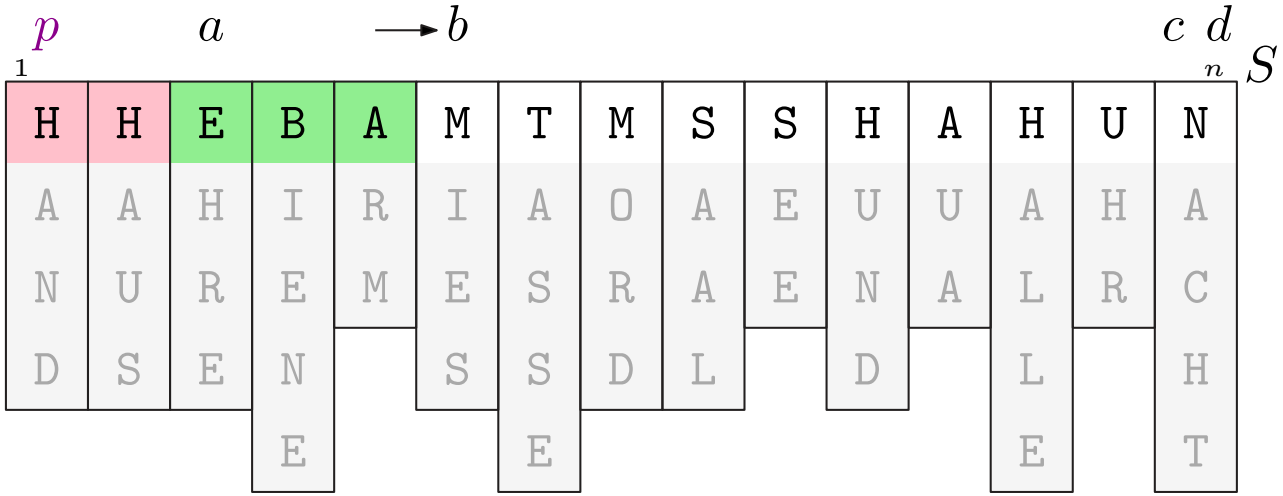


in-place Multikey Quicksort

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 $b > c$

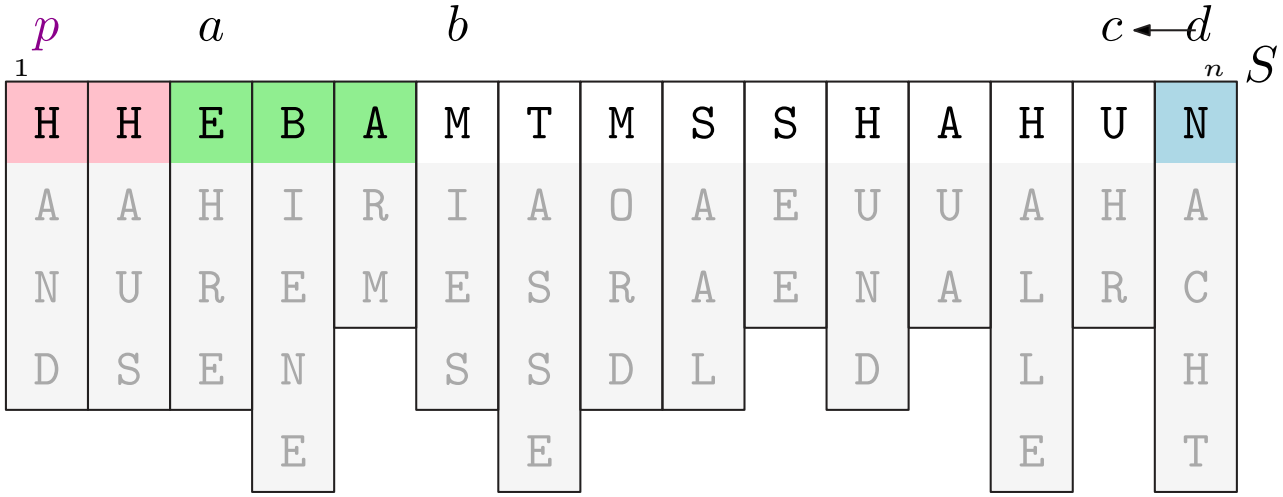


in-place Multikey Quicksort

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 $b > c$

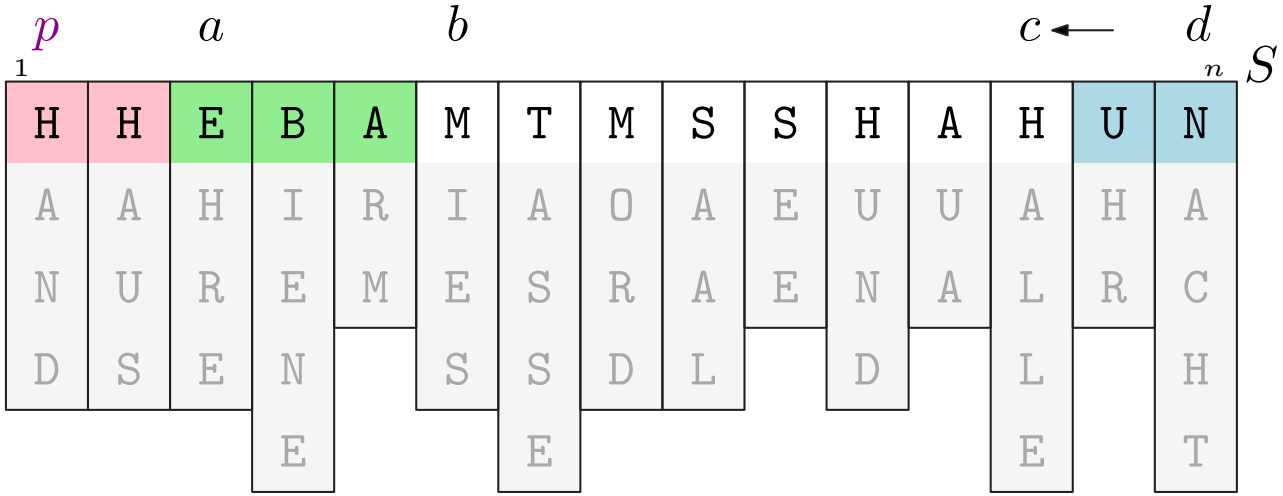


in-place Multikey Quicksort

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 $b > c$

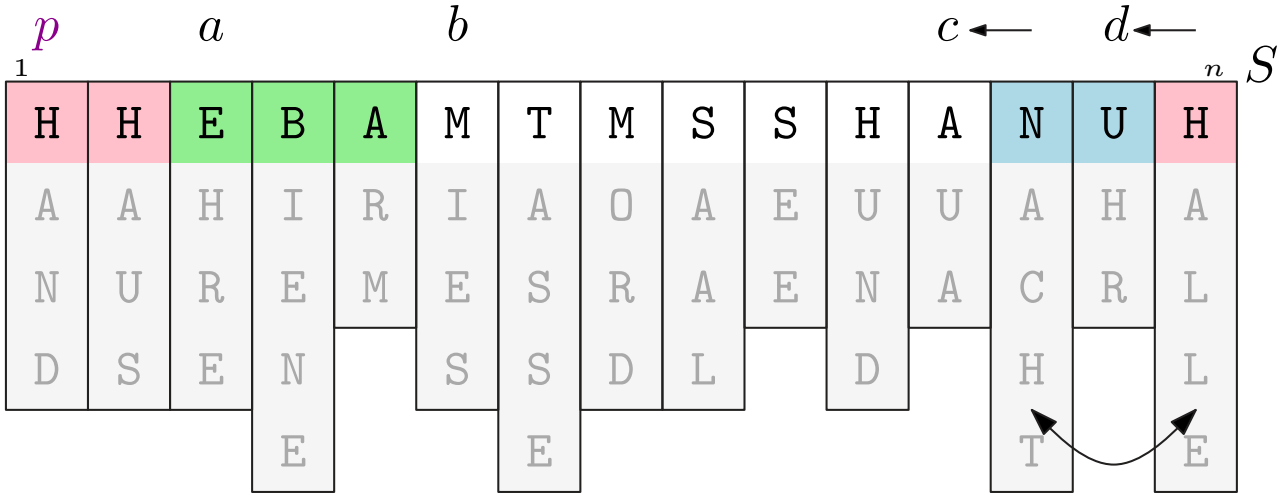


in-place Multikey Quicksort

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 $b > c$

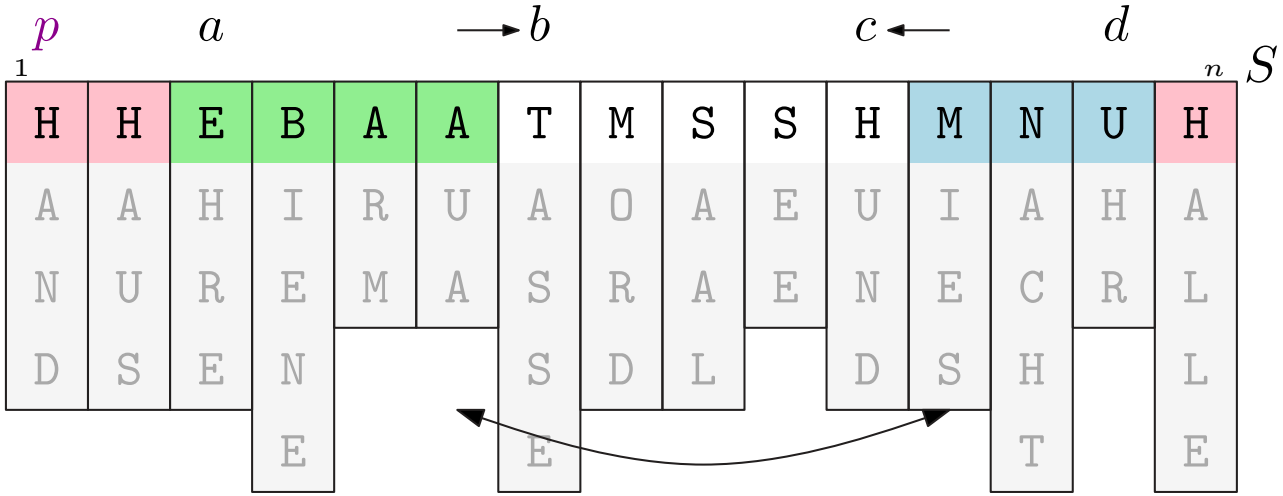


in-place Multikey Quicksort

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 $b > c$

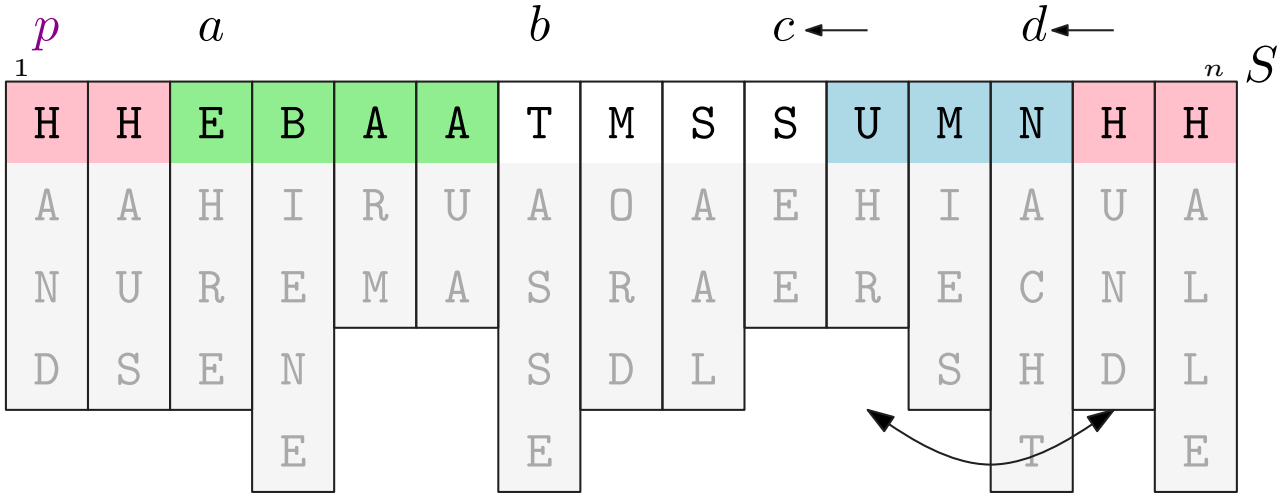


in-place Multikey Quicksort

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 $b > c$

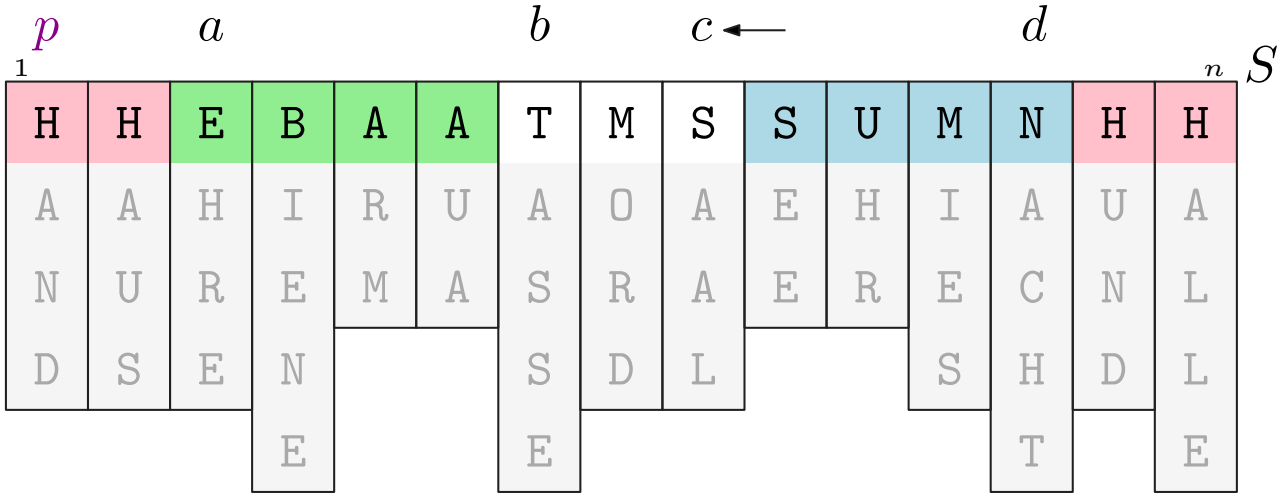


in-place Multikey Quicksort

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 $b > c$

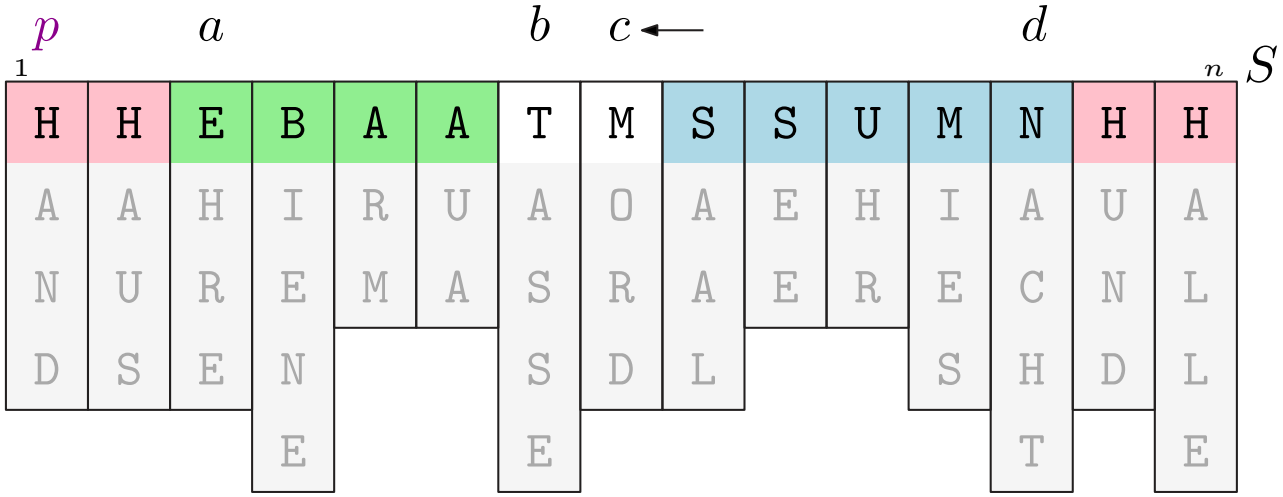


in-place Multikey Quicksort

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 $b > c$

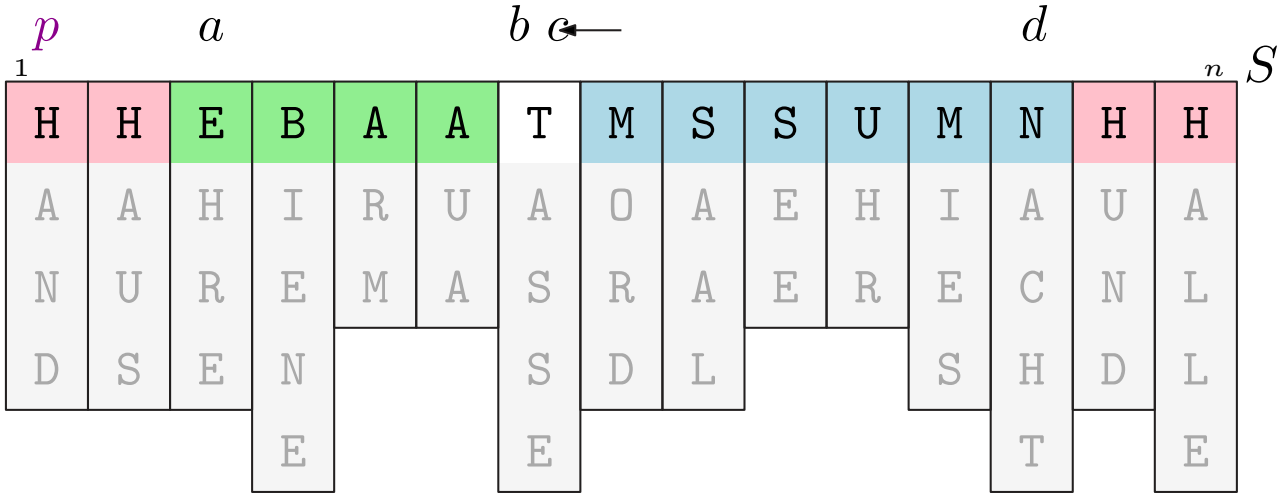


in-place Multikey Quicksort

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 $b > c$

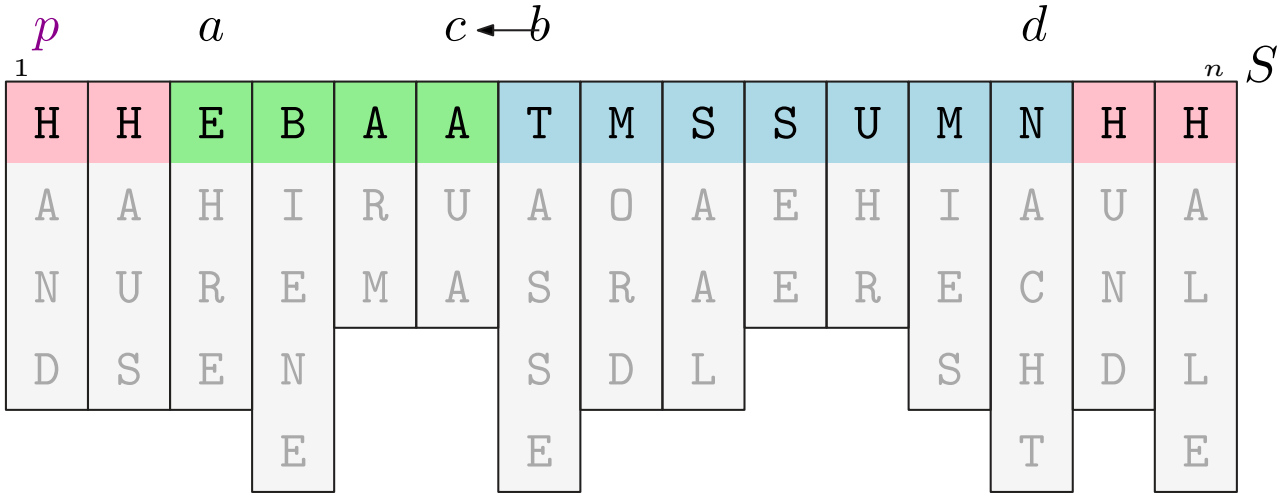


in-place Multikey Quicksort

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 $b > c$

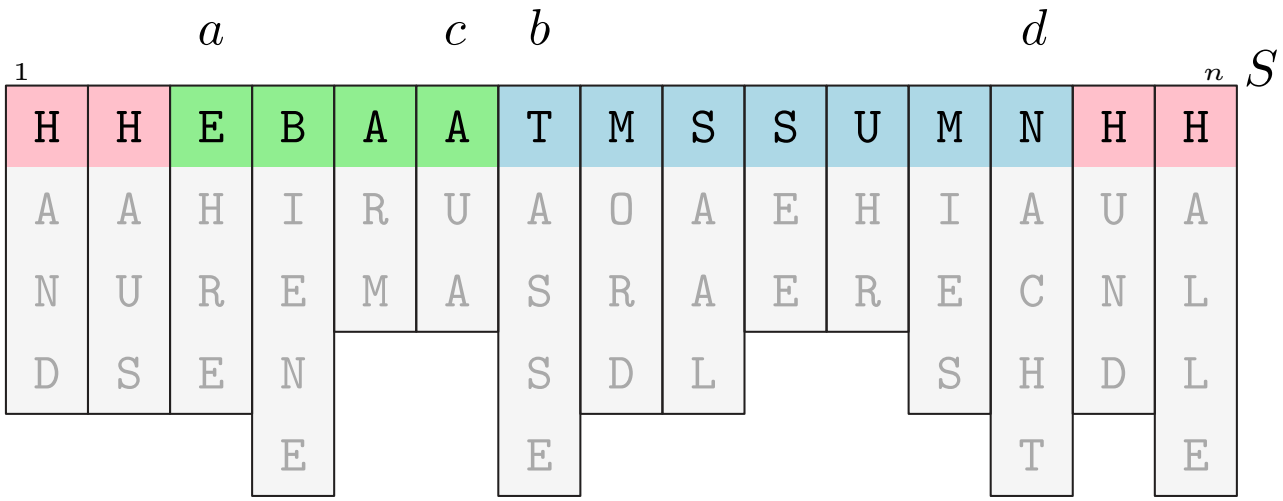


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort 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 $[n-r+1, n+1)$

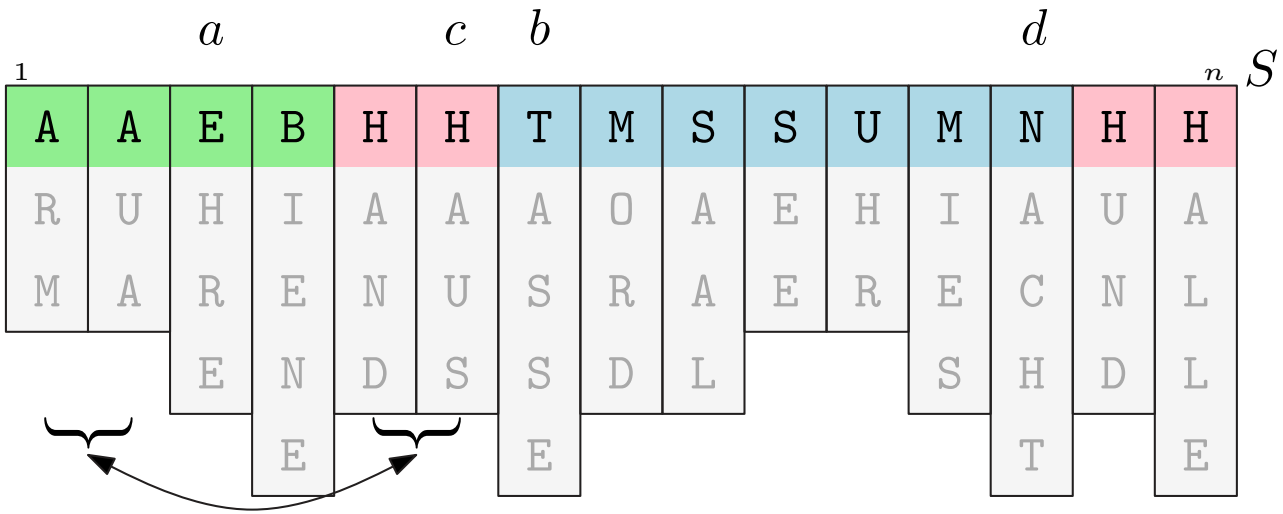


in-place Multikey Quicksort

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 $[n-r+1, n+1)$

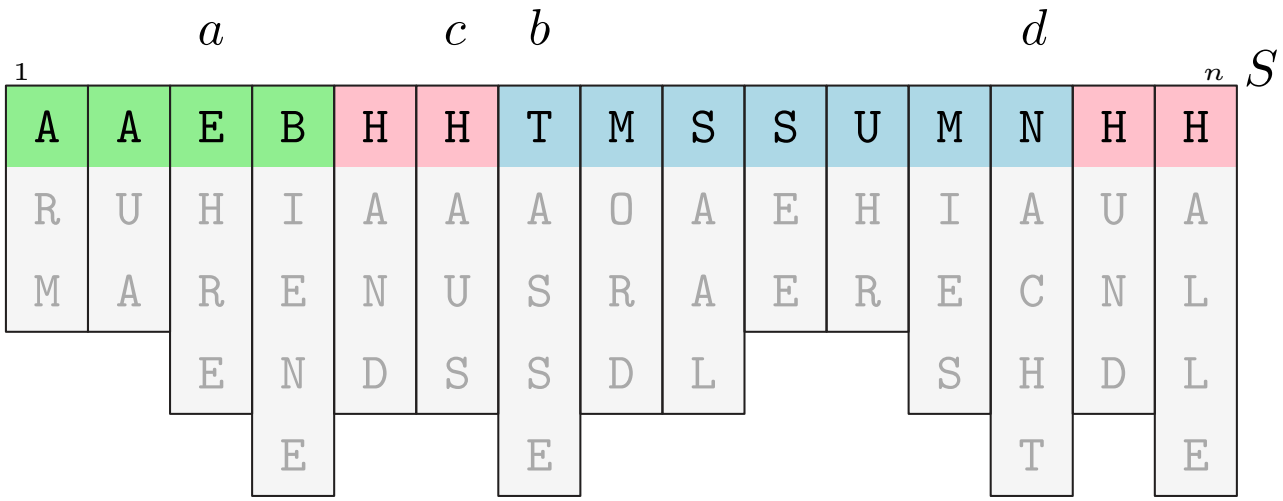


in-place Multikey Quicksort

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 $[n-r+1, n+1)$

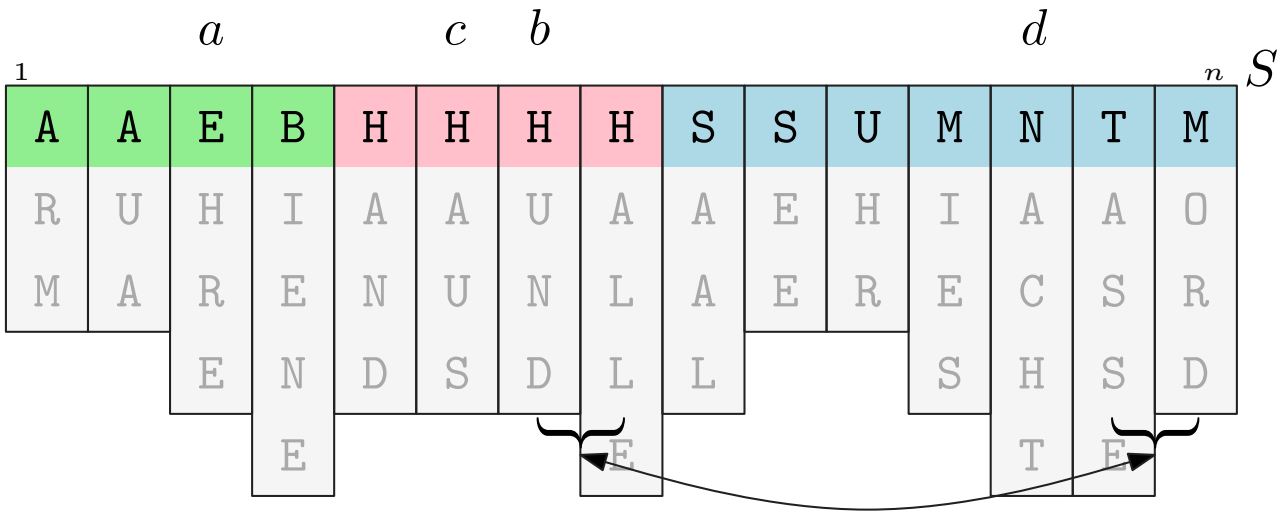


in-place Multikey Quicksort

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 $[n-r+1, n+1)$

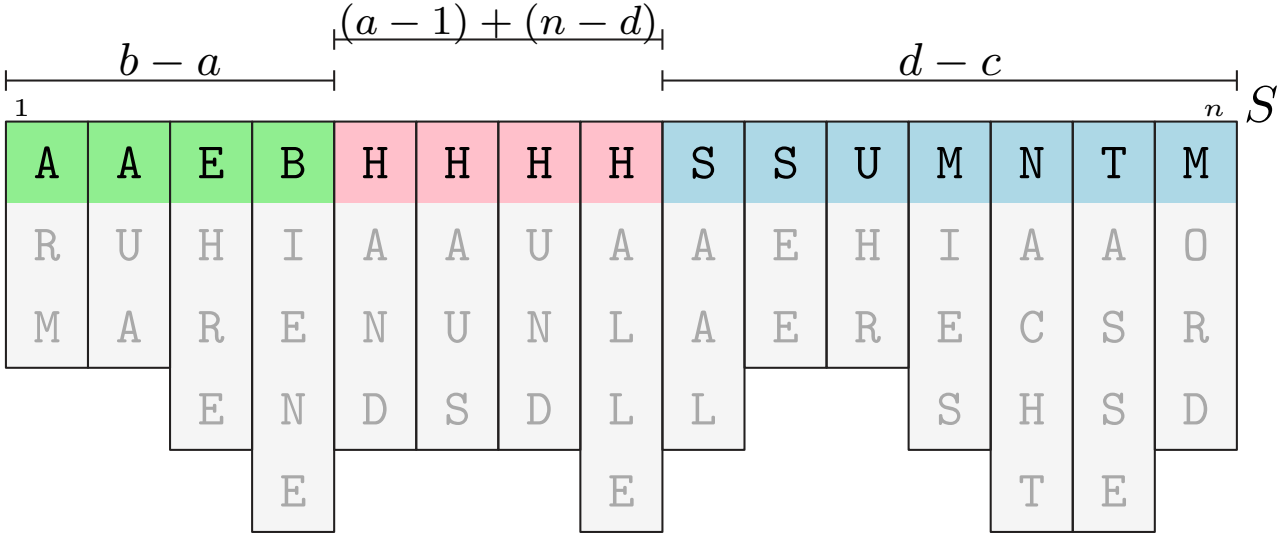


in-place Multikey Quicksort

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 $[n-r+1, n+1)$



in-place Multikey Quicksort

Zusammenfassung

- *Three-way Radix Quicksort*
  Partitionierung in kleiner, gleich, größer über alle Stellen analog zu msd-*Radixsort*
- effizient $\mathcal{O}(|S| \log |S| + d)$
  $d \mathrel{\hat{=}}$ 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

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															
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-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															
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-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

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	$									
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-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	$
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

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	$			
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-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	$											
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	$										
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	$								
4	b	a	r	h	a	b	a	r	b	e	r	$			
12	b	e	r	$											
13	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	$										
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

Suche: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start
- 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	$								
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: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start
- 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	$								
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: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start binäre Suche

  $l = 1, r = n$

  while $(l < r)$ **do**

  $\quad q = \lfloor \frac{l+r}{2} \rfloor$

  $\quad$ **if** $(P > T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q + 1$

  $\quad\quad$ **else** $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

  $\quad$ **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	$								
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: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start binäre Suche

  $l = 1, r = n$

  while $(l < r)$ **do**

  $\quad q = \lfloor \frac{l+r}{2} \rfloor$

  $\quad$ **if** $(P > T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q + 1$

  $\quad\quad$ **else** $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

  $\quad$ **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: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start binäre Suche

  $l = 1, r = n$

  while $(l < r)$ **do**

  $\quad q = \lfloor \frac{l+r}{2} \rfloor$

  $\quad$ **if** $(P > T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q + 1$

  $\quad\quad$ **else** $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

  $\quad$ **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	$	
$q =$ 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	$					

$l = 1, r = 8$

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	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	$			
$q =$ 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	$				

$l = 5, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start binäre Suche

  $l = 1, r = n$

  while $(l < r)$ **do**

  $\quad q = \lfloor \frac{l+r}{2} \rfloor$

  $\quad$ **if** $(P > T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q + 1$

  $\quad\quad$ **else** $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

  $\quad$ **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	$									
$q =$ 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	$					

$l = 5, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

- (SA bestimmen)
- finde Start binäre Suche

  $l = 1, r = n$

  while $(l < r)$ **do**

  $\quad q = \lfloor \frac{l+r}{2} \rfloor$

  $\quad$ **if** $(P > T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q + 1$

  $\quad\quad$ **else** $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

  $\quad$ **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	$								
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	$					

$l = 6, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

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

  $l = s, r = n$

  while $(l < r)$ **do**

  $\quad q = \lceil \frac{l+r}{2} \rceil$

  $\quad$ **if** $(P = T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q$

  $\quad\quad$ **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	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	$											
$q =$ 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	$					

$l = 5, r = 15$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

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

  $l = s, r = n$

  while $(l < r)$ **do**

  $\quad q = \lceil \frac{l+r}{2} \rceil$

  $\quad$ **if** $(P = T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q$

  $\quad\quad$ **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	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 $
$q =$	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	$				

$l = 5, r = 9$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

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

  $l = s, r = n$

  while $(l < r)$ **do**

  $\quad q = \lceil \frac{l+r}{2} \rceil$

  $\quad$ **if** $(P = T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q$

  $\quad\quad$ **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	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	$				

$q =$

$l = 7, r = 9$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

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

  $l = s, r = n$

  while $(l < r)$ **do**

  $\quad q = \lceil \frac{l+r}{2} \rceil$

  $\quad$ **if** $(P = T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q$

  $\quad\quad$ **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	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	$			
$q =$ 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	$					

$l = 8, r = 9$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

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

  $l = s, r = n$

  while $(l < r)$ **do**

  $\quad q = \lceil \frac{l+r}{2} \rceil$

  $\quad$ **if** $(P = T_{SA[q]..SA[q]+m-1})$

  $\quad\quad$ **then** $l = q$

  $\quad\quad$ **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	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	$				

$l = 8, r = 8$

Suche mit Suffix-Arrays

Ablauf

Karlsruher Institut für Technologie

Suche: $P = \texttt{bar}$

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

$s = 6, t = 8$

Suche mit Suffix-Arrays

Zusammenfassung

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

  gut, wenn auf einem Text viele Anfragen stattfinden

- Ausnutzung der Eigenschaften des Suffix-Arrays
  - $\rightarrow$ jeder Substring ist Präfix eines Suffix
  - $\rightarrow$ alle Substrings liegen "sortiert" vor

    mögliche Ausnahme: Substring ist Präfix von Substring

  das Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge

LCP-Array

Längster gemeinsamer Präfix

Definition:

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

Erweiterung auf beliebige Suffixe

- $\mathrm{LCP}[i][j]$: Länge des längsten gemeinsamen Präfix beliebiger lexikographischer Suffixe $A[\mathrm{SA}[i]\dots n]$ und $A[\mathrm{SA}[j]\dots n]$
- 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	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	$				
$L =$ 6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
$q =$ 7	9	b	a	r	b	e	r	$								
8	4	b	a	r	h	a	b	a	r	b	e	r	$			
$R =$ 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	$					

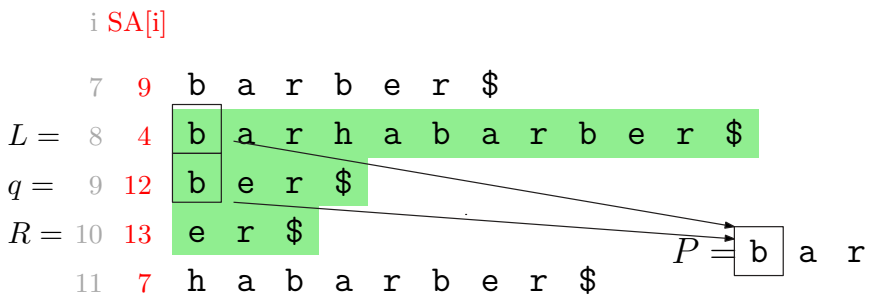
$L = 6, R = 9$
$l = 3, r = 1$
$mlr := 1 = \min(l, r)$

Schnelle Suche mit Suffix-Arrays

Redundante Vergleiche

Problem

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



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 $\max(l, r)$ 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

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

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 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

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

Suche: $P =$ `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** $(l > r \wedge \mathrm{LCP}[L, q] = l)$

$q =$ 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

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

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

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

$L =$ 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

$l = 7, r = 4$

Suche mit Suffix-Arrays

Ablauf

Karlsruher Institut für Technologie

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

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

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

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

$q =$ 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

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

$l = 7, r = 4$

Suche mit Suffix-Arrays

Ablauf

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

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

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 ...

$L =$ b a r b e r a c c $
$q =$ b a r b e r c b c ...
$R =$ b a r b i

$\mathrm{LCP}[L, q] = 6$
$l = 8, r = 4$

Suche mit Suffix-Arrays

Ablauf

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

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

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 ...
$L =$ b a r b e r a c c $
$R =$ b a r b e r c b c ...
b a r b i

$l = 8, r = 6$

Suche mit Suffix-Arrays

Ablauf

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

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

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	...
$L = q =$	b	a	r	b	e	r	a	c	c	$
$R =$	b	a	r	b	e	r	c	b	c	...
	b	a	r	b	i					

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

$l = 8, r = 6$

Suche mit Suffix-Arrays

Ablauf

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

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** $(l > r \wedge \mathrm{LCP}[L, q] = l)$
  start at l

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 ...
$L = R =$ b a r b e r a c c $
b a r b e r c b c ...
b a r b i

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$

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

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
  - $\rightarrow$ einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ 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
  - $\rightarrow$ einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ einmal LCP-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ einmal erweitertes LCP-Array generieren in $\mathcal{O}(n \log n)$,
  - $\rightarrow$ danach Anfrage in $\mathcal{O}(m + \log n)$

Ende!

Feierabend!