Karlsruher Institut für Technologie

Übung 11 – Algorithmen II

Daniel Seemaier, Tobias Heuer – *daniel.seemaier@kit.edu, tobias.heuer@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS20.php`

Institut für Theoretische Informatik - Algorithmik II

Organisatorisches

Klausurtermin

Mo 15.03.2021 16:00 Uhr
Hörsaaleinteilung wird rechtzeitig bekannt gegeben

Klausuranmeldung

Vorraussichtlich bis zehn Tage vor Klausurtermin
($\approx$ 05.03.2020)

Evaluation der Übung

nachher

Themenübersicht

- *in-place Multikey Quicksort*
  (Sortierung von Zeichenketten *in-place*)
- KMP Algorithmus

in-place 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
- Partitionierungschritt kann *in-place* erfolgen
  $\rightarrow$ ähnlich wie bei normalem *Quicksort*

in-place 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

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

in-place Multikey Quicksort

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

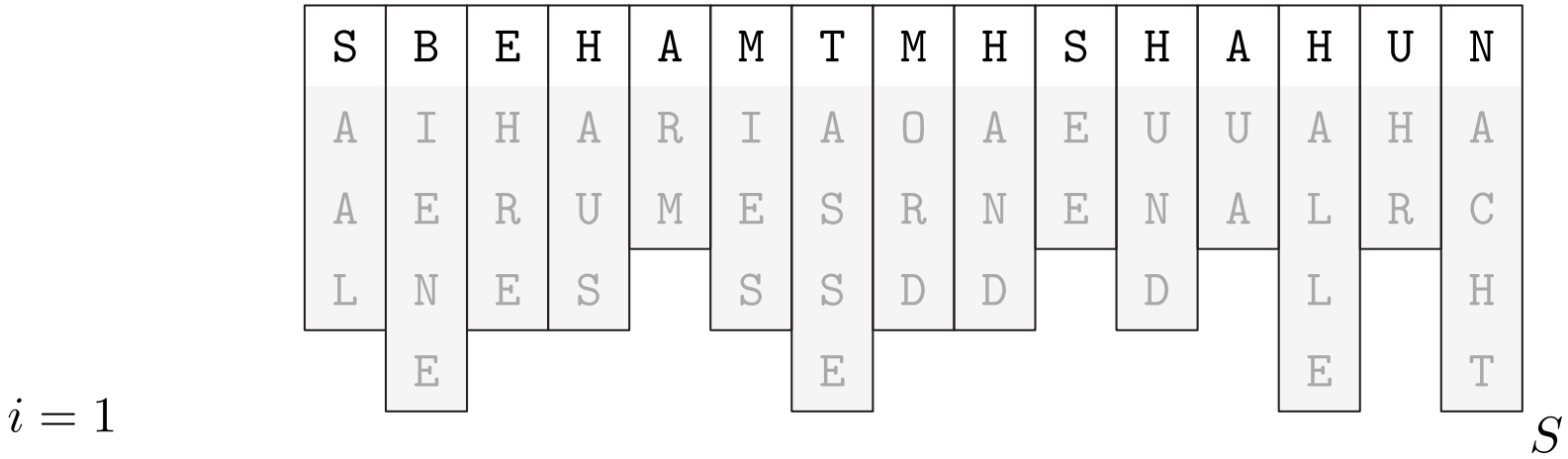
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

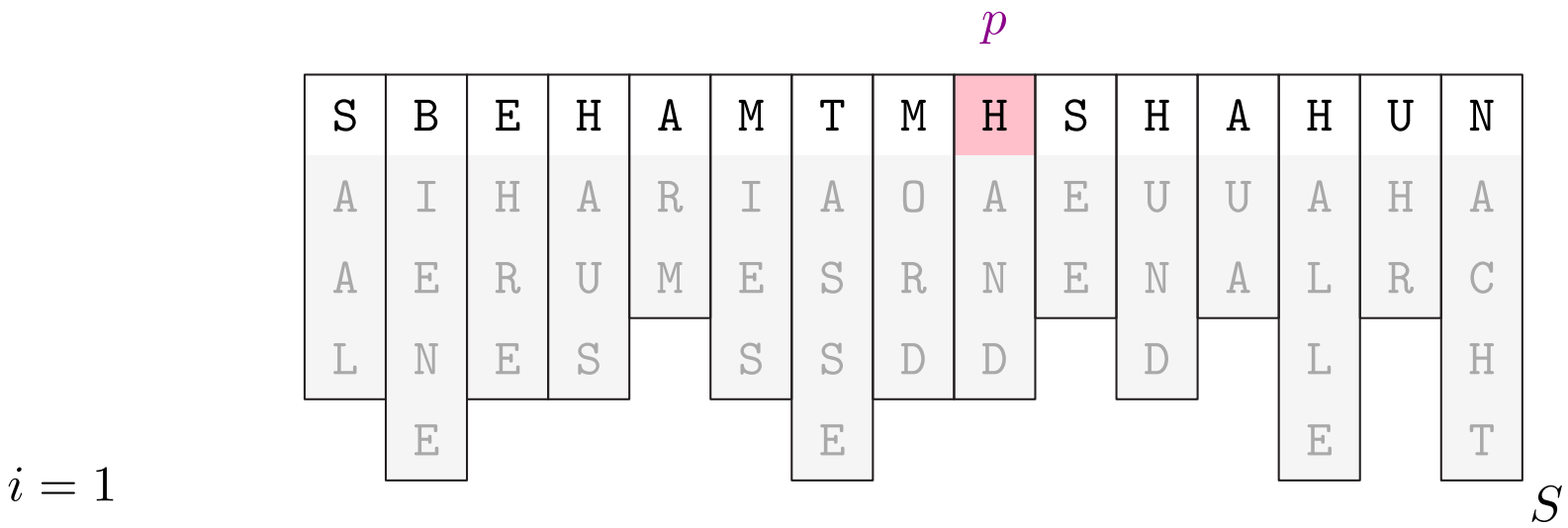
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

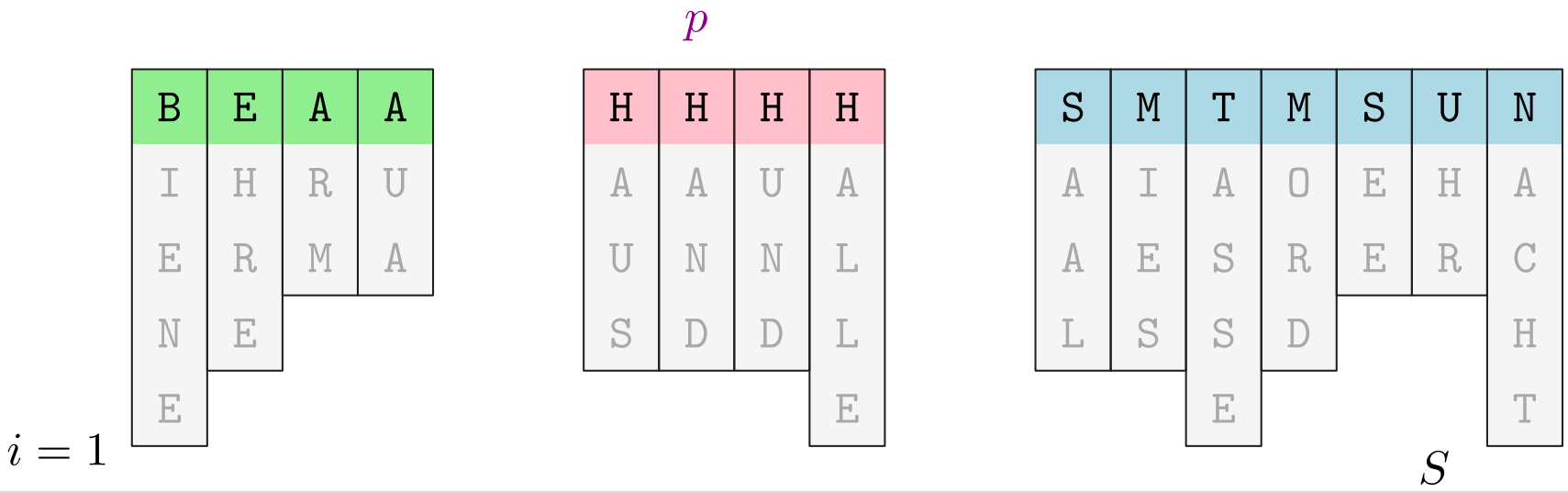
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

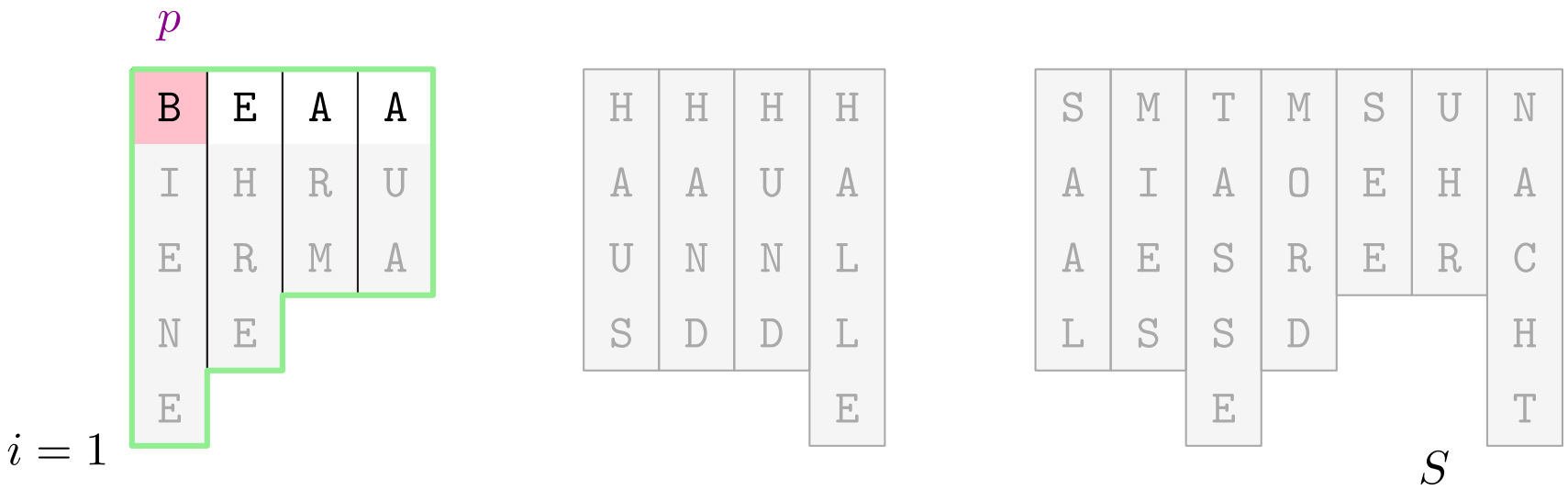
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

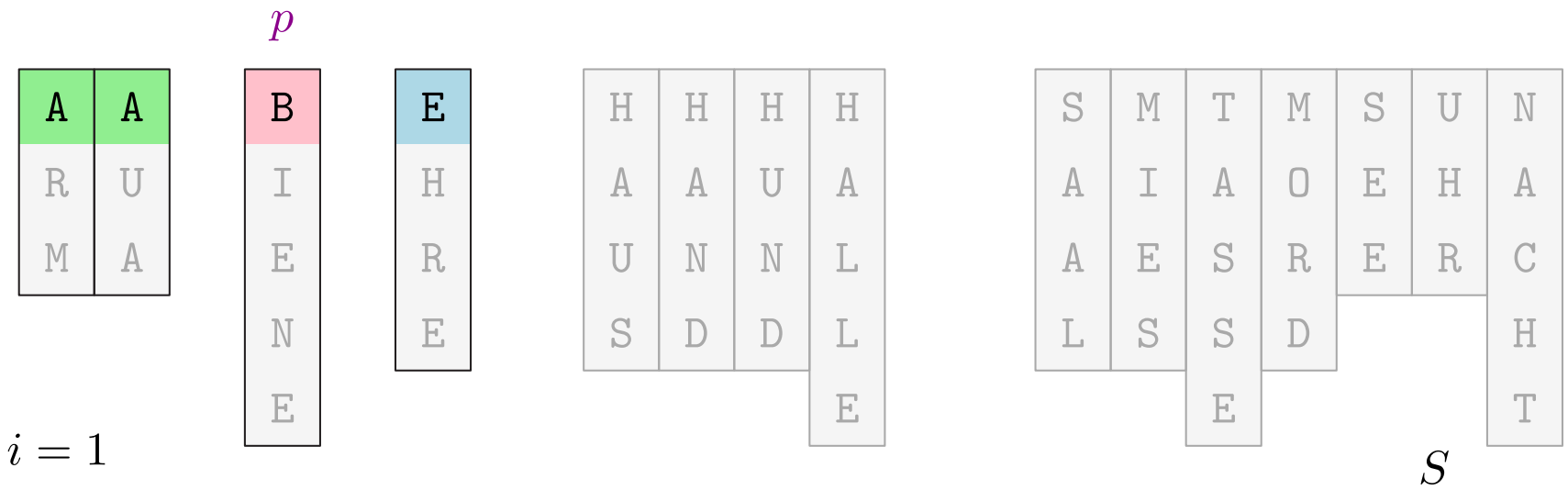
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

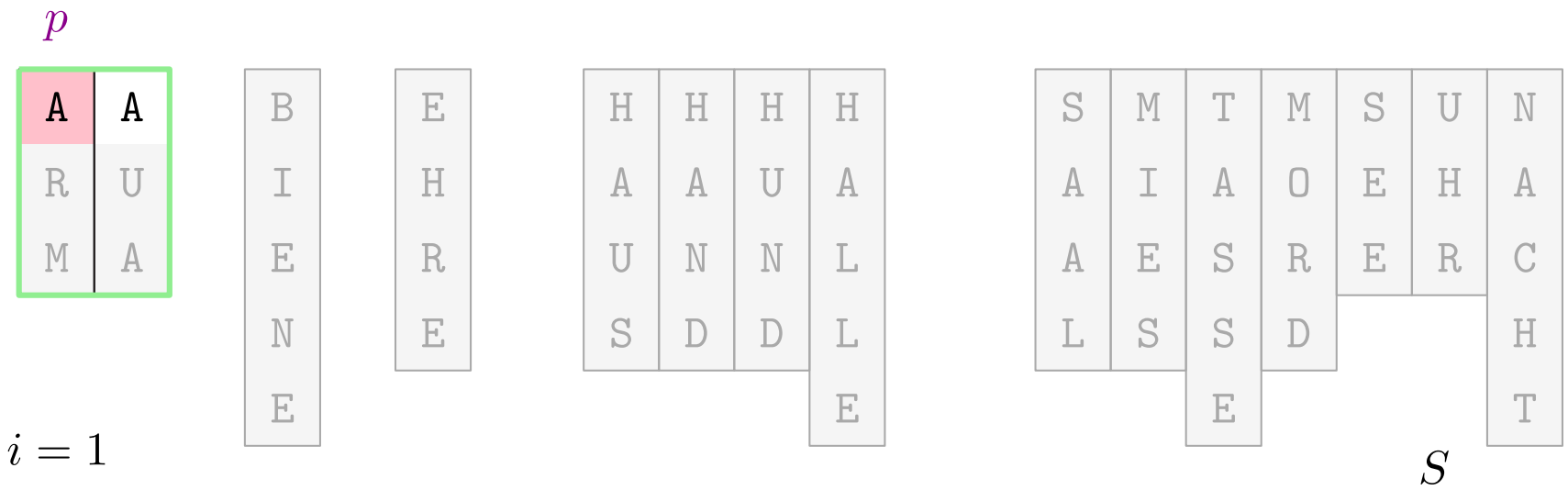
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

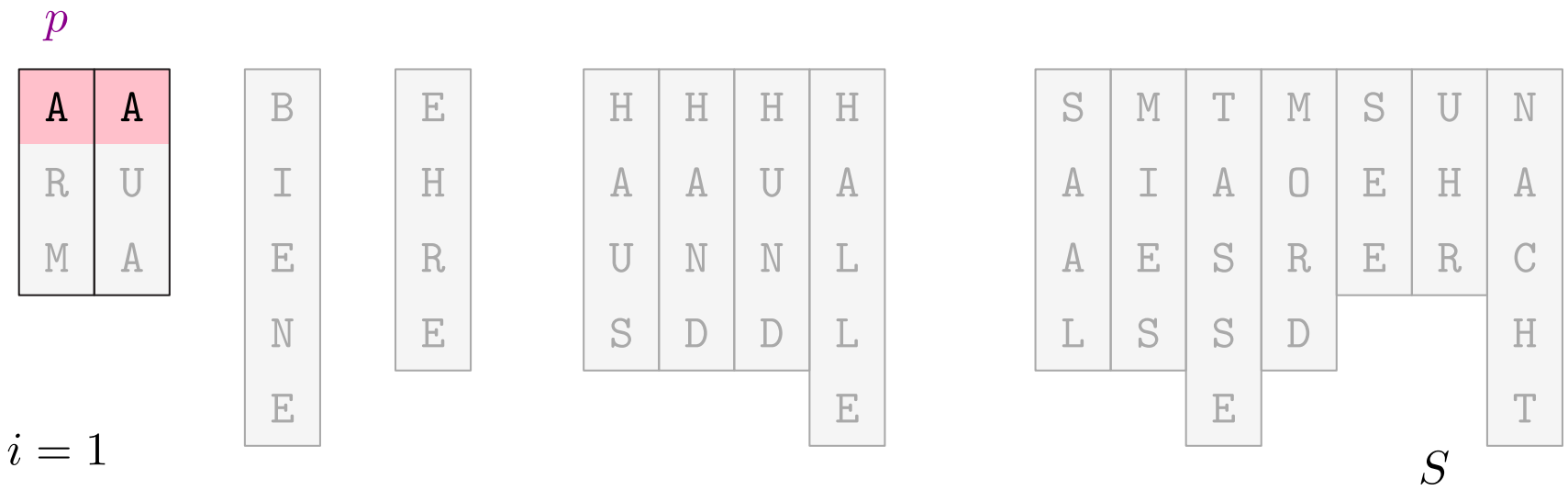
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

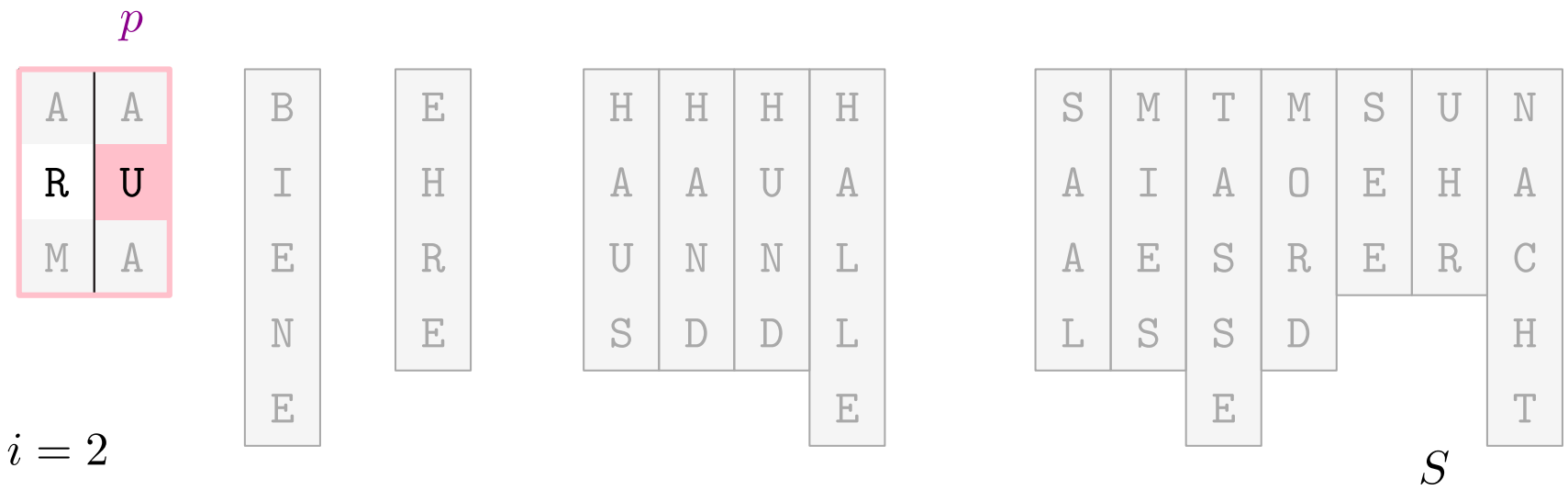
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

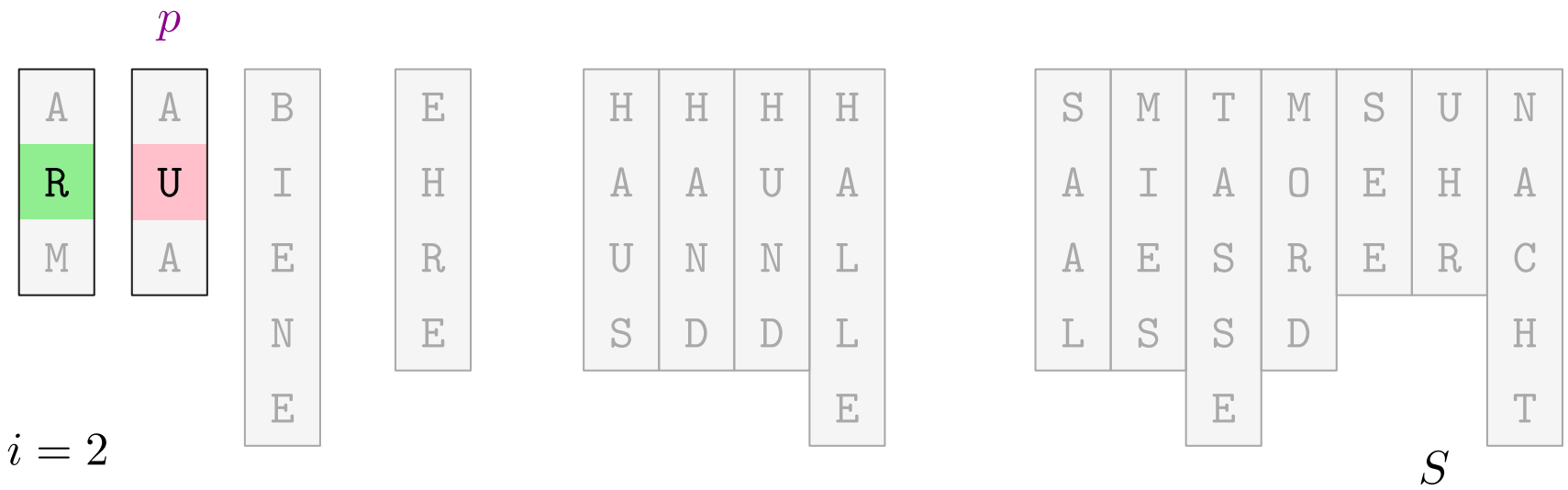
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

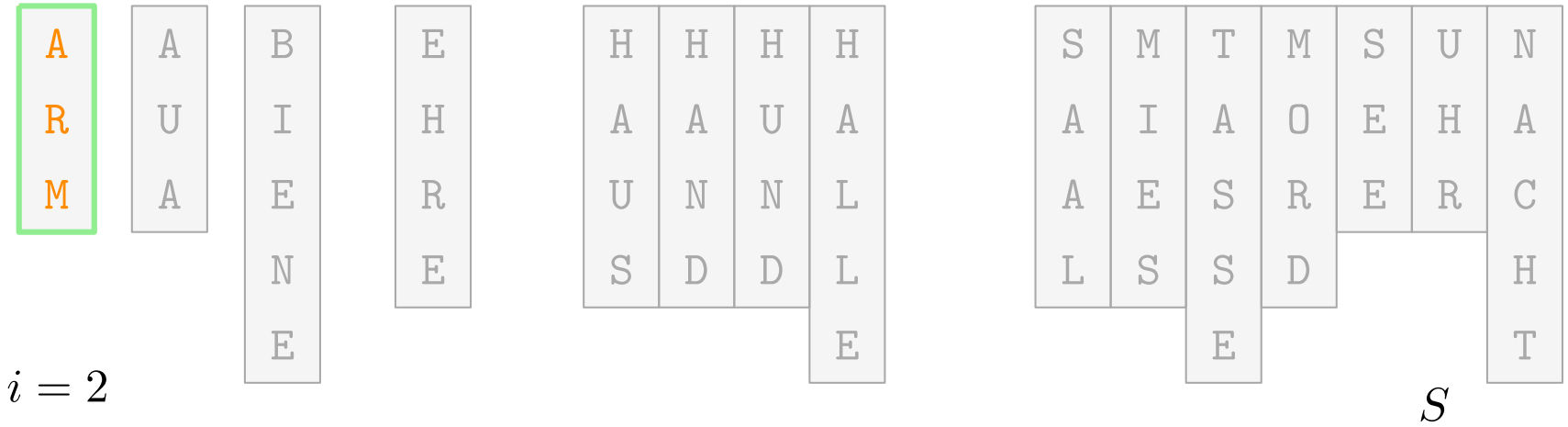
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

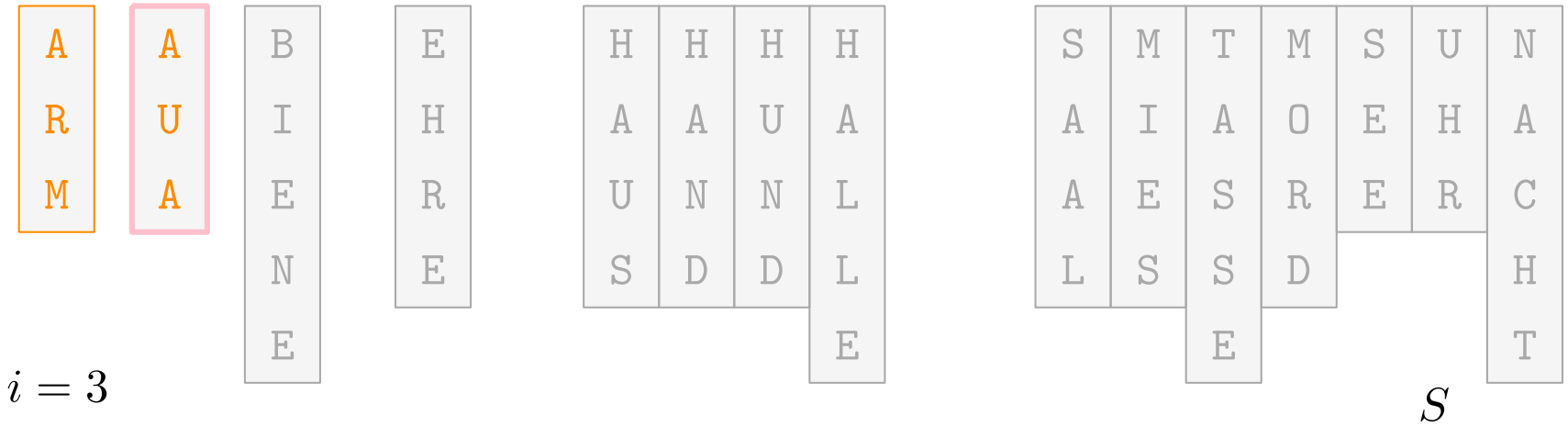
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

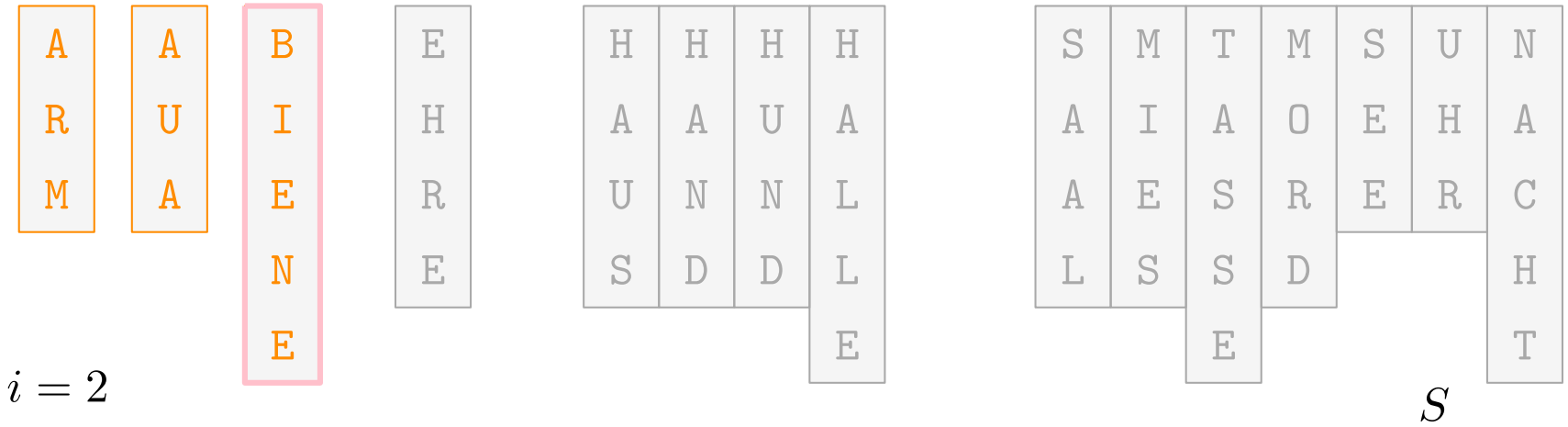
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

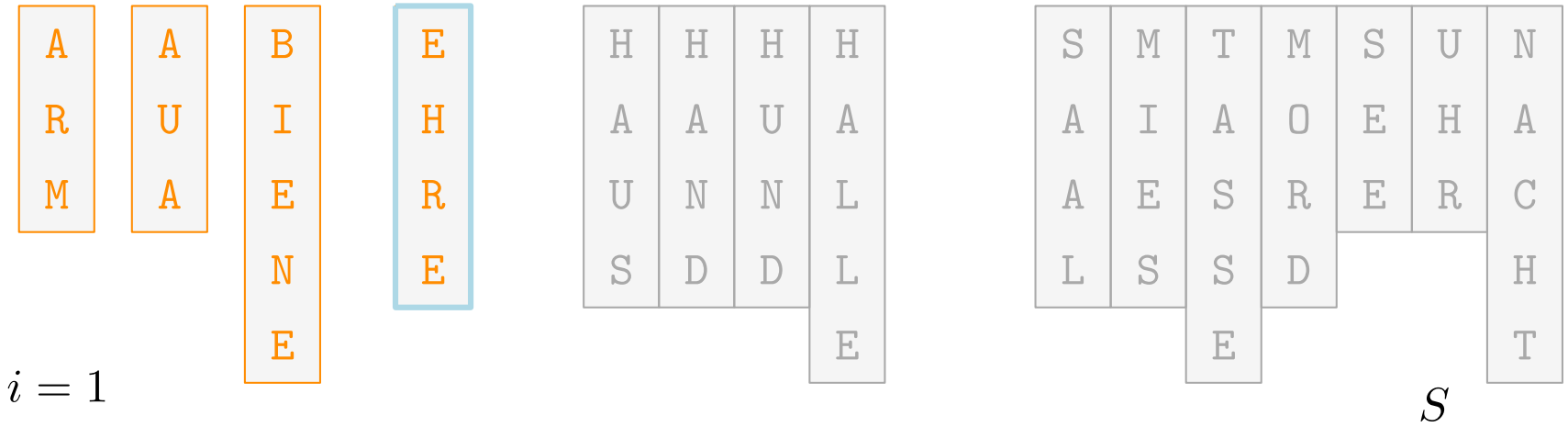
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

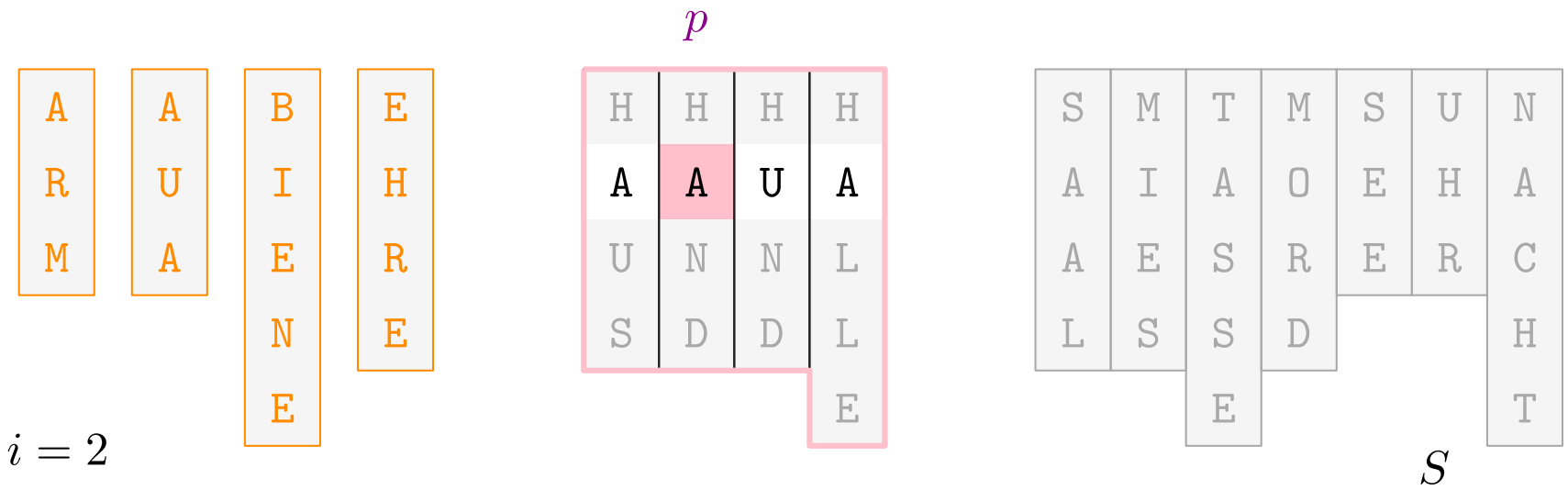
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

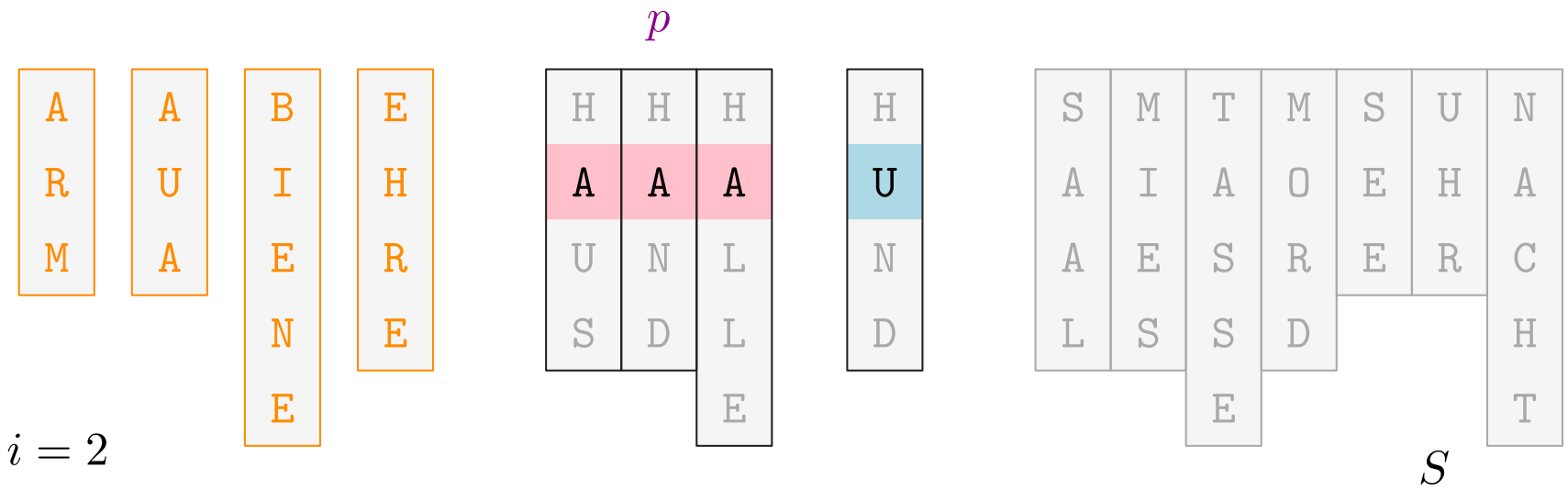
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

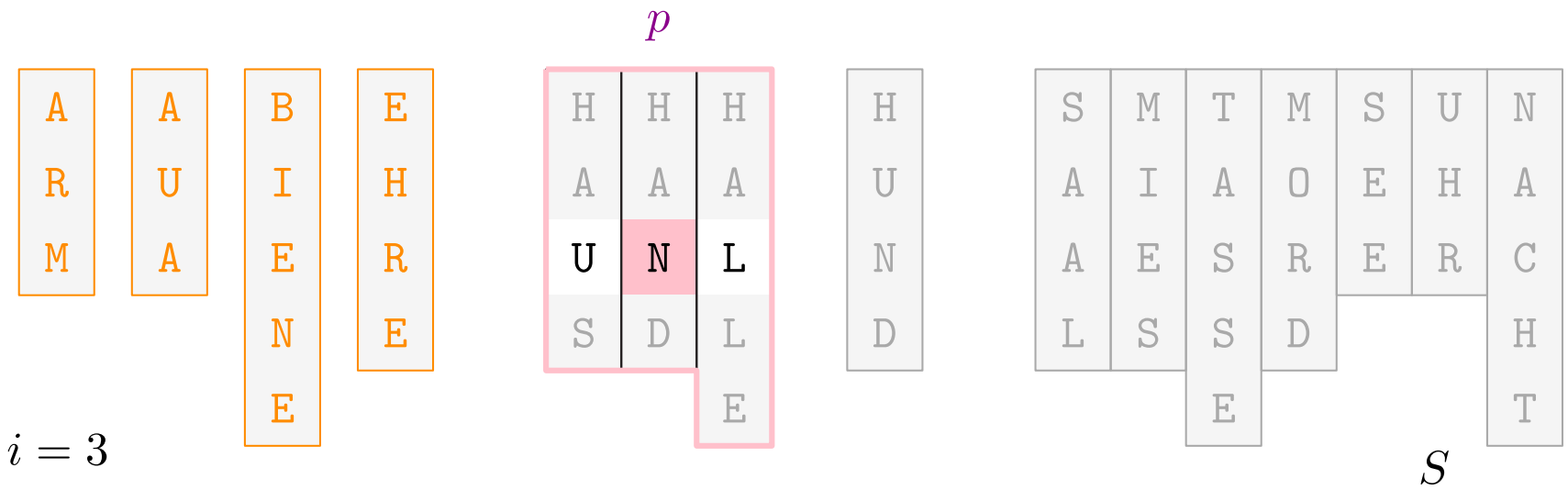
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

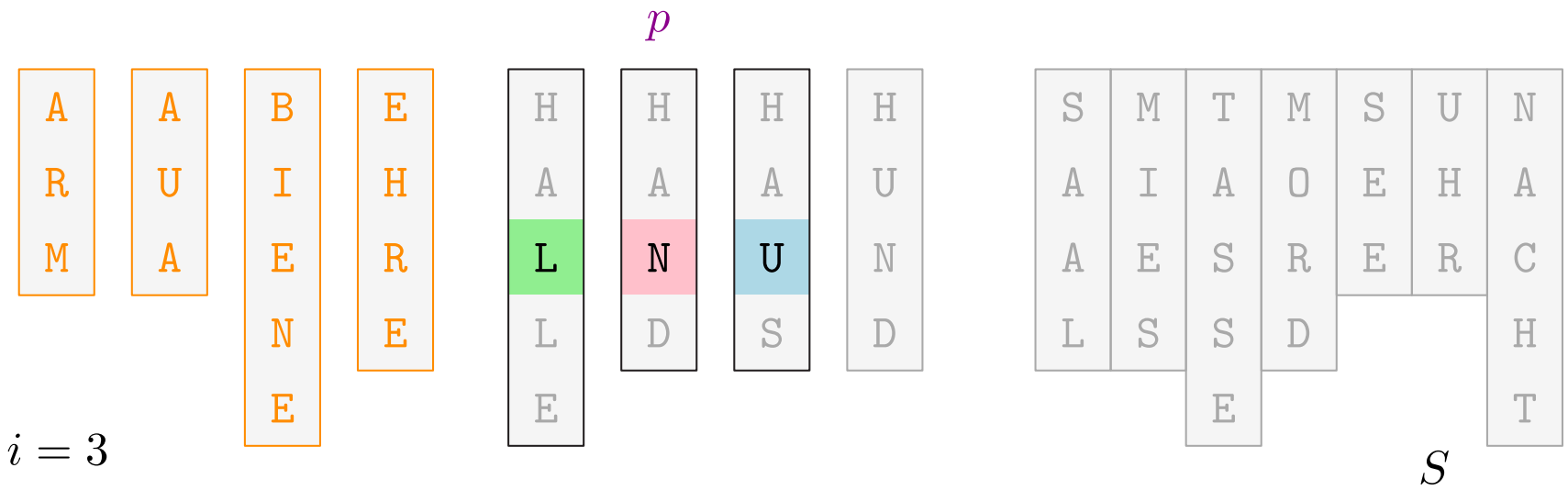
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

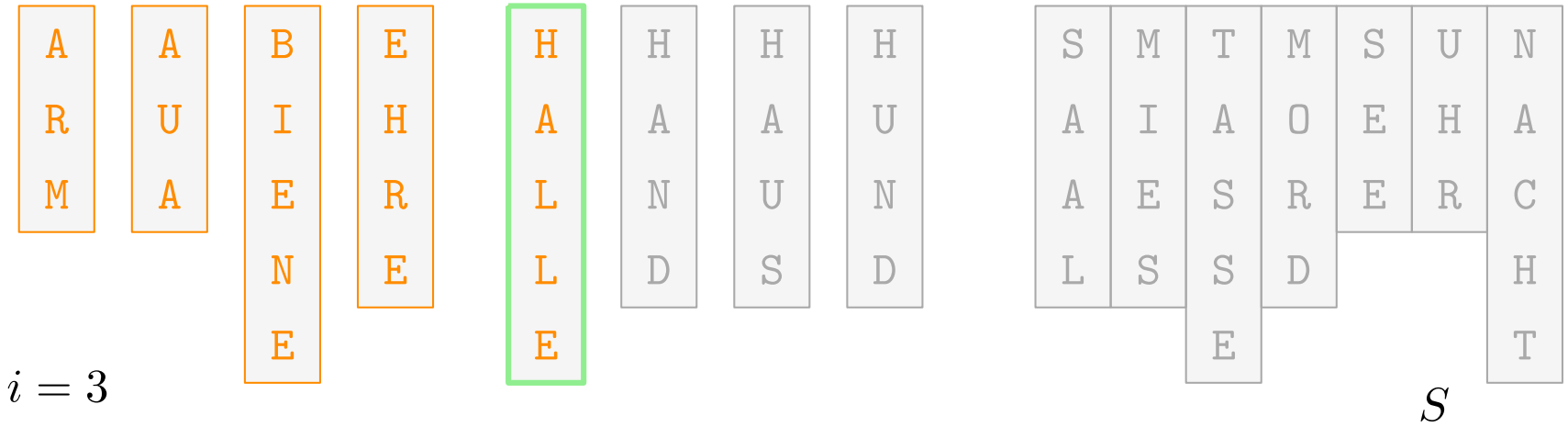
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

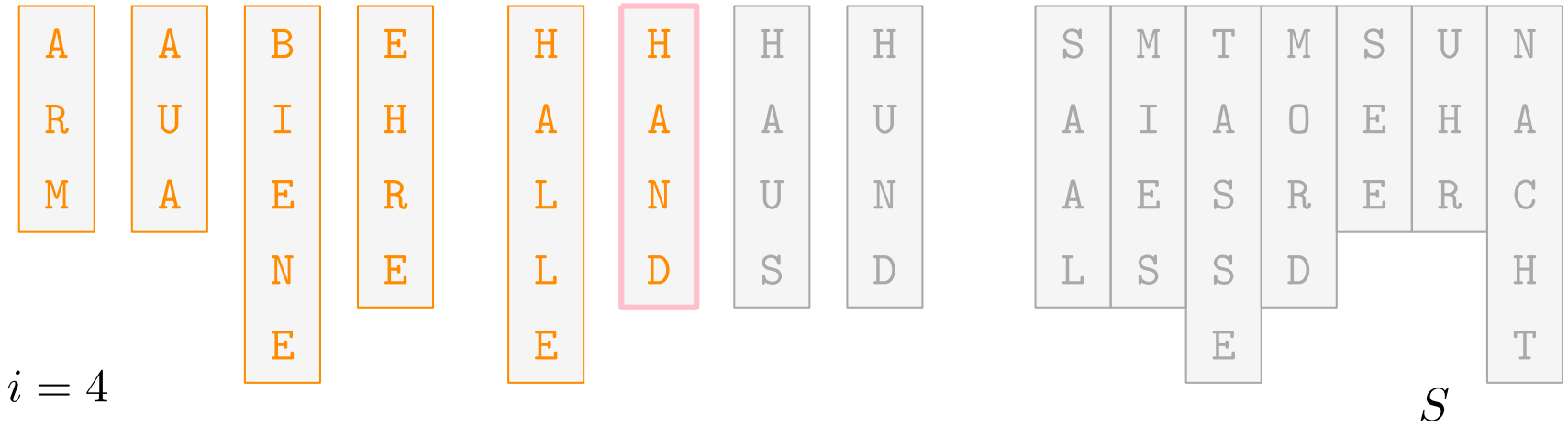
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

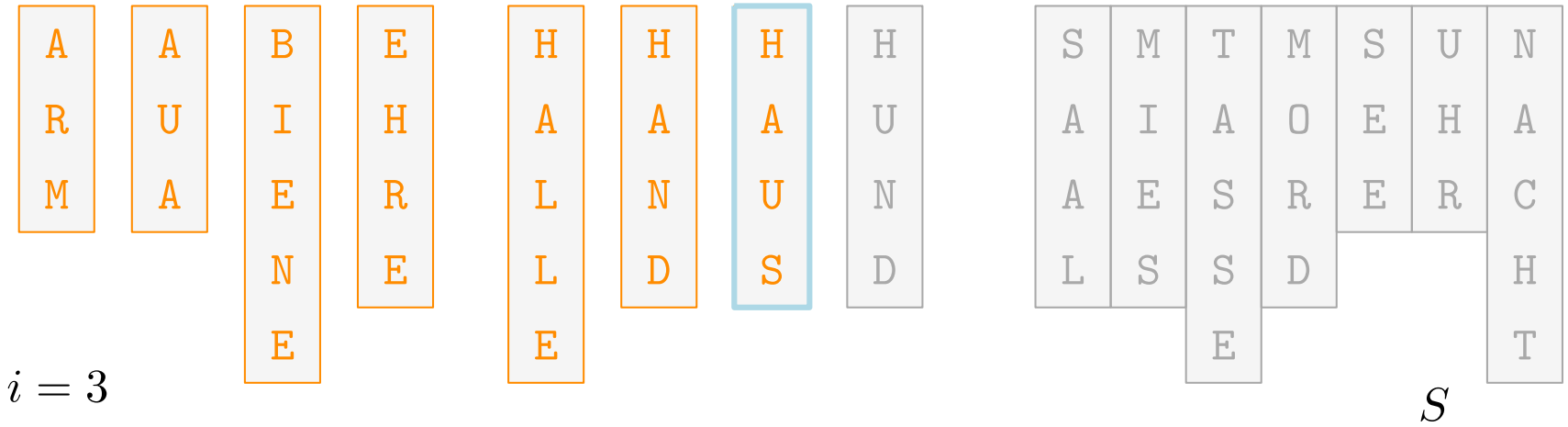
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

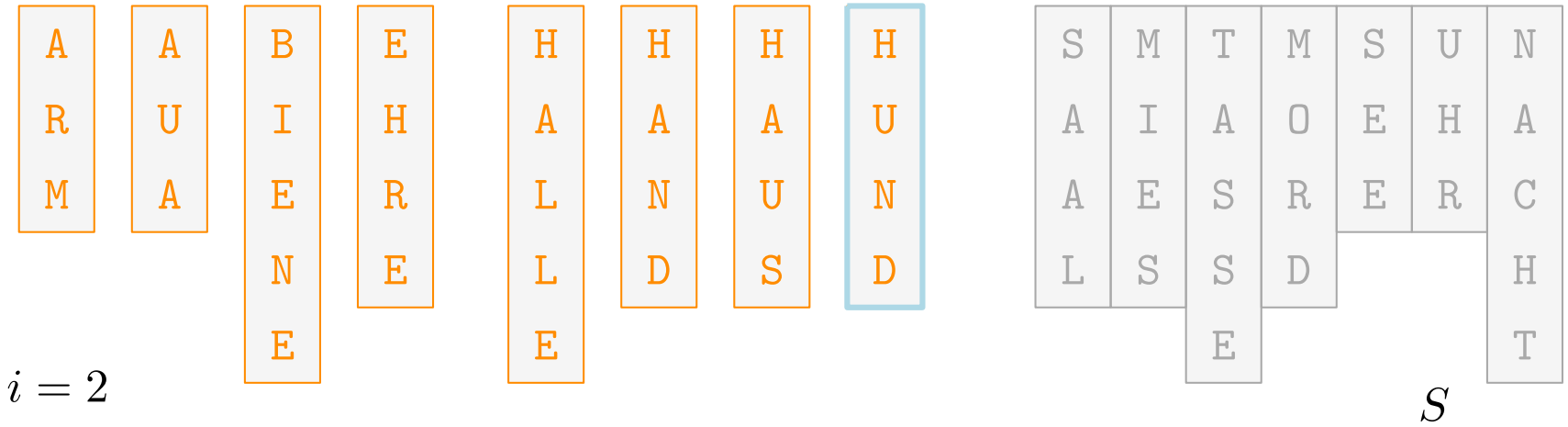
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

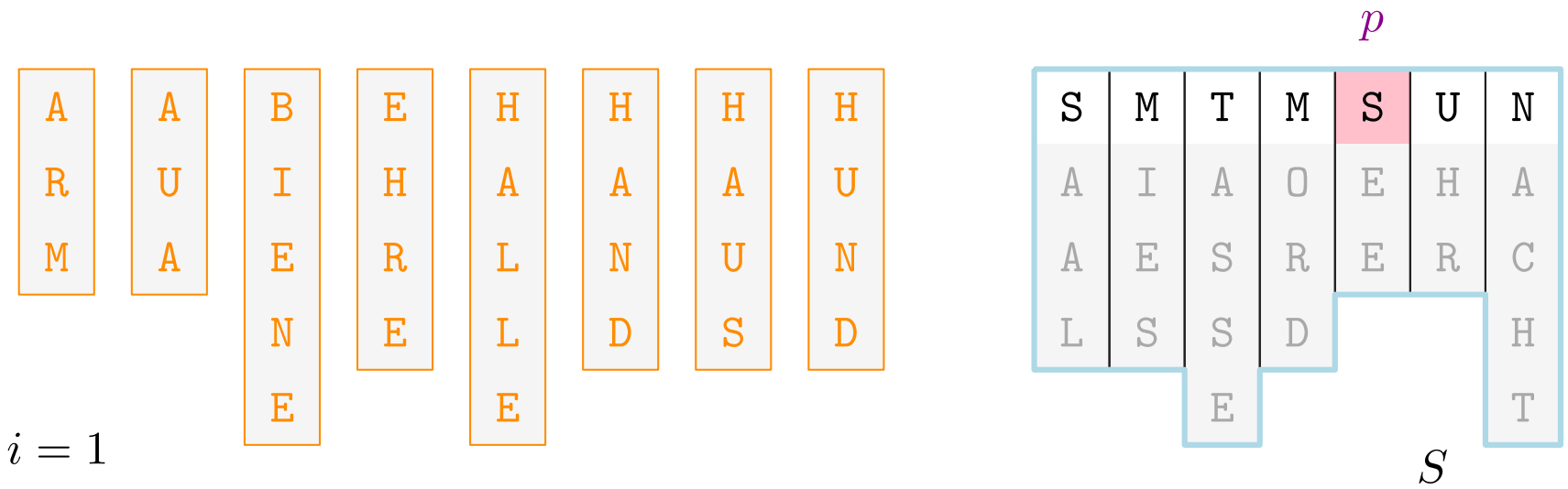
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

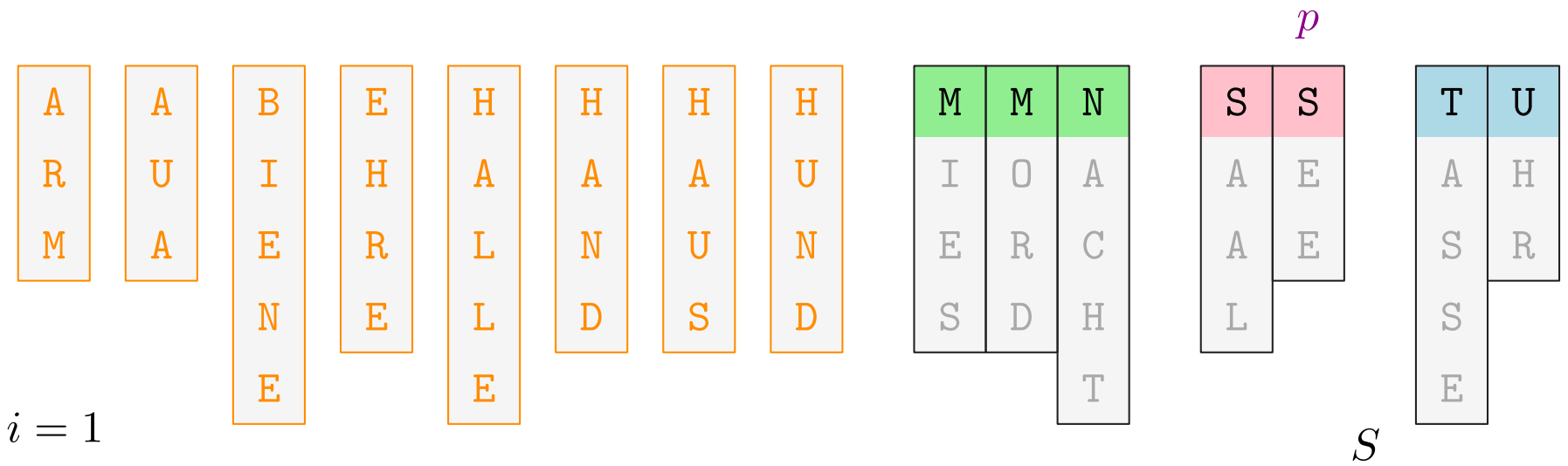
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

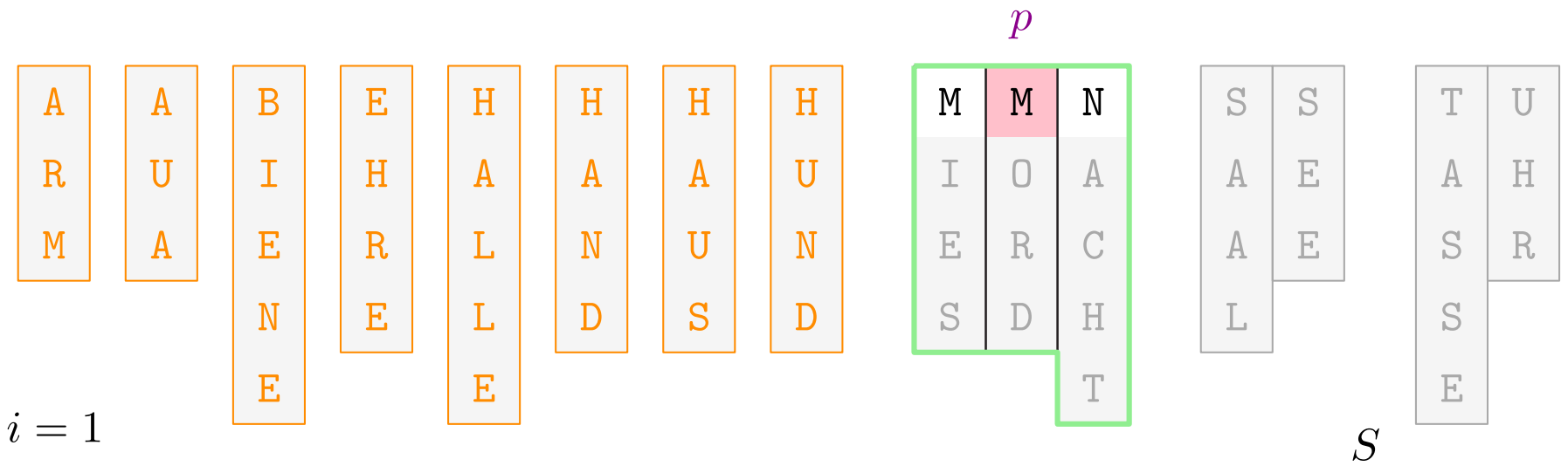
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

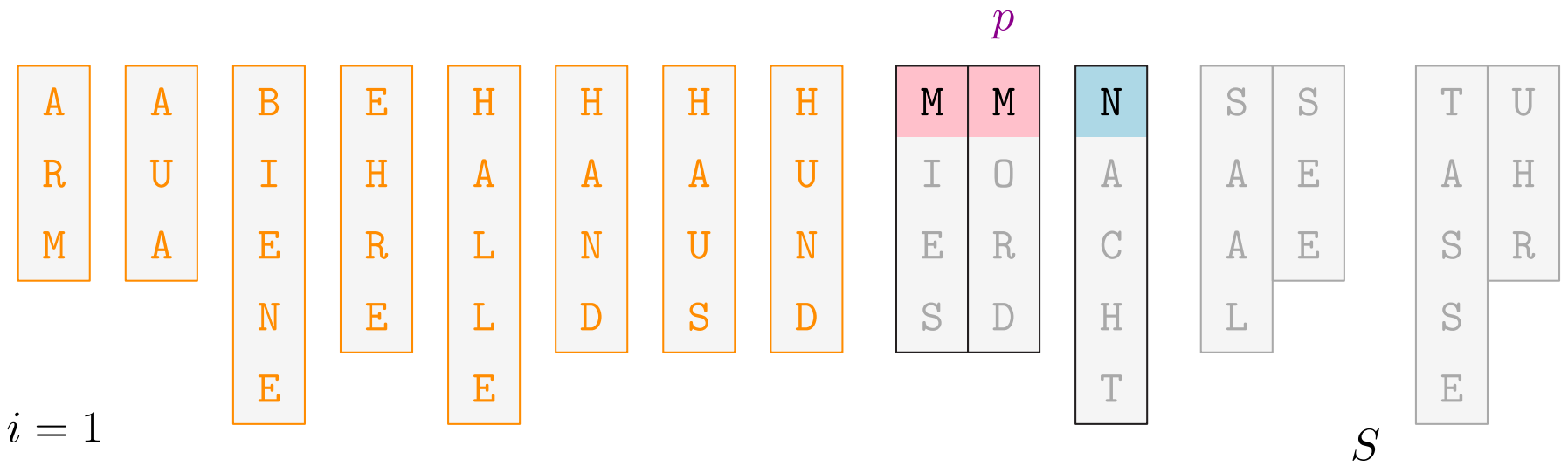
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

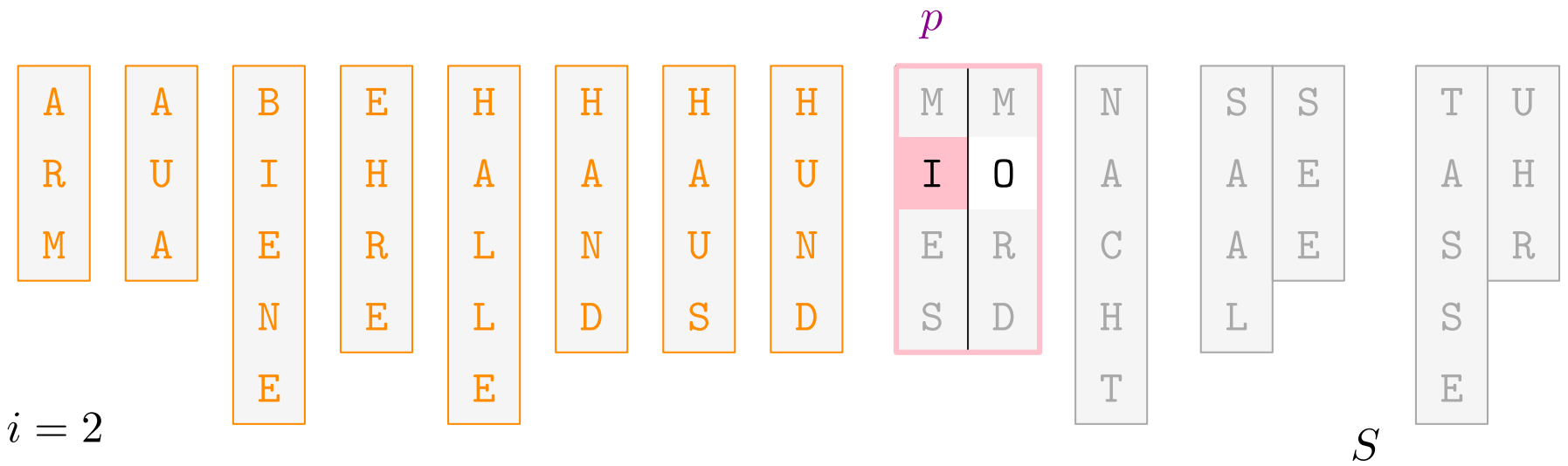
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

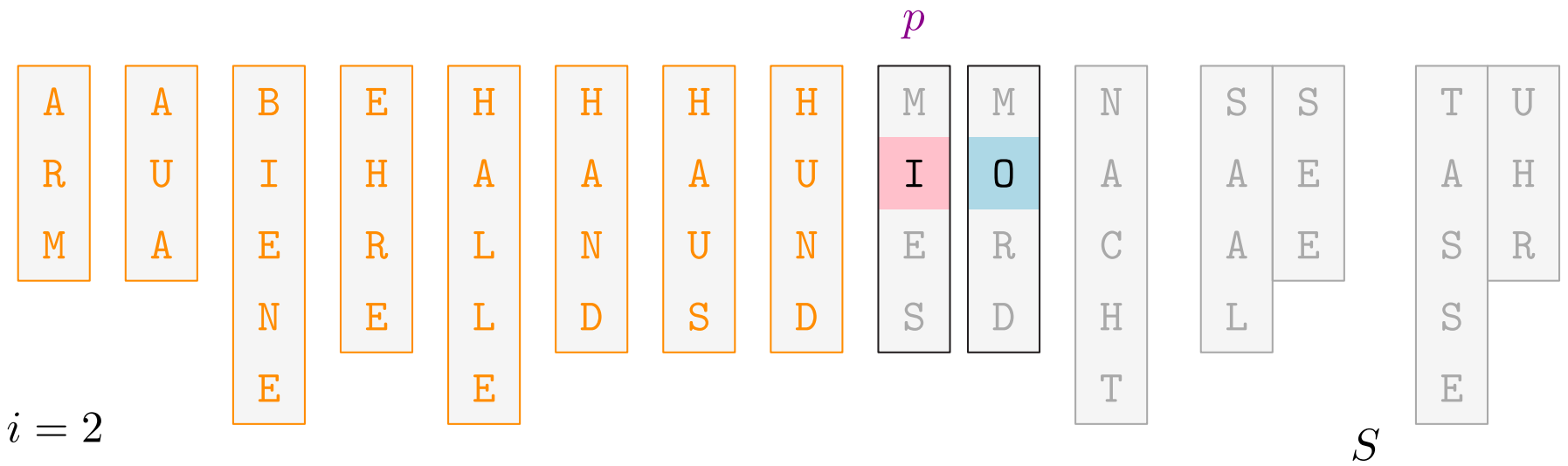
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

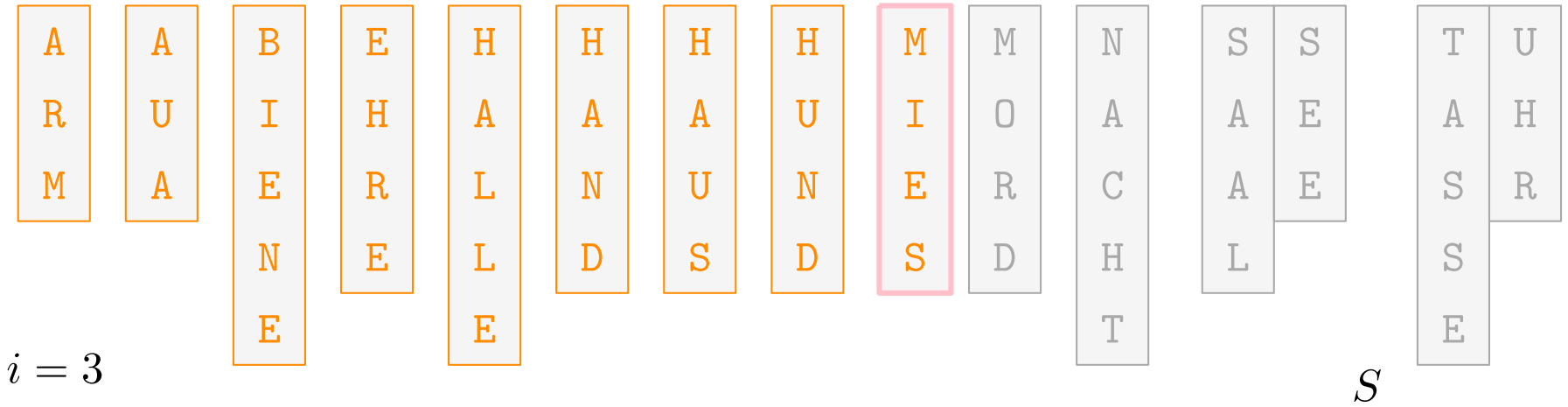
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

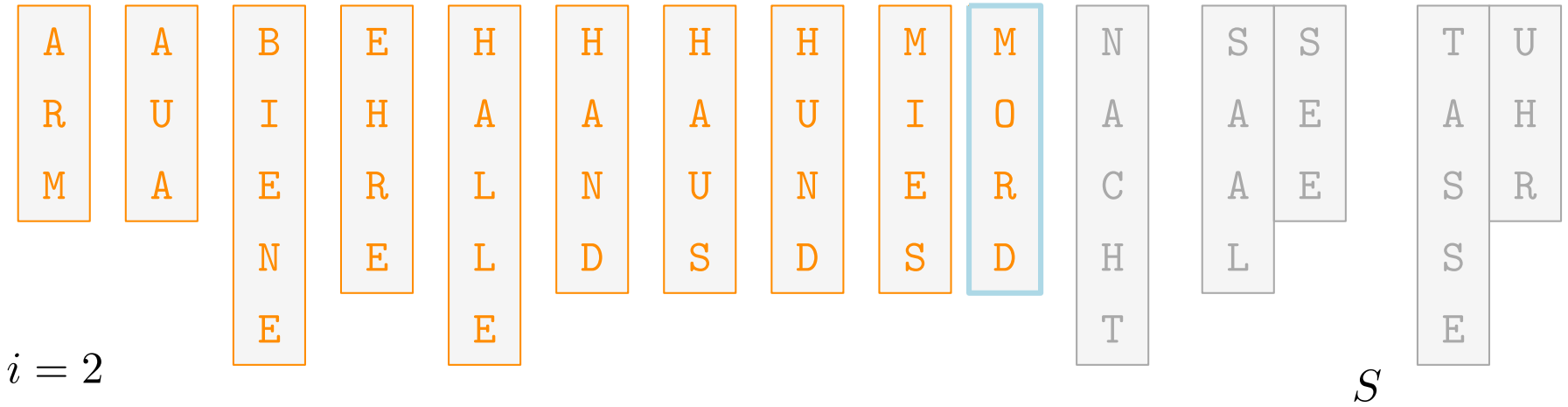
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

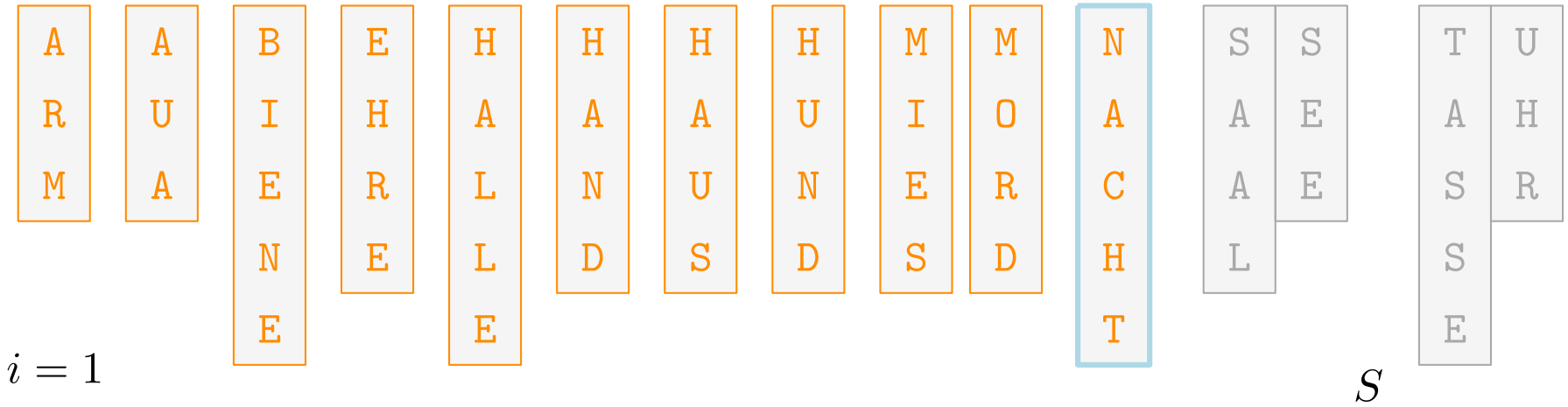
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

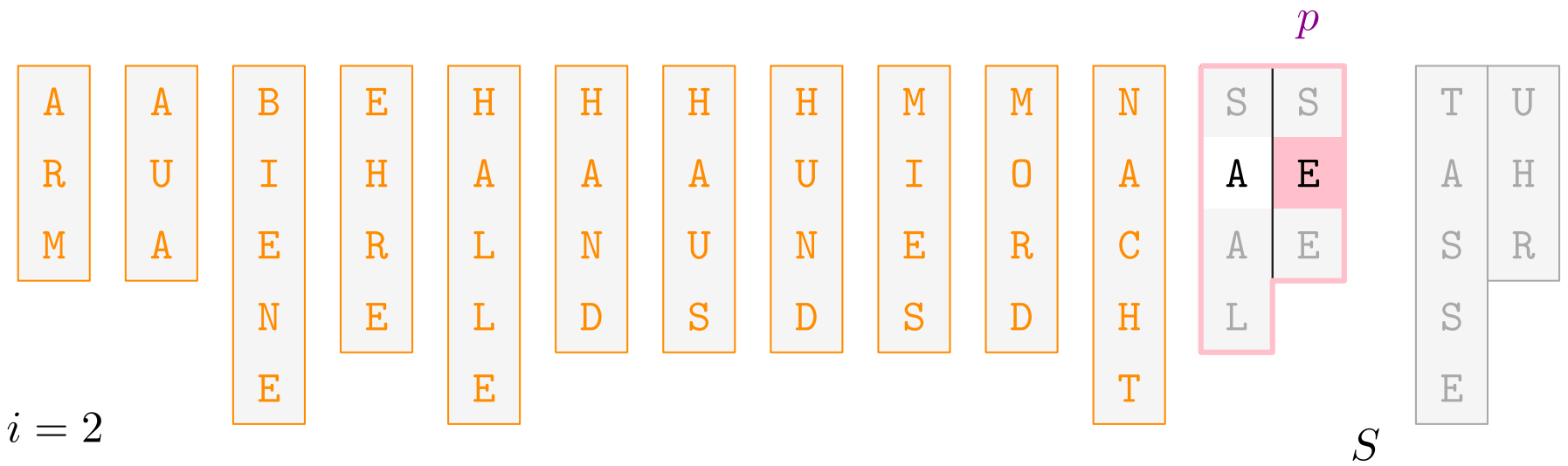
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

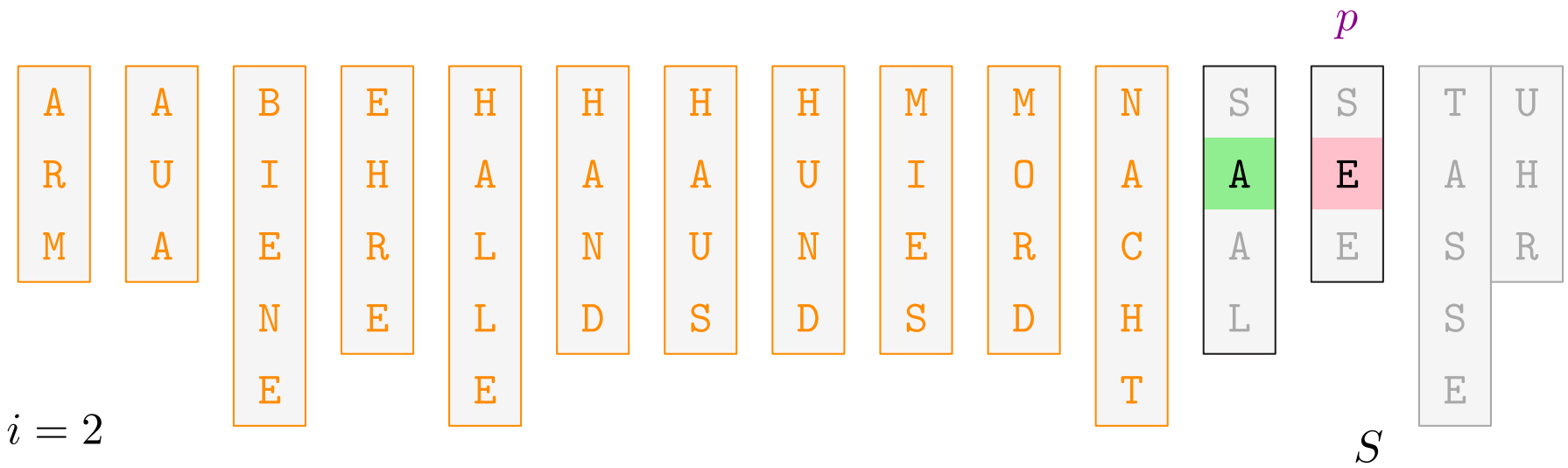
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

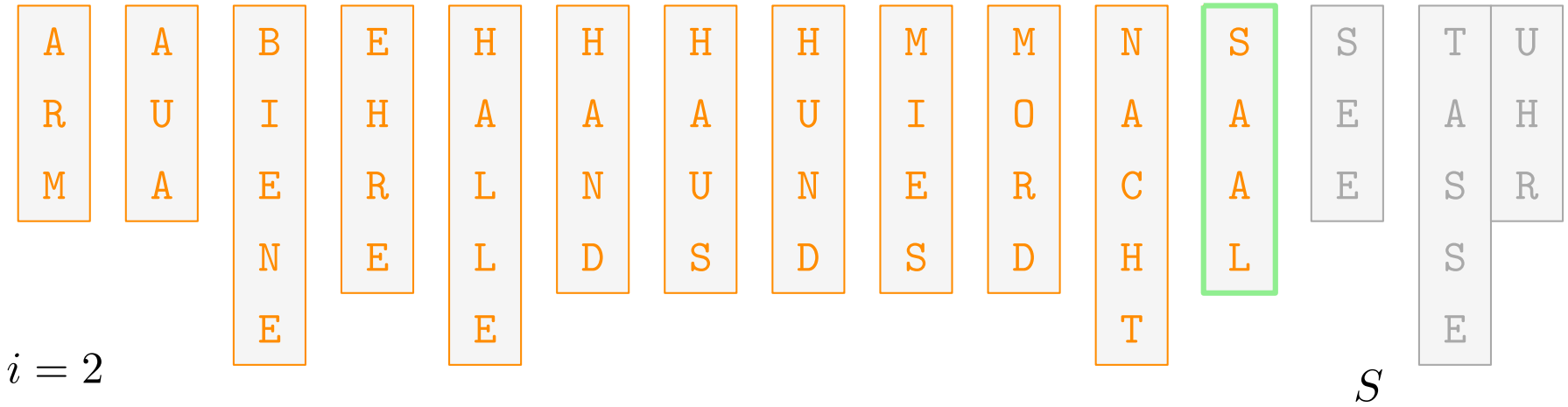
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

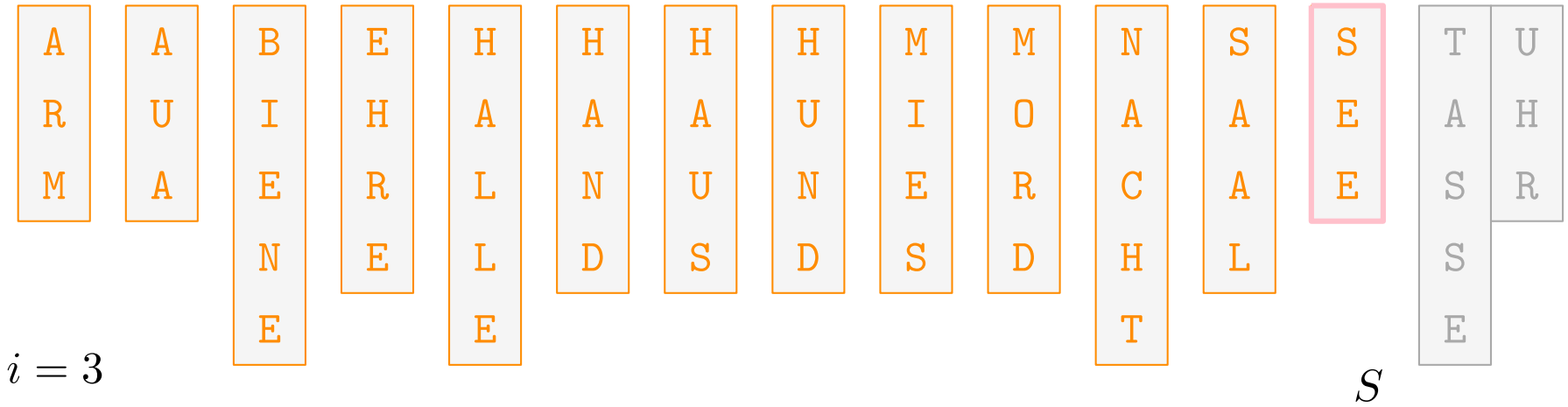
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

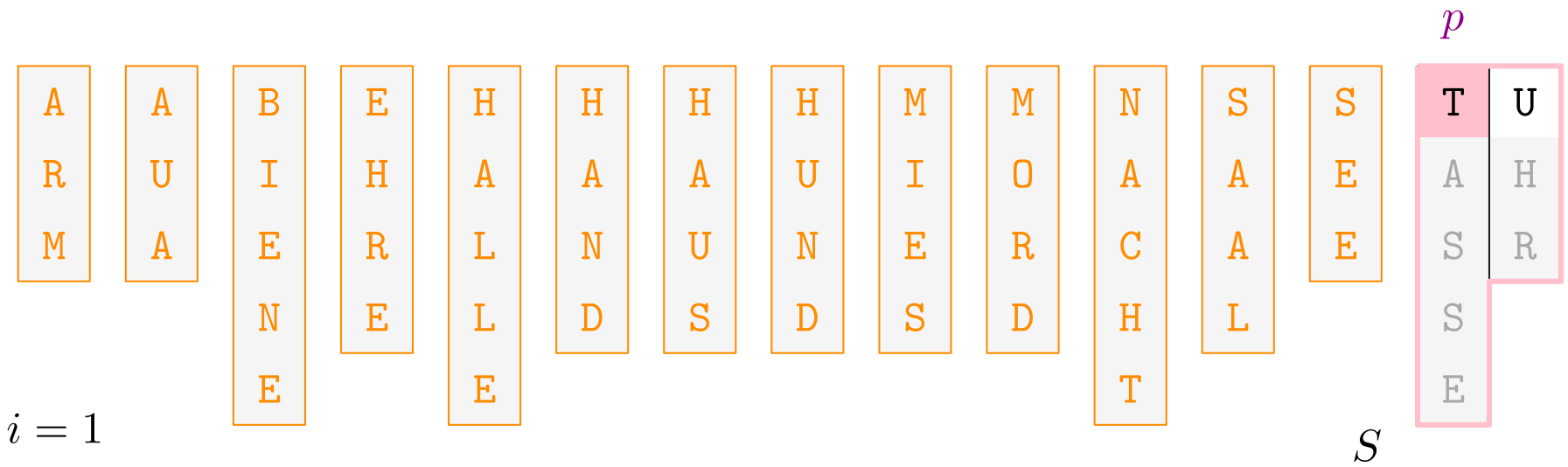
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

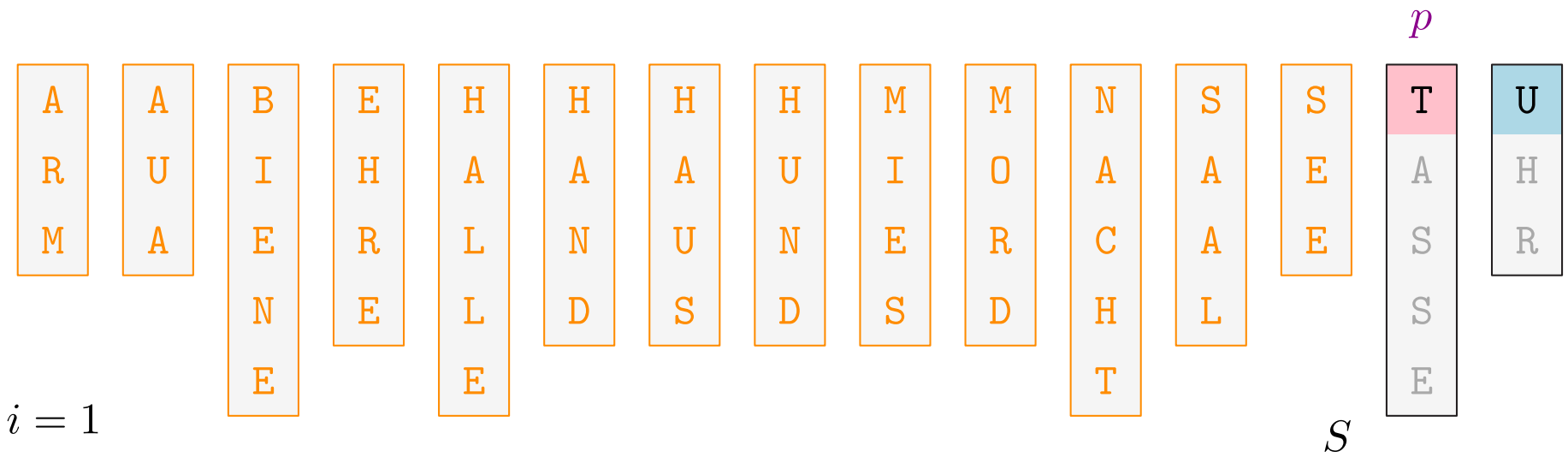
choose $p \in S$ uniformly at random (Pivotelement)

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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

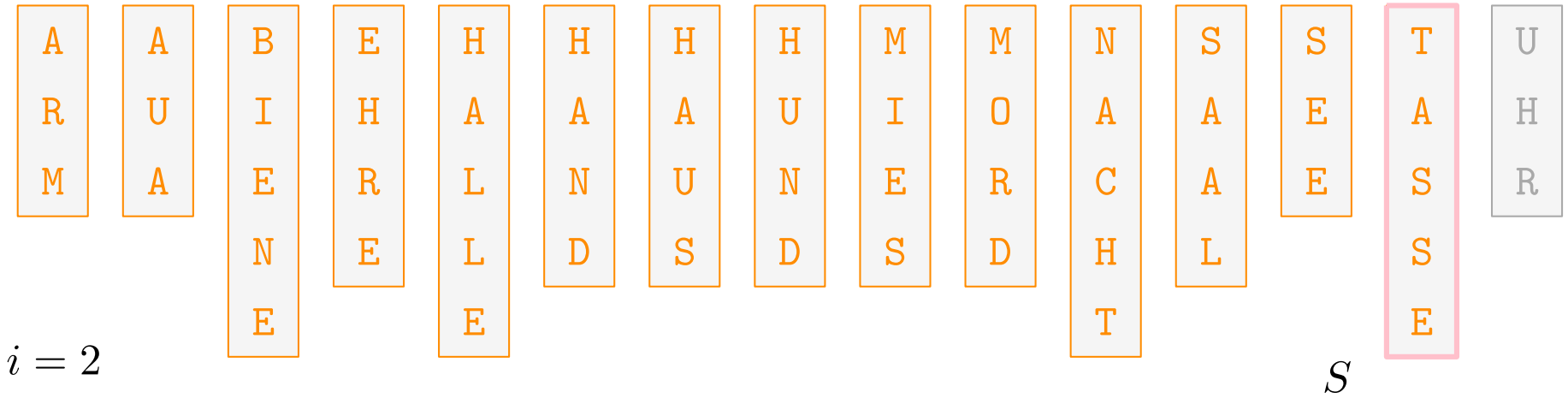
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

  if $|S| \leq 1$ **then return** S (Basisfall)

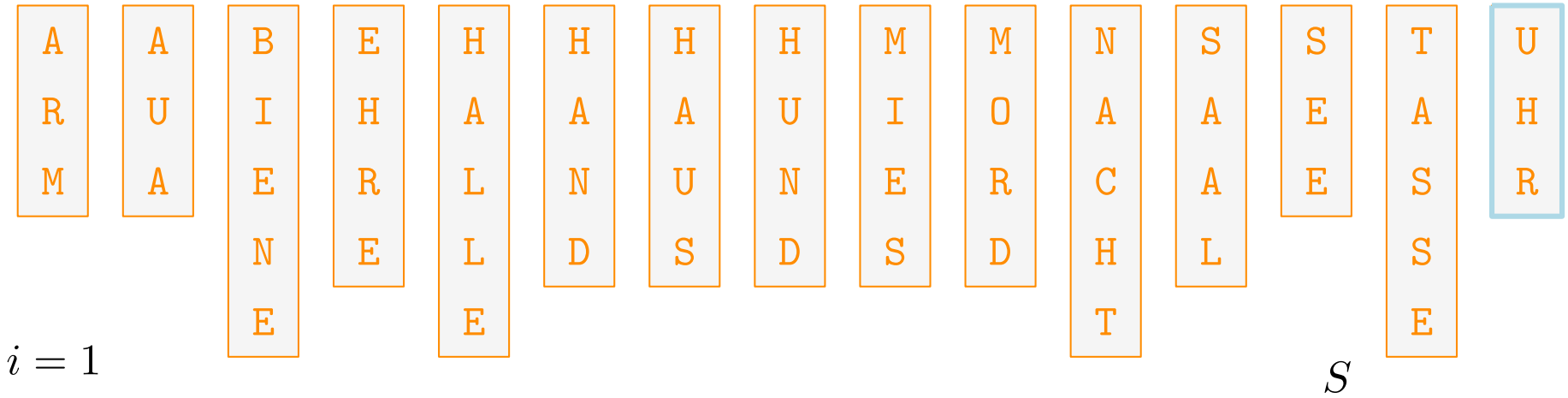
  choose $p \in S$ uniformly at random (Pivotelement)

  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

Ablauf

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

if $|S| \leq 1$ **then return** S (Basisfall)

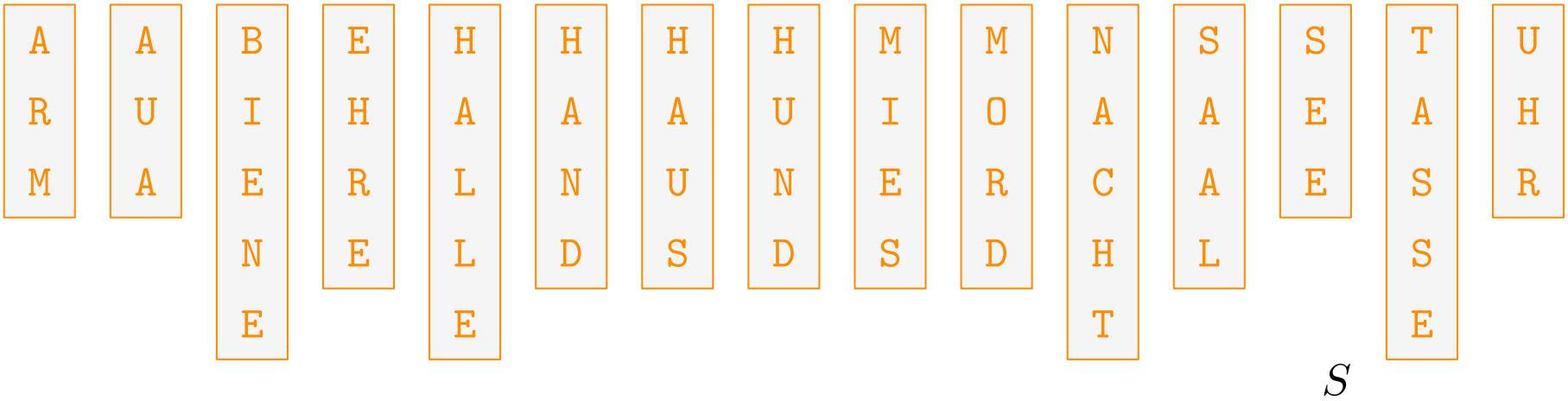
choose $p \in S$ uniformly at random (Pivotelement)

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



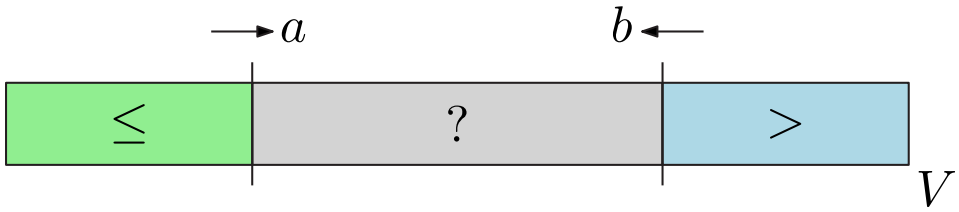
in-place Multikey Quicksort

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 $a > b$



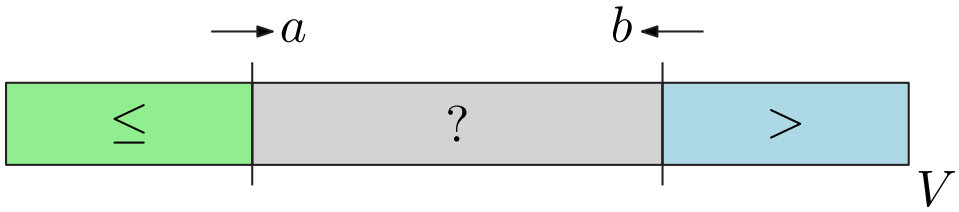
in-place Multikey Quicksort

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

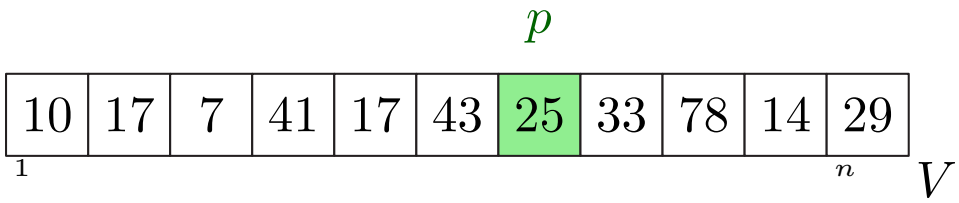
1 n V

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 $a > b$

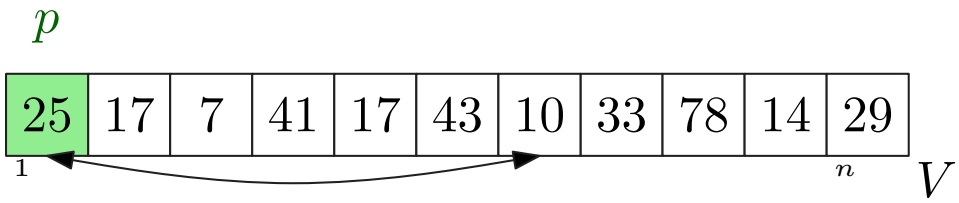


in-place Multikey Quicksort

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

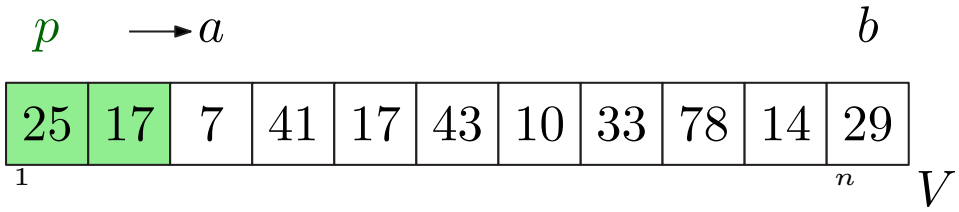
V

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 $a > b$

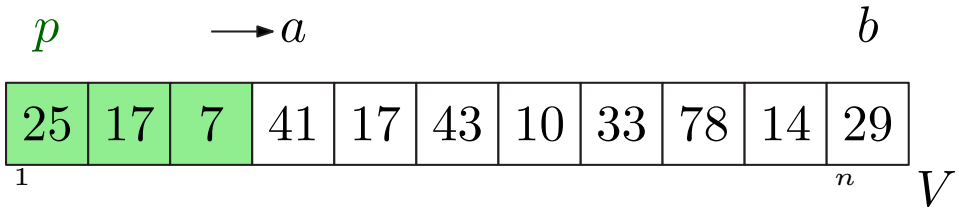


in-place Multikey Quicksort

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 $a > b$

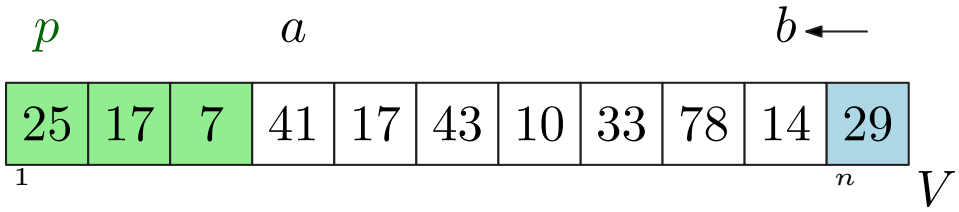


in-place Multikey Quicksort

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 $a > b$

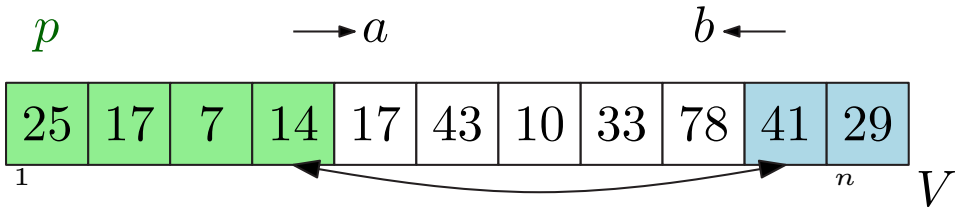


in-place Multikey Quicksort

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 $a > b$

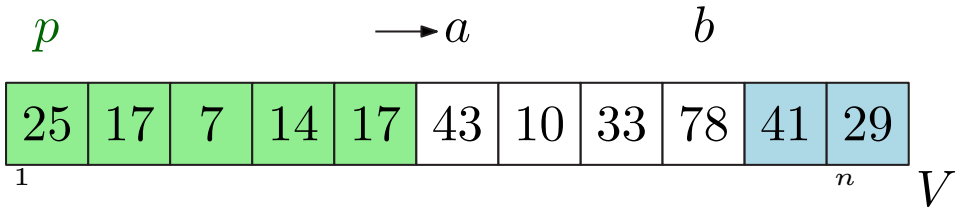


in-place Multikey Quicksort

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 $a > b$

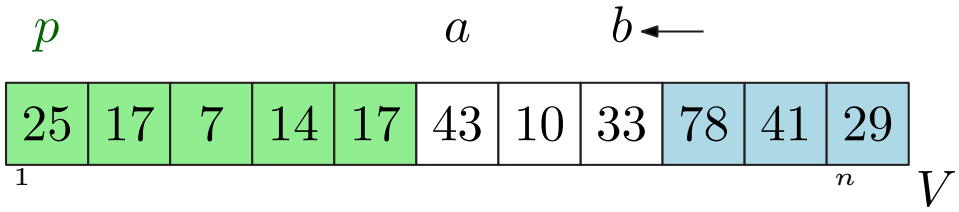


in-place Multikey Quicksort

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 $a > b$

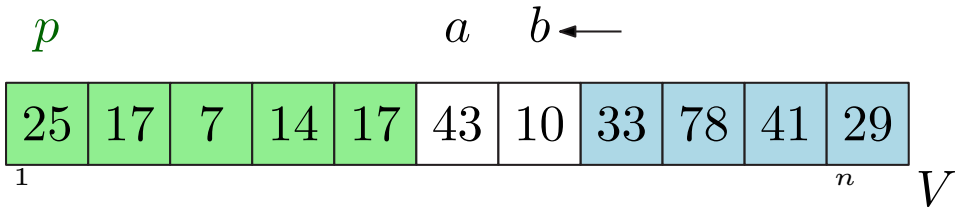


in-place Multikey Quicksort

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 $a > b$

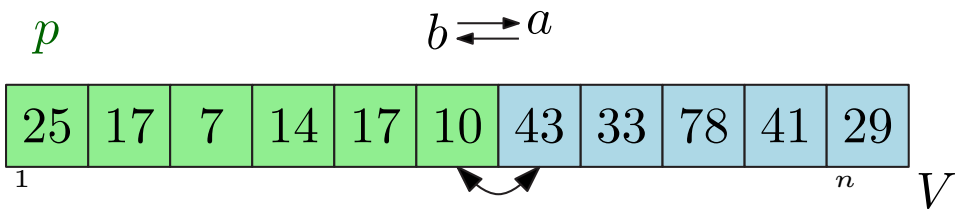


in-place Multikey Quicksort

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 $a > b$

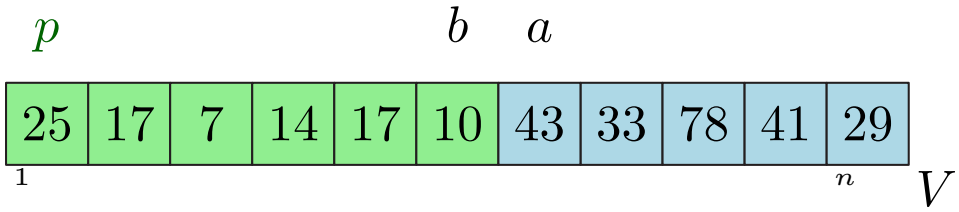


in-place Multikey Quicksort

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 $a > b$



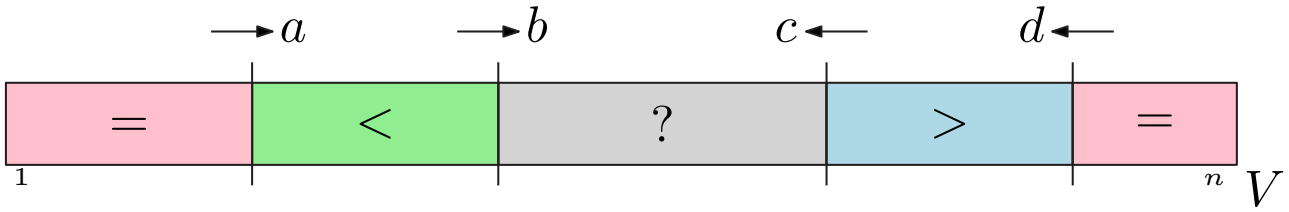
in-place Multikey Quicksort

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

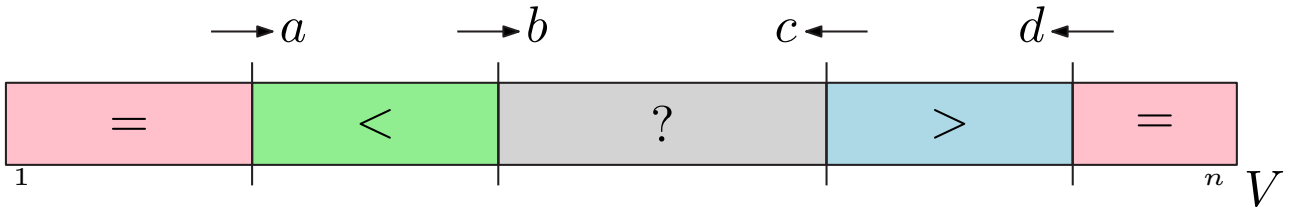


in-place Multikey Quicksort

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

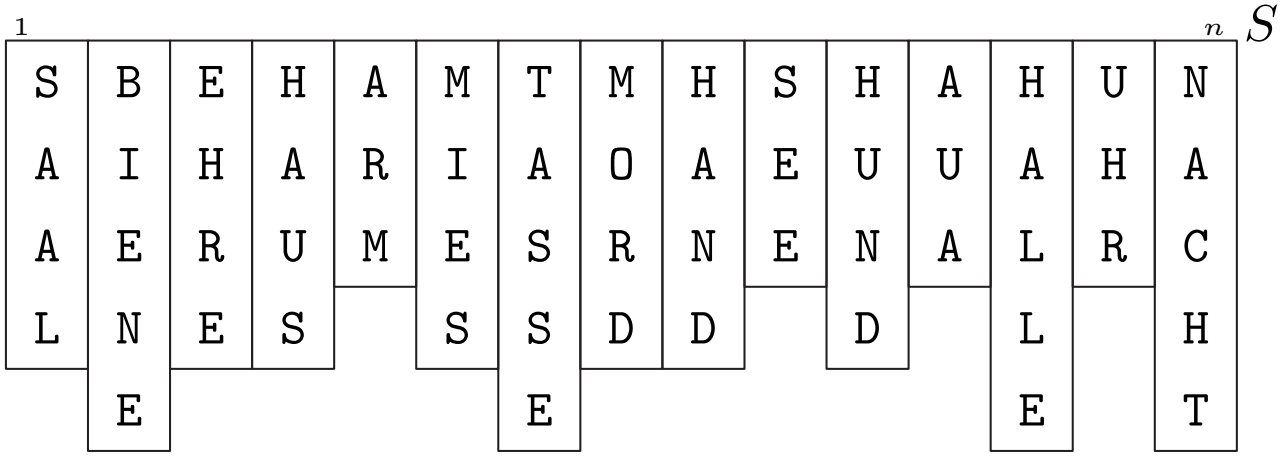


in-place Multikey Quicksort

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

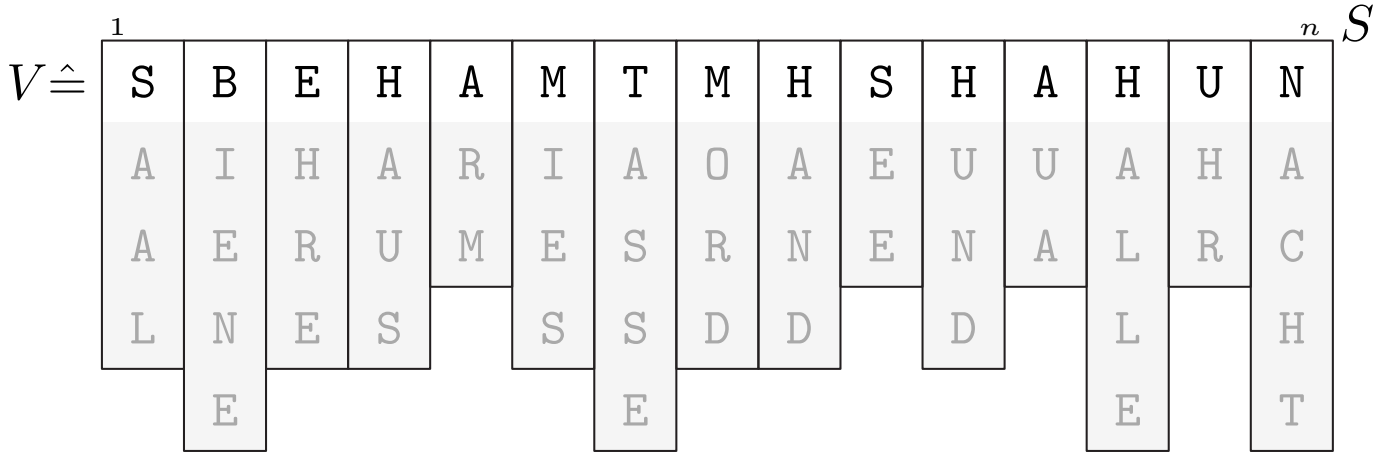


in-place Multikey Quicksort

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

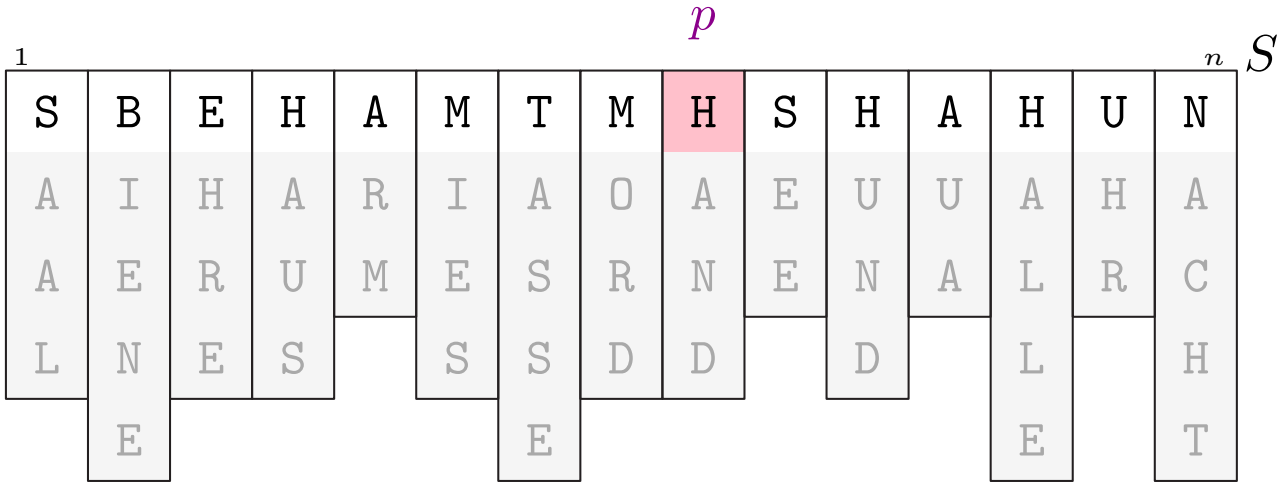


in-place Multikey Quicksort

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 $b > c$

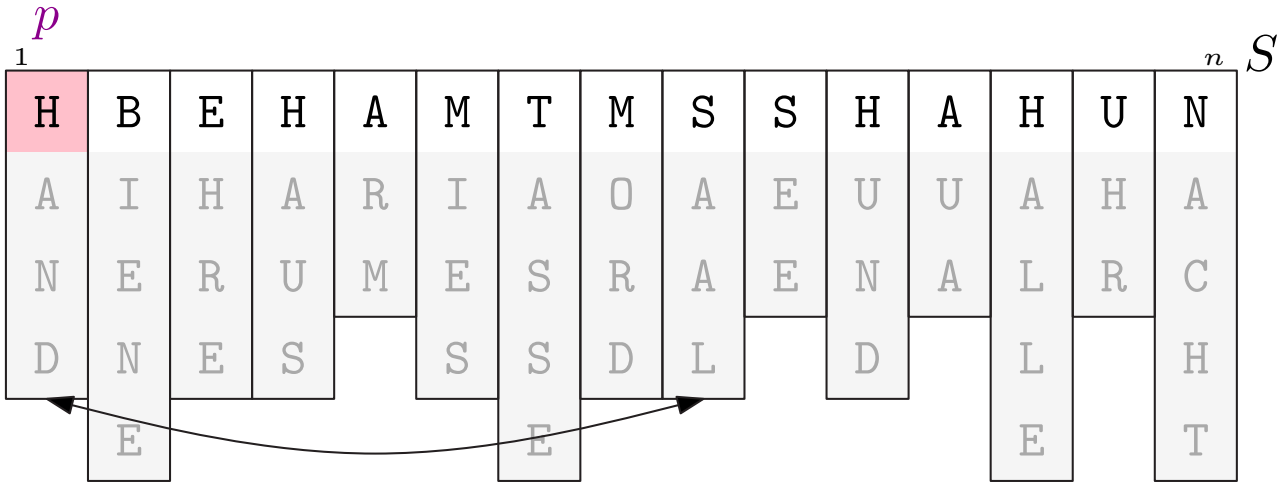


in-place Multikey Quicksort

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 $b > c$

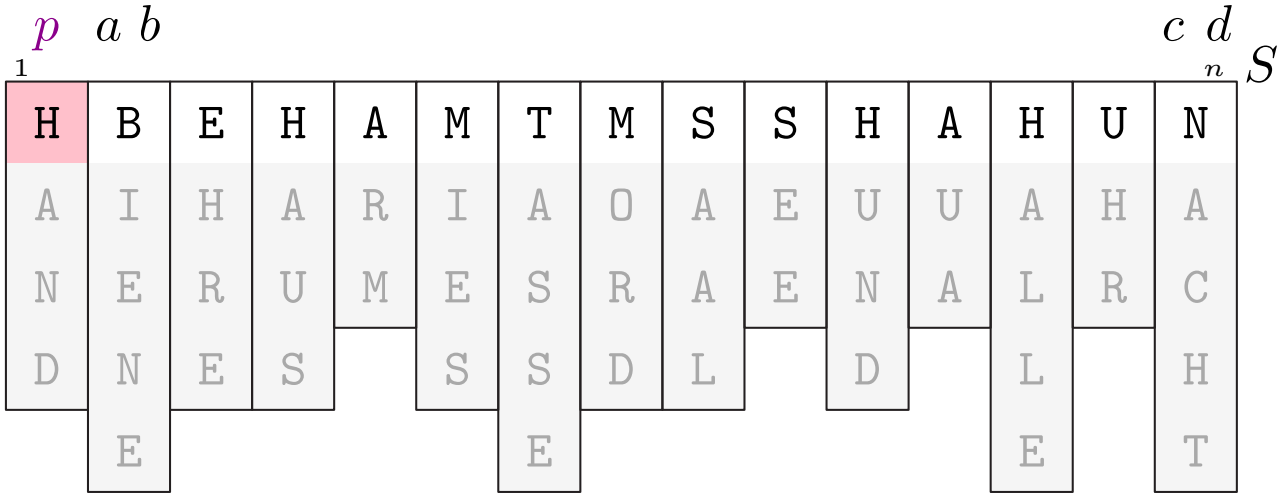


in-place Multikey Quicksort

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 $b > c$

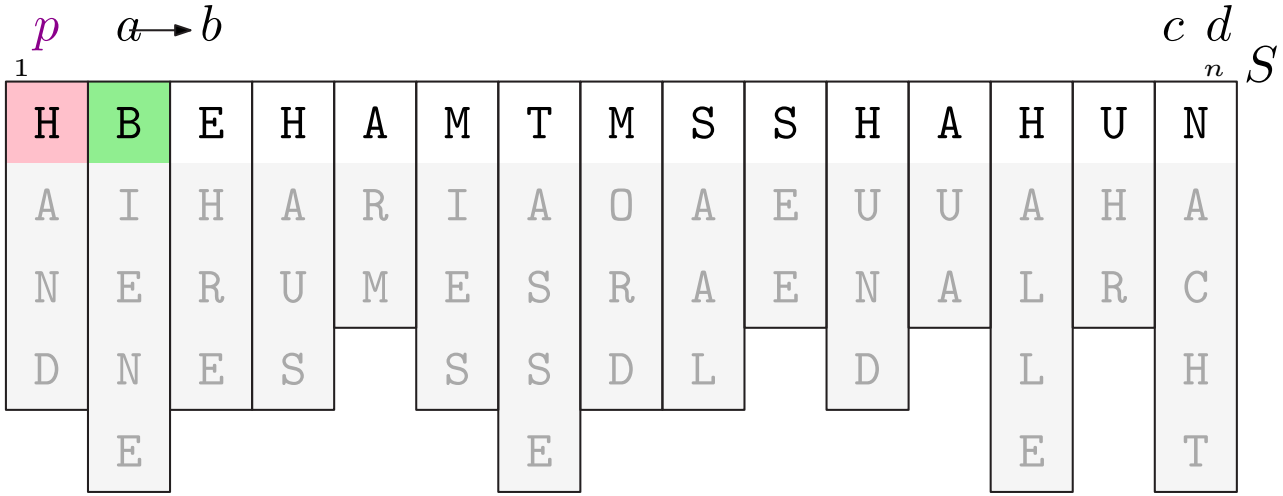


in-place Multikey Quicksort

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 $b > c$

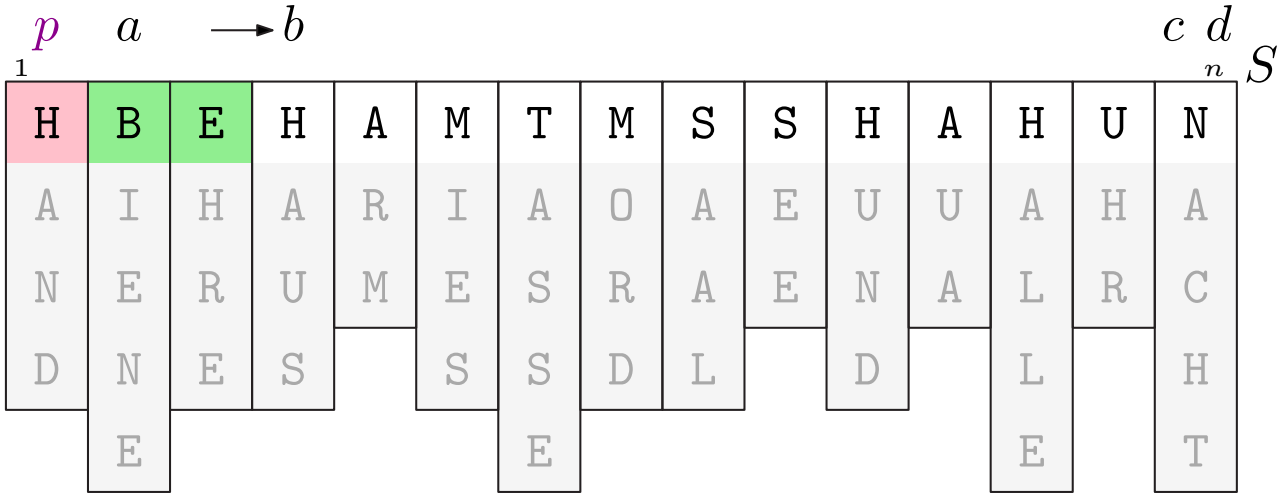


in-place Multikey Quicksort

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 $b > c$

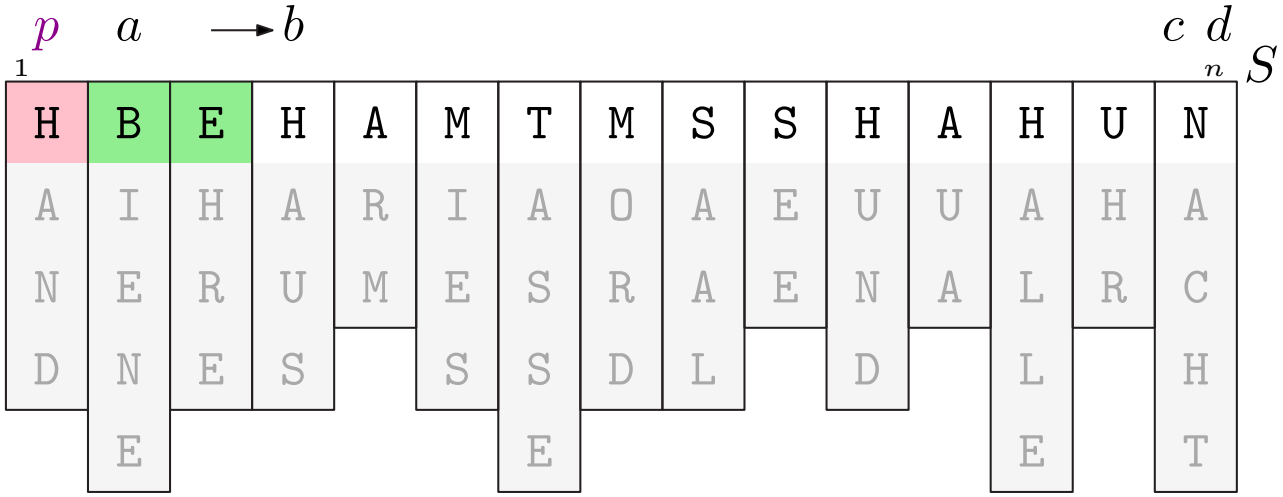


in-place Multikey Quicksort

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 $b > c$

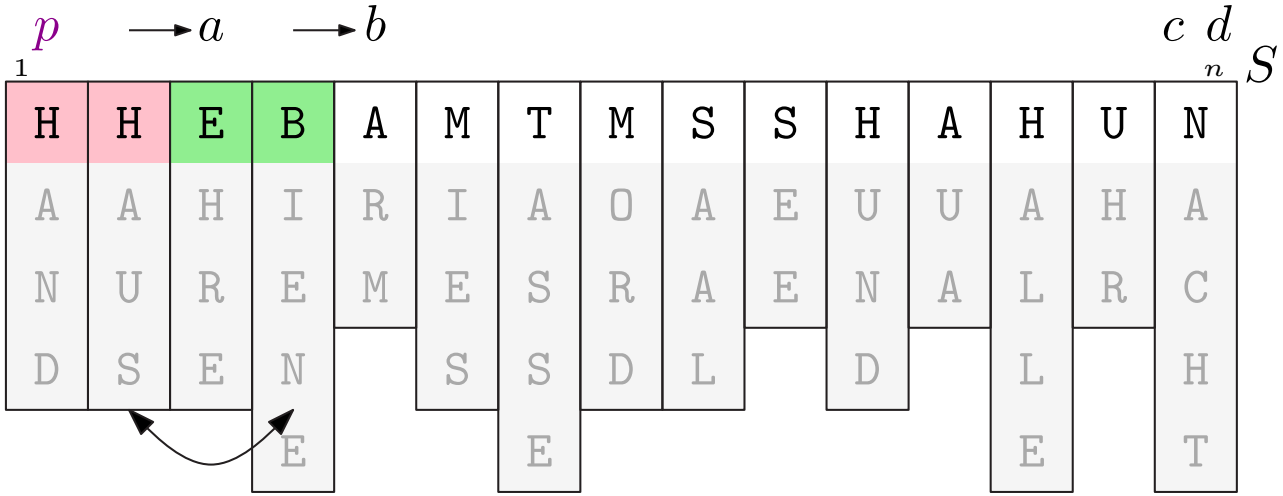


in-place Multikey Quicksort

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 $b > c$

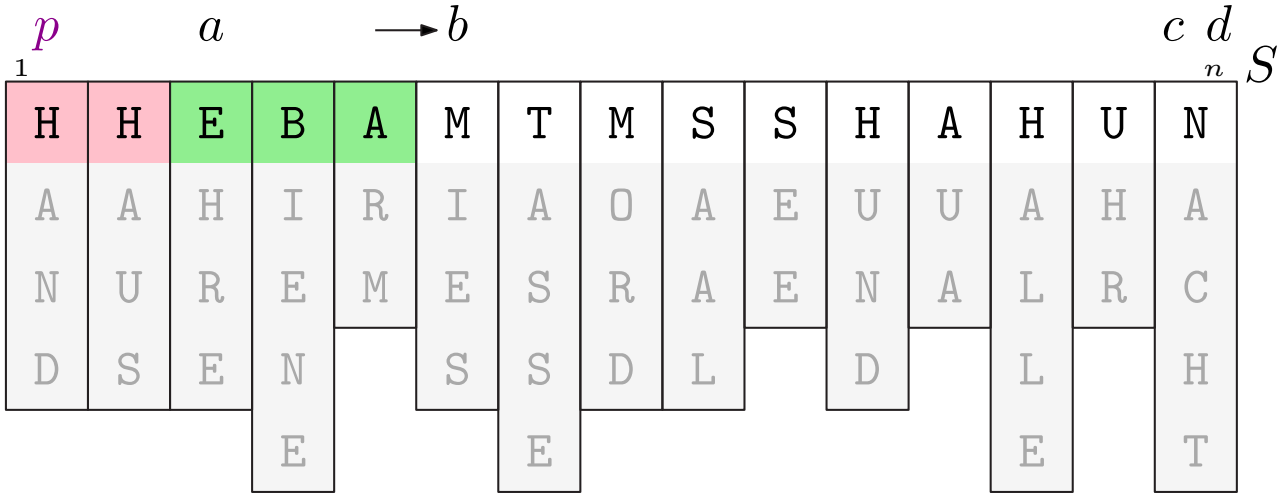


in-place Multikey Quicksort

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 $b > c$

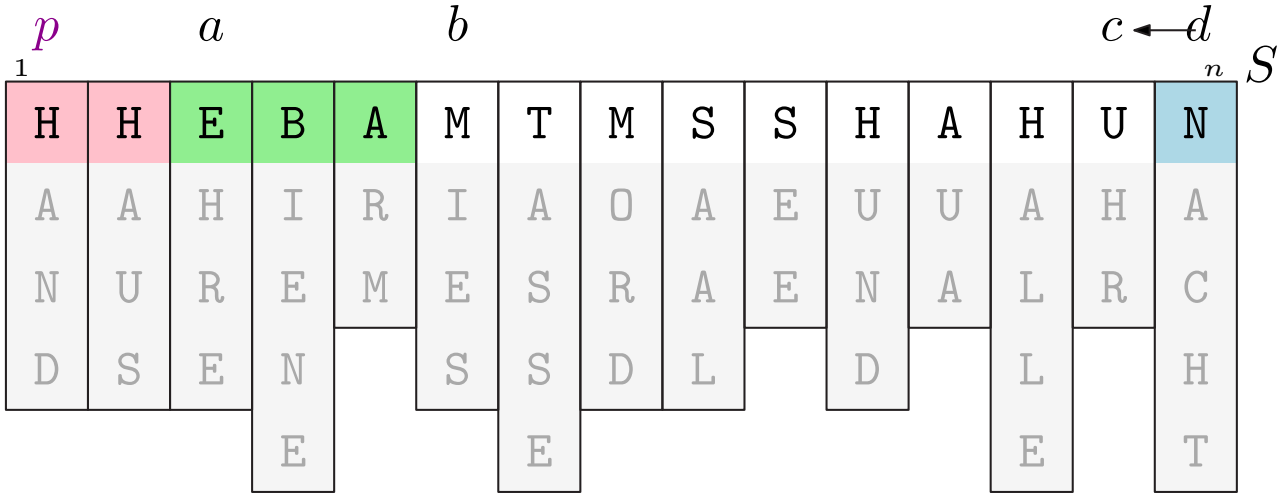


in-place Multikey Quicksort

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 $b > c$

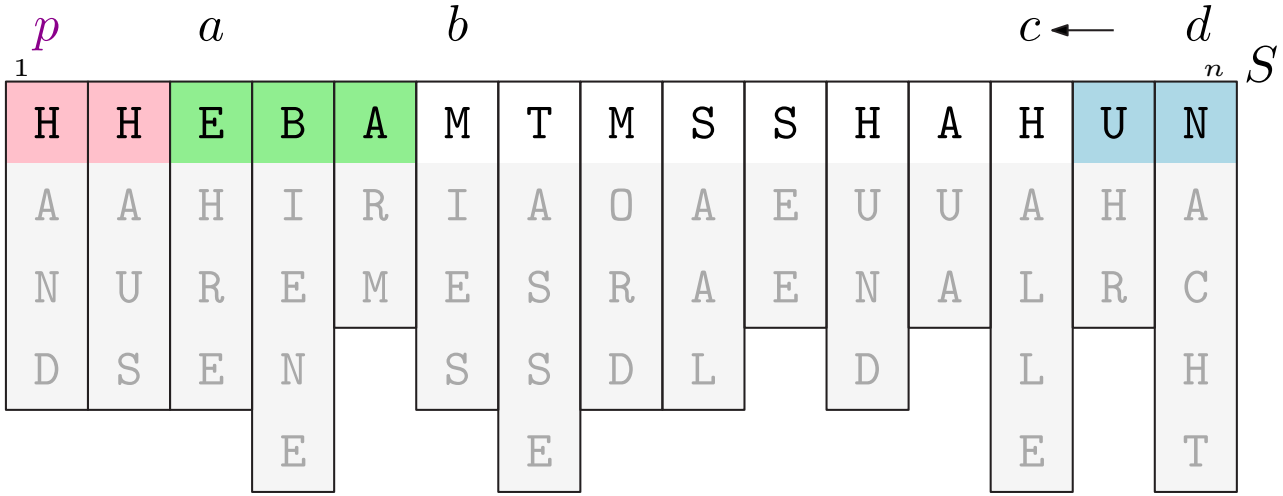


in-place Multikey Quicksort

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 $b > c$

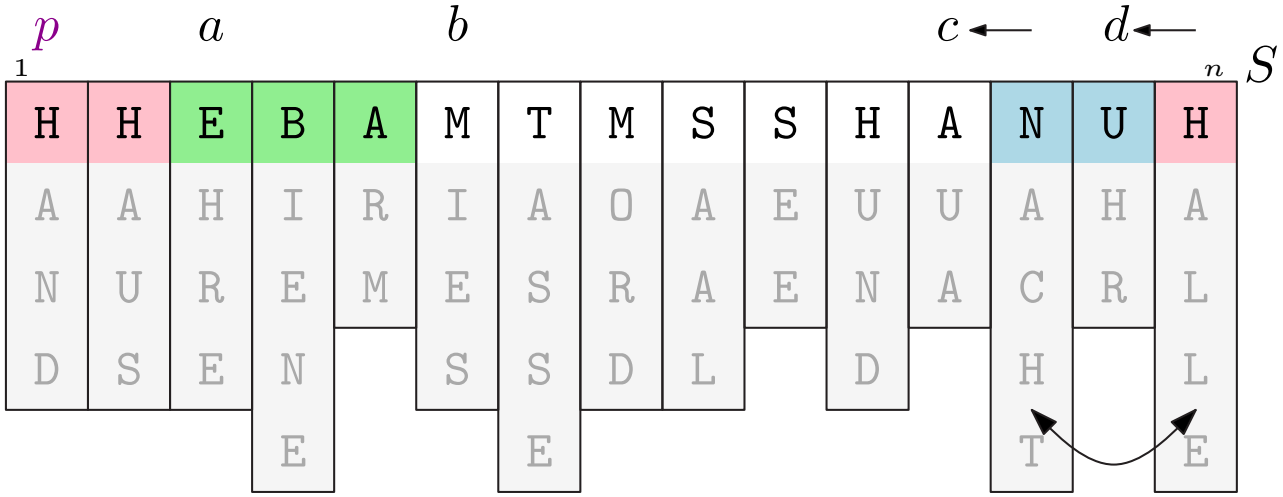


in-place Multikey Quicksort

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 $b > c$

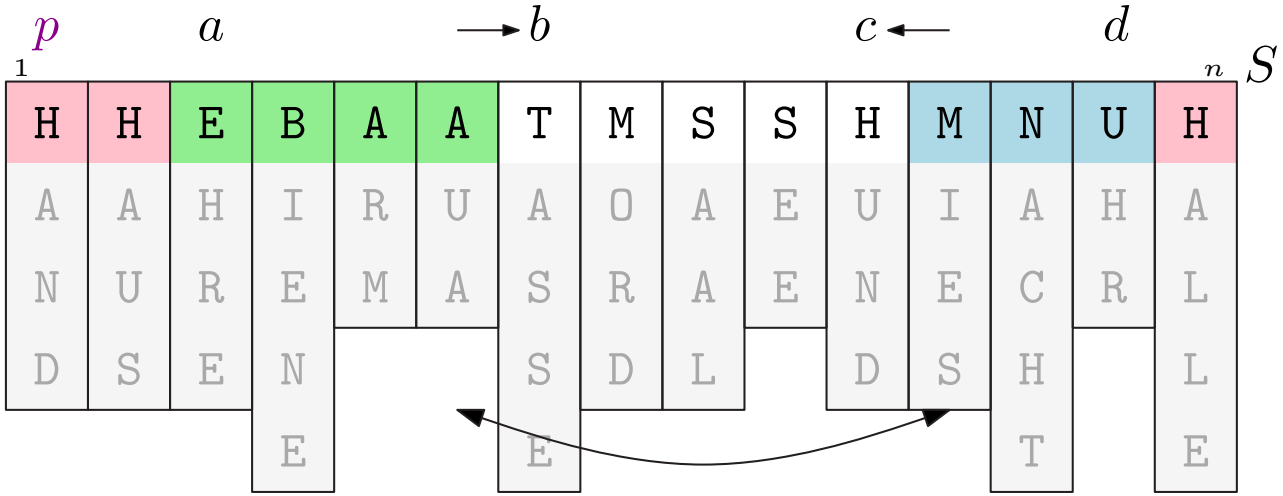


in-place Multikey Quicksort

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 $b > c$

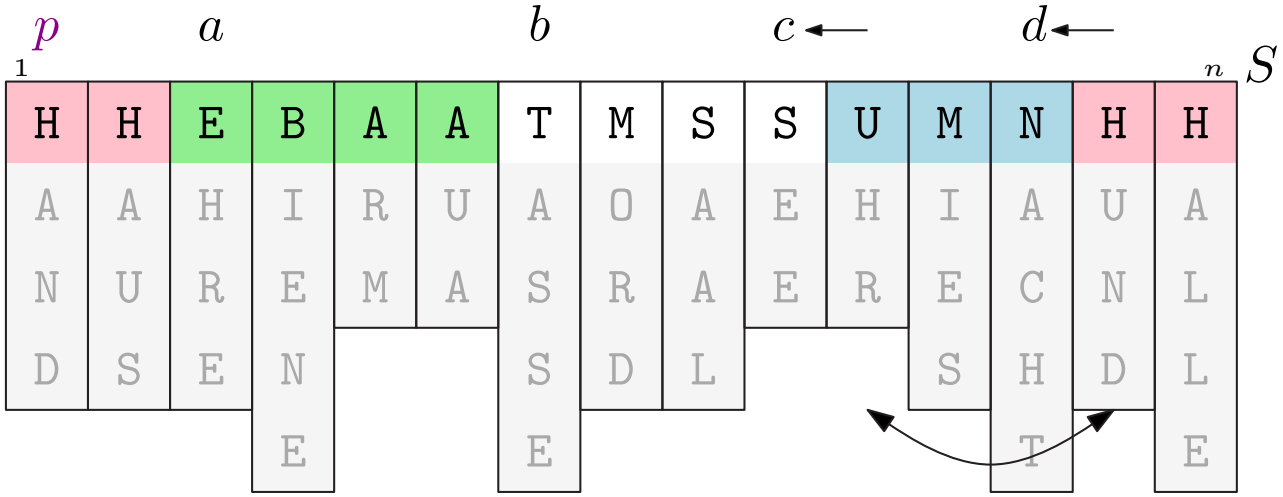


in-place Multikey Quicksort

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 $b > c$

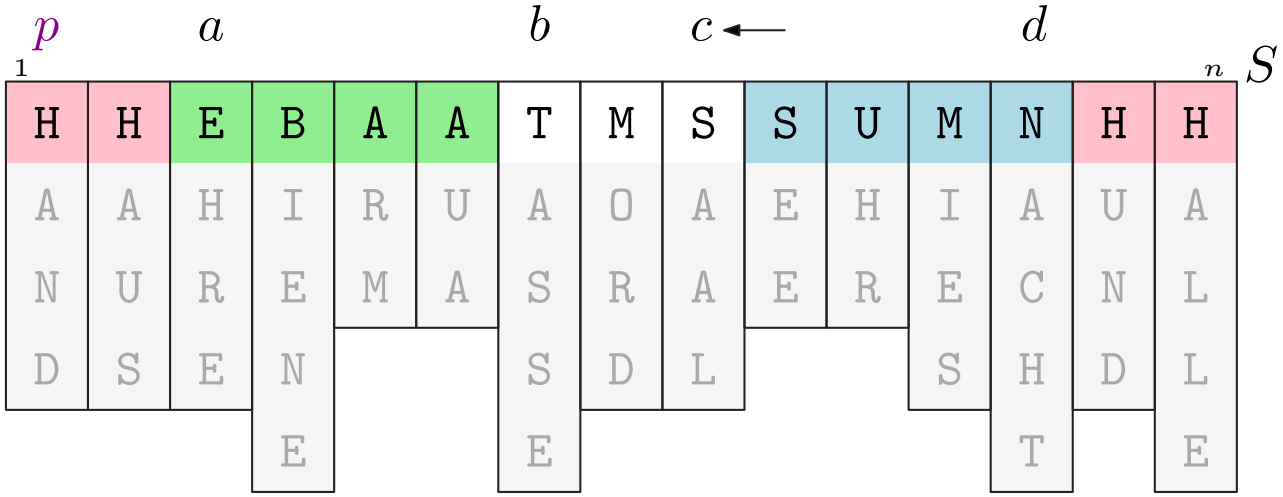


in-place Multikey Quicksort

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 $b > c$

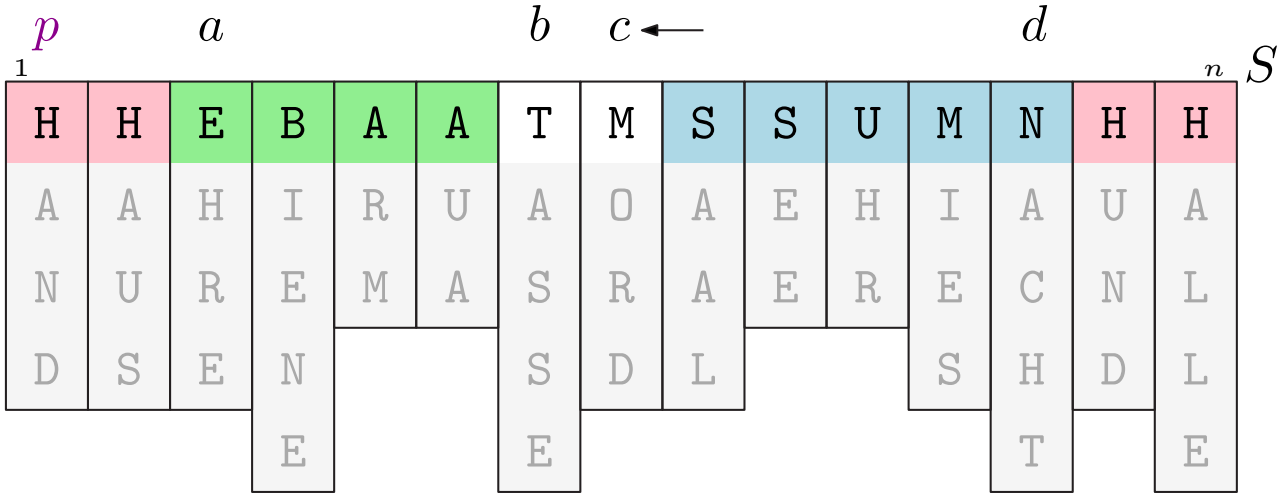


in-place Multikey Quicksort

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 $b > c$

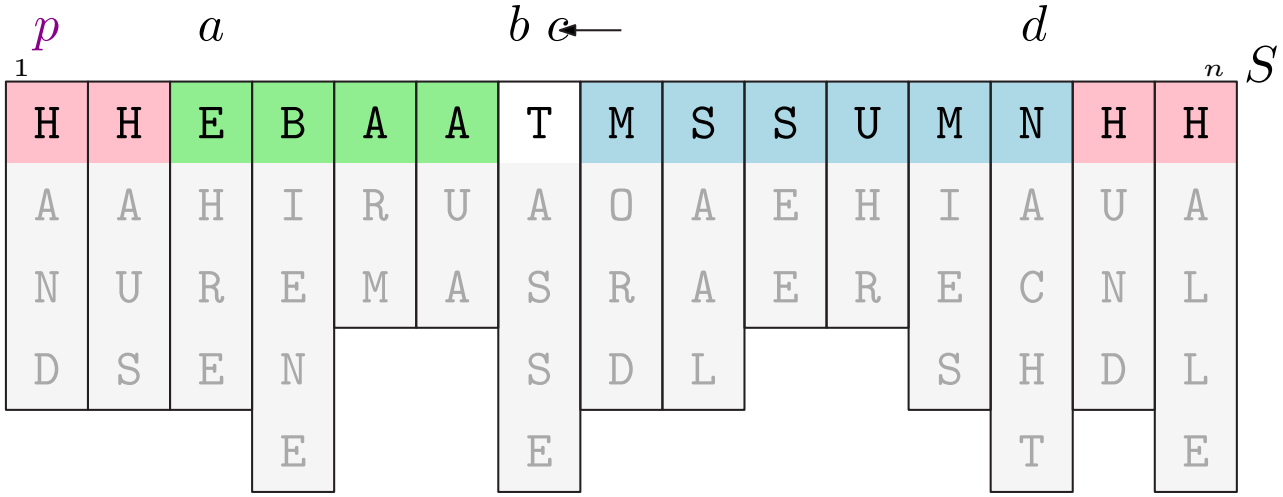


in-place Multikey Quicksort

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 $b > c$

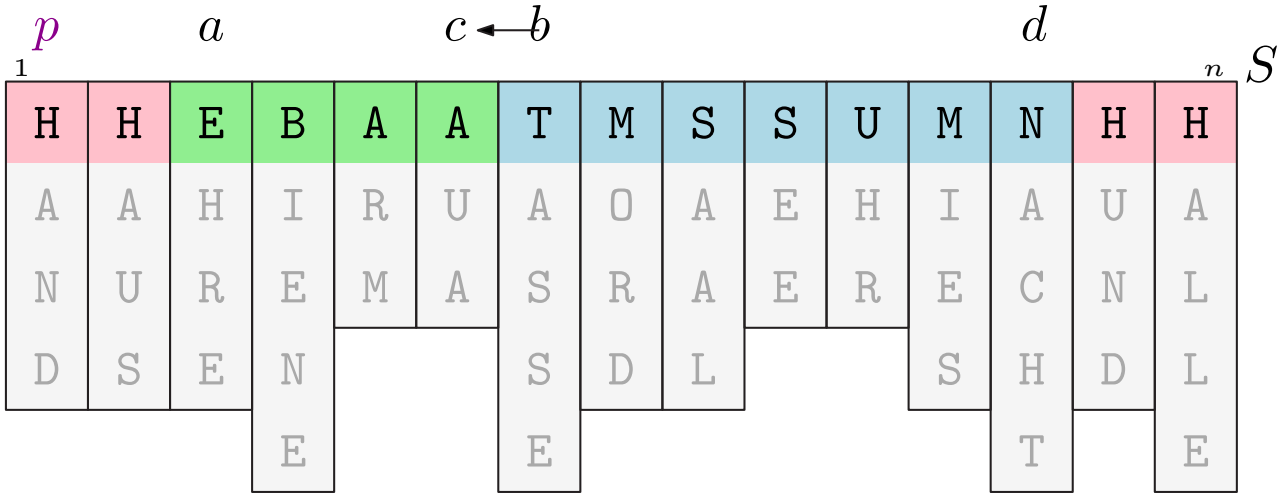


in-place Multikey Quicksort

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 $b > c$

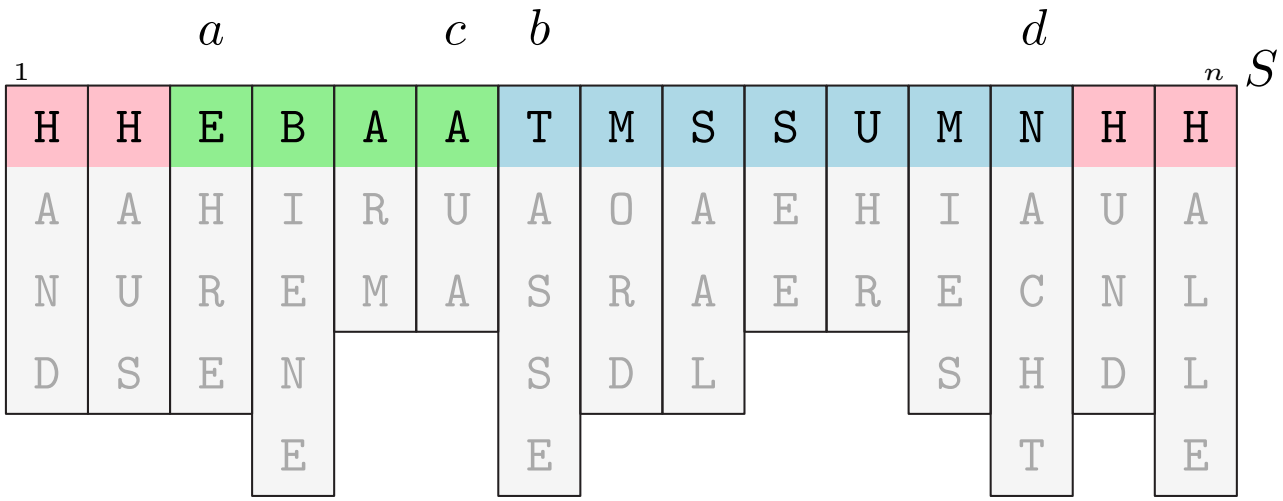


in-place Multikey Quicksort

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

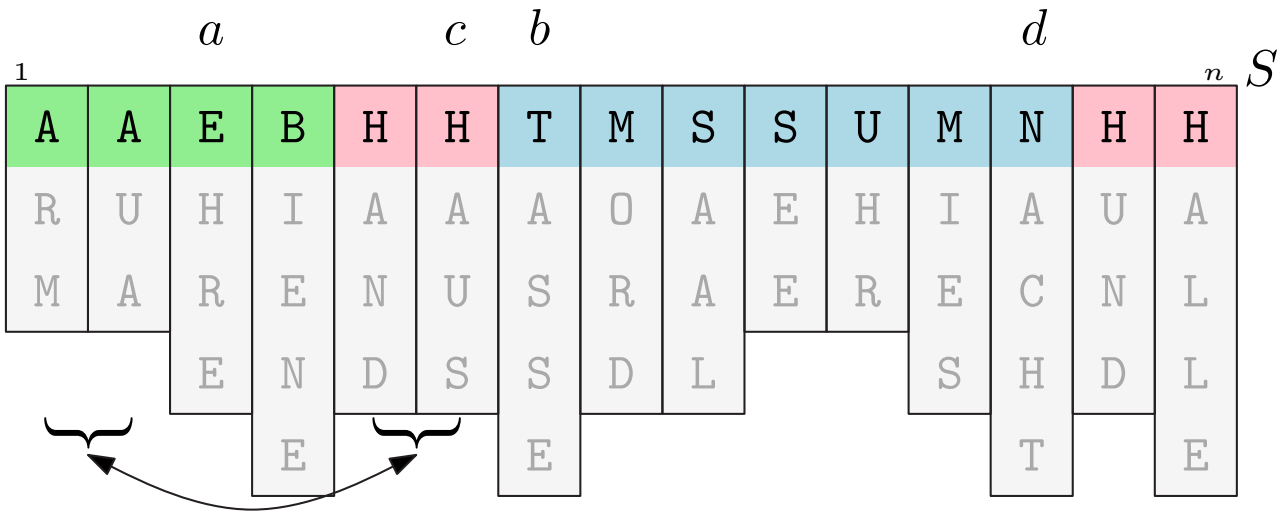


in-place Multikey Quicksort

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

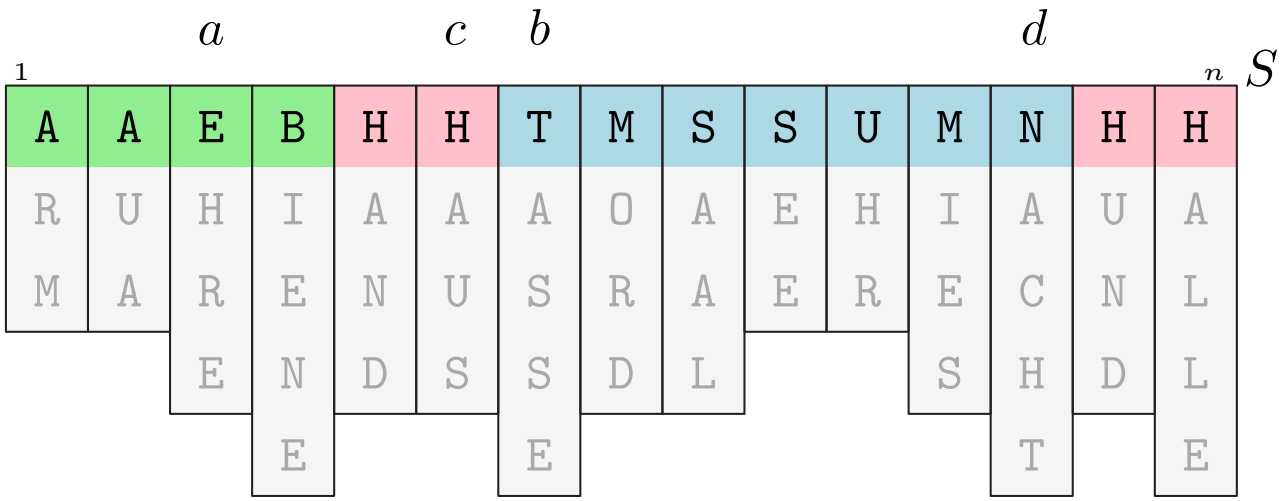


in-place Multikey Quicksort

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

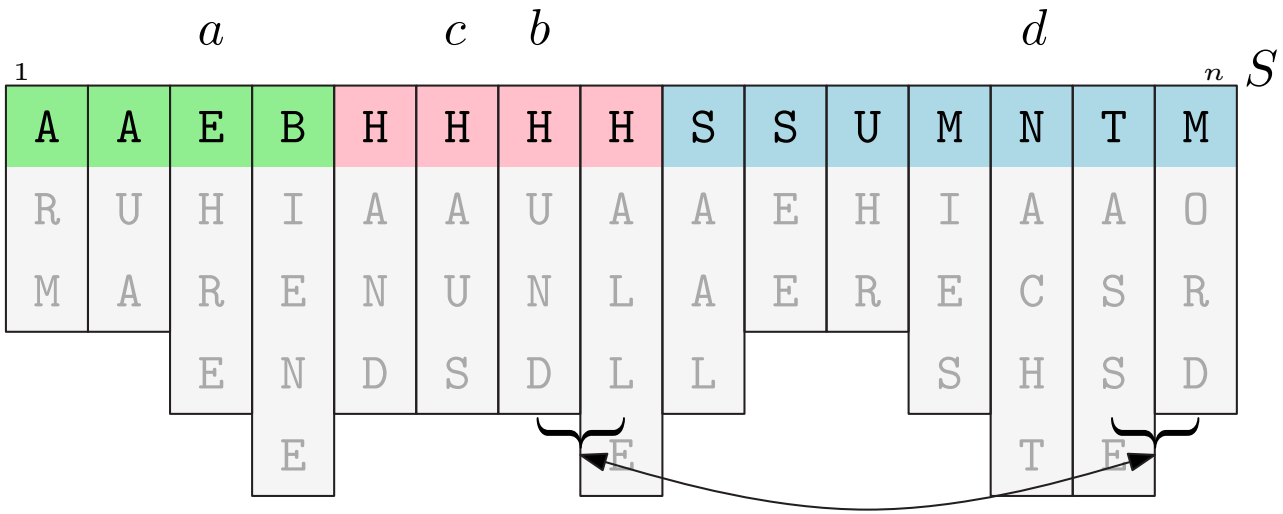


in-place Multikey Quicksort

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

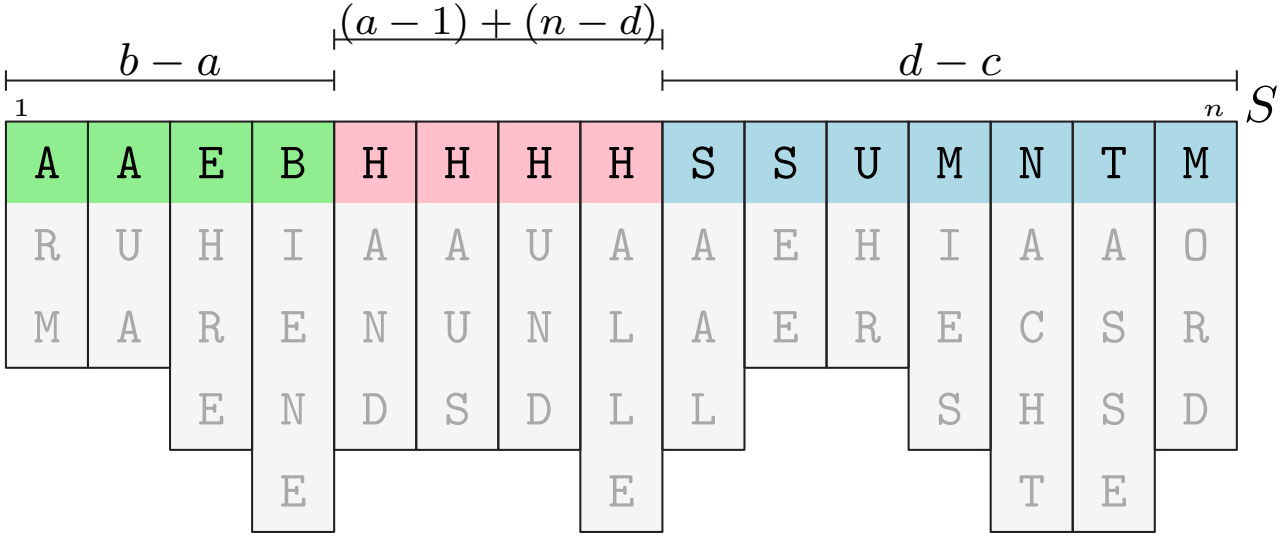


in-place Multikey Quicksort

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

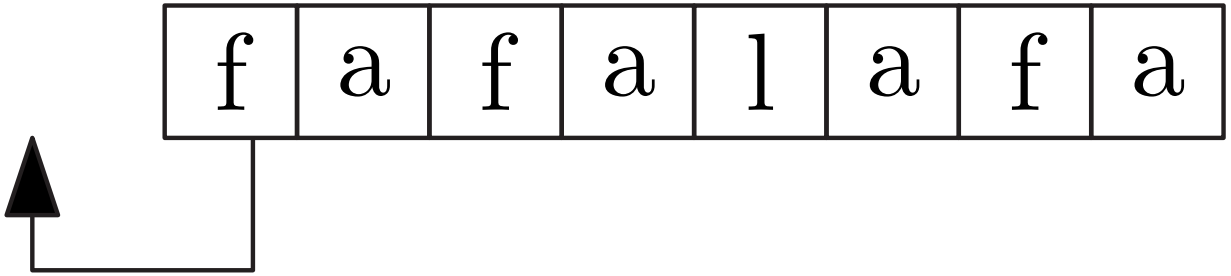
Zusammenfassung

- *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
  (nur 40 Zeilen Quellcode, siehe Anhang)

KMP Algorithmus

Border Array

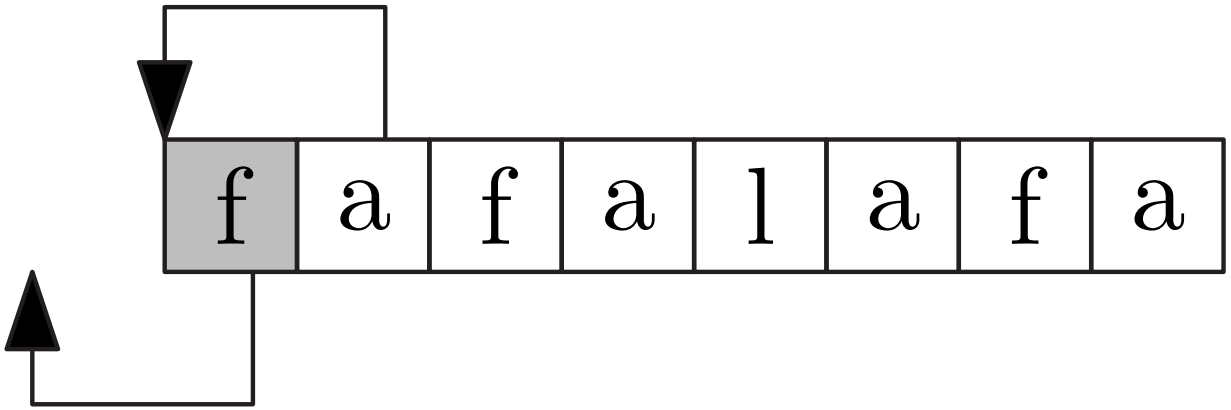
- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



KMP Algorithmus

Border Array

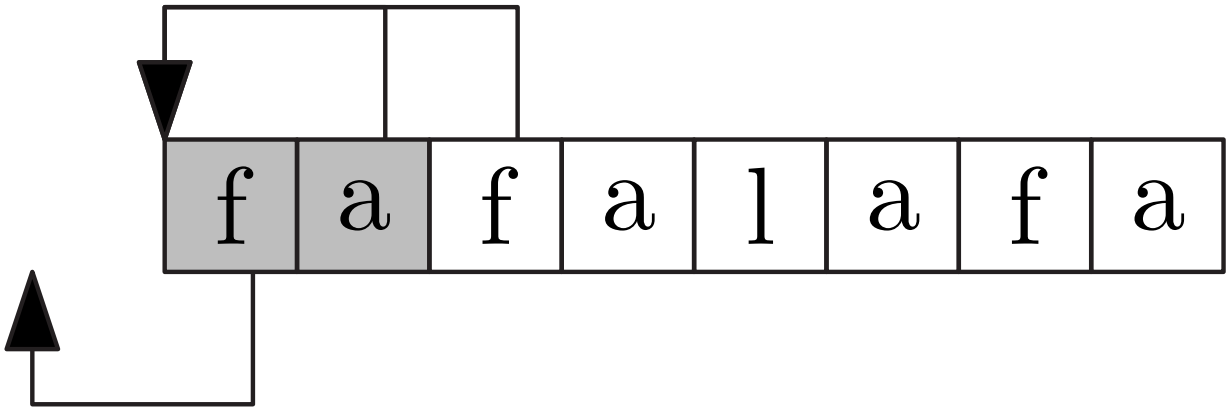
- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



KMP Algorithmus

Border Array

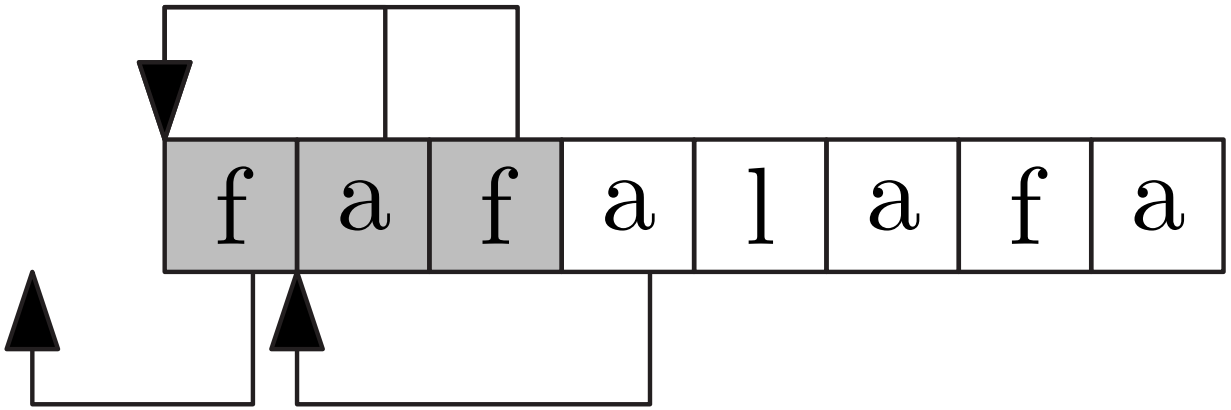
- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



KMP Algorithmus

Border Array

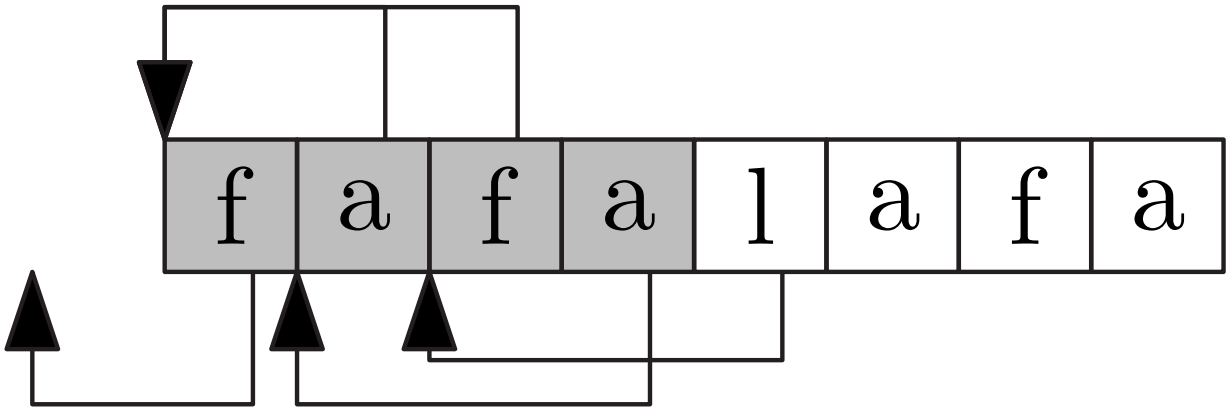
- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



KMP Algorithmus

Border Array

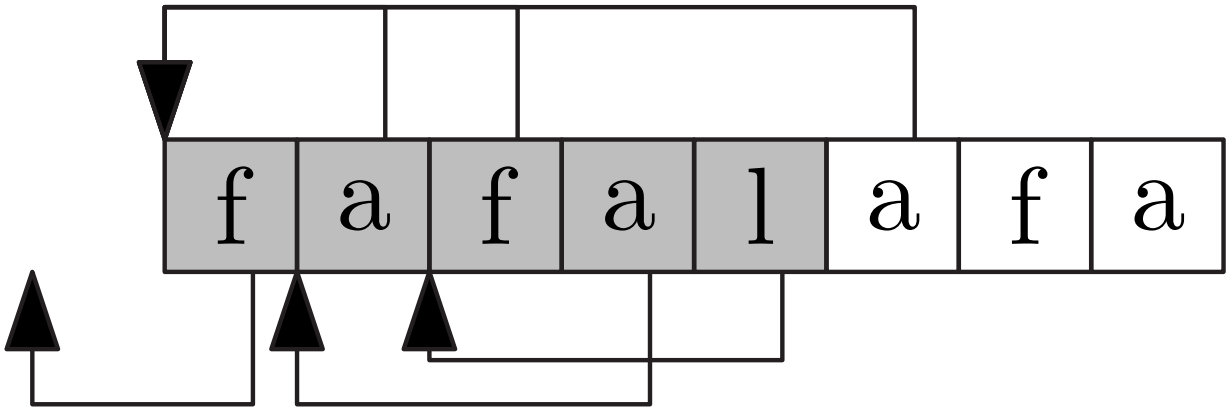
- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



KMP Algorithmus

Border Array

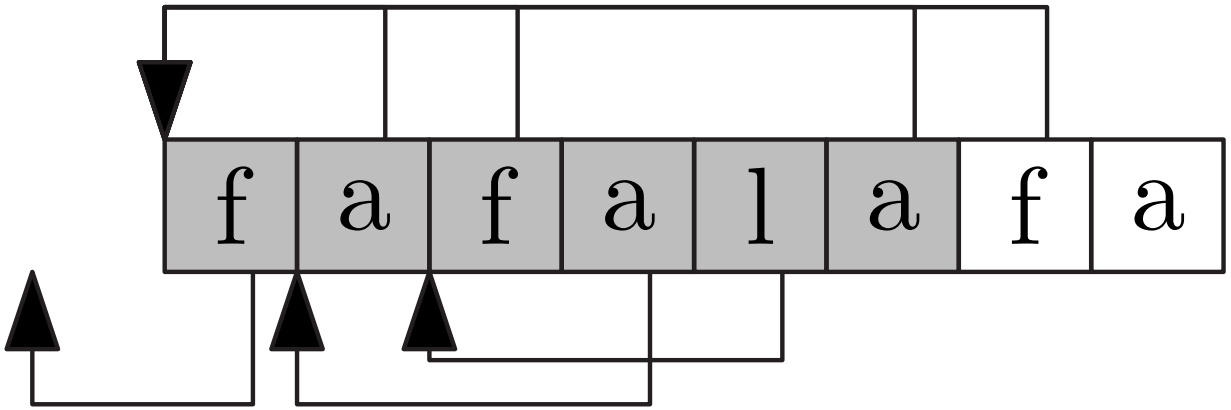
- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



KMP Algorithmus

Border Array

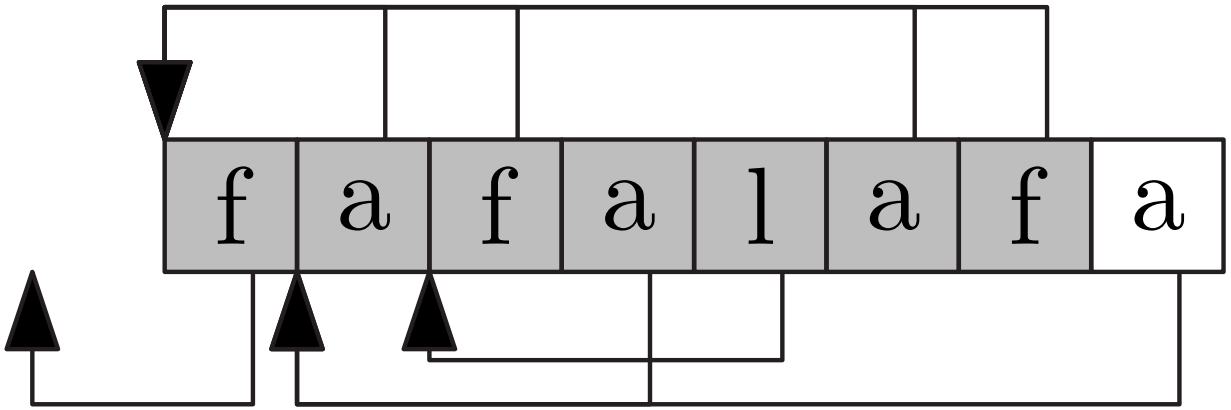
- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



KMP Algorithmus

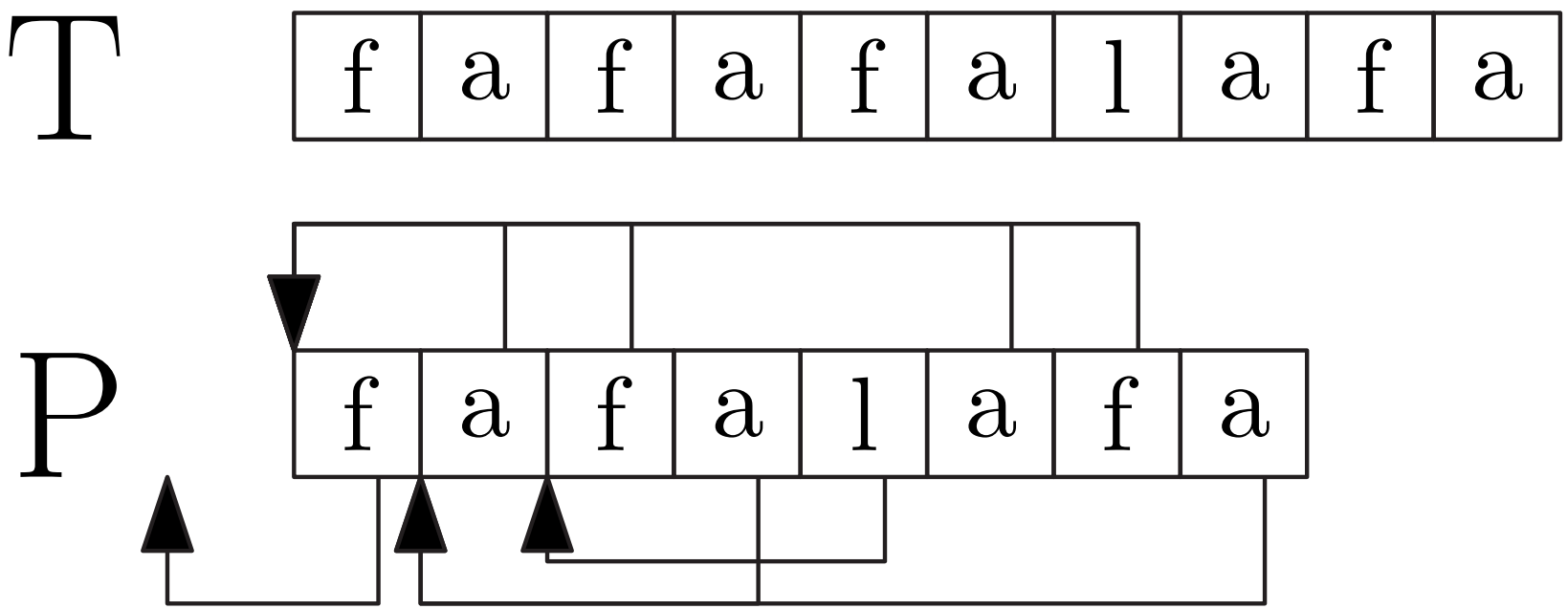
Border Array

- Pattern P
- $\mathrm{border}[j] =$ Länge des Präfixes von $P_{1\ldots j-1}$, das auch echtes Suffix von $P_{1\ldots j-1}$ ist



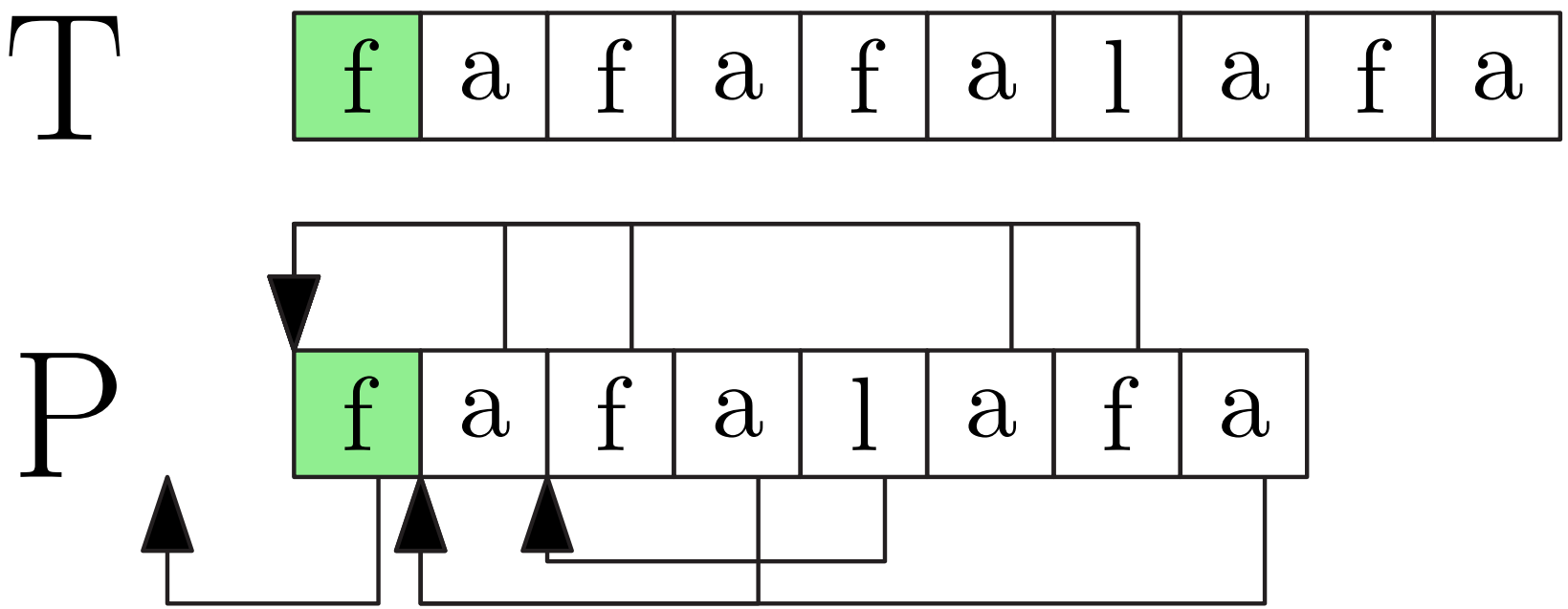
KMP Algorithmus

Pattern Suche



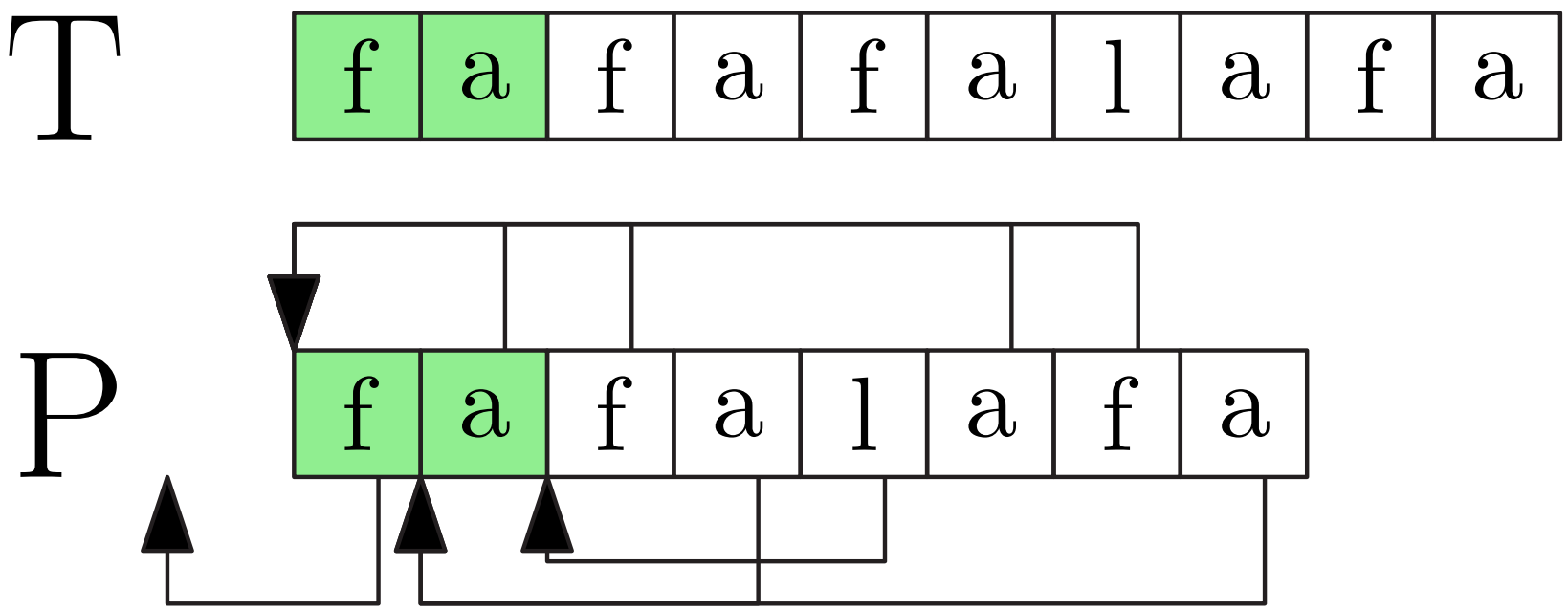
KMP Algorithmus

Pattern Suche



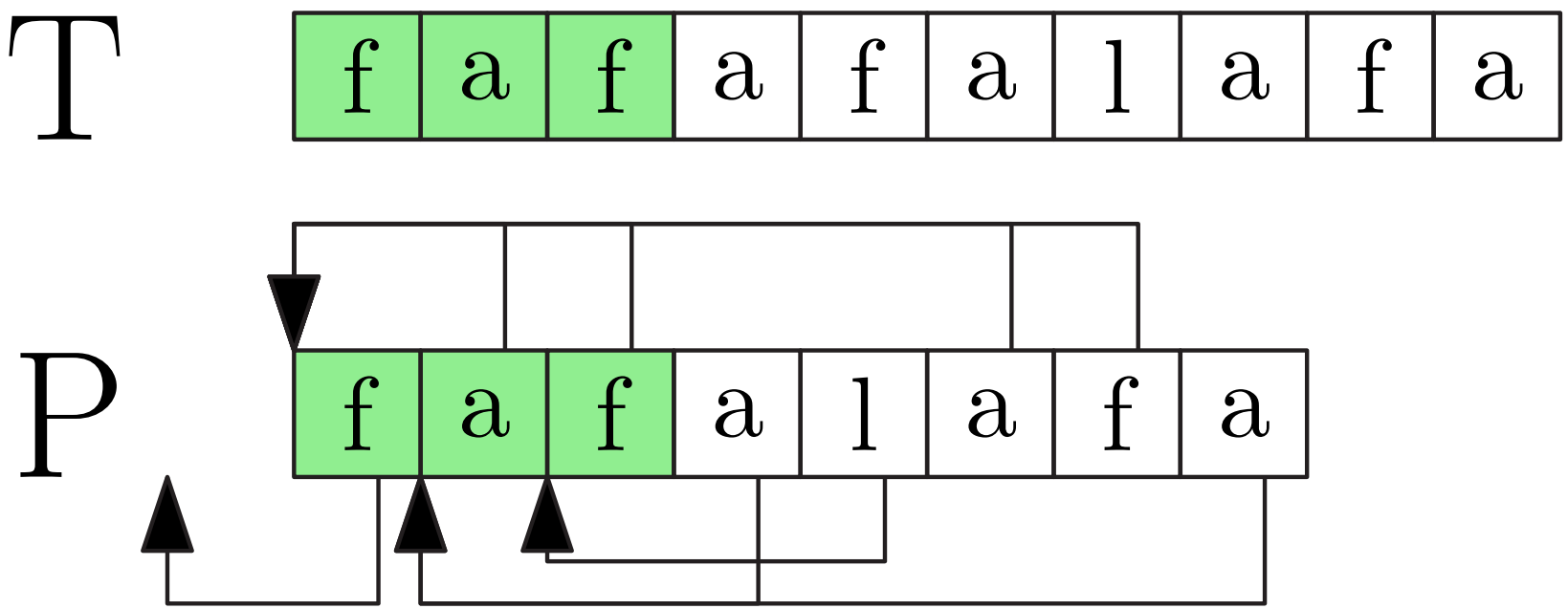
KMP Algorithmus

Pattern Suche



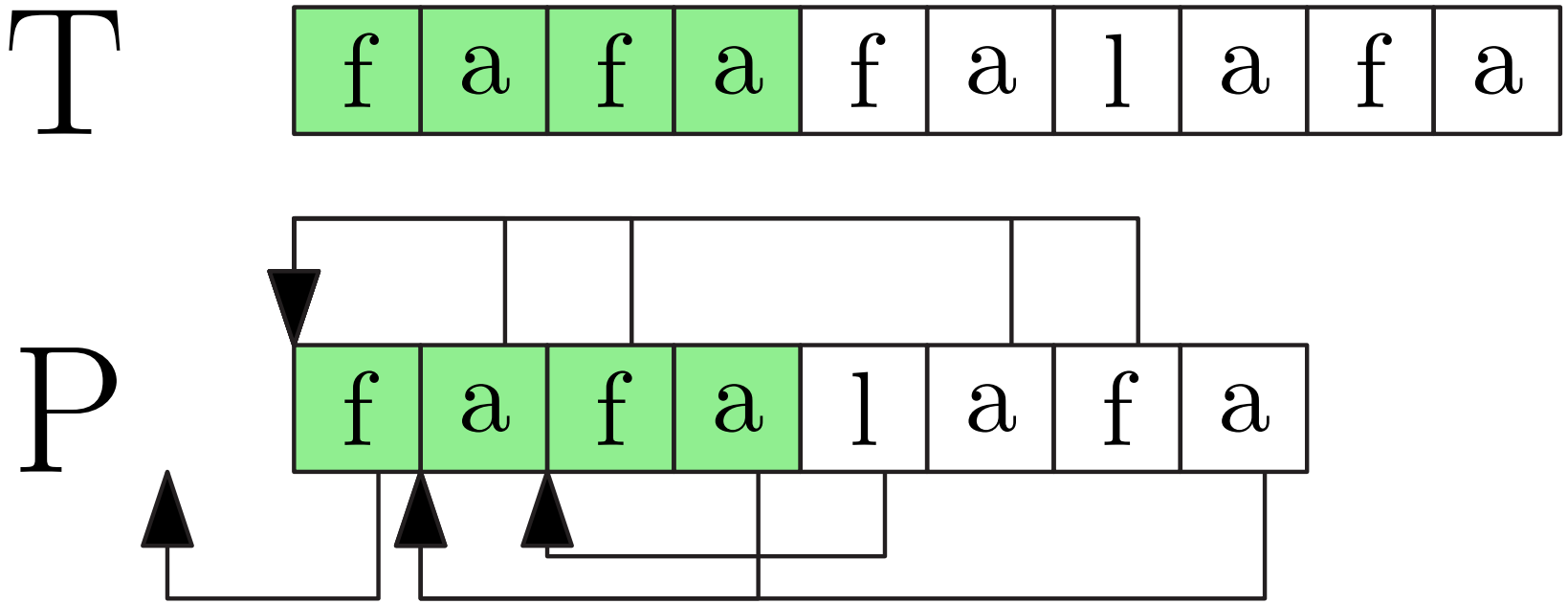
KMP Algorithmus

Pattern Suche



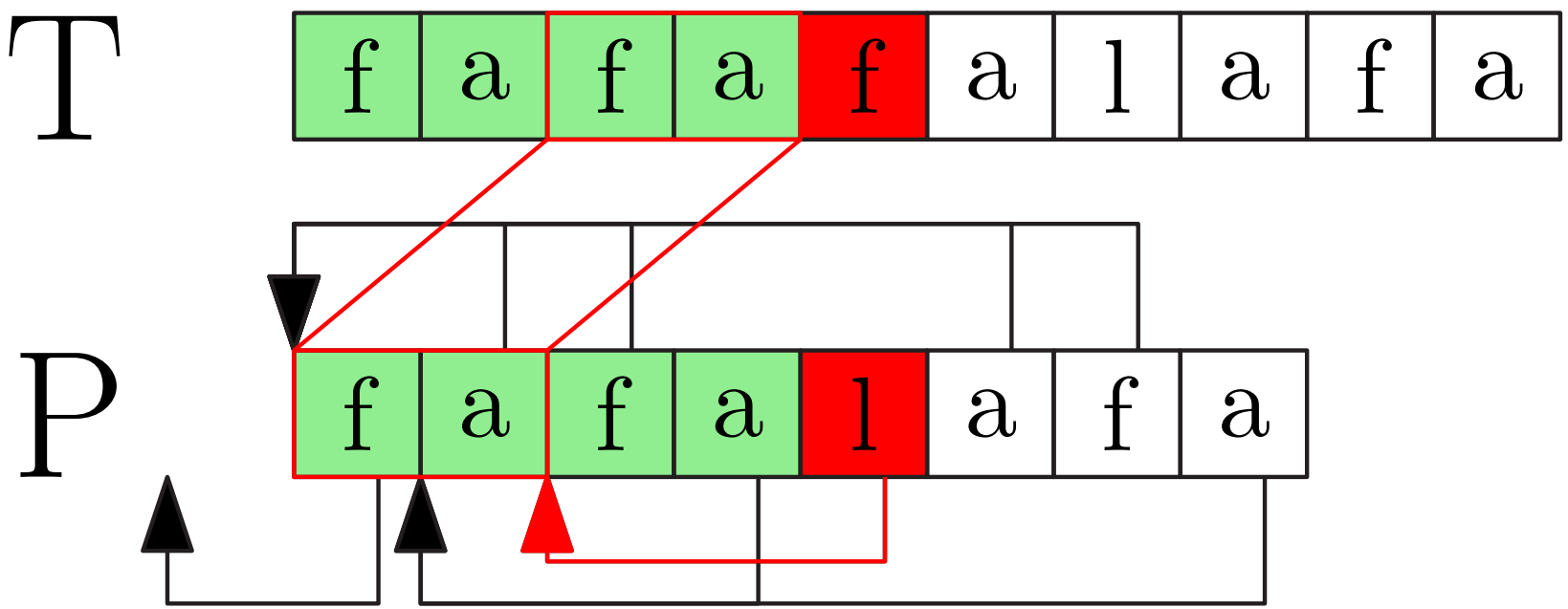
KMP Algorithmus

Pattern Suche



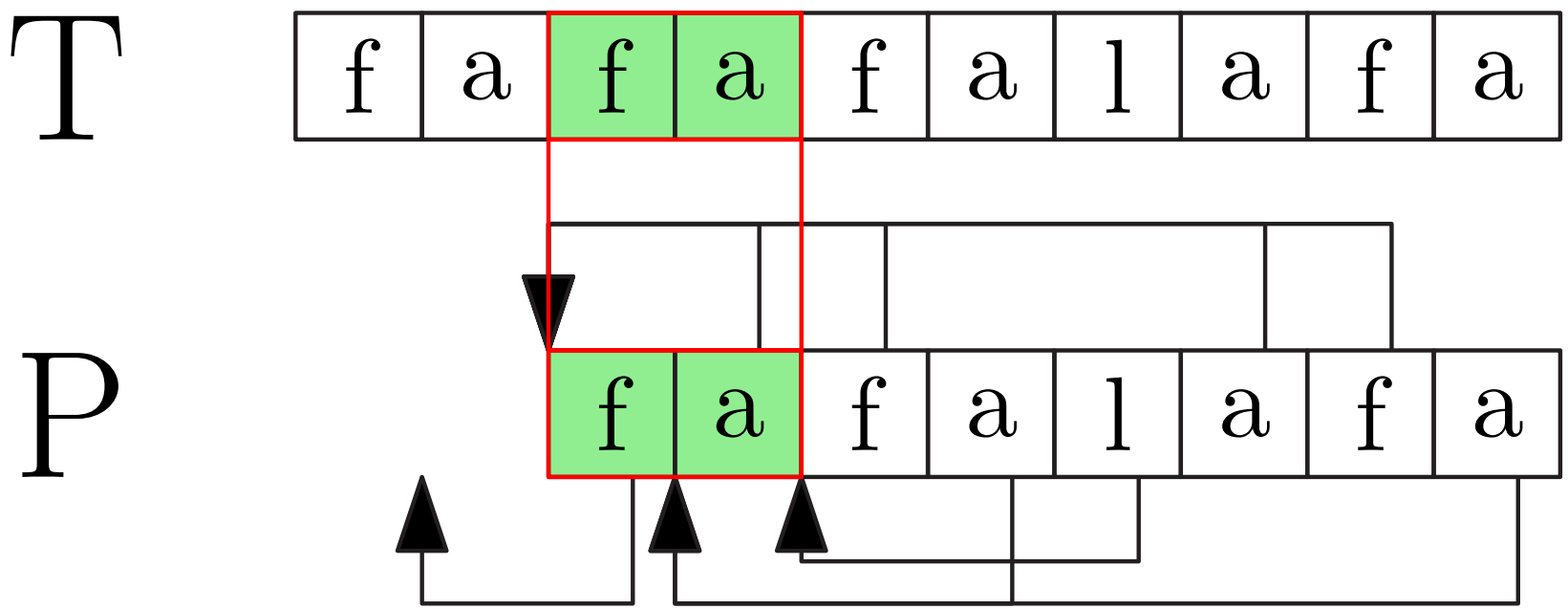
KMP Algorithmus

Pattern Suche



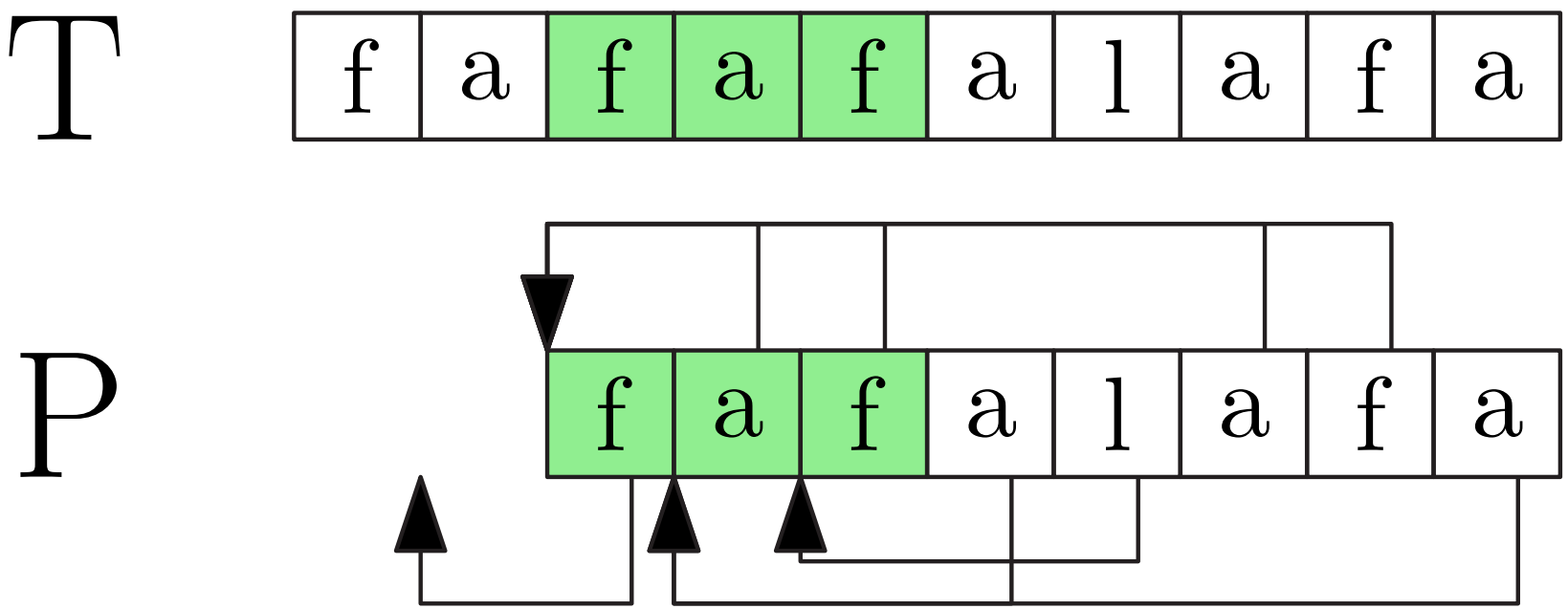
KMP Algorithmus

Pattern Suche



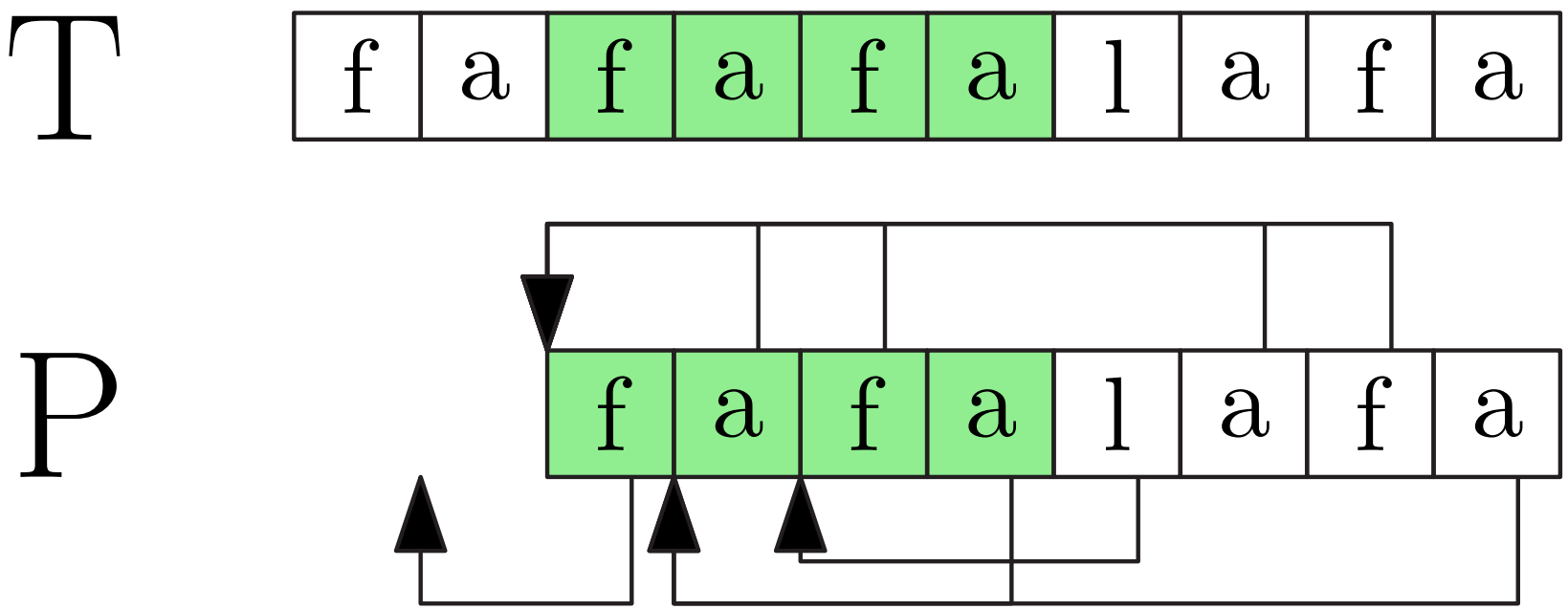
KMP Algorithmus

Pattern Suche



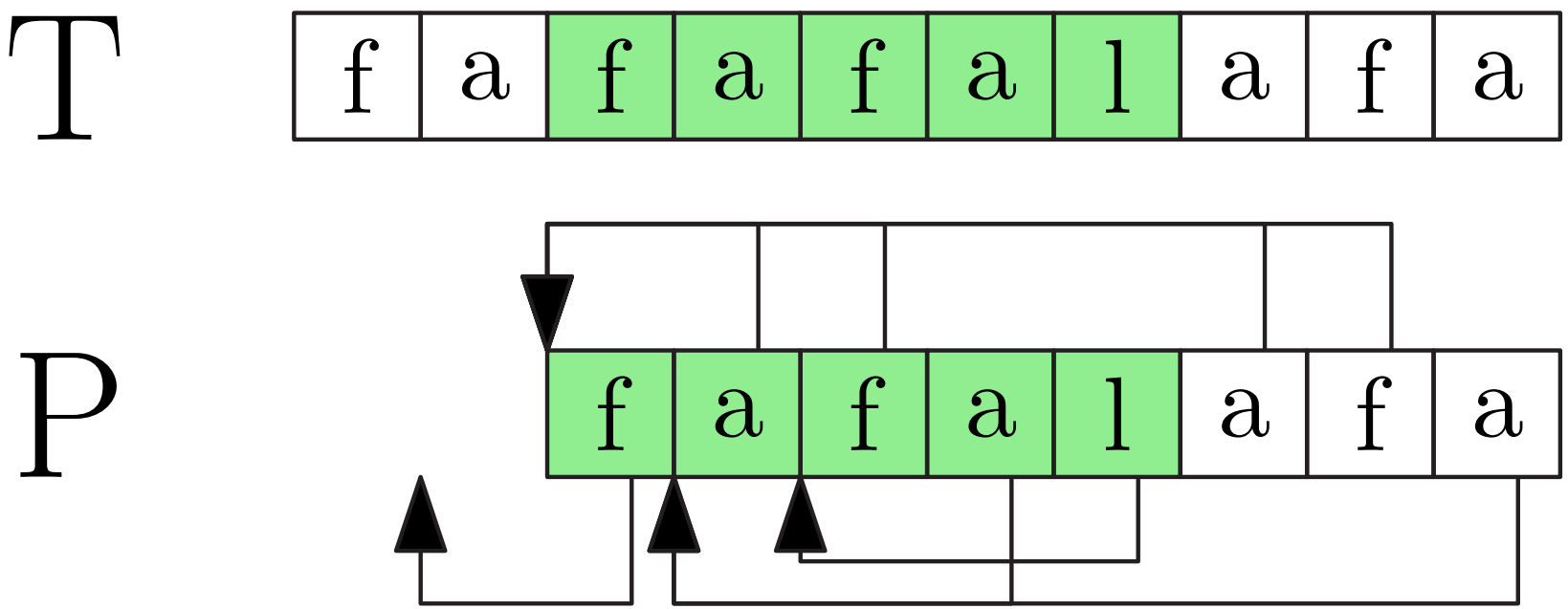
KMP Algorithmus

Pattern Suche



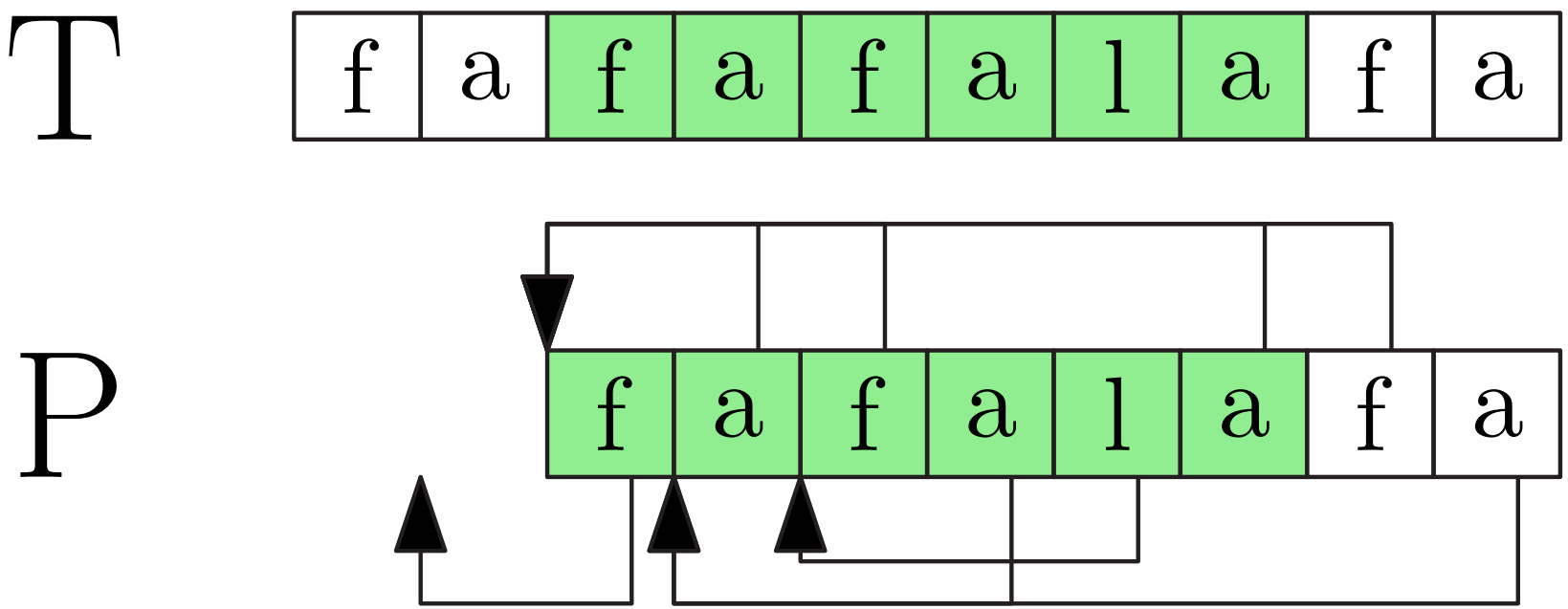
KMP Algorithmus

Pattern Suche



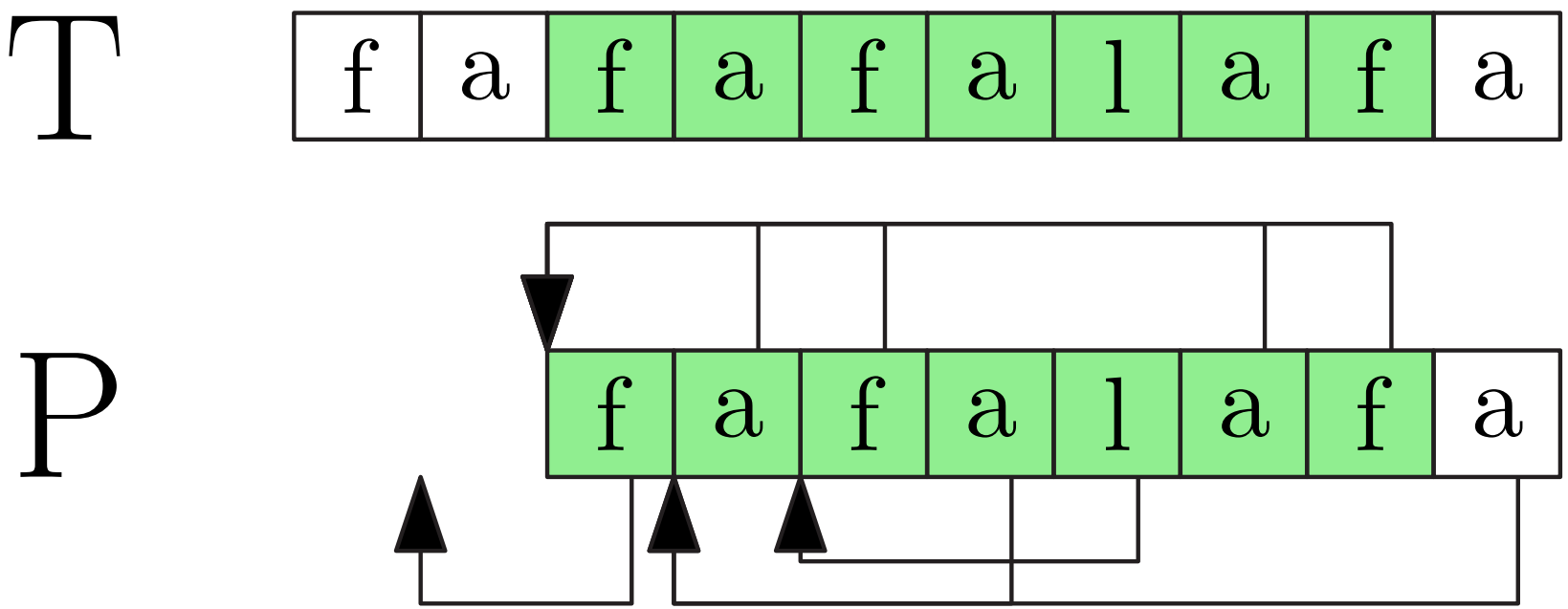
KMP Algorithmus

Pattern Suche



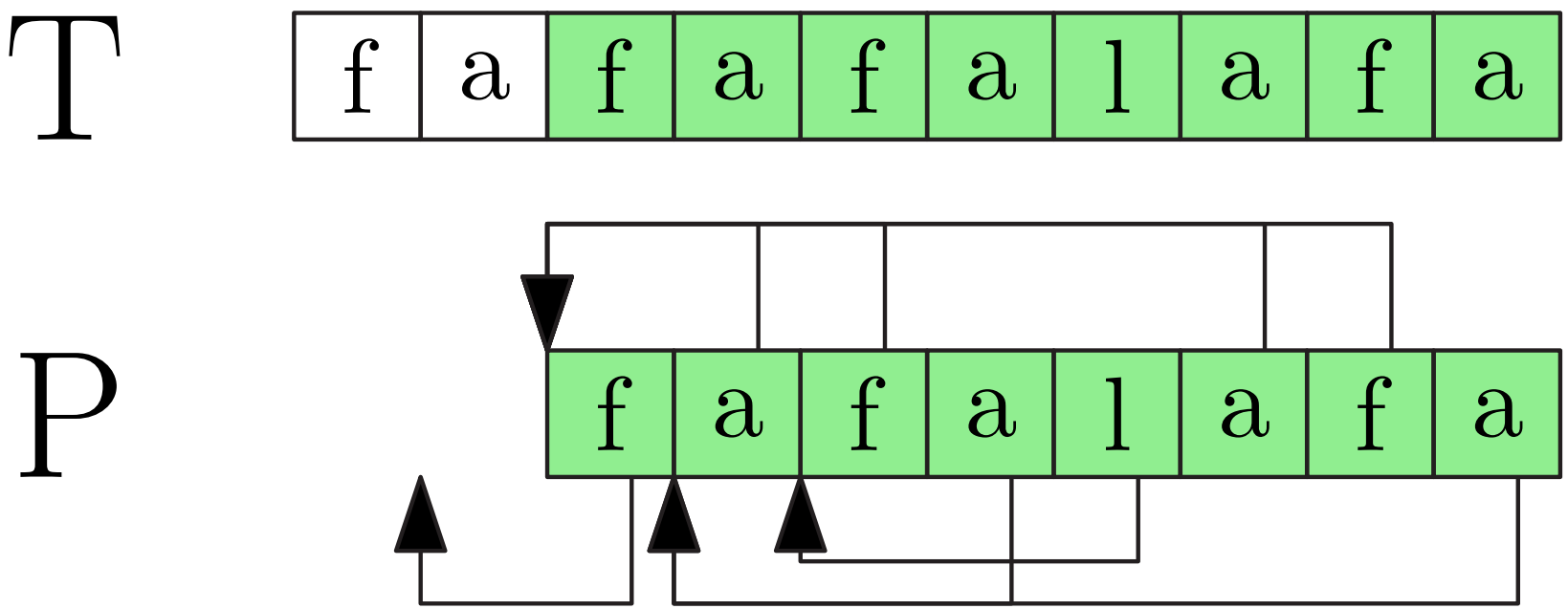
KMP Algorithmus

Pattern Suche



KMP Algorithmus

Pattern Suche



Evaluation der Übung

https://onlineumfrage.kit.edu/evasys/online.php?p=QSDQE
(bis 18 Uhr)

Ende!

Feierabend!

in-place Multikey Quicksort

Definitionen

- nach C-Standard ist ein `string` ist Zeiger auf char
  (die Daten werden als `char` Array mit *Sentinel* '\0' gespeichert)

```
void swap(int a, int b, char *x[]){                    //Dreieckstausch
 char *t=x[a];
 x[a]=x[b];
 x[b]=t;
}
```

```
#define i2c(i) x[i][depth]                             //Buchstabe an Stelle 'depth' aus i-tem String
```

```
void vecswap(int i, int j, int n, char *x[]) {         //Tausch der Sub-Arrays [i:i+n] und [j:j+n]
 while(n--> 0) {
  swap(i,j);
  i++; j++;
 }
}
```

in-place Multikey Quicksort

Algorithmus

```
void mkqsort( char *x[], int n, int depth){
 if(n <= 1) return;                                    //Weniger als zwei Strings zu sortieren?
 a = rand() % n;                                       //Wahl des Pivot-Elements
 swap(0,a,x);                                          //Pivot an den Anfang des Arrays verschieben
 v=i2c(0);                                             //Wert der aktuellen Stelle im Pivot
 a = b = 1;                                            //Anfang des zu sortierenden (Sub-)Arrays
 c = d = n-1;                                          //Ende des zu sortierenden (Sub-)Arrays
 while(true) {
  while(b <= c && (r = i2c(b) -v) <= 0) {              //Von links Elemente scannen, bis Char > Pivot
   if(r == 0) { swap(a,b,x); a++ }                     //Char == Pivot? Dann nach links kopieren
   b++;                                                //Grenze der gleichen Elemente eins nach rechts
  }
  while (b <= c && (r= i2c(c) -v) >= 0) {              //Von rechts Elemente scannen, bis Char < Pivot
   if(r == 0) { swap(c,d,x); d--; }                    //Char == Pivot? Dann nach rechts kopieren
   c--;                                                //Grenze der gleichen Elemente eins nach links
  }
  if(b>c) break;                                       //Zeiger aneinander vorbei? Dann Abbruch
  swap(b,c,x); b++; c--;                               //Inversion gefunden, tauschen, Zeiger je 1 weiter
 }
 r = min(a, b-a);     vecswap(0, b-r, r, x);           //"gleiche" Elemente von links zur Mitte kopieren
 r = min(d-c, n-d-1); vecswap(b, n-r, r, x);           //"gleiche" Elemente von recht zur Mitte kopieren
 r = b-a; mkqsort(x, r, depth);                        //"linke" Rekursion
 if(i2c(r) != 0) { mkqsort(x+r, a+n-d-1, depth+1); }   //"mittlere" Rekursion, falls notwendig
 r = d-c; mkqsort(x+n-r, r, depth);                    //"rechte" Rekursion
}
```

[Bentley&Sedgewick1997]