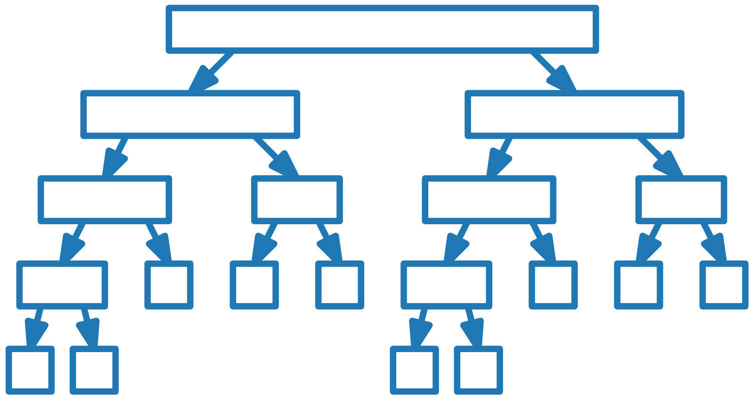
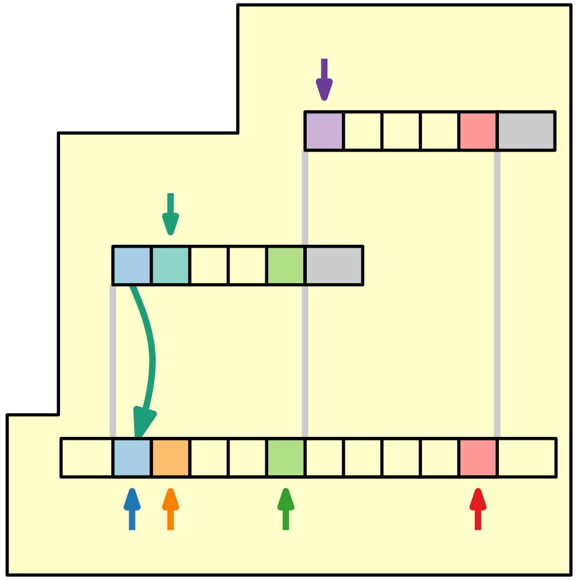
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Algorithmen und Datenstrukturen

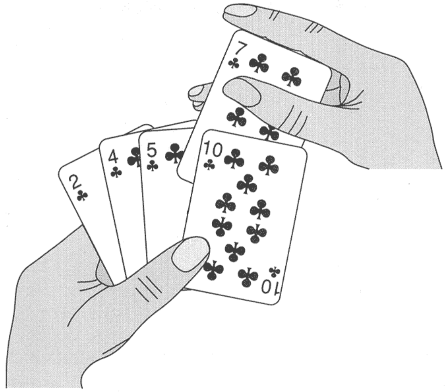


Vorlesung 2:
Teile und Herrsche

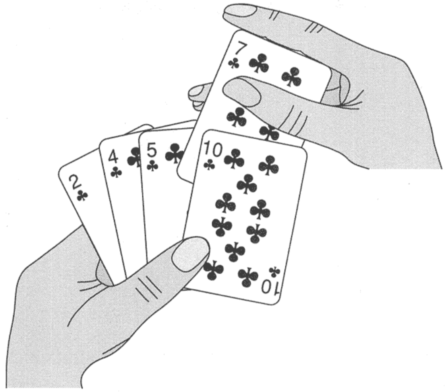


Alexander Wolff

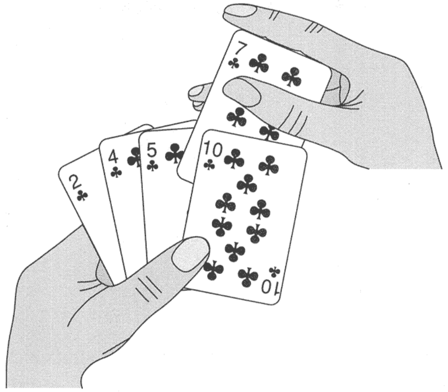
Wintersemester 2025



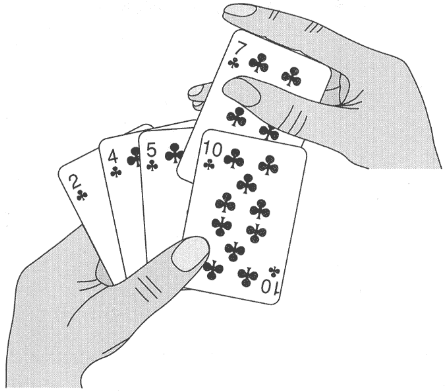
Teile und Herrsche



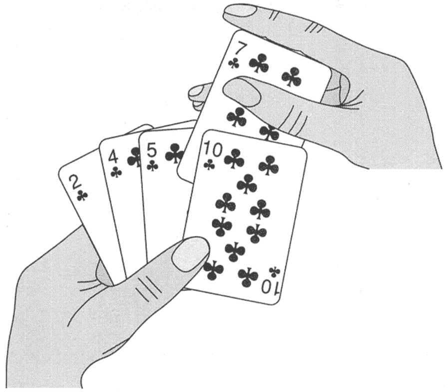
Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



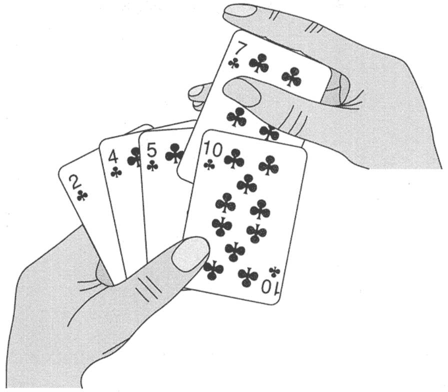
Teile und Herrsche



Idee.

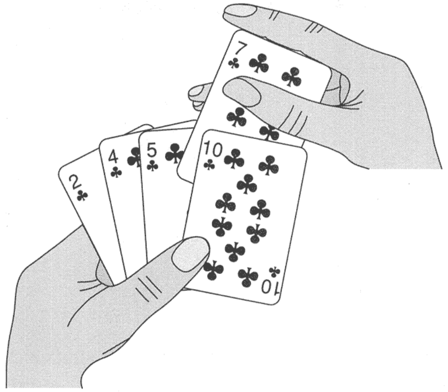


Teile und Herrsche

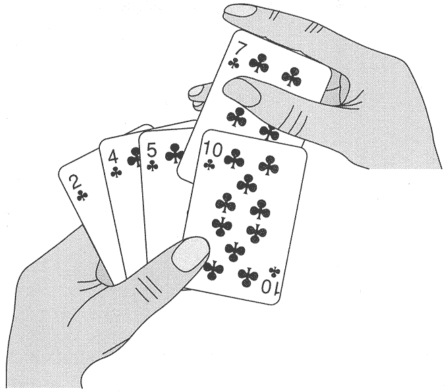


Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,



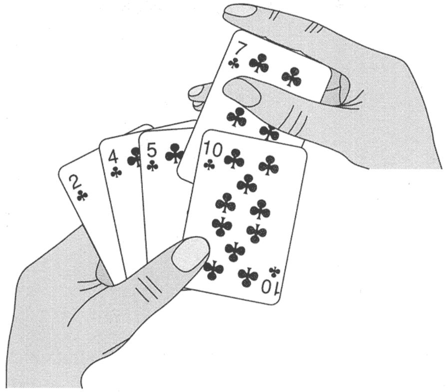
Teile und Herrsche



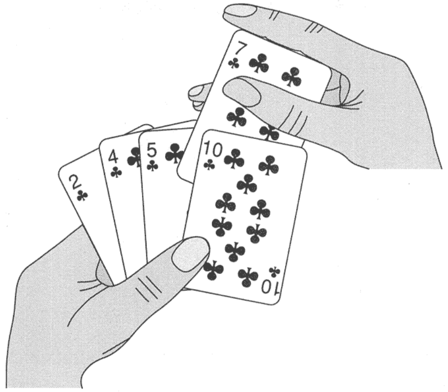
Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]

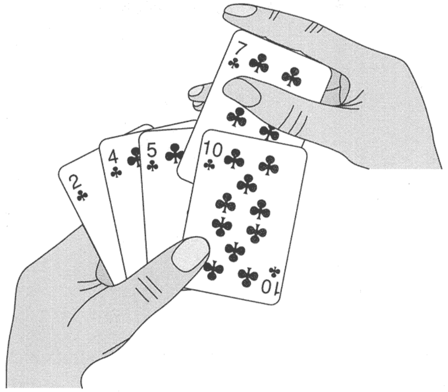


Teile und Herrsche

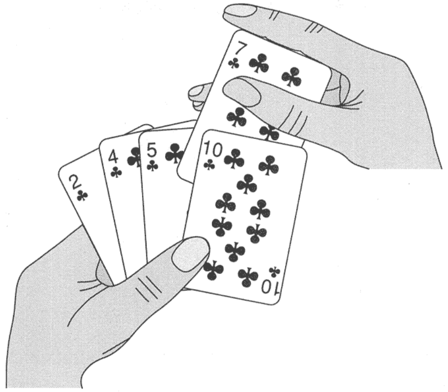


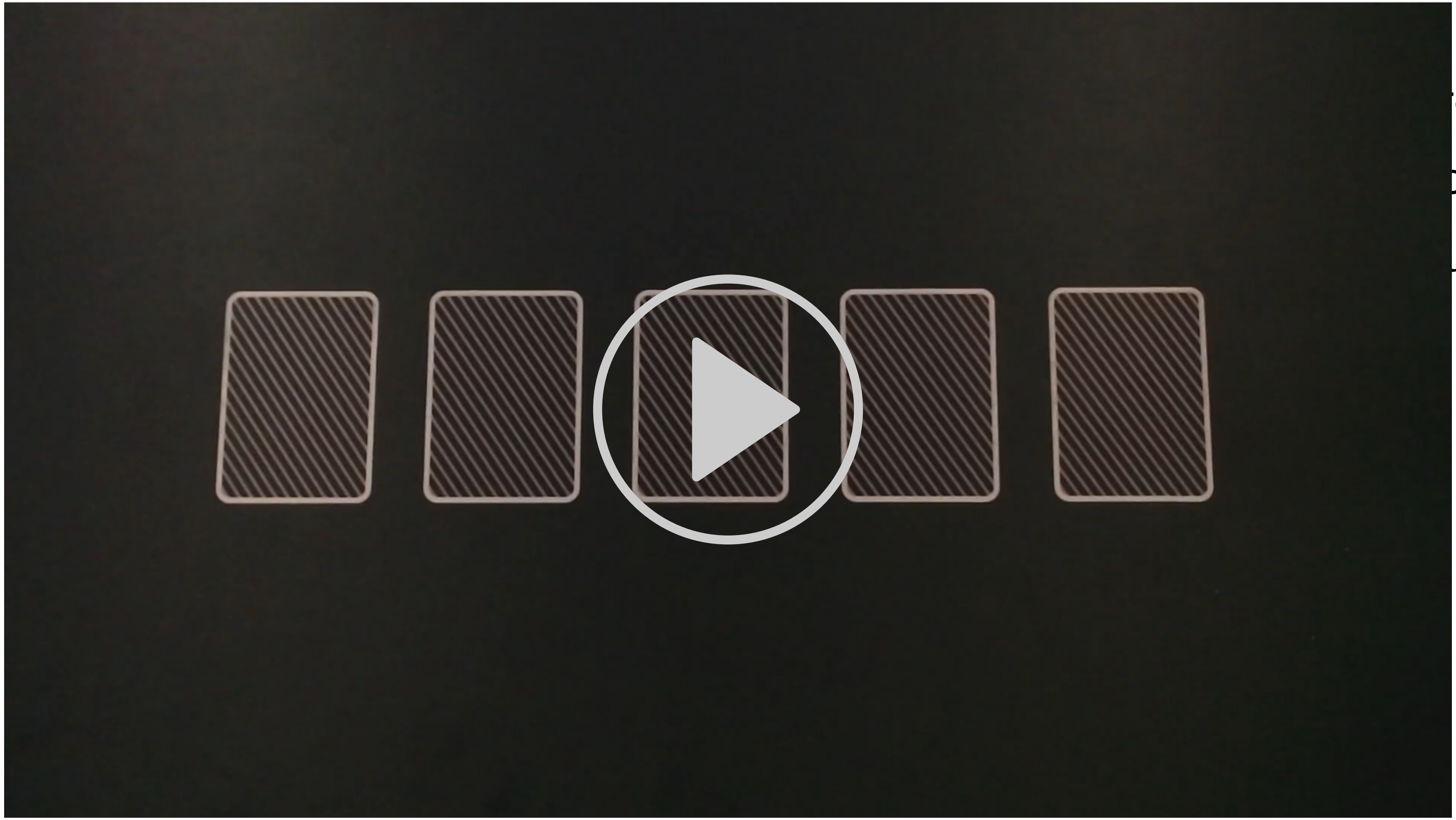
Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.



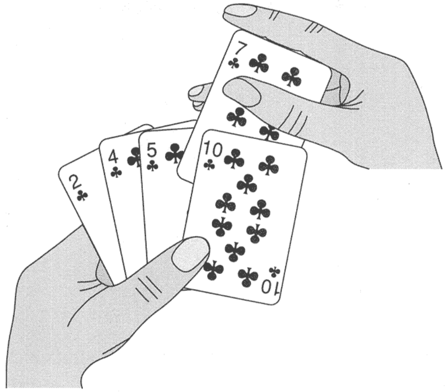
Teile und Herrsche



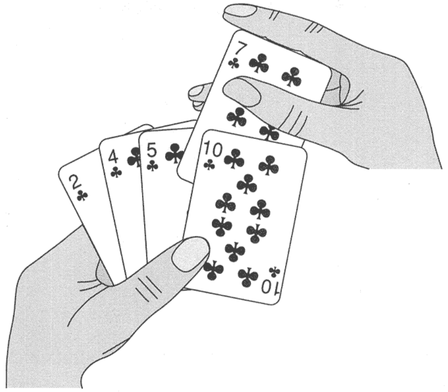


oße Teile,
onen) und
usammen.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche

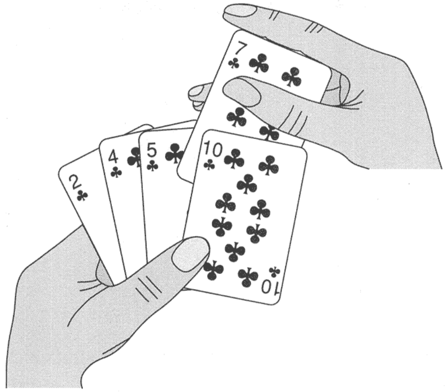


Idee.

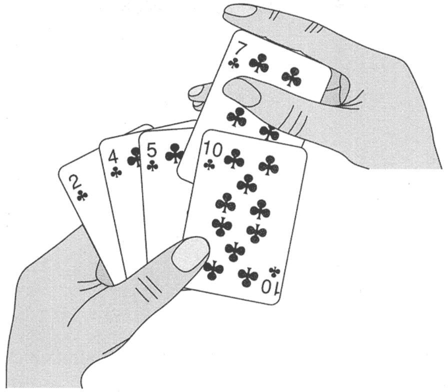
- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

Allgemein.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



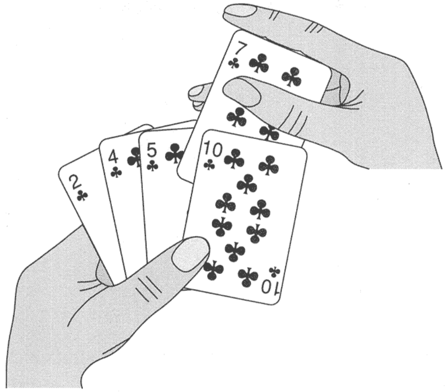
Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

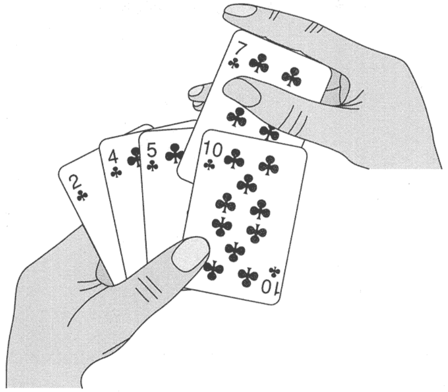
Allgemein.

Teile . . .

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



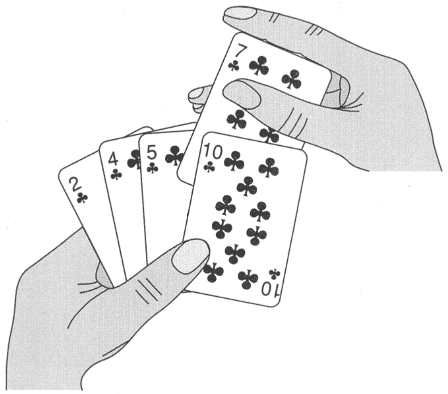
Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

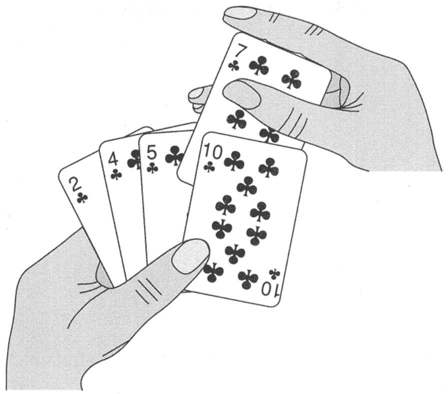
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



Idee.

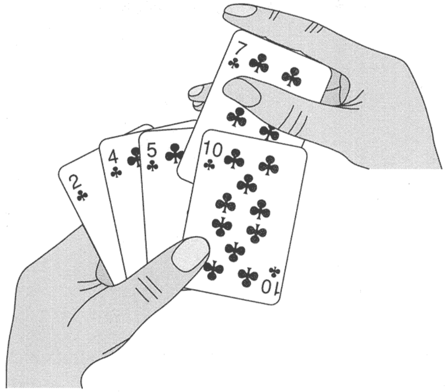
- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

Allgemein.

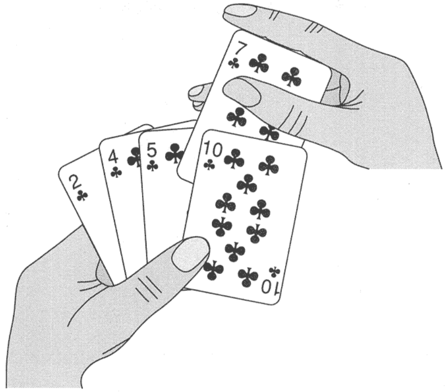
Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

Herrsche . . .

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



Idee.

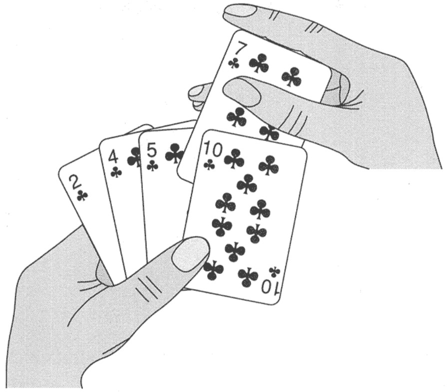
- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

Allgemein.

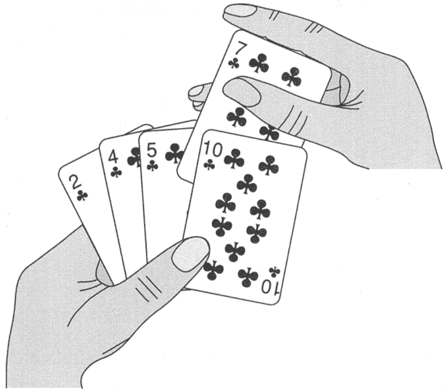
Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

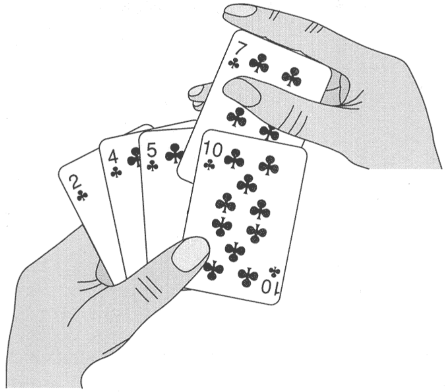
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

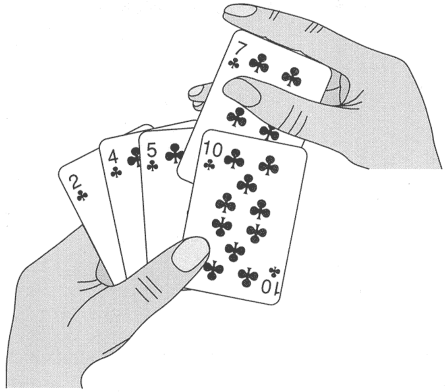
Aufruf einer Funktion durch sich selbst

Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

Allgemein.

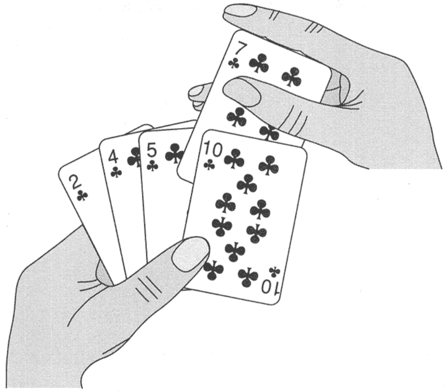
Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

Aufruf einer Funktion durch sich selbst

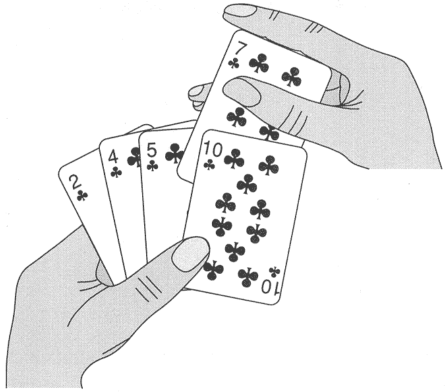
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . .

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

Allgemein.

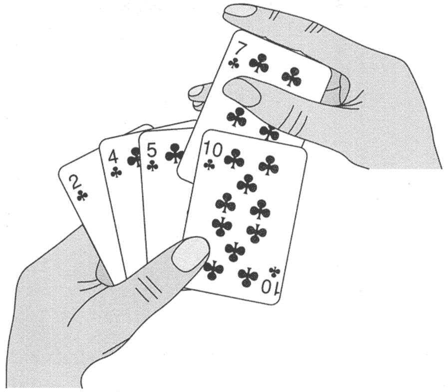
Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

Aufruf einer Funktion durch sich selbst

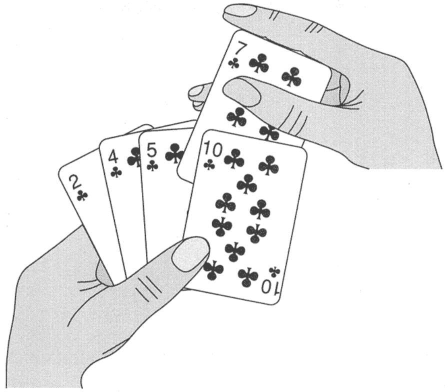
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms" [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

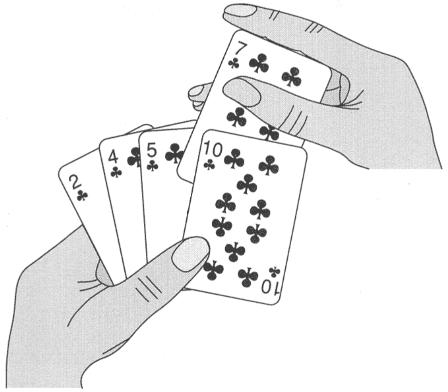
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

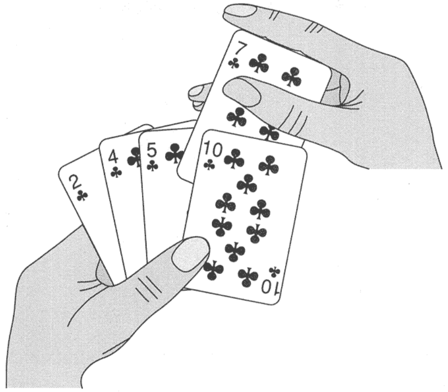
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

Defaultwerte

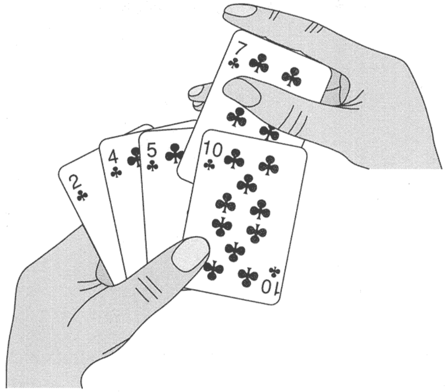
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

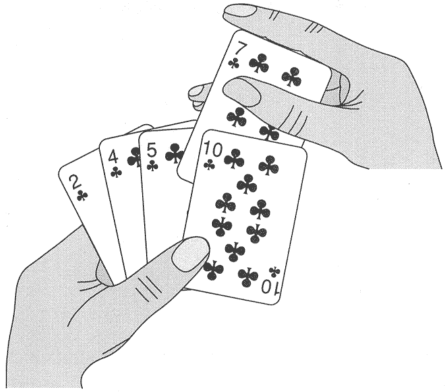
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.length)$

Defaultwerte

Dadurch wird die Funktion
$\textsc{MergeSort}(A) \equiv$
$\textsc{MergeSort}(A, 1, A.length)$
definiert.

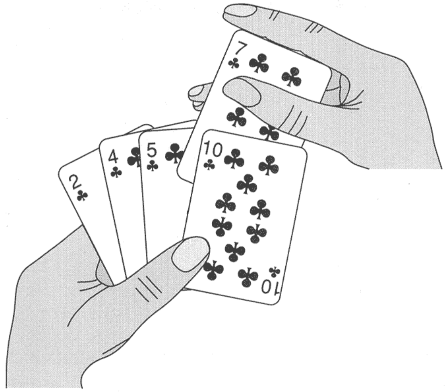
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

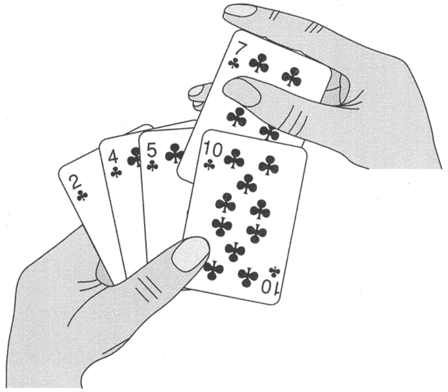
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

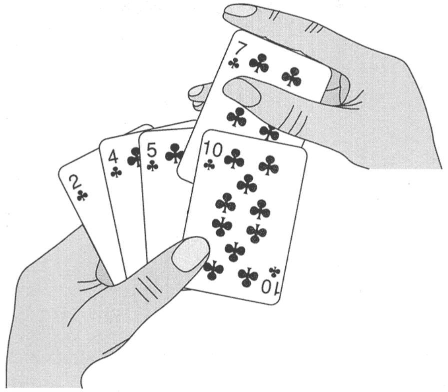
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

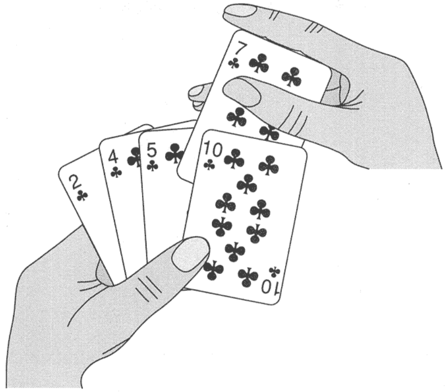
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

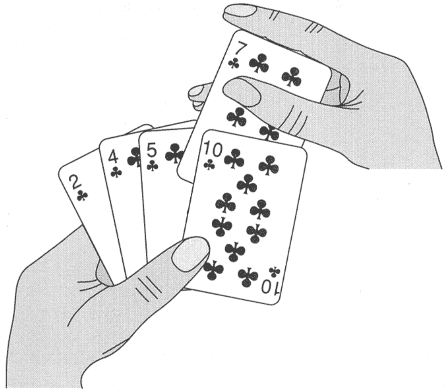
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

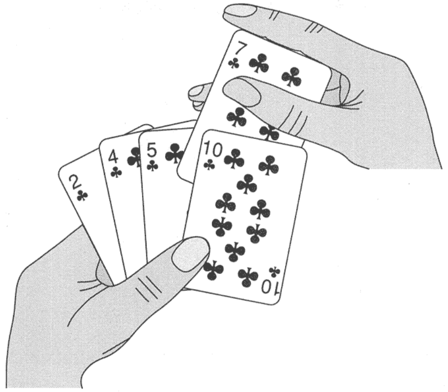
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$

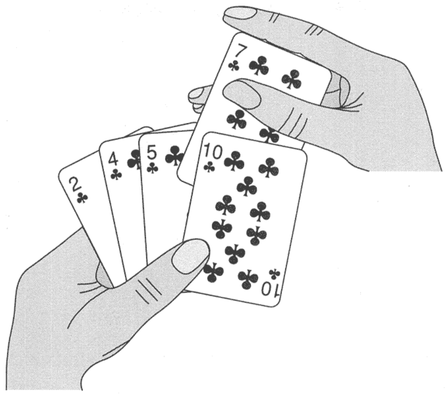
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

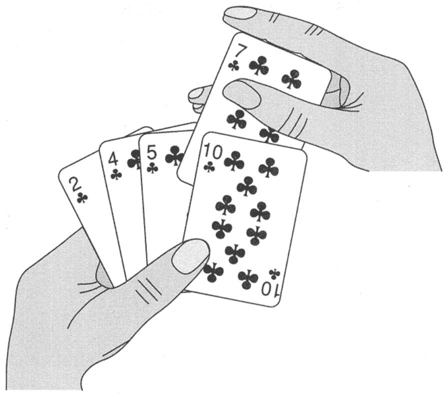
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

  $m = \lfloor(\ell + r)/2\rfloor$ $\}$ **teile**

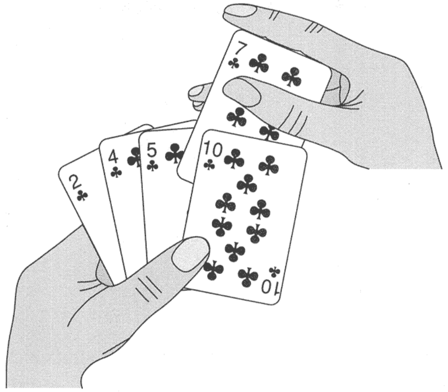
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

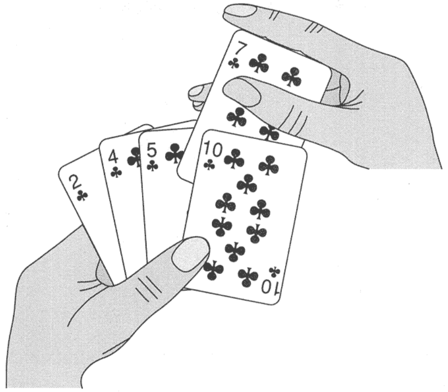
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$
if $\ell < r$ **then**
$\quad m = \lfloor (\ell + r)/2 \rfloor \quad \}$ **teile**
$\quad \textsc{MergeSort}(A, \ell, m)$
$\quad \textsc{MergeSort}(A, m+1, r)$

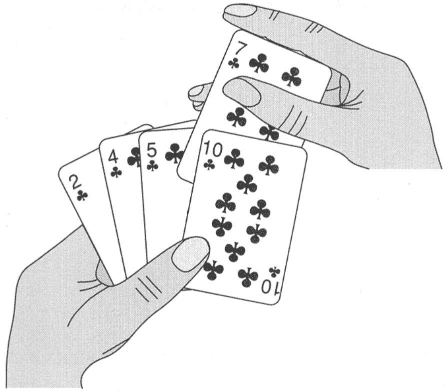
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

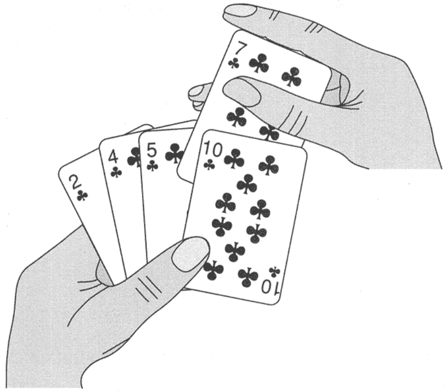
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[\,]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.length)$

if $\ell < r$ **then**

$\quad m = \lfloor(\ell + r)/2\rfloor$ $\Big\}$ **teile**

$\quad \textsc{MergeSort}(A, \ell, m)$

$\quad \textsc{MergeSort}(A, m+1, r)$ $\Big\}$ **herrsche**

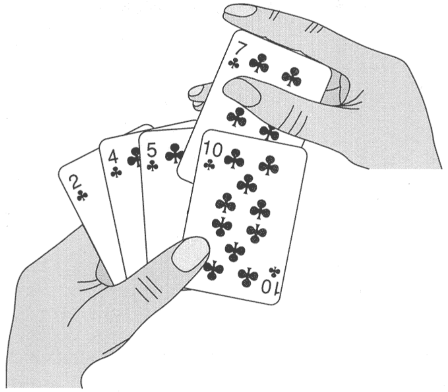
Allgemein.

Teile ... eine Instanz in kleinere Instanzen **desselben** Problems.

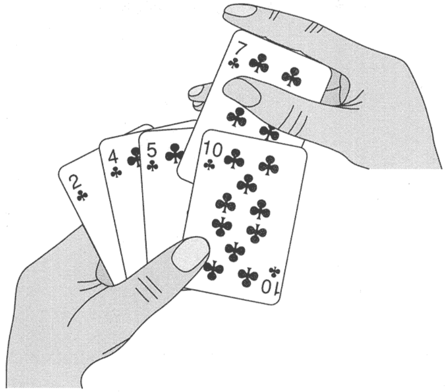
Herrsche ... durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere ... die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$\quad m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\quad \textsc{MergeSort}(A, \ell, m)$

$\quad \textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\quad \textsc{Merge}(A, \ell, m, r)$

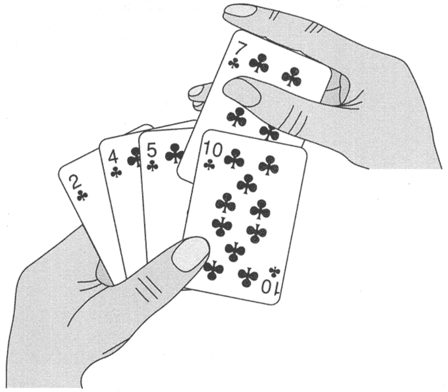
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

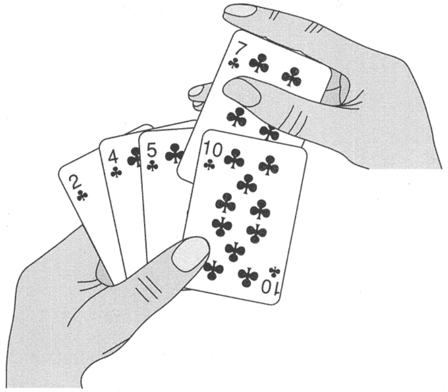
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



MergeSort(int[] A, int $\ell = 1$, int $r = A.\mathit{length}$)
if $\ell < r$ **then**
  $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
  MergeSort(A, ℓ, m)
  MergeSort$(A, m+1, r)$ } **herrsche**
  Merge(A, ℓ, m, r) } **kombiniere**

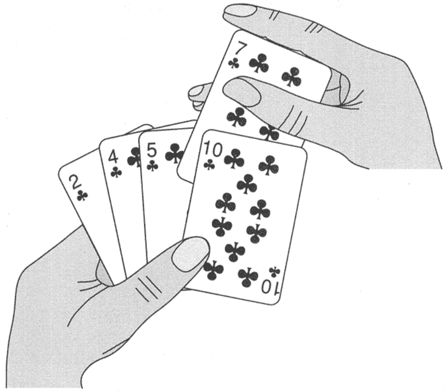
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

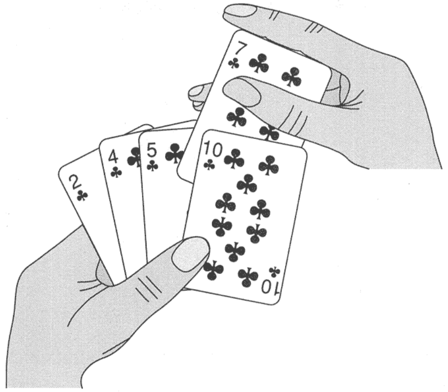
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\mathsf{int}[\,]\ A,\ \mathsf{int}\ \ell = 1,\ \mathsf{int}\ r = A.\mathit{length})$
if $\ell < r$ **then**
- $m = \lfloor(\ell + r)/2\rfloor$ $\}$ **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ $\}$ **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ $\}$ **kombiniere**

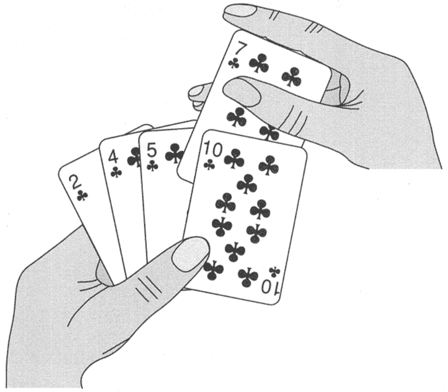
Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

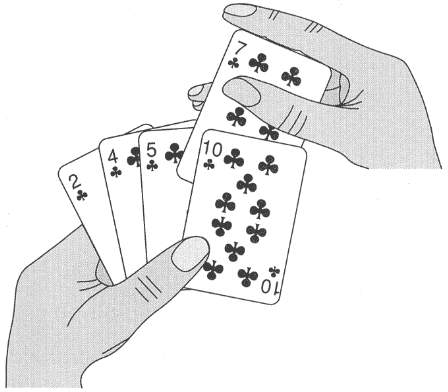
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]



Teile und Herrsche



$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

TODO!

Allgemein.

Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

Kombiniere . . . die Teillösungen zu einer Lösung der ursprünglichen Instanz.

Zeichnungen aus: „Introduction to Algorithms“ [Cormen et al., MIT Press, 2. Aufl., 2001]

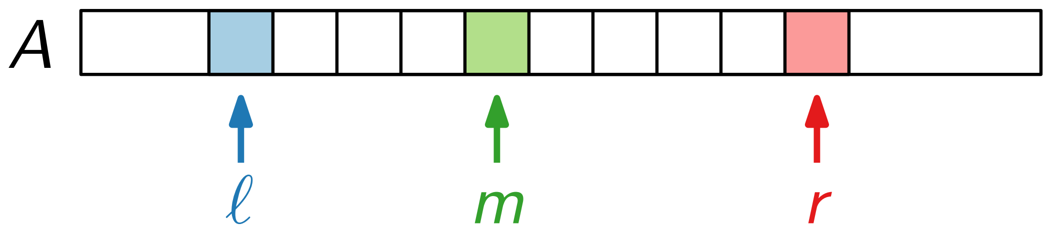
Kombiniere

Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

Kombiniere

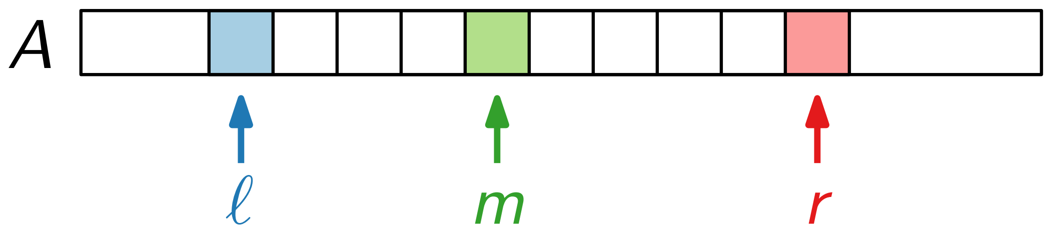
MERGE(int[] A, int ℓ, int m, int r)



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A,\ \text{int}\ \ell,\ \text{int}\ m,\ \text{int}\ r)$

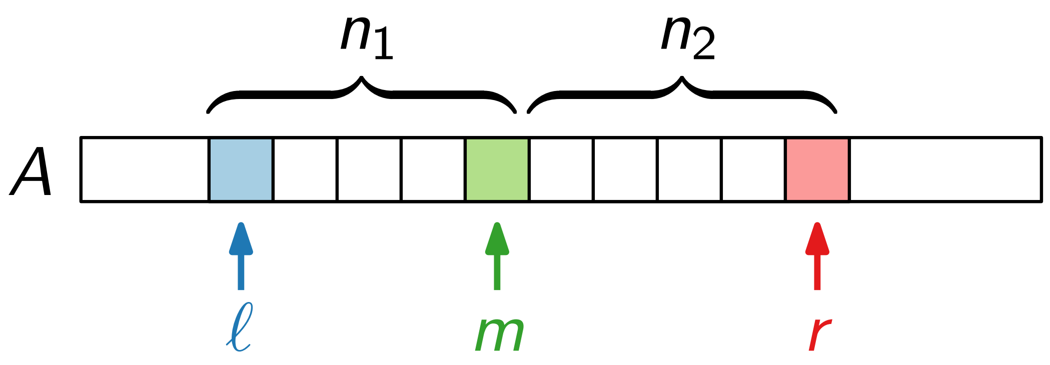
$n_1 = m - \ell + 1;\ \ n_2 = r - m$



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A,\ \text{int}\ \ell,\ \text{int}\ m,\ \text{int}\ r)$

$\quad n_1 = m - \ell + 1;\ \ n_2 = r - m$

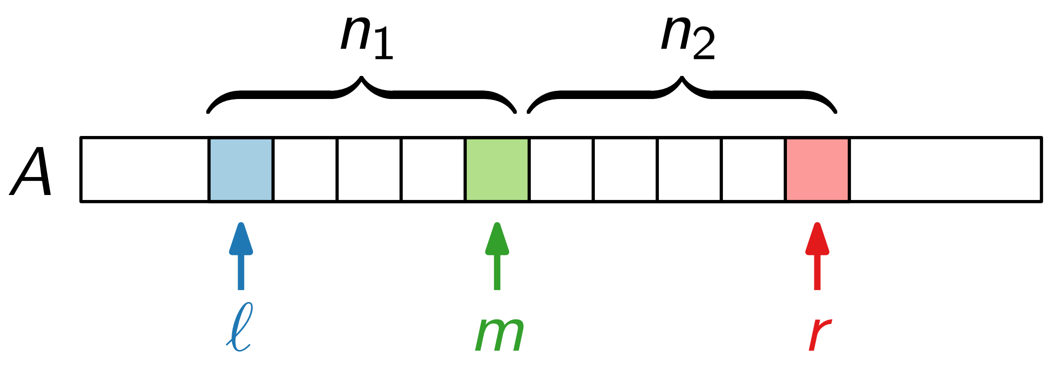


Kombiniere

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$

$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$



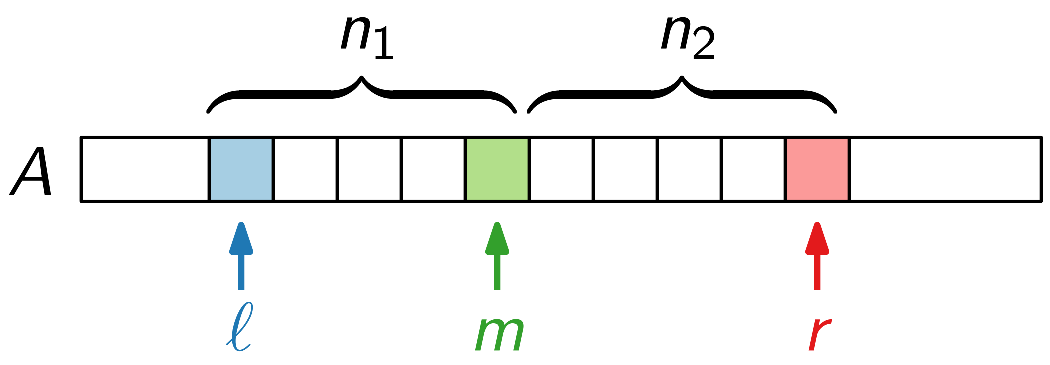
Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1; \quad n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \ldots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \ldots n_2 + 1]$

$L[1 \ldots n_1] = A[\ell \ldots m]$



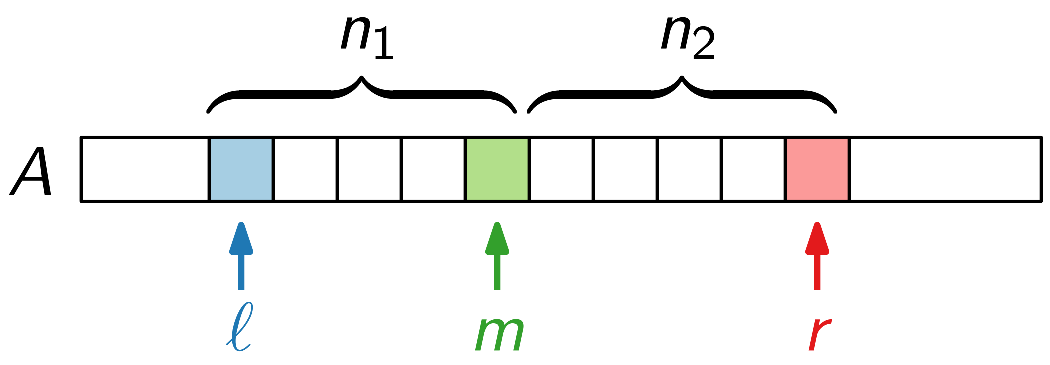
Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$



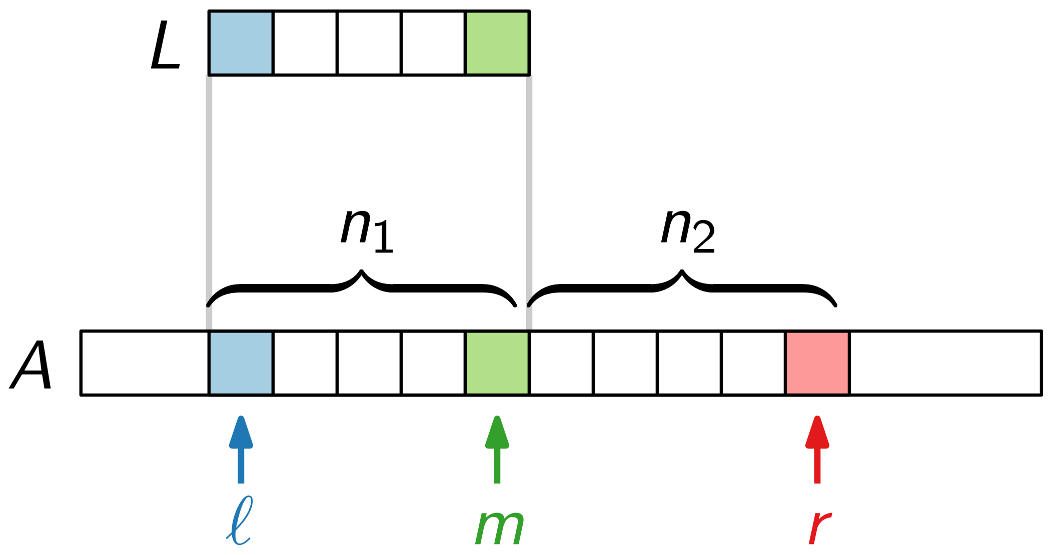
Kombiniere

$\textsc{Merge}(\text{int}[]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$



Kombiniere

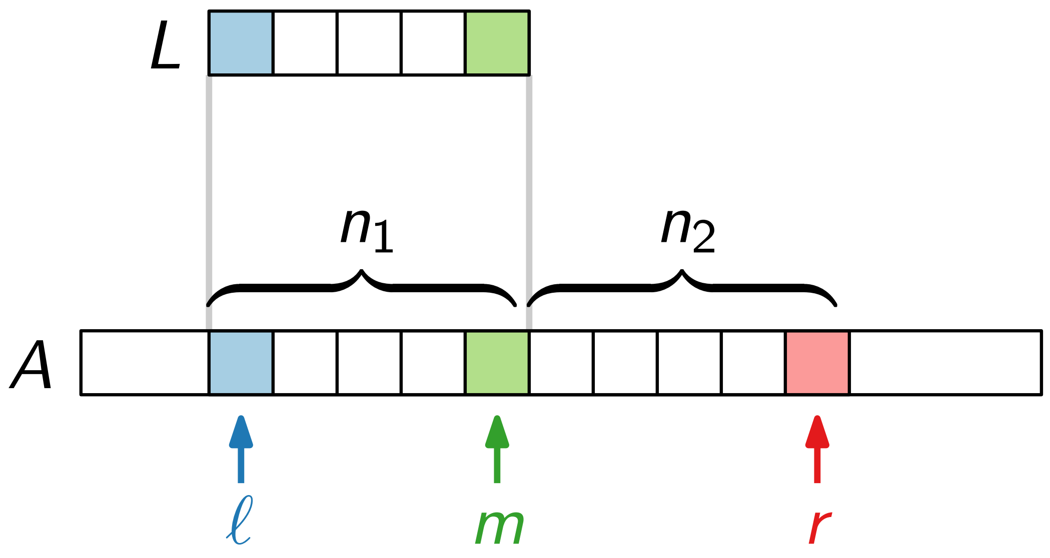
$\textsc{Merge}(\text{int[] } A, \text{ int } \ell, \text{ int } m, \text{ int } r)$

$n_1 = m - \ell + 1;\;\; n_2 = r - m$

$L = \textbf{new } \text{int}[1 \ldots n_1 + 1];\; R = \textbf{new } \text{int}[1 \ldots n_2 + 1]$

$L[1 \ldots n_1] = A[\ell \ldots m]$

for $i = 1$ **to** n_1 **do**
  $L[i] = A[(\ell - 1) + i]$



Kombiniere

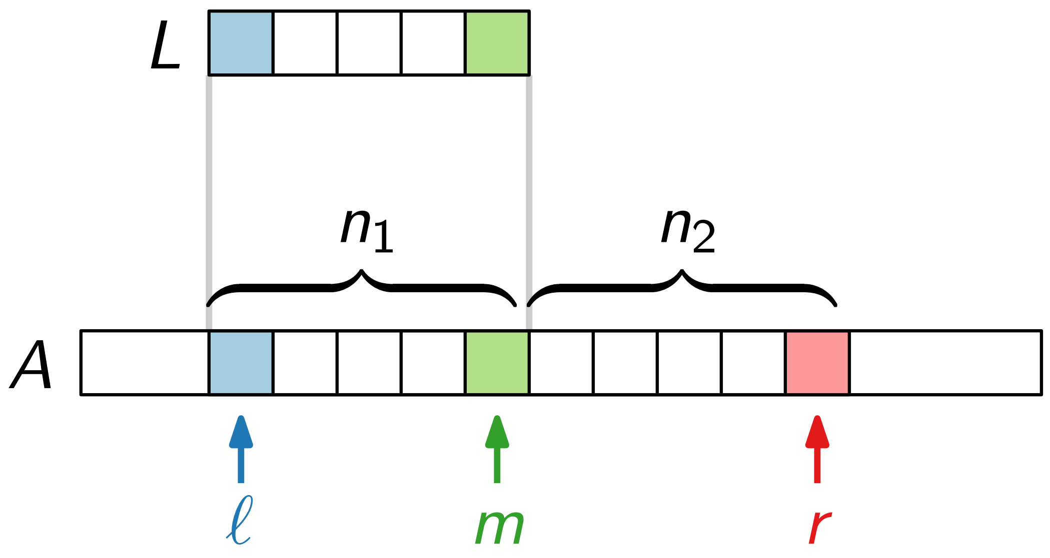
$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \ldots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \ldots n_2 + 1]$

$L[1 \ldots n_1] = A[\ell \ldots m]$

$R[1 \ldots n_2] = A[m + 1 \ldots r]$



Kombiniere

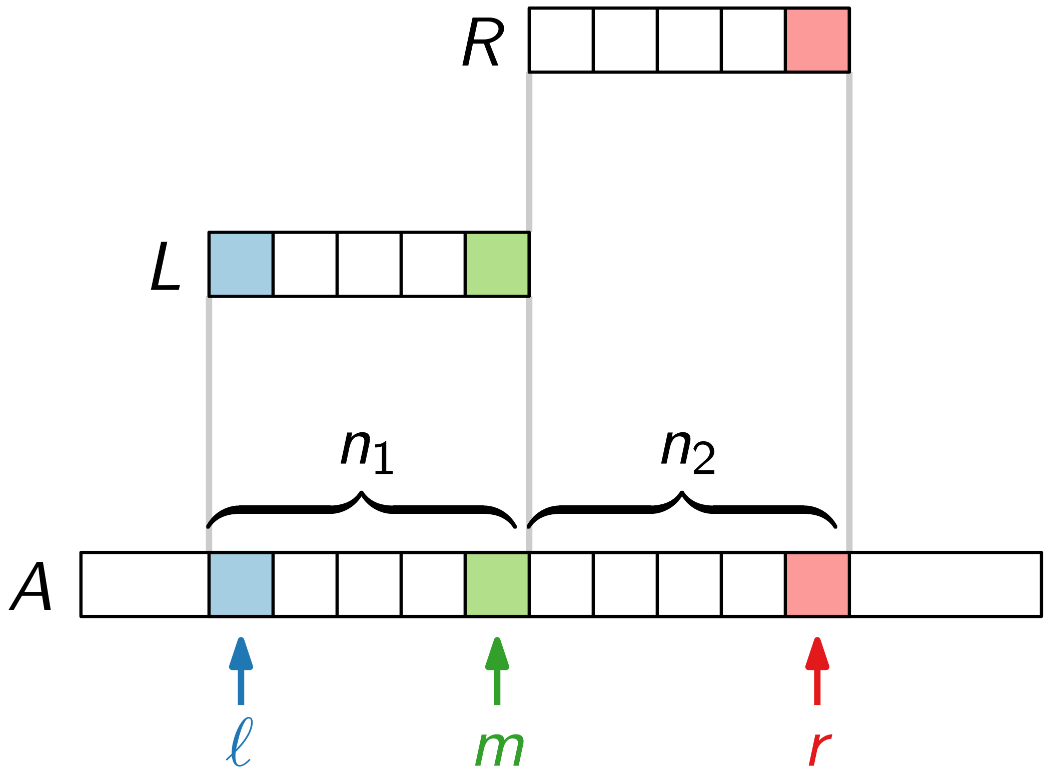
$\textsc{Merge}(\text{int}[\,]\ A,\ \text{int}\ \ell,\ \text{int}\ m,\ \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m + 1 \dots r]$



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

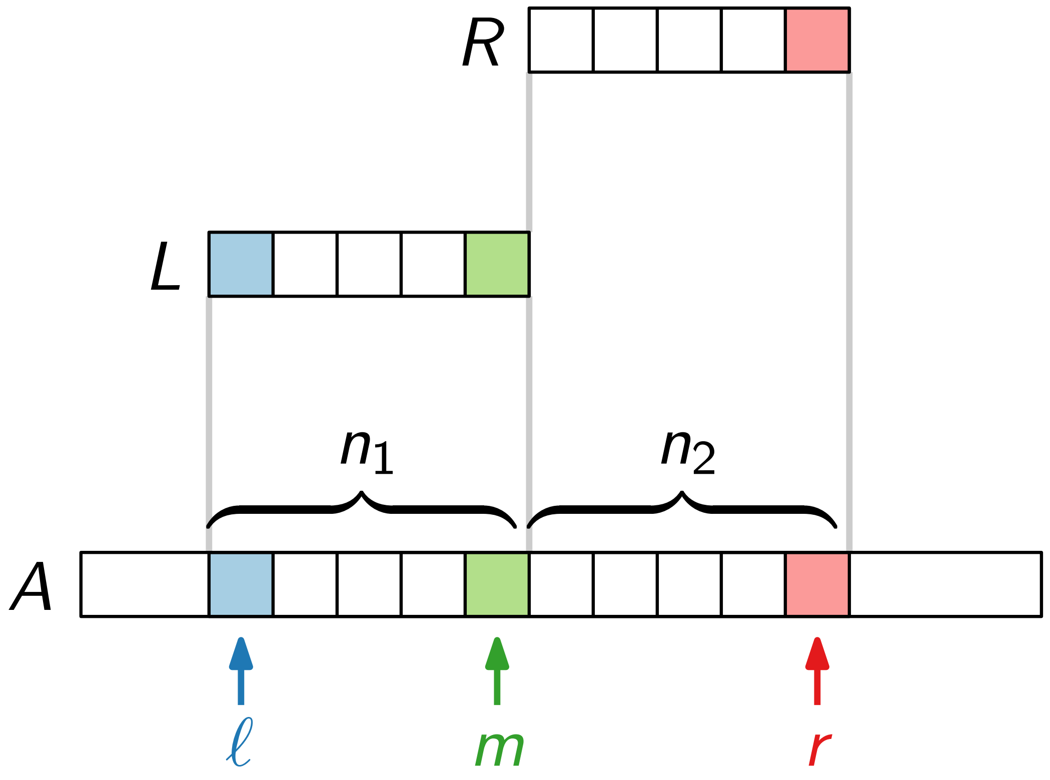
$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m + 1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

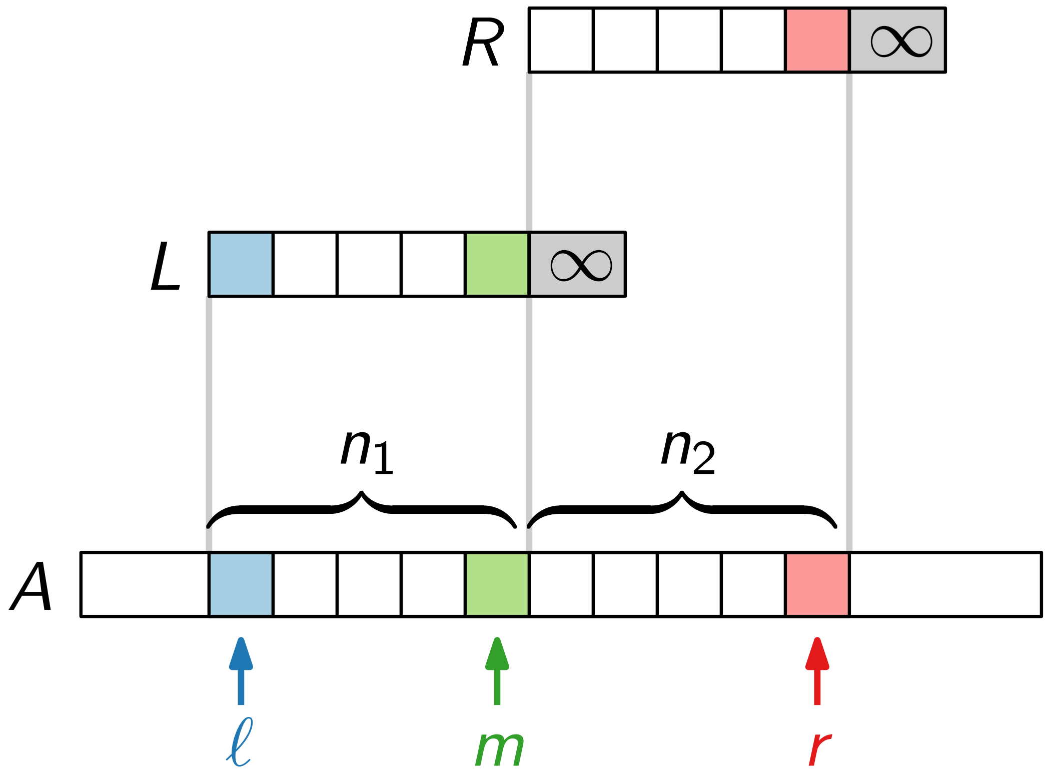
$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \ldots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \ldots n_2 + 1]$

$L[1 \ldots n_1] = A[\ell \ldots m]$

$R[1 \ldots n_2] = A[m + 1 \ldots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$



Kombiniere

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$

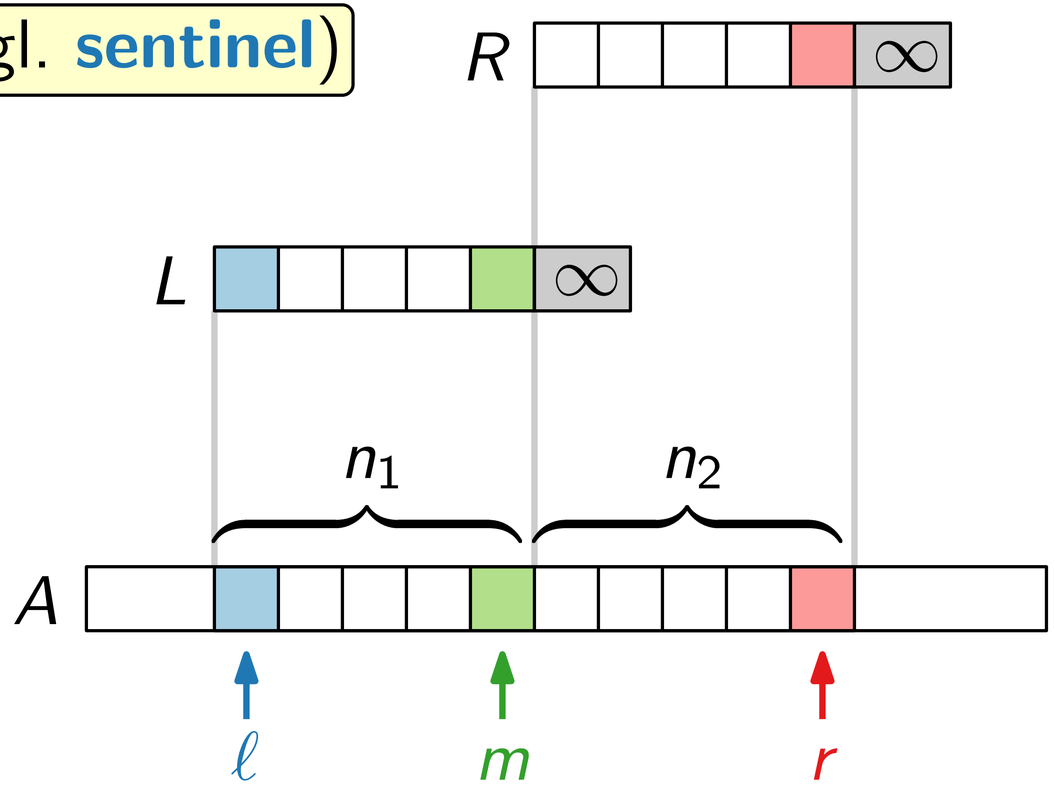
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m + 1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

Stopper (engl. **sentinel**)



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

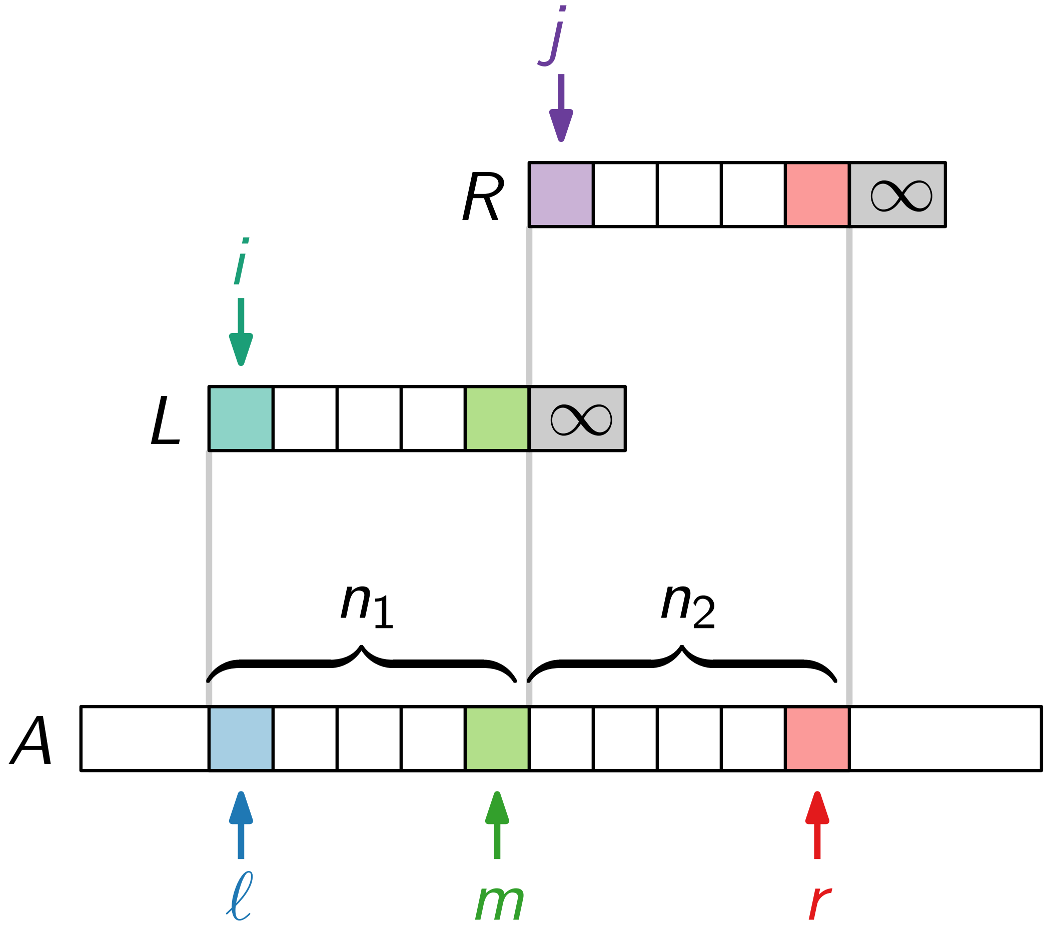
$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m + 1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\quad n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \ldots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \ldots n_2 + 1]$

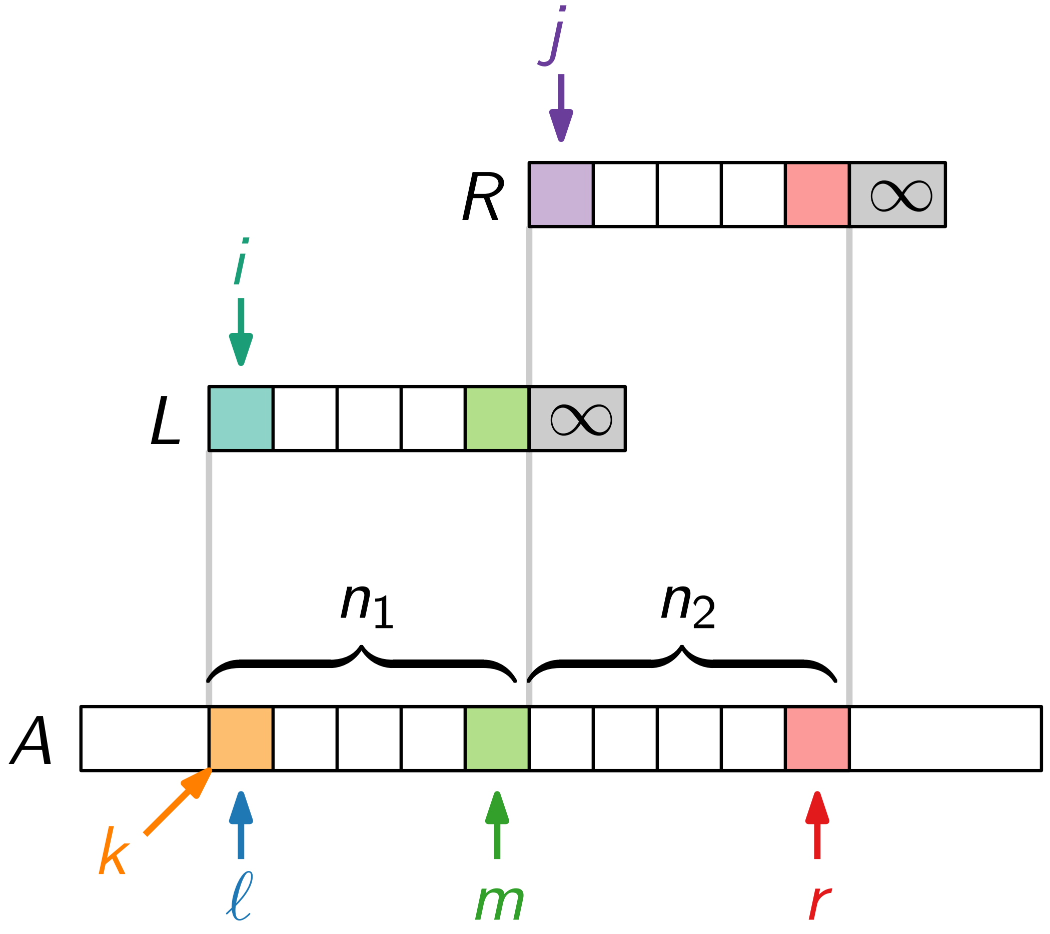
$L[1 \ldots n_1] = A[\ell \ldots m]$

$R[1 \ldots n_2] = A[m + 1 \ldots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for $k = \ell$ **to** r **do**



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1; \quad n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

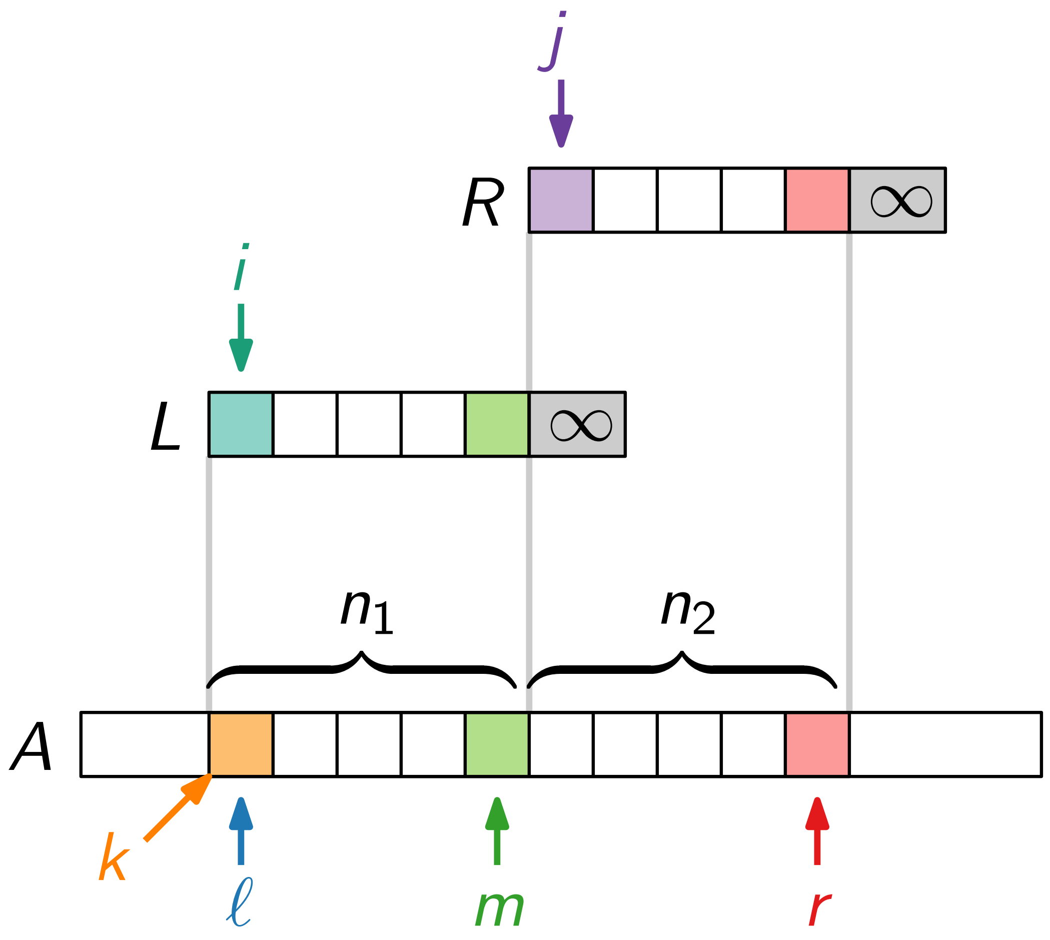
$R[1 \dots n_2] = A[m + 1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for $k = \ell$ **to** r **do**

Aufgabe.
Schreiben Sie den Rest der Routine auf ein Stück Papier!
Benutzen Sie dazu L und R.
[5 Minuten]



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m + 1 \dots r]$

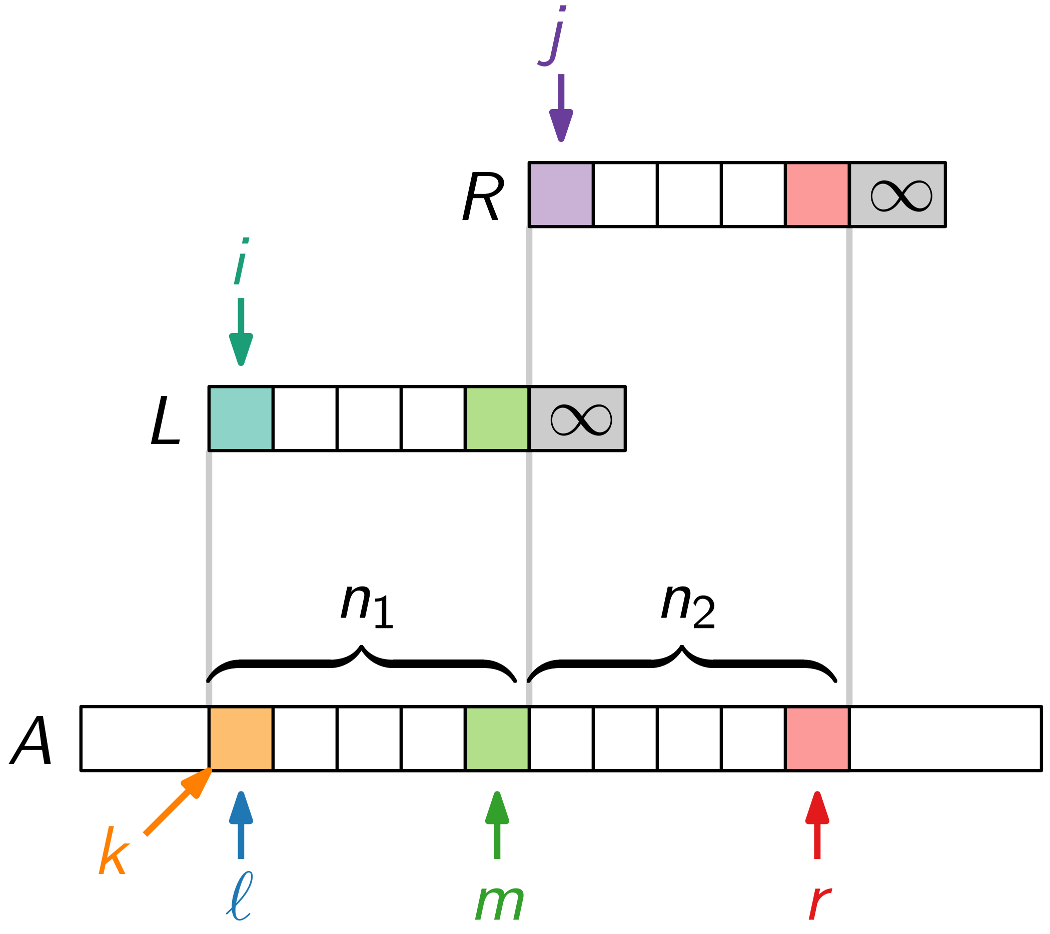
$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for $k = \ell$ **to** r **do**

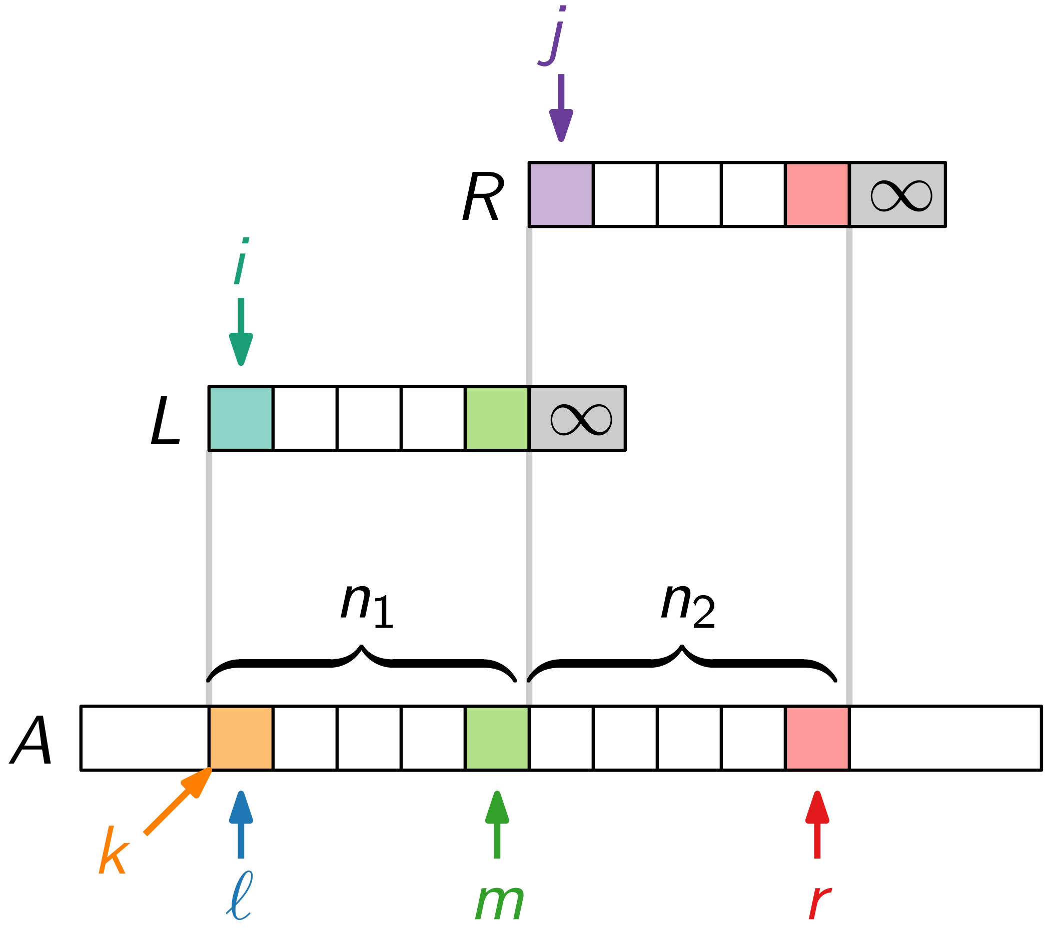
  if $L[i] \leq R[j]$ **then**

  else



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$
  $n_1 = m - \ell + 1;\ \ n_2 = r - m$
  $L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$
  $L[1 \dots n_1] = A[\ell \dots m]$
  $R[1 \dots n_2] = A[m + 1 \dots r]$
  $L[n_1 + 1] = R[n_2 + 1] = \infty$
  $i = j = 1$
  for $k = \ell$ **to** r **do**
    if $L[i] \leq R[j]$ **then**
      $A[k] = L[i]$
      $i = i + 1$
    else



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m + 1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

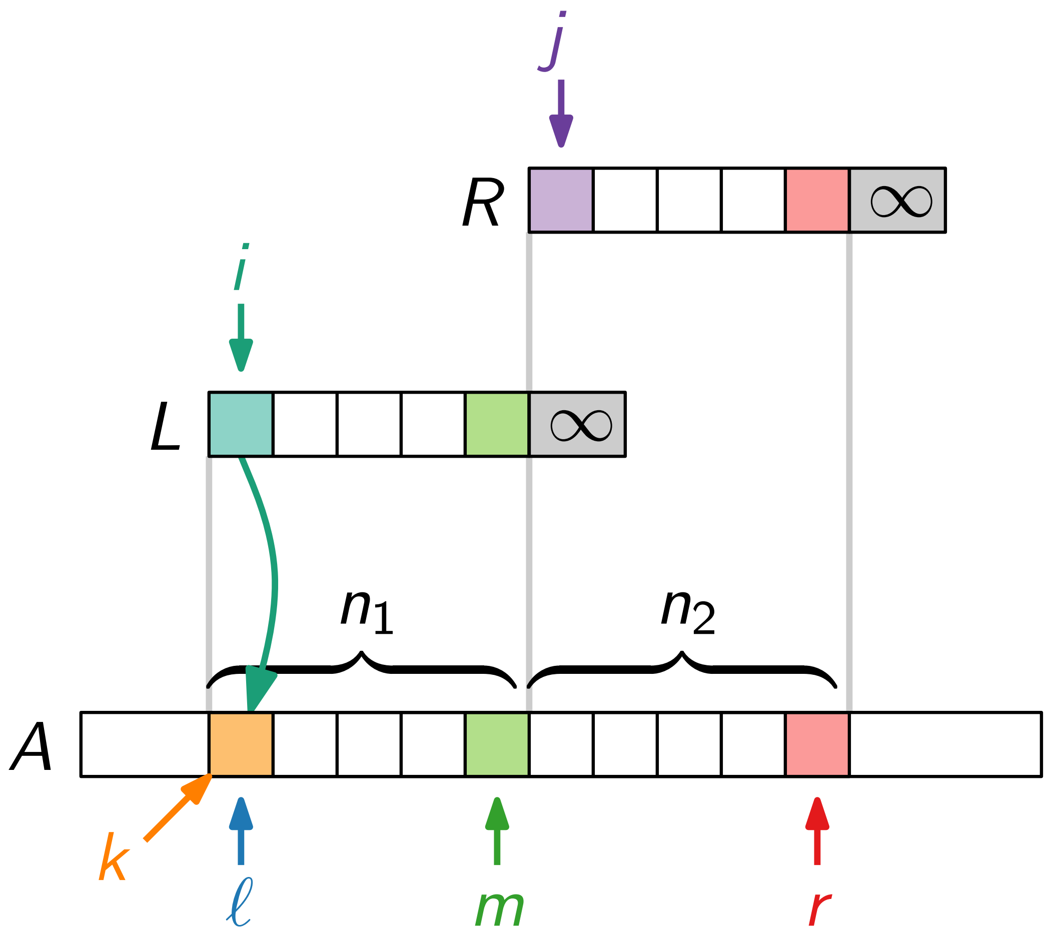
for $k = \ell$ **to** r **do**

    if $L[i] \leq R[j]$ **then**

        $A[k] = L[i]$

        $i = i + 1$

    else



Kombiniere

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$

$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m + 1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

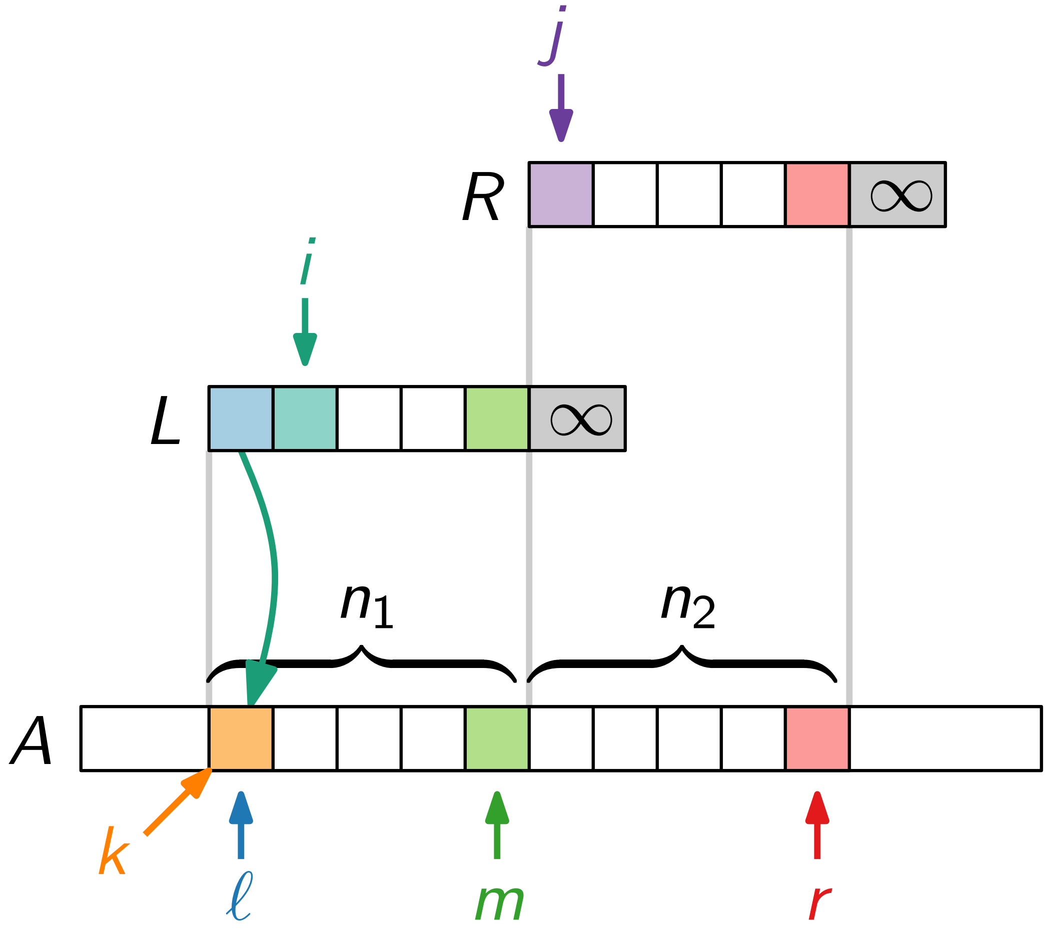
for $k = \ell$ **to** r **do**

  if $L[i] \leq R[j]$ **then**

    $A[k] = L[i]$

    $i = i + 1$

  else



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \ldots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \ldots n_2 + 1]$

$L[1 \ldots n_1] = A[\ell \ldots m]$

$R[1 \ldots n_2] = A[m + 1 \ldots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

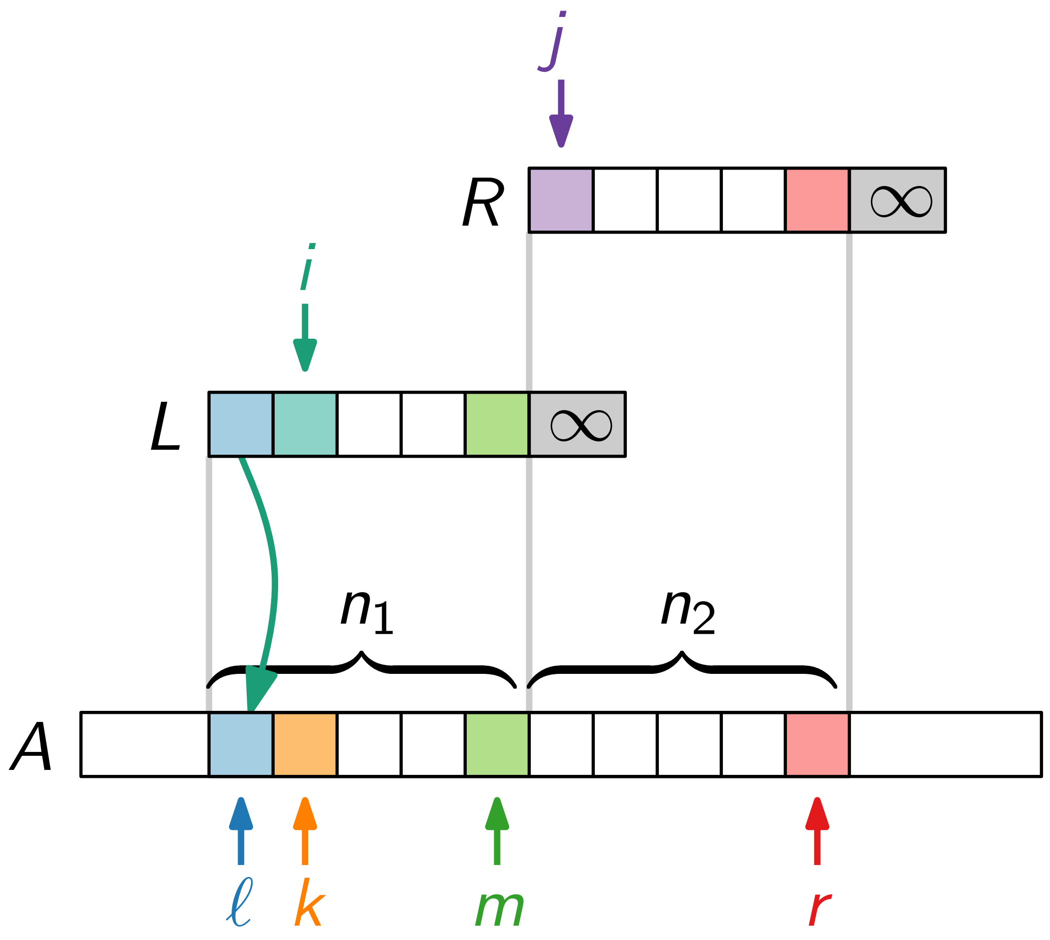
for $k = \ell$ **to** r **do**

  if $L[i] \leq R[j]$ **then**

    $A[k] = L[i]$

    $i = i + 1$

  else



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A,\ \text{int}\ \ell,\ \text{int}\ m,\ \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

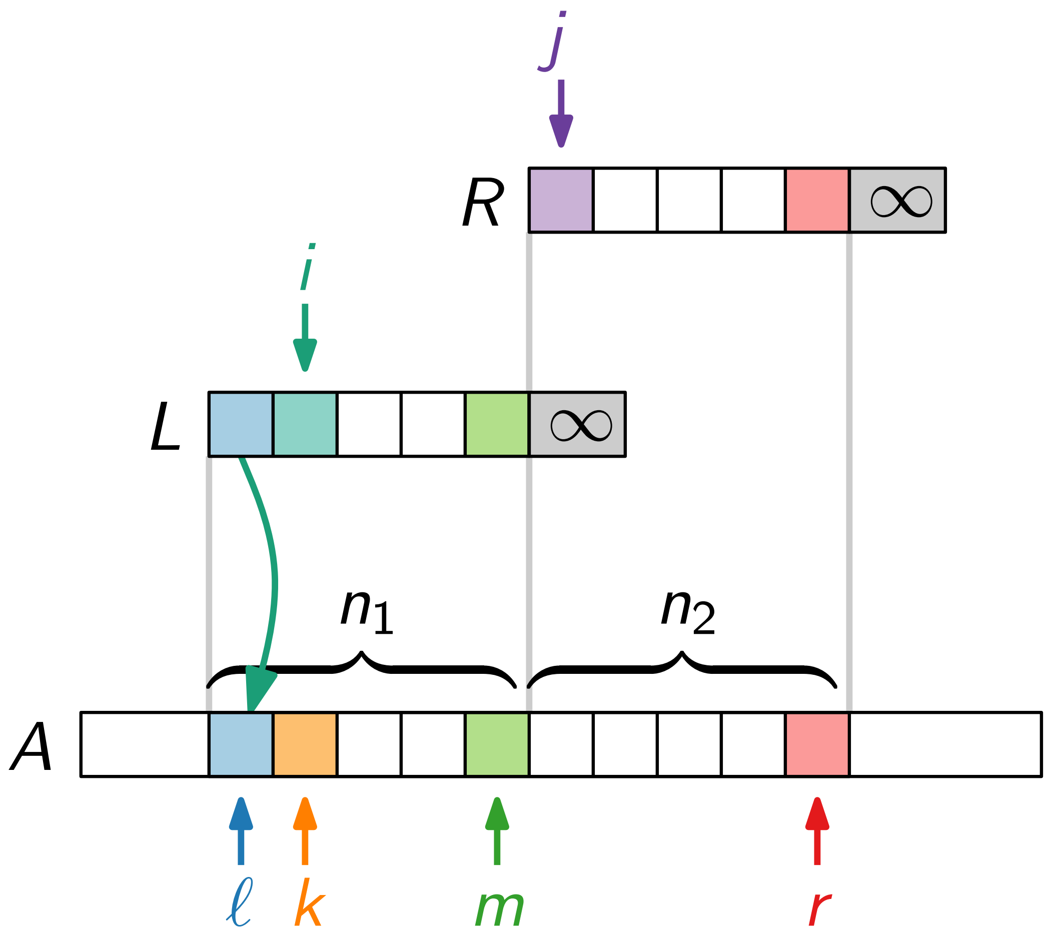
$R[1 \dots n_2] = A[m + 1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for $k = \ell$ **to** r **do**

- **if** $L[i] \le R[j]$ **then**
  - $A[k] = L[i]$
  - $i = i + 1$
- **else**
  - $A[k] = R[j]$
  - $j = j + 1$



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$\quad n_1 = m - \ell + 1;\ \ n_2 = r - m$

$\quad L = \textbf{new}\ \text{int}[1 \ldots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \ldots n_2 + 1]$

$\quad L[1 \ldots n_1] = A[\ell \ldots m]$

$\quad R[1 \ldots n_2] = A[m + 1 \ldots r]$

$\quad L[n_1 + 1] = R[n_2 + 1] = \infty$

$\quad i = j = 1$

$\quad \textbf{for}\ k = \ell\ \textbf{to}\ r\ \textbf{do}$

$\qquad \textbf{if}\ L[i] \leq R[j]\ \textbf{then}$

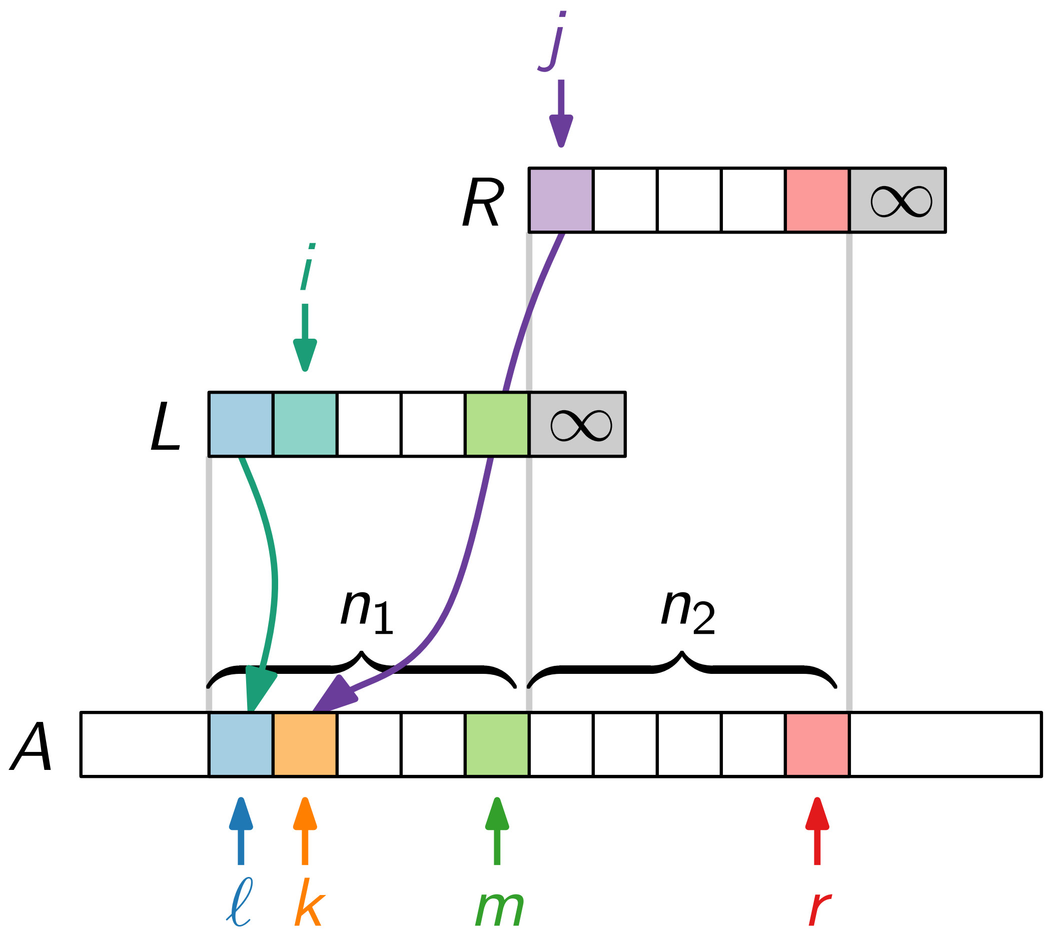
$\qquad\quad A[k] = L[i]$

$\qquad\quad i = i + 1$

$\qquad \textbf{else}$

$\qquad\quad A[k] = R[j]$

$\qquad\quad j = j + 1$



Kombiniere

MERGE(int[] A, int ℓ, int m, int r)

  $n_1 = m - \ell + 1$; $n_2 = r - m$

  $L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$

  $L[1 \dots n_1] = A[\ell \dots m]$

  $R[1 \dots n_2] = A[m + 1 \dots r]$

  $L[n_1 + 1] = R[n_2 + 1] = \infty$

  $i = j = 1$

  for $k = \ell$ **to** r **do**

    if $L[i] \leq R[j]$ **then**

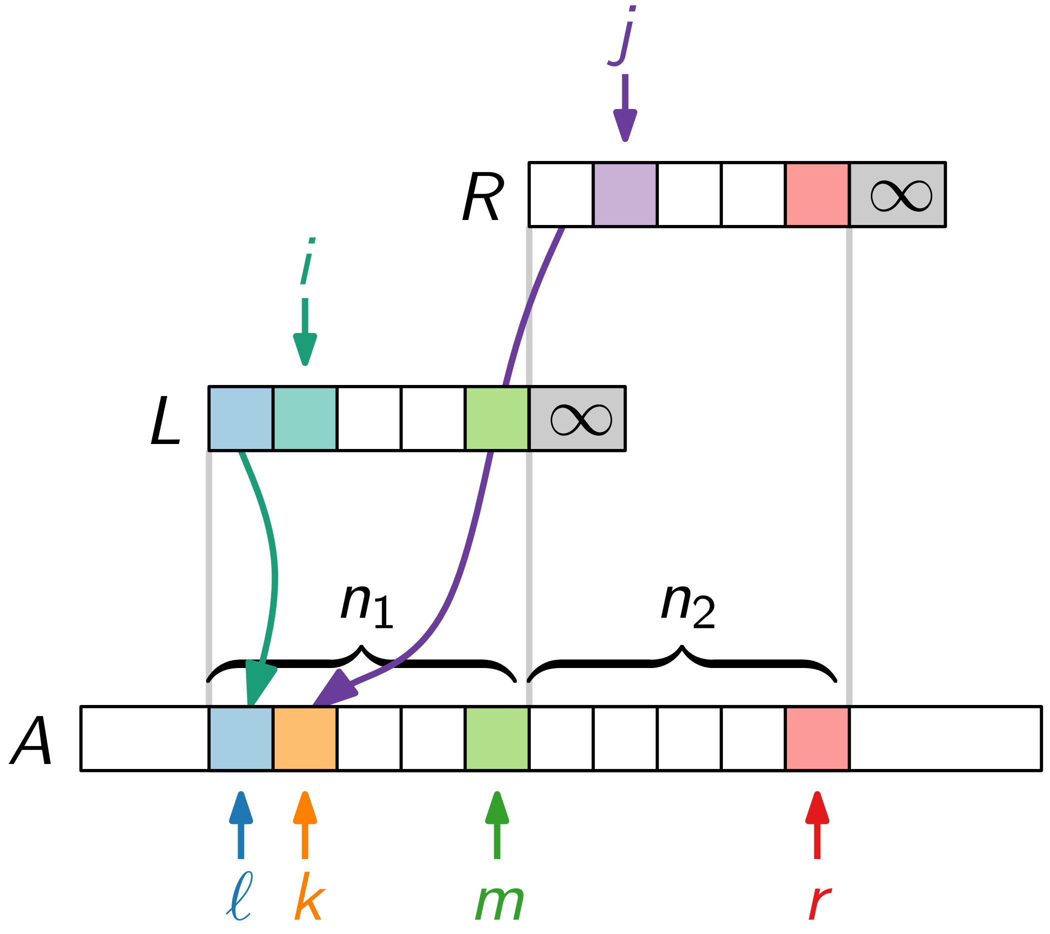
      $A[k] = L[i]$

      $i = i + 1$

    else

      $A[k] = R[j]$

      $j = j + 1$



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ m, \text{int}\ r)$

$n_1 = m - \ell + 1;\ \ n_2 = r - m$

$L = \textbf{new}\ \text{int}[1 \ldots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \ldots n_2 + 1]$

$L[1 \ldots n_1] = A[\ell \ldots m]$

$R[1 \ldots n_2] = A[m + 1 \ldots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for $k = \ell$ **to** r **do**

  if $L[i] \leq R[j]$ **then**

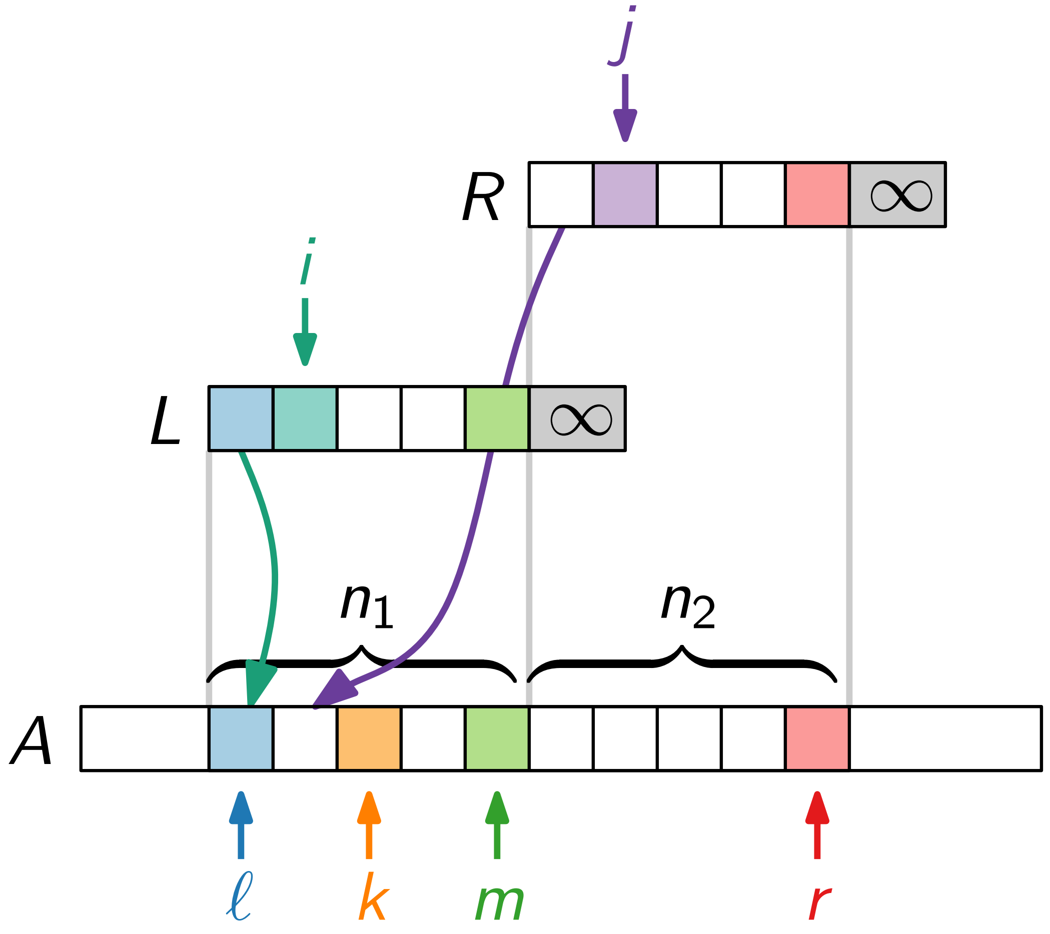
    $A[k] = L[i]$

    $i = i + 1$

  else

    $A[k] = R[j]$

    $j = j + 1$



Kombiniere

$\textsc{Merge}(\text{int}[\,]\ A,\ \text{int}\ \ell,\ \text{int}\ m,\ \text{int}\ r)$

  $n_1 = m - \ell + 1;\ \ n_2 = r - m$

  $L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

  $L[1 \dots n_1] = A[\ell \dots m]$

  $R[1 \dots n_2] = A[m + 1 \dots r]$

  $L[n_1 + 1] = R[n_2 + 1] = \infty$

  $i = j = 1$

  for $k = \ell$ **to** r **do**

    if $L[i] \leq R[j]$ **then**

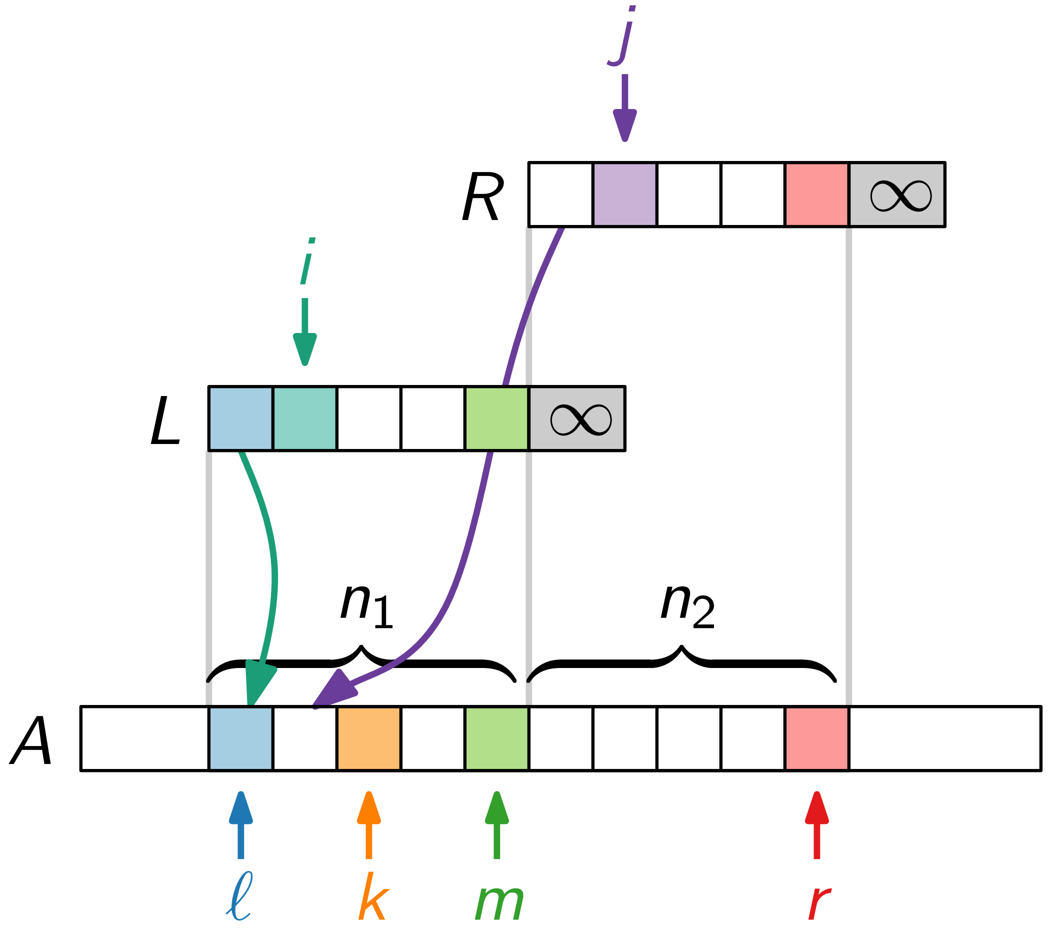
      $A[k] = L[i]$

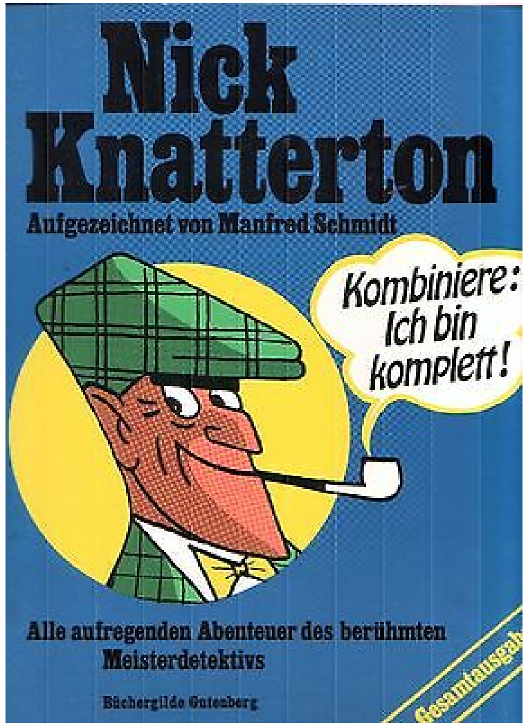
      $i = i + 1$

    else

      $A[k] = R[j]$

      $j = j + 1$





Aber… stimmt das denn alles?

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

Demo.

`https://algo.uni-trier.de/demos/sort.html`

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

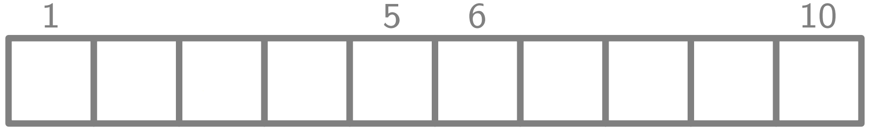
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

1				5	6				10
8	3	4	0	7	9	1	6	5	2

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

1				5	6				10
8	3	4	0	7	9	1	6	5	2

MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

1				5	6				10
8	3	4	0	7	9	1	6	5	2

1				5
8	3	4	0	7

MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

MergeSort(A, ℓ, m)
MergeSort($A, m+1, r$) } **herrsche**

Merge(A, ℓ, m, r) } **kombiniere**

1				5	6				10
8	3	4	0	7	9	1	6	5	2

1		3		5
8	3	4	0	7

MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- MergeSort(A, ℓ, m) } **herrsche**
- MergeSort($A, m+1, r$)
- Merge(A, ℓ, m, r) } **kombiniere**

1				5	6				10
8	3	4	0	7	9	1	6	5	2

1		3		5
8	3	4	0	7

1		3
8	3	4

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

1				5	6				10
8	3	4	0	7	9	1	6	5	2

1		3		5
8	3	4	0	7

1	2	3
8	3	4

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$$\left.\begin{array}{l} m = \lfloor (\ell + r)/2 \rfloor \end{array}\right\}\ \textbf{teile}$$

$$\left.\begin{array}{l} \textsc{MergeSort}(A, \ell, m) \\ \textsc{MergeSort}(A, m+1, r) \end{array}\right\}\ \textbf{herrsche}$$

$$\left.\begin{array}{l} \textsc{Merge}(A, \ell, m, r) \end{array}\right\}\ \textbf{kombiniere}$$

1				5	6				10
8	3	4	0	7	9	1	6	5	2

1		3		5
8	3	4	0	7

1	2	3
8	3	4

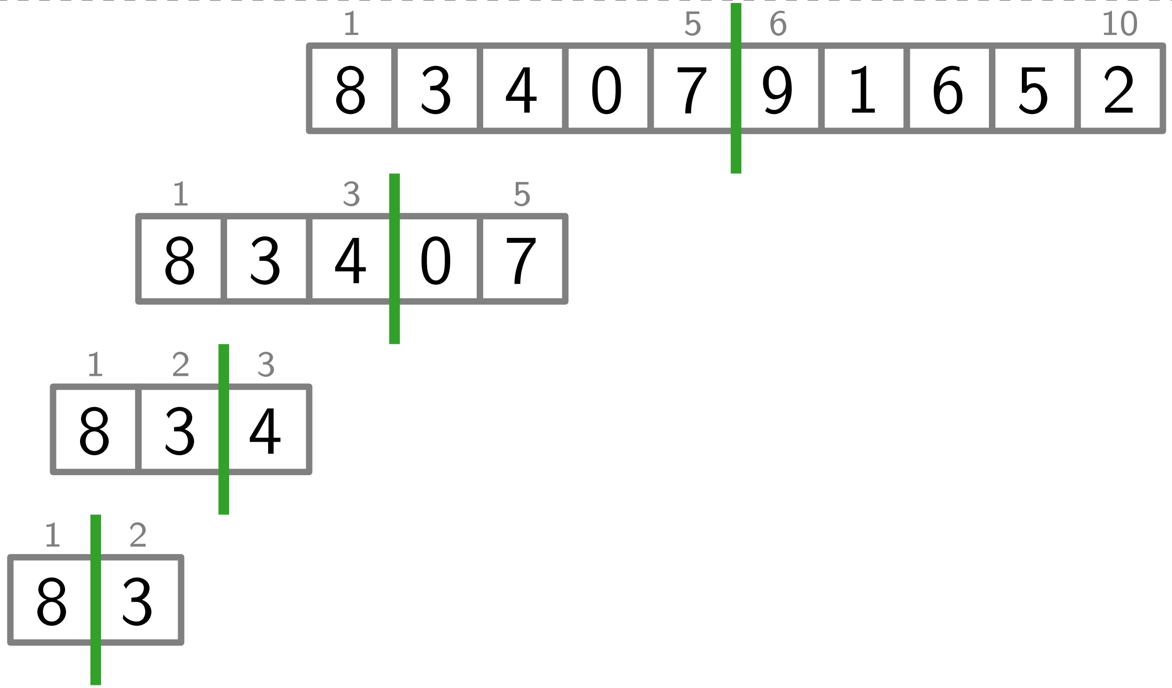
1	2
8	3

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

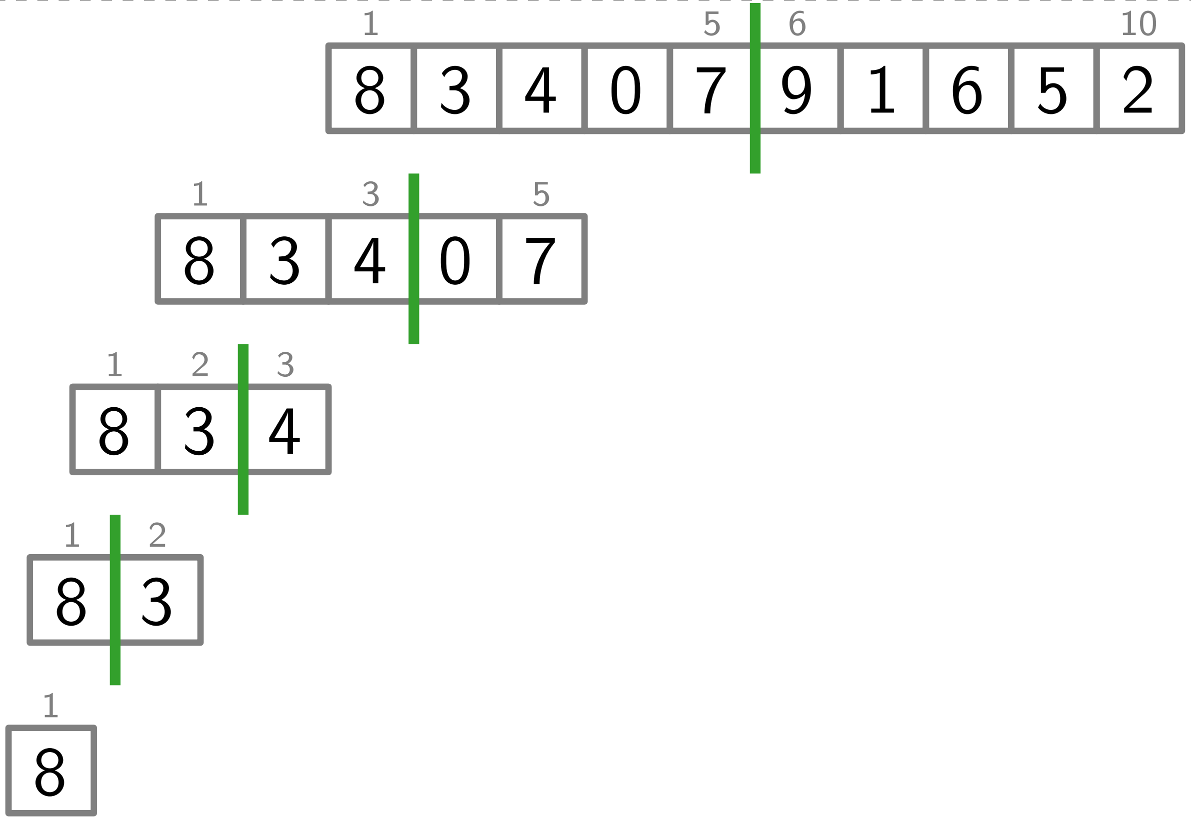


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

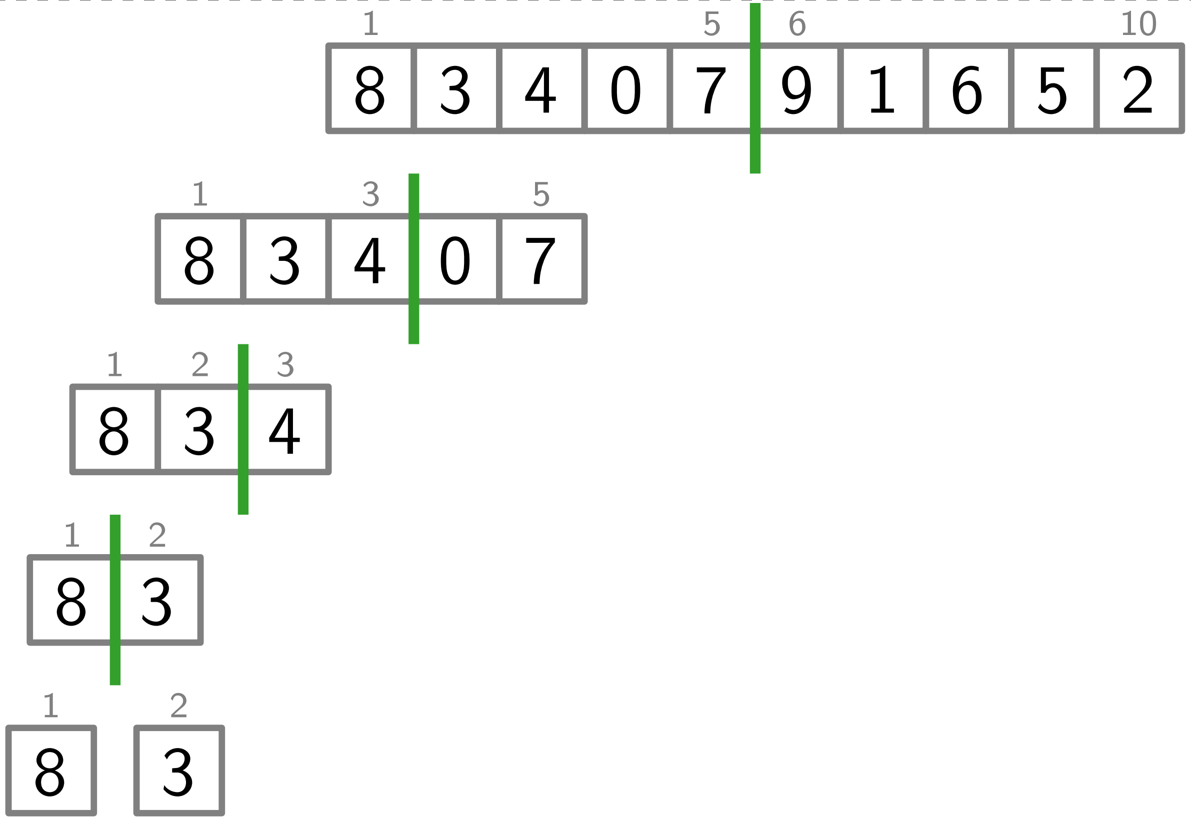


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

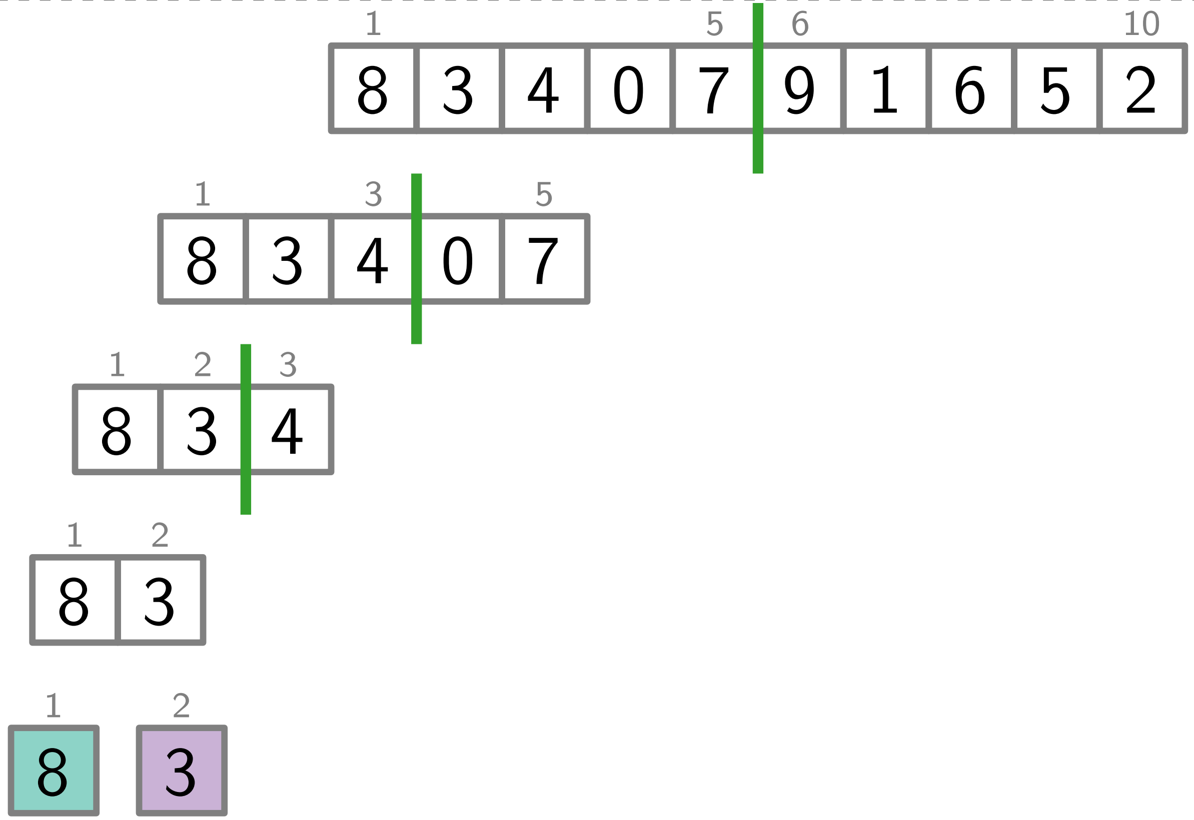
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

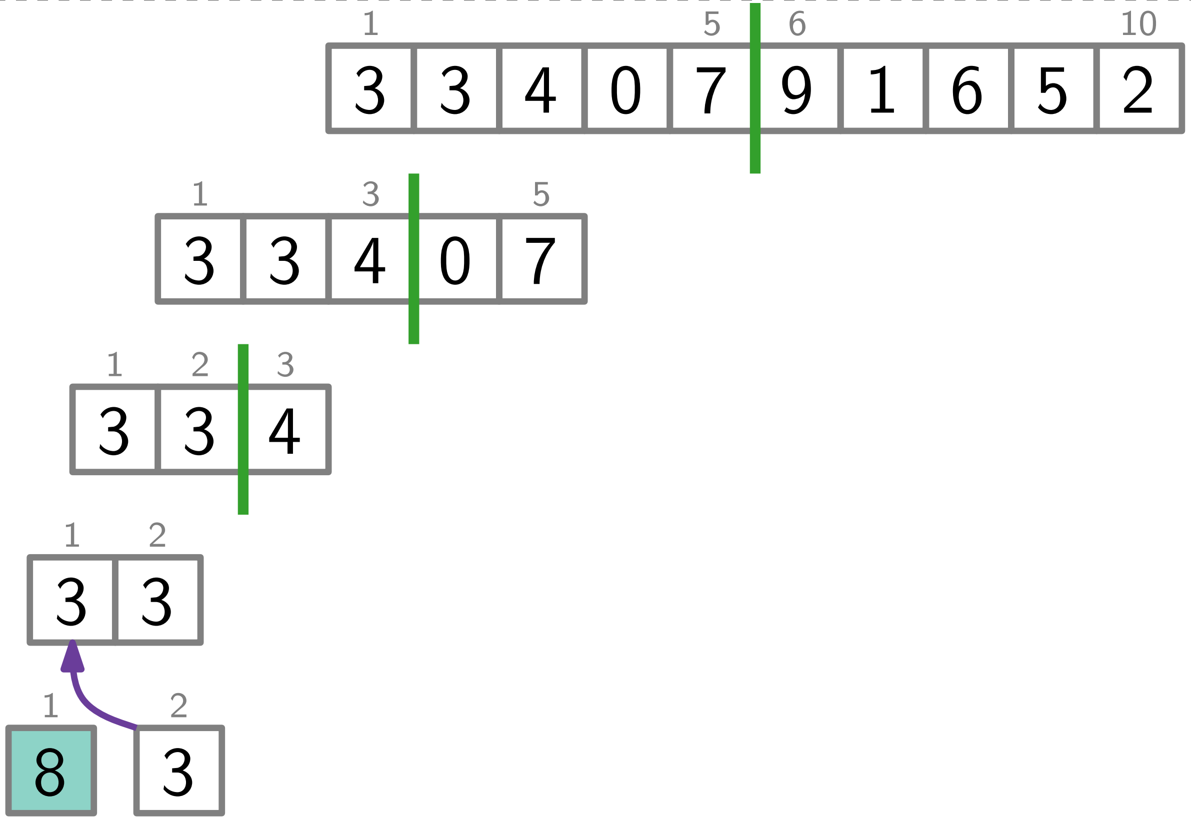


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m + 1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

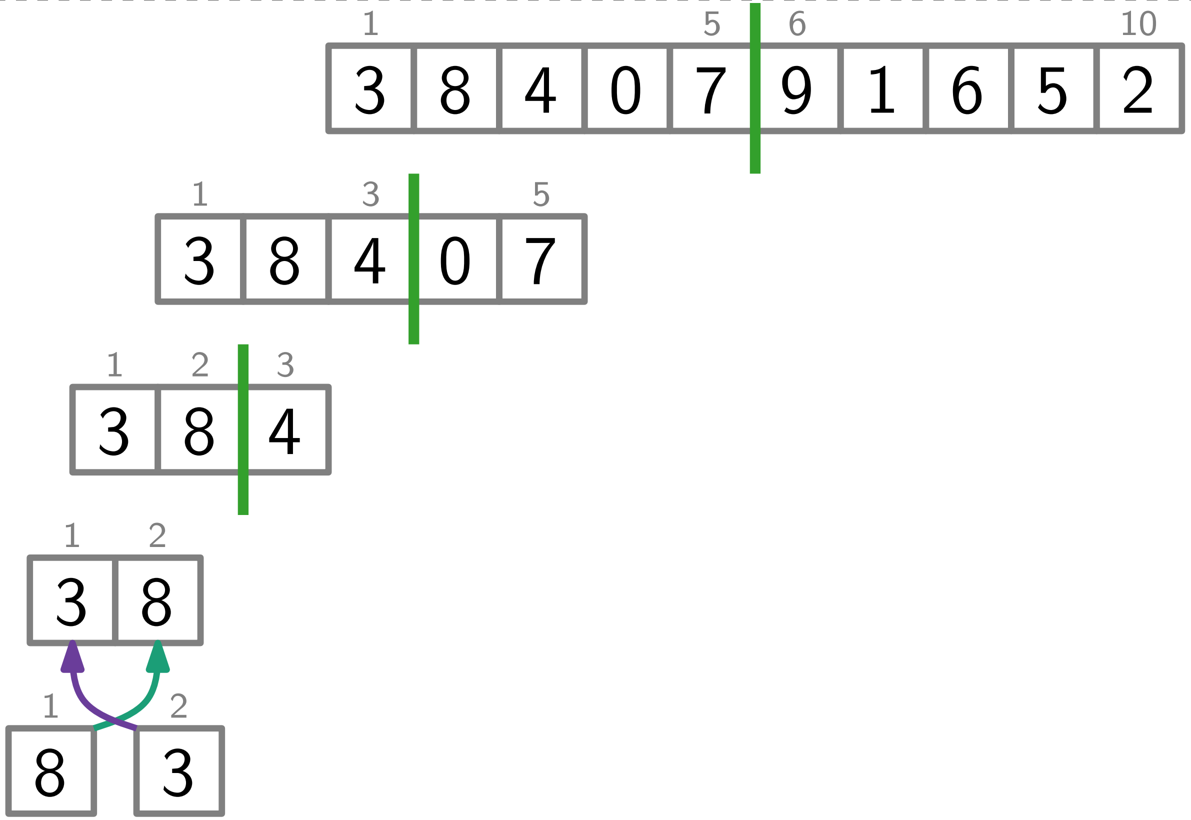


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m + 1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

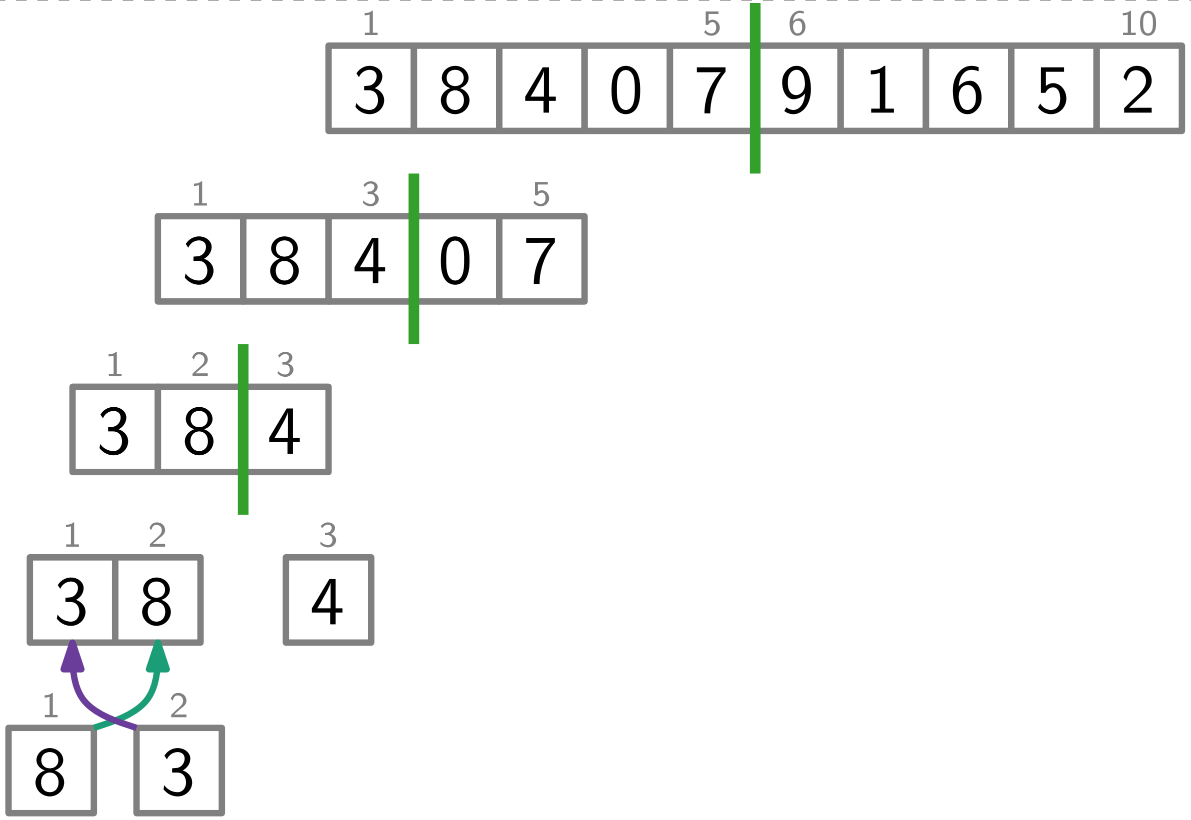
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

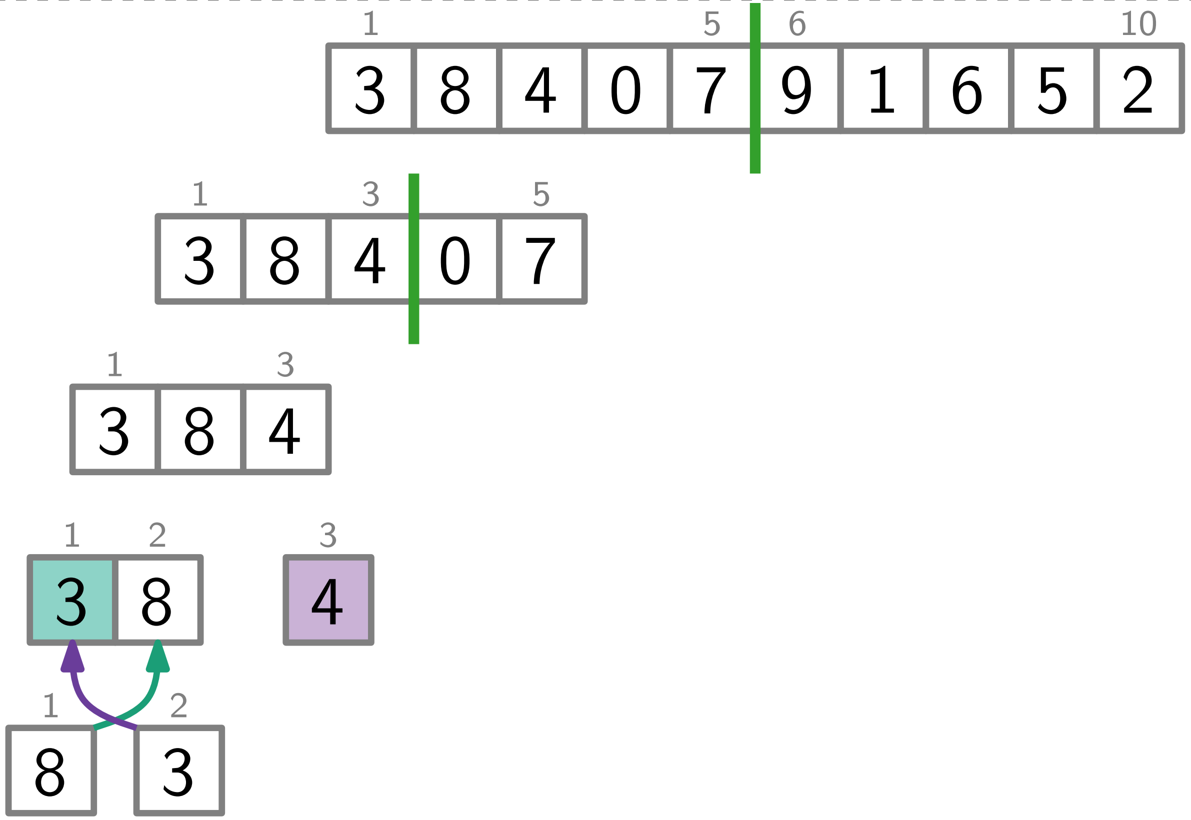
MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

MergeSort(A, ℓ, m)
MergeSort($A, m + 1, r$) } **herrsche**

Merge(A, ℓ, m, r) } **kombiniere**



MergeSort – ein Beispiel

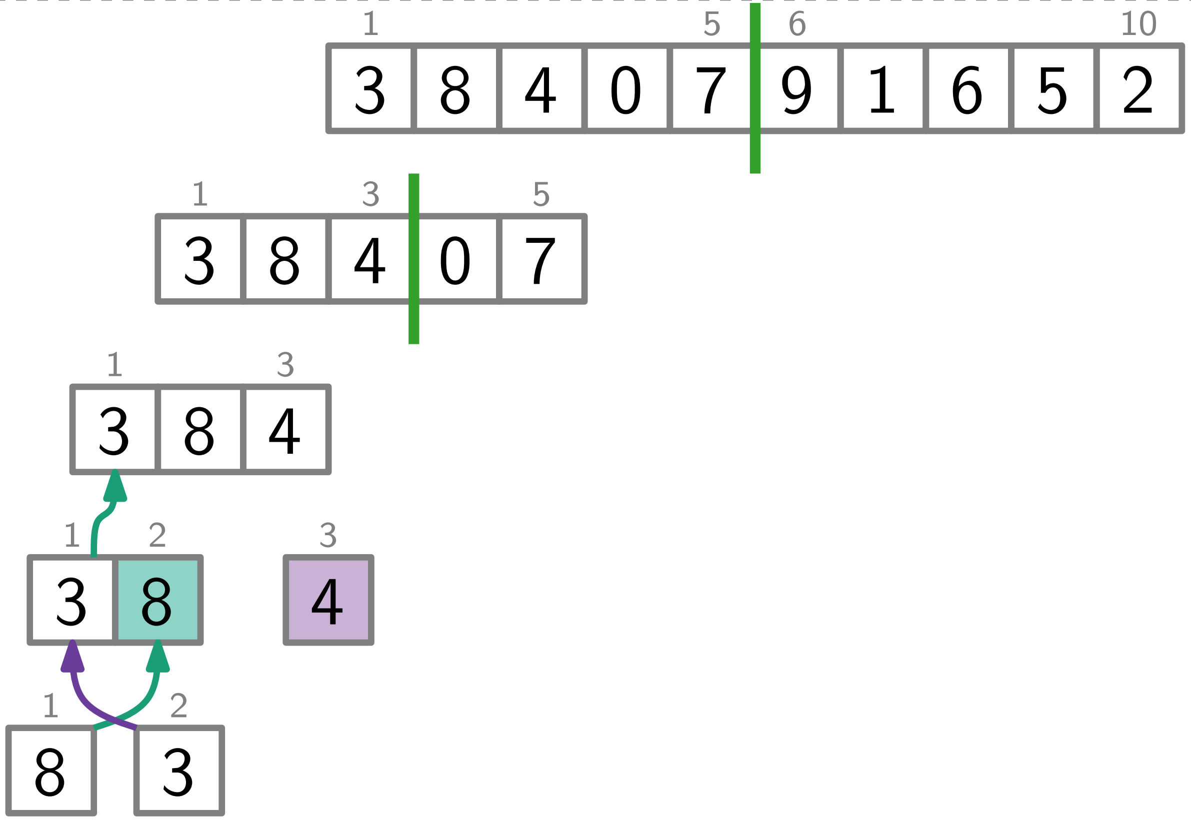
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$$\left.\begin{array}{l} m = \lfloor(\ell + r)/2\rfloor \end{array}\right\}\ \textbf{teile}$$

$$\left.\begin{array}{l} \textsc{MergeSort}(A, \ell, m) \\ \textsc{MergeSort}(A, m+1, r) \end{array}\right\}\ \textbf{herrsche}$$

$$\left.\begin{array}{l} \textsc{Merge}(A, \ell, m, r) \end{array}\right\}\ \textbf{kombiniere}$$



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

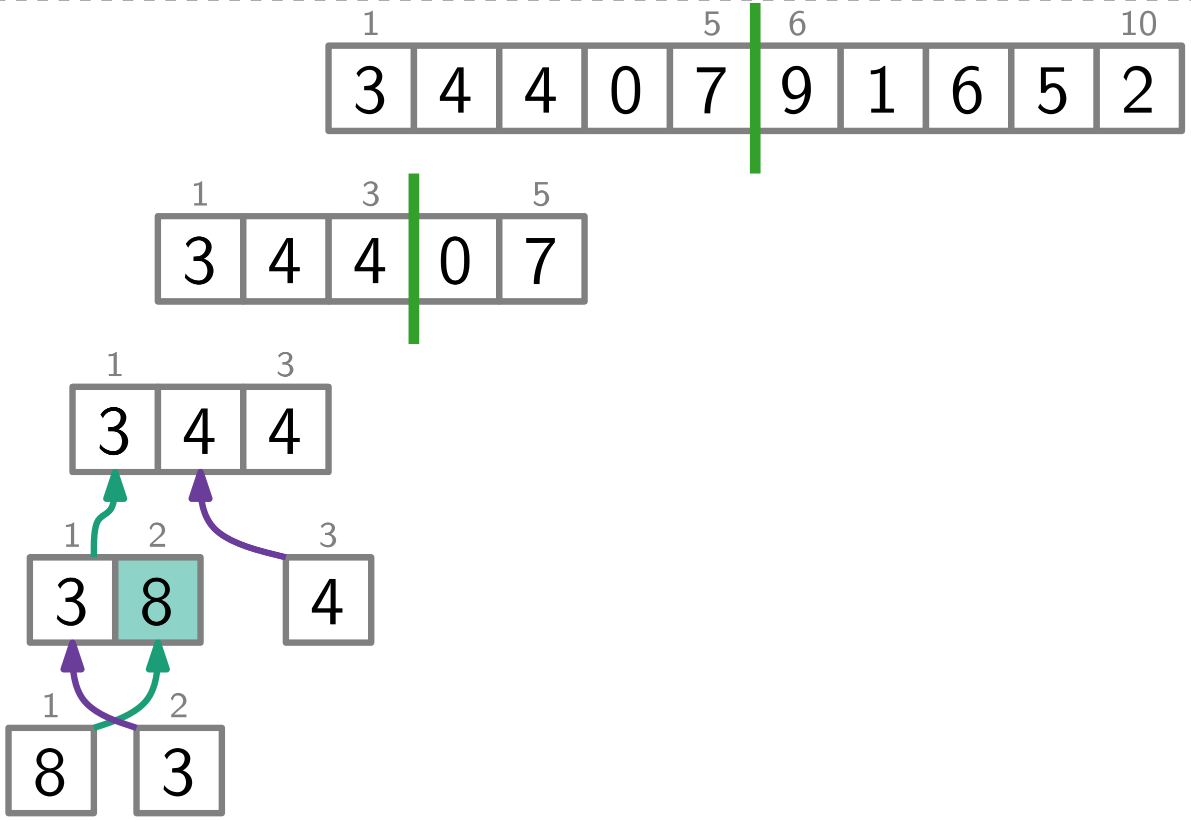
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

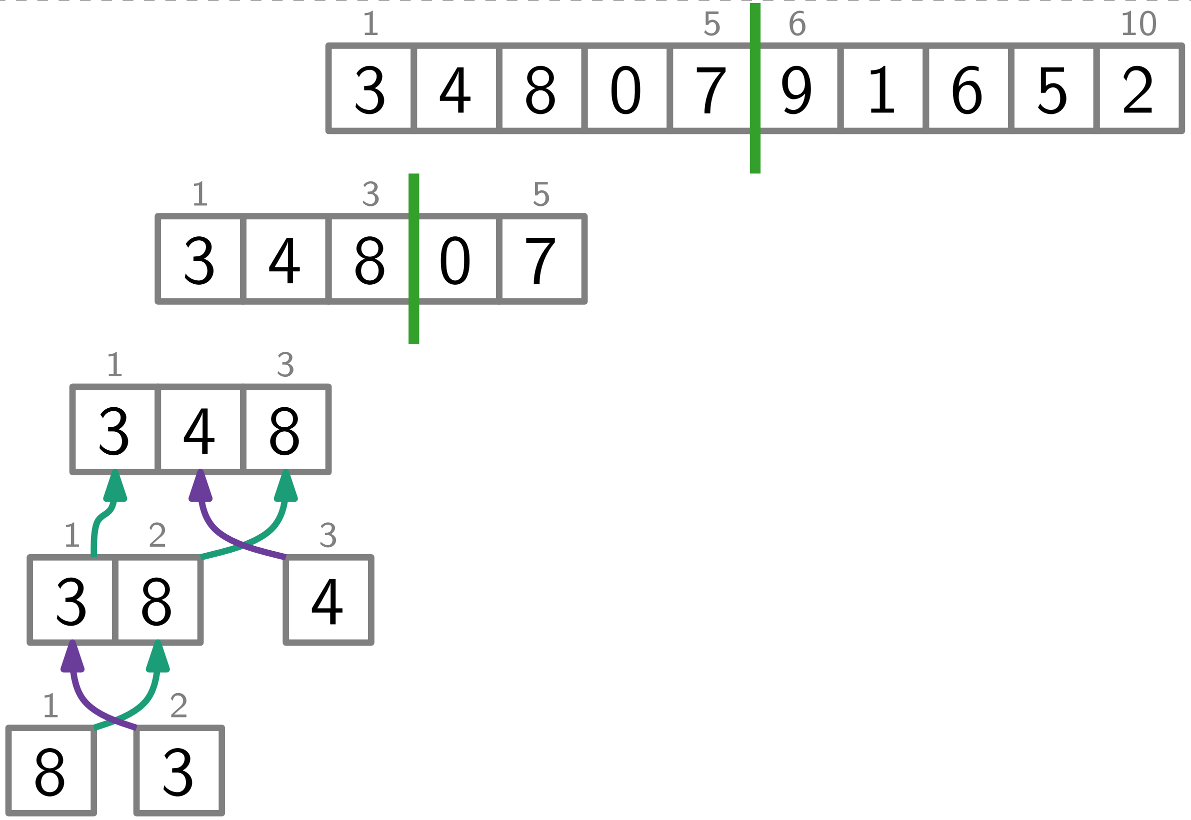


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

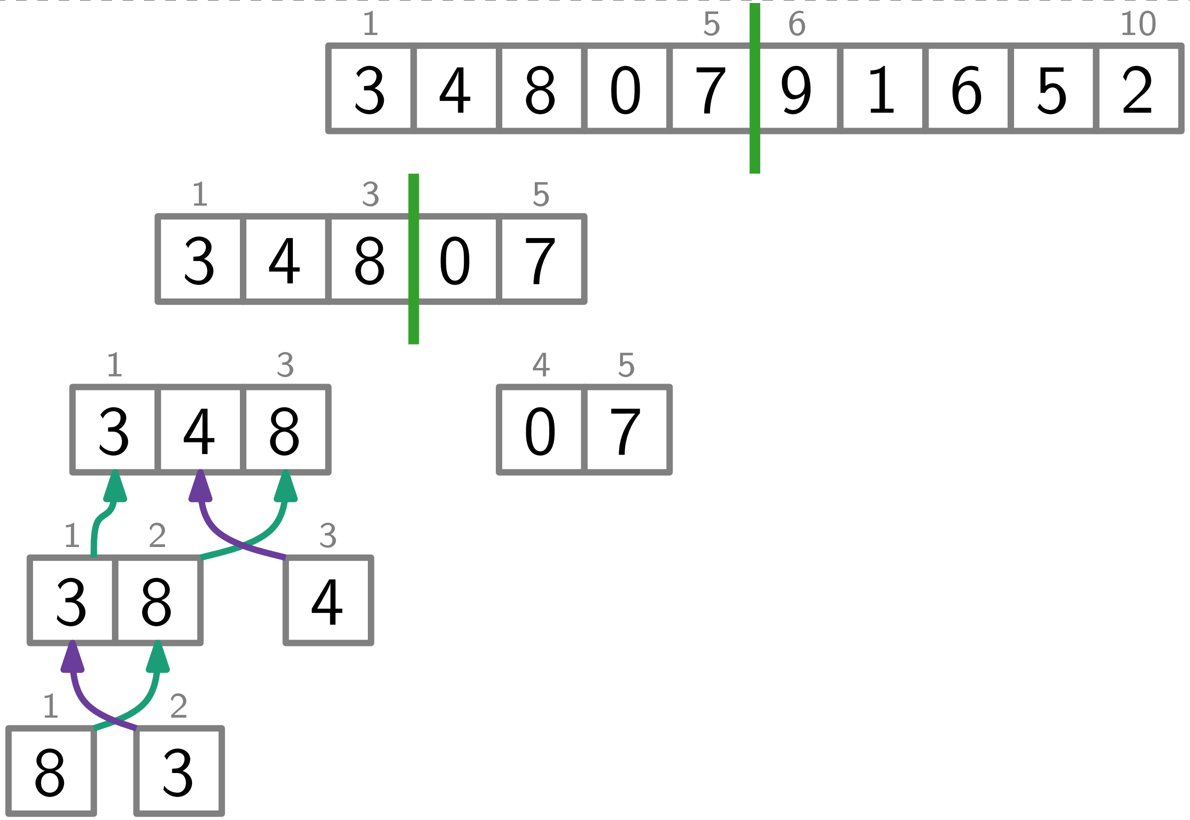
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

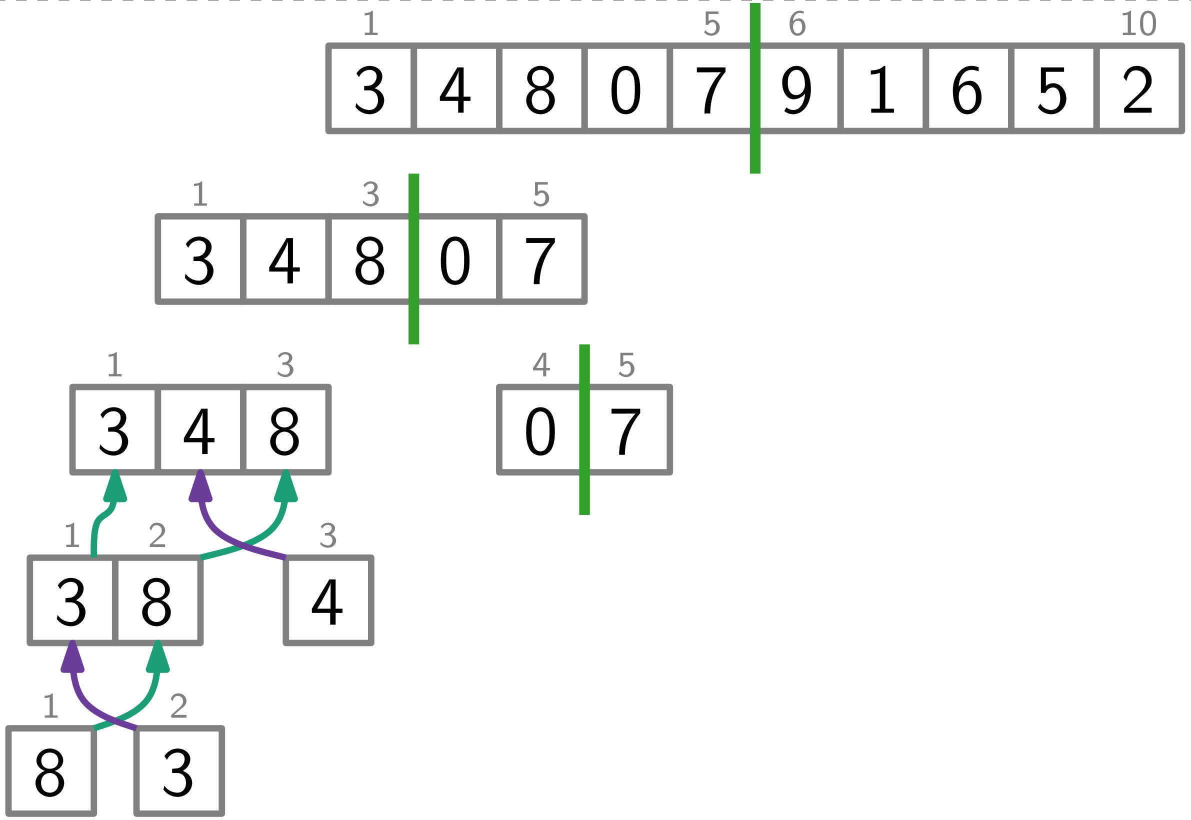


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

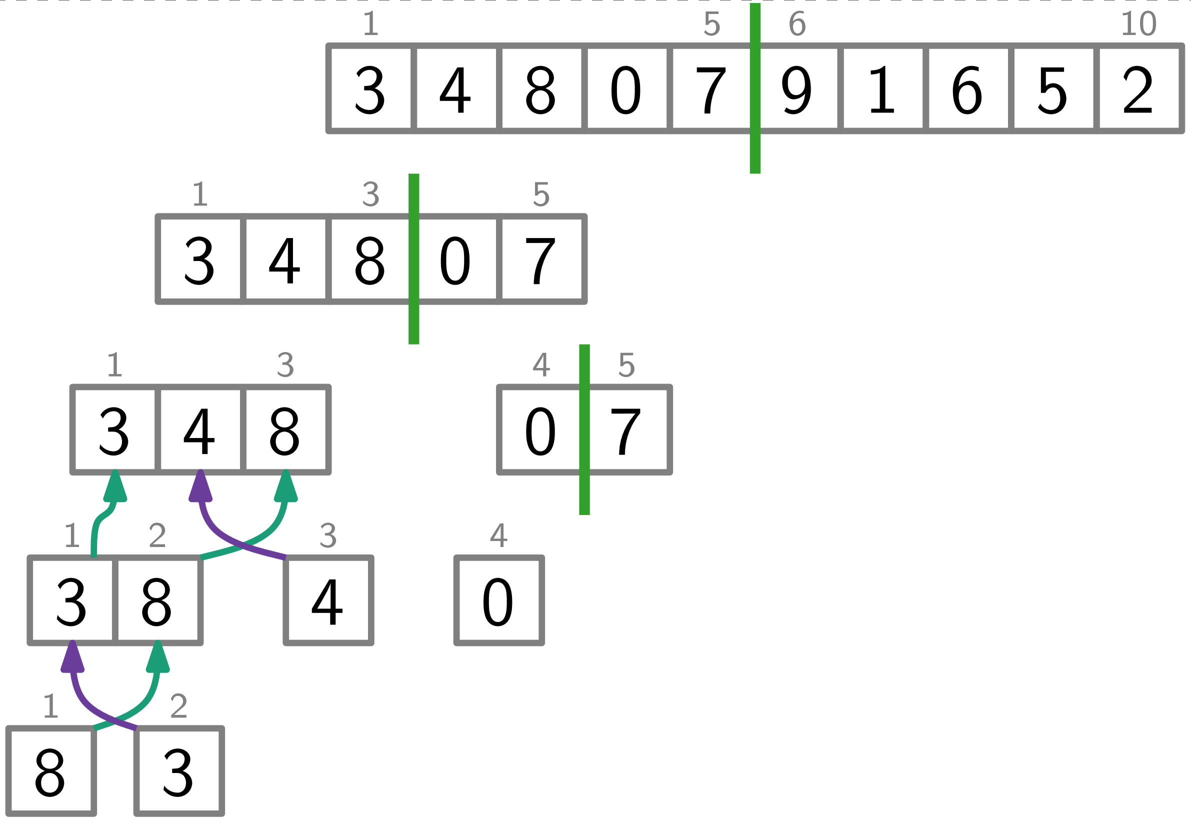
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

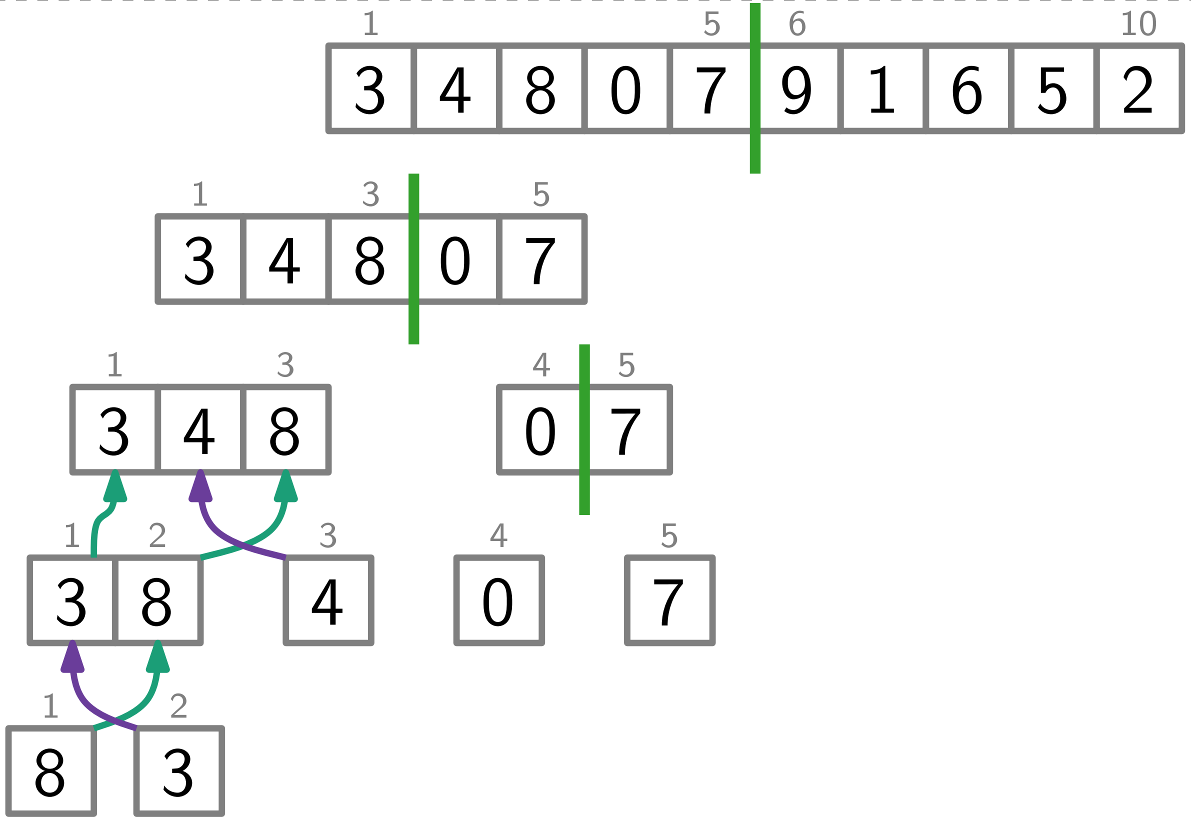


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

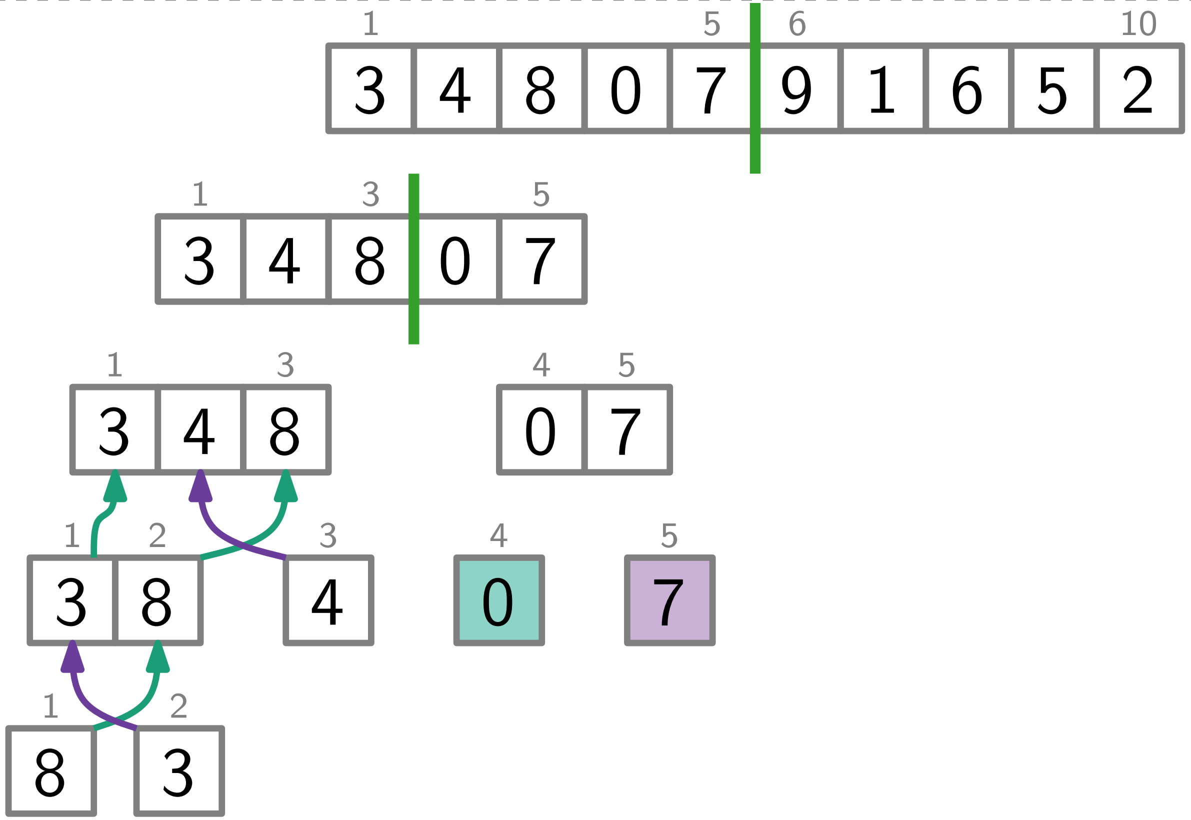


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

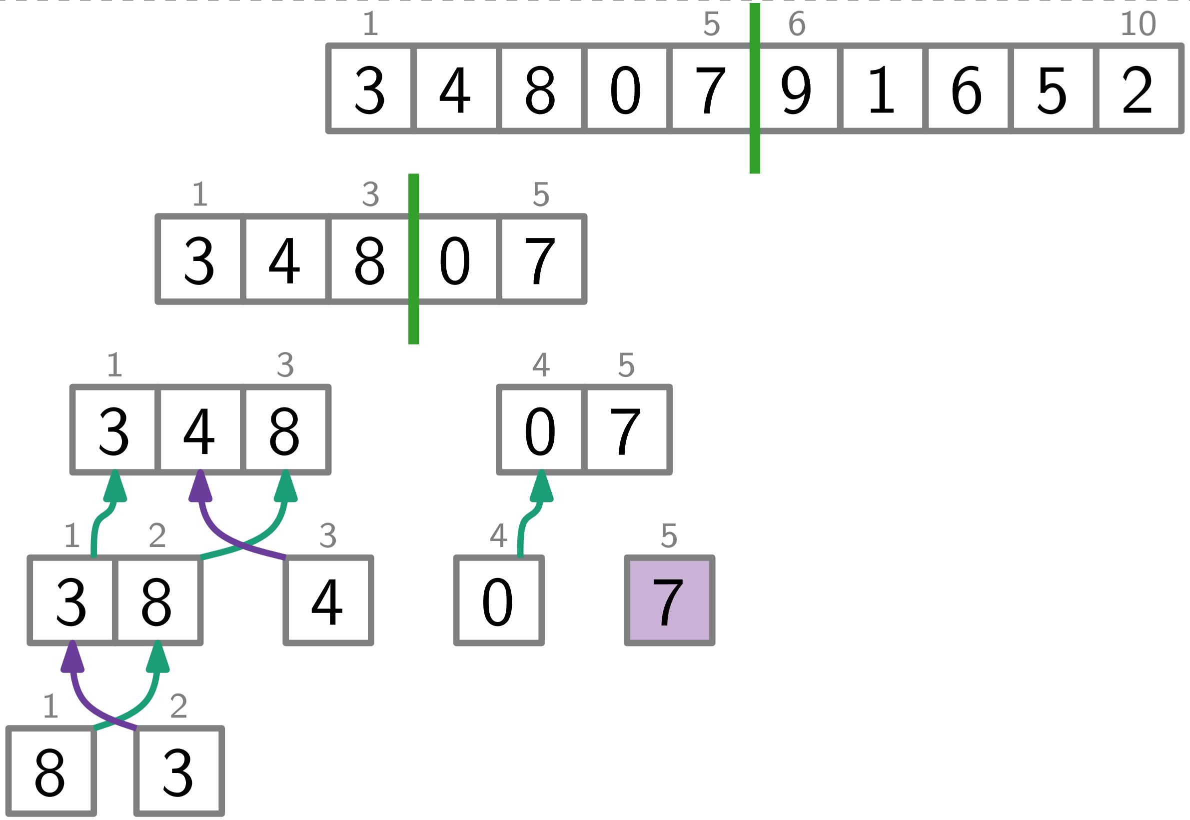


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$ } **herrsche**
- $\textsc{MergeSort}(A, m + 1, r)$
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.\mathit{length})$

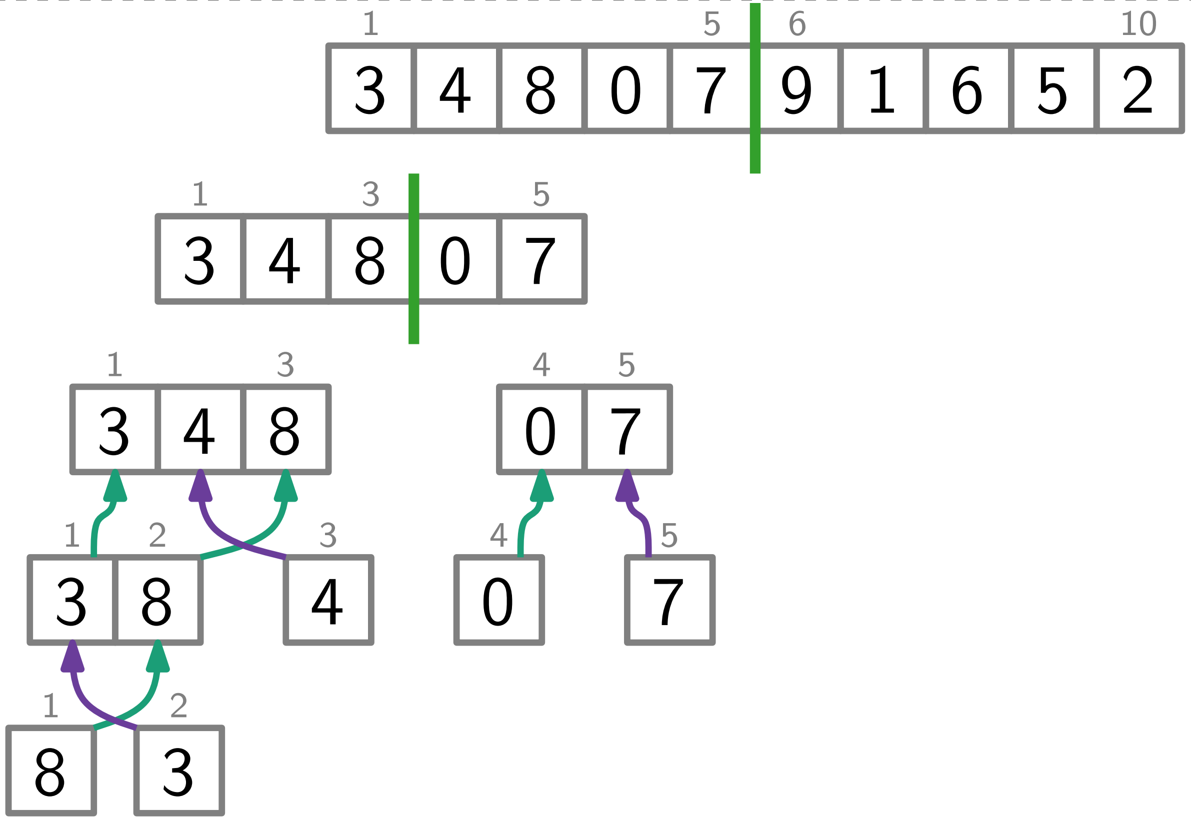
if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

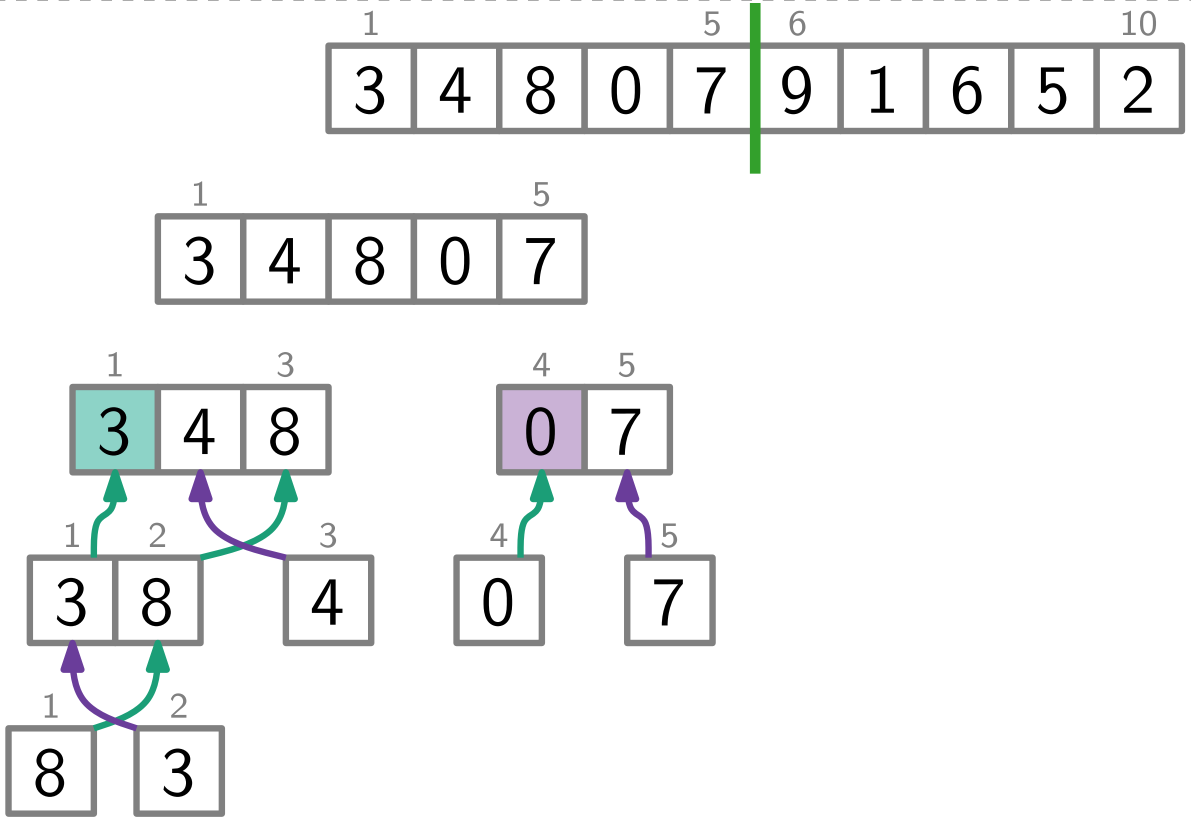


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.\mathit{length})$

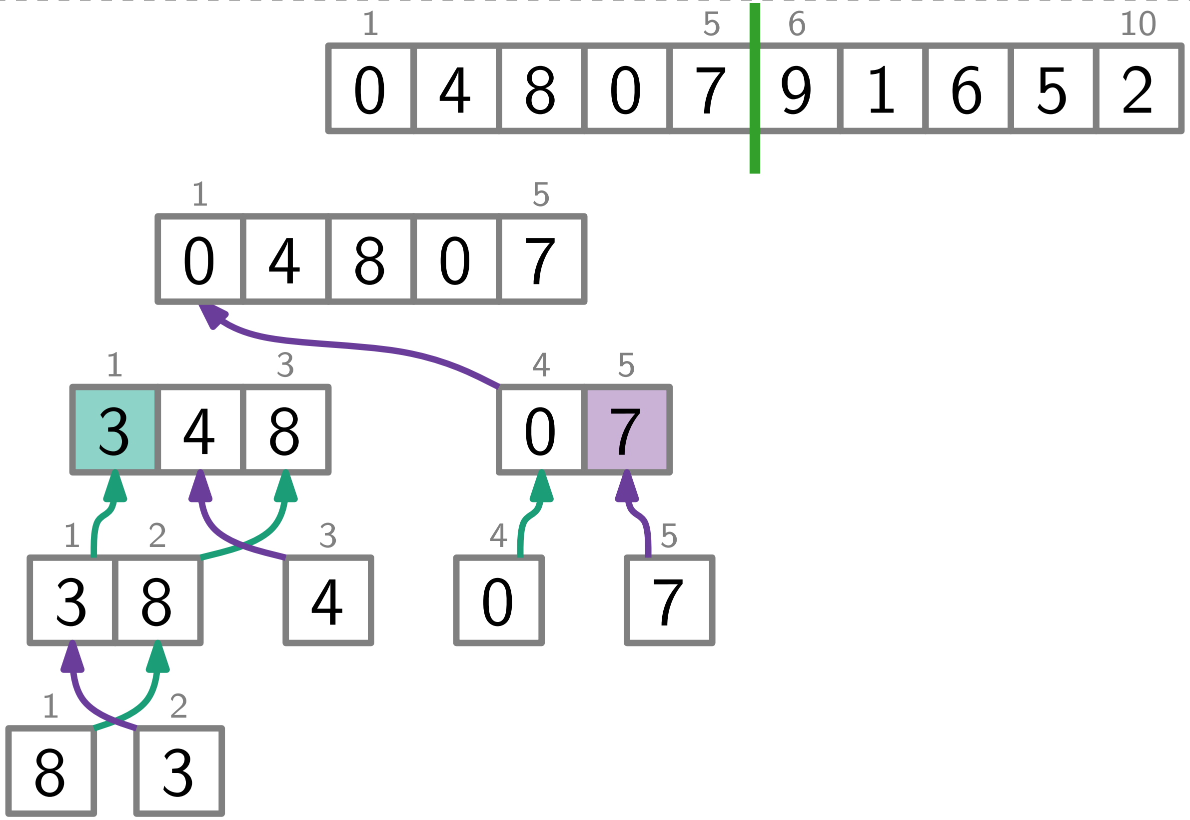
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

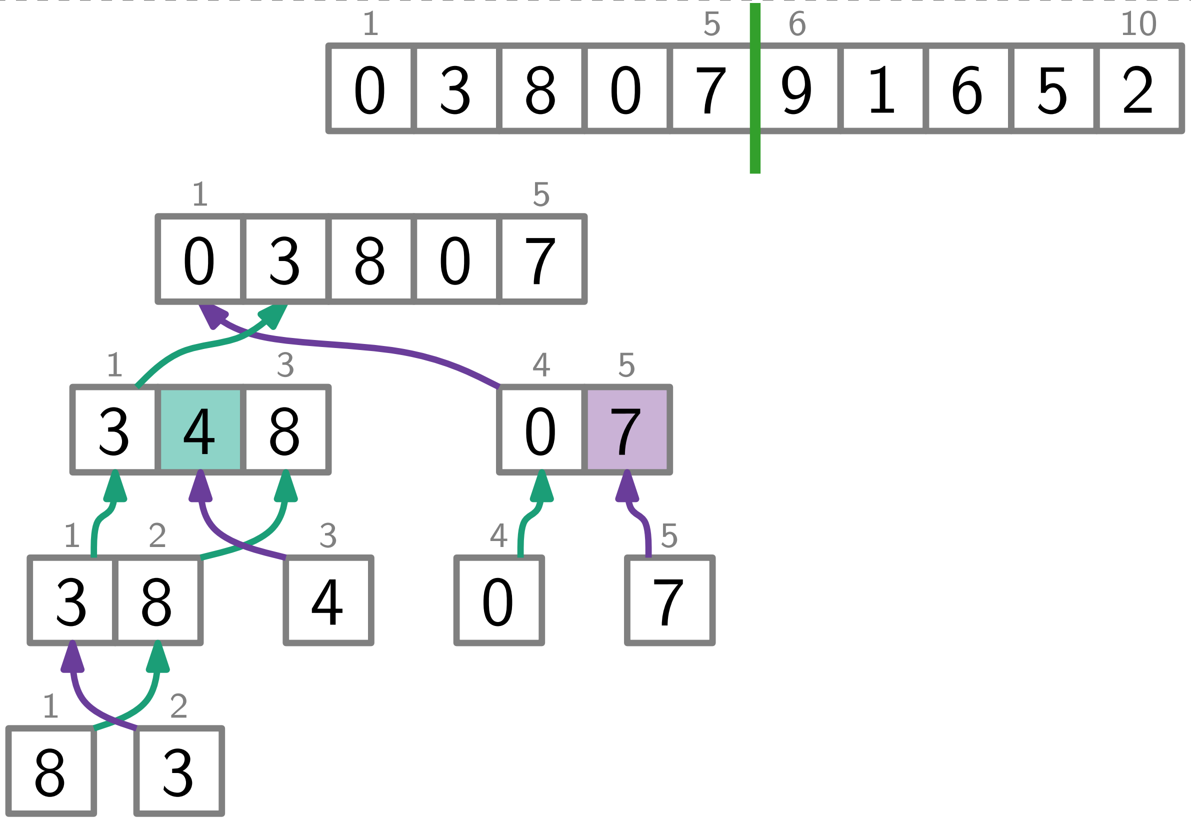


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

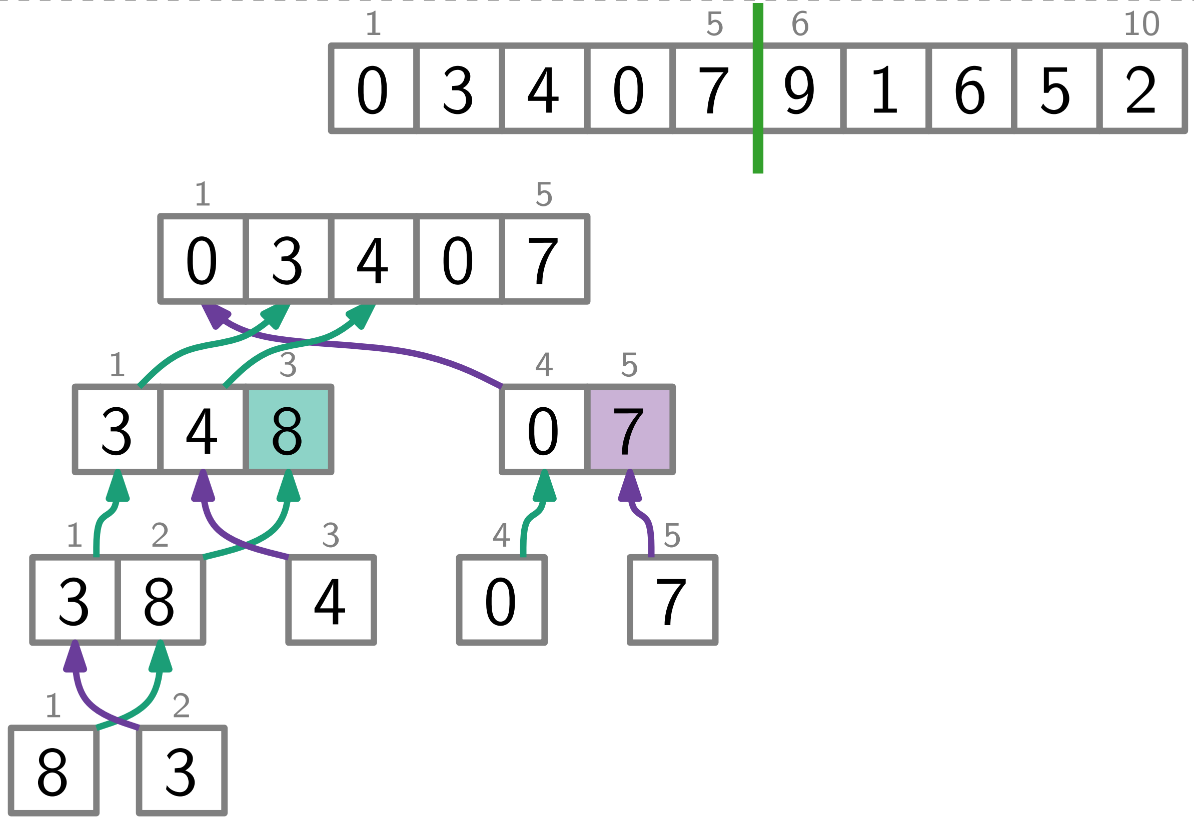


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

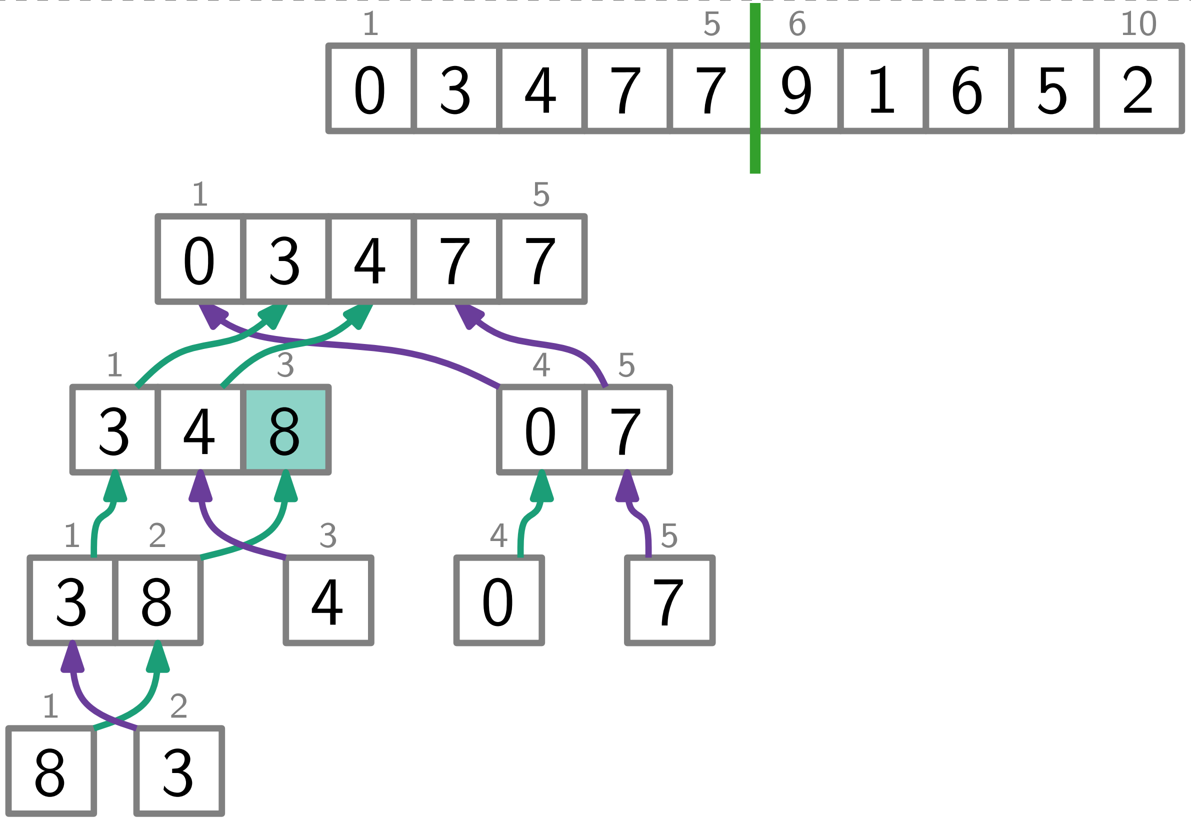


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

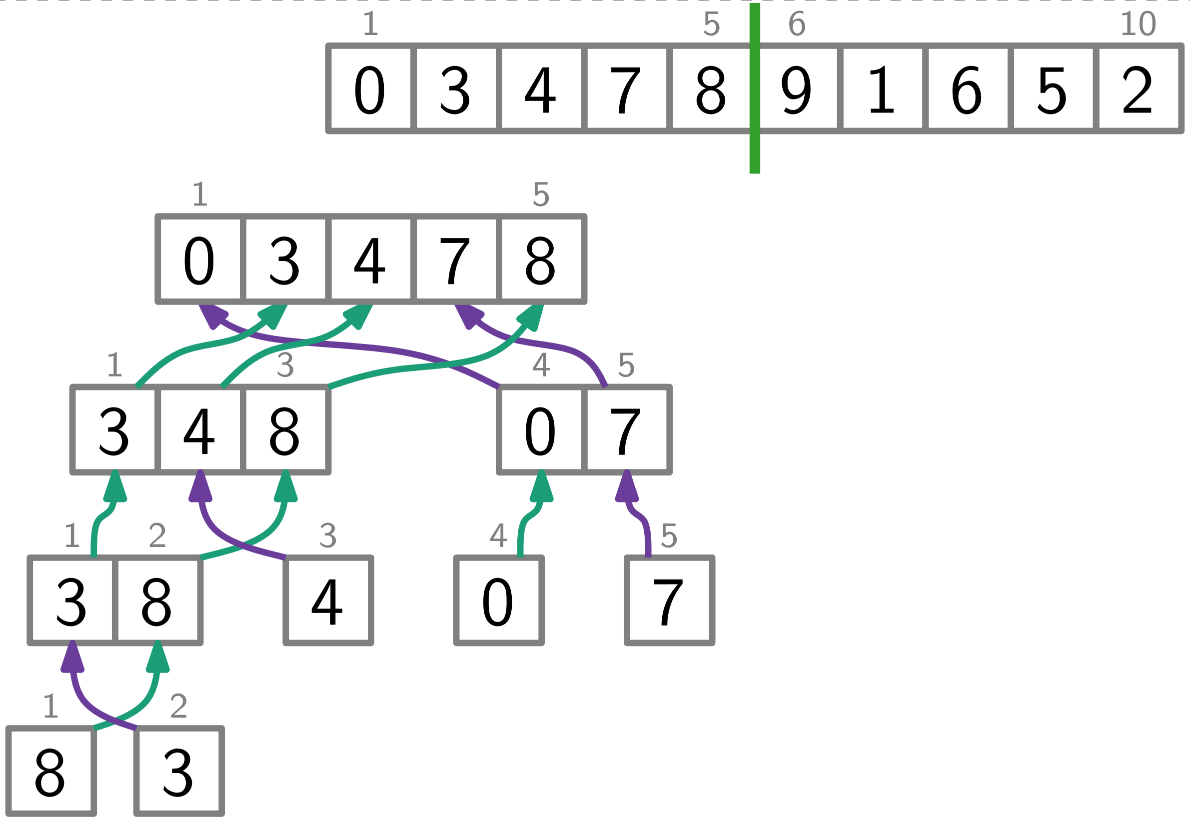


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

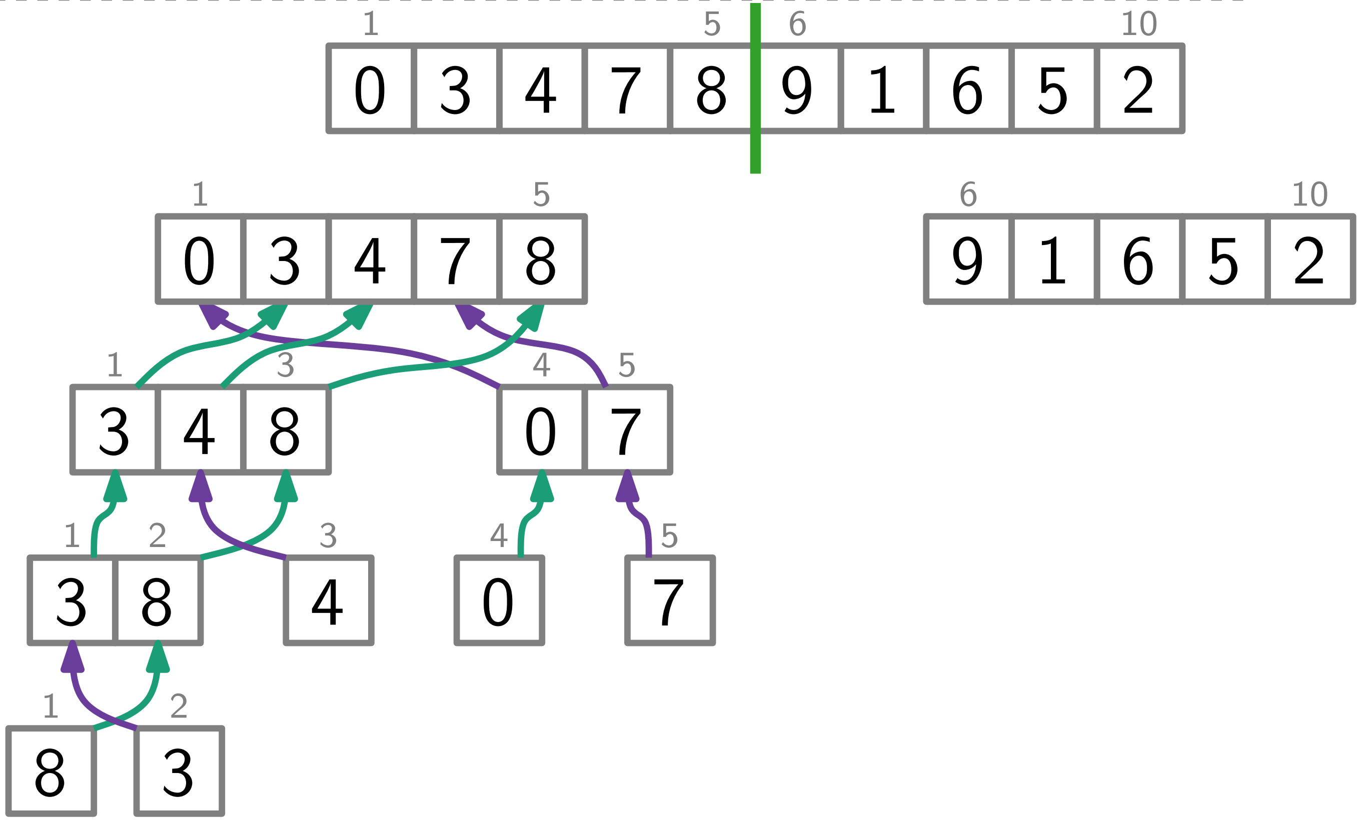
$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

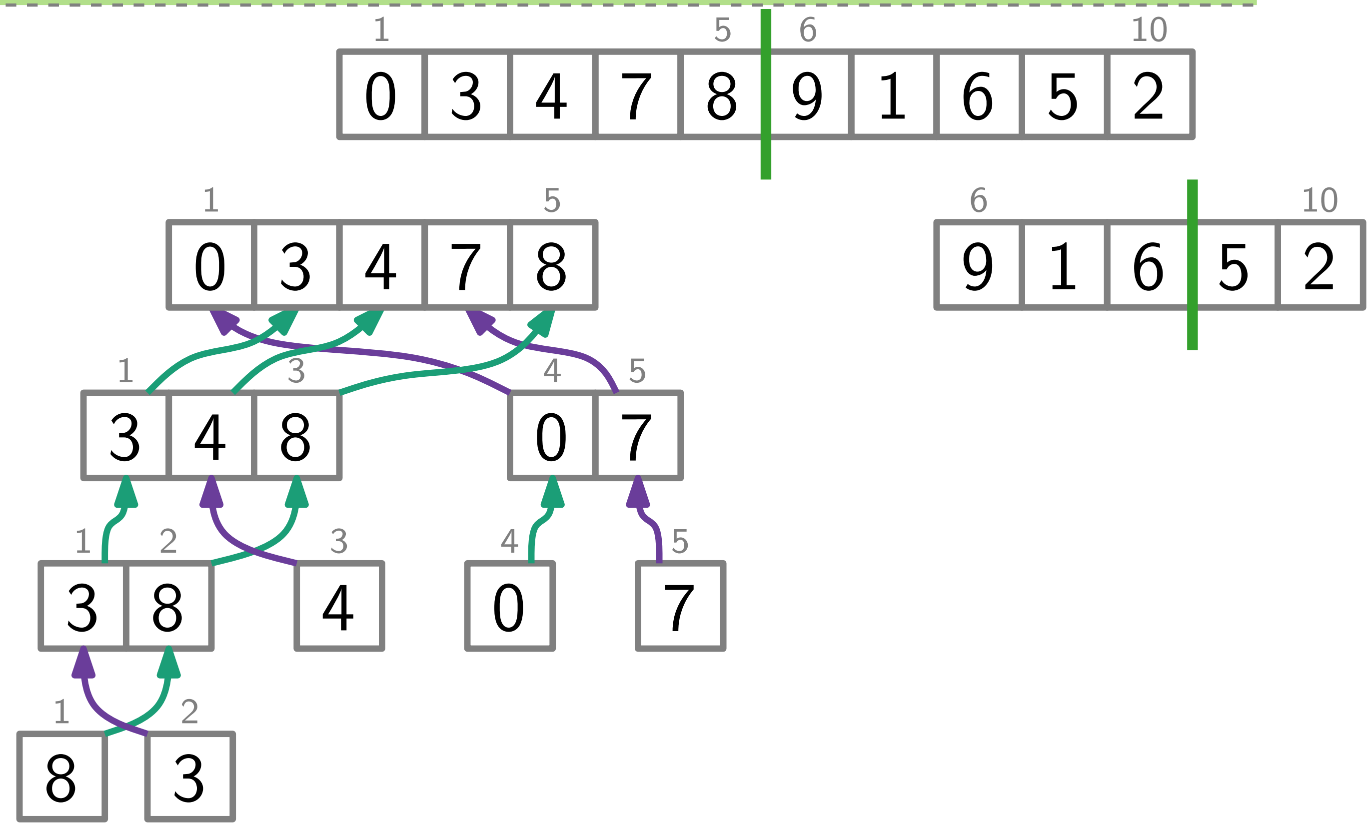
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

MergeSort(A, ℓ, m)

MergeSort($A, m+1, r$) } **herrsche**

Merge(A, ℓ, m, r) } **kombiniere**



MergeSort – ein Beispiel

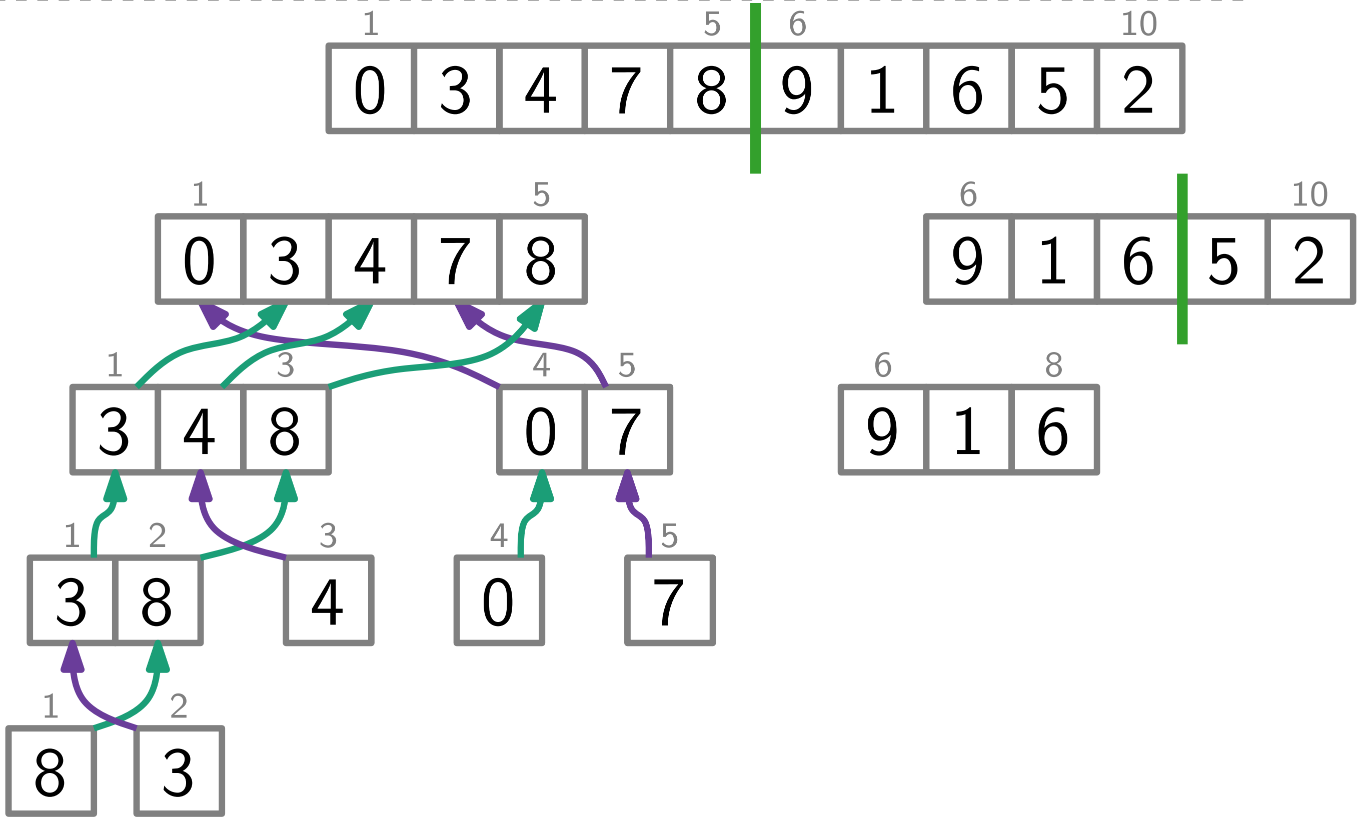
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

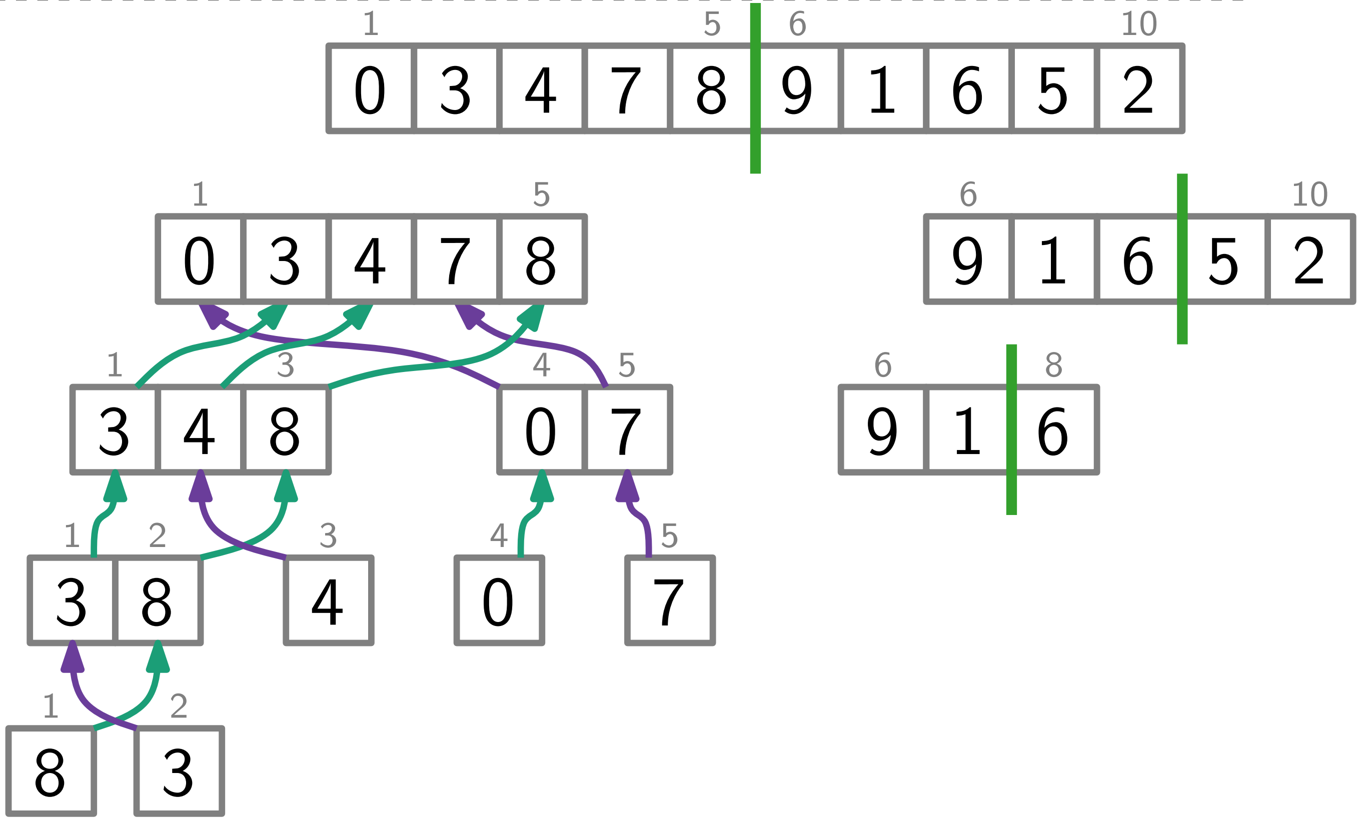


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

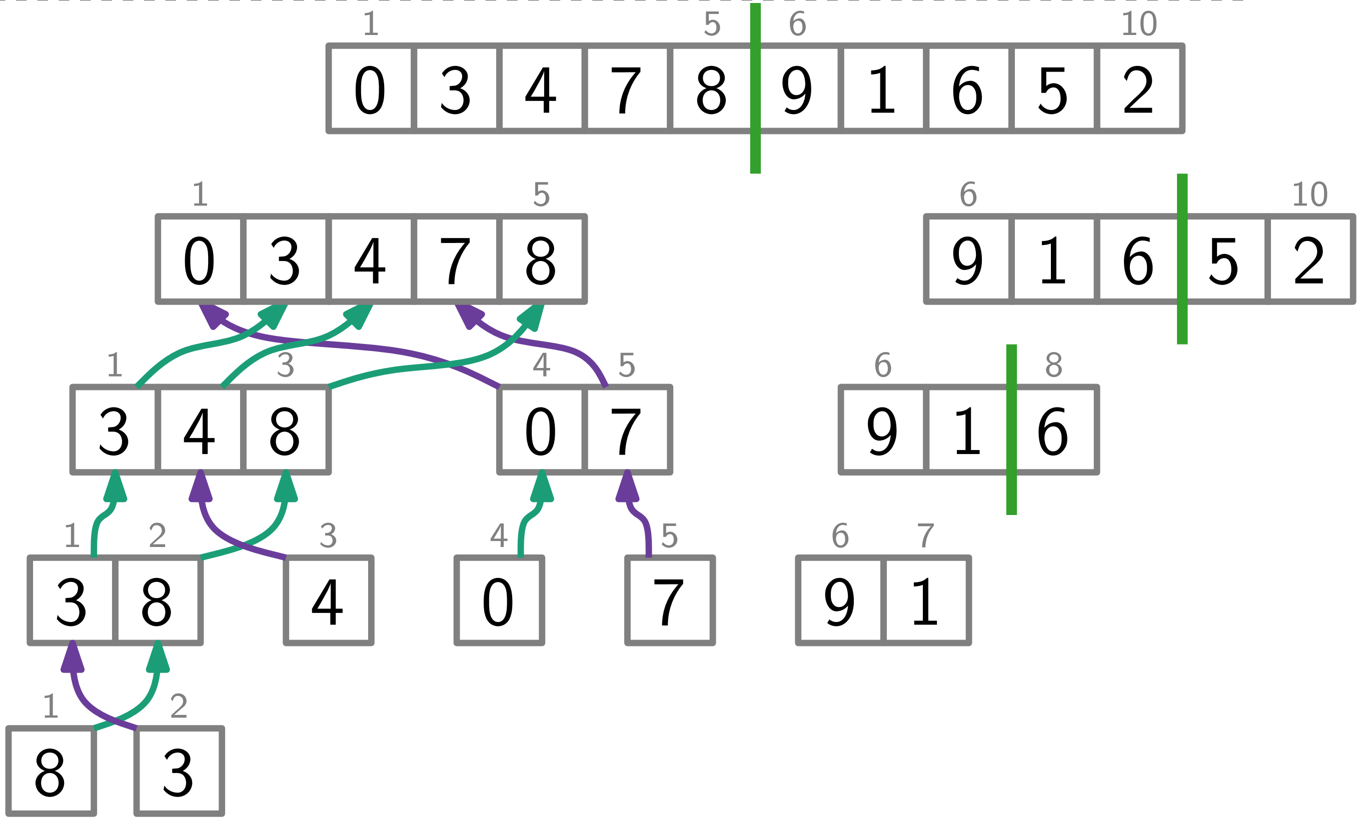
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

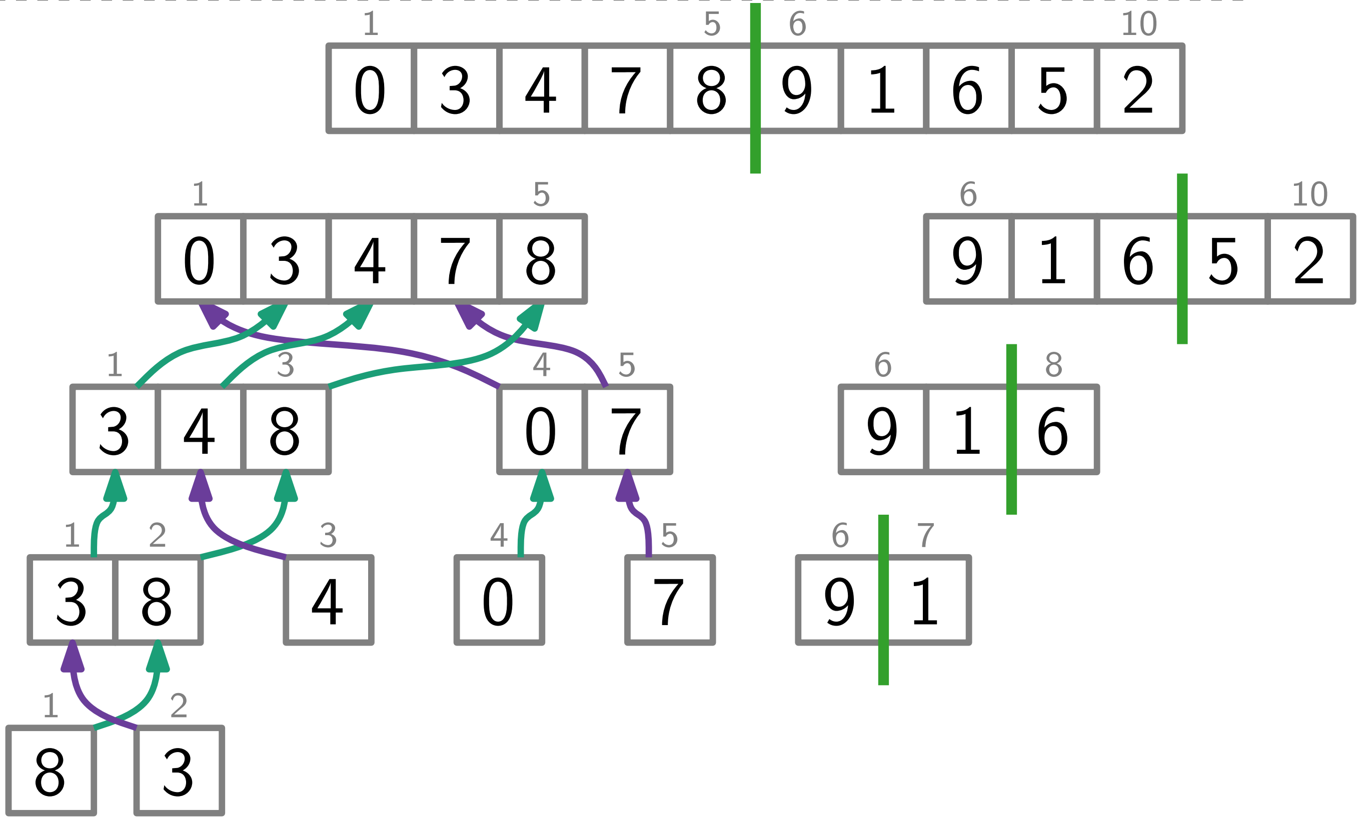


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

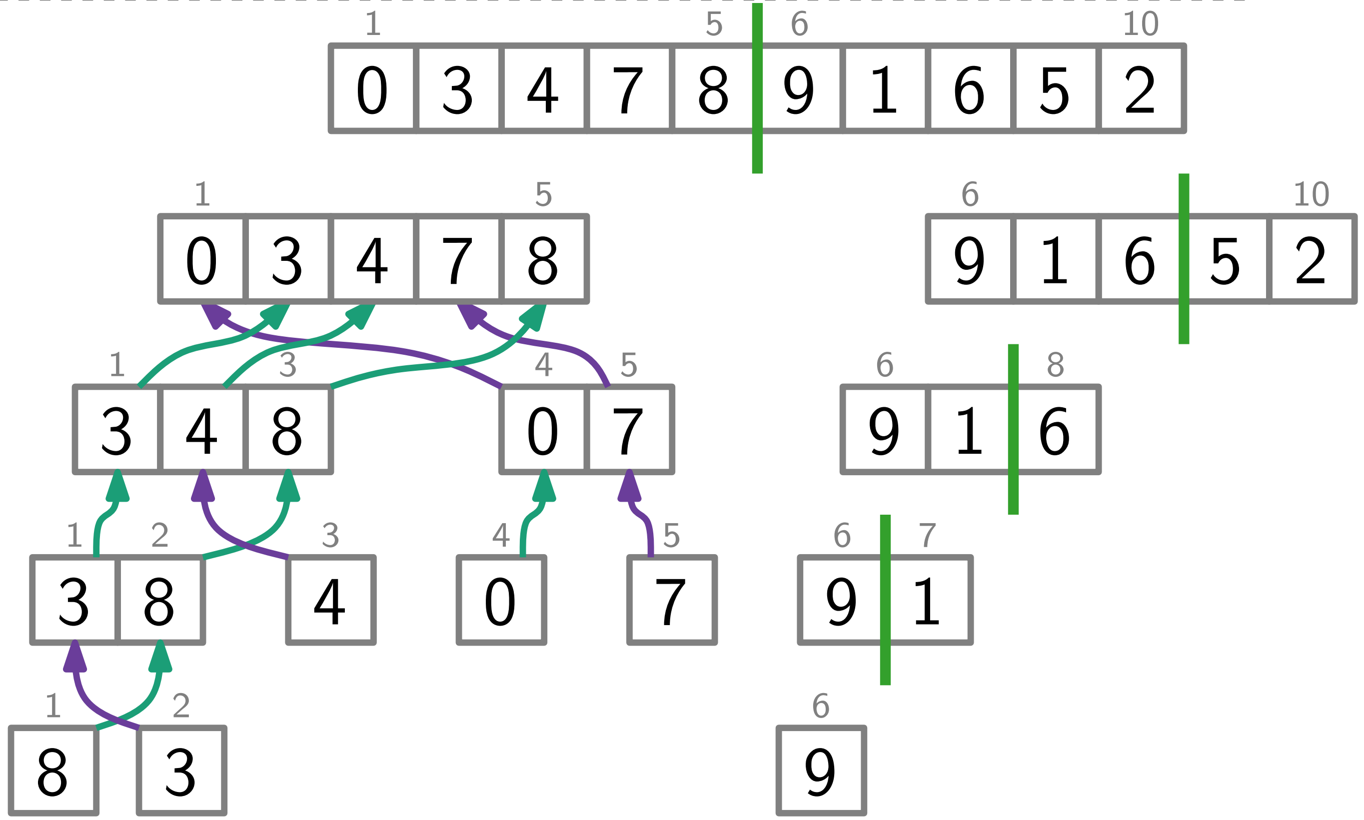


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$ and $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

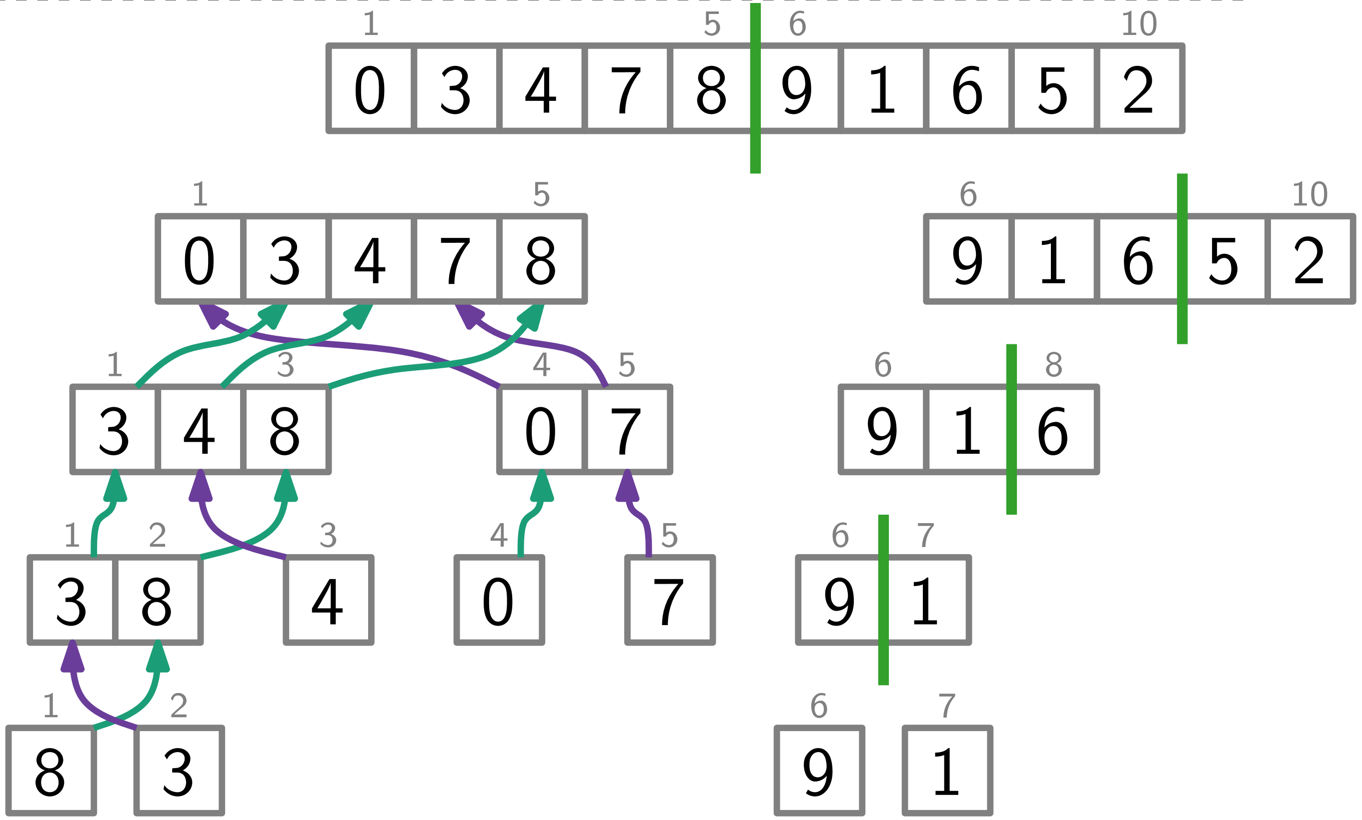
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$$\left.\begin{array}{l} m = \lfloor(\ell + r)/2\rfloor \end{array}\right\}\ \textbf{teile}$$

$$\left.\begin{array}{l} \textsc{MergeSort}(A, \ell, m) \\ \textsc{MergeSort}(A, m+1, r) \end{array}\right\}\ \textbf{herrsche}$$

$$\left.\begin{array}{l} \textsc{Merge}(A, \ell, m, r) \end{array}\right\}\ \textbf{kombiniere}$$



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

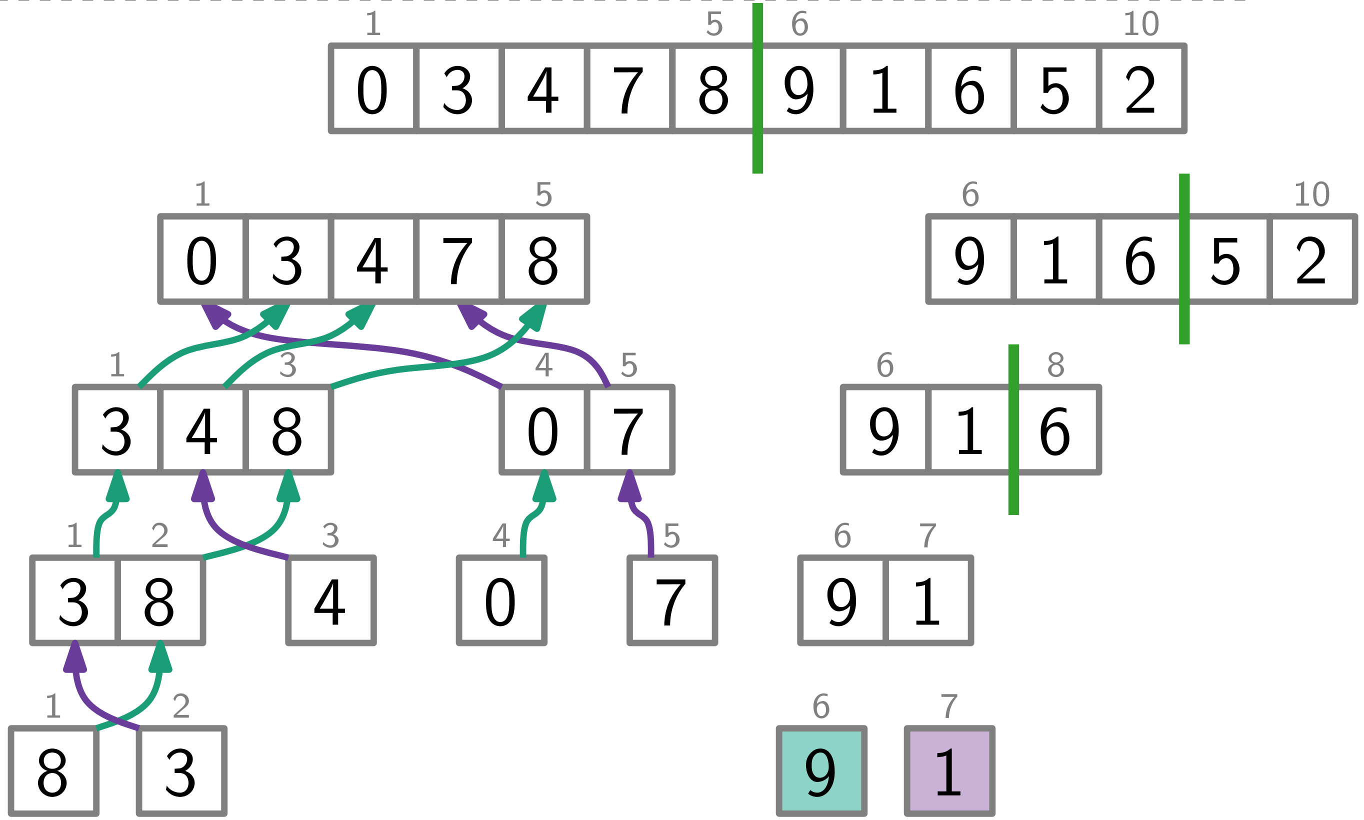
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

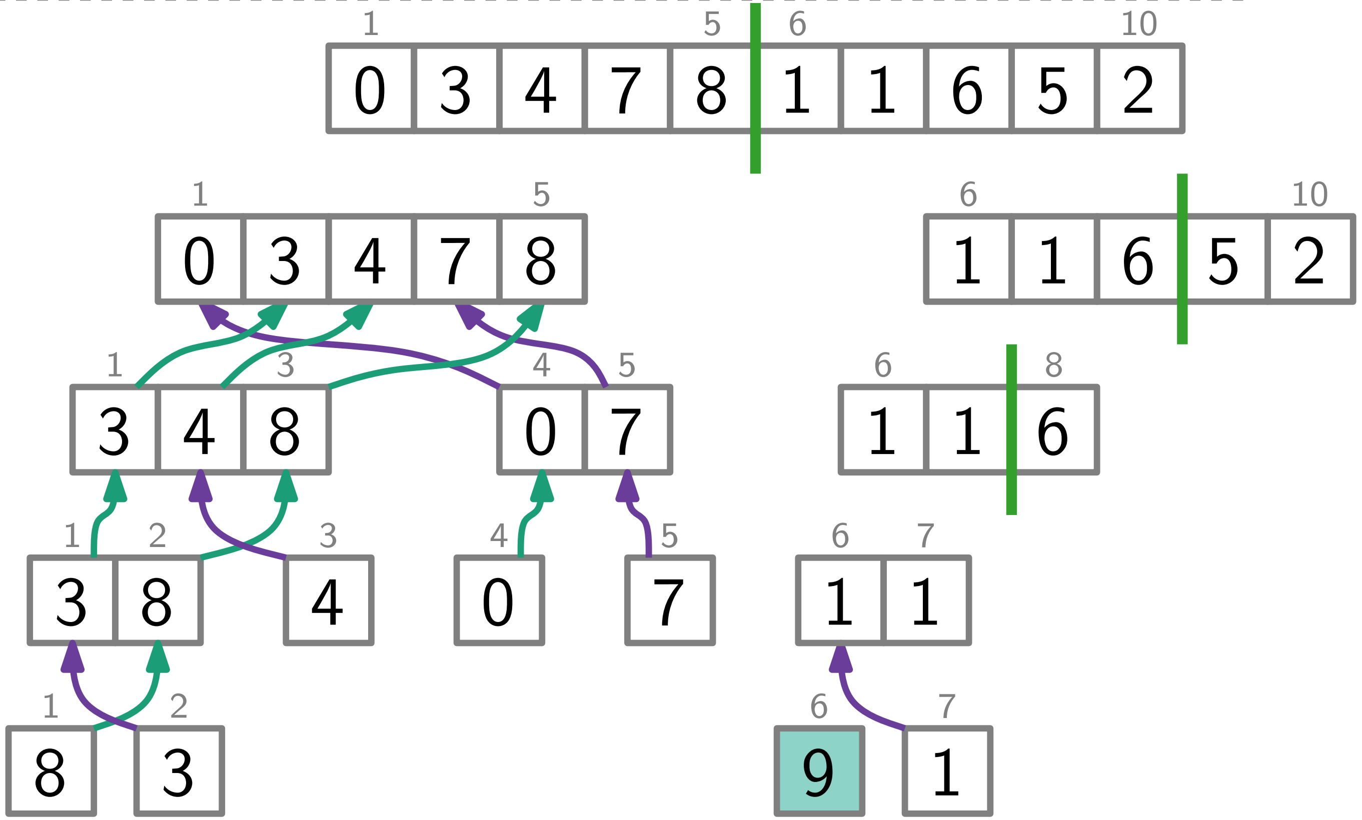


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m) } **herrsche**
- MergeSort($A, m+1, r$)
- Merge(A, ℓ, m, r) } **kombiniere**

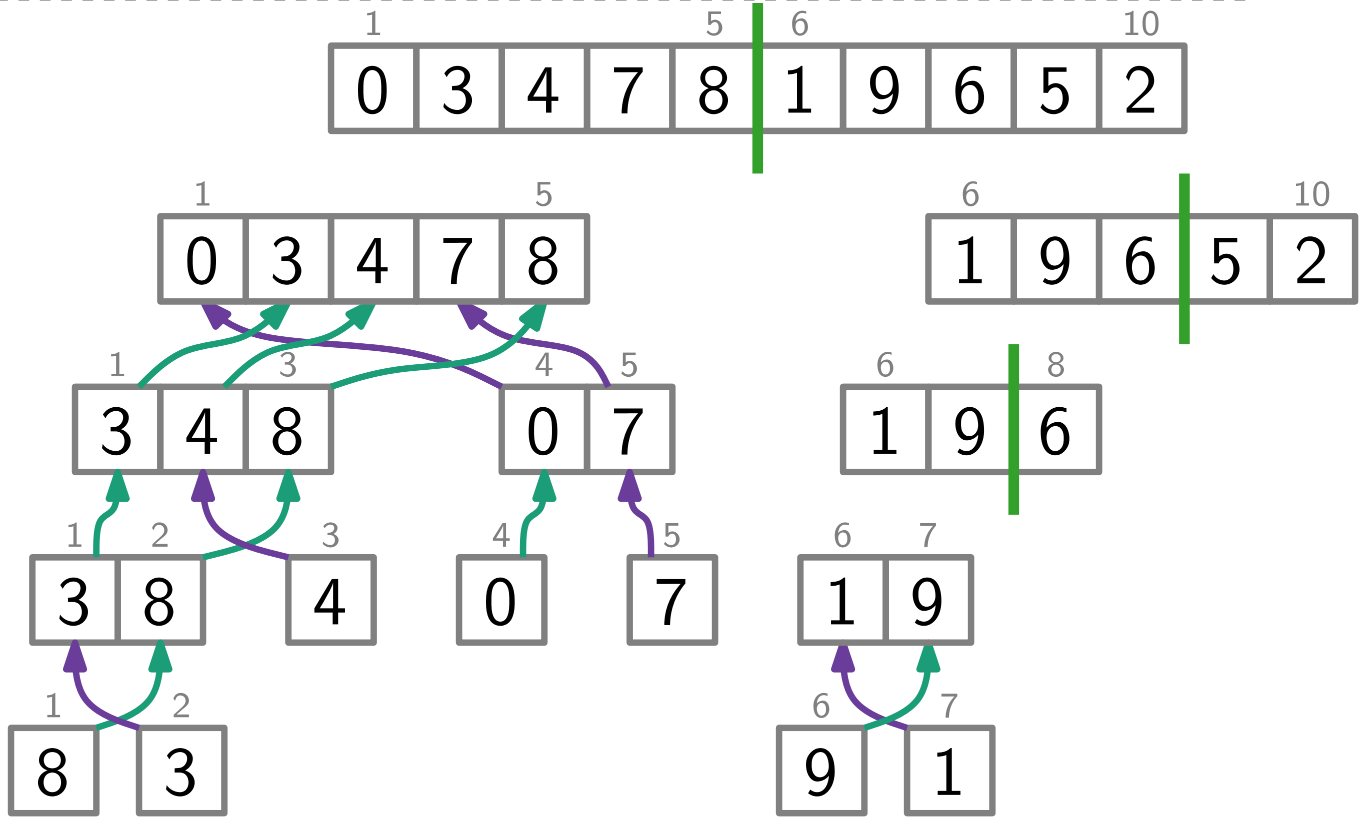


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- MergeSort(A, ℓ, m) }
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

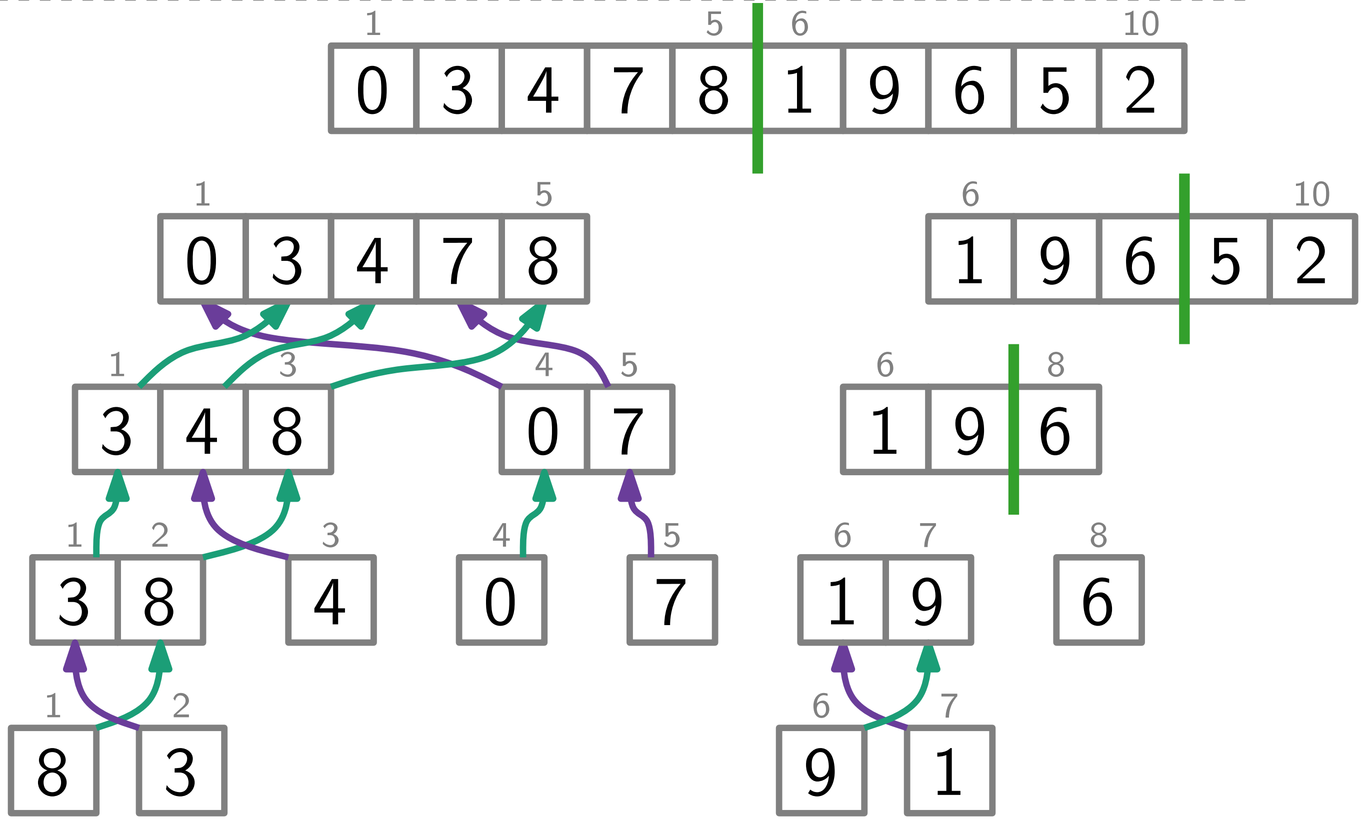


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$ } **herrsche**
- $\textsc{MergeSort}(A, m + 1, r)$
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

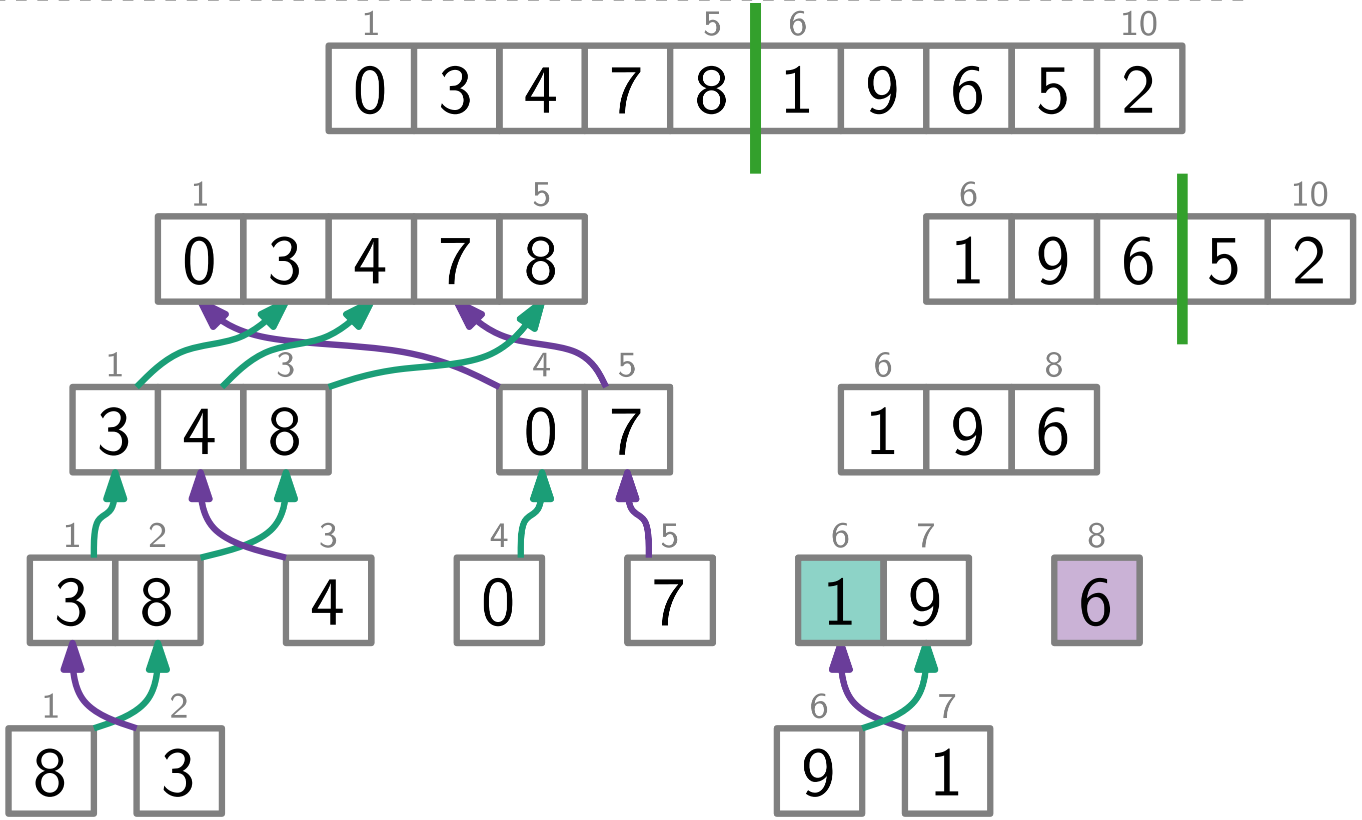
$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

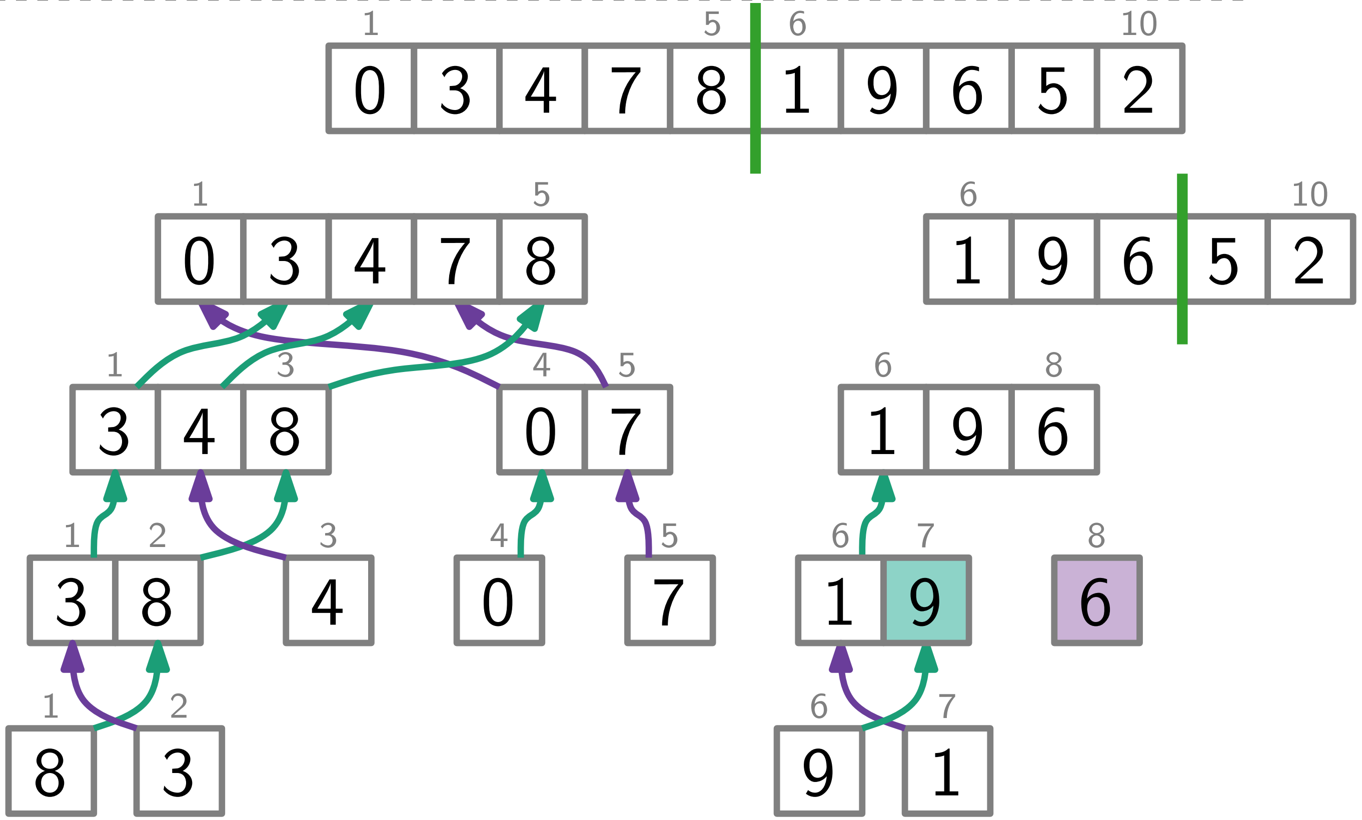
$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

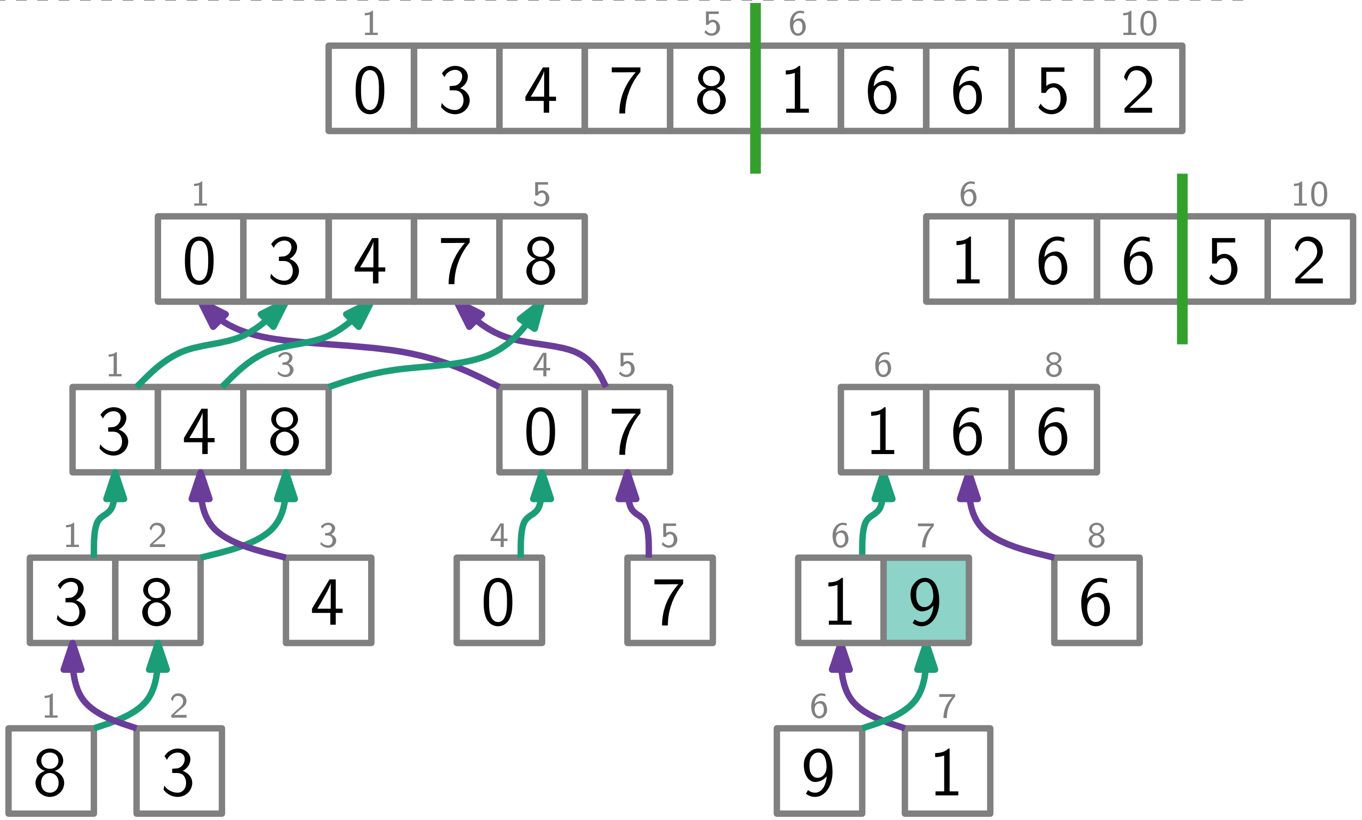
if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

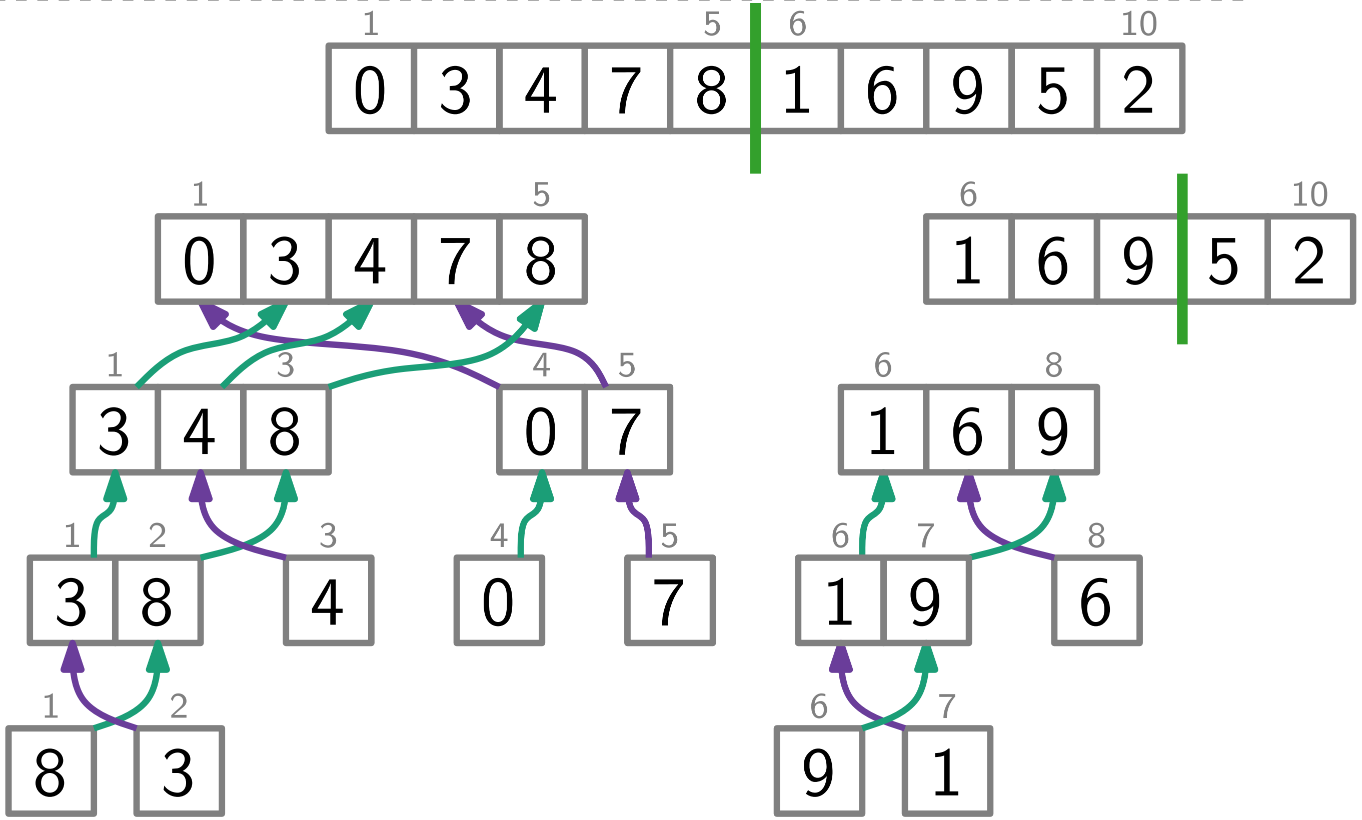
$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

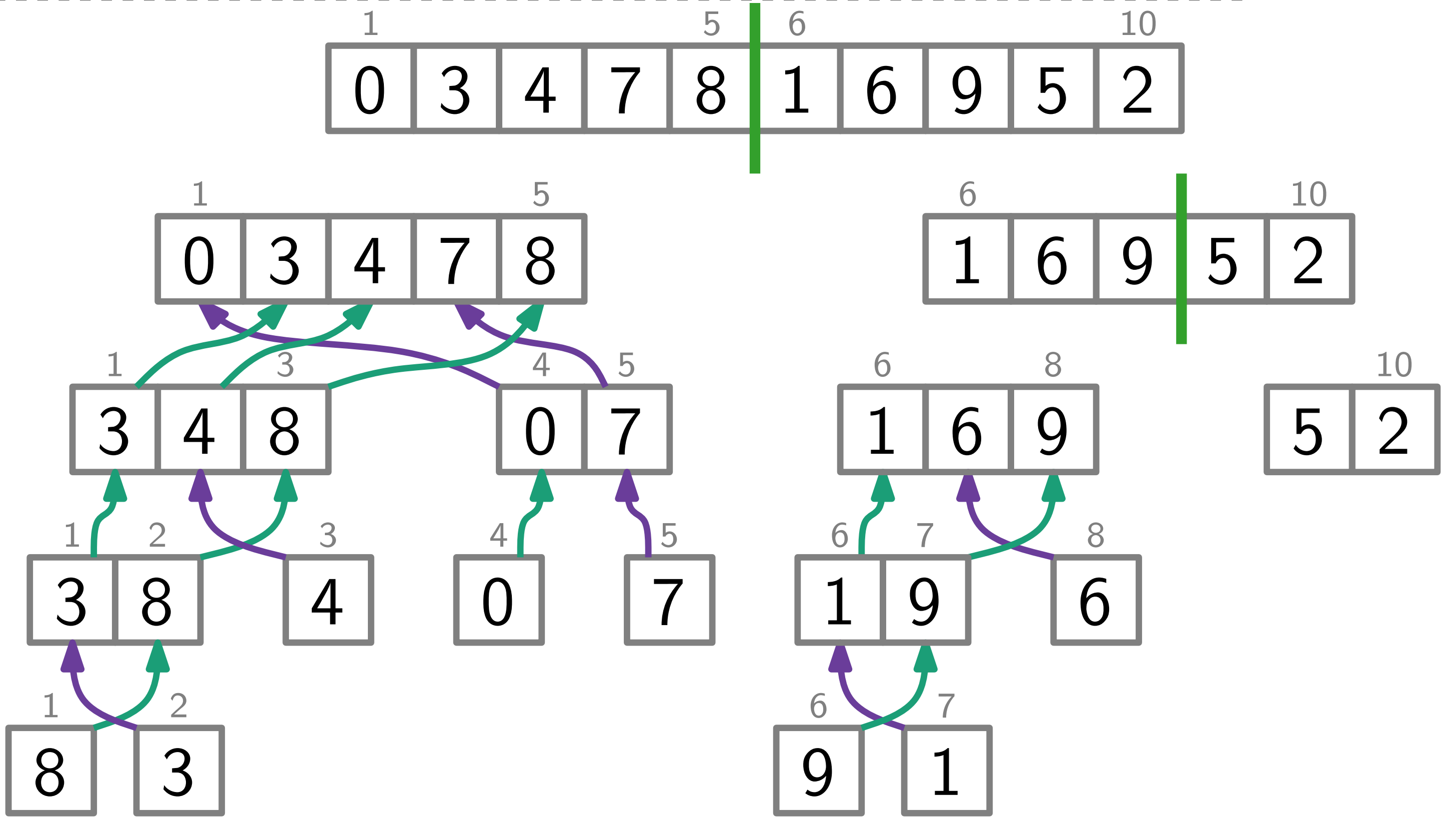
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

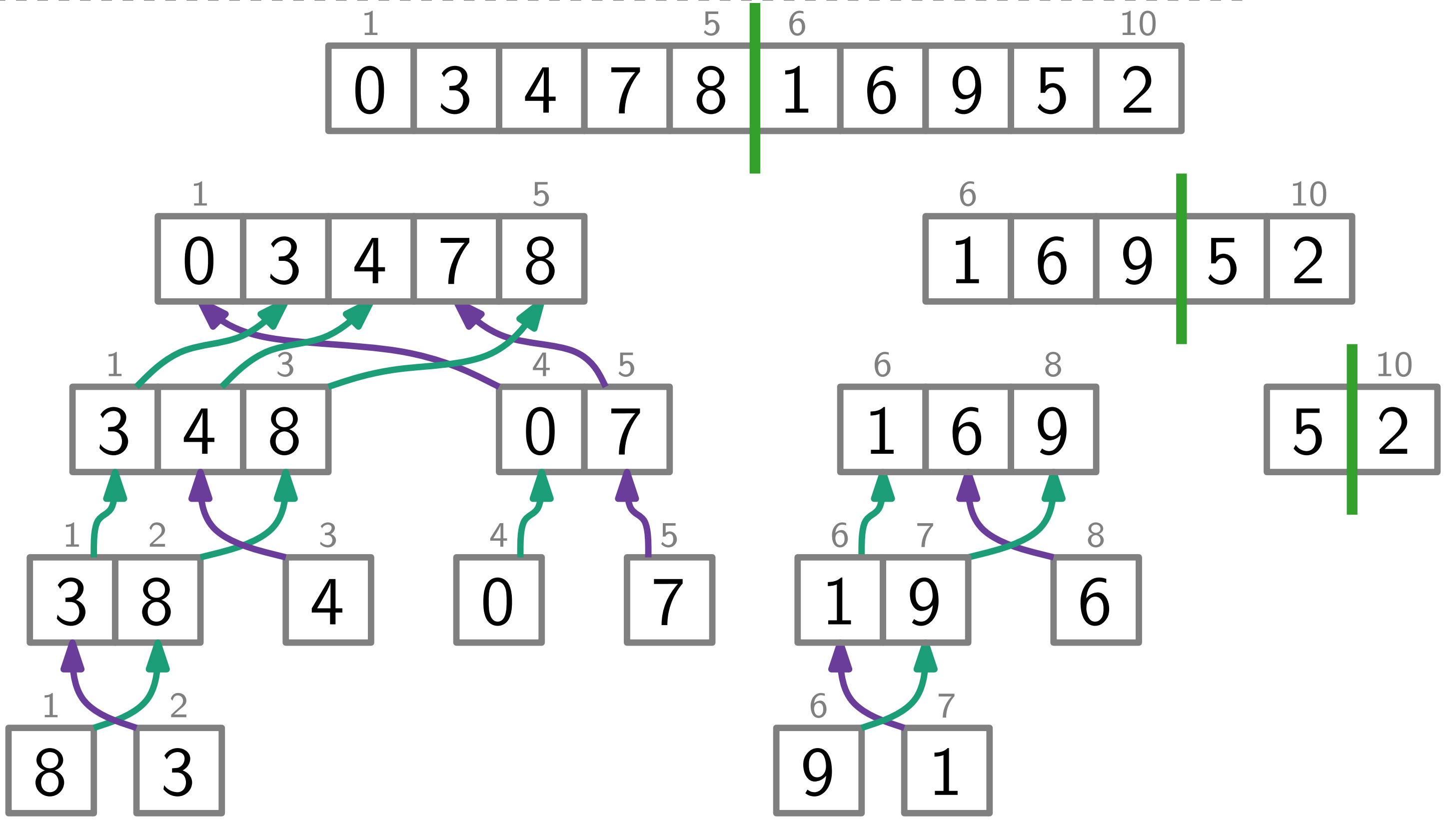
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

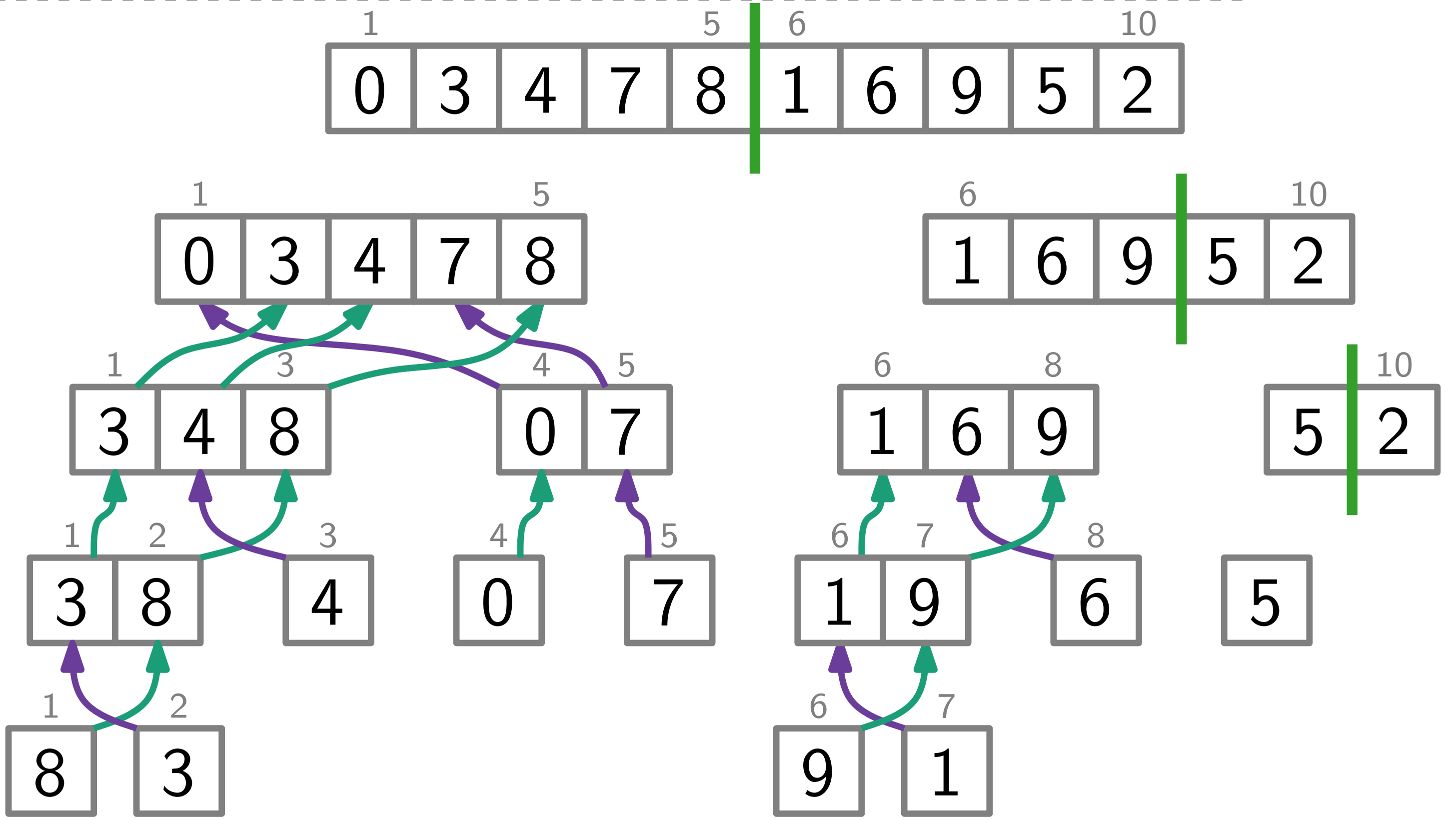


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

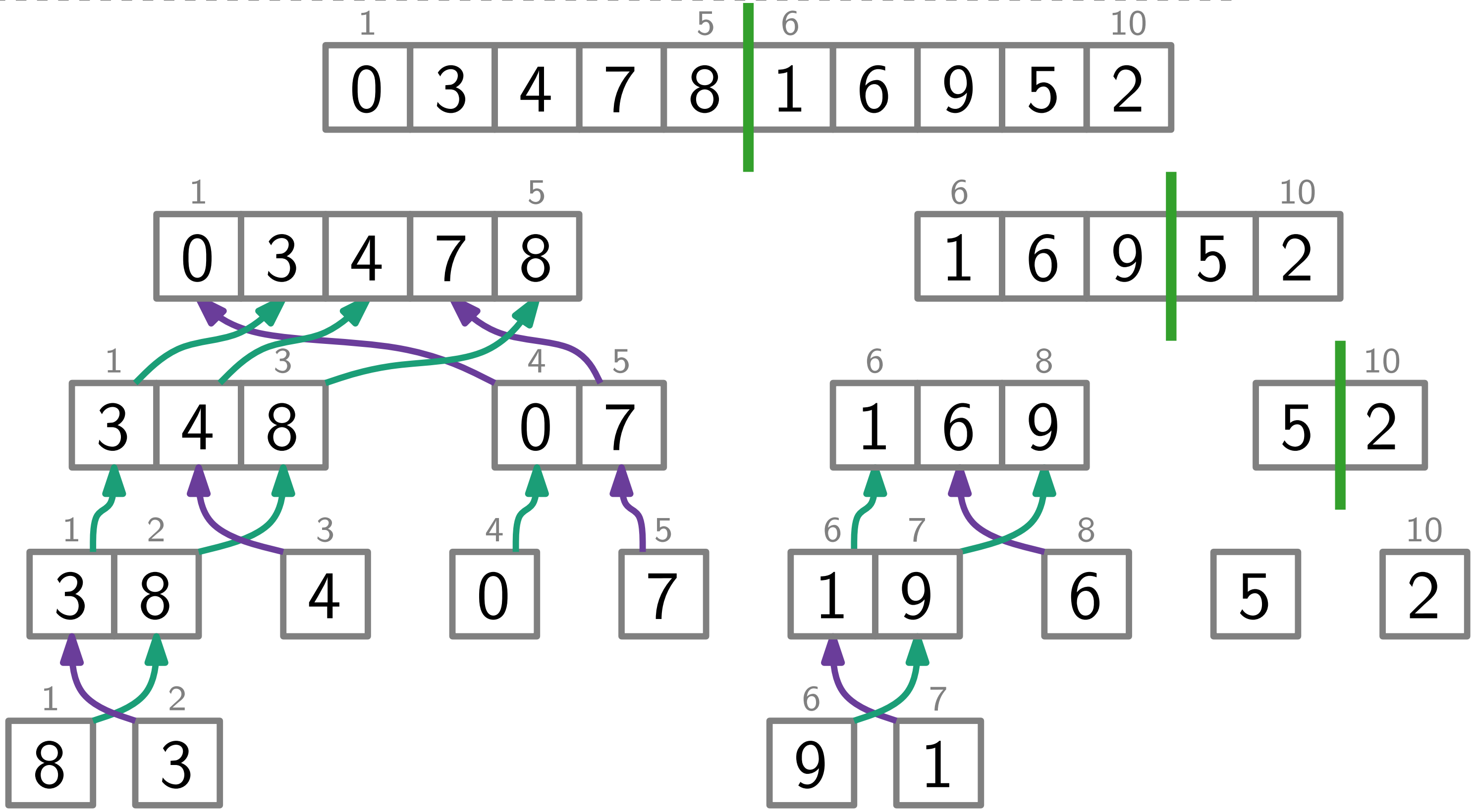
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

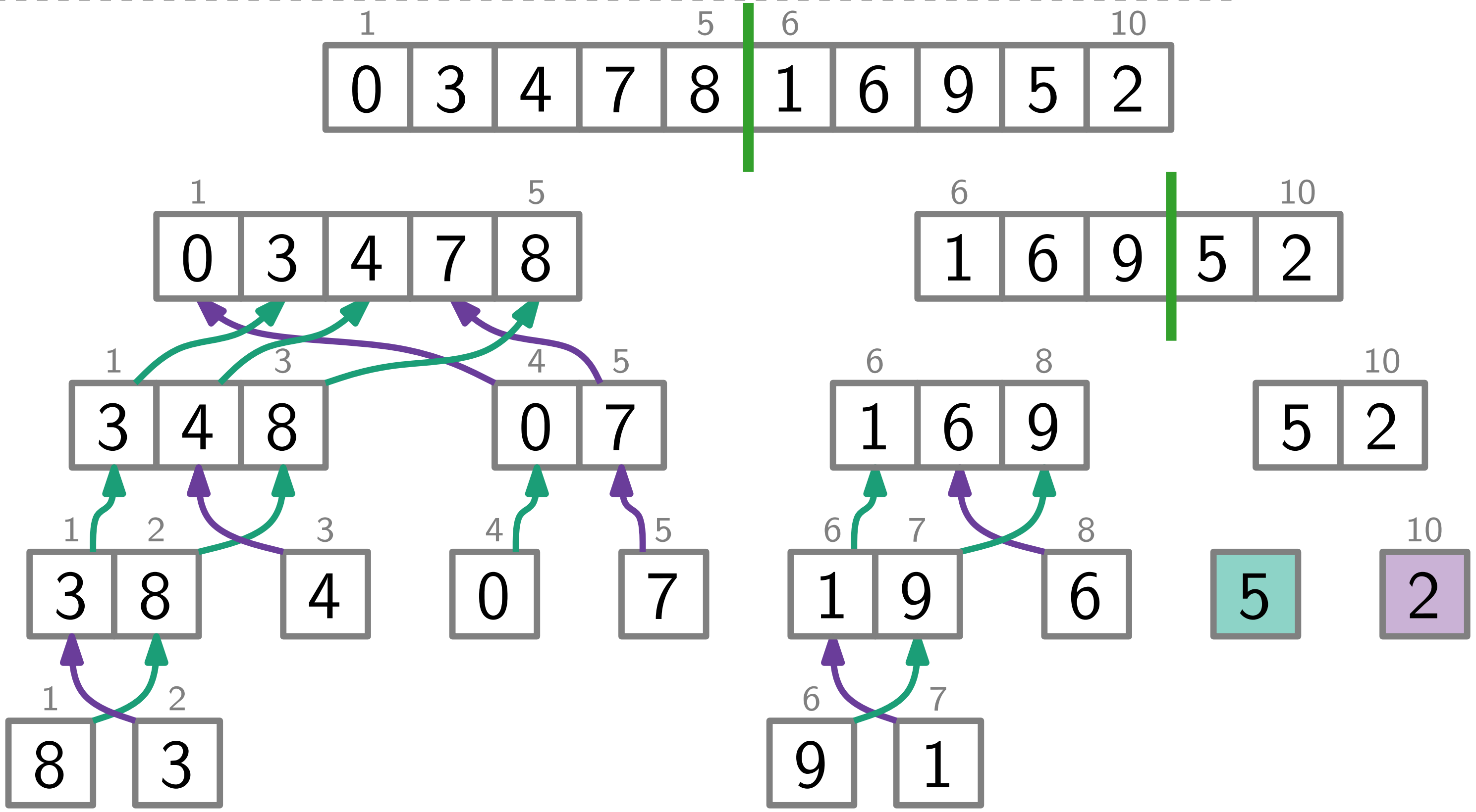
$\textsc{MergeSort}(\mathsf{int}[]\ A,\ \mathsf{int}\ \ell = 1,\ \mathsf{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

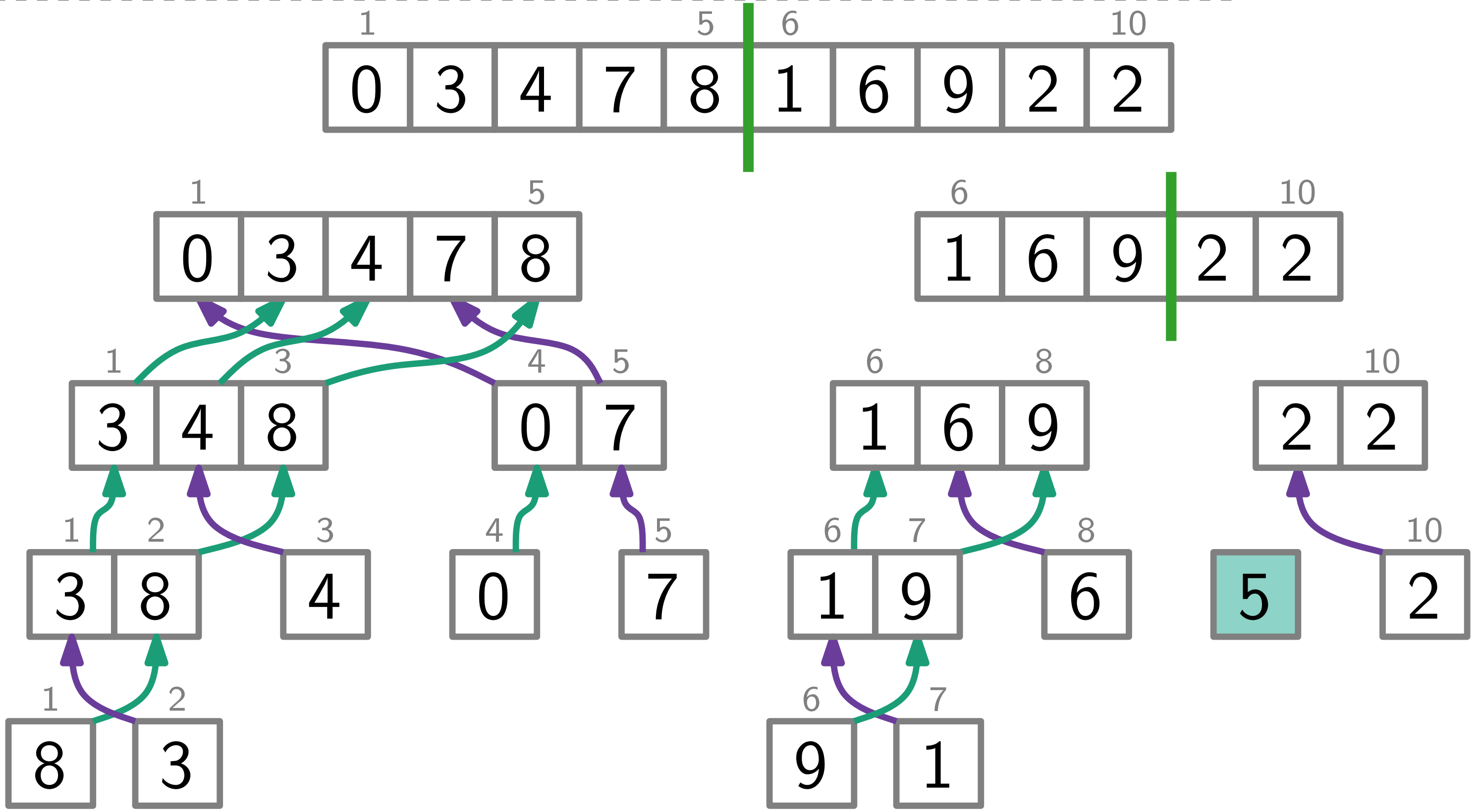
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

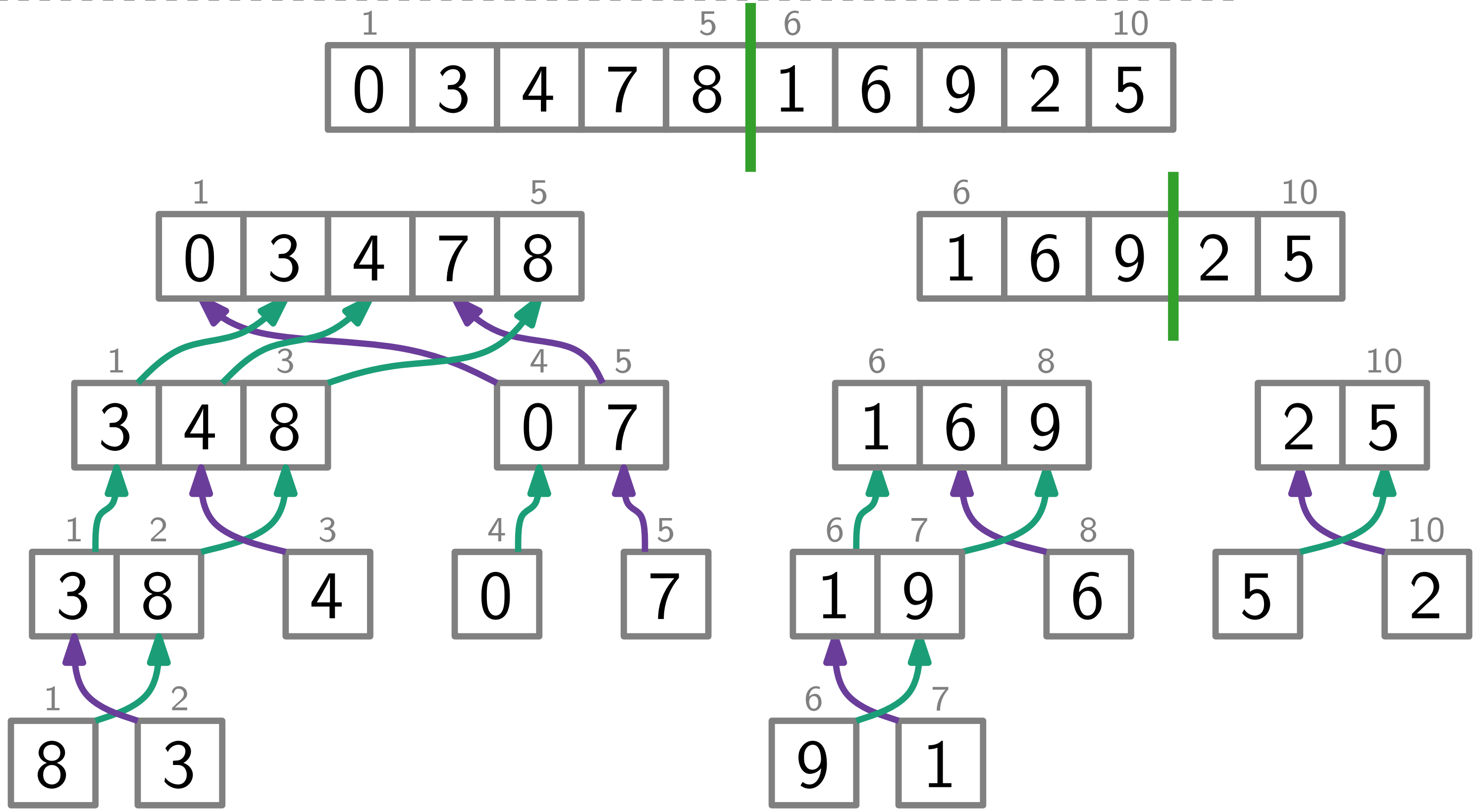
if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

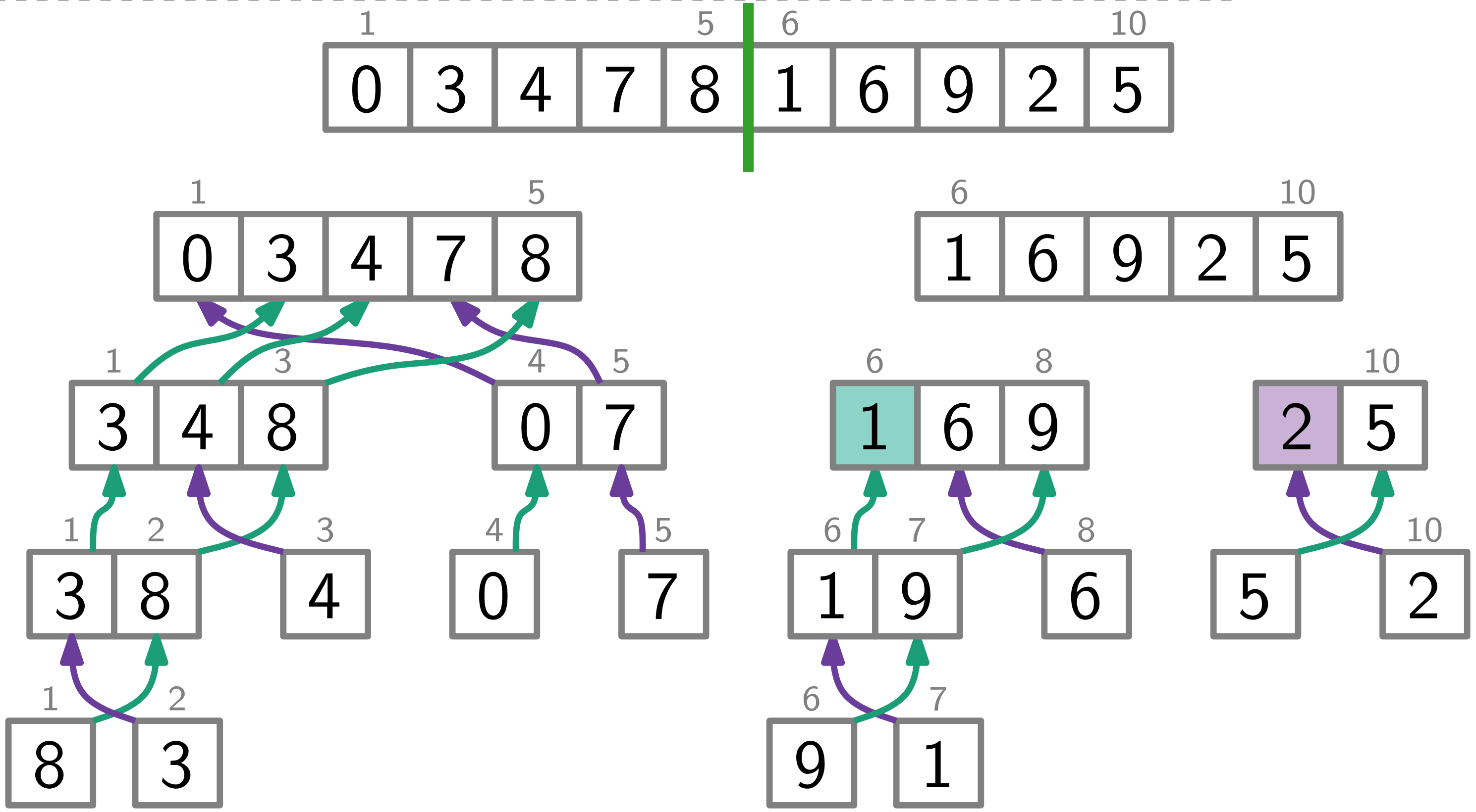


MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

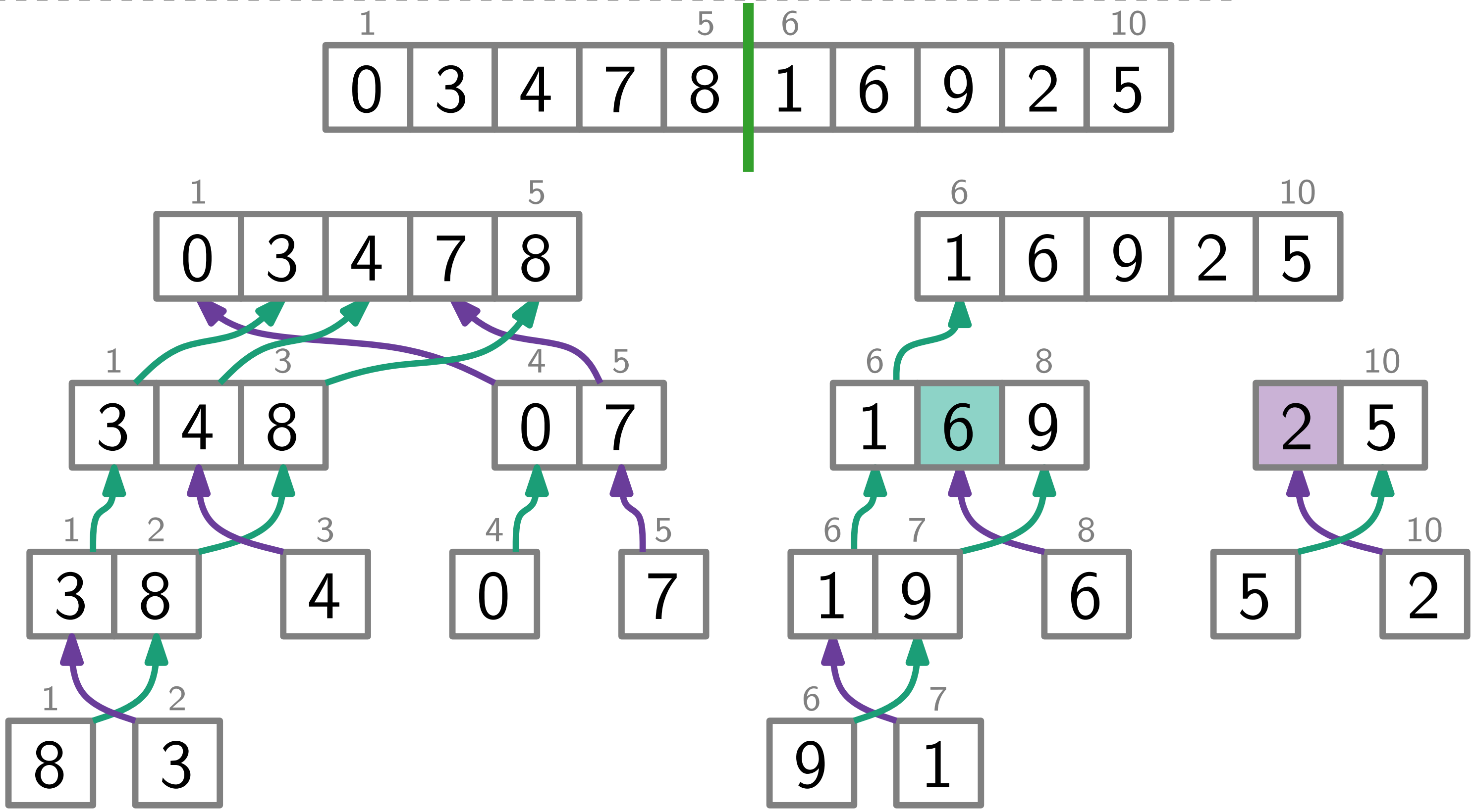


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

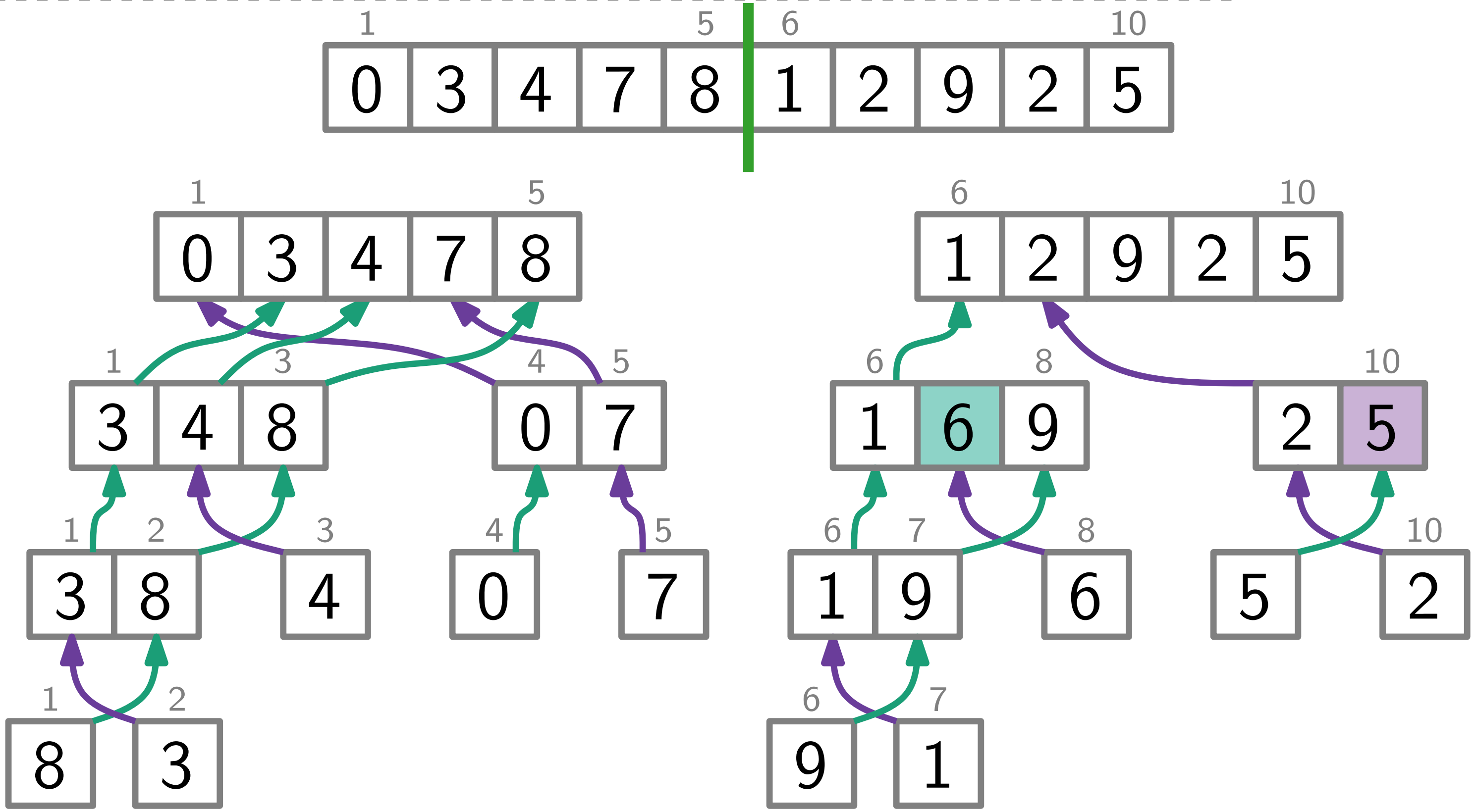
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

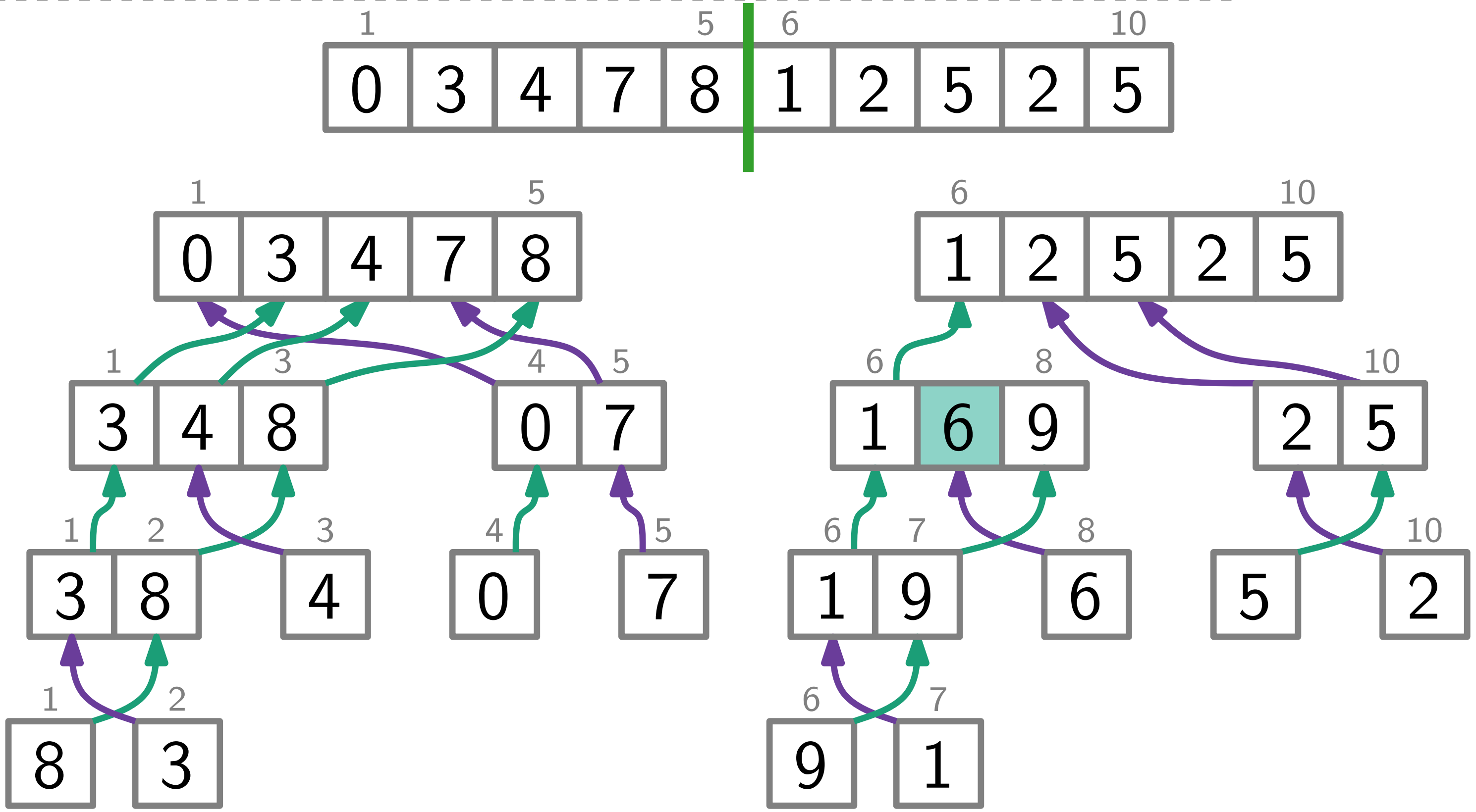
if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

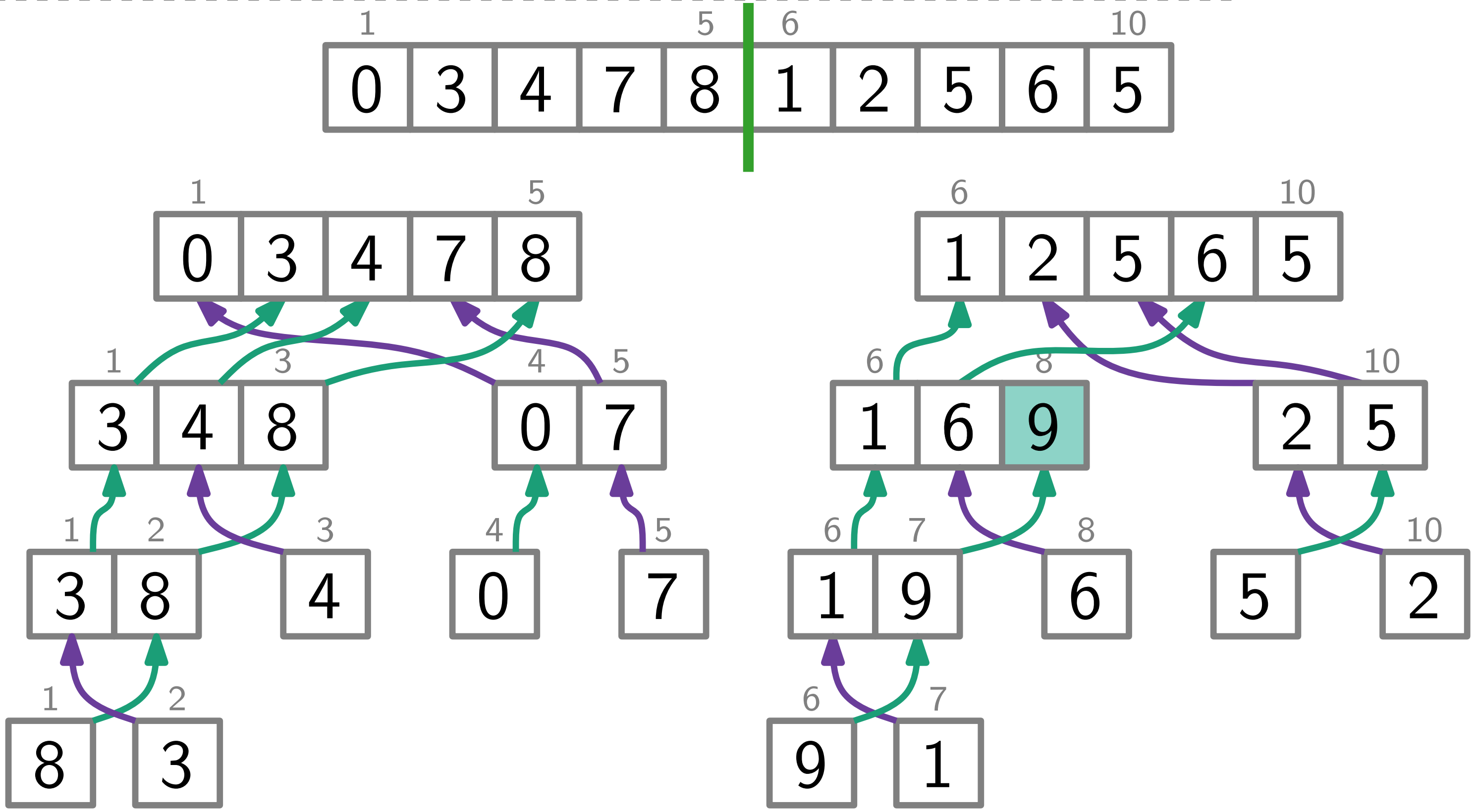
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

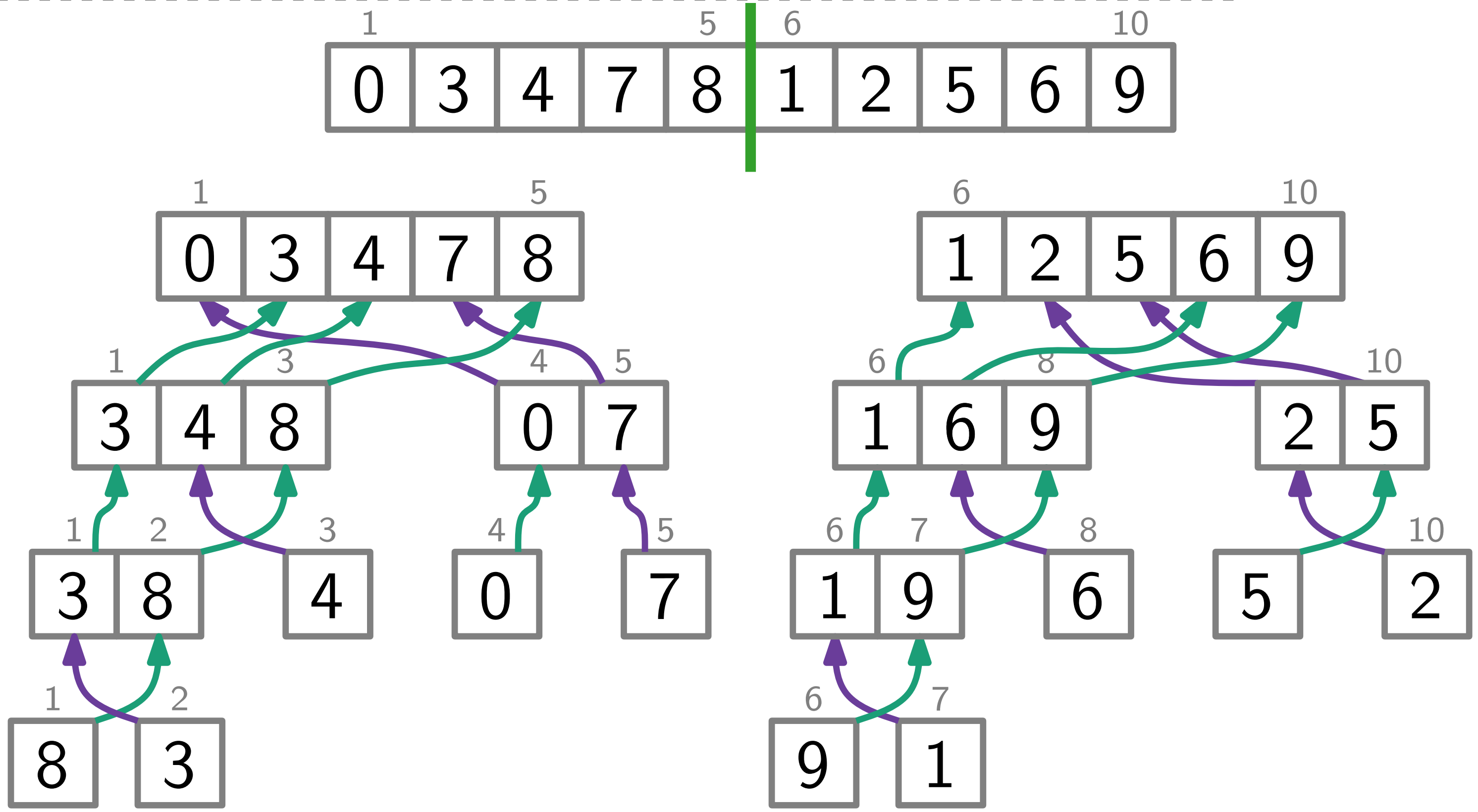


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$ } **herrsche**
- $\textsc{MergeSort}(A, m + 1, r)$
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

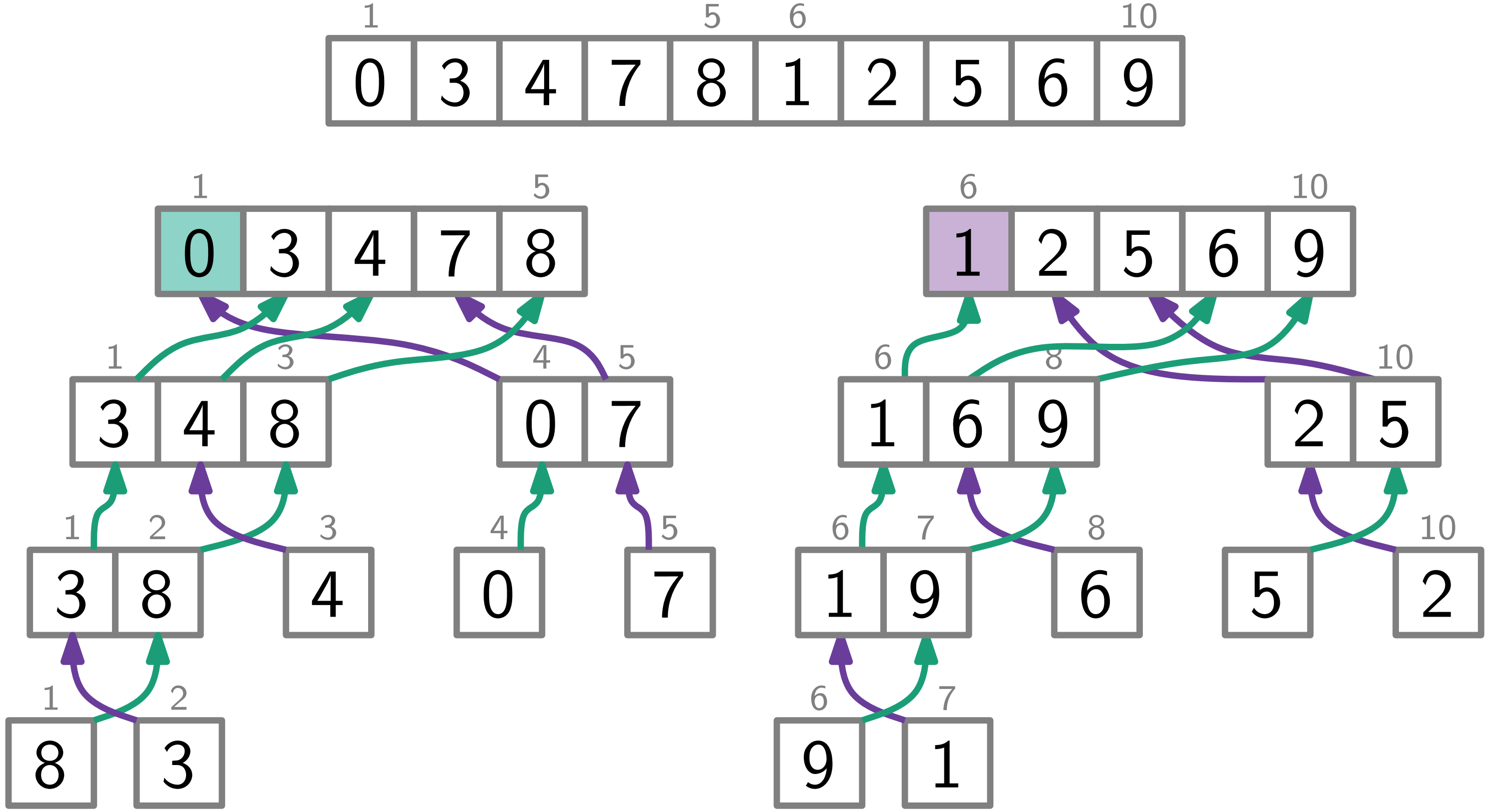
if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

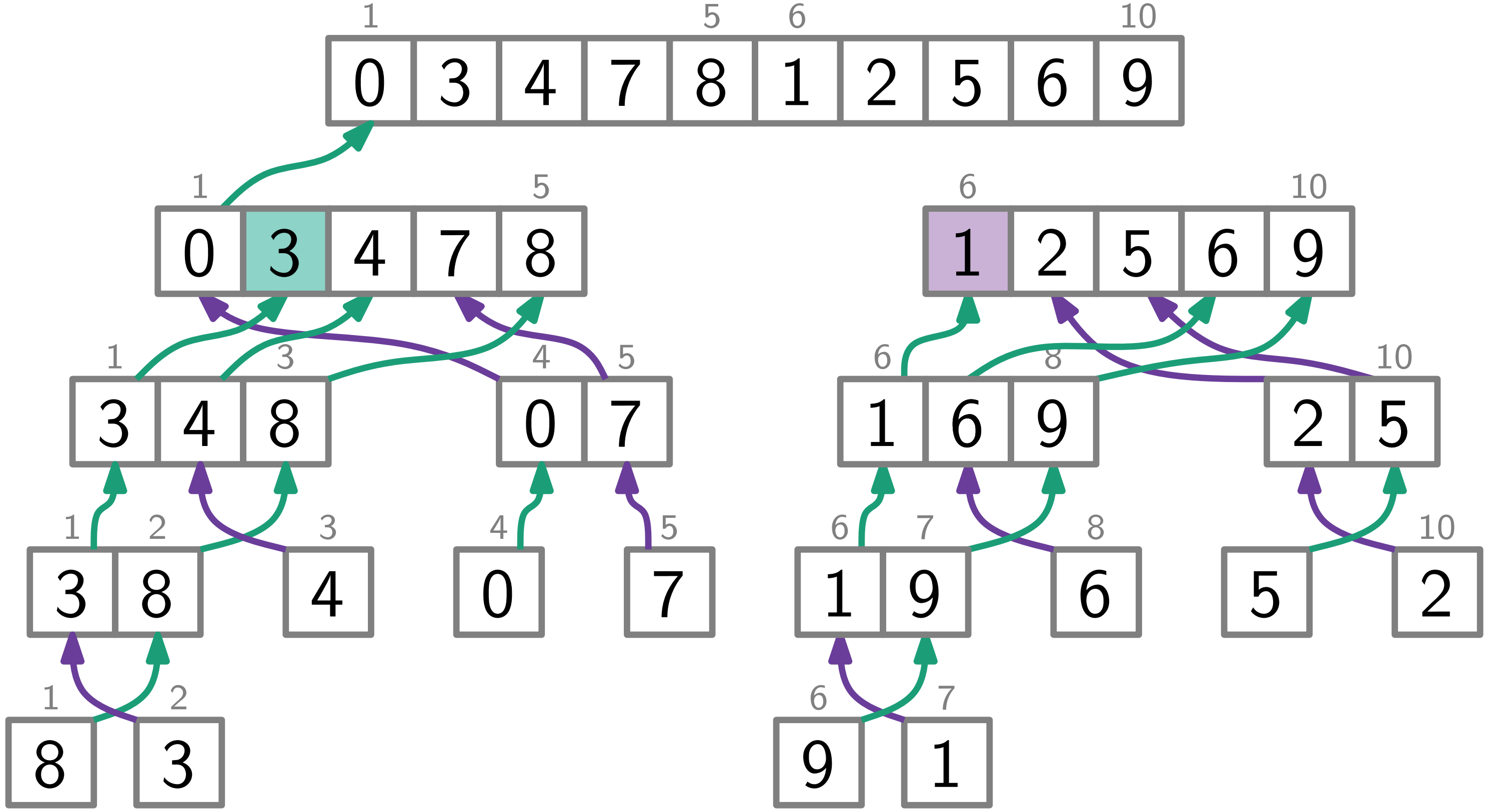
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

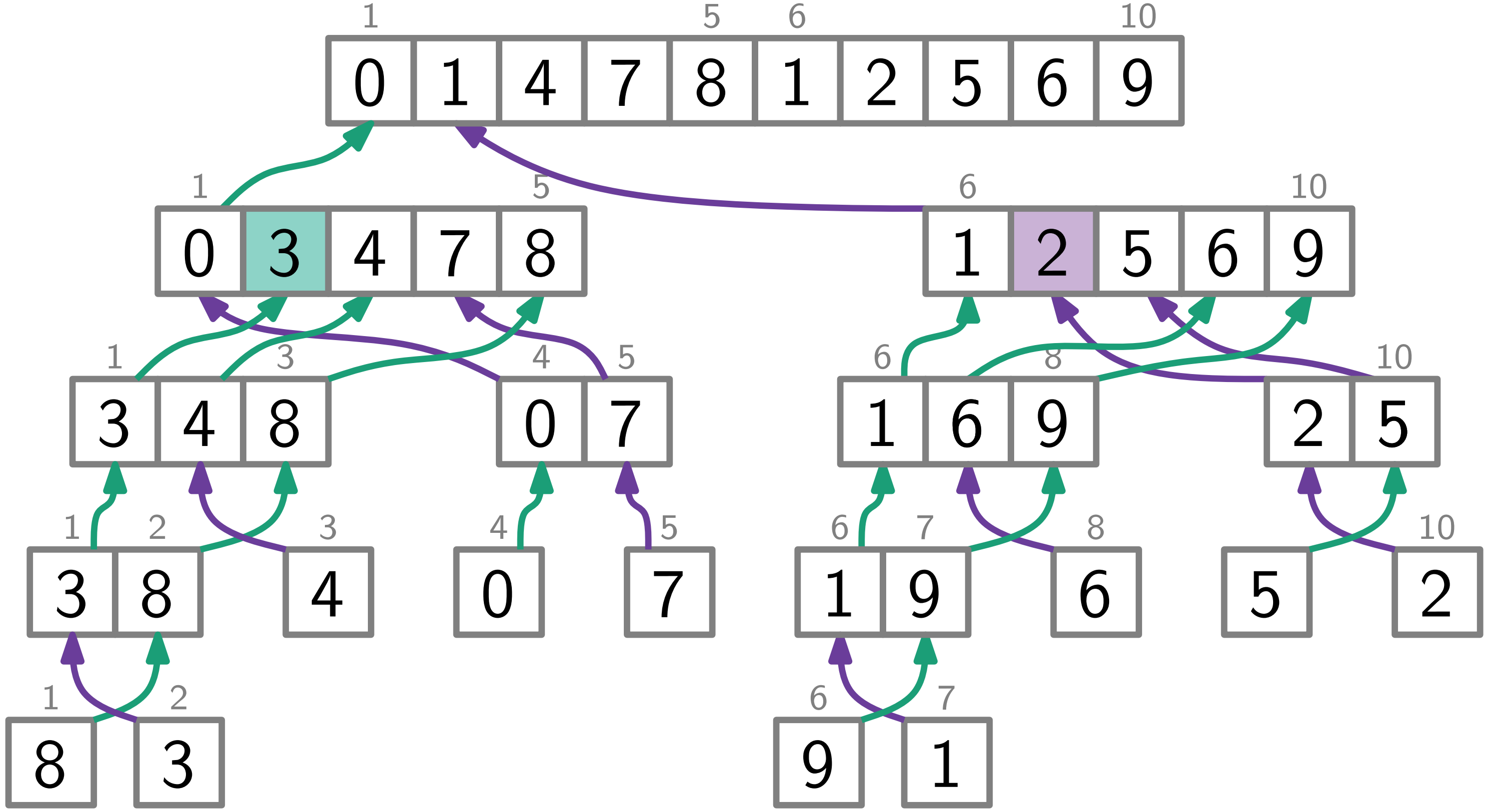
if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

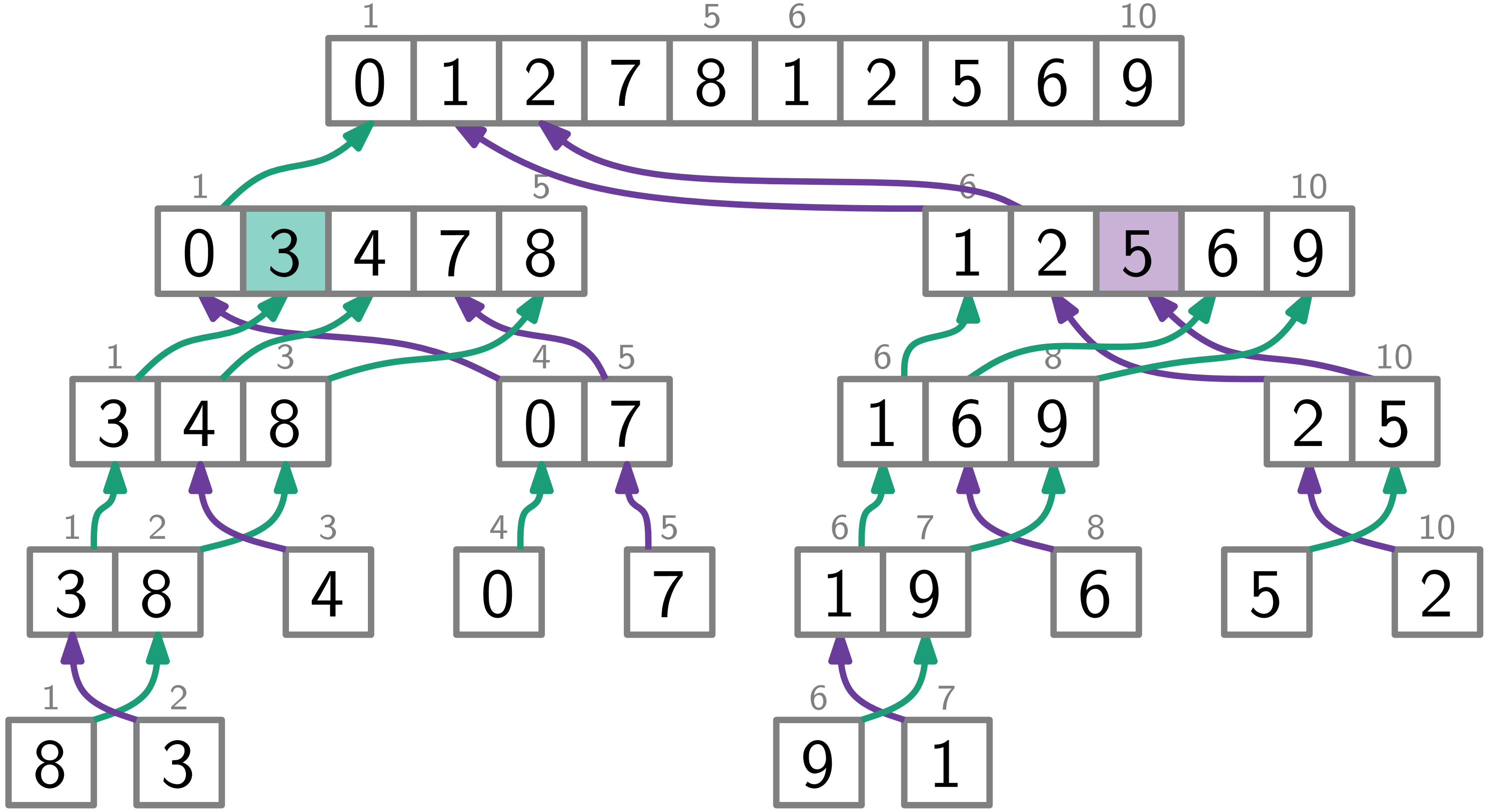
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m)
  MergeSort($A, m+1, r$) } **herrsche**
  Merge(A, ℓ, m, r) } **kombiniere**

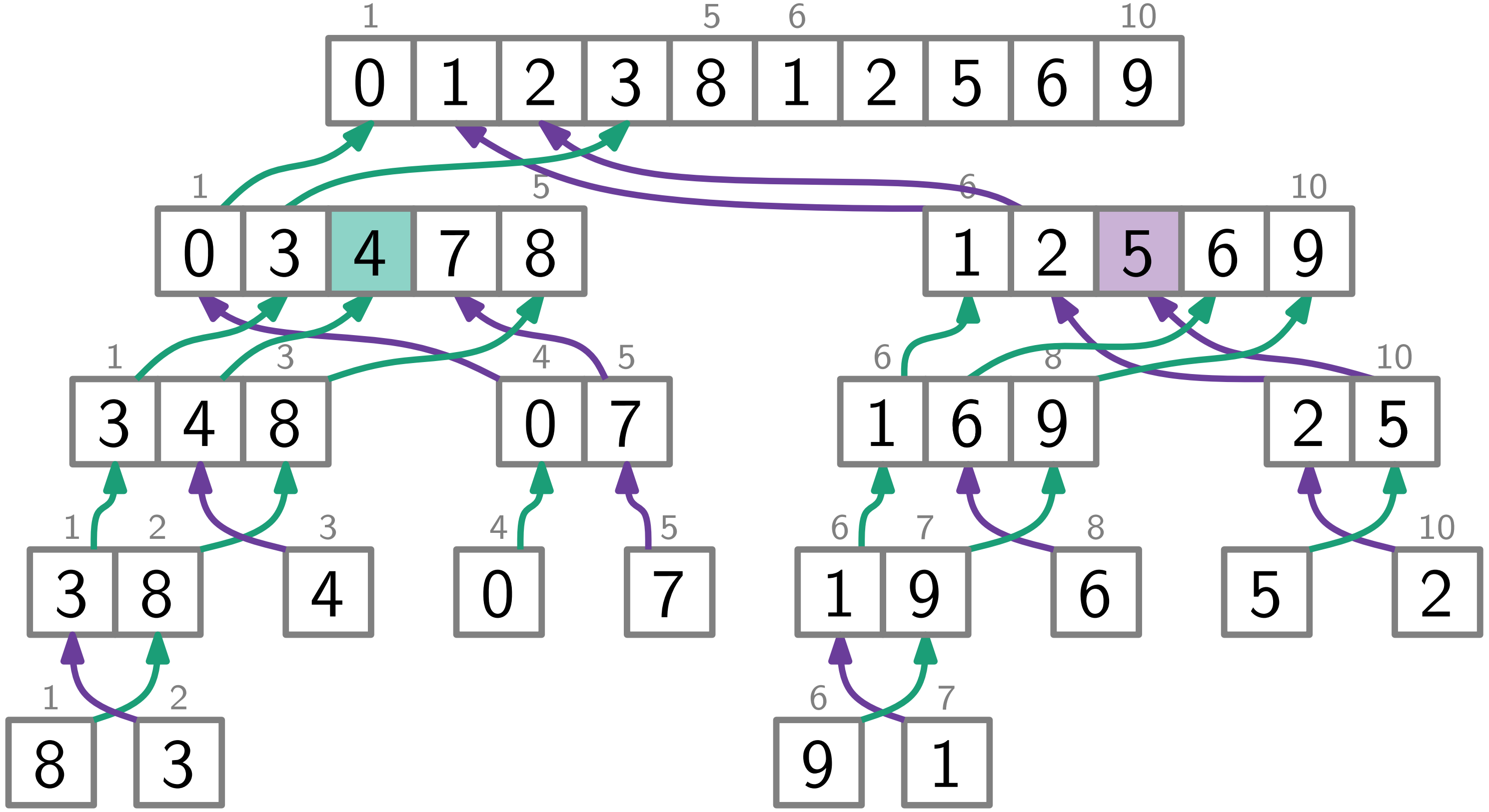


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

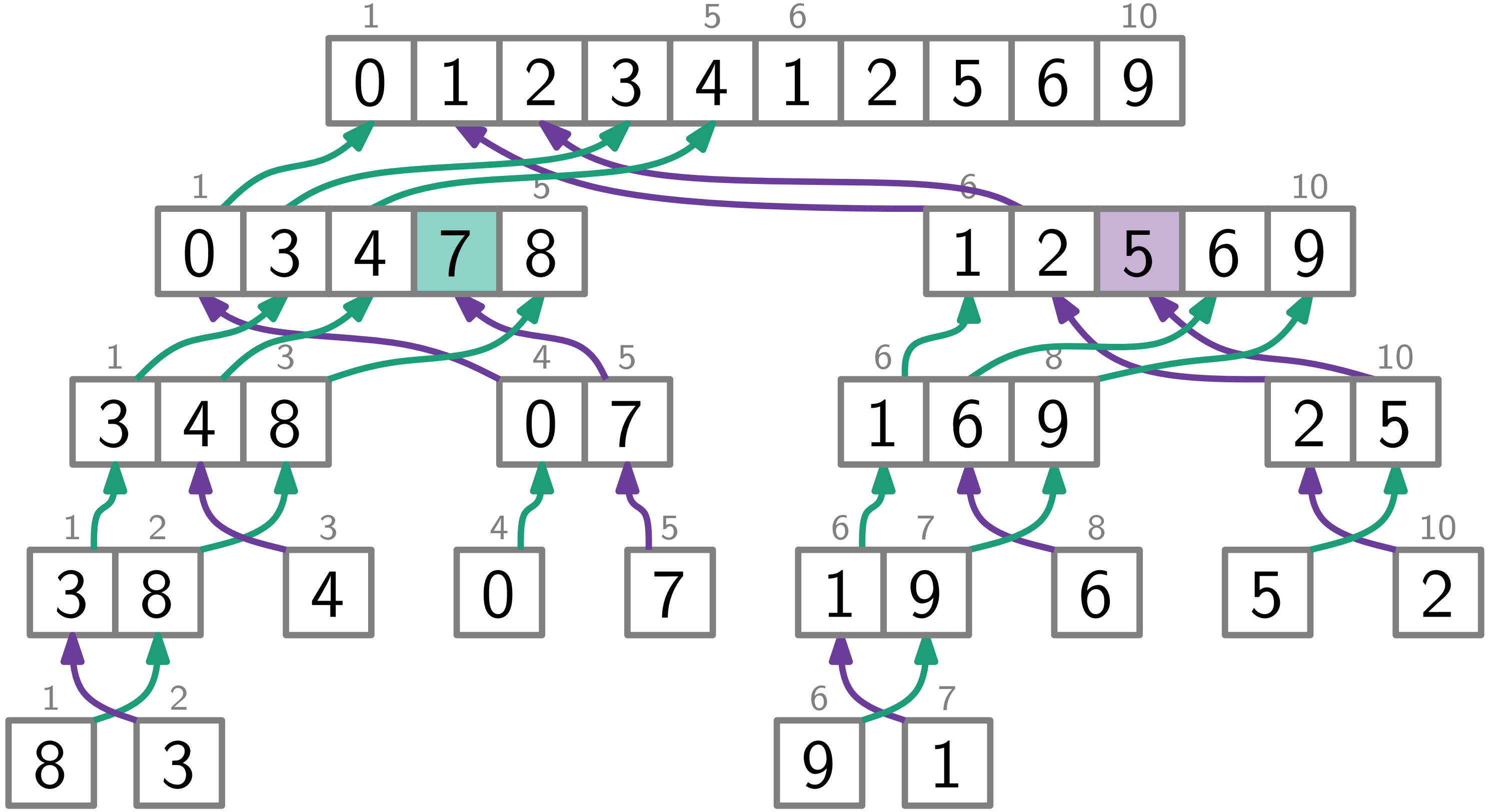


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

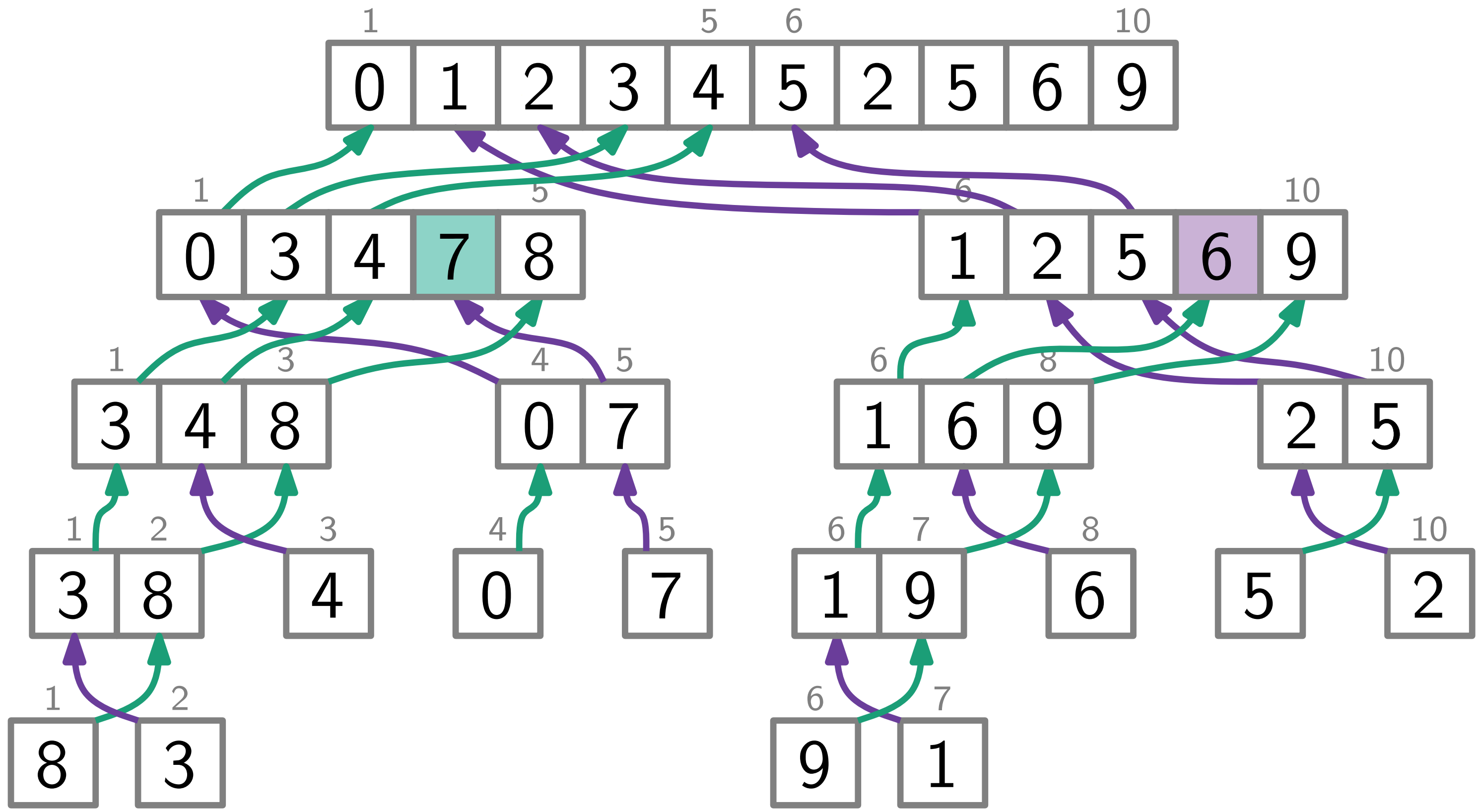


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$ } **herrsche**
- $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

$\text{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

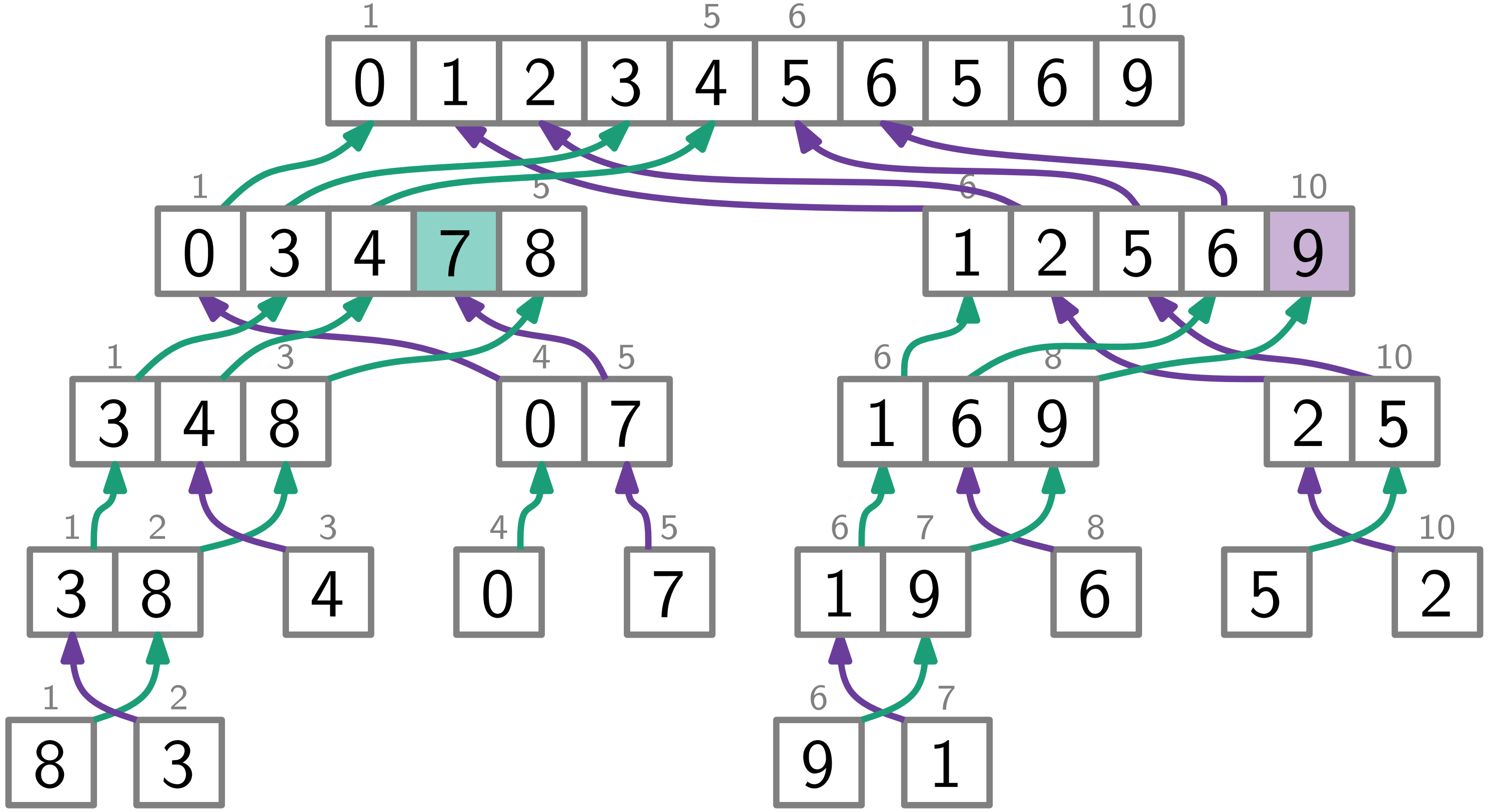
if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

$\text{MergeSort}(A, \ell, m)$

$\text{MergeSort}(A, m + 1, r)$ } **herrsche**

$\text{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

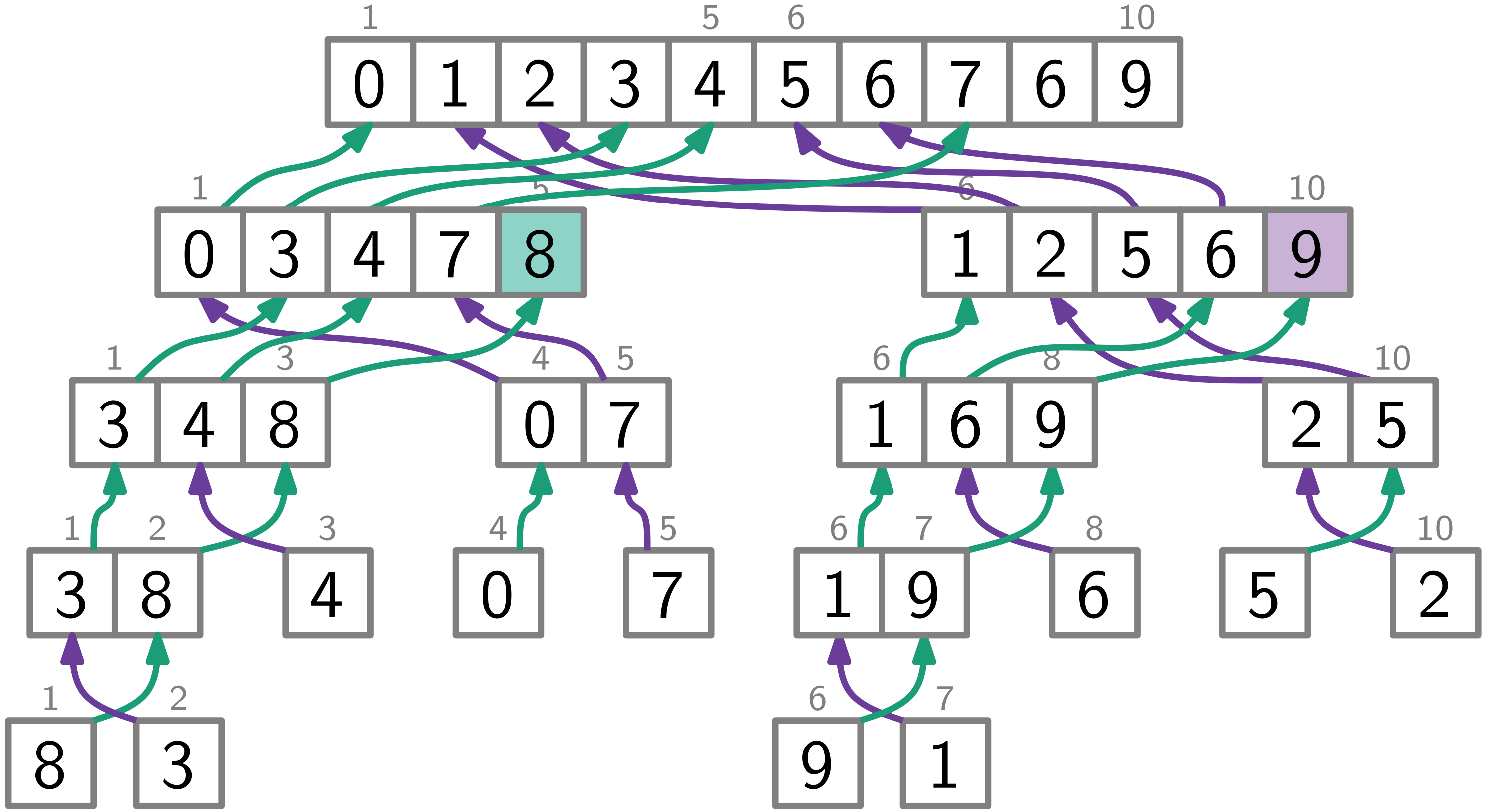
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

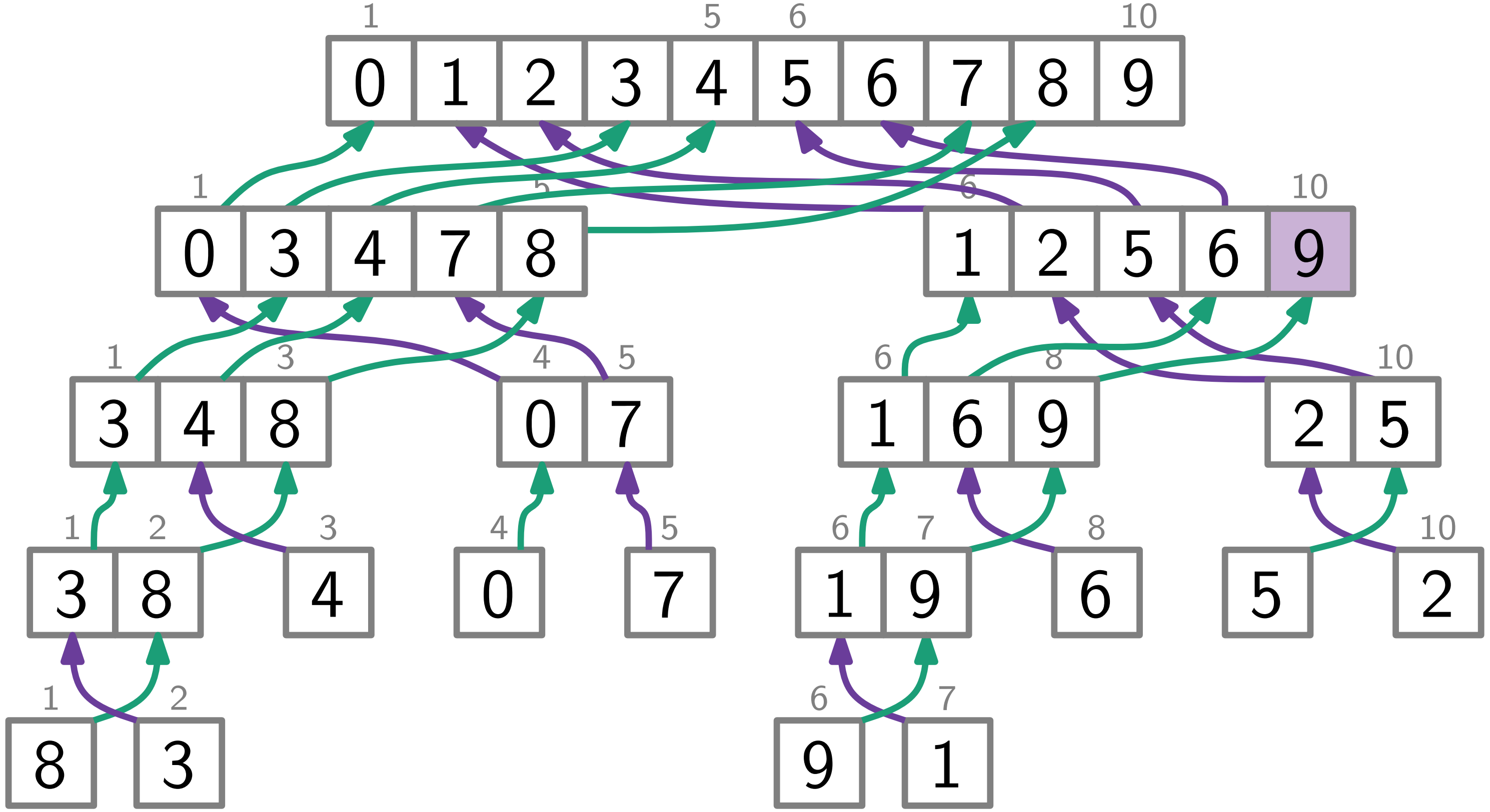
$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m) } **herrsche**
  MergeSort($A, m + 1, r$)
  Merge(A, ℓ, m, r) } **kombiniere**

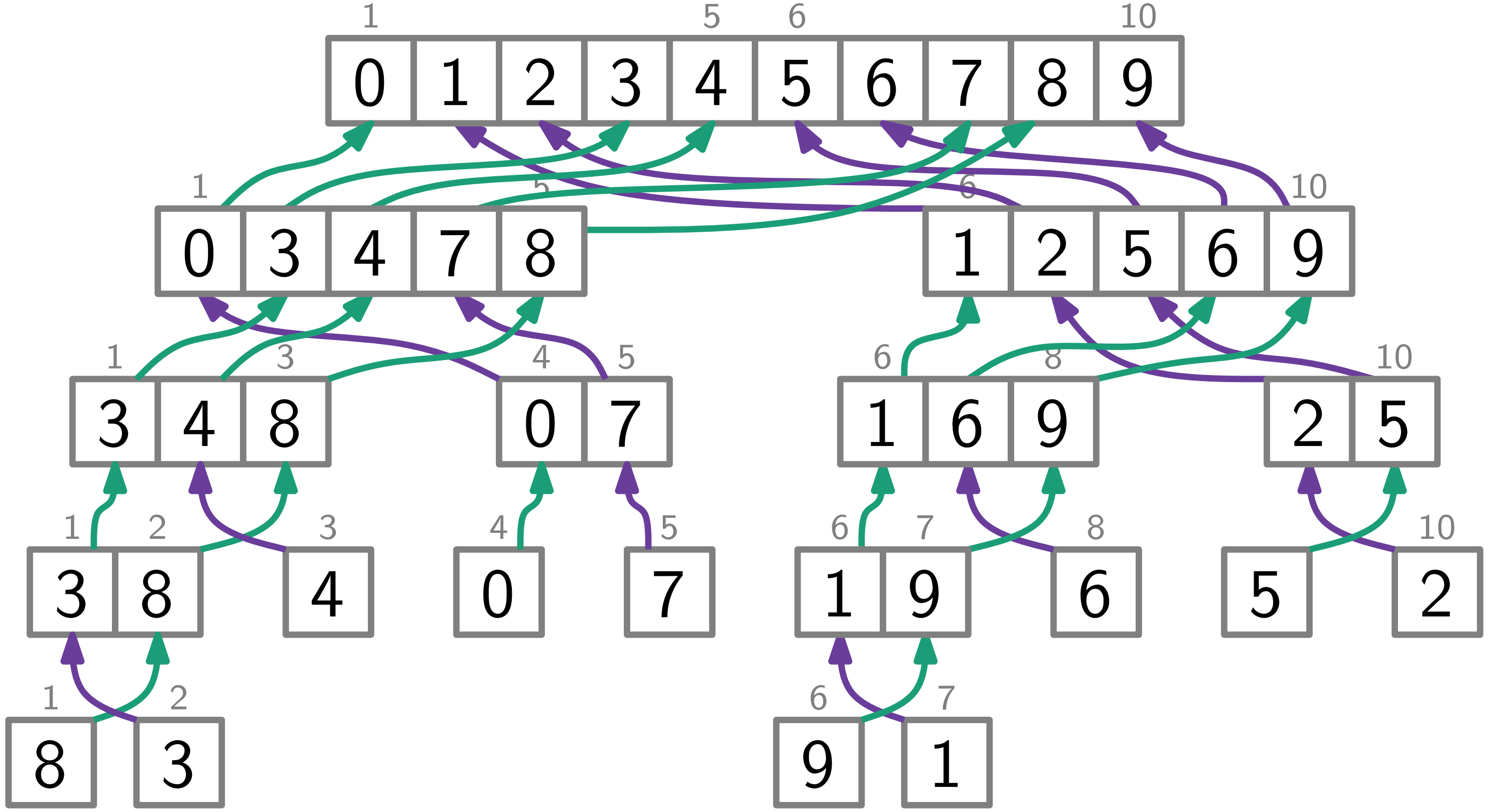


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

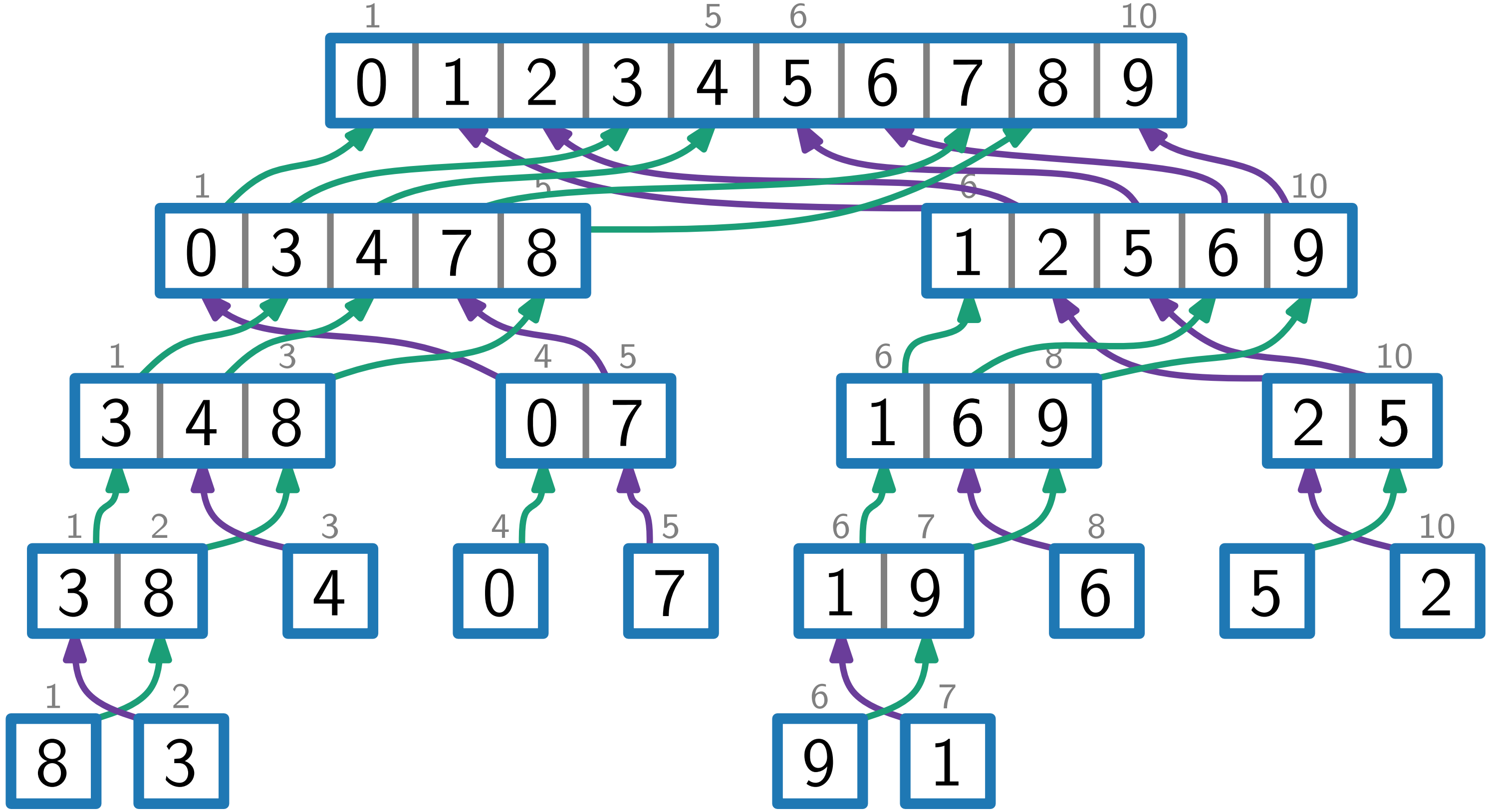


MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



MergeSort – ein Beispiel

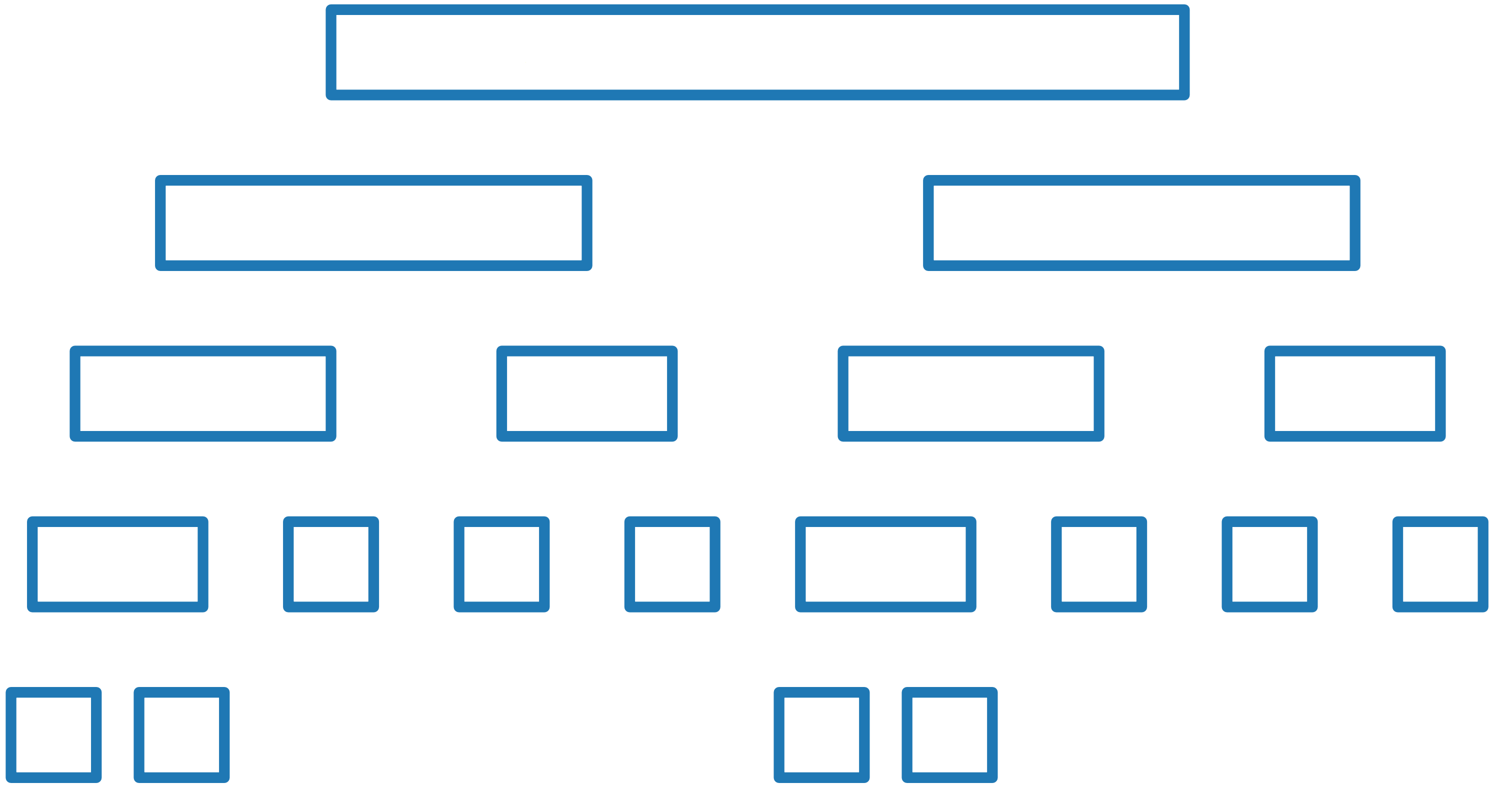
$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

if $\ell < r$ **then**

$\left.\begin{array}{l} m = \lfloor(\ell + r)/2\rfloor \end{array}\right\}$ **teile**

$\left.\begin{array}{l} \textsc{MergeSort}(A, \ell, m) \\ \textsc{MergeSort}(A, m+1, r) \end{array}\right\}$ **herrsche**

$\left.\begin{array}{l} \textsc{Merge}(A, \ell, m, r) \end{array}\right\}$ **kombiniere**



MergeSort – ein Beispiel

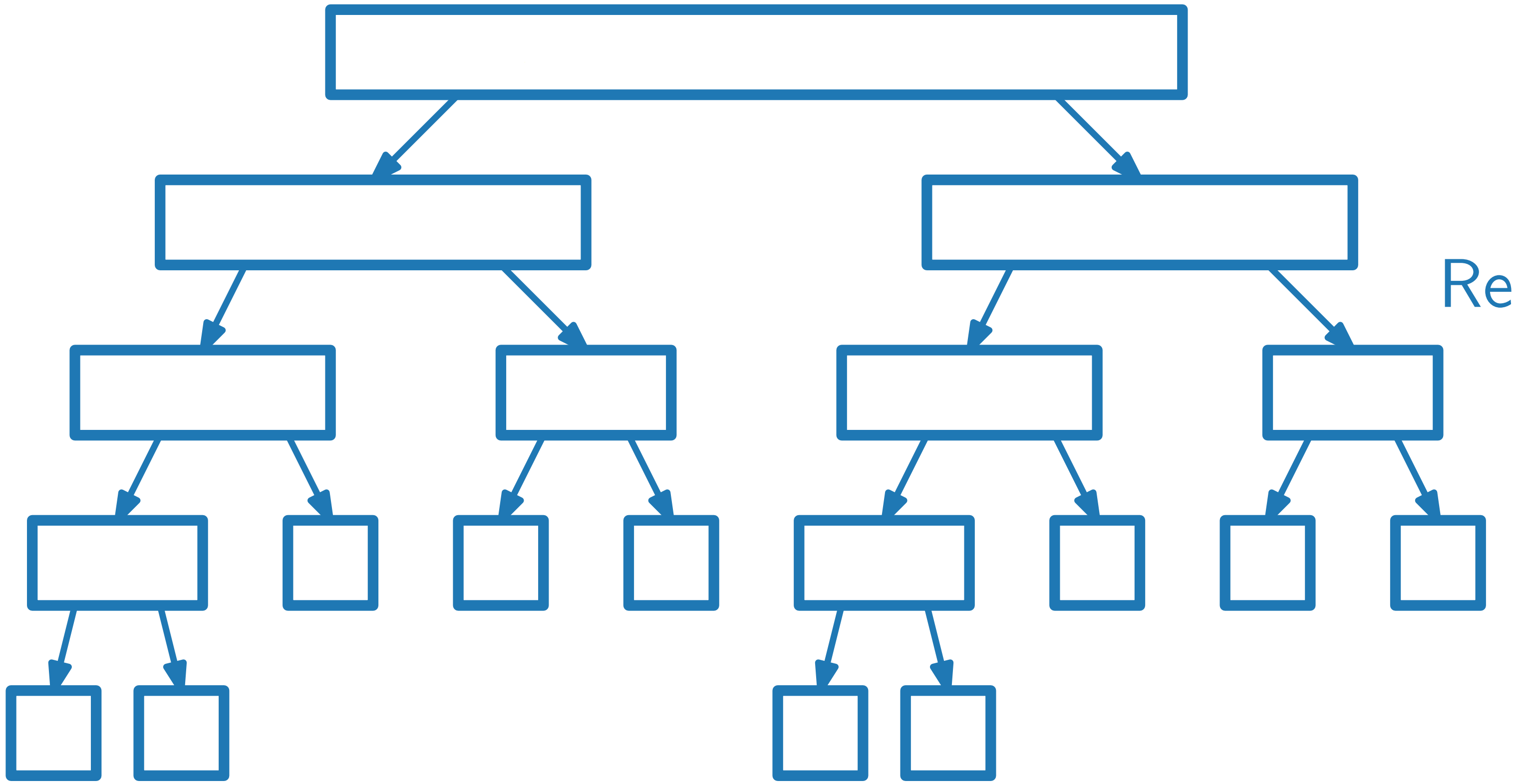
$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



Rekursionsbaum von $\textsc{MergeSort}$

MergeSort – ein Beispiel

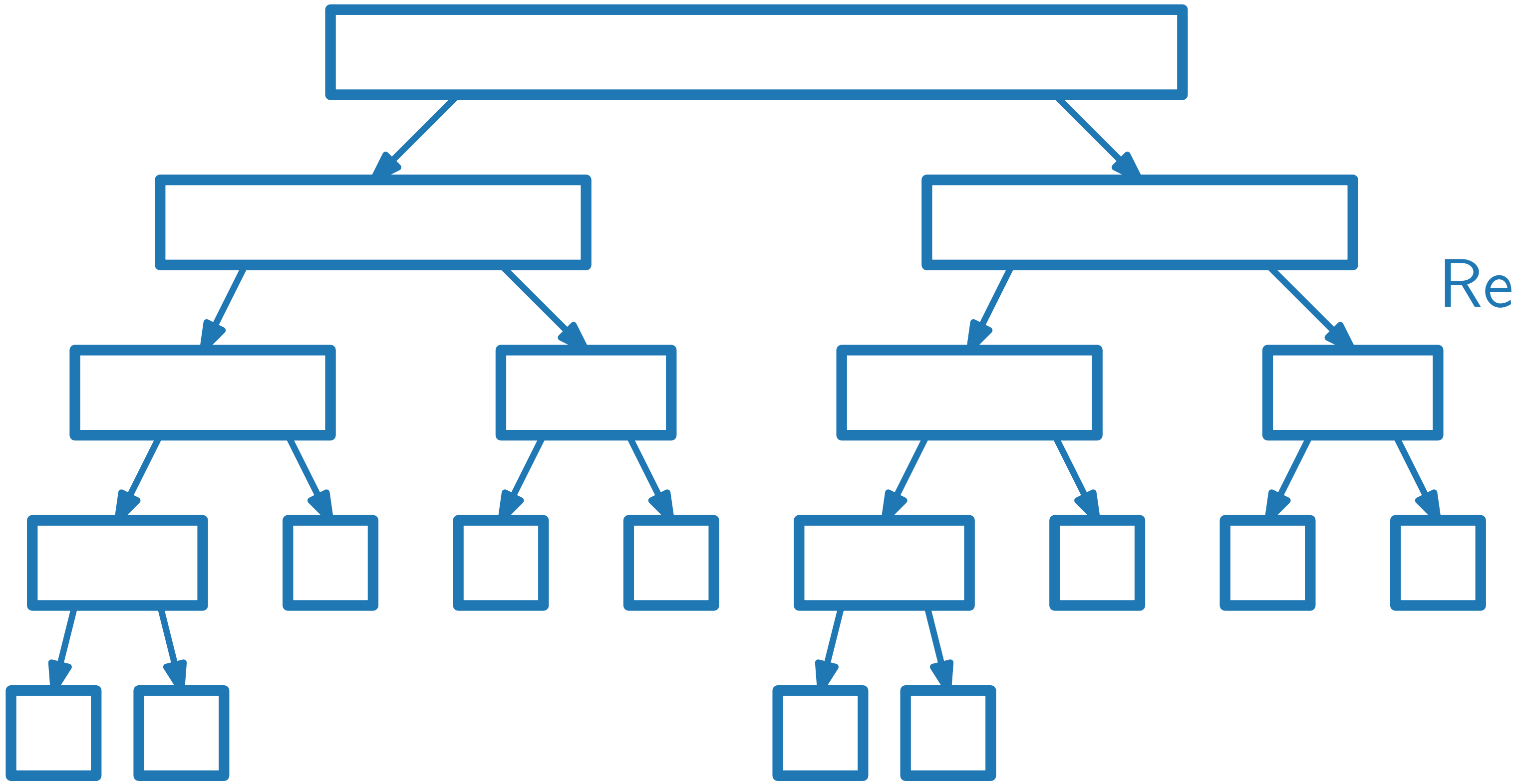
$\textsc{MergeSort}(\text{int}[]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



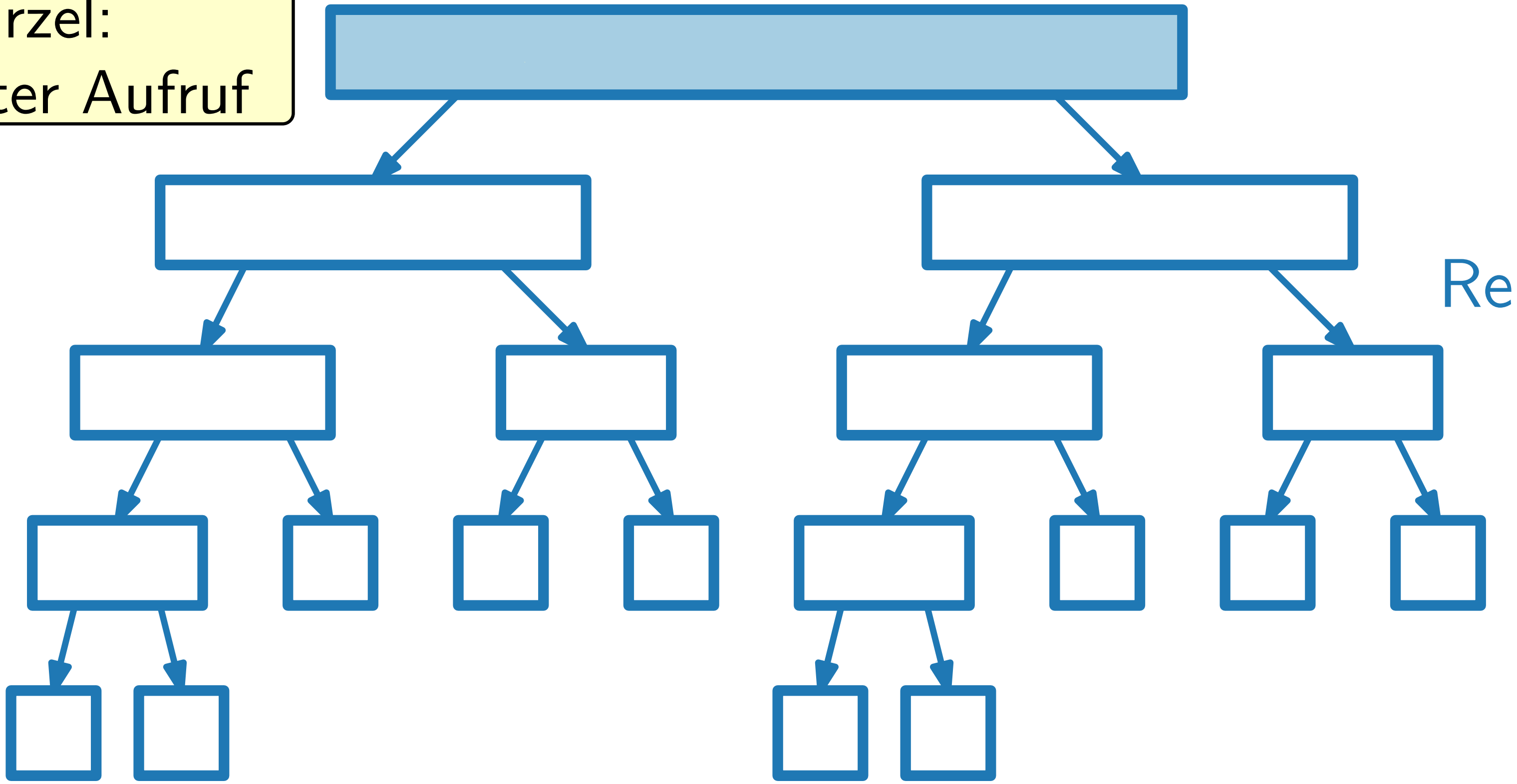
Rekursionsbaum von MergeSort

Baum der rekursiven Aufrufe

MergeSort – ein Beispiel

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m) } **herrsche**
  MergeSort($A, m + 1, r$)
  Merge(A, ℓ, m, r) } **kombiniere**

Wurzel:
erster Aufruf



Rekursionsbaum von MergeSort

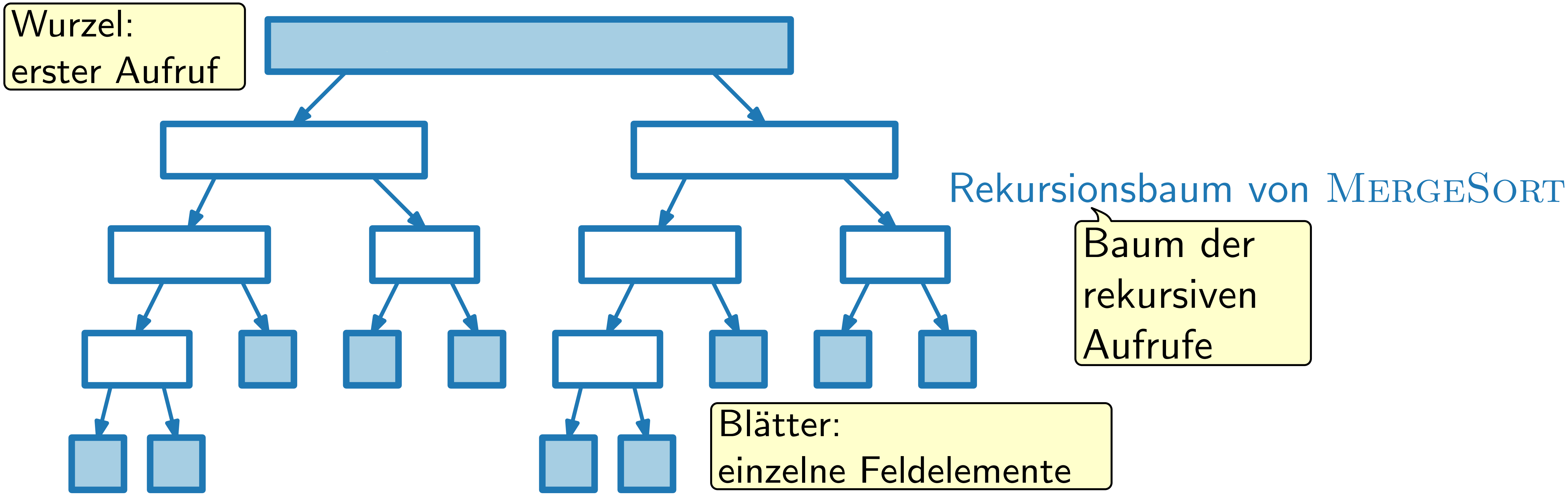
Baum der
rekursiven
Aufrufe

MergeSort – ein Beispiel

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**



Korrektheit von Merge

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von Merge

... nach Schema „F“!

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

MERGE(int[] A, int ℓ, int m, int r)

```
n_1 = m − ℓ + 1;  n_2 = r − m
L = new int[1 ... n_1 + 1]; R = new int[1 ... n_2 + 1]
L[1 ... n_1] = A[ℓ ... m]
R[1 ... n_2] = A[m + 1 ... r]
L[n_1 + 1] = R[n_2 + 1] = ∞
i = j = 1
for k = ℓ to r do
    if L[i] ≤ R[j] then
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
```

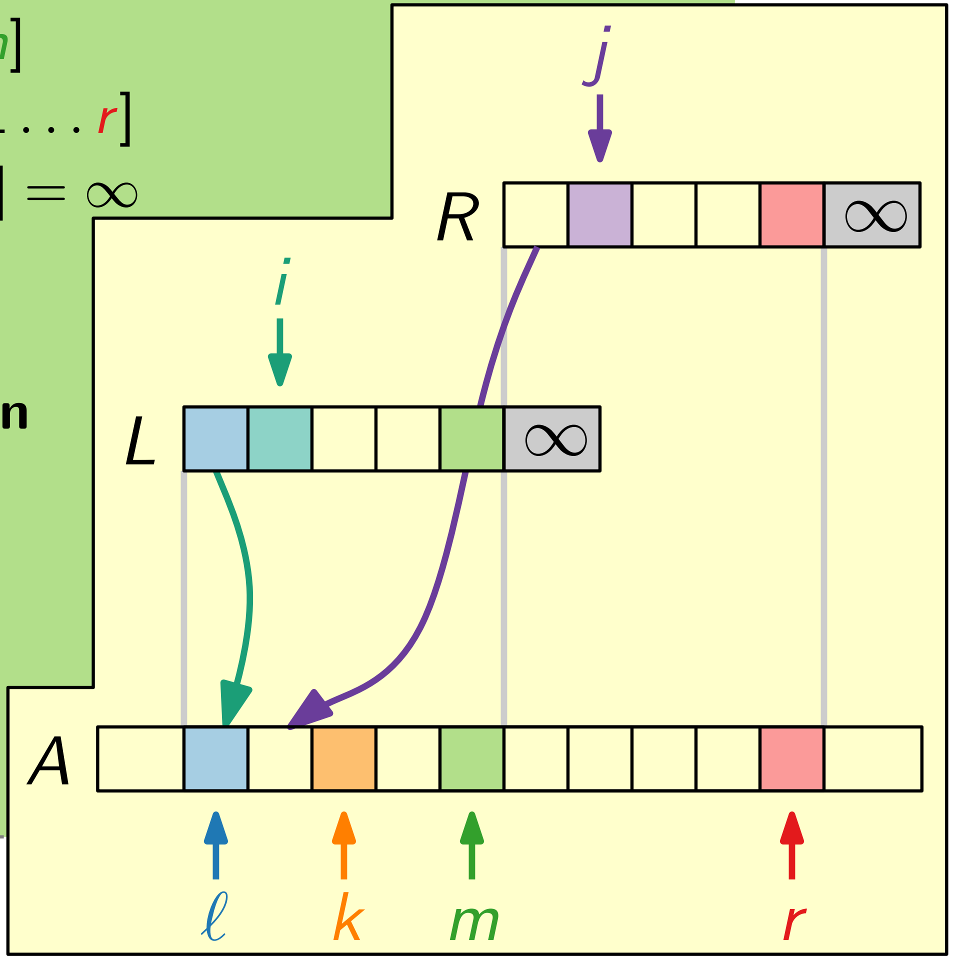
Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m + 1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von MERGE

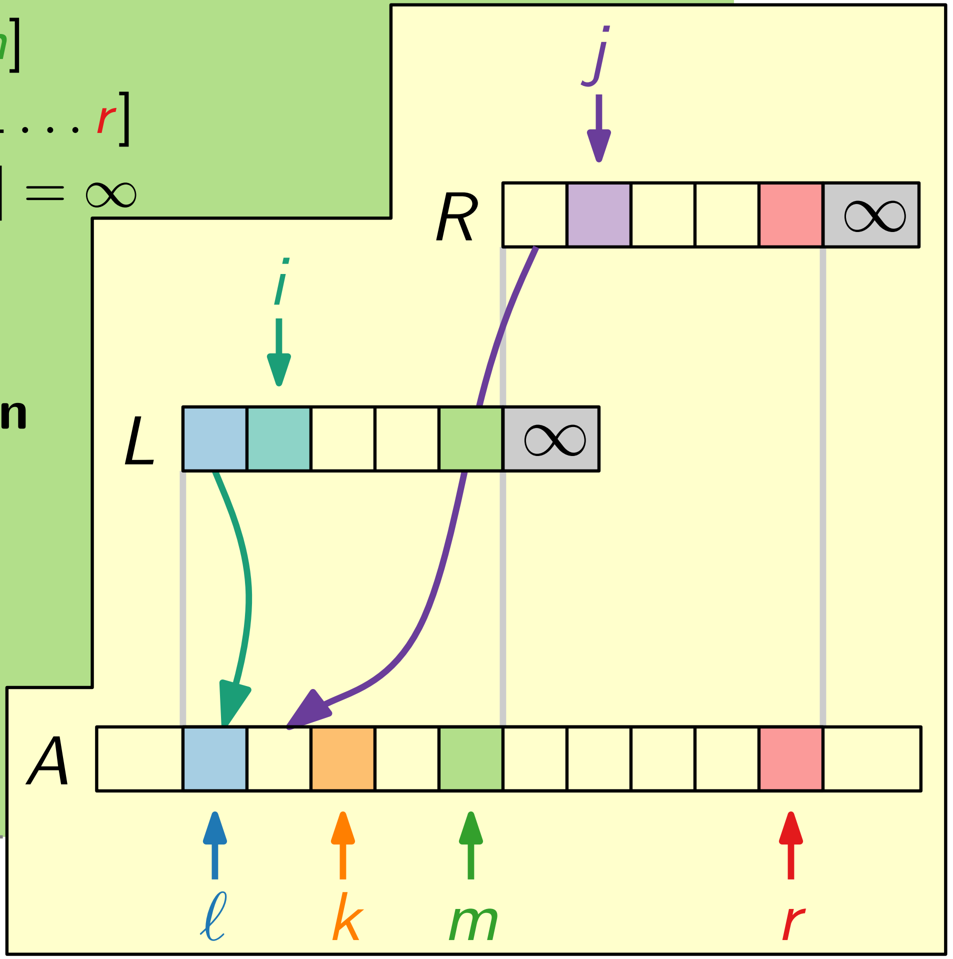
... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m + 1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
    if $L[i] \leq R[j]$ **then**
        $A[k] = L[i]$
        $i = i + 1$
    else
        $A[k] = R[j]$
        $j = j + 1$



Korrektheit von MERGE

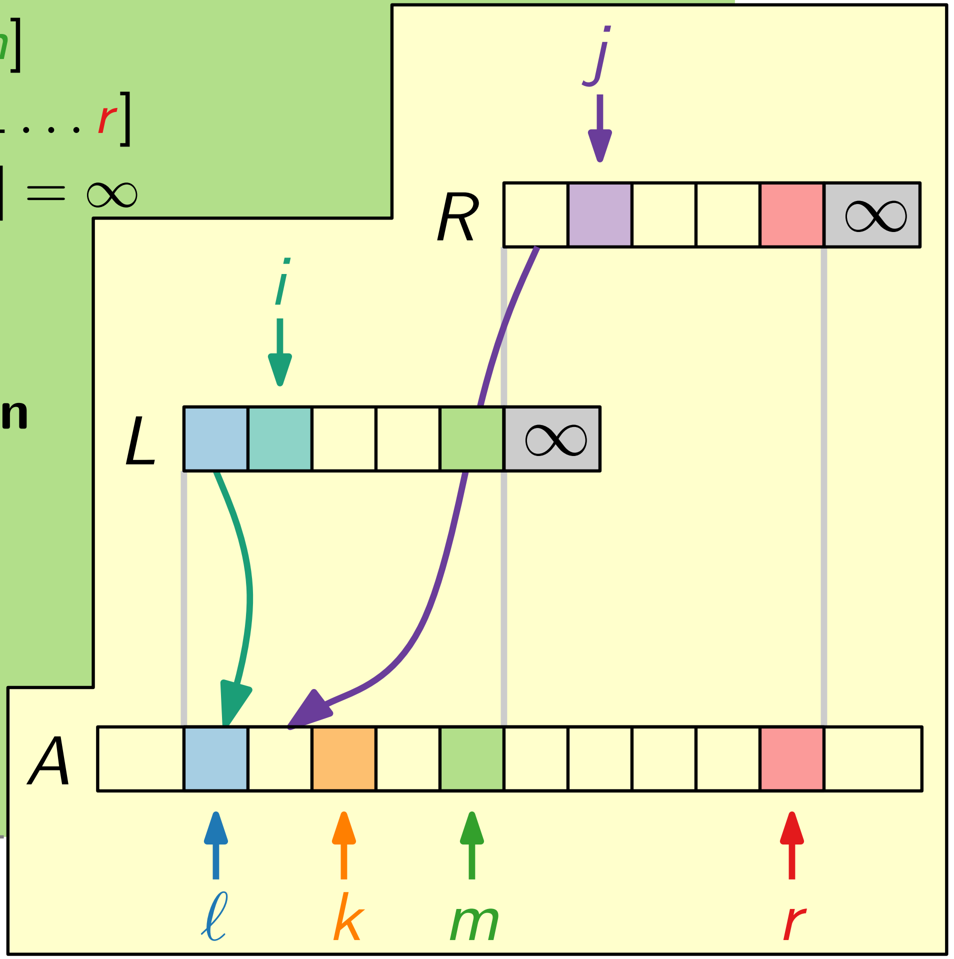
. . . nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1;\ \ n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von MERGE

... nach Schema „F“!

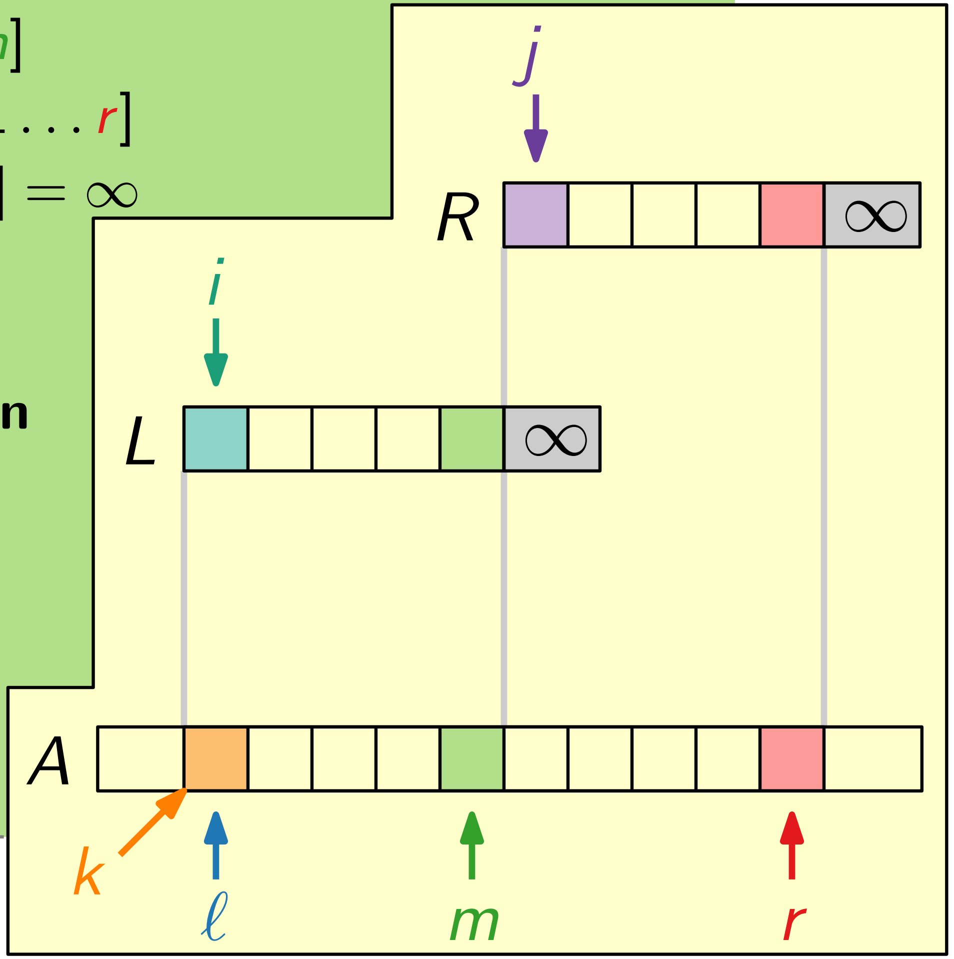
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

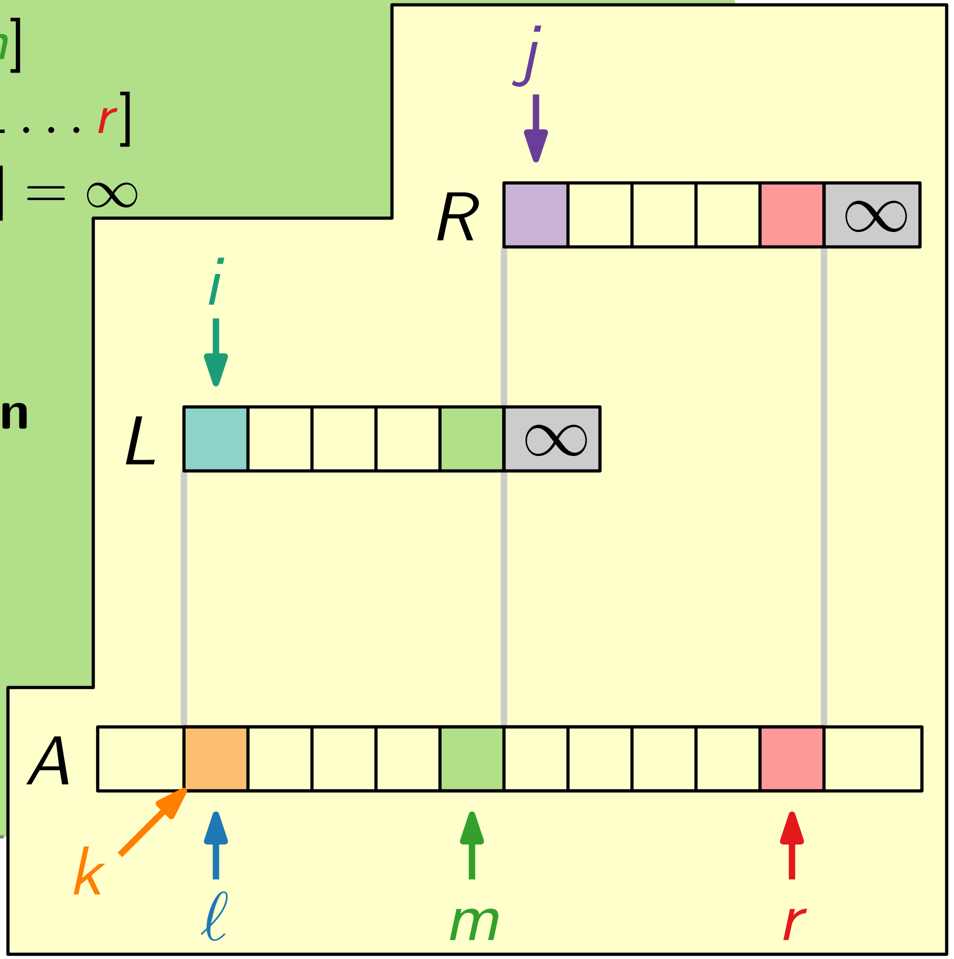
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung

- Da beim ersten Schleifendurchlauf $k = \ell$ gilt, enthält $A[\ell \dots k-1] = \langle\rangle$ die 0 kleinsten Elemente von $L \cup R$.

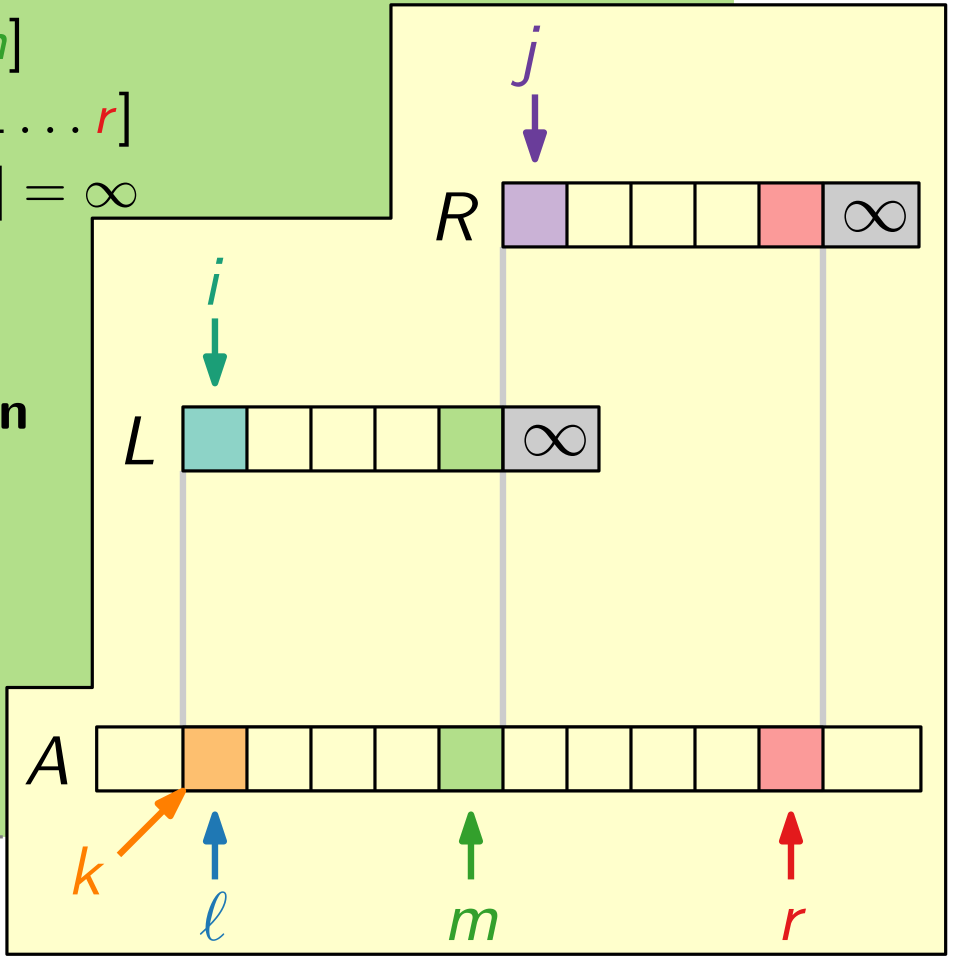
MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \le R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von MERGE

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

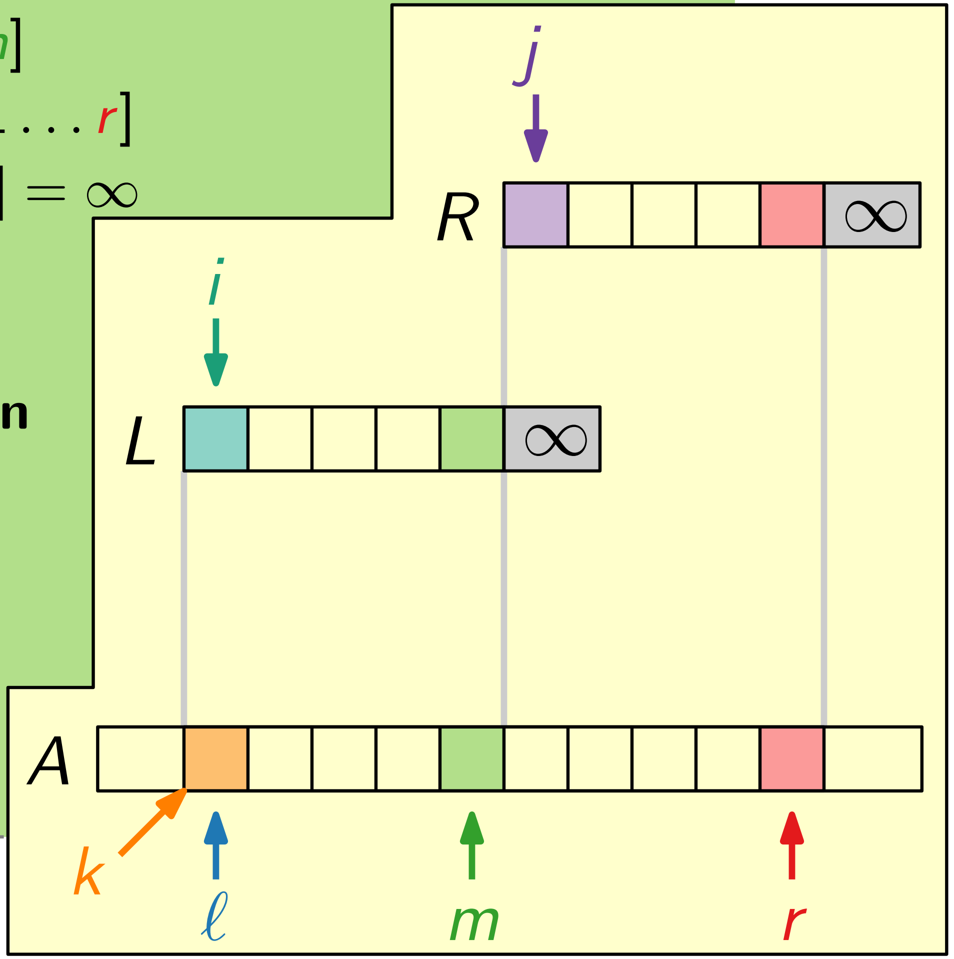
1. Initialisierung

- Da beim ersten Schleifendurchlauf $k = \ell$ gilt, enthält $A[\ell \dots k-1] = \langle\rangle$ die 0 kleinsten Elemente von $L \cup R$.
- Da $i = j = 1$, sind $L[i]$ und $R[j]$ die kleinsten noch nicht kopierten Elemente

Korrektheit von MERGE

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m + 1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
    if $L[i] \leq R[j]$ **then**
        $A[k] = L[i]$
        $i = i + 1$
    else
        $A[k] = R[j]$
        $j = j + 1$



… nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k - \ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

- Da beim ersten Schleifendurchlauf $k = \ell$ gilt, enthält $A[\ell \dots k-1] = \langle\rangle$ die 0 kleinsten Elemente von $L \cup R$.
- Da $i = j = 1$, sind $L[i]$ und $R[j]$ die kleinsten noch nicht kopierten Elemente

Korrektheit von MERGE

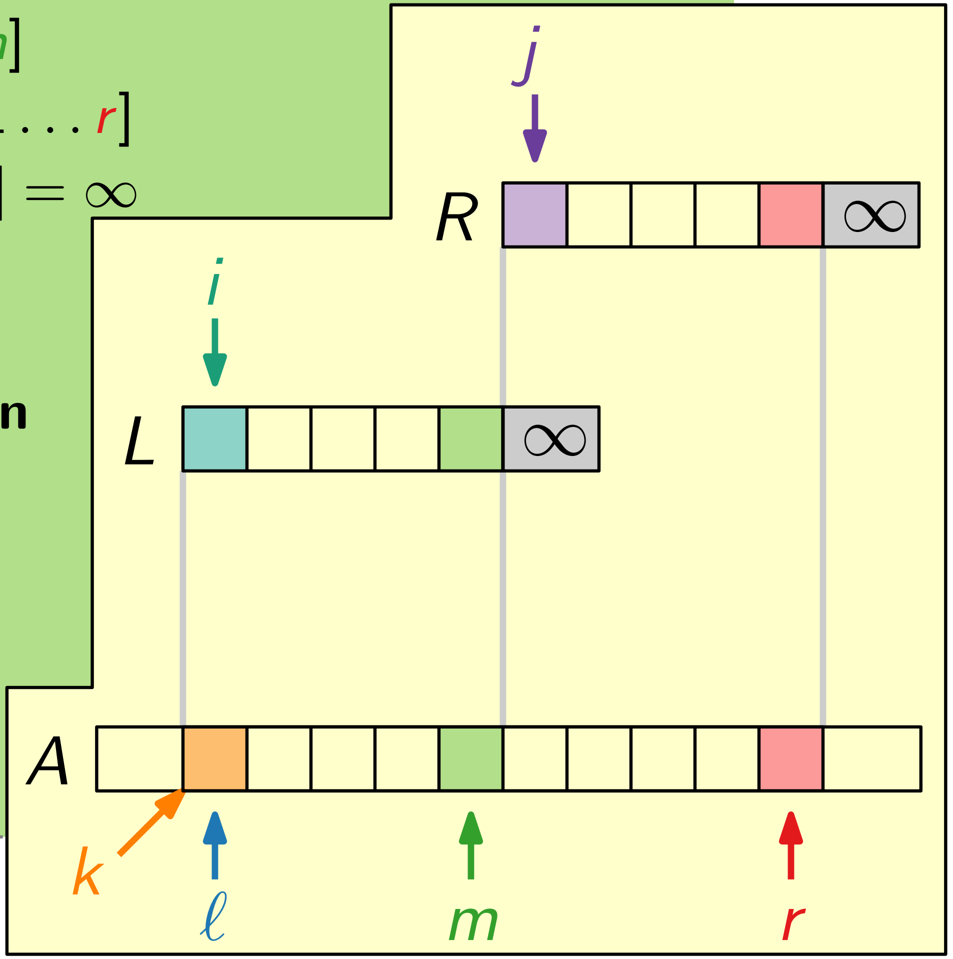
... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



Korrektheit von MERGE

. . . nach Schema „F“!

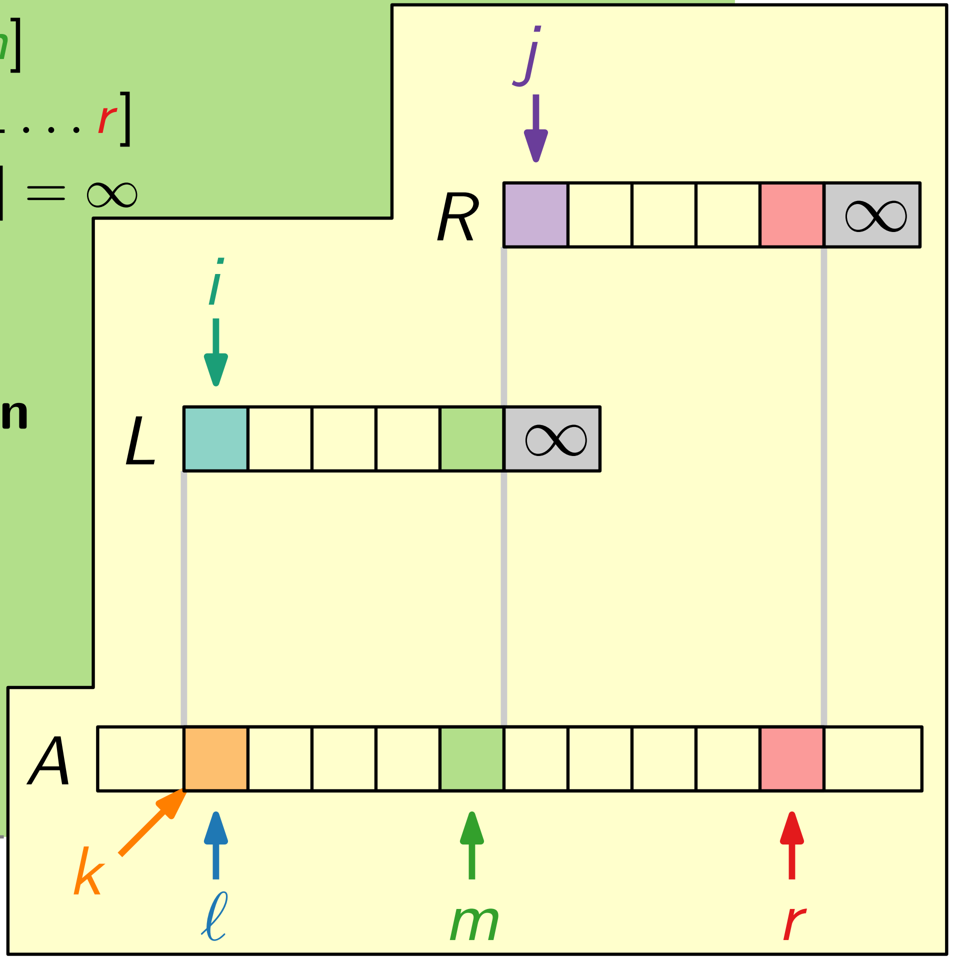
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

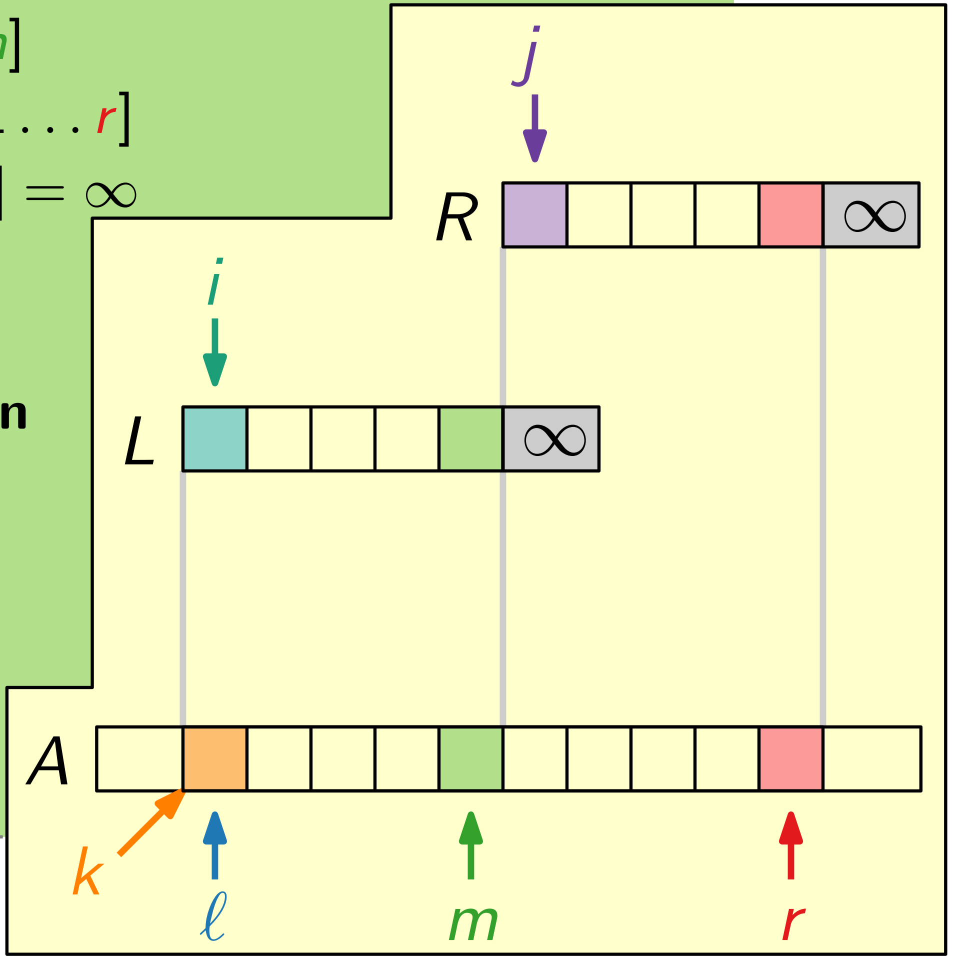
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1;\ \ n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.

Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$

$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m+1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for $k = \ell$ **to** r **do**

  if $L[i] \leq R[j]$ **then**

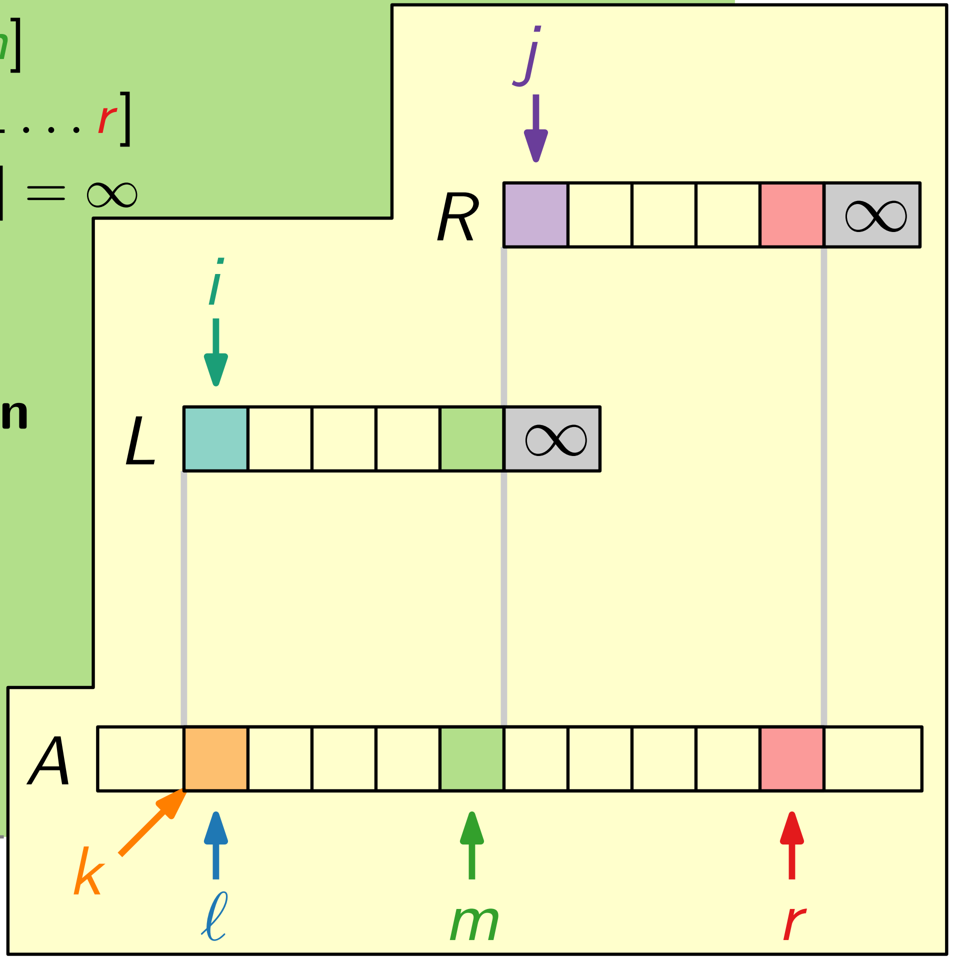
    $A[k] = L[i]$

    $i = i + 1$

  else

    $A[k] = R[j]$

    $j = j + 1$



Korrektheit von Merge

... nach Schema „F“!

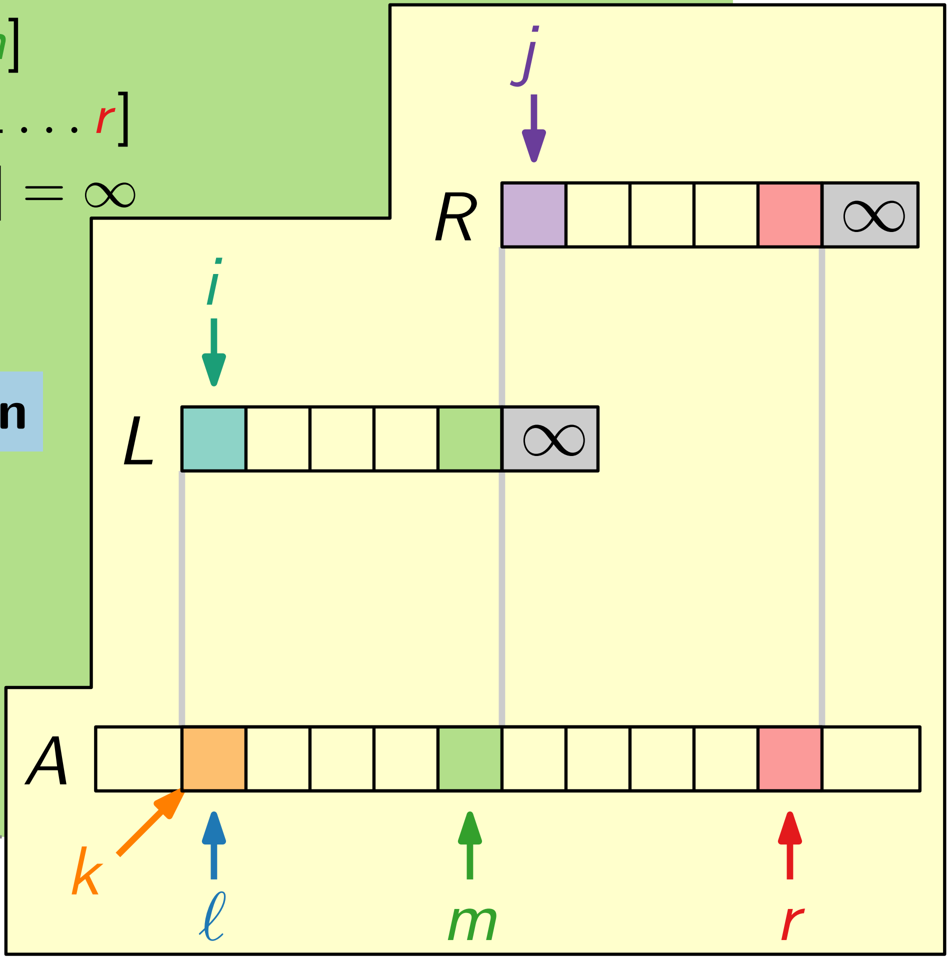
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.

Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

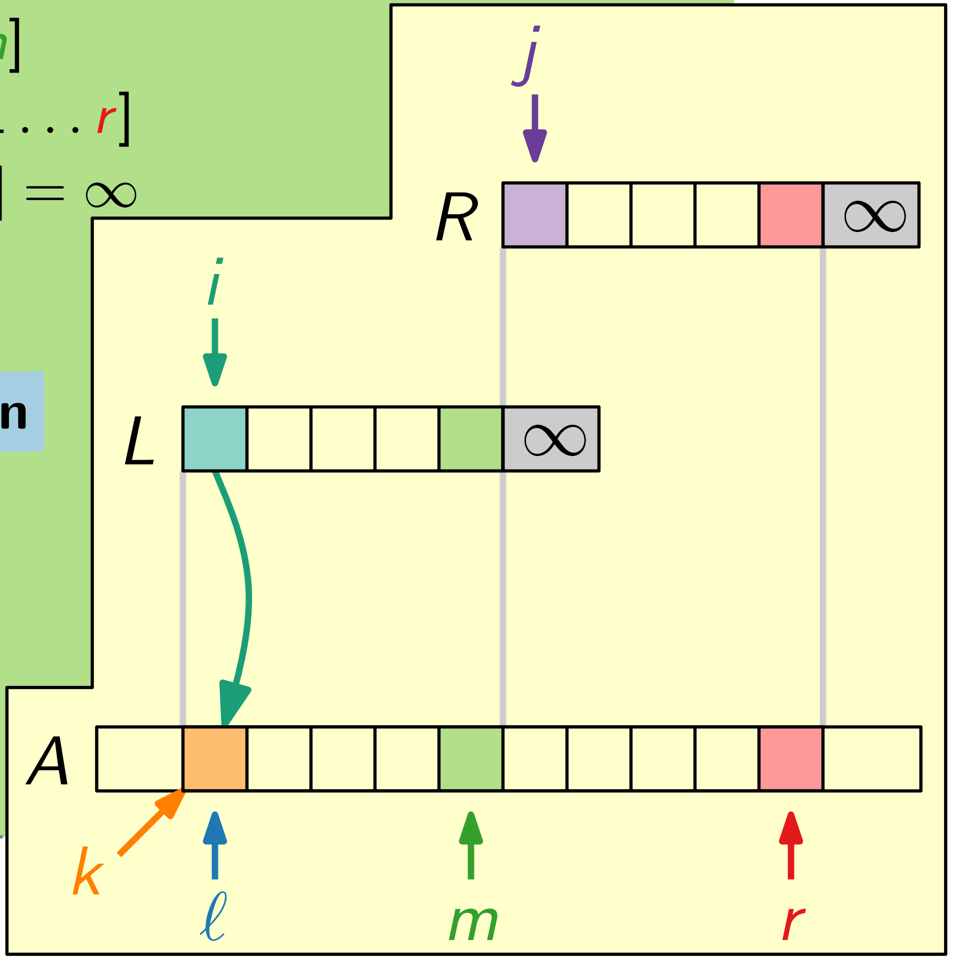
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

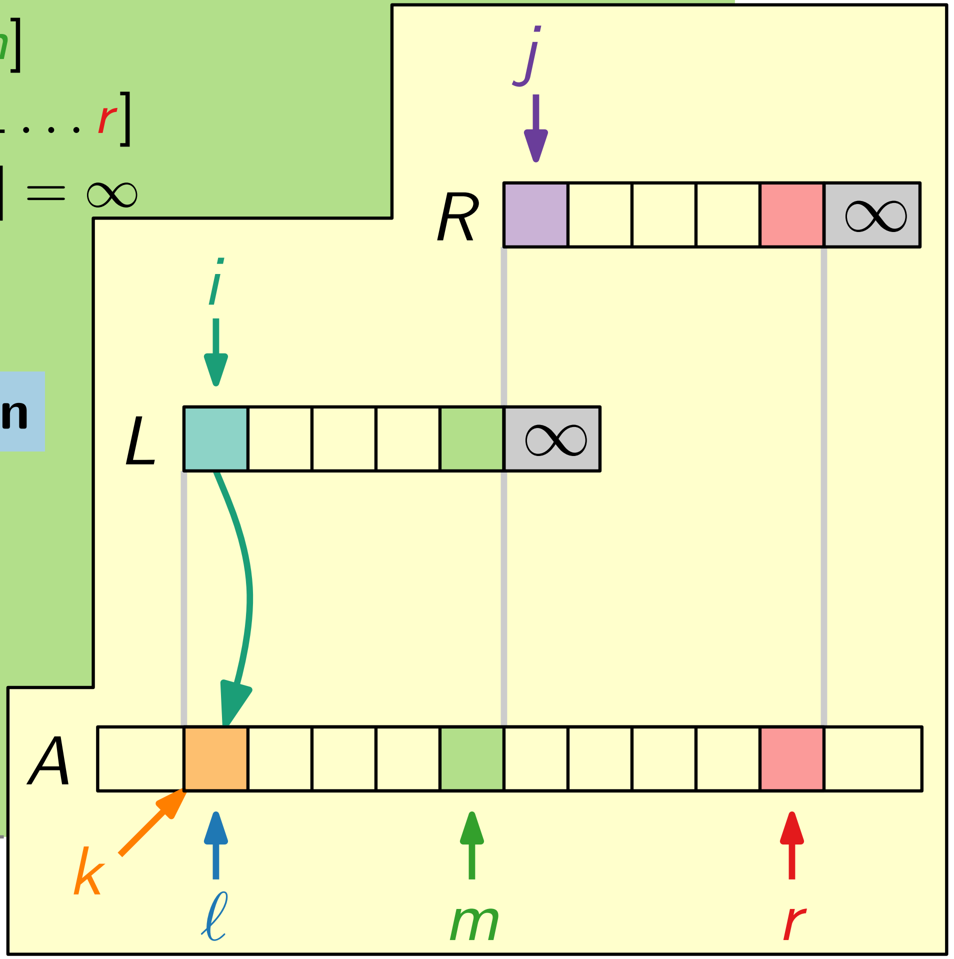
1. Initialisierung ✓

2. Aufrechterhaltung

- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

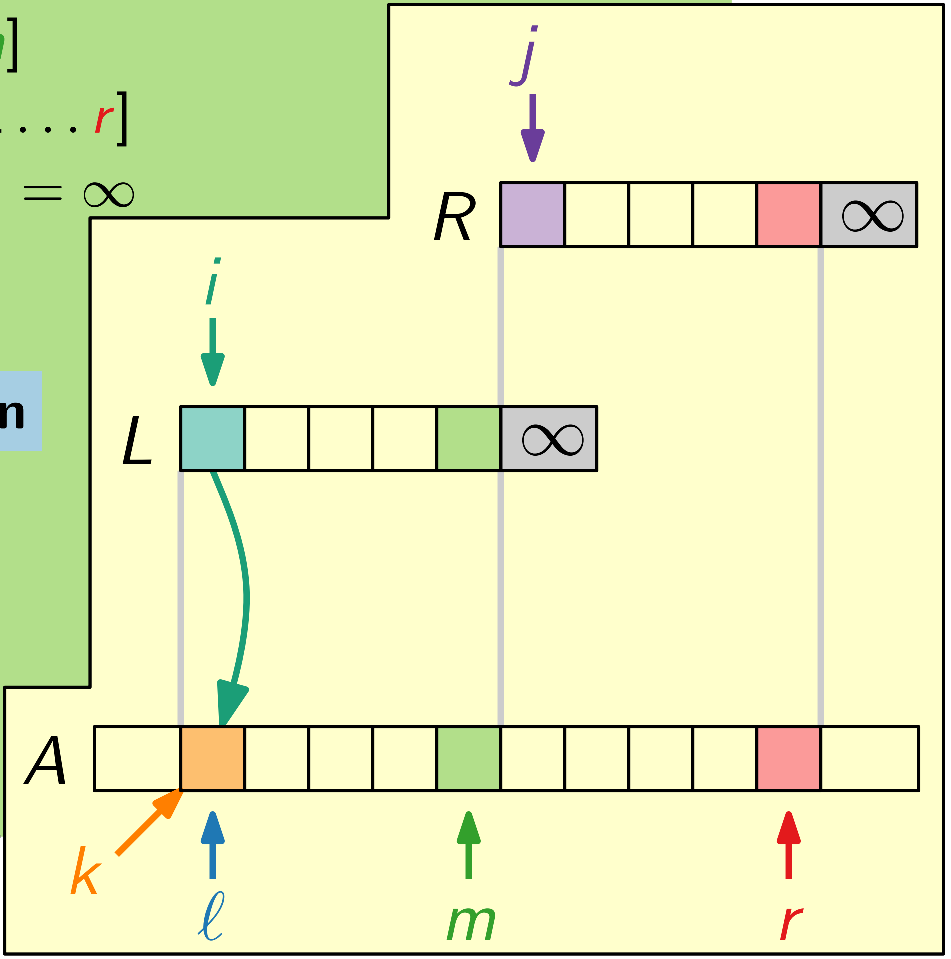
- $A[\ell \ldots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \ldots n_1 + 1]$; $R =$ **new** int$[1 \ldots n_2 + 1]$
$L[1 \ldots n_1] = A[\ell \ldots m]$
$R[1 \ldots n_2] = A[m+1 \ldots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt: dank Invariante
  - $A[\ell \ldots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.

Korrektheit von MERGE

... nach Schema „F“!

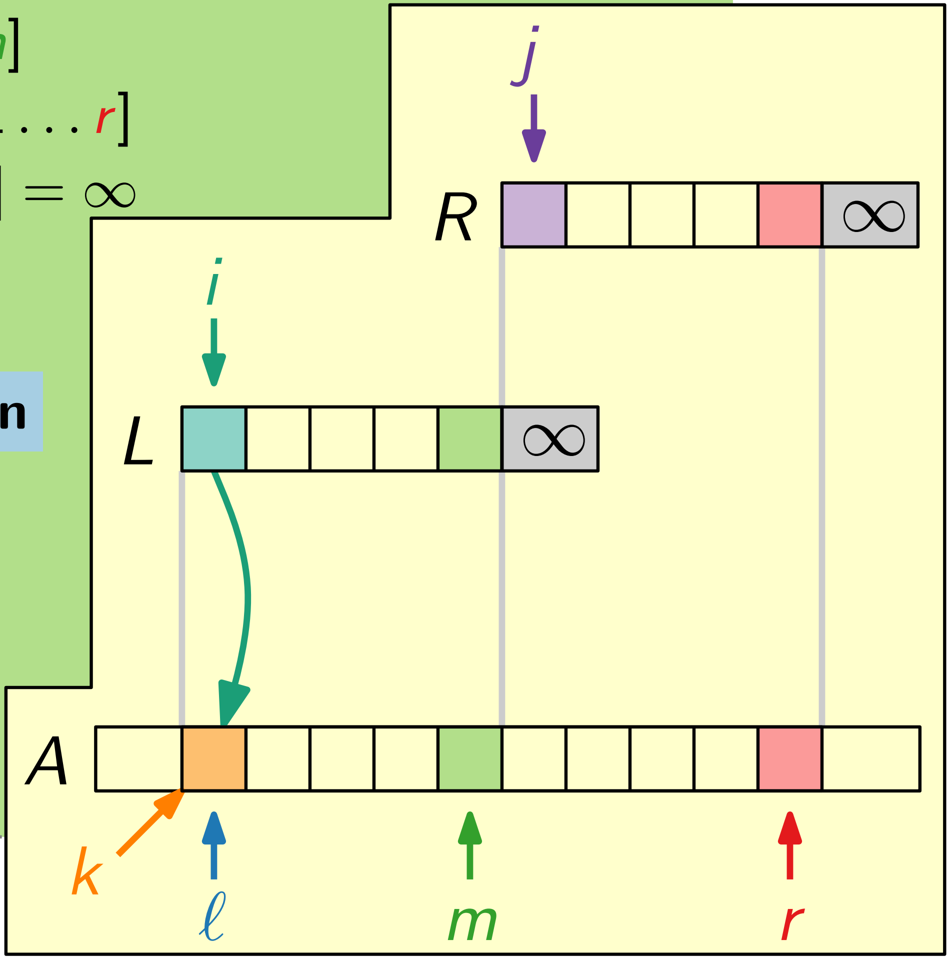
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



- Zwei Fälle: (a) $L[i] \le R[j]$, (b) $R[j] < L[i]$.
- Nun gilt: dank Invariante
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

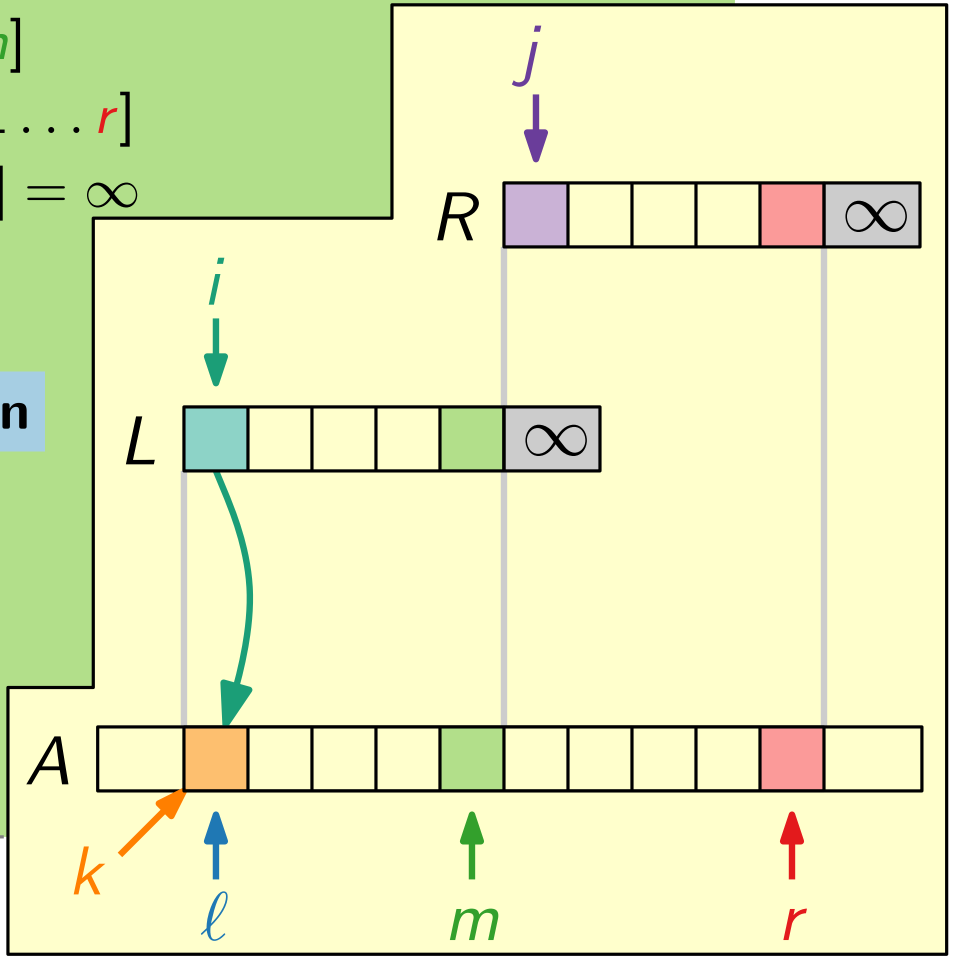
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i

Korrektheit von Merge

… nach Schema „F“!

0. Schleifeninvariante

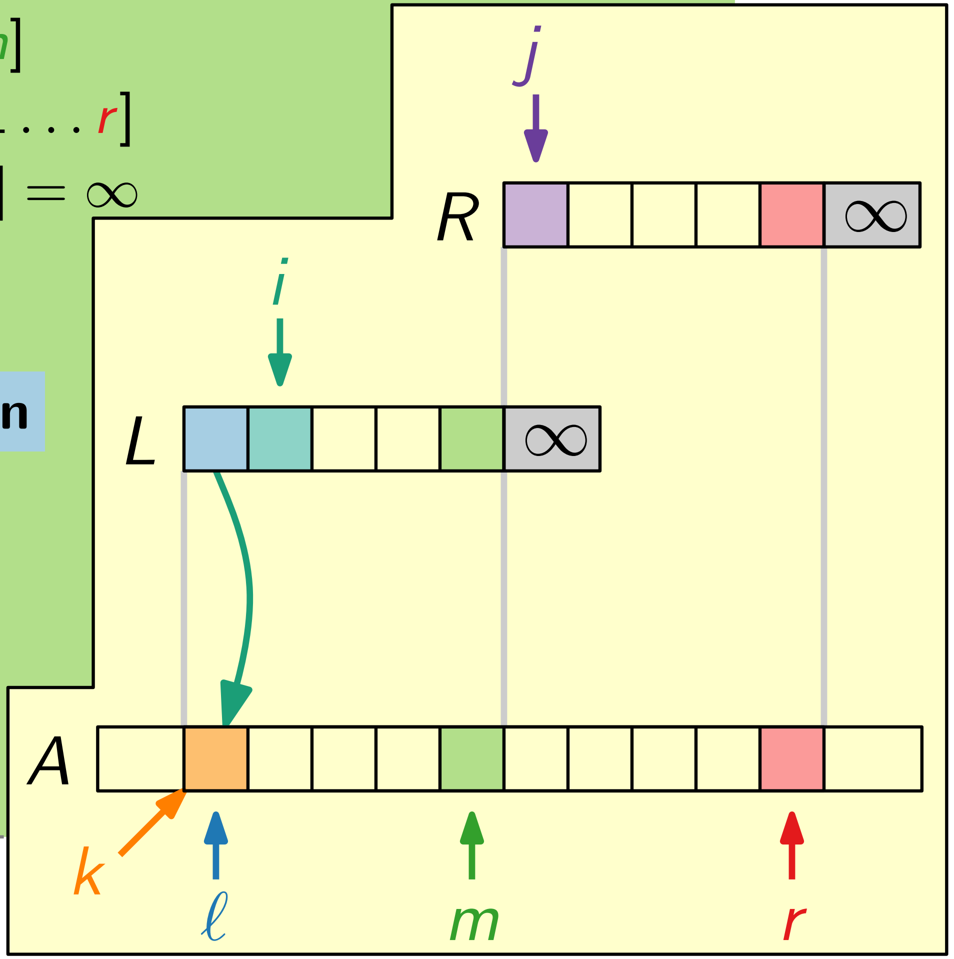
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1;\ \ n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i

Korrektheit von MERGE

… nach Schema „F“!

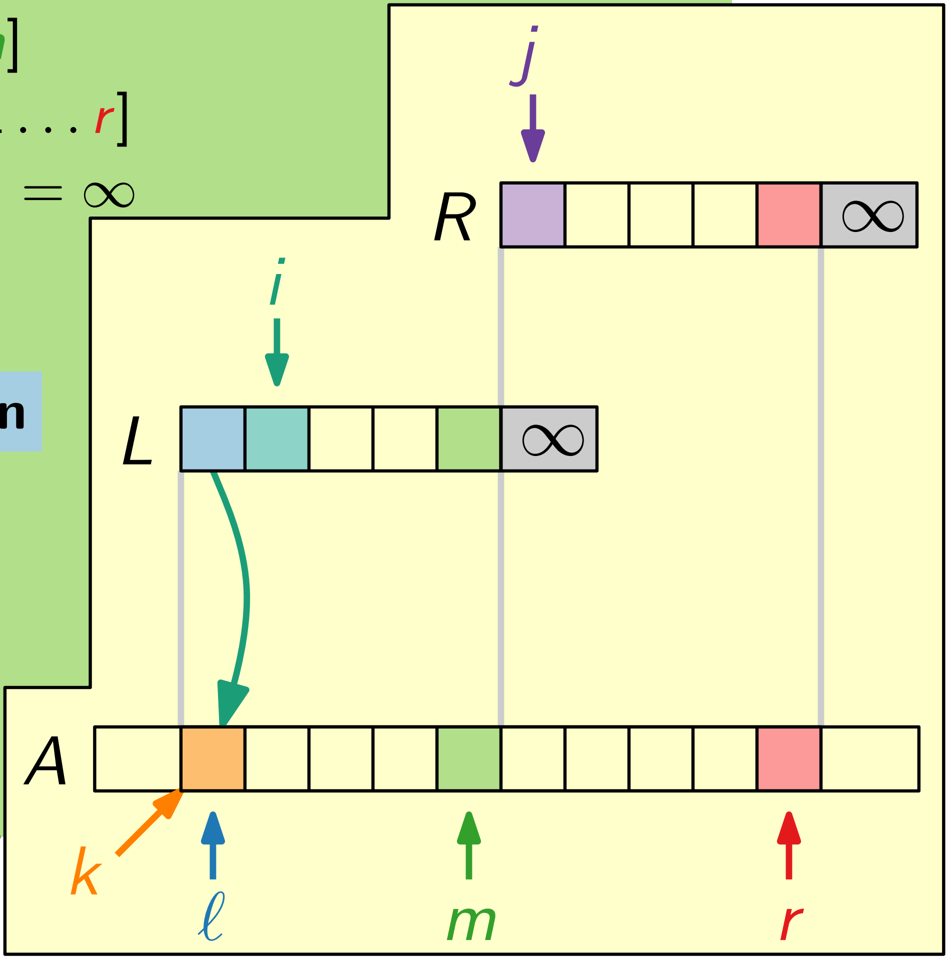
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$

Korrektheit von MERGE

... nach Schema „F“!

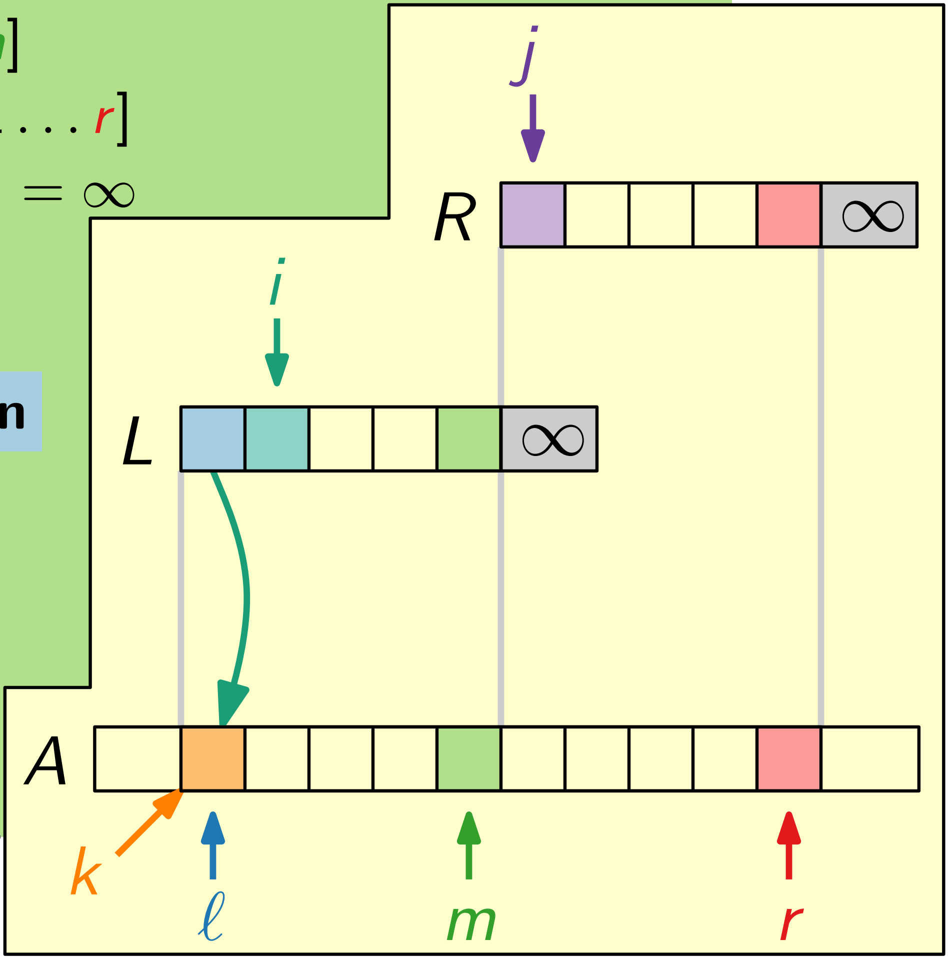
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

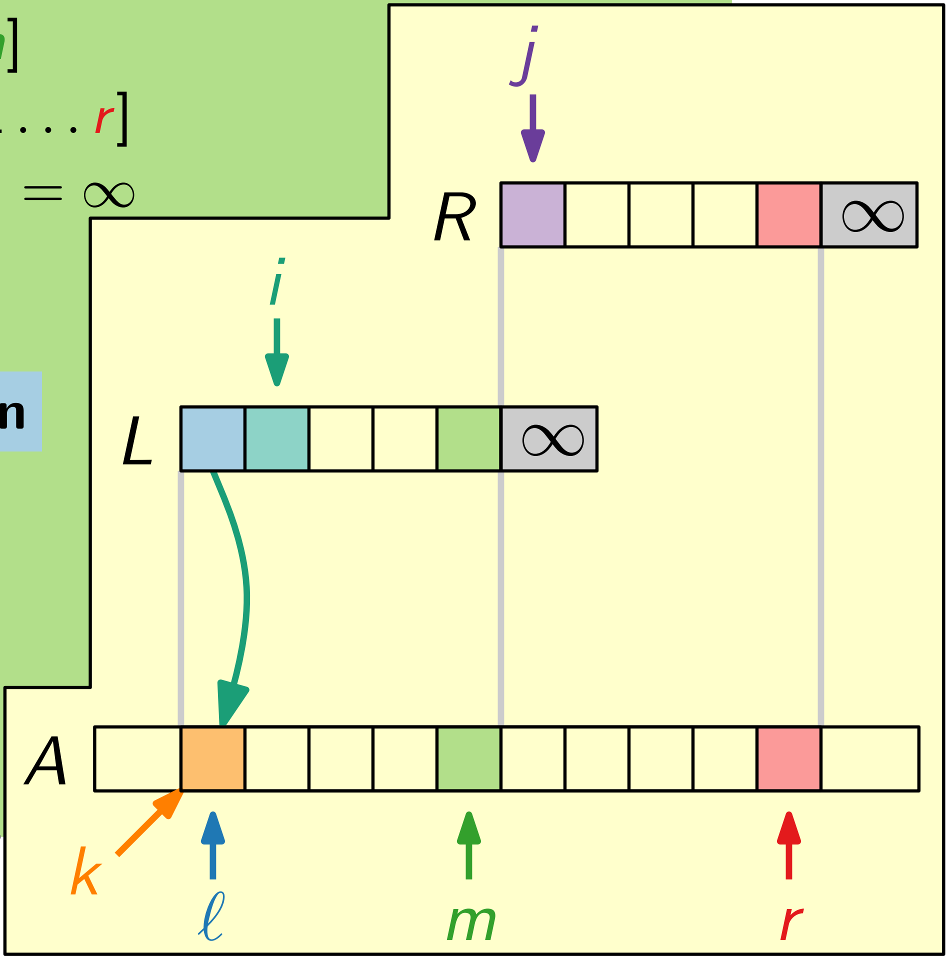
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  dank Invariante
  - $A[\ell \dots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k

Korrektheit von Merge

… nach Schema „F“!

0. Schleifeninvariante

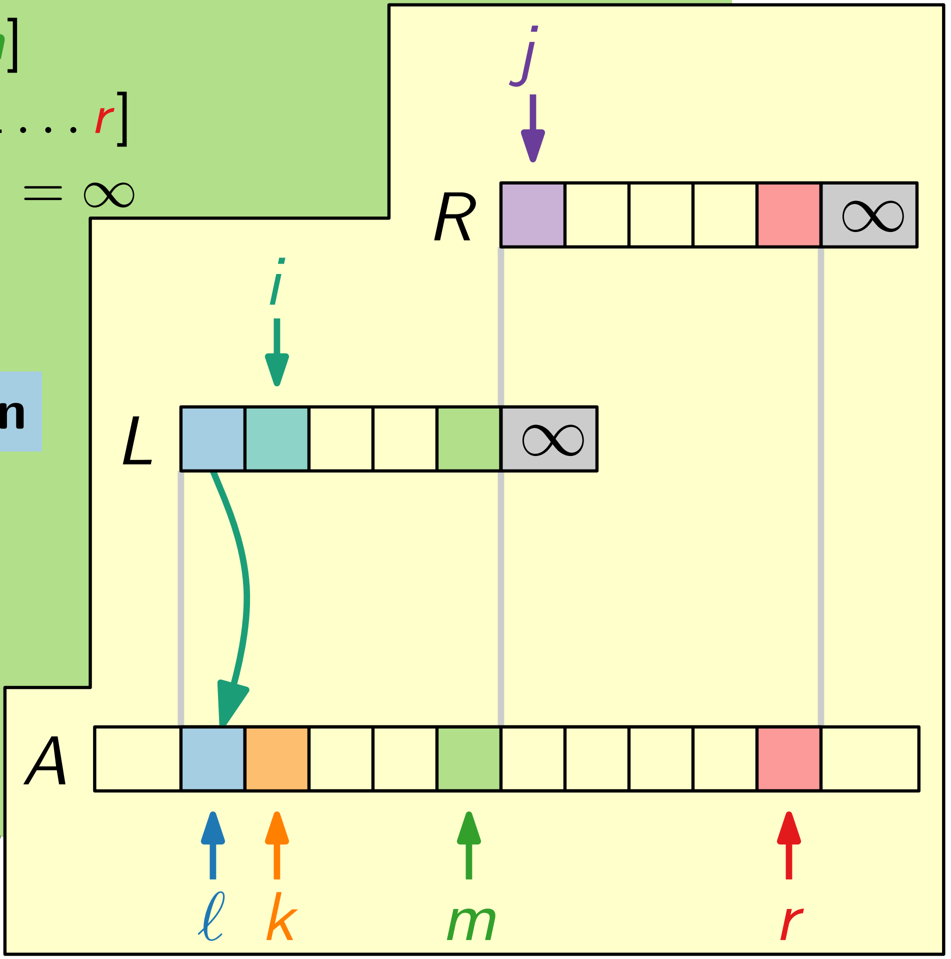
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1;\ \ n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k

Korrektheit von Merge

… nach Schema „F“!

0. Schleifeninvariante

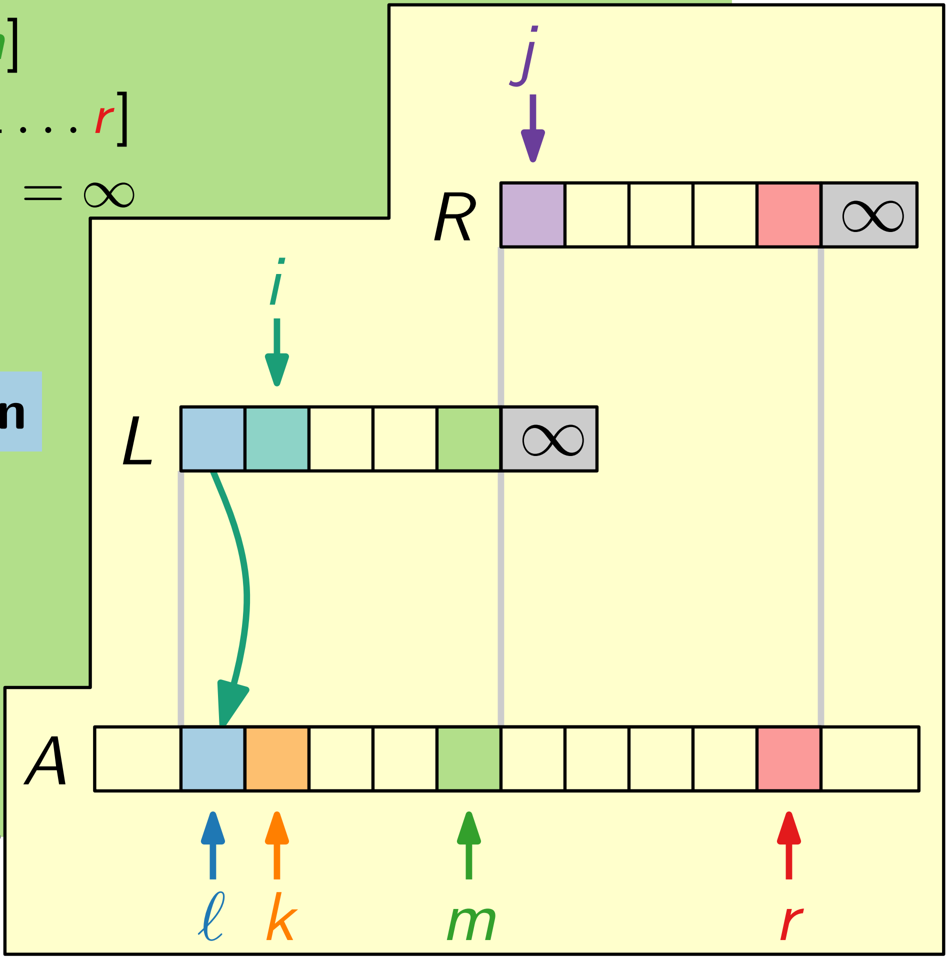
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k $\Rightarrow$

Korrektheit von MERGE

... nach Schema „F“!

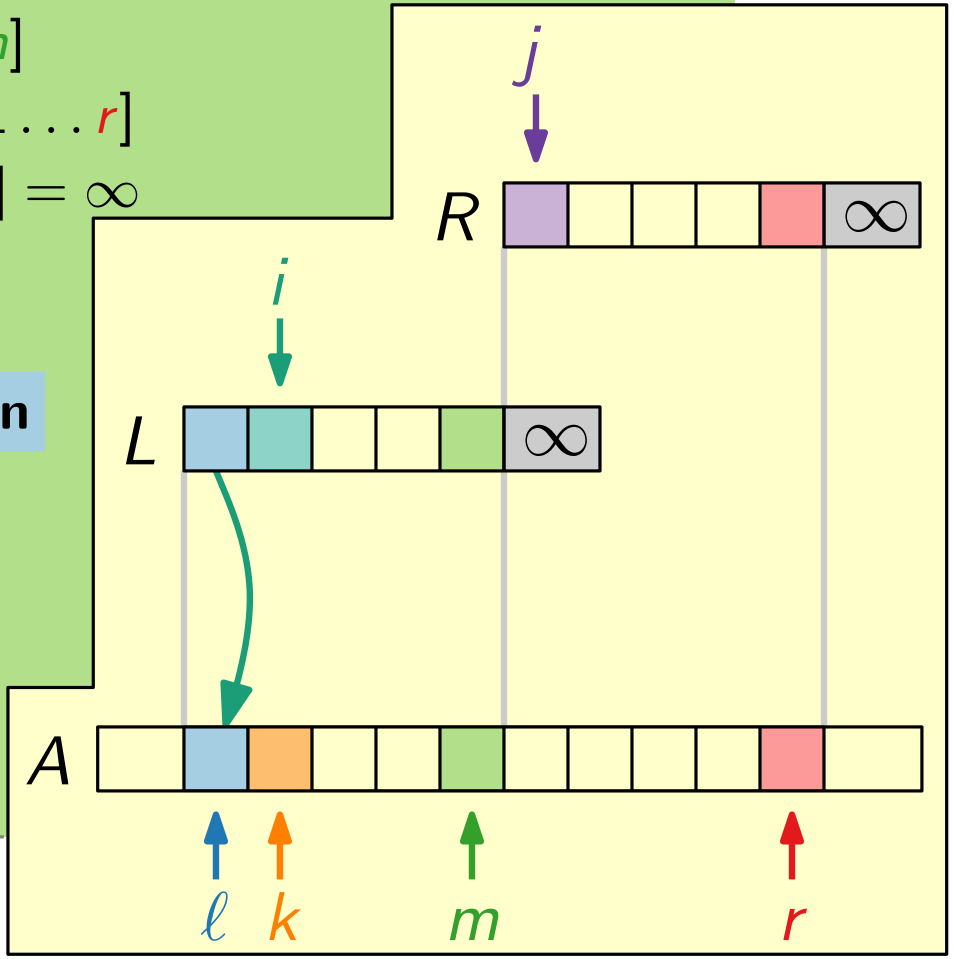
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k-\ell$ Elemente sortiert.

Korrektheit von MERGE

. . . nach Schema „F"!

0. Schleifeninvariante

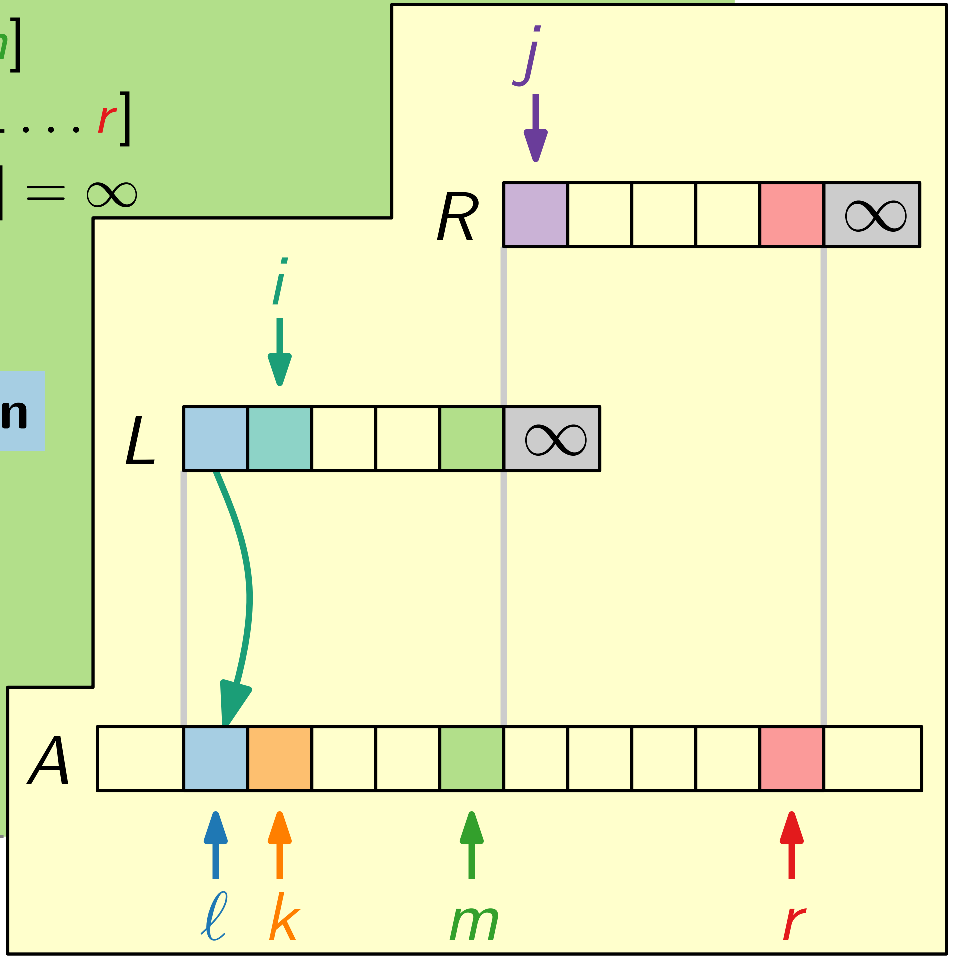
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

MERGE(int[] A, int ℓ, int m, int r)

- $n_1 = m - \ell + 1$; $n_2 = r - m$
- $L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
- $L[1 \dots n_1] = A[\ell \dots m]$
- $R[1 \dots n_2] = A[m+1 \dots r]$
- $L[n_1 + 1] = R[n_2 + 1] = \infty$
- $i = j = 1$
- **for** $k = \ell$ **to** r **do**
  - **if** $L[i] \leq R[j]$ **then**
    - $A[k] = L[i]$
    - $i = i + 1$
  - **else**
    - $A[k] = R[j]$
    - $j = j + 1$



- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt: dank Invariante
  - $A[\ell \dots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.
erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k - \ell$ Elemente sortiert.
$\Rightarrow$

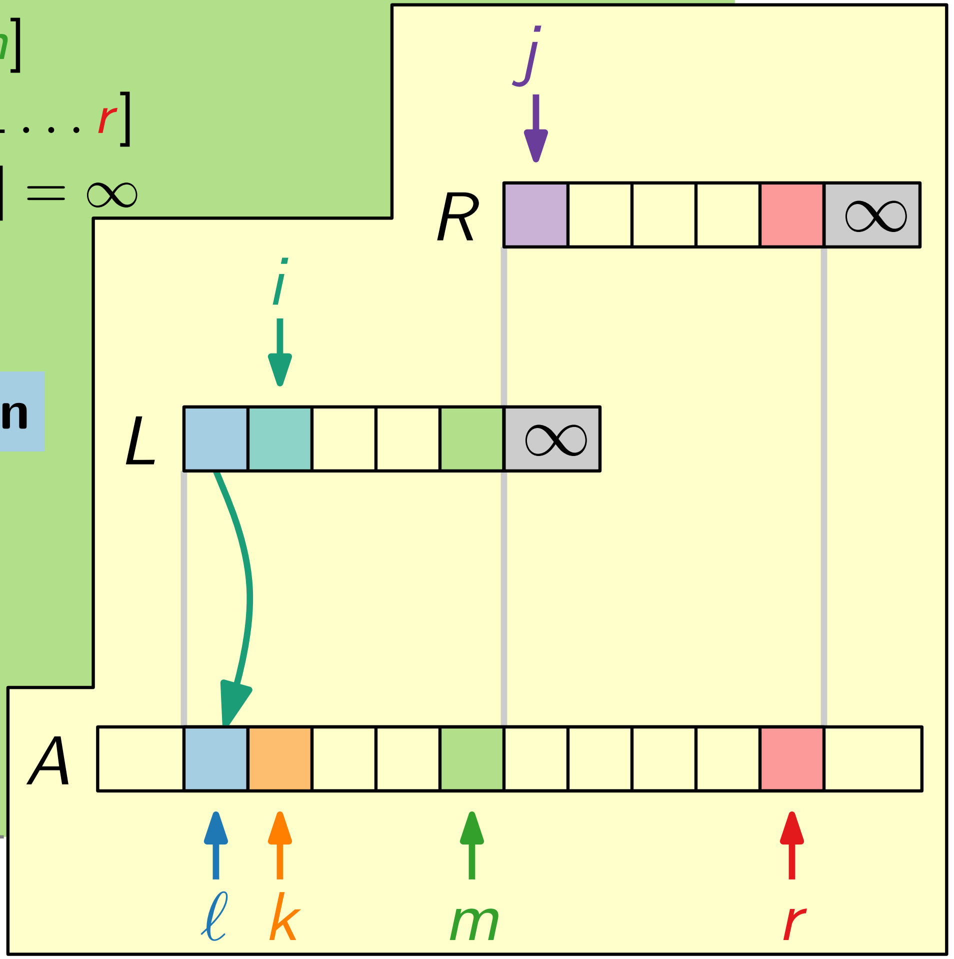
Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



1. Initialisierung ✓

2. Aufrechterhaltung

- Zwei Fälle: (a) $L[i] \leq R[j]$, (b) $R[j] < L[i]$.
- Nun gilt:
  dank Invariante
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k-\ell$ Elemente sortiert.

$\Rightarrow$ Invariante!

Korrektheit von MERGE

… nach Schema „F“!

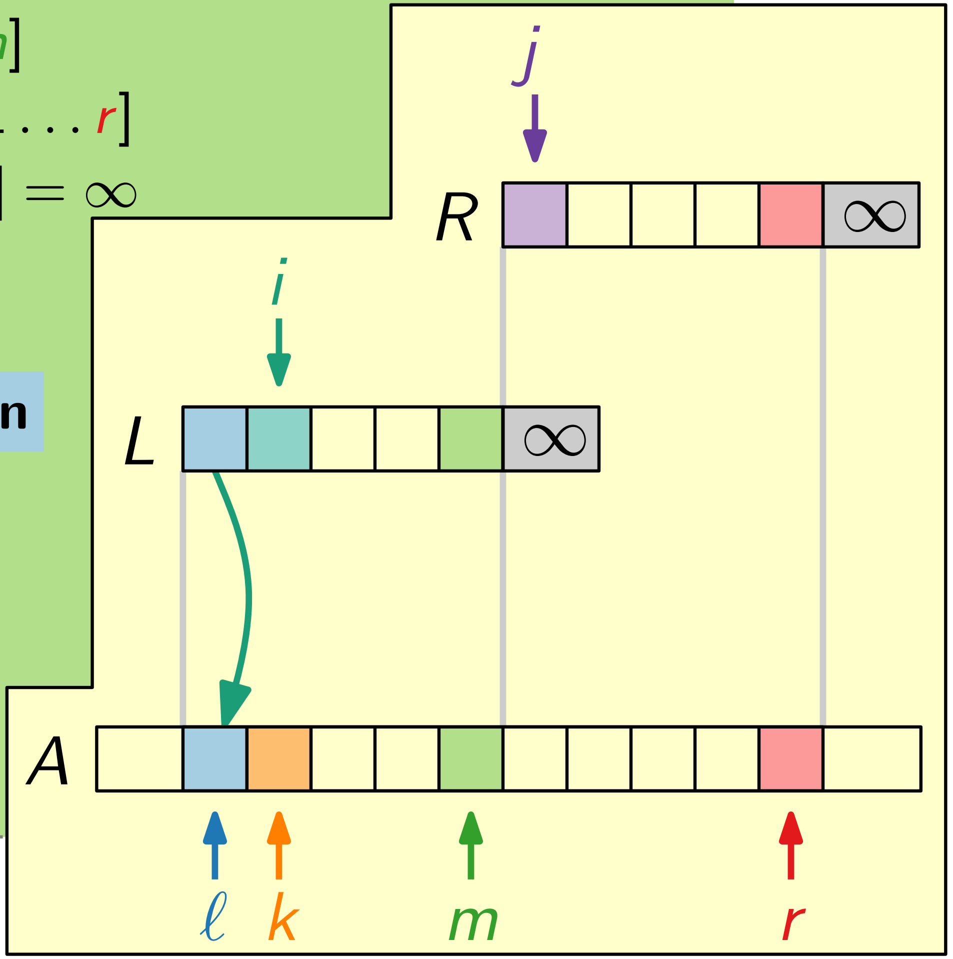
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



- Zwei Fälle: (a) $L[i] \le R[j]$, ✓ (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k-\ell$ Elemente sortiert.

$\Rightarrow$ Invariante!

Korrektheit von MERGE

. . . nach Schema „F“!

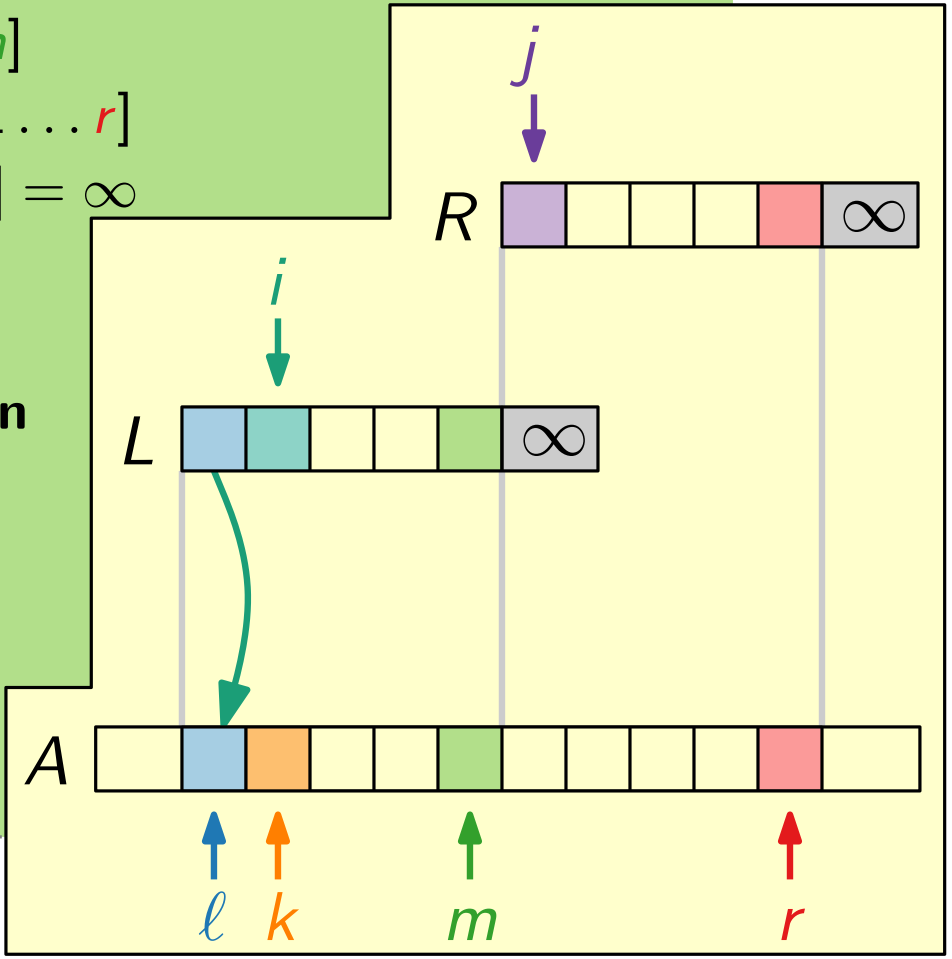
0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



- Zwei Fälle: (a) $L[i] \le R[j]$, ✓ (b) $R[j] < L[i]$.
- Nun gilt:
  dank Invariante
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k-\ell$ Elemente sortiert.

$\Rightarrow$ Invariante!

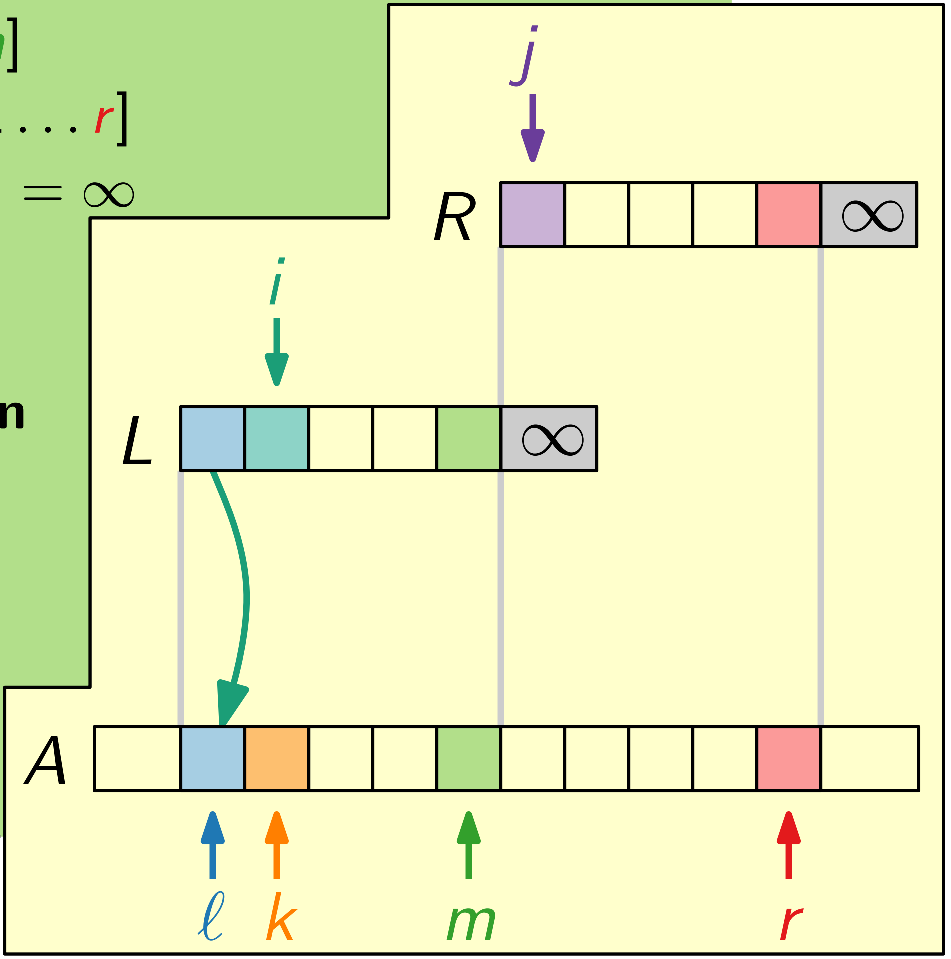
Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



Fall (b) symmetrisch.

1. Initialisierung ✓

2. Aufrechterhaltung

- Zwei Fälle: (a) $L[i] \le R[j]$, ✓ (b) $R[j] < L[i]$.
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.

erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k-\ell$ Elemente sortiert.

$\Rightarrow$ Invariante!

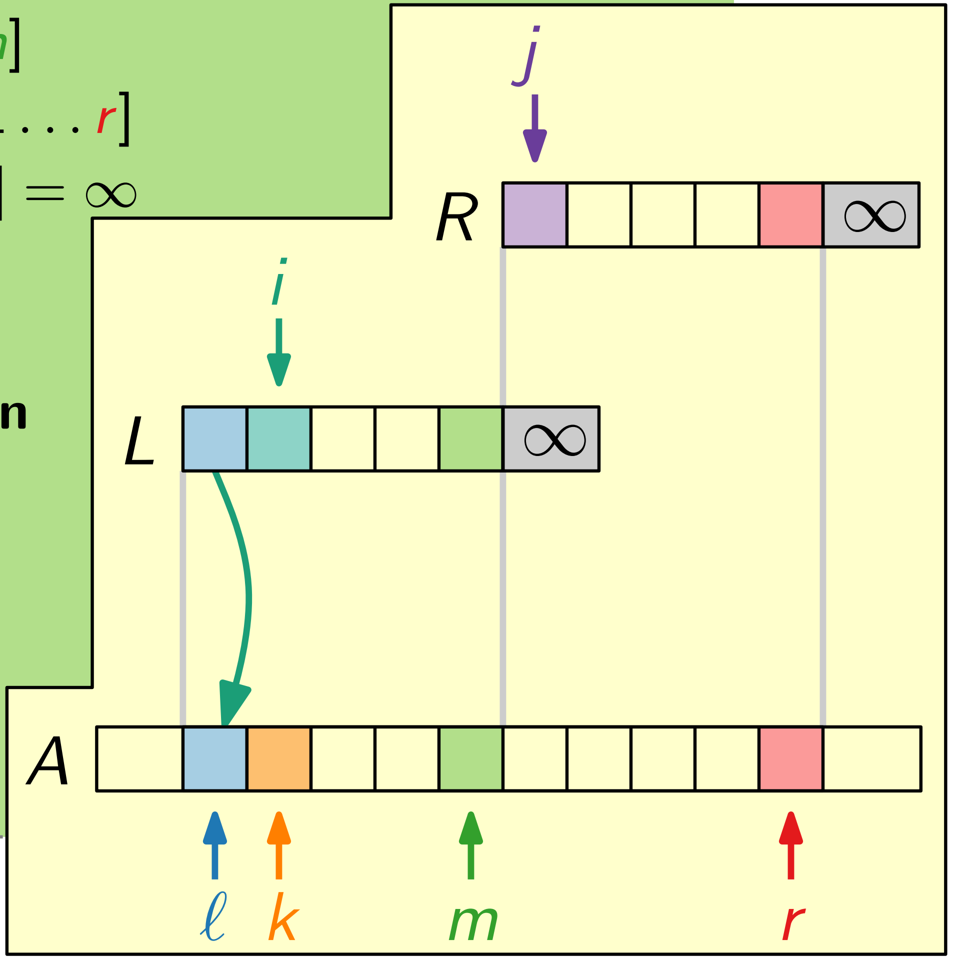
Korrektheit von Merge

… nach Schema „F"!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



Fall (b) symmetrisch.

1. Initialisierung ✓

2. Aufrechterhaltung

- Zwei Fälle: (a) $L[i] \le R[j]$, ✓ (b) $R[j] < L[i]$. ✓
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k-\ell+1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.
erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k-\ell$ Elemente sortiert.

$\Rightarrow$ Invariante!

Korrektheit von MERGE

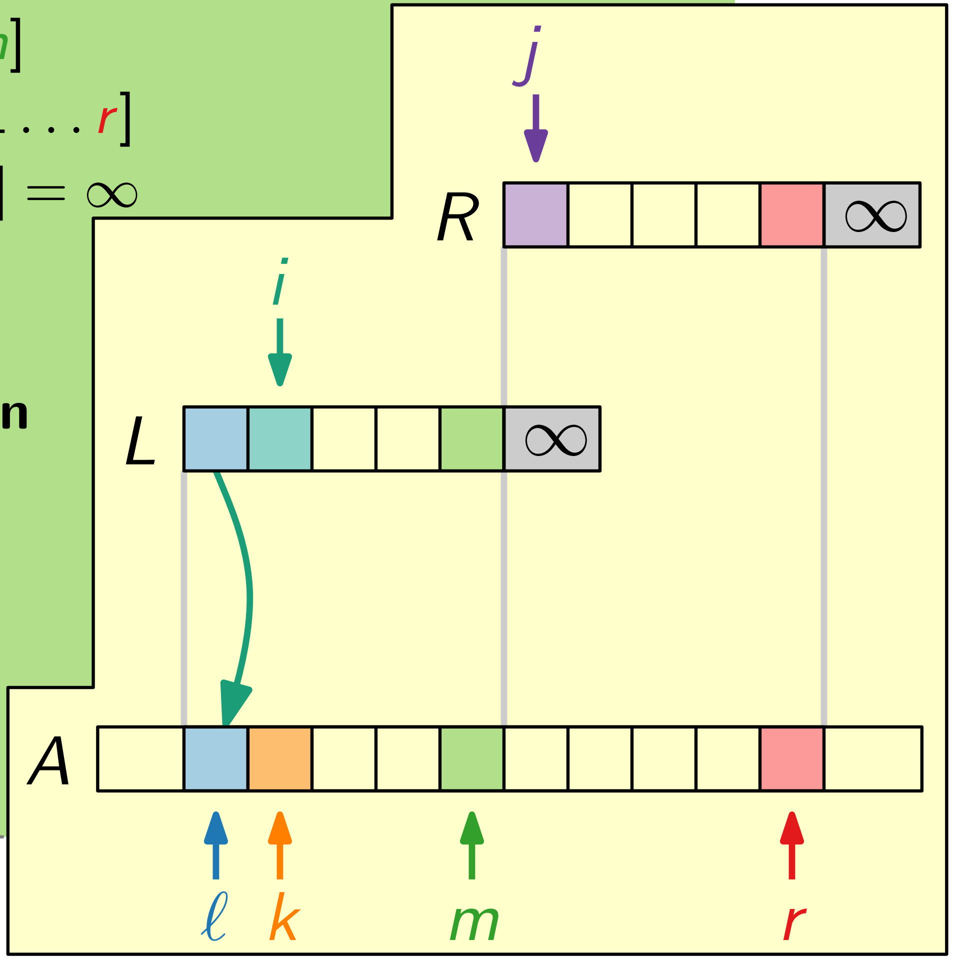
… nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1;\ \ n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Fall (b) symmetrisch.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

- Zwei Fälle: (a) $L[i] \leq R[j]$, ✓ (b) $R[j] < L[i]$. ✓
- Nun gilt:
  - $A[\ell \dots k]$ enthält die kleinsten $k - \ell + 1$ Elemente sortiert.
  - $L[i+1]$ ist kleinstes noch nicht kopiertes Element in L.

dank Invariante

erhöhe i $\Rightarrow$ $L[i]$ ist kleinstes noch nicht kopiertes Element in L.
erhöhe k $\Rightarrow$ $A[\ell \dots k-1]$ enthält die kleinsten $k-\ell$ Elemente sortiert.

$\Rightarrow$ Invariante!

Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

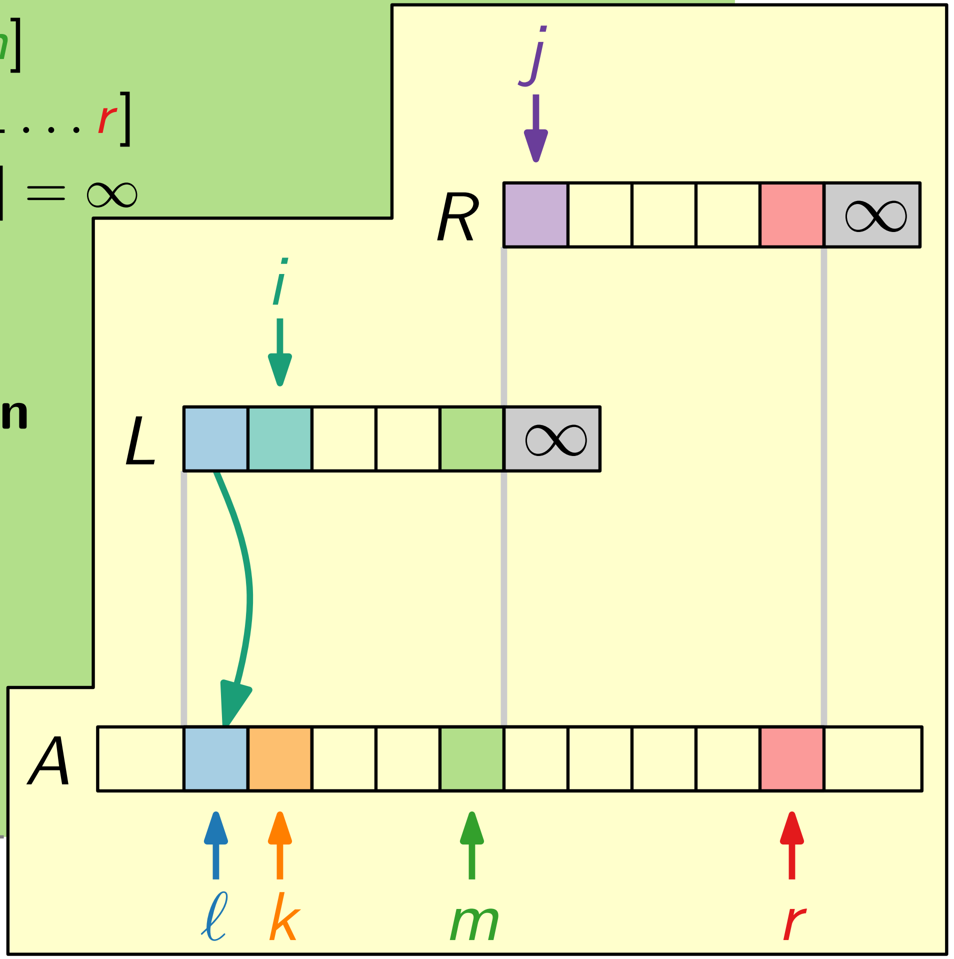
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

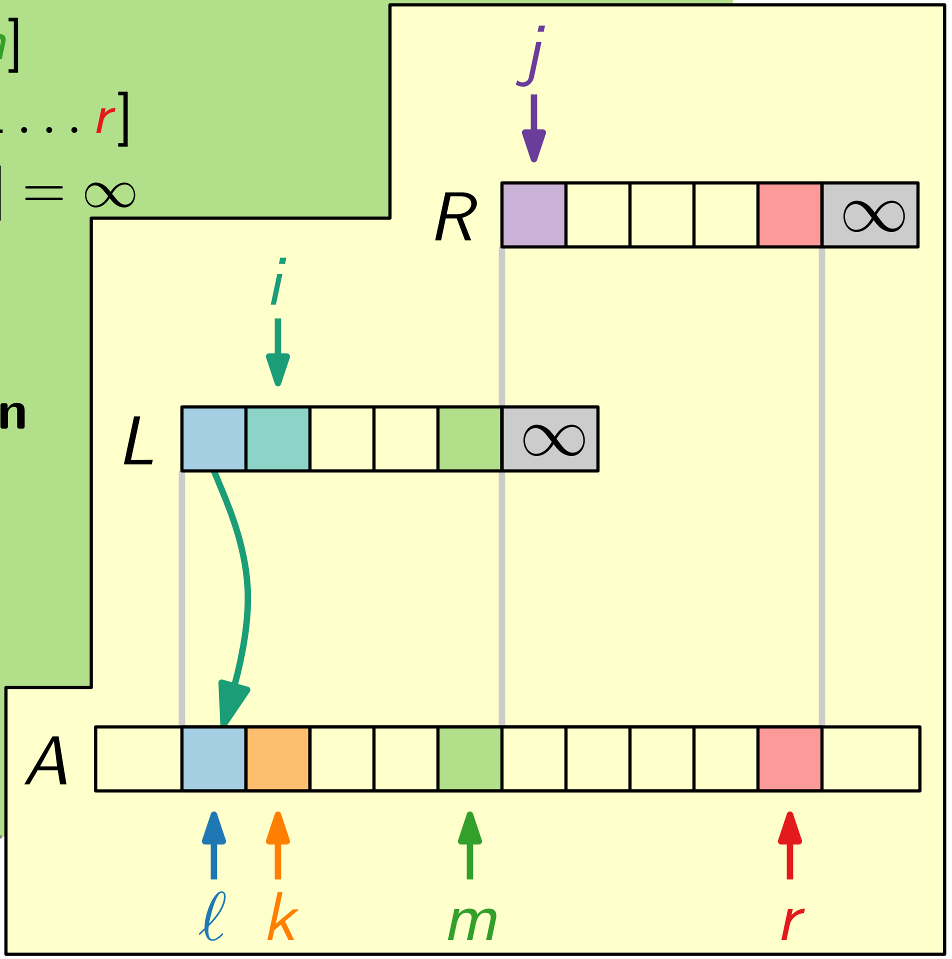
- $A[\ell \dots k-1]$ enthält die $k - \ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

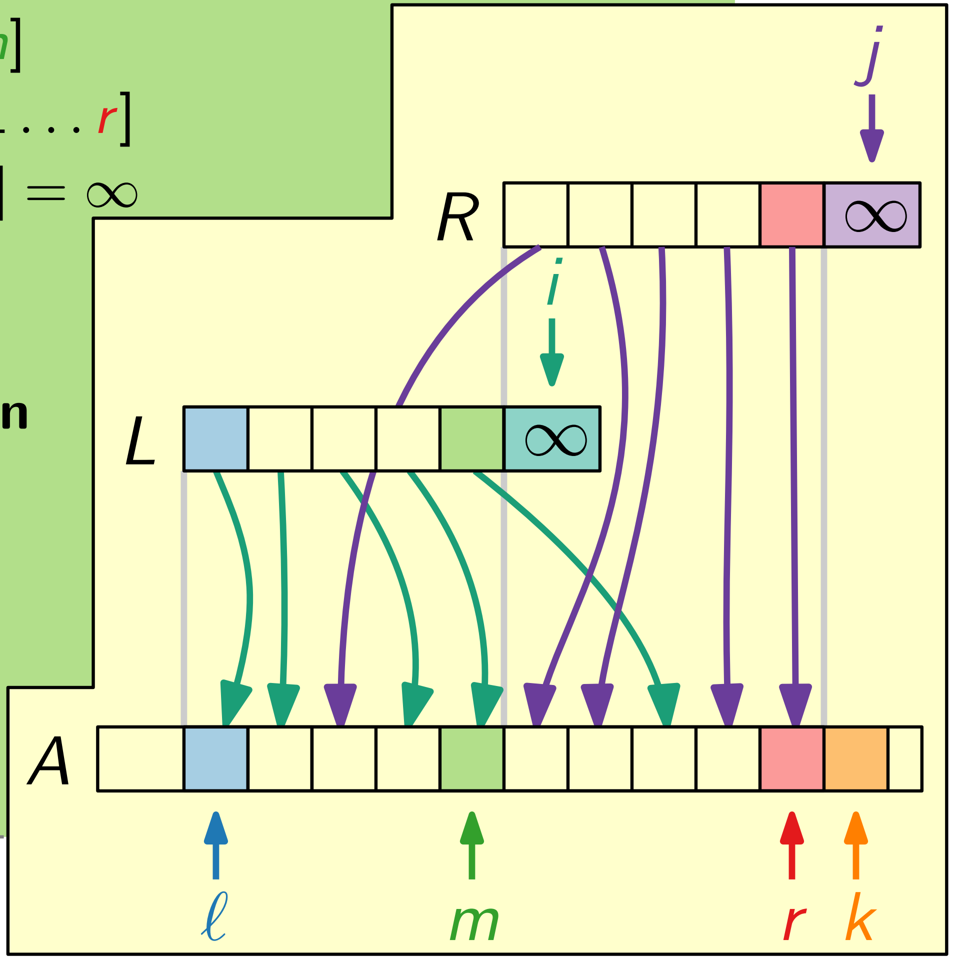
1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \ldots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

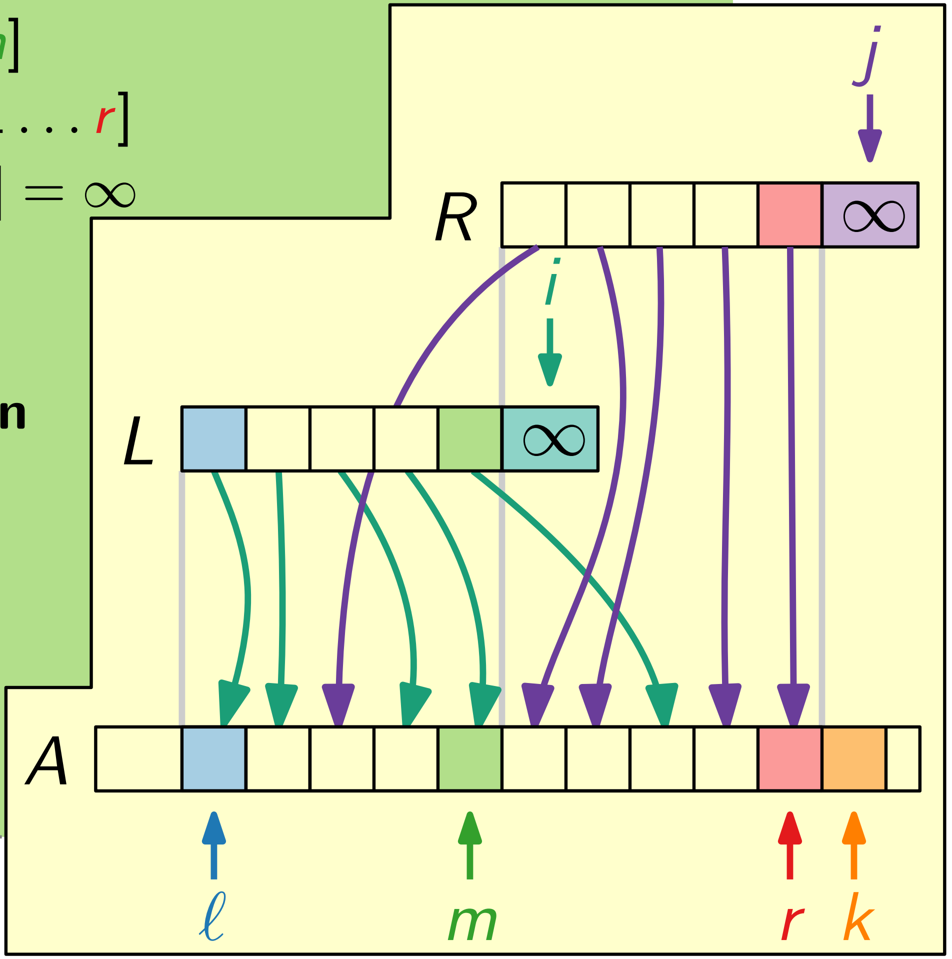
1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

- Nach Abbruch der for-Schleife gilt $k = r + 1$.

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

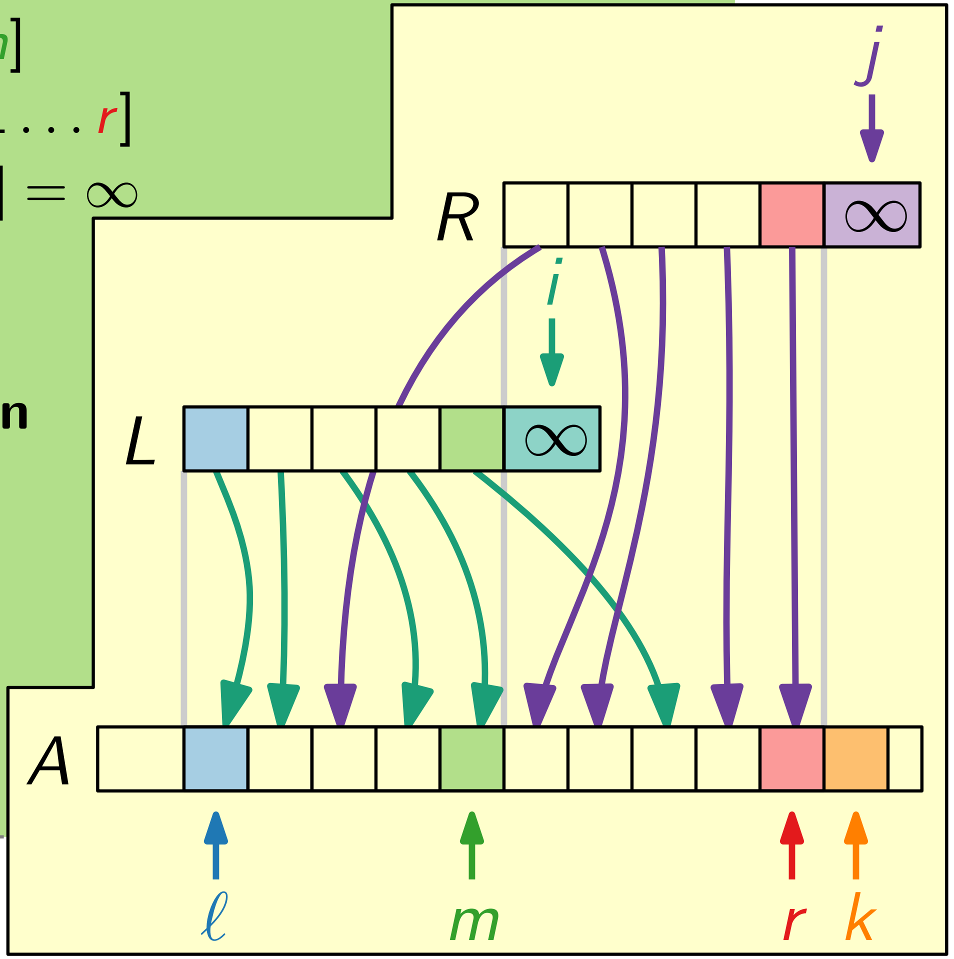
3. Terminierung

- Nach Abbruch der for-Schleife gilt $k = r + 1$.

$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r - \ell + 1$ kleinsten Elemente von $L \cup R$ sortiert.

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m + 1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

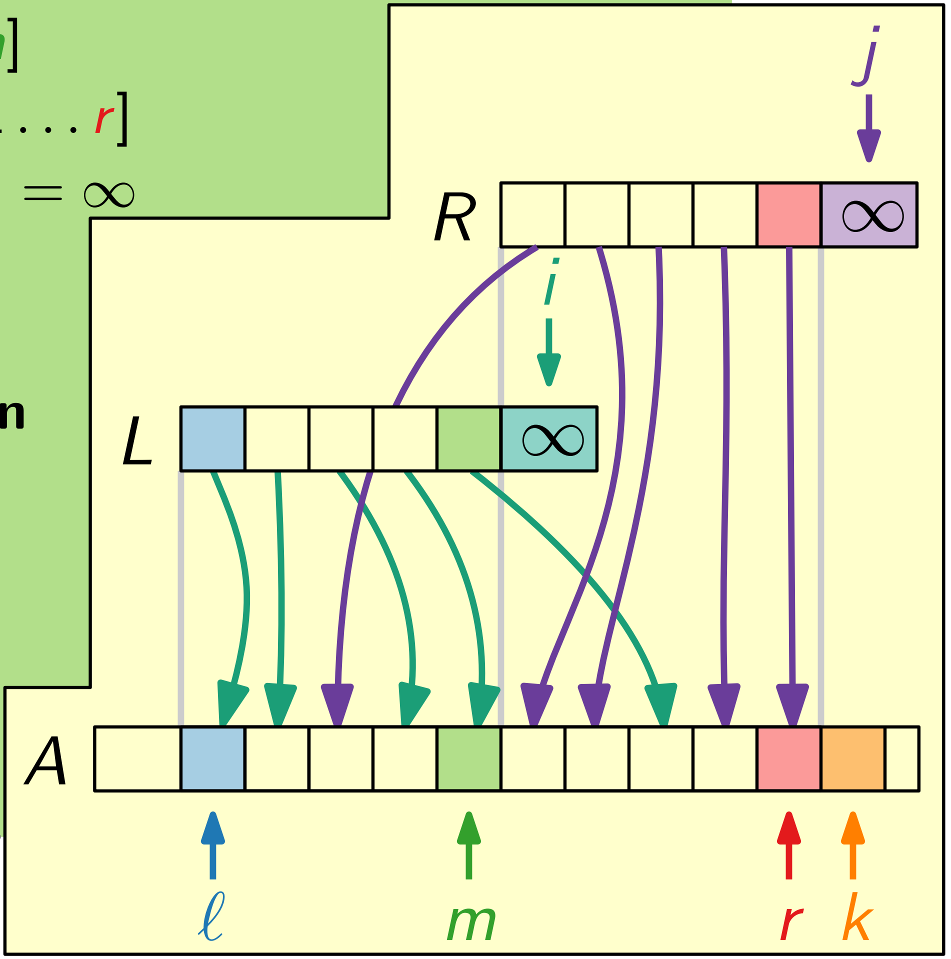
- Nach Abbruch der for-Schleife gilt $k = r + 1$.

$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r - \ell + 1$ kleinsten Elemente von $L \cup R$ sortiert.

- $|L \cup R| = n_1 + n_2 + 2$

Merge(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m + 1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von MERGE

… nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

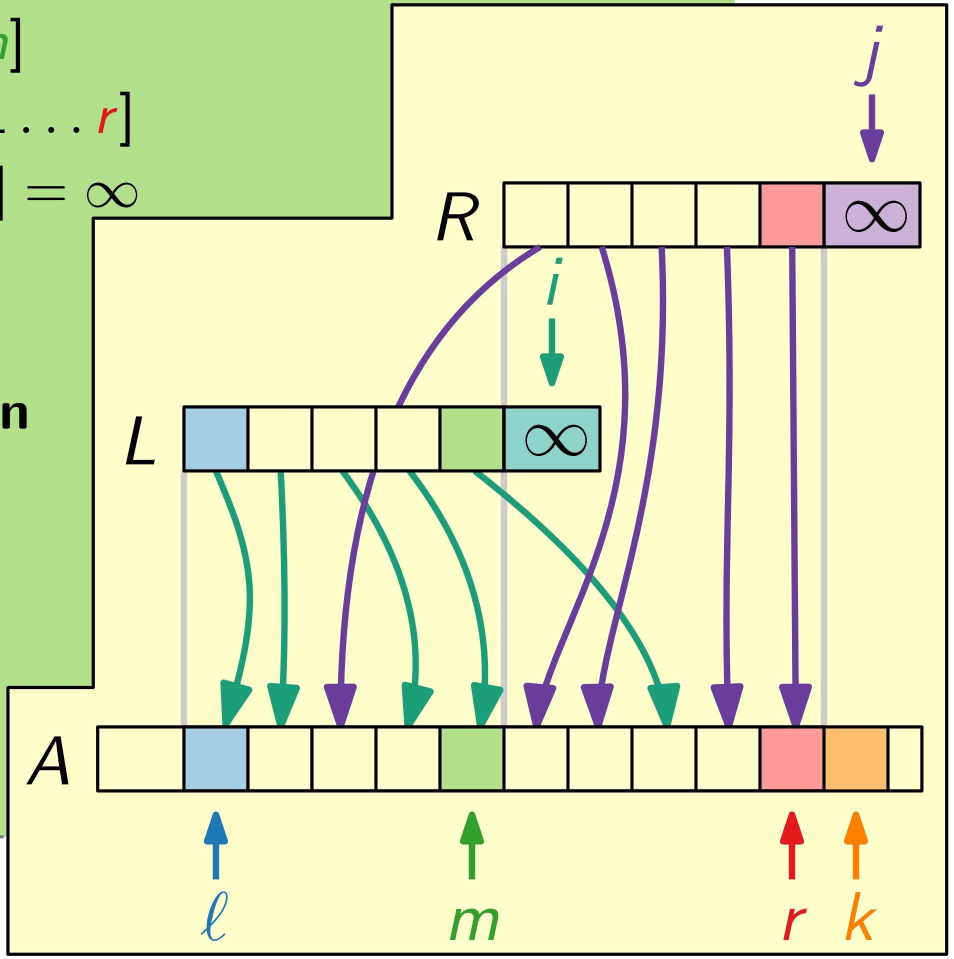
- Nach Abbruch der for-Schleife gilt $k = r + 1$.

$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r - \ell + 1$ kleinsten Elemente von $L \cup R$ sortiert.

- $|L \cup R| = n_1 + n_2 + 2 = r - \ell + 3$

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m + 1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

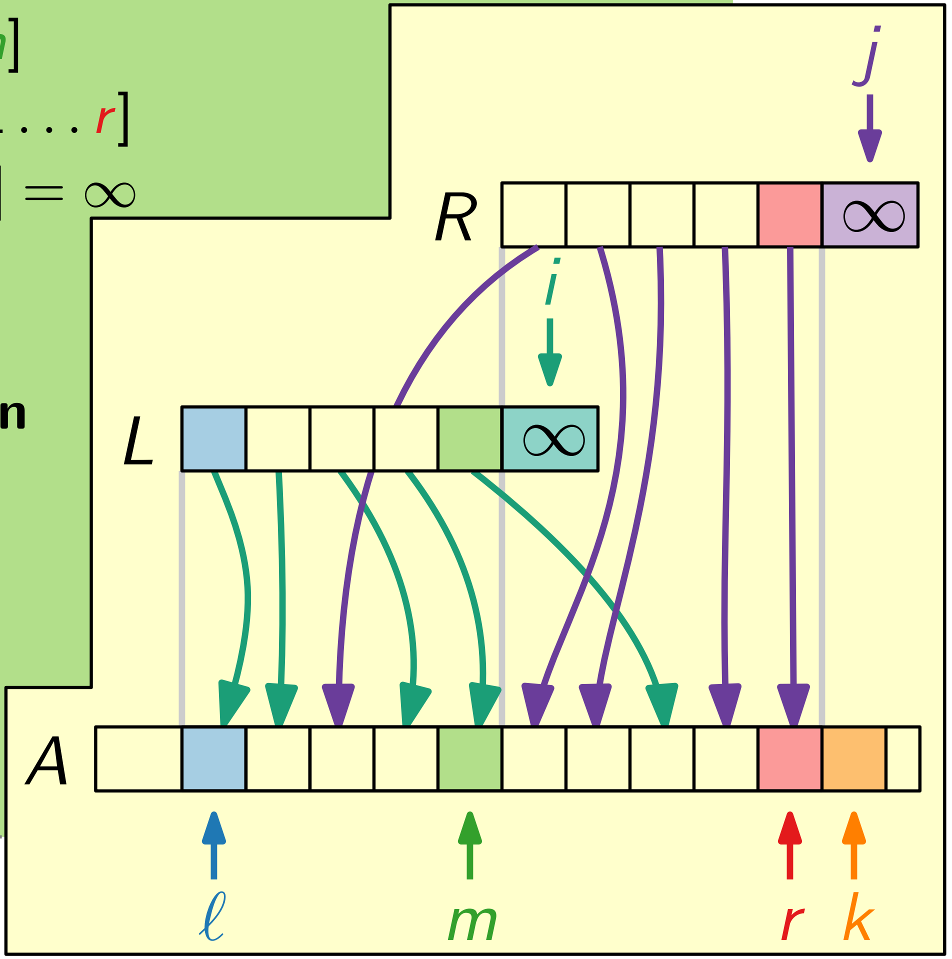
3. Terminierung

- Nach Abbruch der for-Schleife gilt $k = r + 1$.

$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r - \ell + 1$ kleinsten Elemente von $L \cup R$ sortiert.

- $|L \cup R| = n_1 + n_2 + 2 = r - \ell + 3$

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 . . . n1 + 1]; R = new int[1 . . . n2 + 1]
  L[1 . . . n1] = A[ℓ . . . m]
  R[1 . . . n2] = A[m + 1 . . . r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```



Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

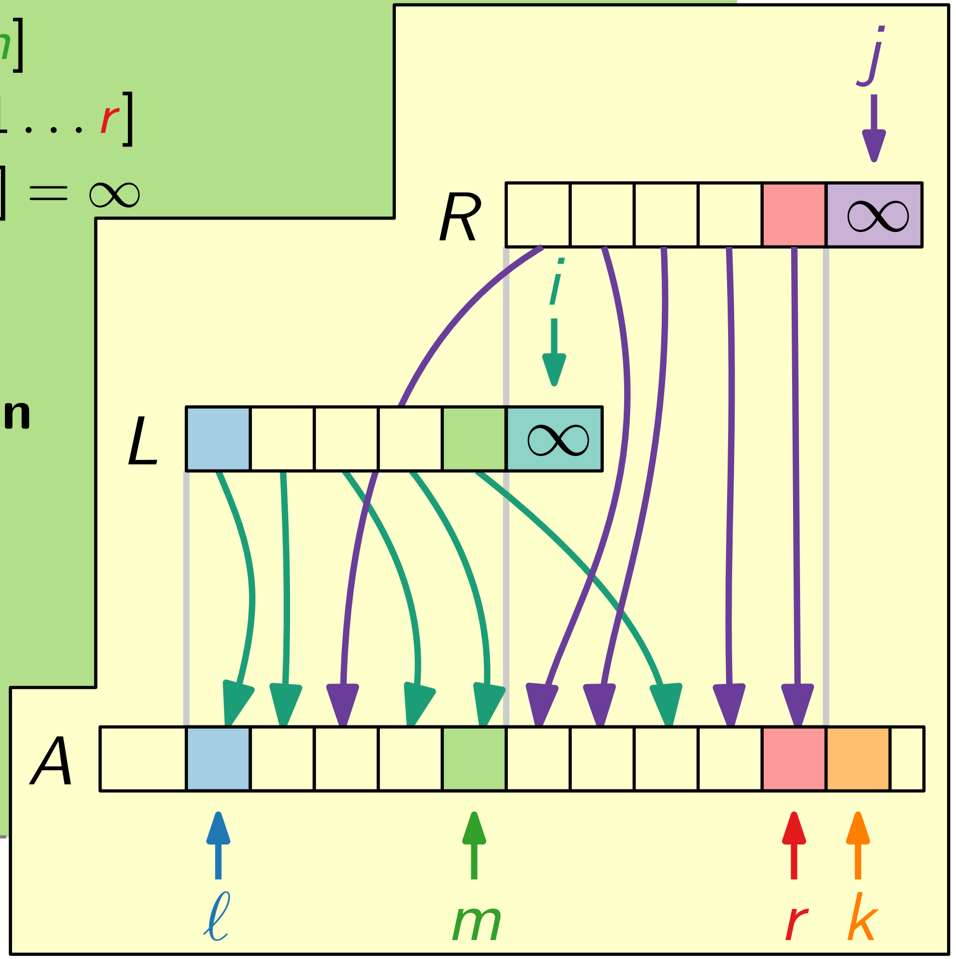
- Nach Abbruch der for-Schleife gilt $k = r+1$.

$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r-\ell+1$ kleinsten Elemente von $L \cup R$ sortiert.

- $|L \cup R| = n_1 + n_2 + 2 = r - \ell + 3$

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

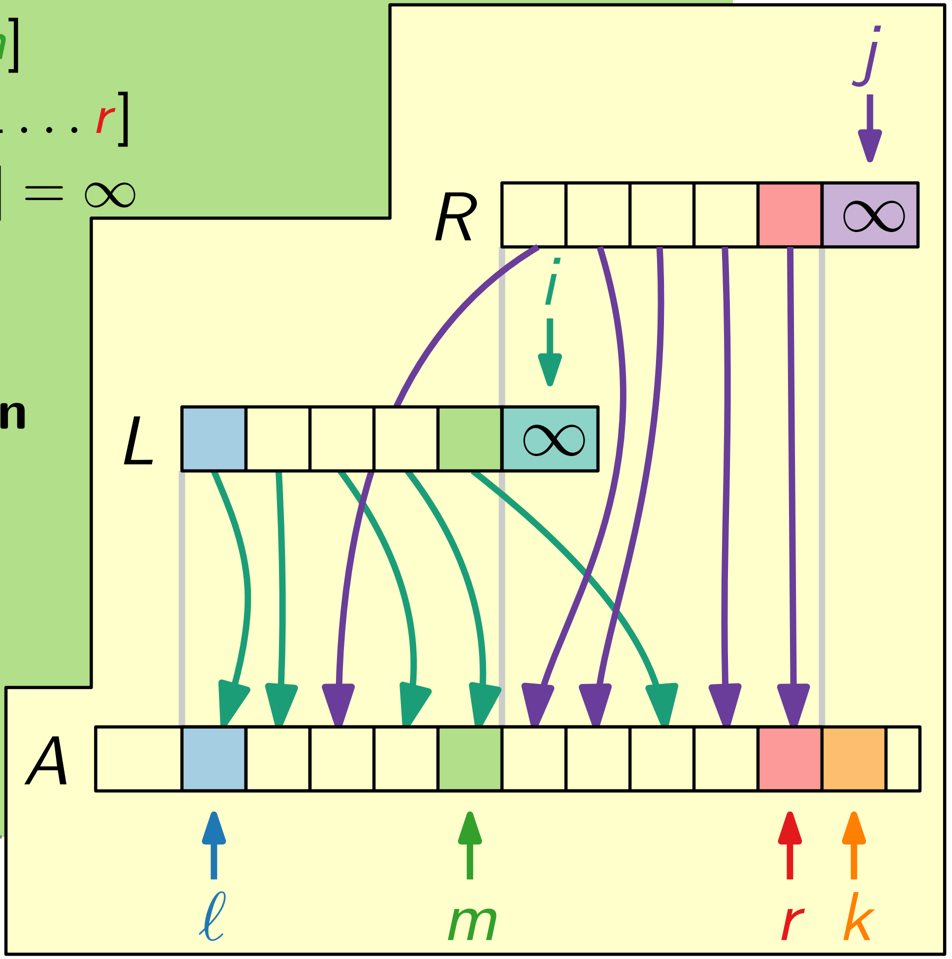
- Nach Abbruch der for-Schleife gilt $k = r + 1$.

$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r - \ell + 1$ kleinsten Elemente von $L \cup R$ sortiert.

+2 Stopper

- $|L \cup R| = n_1 + n_2 + 2 = r - \ell + 3$

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung

- Nach Abbruch der for-Schleife gilt $k = r + 1$.

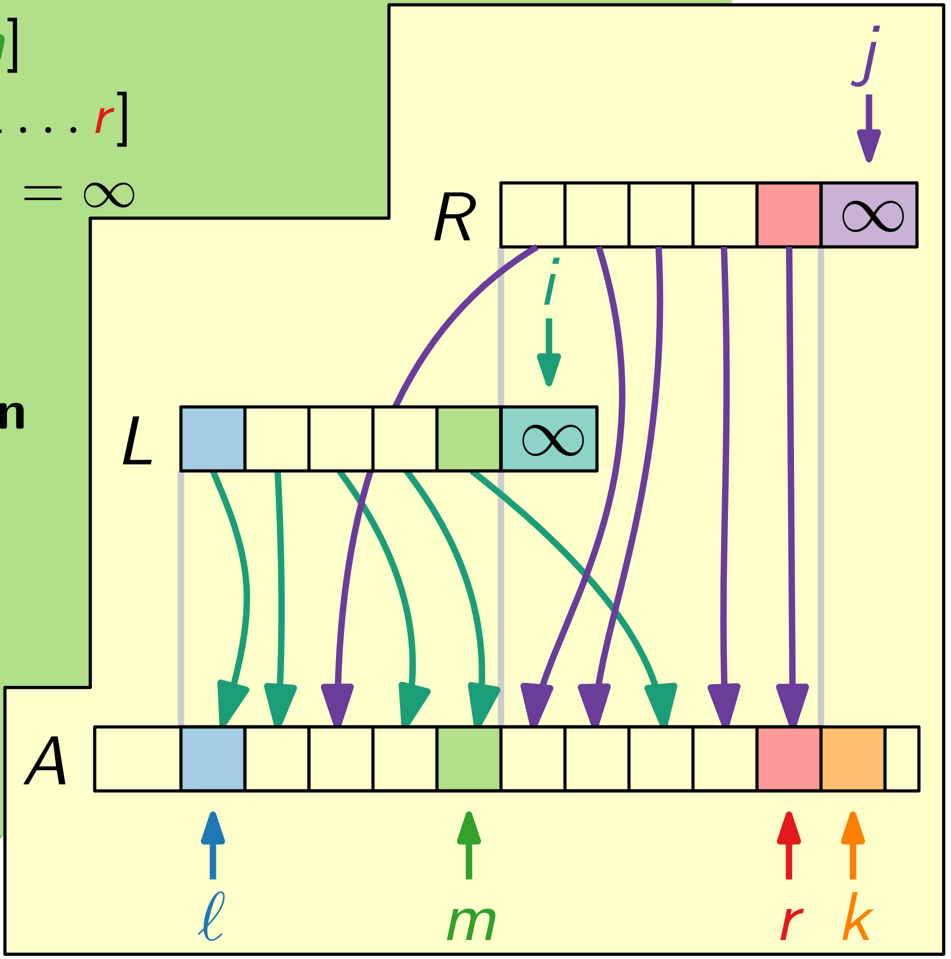
$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r - \ell + 1$ kleinsten Elemente von $L \cup R$ sortiert.

+2 Stopper

- $|L \cup R| = n_1 + n_2 + 2 = r - \ell + 3$, d.h. $A[\ell \dots r]$ korrekt sortiert

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$
$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$
$L[1 \dots n_1] = A[\ell \dots m]$
$R[1 \dots n_2] = A[m+1 \dots r]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$
$i = j = 1$
for $k = \ell$ **to** r **do**
  if $L[i] \le R[j]$ **then**
    $A[k] = L[i]$
    $i = i + 1$
  else
    $A[k] = R[j]$
    $j = j + 1$



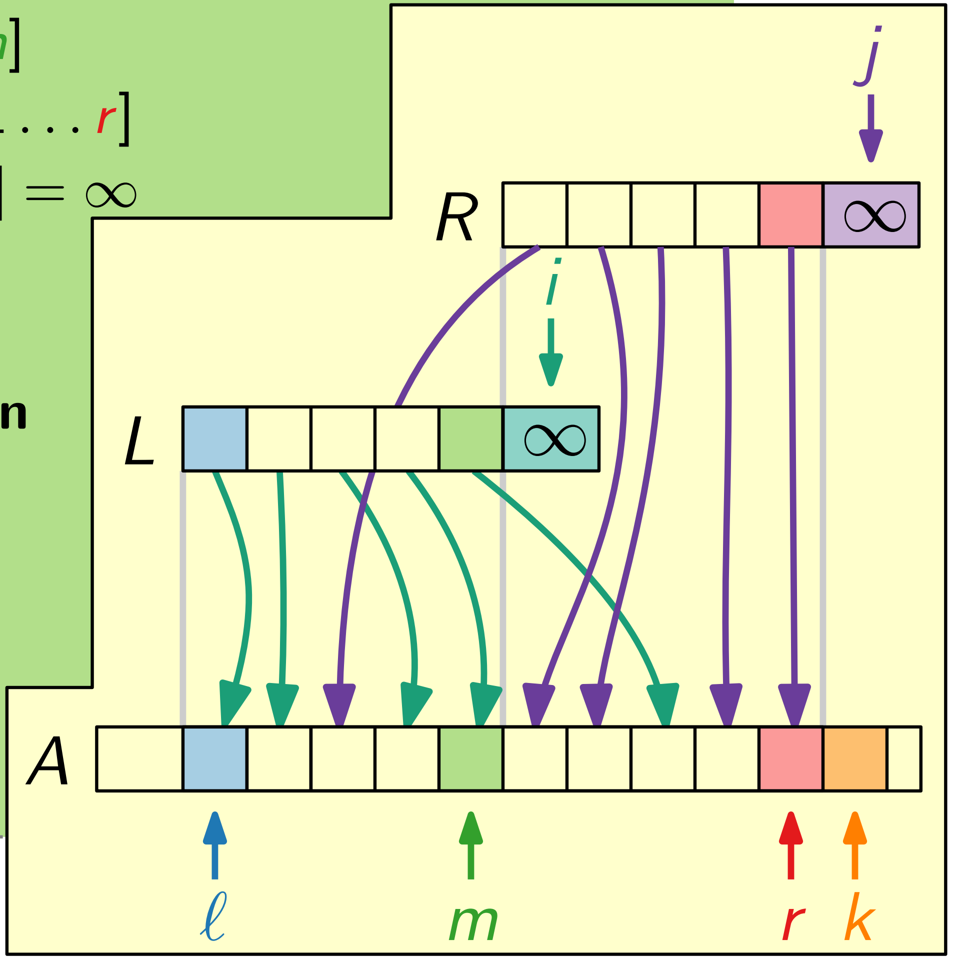
Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```



1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

- Nach Abbruch der for-Schleife gilt $k = r + 1$.

$\Rightarrow A[\ell \dots k-1] = A[\ell \dots r]$ enthält die $r - \ell + 1$ kleinsten Elemente von $L \cup R$ sortiert.

+2 Stopper

- $|L \cup R| = n_1 + n_2 + 2 = r - \ell + 3$, d.h. $A[\ell \dots r]$ korrekt sortiert

Korrektheit von Merge

… nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist Merge korrekt! □

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist MERGE korrekt! □

Laufzeit?

MERGE(int[] A, int ℓ, int m, int r)

```
  n₁ = m − ℓ + 1;  n₂ = r − m
  L = new int[1 … n₁ + 1]; R = new int[1 … n₂ + 1]
  L[1 … n₁] = A[ℓ … m]
  R[1 … n₂] = A[m + 1 … r]
  L[n₁ + 1] = R[n₂ + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist MERGE korrekt! □

Laufzeit?

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist MERGE korrekt!

Laufzeit?

MERGE(int[] A, int ℓ, int m, int r)

$n_1 = m - \ell + 1$; $n_2 = r - m$

$L =$ **new** int$[1 \dots n_1 + 1]$; $R =$ **new** int$[1 \dots n_2 + 1]$

$L[1 \dots n_1] = A[\ell \dots m]$

$R[1 \dots n_2] = A[m+1 \dots r]$

$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for $k = \ell$ **to** r **do**

  if $L[i] \leq R[j]$ **then**

    $A[k] = L[i]$

    $i = i + 1$

  else

    $A[k] = R[j]$

    $j = j + 1$

$\square$

Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist MERGE korrekt! □

Laufzeit? MERGE macht genau Vergleiche.

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
      if L[i] ≤ R[j] then
          A[k] = L[i]
          i = i + 1
      else
          A[k] = R[j]
          j = j + 1
```

Korrektheit von MERGE

... nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist MERGE korrekt! □

```
MERGE(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```

Laufzeit? MERGE macht genau $r - \ell + 1$ Vergleiche.

Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist MERGE korrekt! □

Laufzeit? MERGE macht genau $r-\ell+1$ Vergleiche.

Und MERGESORT?

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```

Korrektheit von MERGE

. . . nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

MERGE(int[] A, int ℓ, int m, int r)

```
n1 = m − ℓ + 1;  n2 = r − m
L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
L[1 ... n1] = A[ℓ ... m]
R[1 ... n2] = A[m + 1 ... r]
L[n1 + 1] = R[n2 + 1] = ∞
i = j = 1
for k = ℓ to r do
    if L[i] ≤ R[j] then
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
```

Also ist MERGE korrekt! $\square$

Laufzeit? MERGE macht genau $r - \ell + 1$ Vergleiche.

Und MERGESORT? Korrekt?

Korrektheit von Merge

… nach Schema „F“!

0. Schleifeninvariante

- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist Merge korrekt! □

```
Merge(int[] A, int ℓ, int m, int r)
  n1 = m − ℓ + 1;  n2 = r − m
  L = new int[1 ... n1 + 1]; R = new int[1 ... n2 + 1]
  L[1 ... n1] = A[ℓ ... m]
  R[1 ... n2] = A[m + 1 ... r]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = ℓ to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```

Laufzeit? Merge macht genau $r - \ell + 1$ Vergleiche.

Und MergeSort? Korrekt? Effizient?

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int[] } A, \text{ int } \ell = 1, \text{ int } r = A.length)$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$

$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

Korrekt?

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m)
  MergeSort($A, m+1, r$) } **herrsche**
  Merge(A, ℓ, m, r) } **kombiniere**

Korrekt? Welche Beweistechnik?

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m) } **herrsche**
  MergeSort($A, m + 1, r$)
  Merge(A, ℓ, m, r) } **kombiniere**

Korrekt? Welche Beweistechnik? Hm, MergeSort ist **rekursiv**...

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
    $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
    MergeSort(A, ℓ, m)
    MergeSort($A, m+1, r$) } **herrsche**
    Merge(A, ℓ, m, r) } **kombiniere**

Korrekt? Welche Beweistechnik? Hm, MergeSort ist **rekursiv**...

Vollständige Induktion über $n = r - \ell + 1$ $(= A[\ell \dots r].length)$:

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$
if $\ell < r$ **then**
$m = \lfloor(\ell + r)/2\rfloor$ } **teile**
$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

Korrekt? Welche Beweistechnik? Hm, MergeSort ist **rekursiv**...

Vollständige Induktion über $n = r - \ell + 1$ $(= A[\ell \dots r].\mathit{length})$:

$n = 1$: **Induktionsanfang**

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m) } **herrsche**
  MergeSort($A, m + 1, r$)
  Merge(A, ℓ, m, r) } **kombiniere**

Korrekt? Welche Beweistechnik? Hm, MergeSort ist **rekursiv**...

Vollständige Induktion über $n = r - \ell + 1$ $(= A[\ell \dots r].length)$:

$n = 1$: **Induktionsanfang**

Dann ist $\ell = r$.

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  $\textsc{MergeSort}(A, \ell, m)$
  $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
  $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

Korrekt? Welche Beweistechnik? Hm, MergeSort ist **rekursiv**. . .

Vollständige Induktion über $n = r - \ell + 1$ $(= A[\ell \dots r].length)$:

$n = 1$: **Induktionsanfang**

Dann ist $\ell = r$.

$\Rightarrow$ **if**-Block wird nicht betreten.

Korrektheit von MERGESORT

MERGESORT(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MERGESORT(A, ℓ, m)
  MERGESORT($A, m + 1, r$) } **herrsche**
  MERGE(A, ℓ, m, r) } **kombiniere**

Korrekt? Welche Beweistechnik? Hm, MERGESORT ist **rekursiv**. . .

Vollständige Induktion über $n = r - \ell + 1$ $(= A[\ell \dots r].length)$:

$n = 1$: **Induktionsanfang**

Dann ist $\ell = r$.

$\Rightarrow$ **if**-Block wird nicht betreten.

D.h. nichts passiert.

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  $\textsc{MergeSort}(A, \ell, m)$
  $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
  $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

Korrekt? Welche Beweistechnik? Hm, MergeSort ist **rekursiv**...

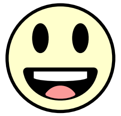
Vollständige Induktion über $n = r - \ell + 1$ $(= A[\ell \dots r].\mathit{length})$:

$n = 1$: **Induktionsanfang**

Dann ist $\ell = r$.

$\Rightarrow$ **if**-Block wird nicht betreten.

D.h. nichts passiert.

OK, da $A[\ell \dots \ell]$ schon sortiert. 

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m) } **herrsche**
  MergeSort($A, m + 1, r$)
  Merge(A, ℓ, m, r) } **kombiniere**

$n > 1$: **Induktionsschritt**

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[\,]\ A,\ \text{int}\ \ell = 1,\ \text{int}\ r = A.length)$

if $\ell < r$ **then**

$$\left.m = \lfloor (\ell + r)/2 \rfloor\right\}\ \textbf{teile}$$

$$\left.\begin{array}{l}\textsc{MergeSort}(A, \ell, m)\\ \textsc{MergeSort}(A, m+1, r)\end{array}\right\}\ \textbf{herrsche}$$

$$\left.\textsc{Merge}(A, \ell, m, r)\right\}\ \textbf{kombiniere}$$

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
MergeSort(A, ℓ, m)
MergeSort($A, m+1, r$) } **herrsche**
Merge(A, ℓ, m, r) } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$.

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

if $\ell < r$ **then**

$m = \lfloor(\ell + r)/2\rfloor$ } **teile**

MergeSort(A, ℓ, m)

MergeSort($A, m + 1, r$) } **herrsche**

Merge(A, ℓ, m, r) } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Korrektheit von MergeSort

```
MergeSort(int[] A, int ℓ = 1, int r = A.length)
if ℓ < r then
    m = ⌊(ℓ + r)/2⌋                 } teile
    MergeSort(A, ℓ, m)              }
    MergeSort(A, m + 1, r)          } herrsche
    Merge(A, ℓ, m, r)               } kombiniere
```

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.\mathit{length})$

if $\ell < r$ **then**

$$
\left.\begin{array}{l} m = \lfloor(\ell + r)/2\rfloor \end{array}\right\} \text{teile}
$$

$$
\left.\begin{array}{l} \textsc{MergeSort}(A, \ell, m) \\ \textsc{MergeSort}(A, m+1, r) \end{array}\right\} \text{herrsche}
$$

$$
\left.\begin{array}{l} \textsc{Merge}(A, \ell, m, r) \end{array}\right\} \text{kombiniere}
$$

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

$\Rightarrow A[\ell \dots m]$ und $A[m+1 \dots r]$ sind **kürzer** als $A[\ell \dots r]$.

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m + 1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

$\Rightarrow A[\ell \ldots m]$ und $A[m + 1 \ldots r]$ sind **kürzer** als $A[\ell \ldots r]$.

$\underset{\boxed{\text{I.A.}}}{\Rightarrow}$

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  MergeSort(A, ℓ, m) } **herrsche**
  MergeSort($A, m + 1, r$)
  Merge(A, ℓ, m, r) } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

$\Rightarrow$ $A[\ell \dots m]$ und $A[m + 1 \dots r]$ sind **kürzer** als $A[\ell \dots r]$.

$\underset{\boxed{\text{I.A.}}}{\Rightarrow}$ MergeSort(A, ℓ, m) ist korrekt und

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[\,]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.length)$
if $\ell < r$ **then**
  $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
  $\textsc{MergeSort}(A, \ell, m)$
  $\textsc{MergeSort}(A, m + 1, r)$ } **herrsche**
  $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

$\Rightarrow A[\ell \dots m]$ und $A[m + 1 \dots r]$ sind **kürzer** als $A[\ell \dots r]$.

$\underset{\text{I.A.}}{\Rightarrow}$ $\textsc{MergeSort}(A, \ell, m)$ ist korrekt und
$\textsc{MergeSort}(A, m + 1, r)$ ist korrekt.

Korrektheit von $\textsc{MergeSort}$

$\textsc{MergeSort}$(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
- $m = \lfloor (\ell + r)/2 \rfloor$ } **teile**
- $\textsc{MergeSort}(A, \ell, m)$
- $\textsc{MergeSort}(A, m+1, r)$ } **herrsche**
- $\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: $\textsc{MergeSort}$ korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \le m < r$.

$\Rightarrow A[\ell \dots m]$ und $A[m+1 \dots r]$ sind **kürzer** als $A[\ell \dots r]$.

$\underset{\text{I.A.}}{\Rightarrow}$ $\textsc{MergeSort}(A, \ell, m)$ ist korrekt und
$\textsc{MergeSort}(A, m+1, r)$ ist korrekt.

Schon bewiesen:

Korrektheit von MergeSort

MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)
if $\ell < r$ **then**
- $m = \lfloor(\ell + r)/2\rfloor$ } **teile**
- MergeSort(A, ℓ, m)
- MergeSort($A, m+1, r$) } **herrsche**
- Merge(A, ℓ, m, r) } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

$\Rightarrow A[\ell \dots m]$ und $A[m+1 \dots r]$ sind **kürzer** als $A[\ell \dots r]$.

$\underset{\text{I.A.}}{\Rightarrow}$ MergeSort(A, ℓ, m) ist korrekt und

MergeSort($A, m+1, r$) ist korrekt.

Schon bewiesen: Merge ist korrekt.

Korrektheit von MergeSort

$\textsc{MergeSort}(\text{int}[\,]\ A, \text{int}\ \ell = 1, \text{int}\ r = A.length)$

if $\ell < r$ **then**

$m = \lfloor (\ell + r)/2 \rfloor$ } **teile**

$\textsc{MergeSort}(A, \ell, m)$
$\textsc{MergeSort}(A, m+1, r)$ } **herrsche**

$\textsc{Merge}(A, \ell, m, r)$ } **kombiniere**

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

$\Rightarrow A[\ell \dots m]$ und $A[m+1 \dots r]$ sind **kürzer** als $A[\ell \dots r]$.

$\xRightarrow{\text{I.A.}}$ $\textsc{MergeSort}(A, \ell, m)$ ist korrekt und
$\textsc{MergeSort}(A, m+1, r)$ ist korrekt.
Schon bewiesen: Merge ist korrekt.

} $\textsc{MergeSort}(A, \ell, r)$ ist korrekt. □

Übersicht

Techniken für Korrektheitsbeweise

- iterative Algorithmen (à la InsertionSort, Factorial, Merge)

- rekursive Algorithmen (à la MergeSort)

Übersicht

Techniken für Korrektheitsbeweise

- iterative Algorithmen (à la InsertionSort, Factorial, Merge)

  per **Schleifeninvariante** (Schema „F“)

- rekursive Algorithmen (à la MergeSort)

Übersicht

Techniken für Korrektheitsbeweise

- iterative Algorithmen (à la InsertionSort, Factorial, Merge)

  per **Schleifeninvariante** (Schema „F“)

- rekursive Algorithmen (à la MergeSort)

  per **vollständige Induktion**

Übersicht

Techniken für Korrektheitsbeweise

- iterative Algorithmen (à la InsertionSort, Factorial, Merge)

  per **Schleifeninvariante** (Schema „F“)

- rekursive Algorithmen (à la MergeSort)

  per **vollständige Induktion**

Laufzeit?