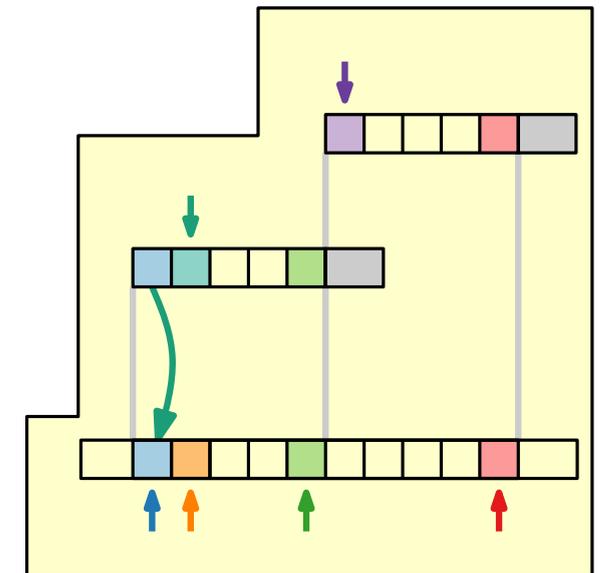
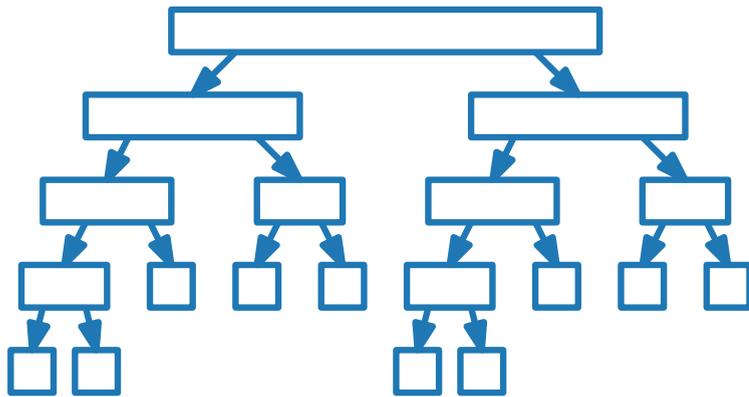
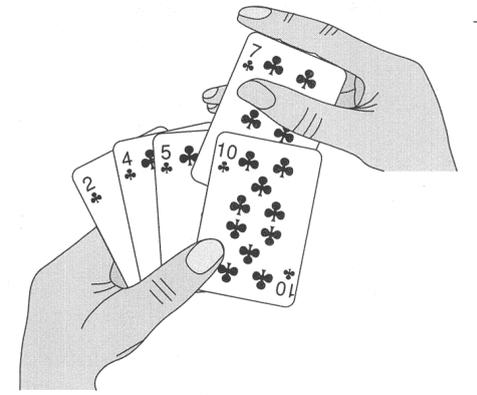
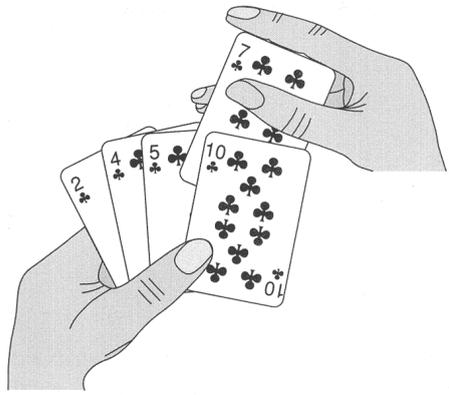


# Algorithmen und Datenstrukturen

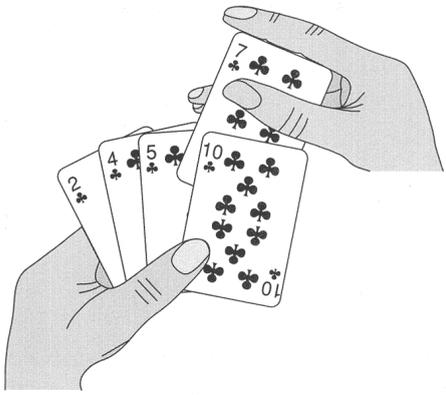
## Vorlesung 2: Teile und Herrsche



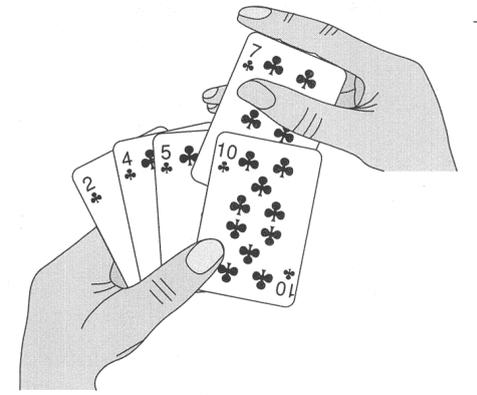
# Teile und Herrsche



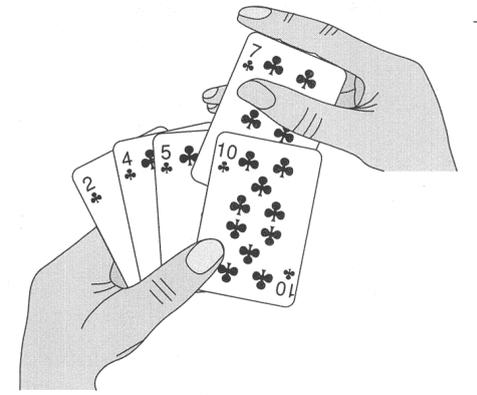
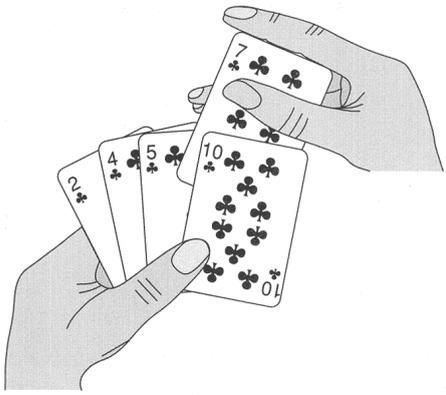
# Teile und Herrsche



Idee.



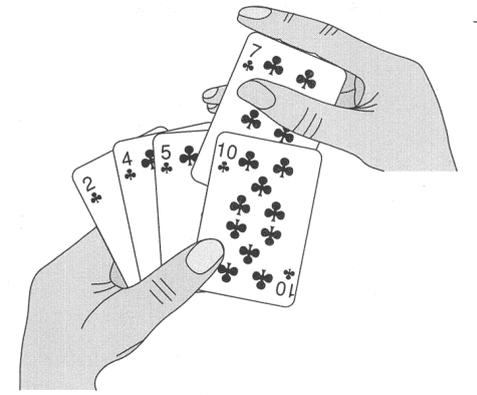
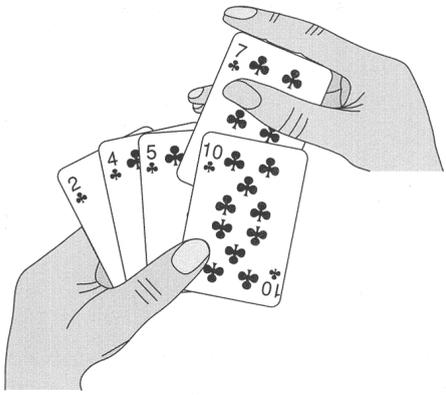
# Teile und Herrsche



## Idee.

- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,

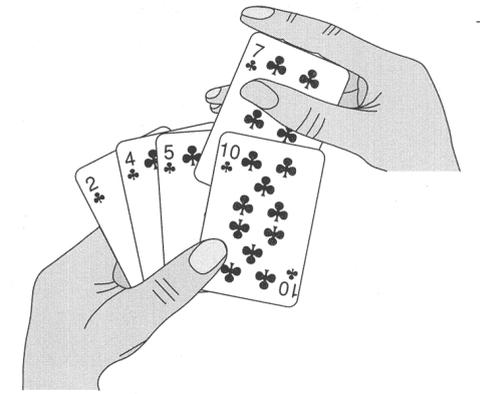
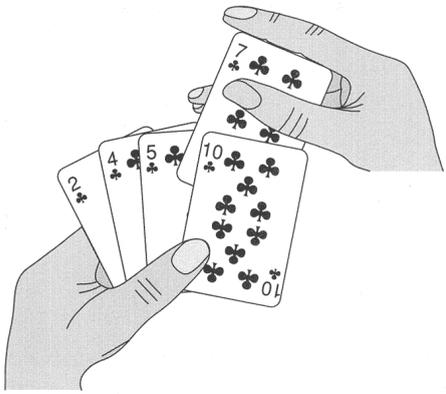
# Teile und Herrsche



## Idee.

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

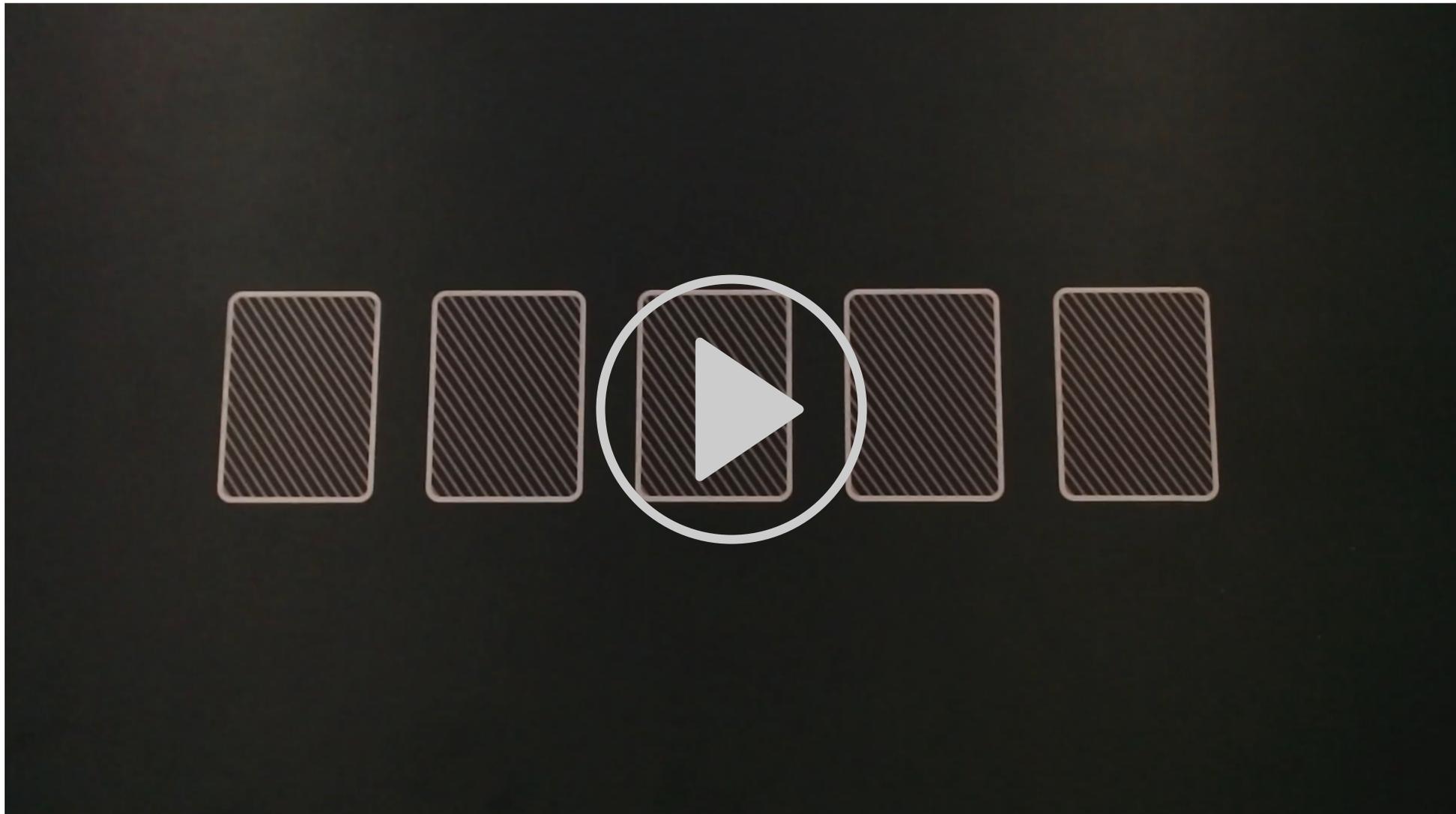
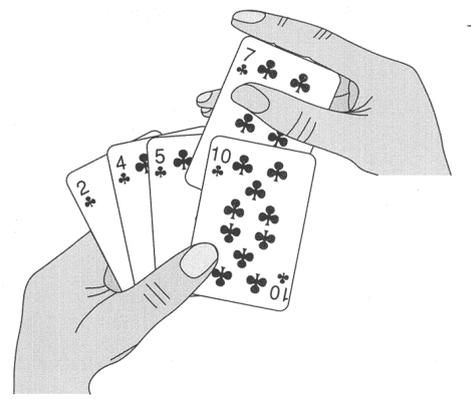
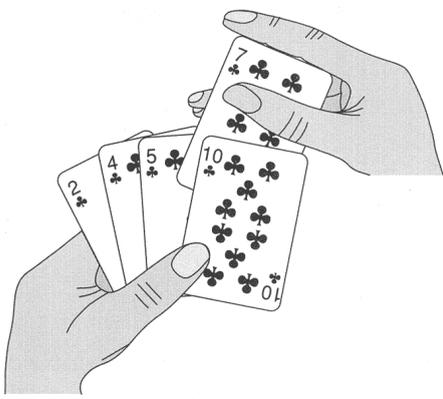
# Teile und Herrsche



## Idee.

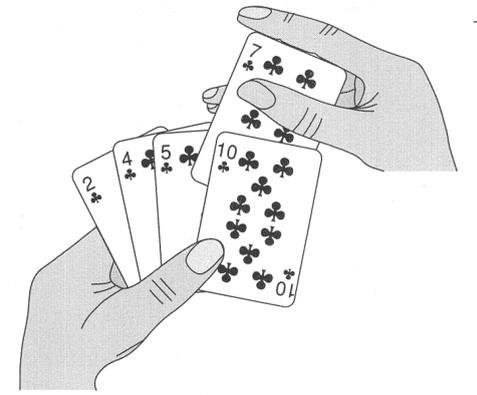
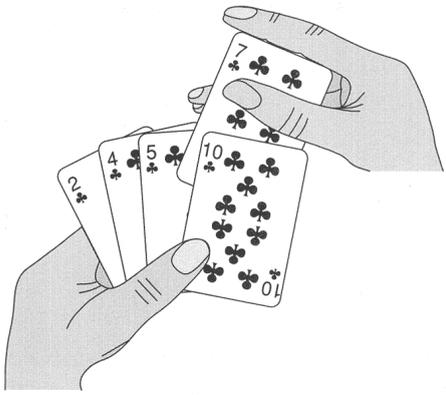
- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

# Teile und Herrsche



große Teile,  
(...onen) und  
zusammen.

# Teile und Herrsche

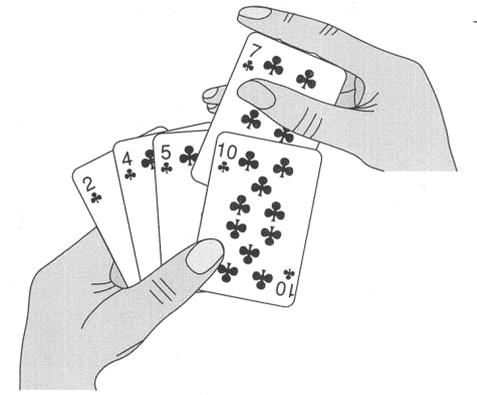
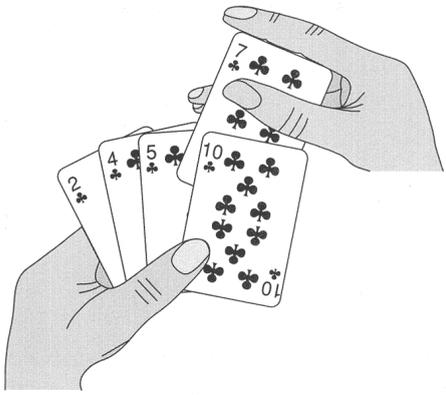


## Idee.

- 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 und Herrsche



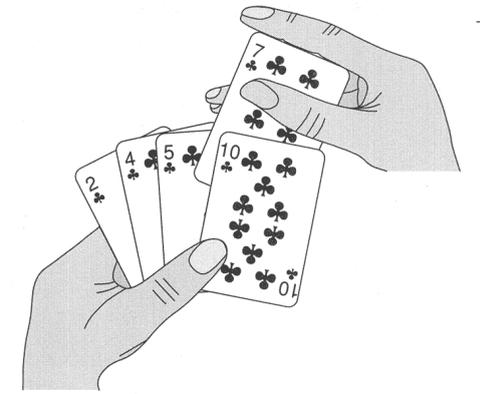
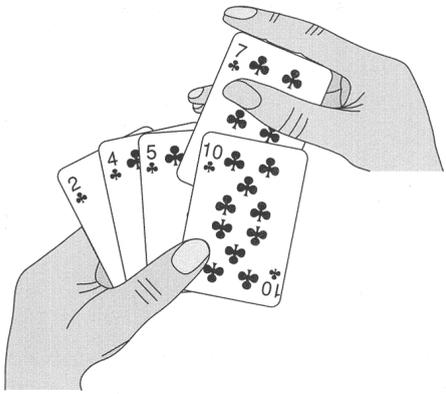
## Idee.

- 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 ...

# Teile und Herrsche



## Idee.

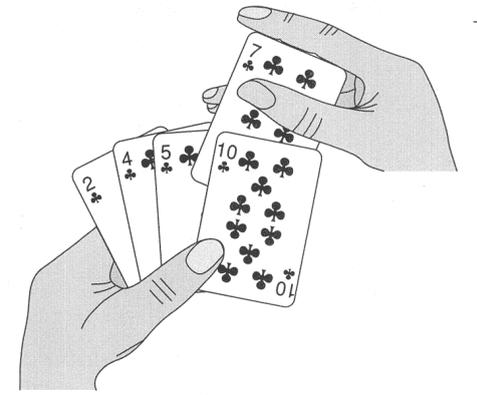
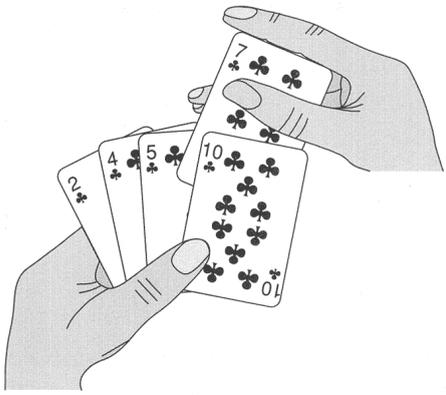
- teile den Kartenstapel in zwei ungefähr gleichgroße Teile,
- sortiere die Teile (z.B. durch verschiedene Personen) und
- füge die Teilstapel zu einem sortierten Stapel zusammen.

## Allgemein.

### Teile ...

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

# 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.

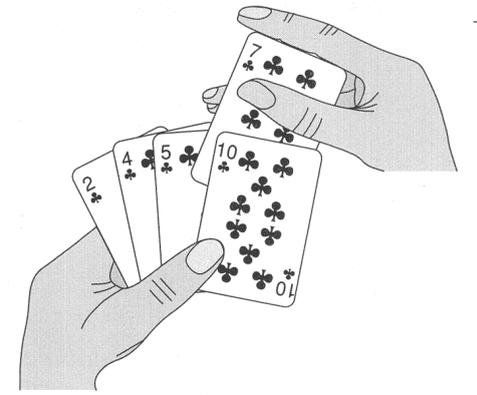
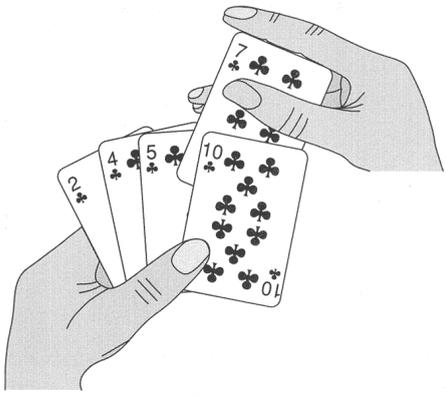
## Allgemein.

### Teile ...

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

### Herrsche ...

# 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.

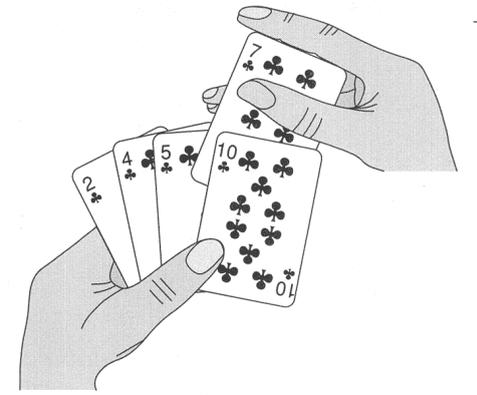
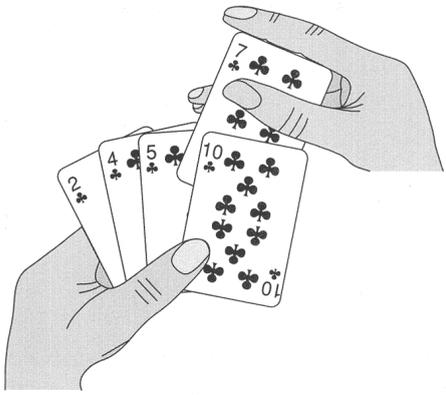
### Teile ...

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

### Herrsche ...

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

# 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 ...

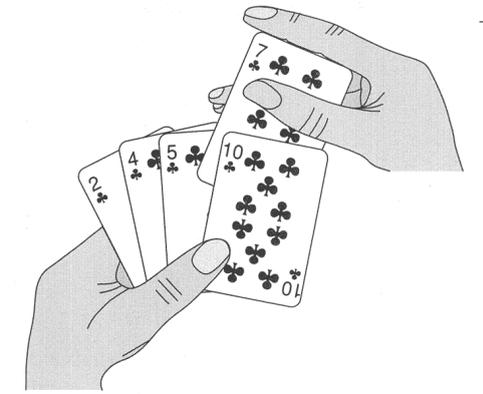
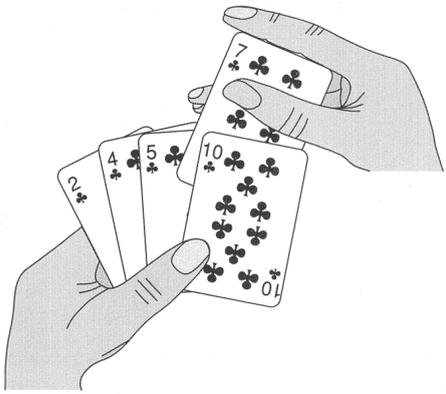
eine Instanz in kleinere Instanzen **desselben** Problems.

Aufruf einer Funktion durch sich selbst

### Herrsche ...

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

# 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.

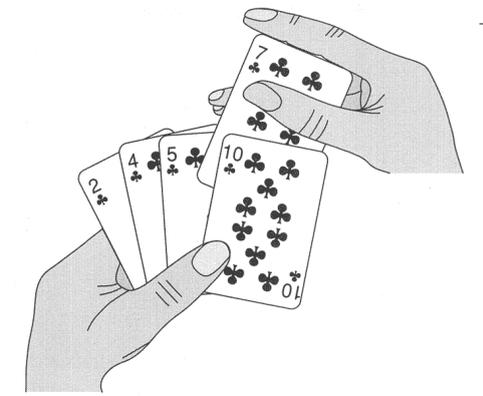
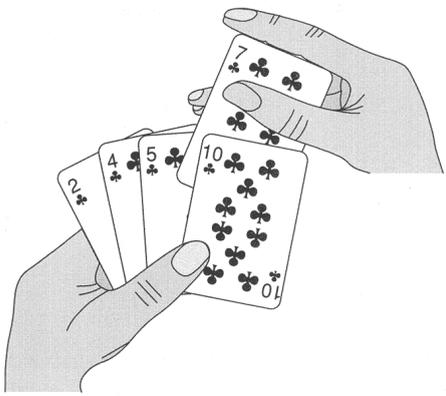
Aufruf einer Funktion durch sich selbst

### Herrsche ...

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

### Kombiniere ...

# 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

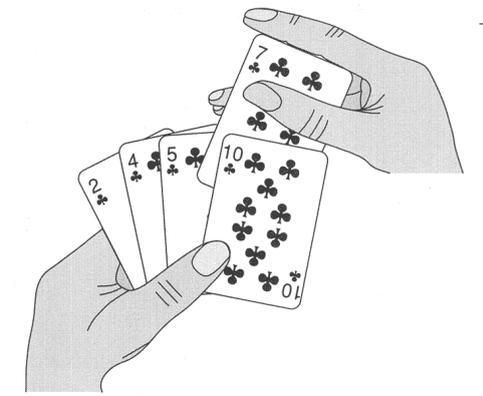
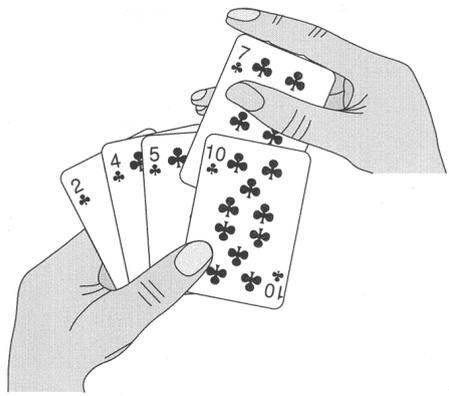
### Herrsche ...

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

### Kombiniere ...

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

**Allgemein.**

**Teile ...**

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

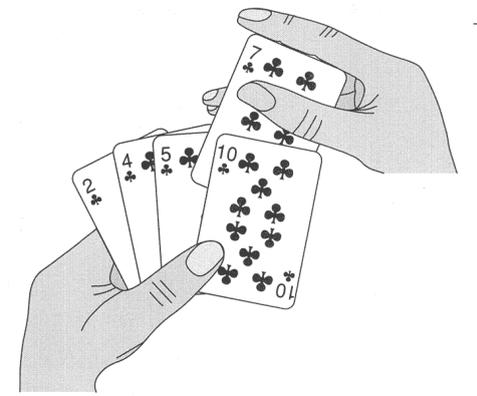
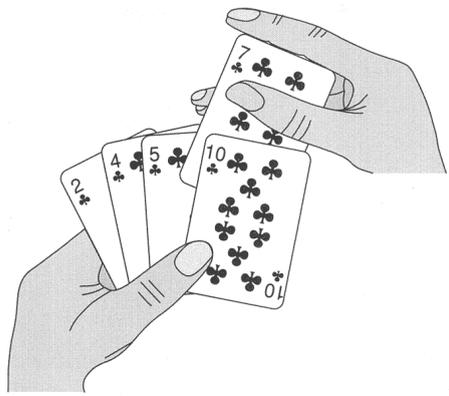
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

Defaultwerte

**Allgemein.**

**Teile ...**

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

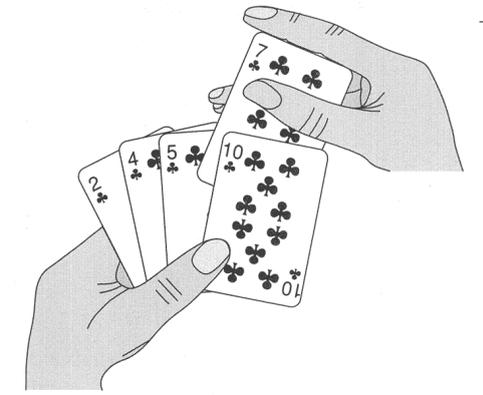
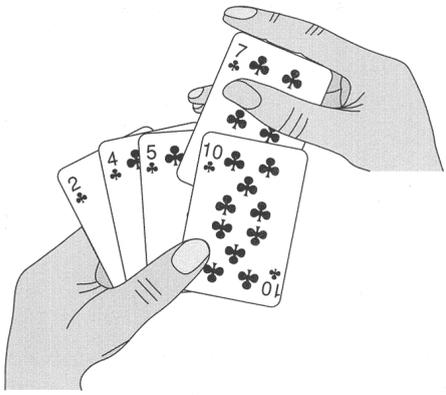
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

Defaultwerte

Dadurch wird die Funktion  
 $MERGESORT(A) \equiv$   
 $MERGESORT(A, 1, A.length)$   
 definiert.

**Allgemein.**

**Teile ...**

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

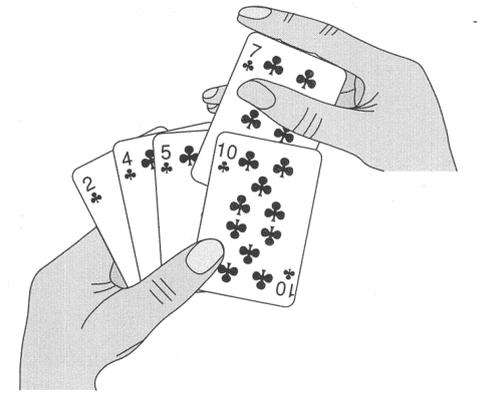
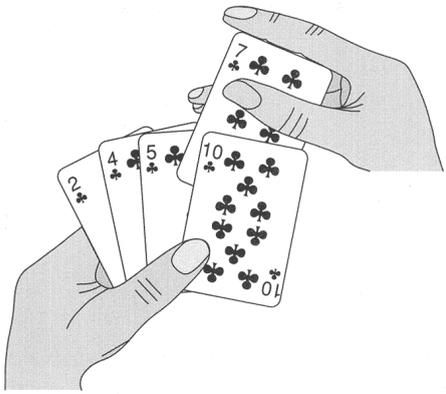
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

**Allgemein.**

**Teile ...**

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

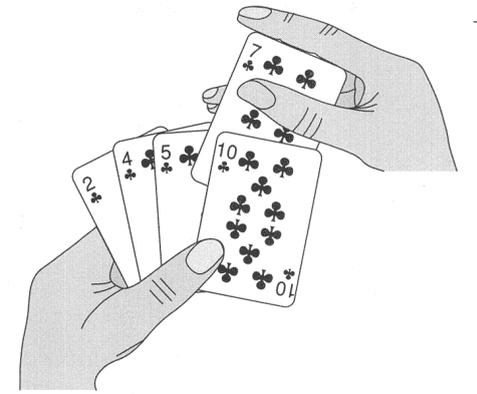
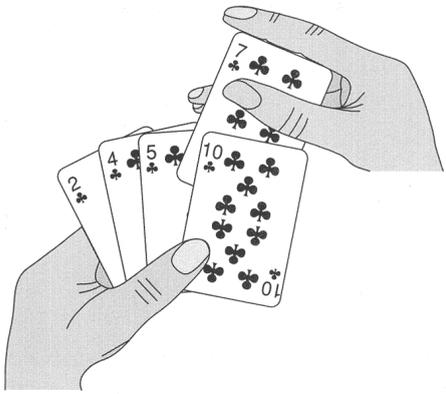
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
if  $l < r$  then
```

**Allgemein.**

**Teile ...**

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

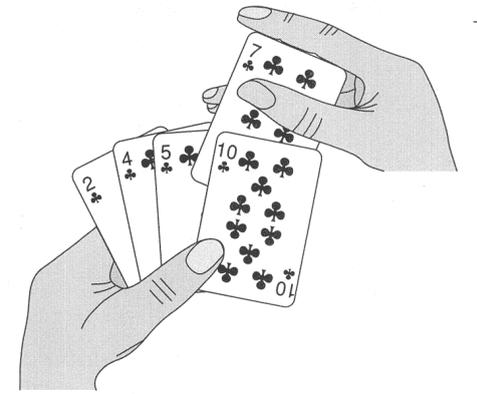
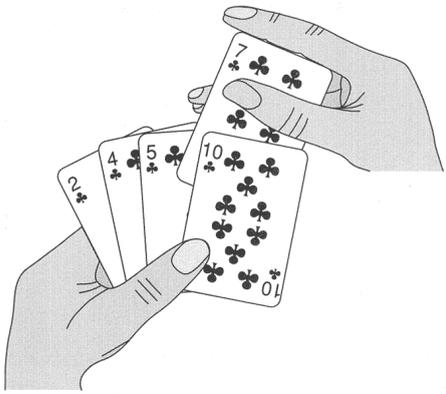
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
  if  $l < r$  then
     $m = \lfloor (l + r) / 2 \rfloor$ 

```

**Allgemein.**

**Teile ...**

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

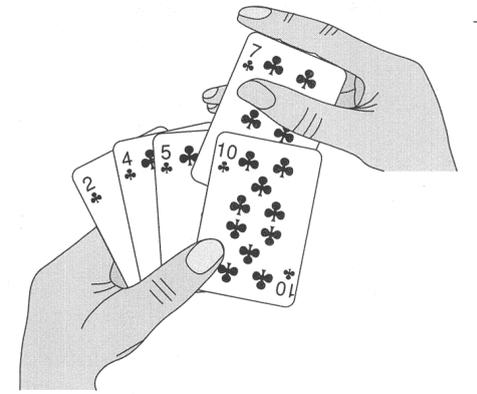
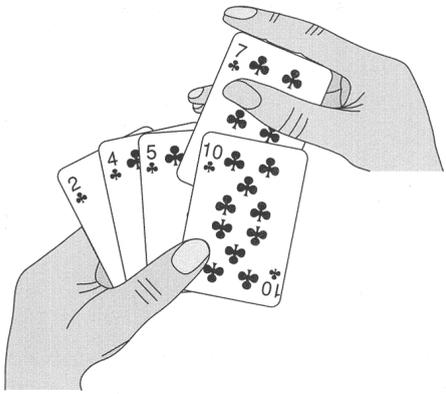
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
  if  $l < r$  then
     $m = \lfloor (l + r) / 2 \rfloor$  } teile

```

**Allgemein.**

**Teile ...**

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

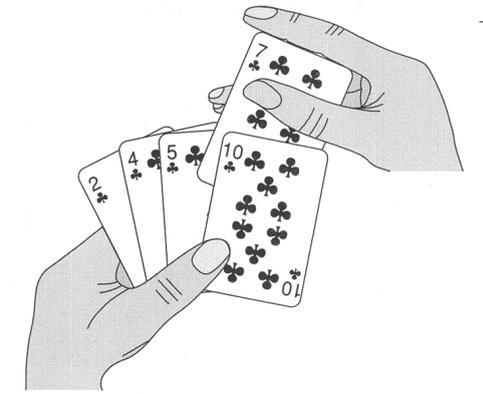
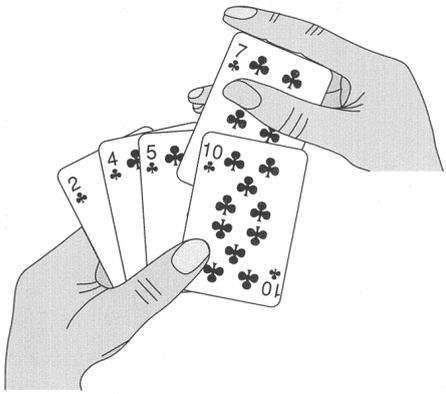
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
if  $l < r$  then
     $m = \lfloor (l + r) / 2 \rfloor$  } teile
    MERGESORT(A,  $l$ ,  $m$ )
    MERGESORT(A,  $m + 1$ ,  $r$ )
  
```

**Allgemein.**

**Teile ...**

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

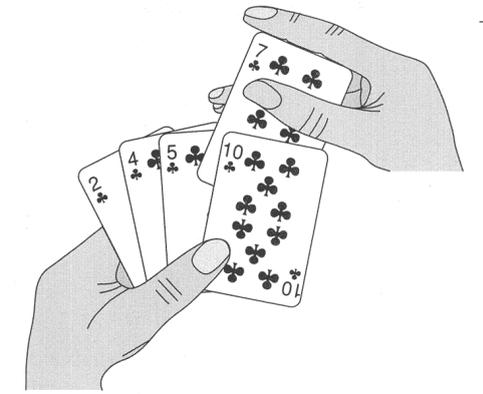
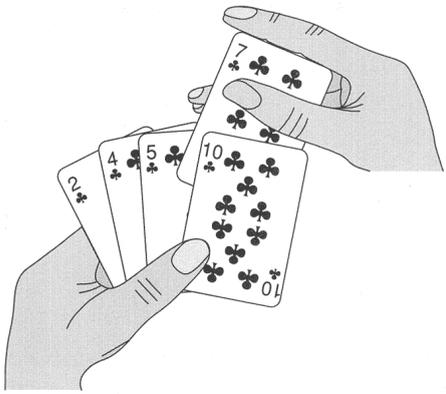
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
if  $l < r$  then
     $m = \lfloor (l + r) / 2 \rfloor$  } teile
    MERGESORT(A,  $l$ ,  $m$ ) } herrsche
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
  
```

**Allgemein.**

**Teile ...**

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

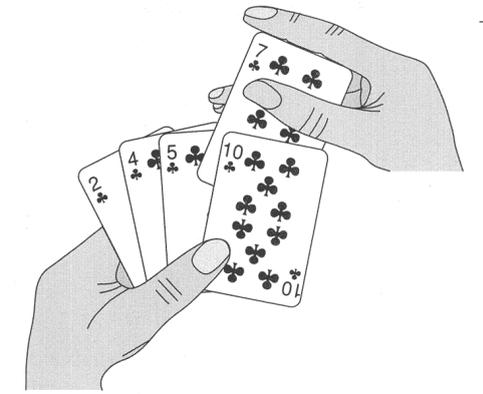
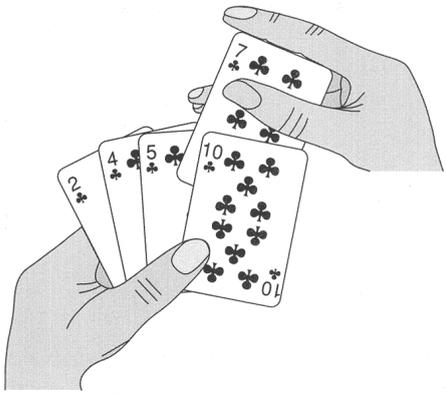
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

MERGESORT(int[] A, int  $\ell = 1$ , int  $r = A.length$ )
if  $\ell < r$  then
     $m = \lfloor (\ell + r) / 2 \rfloor$  } teile
    MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
    MERGE(A,  $\ell$ ,  $m$ ,  $r$ )
  
```

**Allgemein.**

**Teile ...**

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

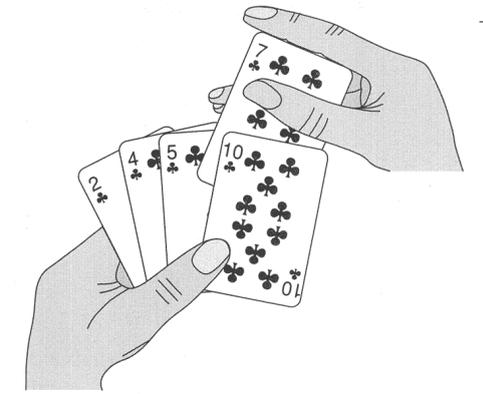
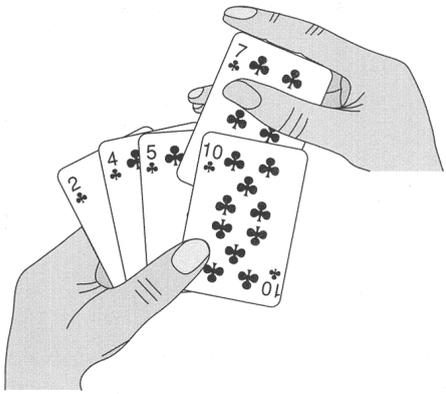
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

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

**Allgemein.**

**Teile ...**

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

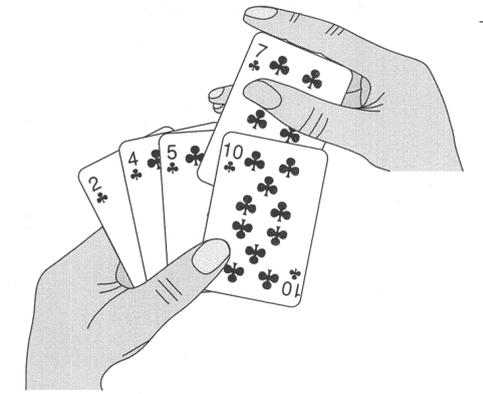
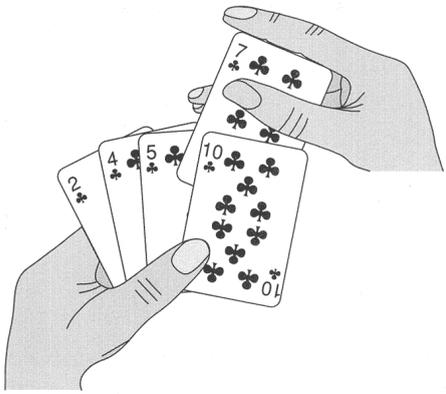
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

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

**Allgemein.**

**Teile ...**

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

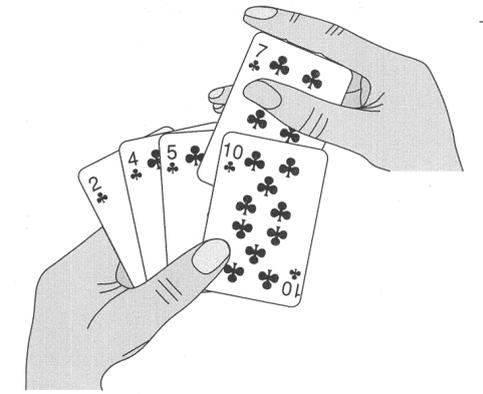
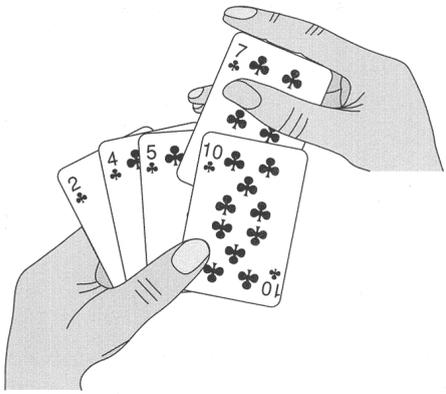
**Herrsche ...**

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

**Kombiniere ...**

die Teillösungen zu einer Lösung der ursprünglichen Instanz.

# Teile und Herrsche



```

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

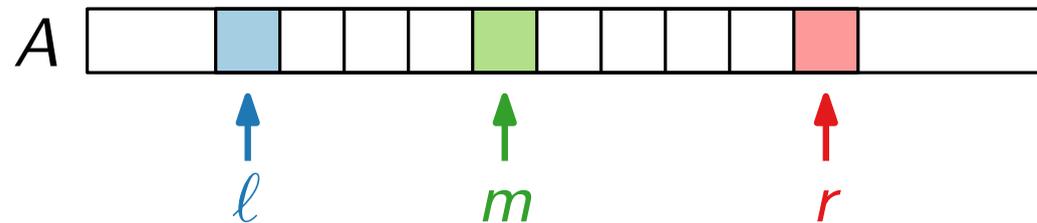
# Kombiniere

# Kombiniere

```
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )
```

# Kombiniere

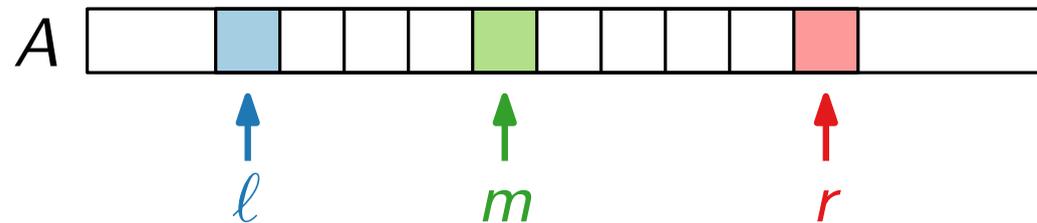
```
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )
```



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

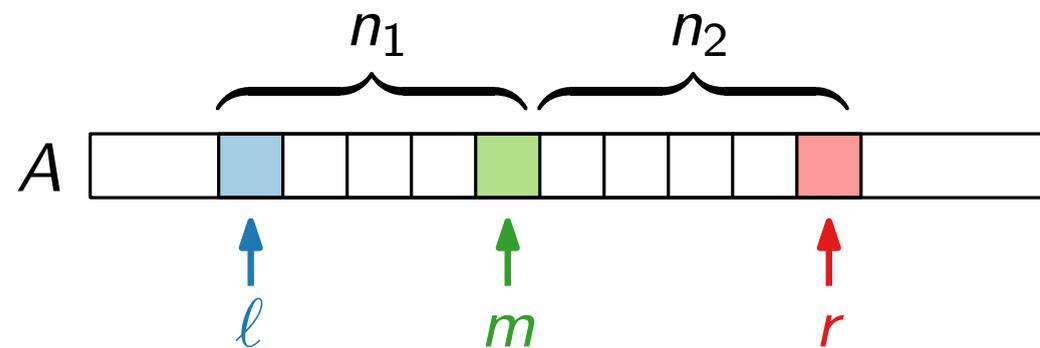
$$n_1 = m - l + 1; \quad n_2 = r - m$$



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

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

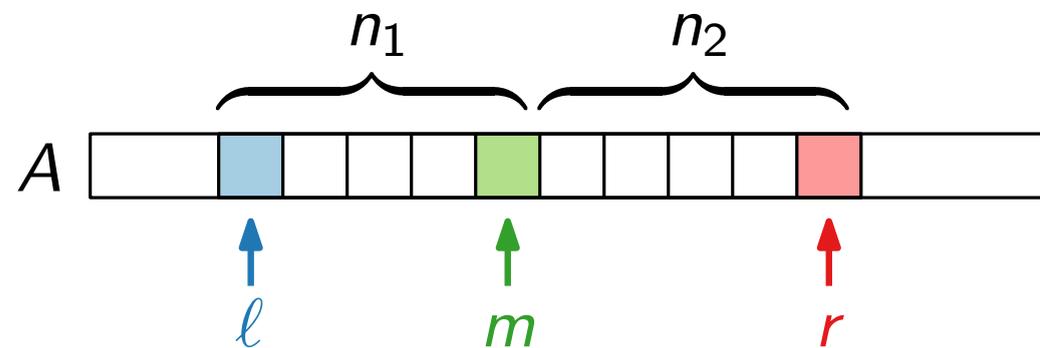


# Kombiniere

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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



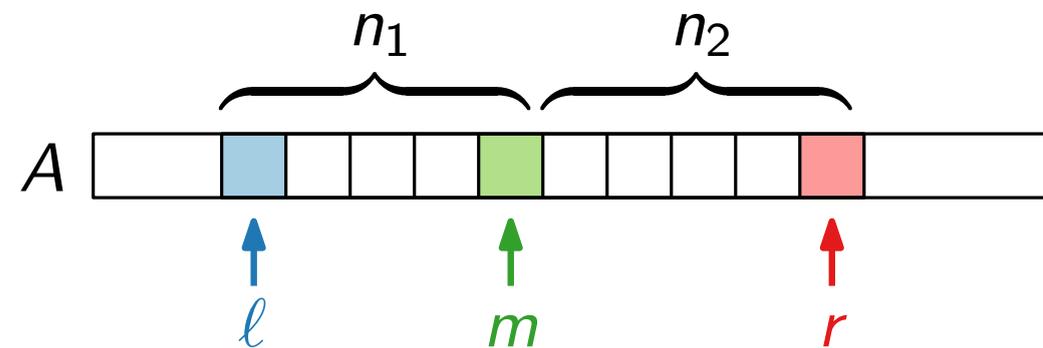
# Kombiniere

```
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )
```

```
   $n_1 = m - l + 1$ ;  $n_2 = r - m$ 
```

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

```
   $L[1 \dots n_1] = A[l \dots m]$ 
```



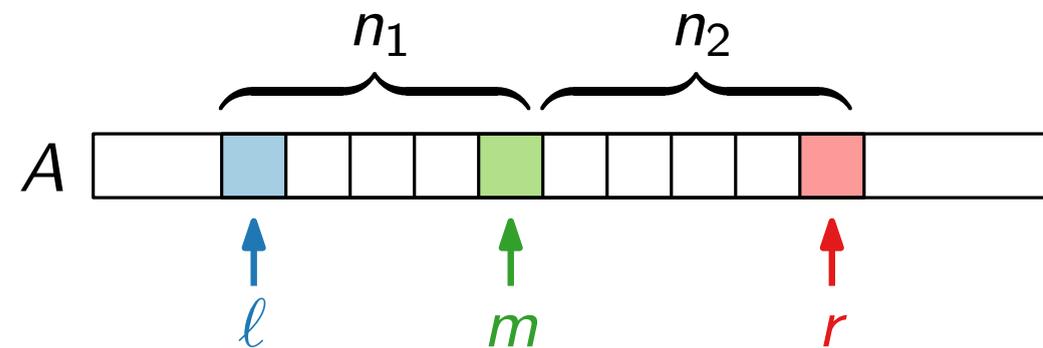
# Kombiniere

```
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )
```

```
 $n_1 = m - l + 1$ ;  $n_2 = r - m$ 
```

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

```
 $L[1 \dots n_1] = A[l \dots m]$ 
```



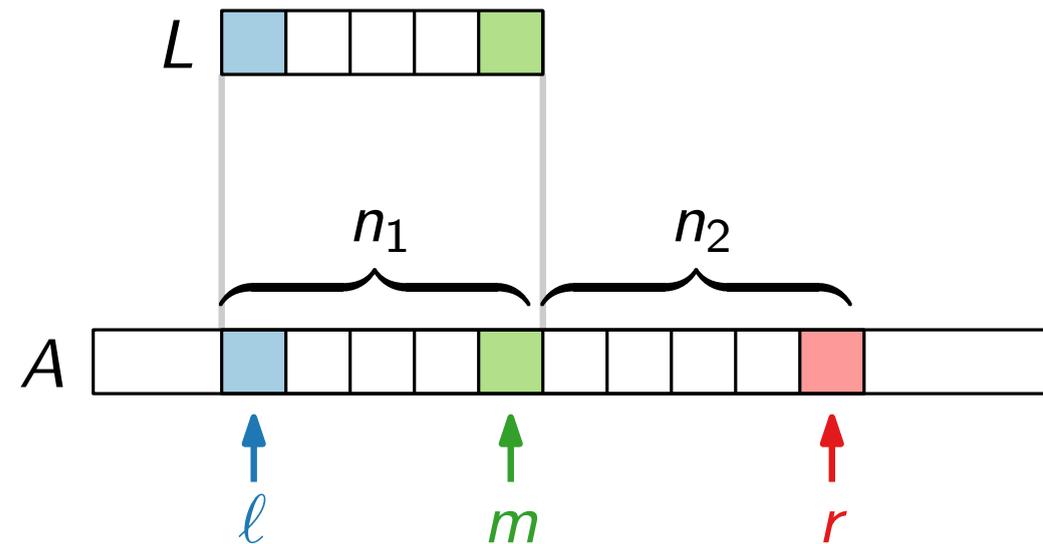
# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

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

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



# Kombiniere

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

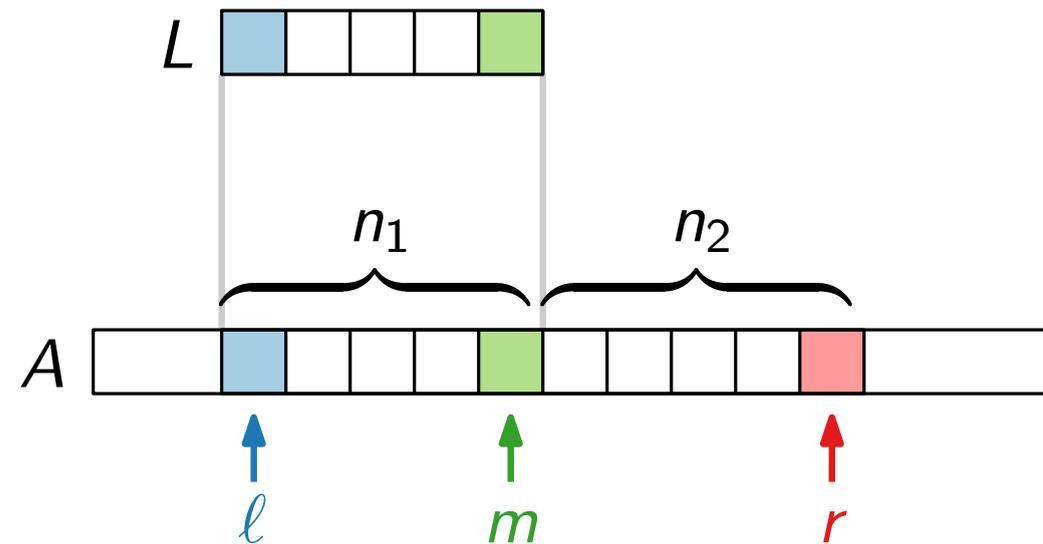
$n_1 = m - \ell + 1$ ;  $n_2 = r - m$

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

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

for  $i = 1$  to  $n_1$  do

└  $L[i] = A[(\ell - 1) + i]$



# Kombiniere

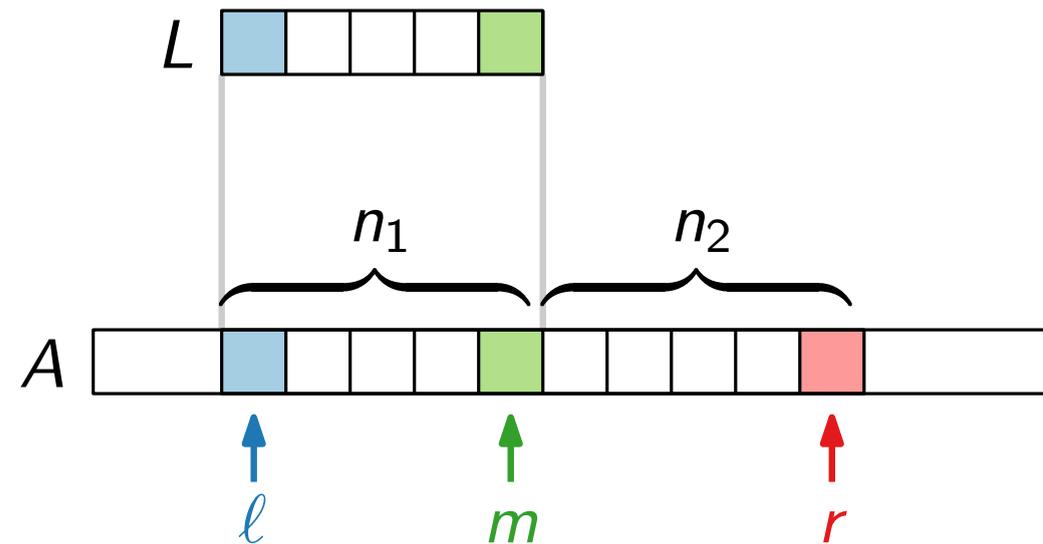
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

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

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

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



# Kombiniere

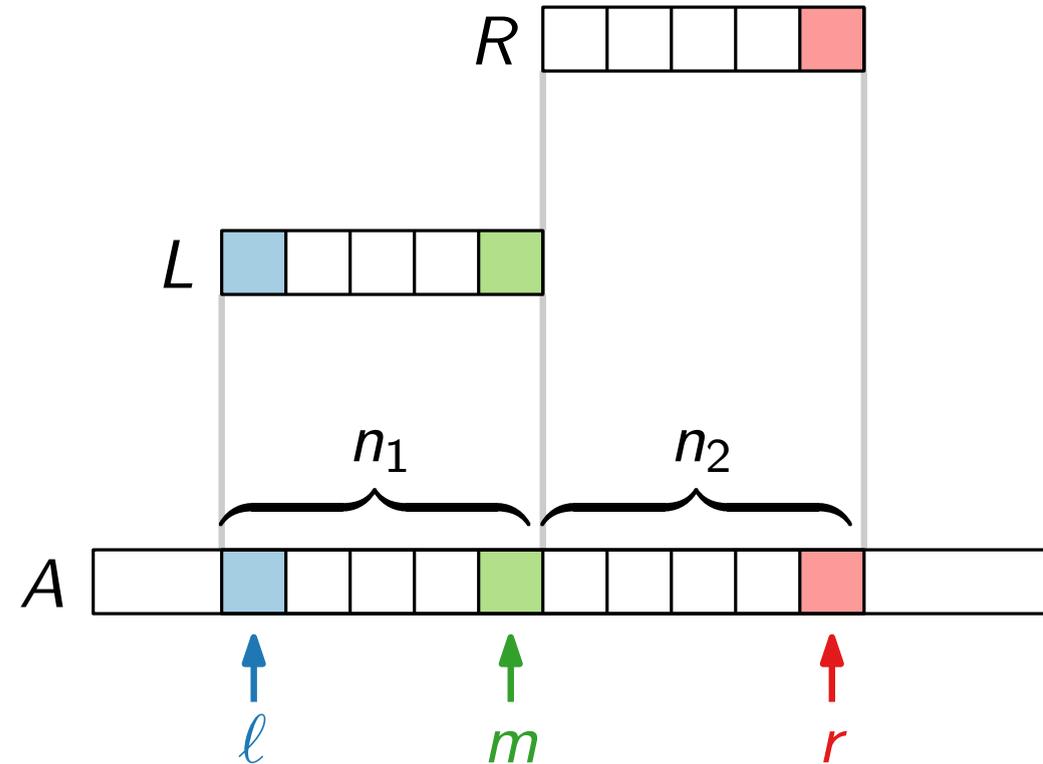
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

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

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

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



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

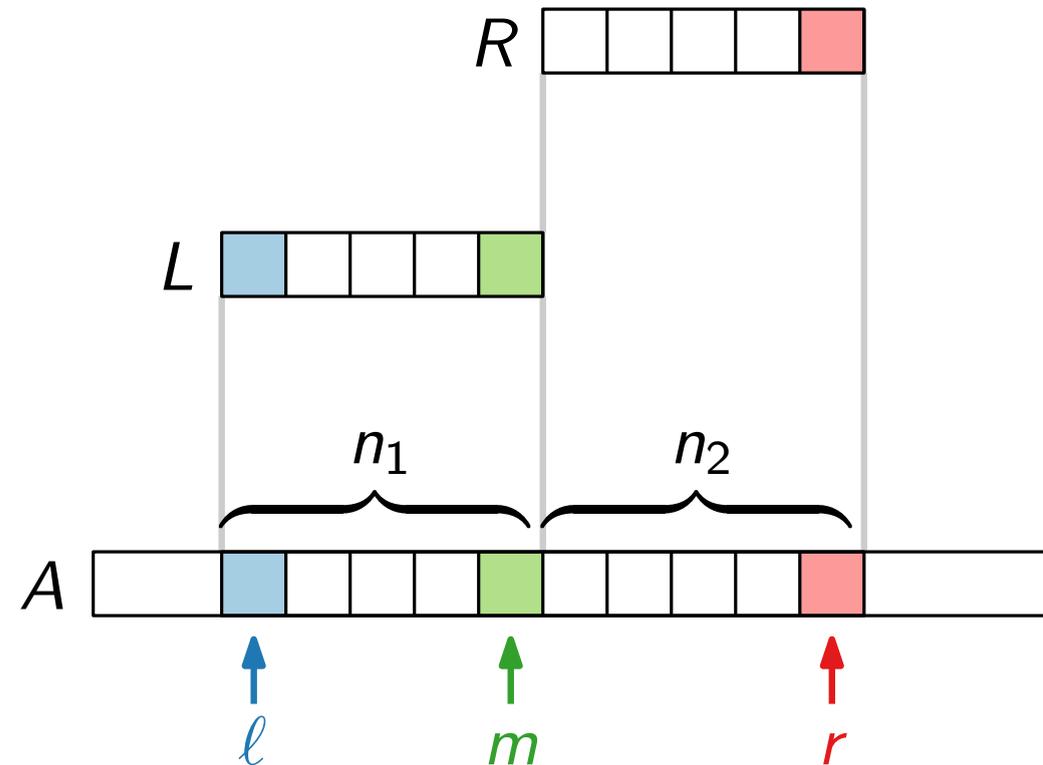
$$n_1 = m - l + 1; \quad n_2 = r - m$$

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

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

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

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



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

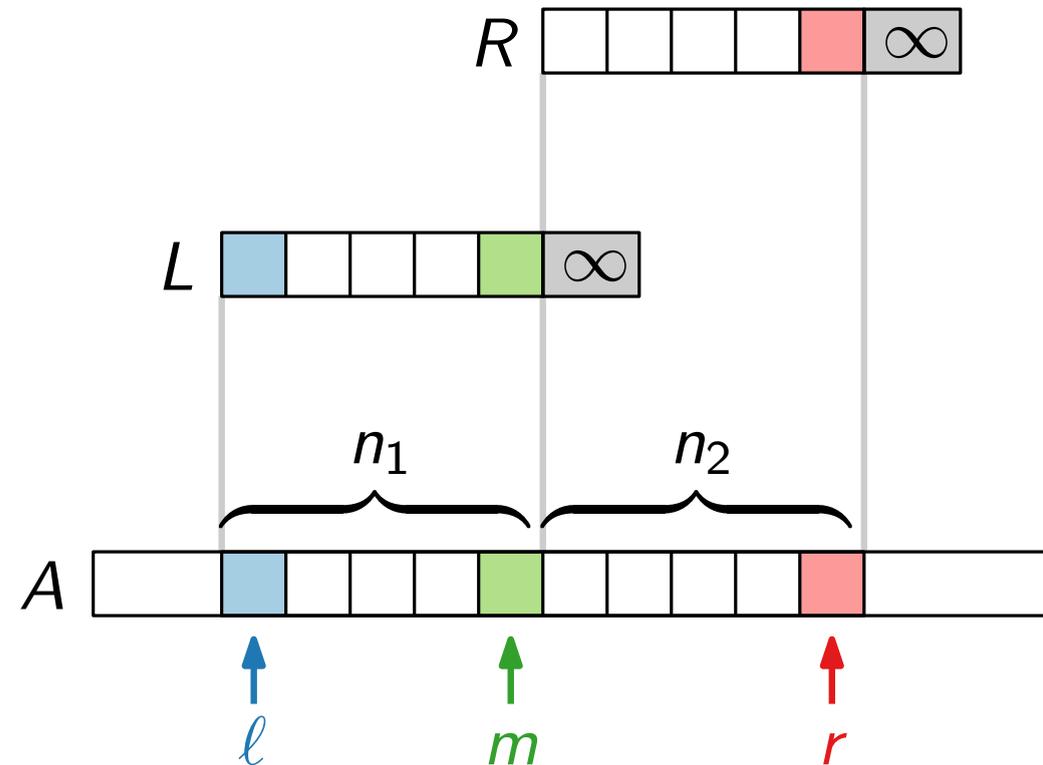
$$n_1 = m - l + 1; \quad n_2 = r - m$$

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

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

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

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



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

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

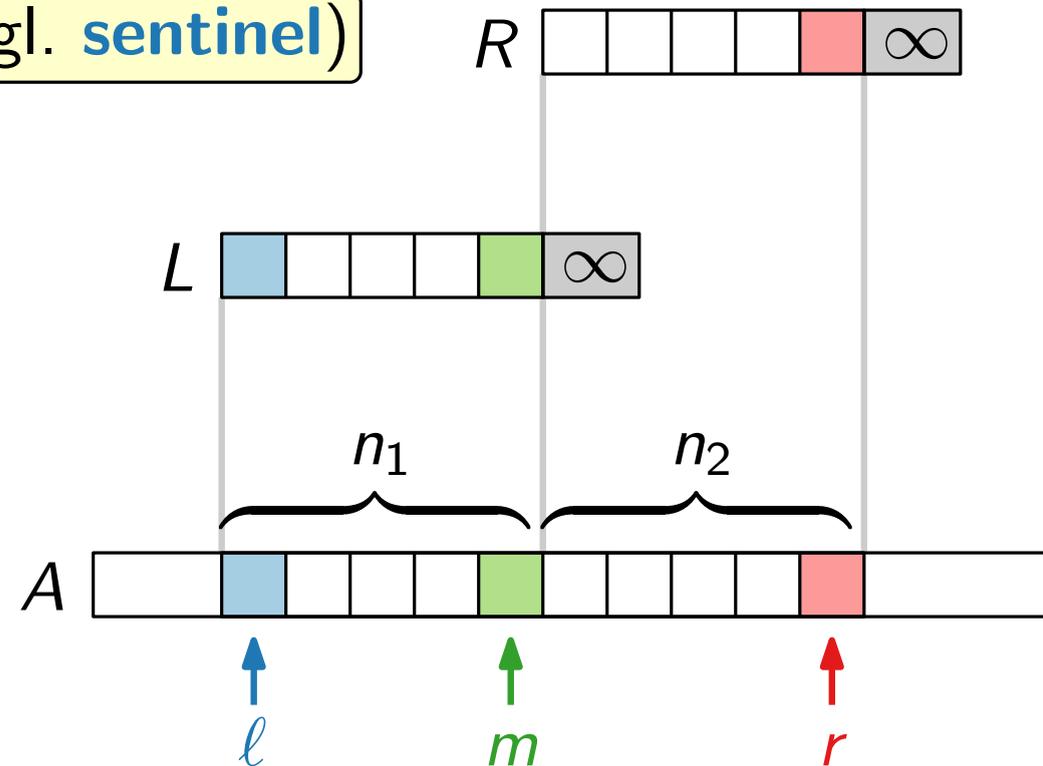
$L = \text{new int}[1 \dots n_1 + 1];$   $R = \text{new int}[1 \dots n_2 + 1]$

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

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

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

Stopper (engl. **sentinel**)



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

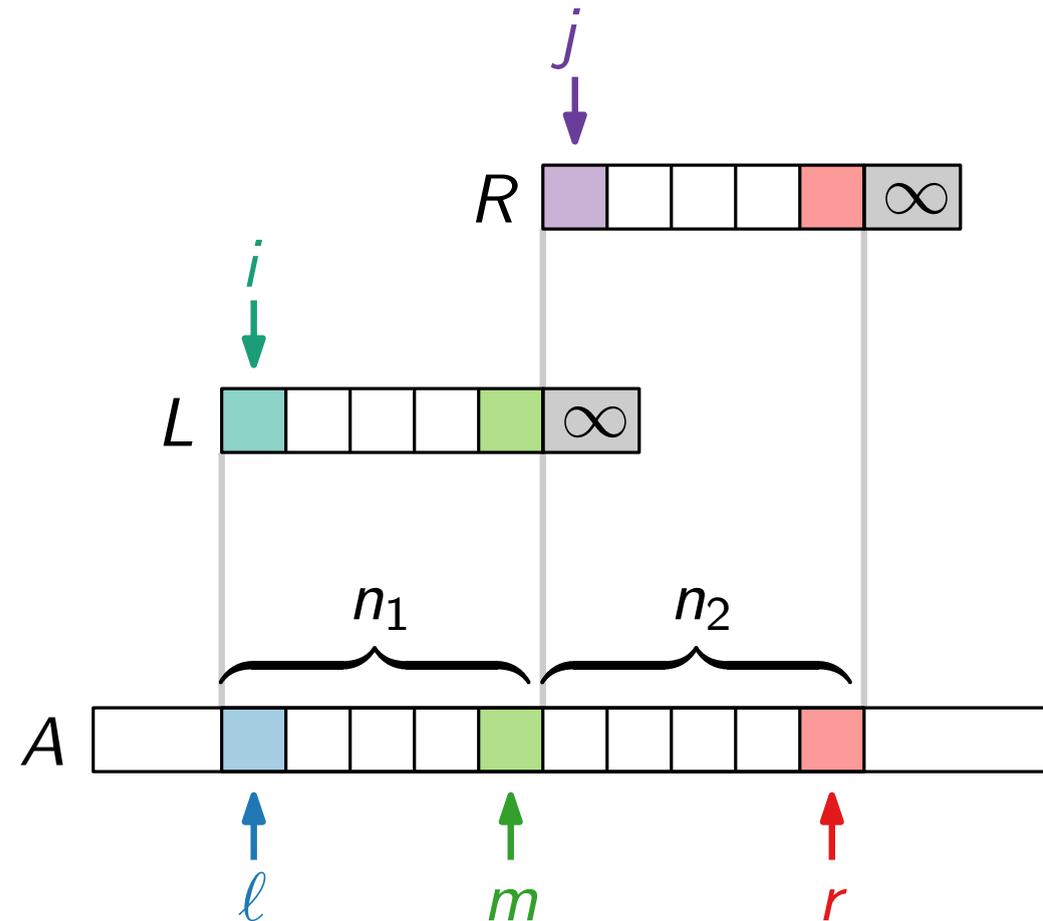
$L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$

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

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

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

$i = j = 1$



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

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

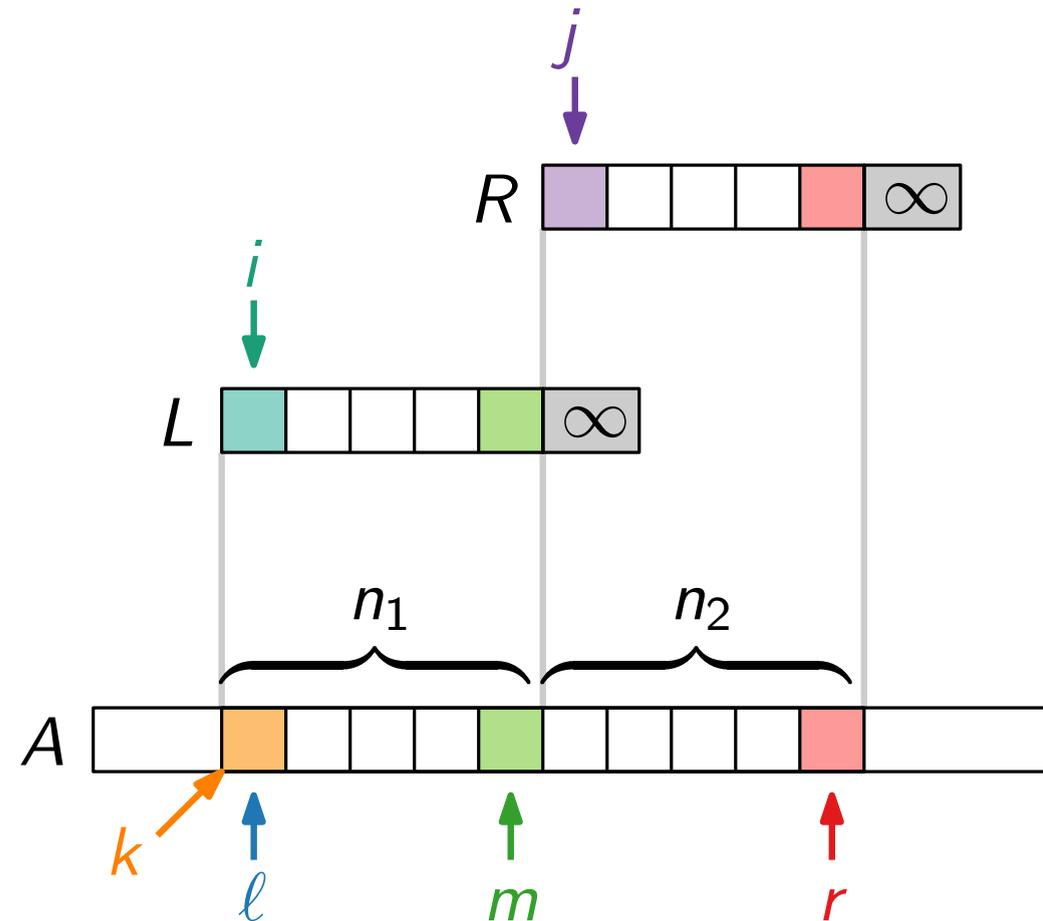
$L[1 \dots n_1] = A[l \dots m]$

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

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

$i = j = 1$

for  $k = l$  to  $r$  do



# Kombiniere

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

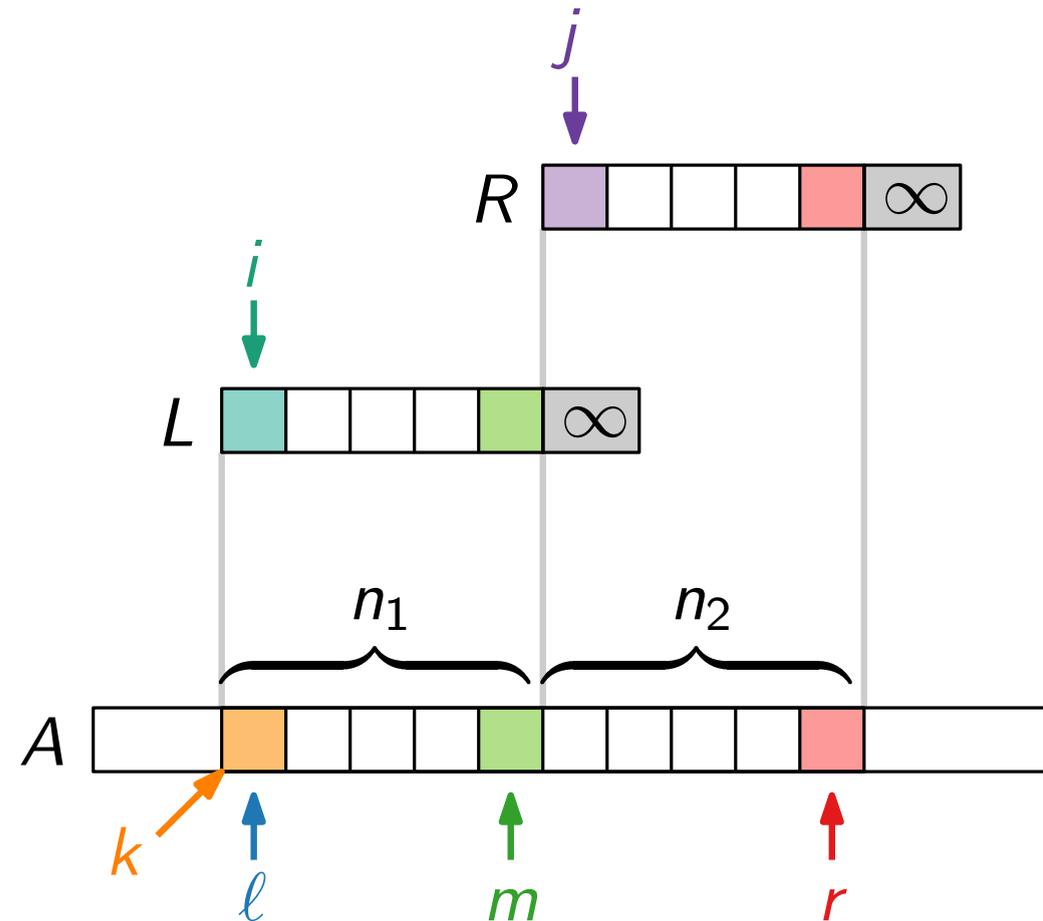
$i = j = 1$

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

## Aufgabe.

Schreiben Sie den Rest der Routine auf ein Stück Papier!  
Benutzen Sie dazu  $L$  und  $R$ .

[5 Minuten]



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

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

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

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

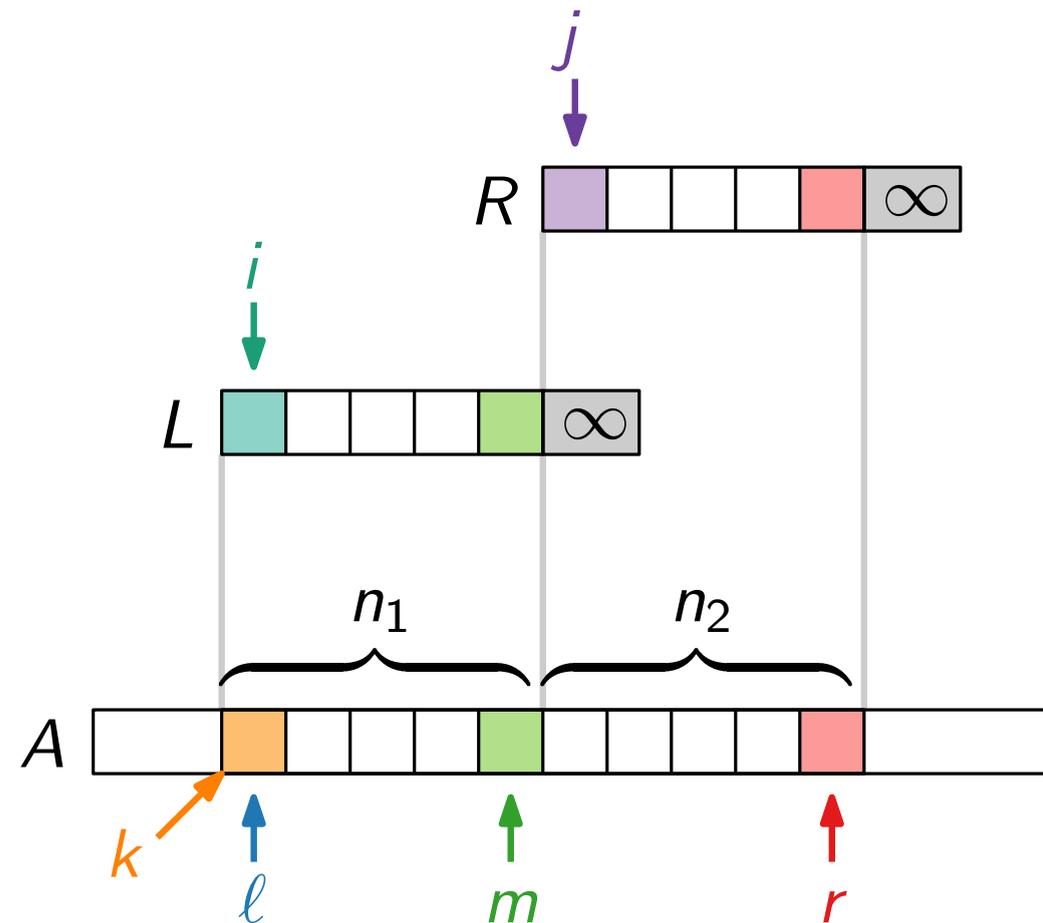
$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$

for  $k = l$  to  $r$  do

    if  $L[i] \leq R[j]$  then

        else



# Kombiniere

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

```
  L = new int[1... $n_1 + 1$ ]; R = new int[1... $n_2 + 1$ ]
```

```
  L[1... $n_1$ ] = A[ $\ell$ ... $m$ ]
```

```
  R[1... $n_2$ ] = A[ $m + 1$ ... $r$ ]
```

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

```
   $i = j = 1$ 
```

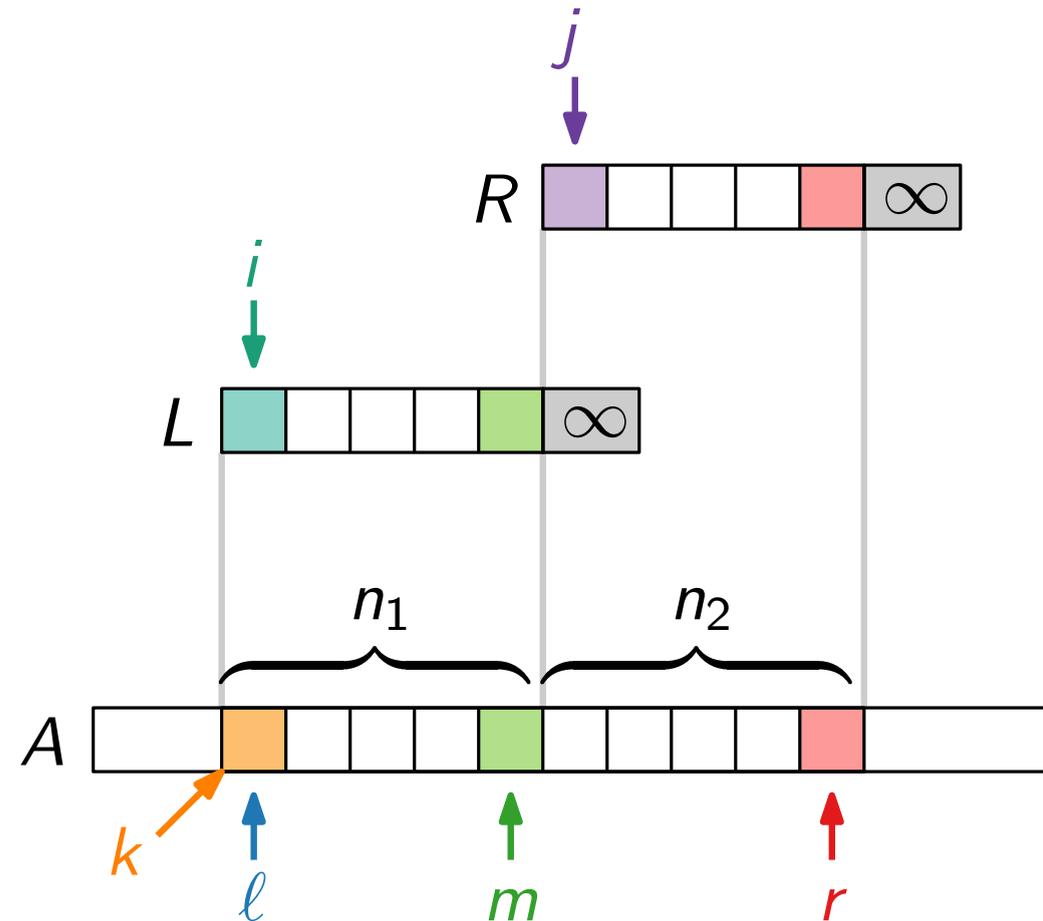
```
  for  $k = \ell$  to  $r$  do
```

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

```
      |  $A[k] = L[i]$ 
```

```
      |  $i = i + 1$ 
```

```
    else
```



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

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

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

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

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

$i = j = 1$

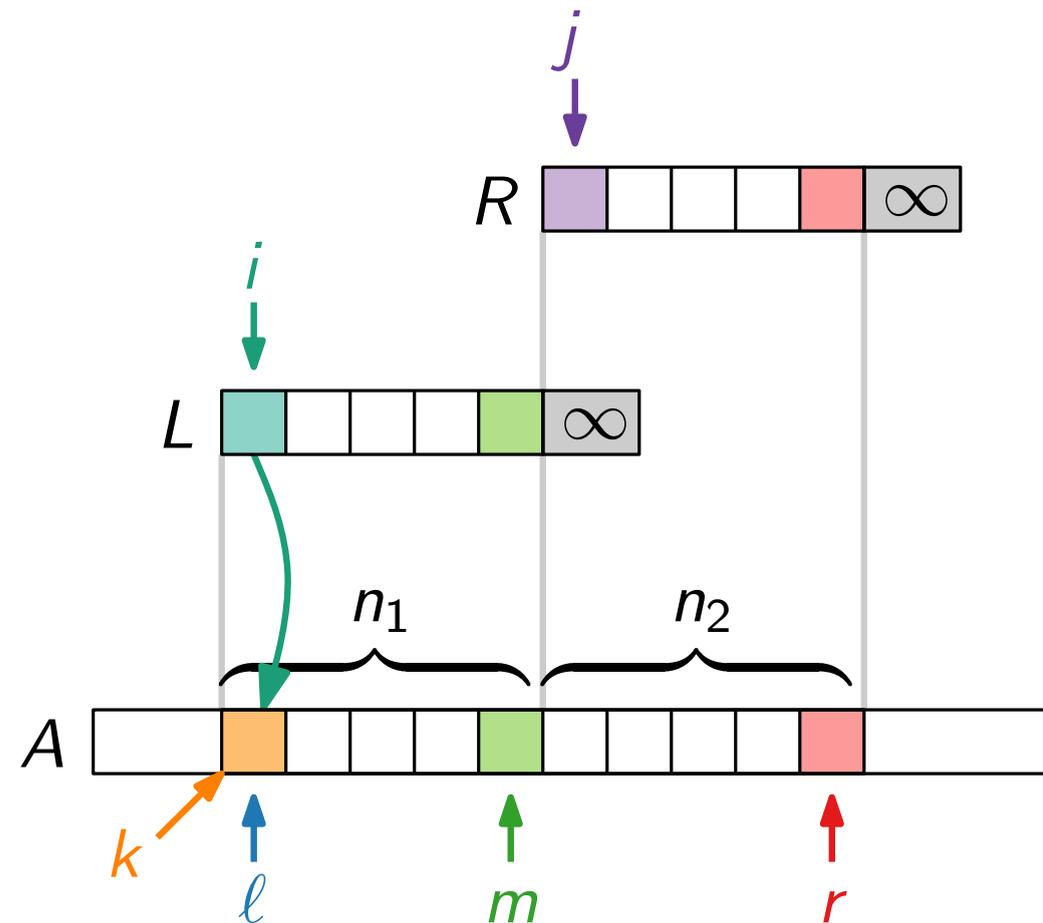
for  $k = l$  to  $r$  do

    if  $L[i] \leq R[j]$  then

$A[k] = L[i]$

$i = i + 1$

    else



# Kombiniere

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

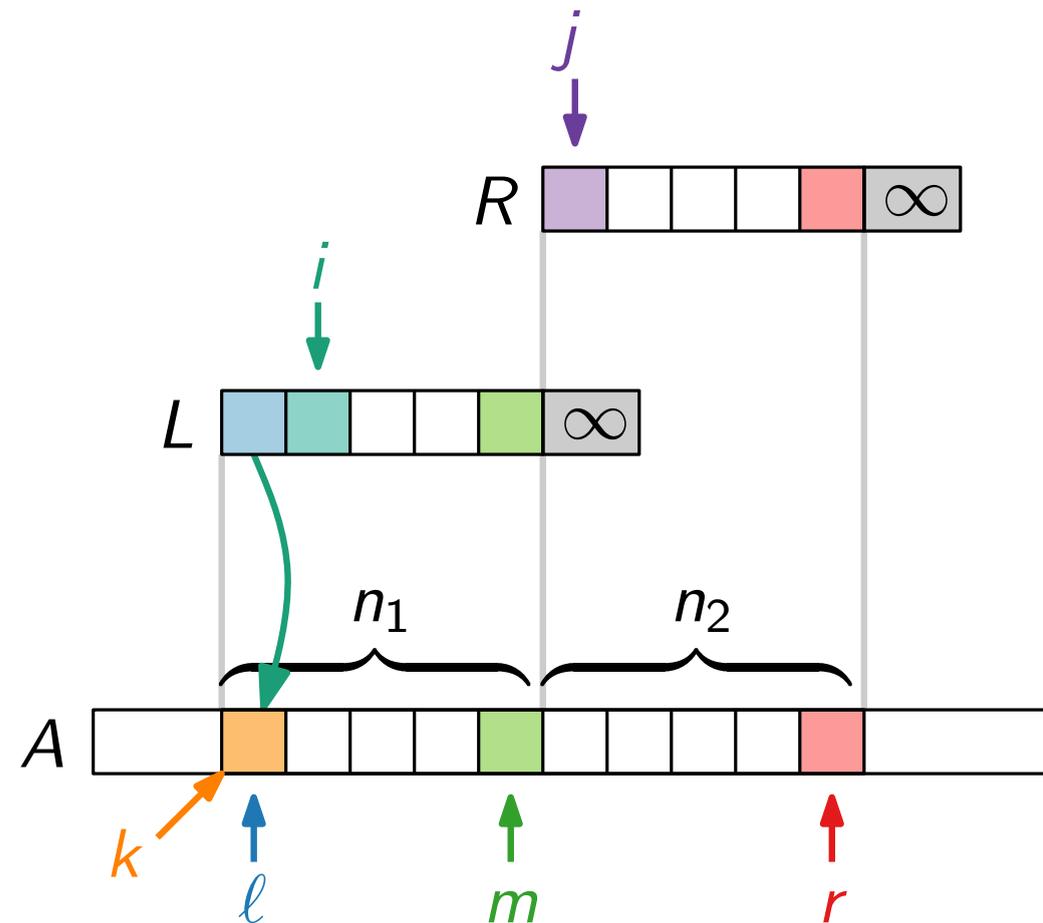
for  $k = \ell$  to  $r$  do

    if  $L[i] \leq R[j]$  then

$A[k] = L[i]$

$i = i + 1$

    else



# Kombiniere

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

```
  L = new int[1... $n_1 + 1$ ]; R = new int[1... $n_2 + 1$ ]
```

```
  L[1... $n_1$ ] = A[ $\ell$ ... $m$ ]
```

```
  R[1... $n_2$ ] = A[ $m + 1$ ... $r$ ]
```

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

```
   $i = j = 1$ 
```

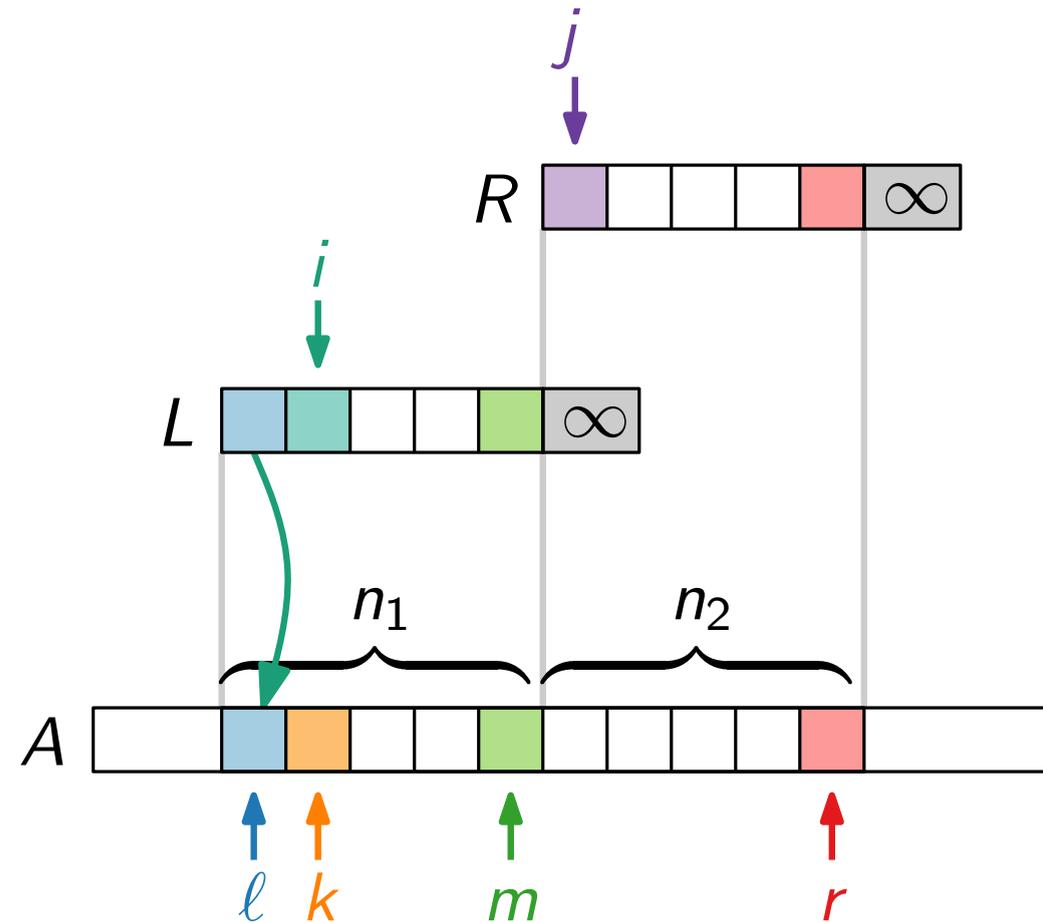
```
  for  $k = \ell$  to  $r$  do
```

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

```
      |  $A[k] = L[i]$ 
```

```
      |  $i = i + 1$ 
```

```
    else
```



# Kombiniere

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

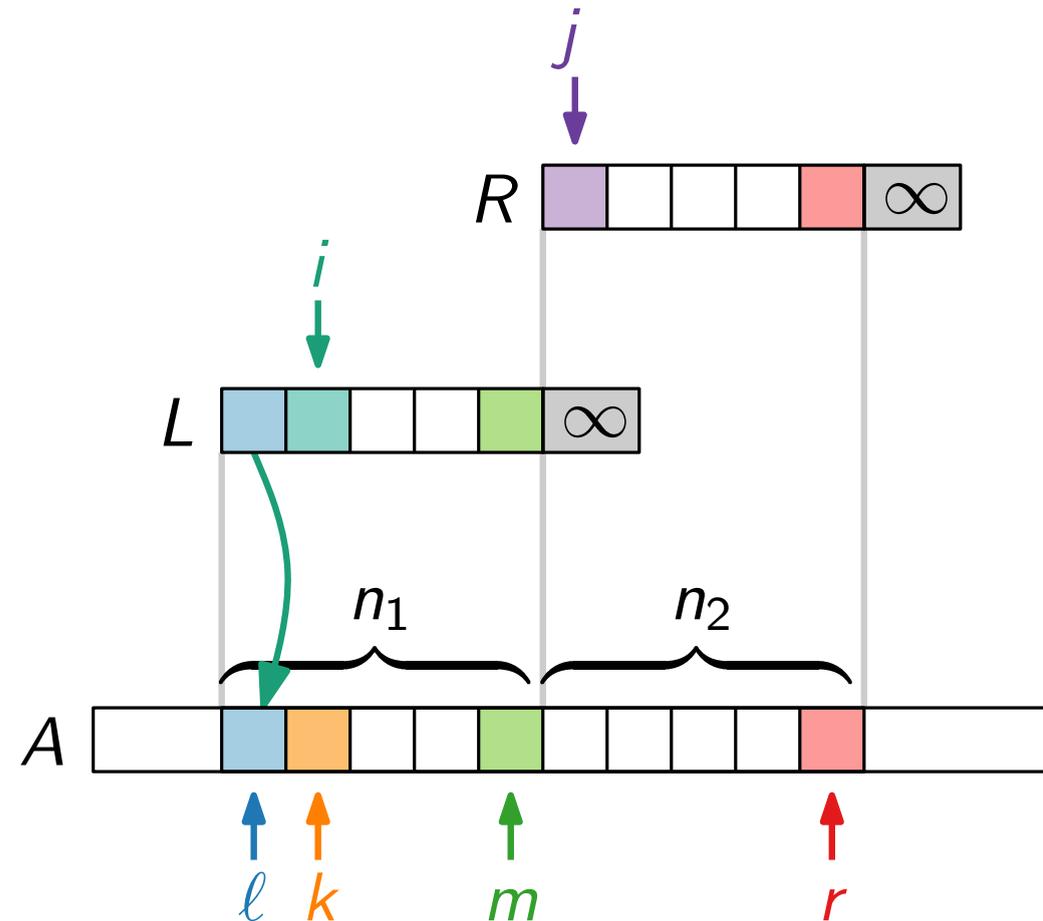
$A[k] = L[i]$

$i = i + 1$

  else

$A[k] = R[j]$

$j = j + 1$



# Kombiniere

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

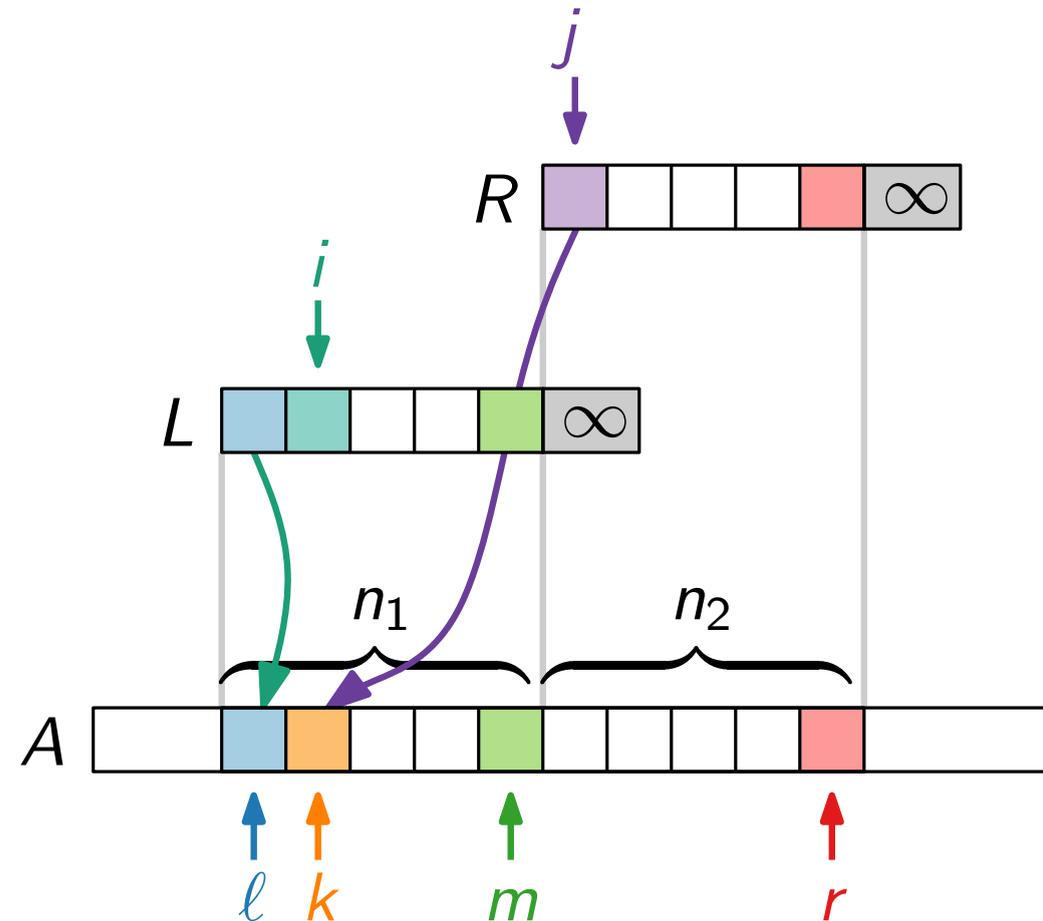
$A[k] = L[i]$

$i = i + 1$

  else

$A[k] = R[j]$

$j = j + 1$



# Kombiniere

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

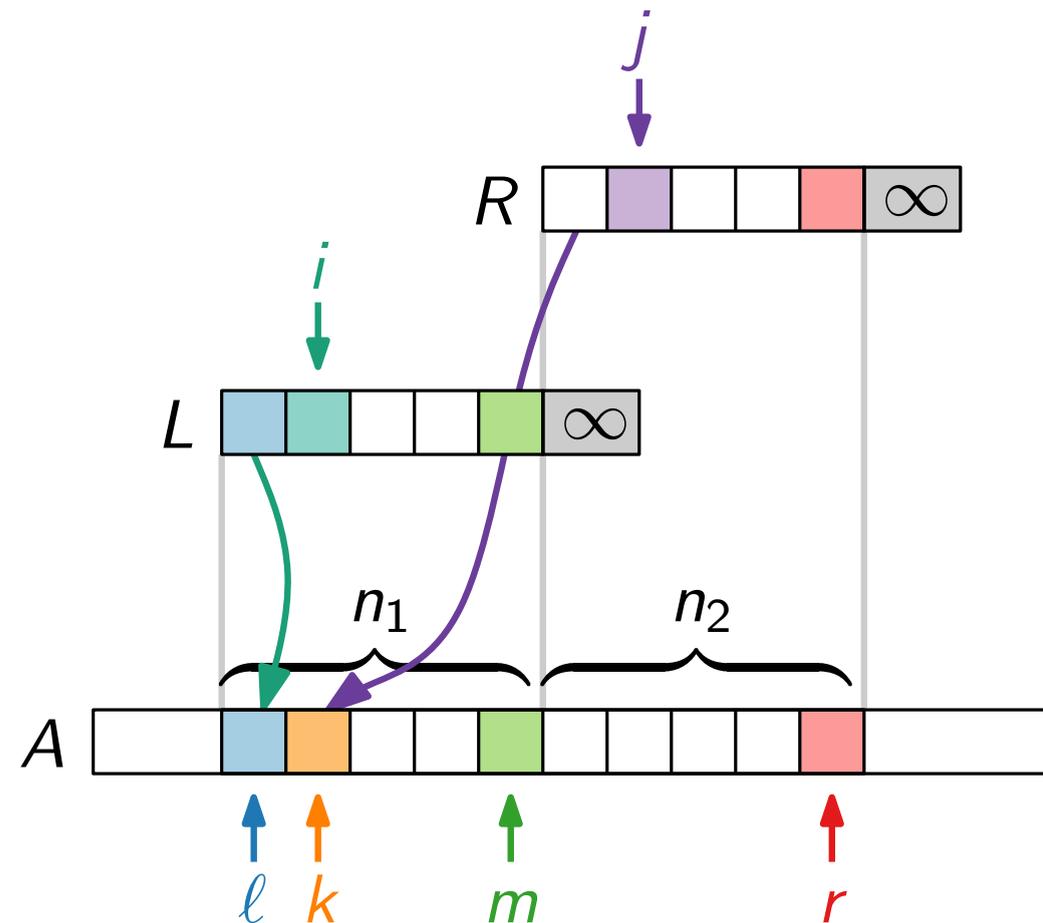
$A[k] = L[i]$

$i = i + 1$

  else

$A[k] = R[j]$

$j = j + 1$



# Kombiniere

MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )

$n_1 = m - l + 1$ ;  $n_2 = r - m$

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

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

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

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

$i = j = 1$

for  $k = l$  to  $r$  do

  if  $L[i] \leq R[j]$  then

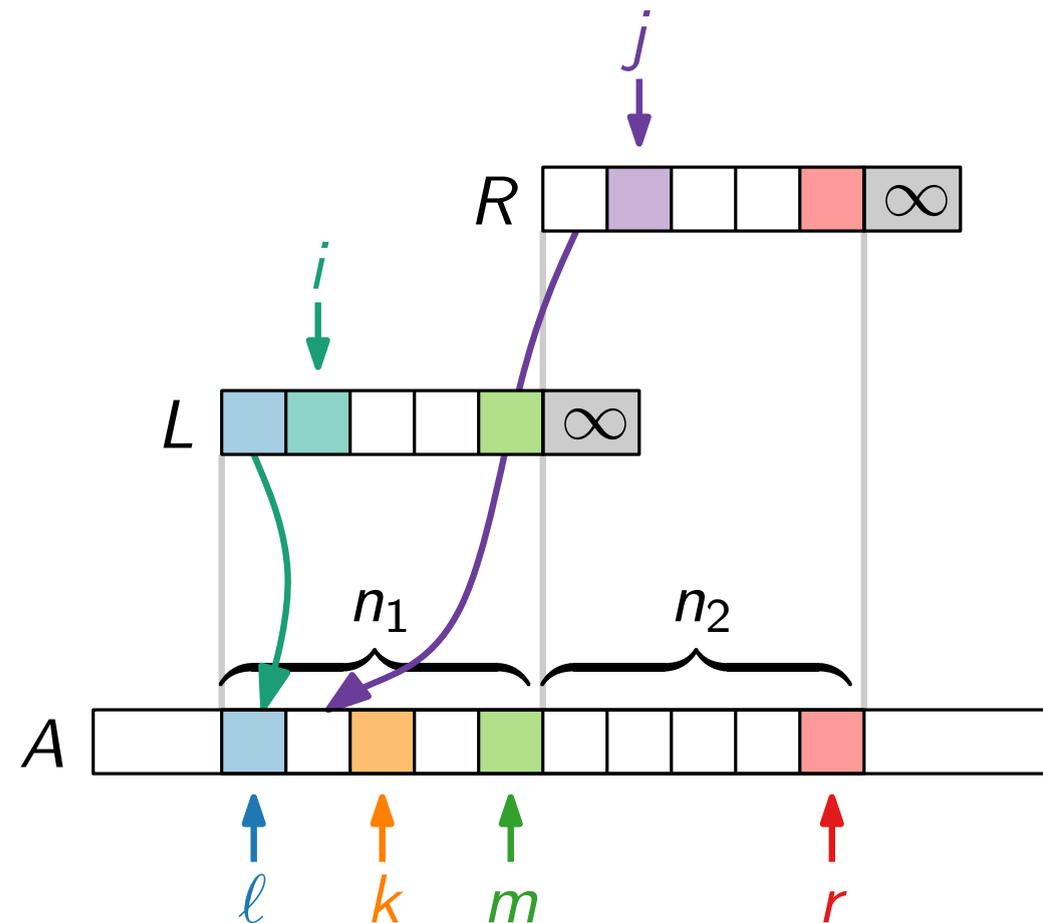
$A[k] = L[i]$

$i = i + 1$

  else

$A[k] = R[j]$

$j = j + 1$



# Kombiniere

```
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )
```

```
 $n_1 = m - l + 1$ ;  $n_2 = r - m$ 
```

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

```
 $L[1 \dots n_1] = A[l \dots m]$ 
```

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

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

```
 $i = j = 1$ 
```

```
for  $k = l$  to  $r$  do
```

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

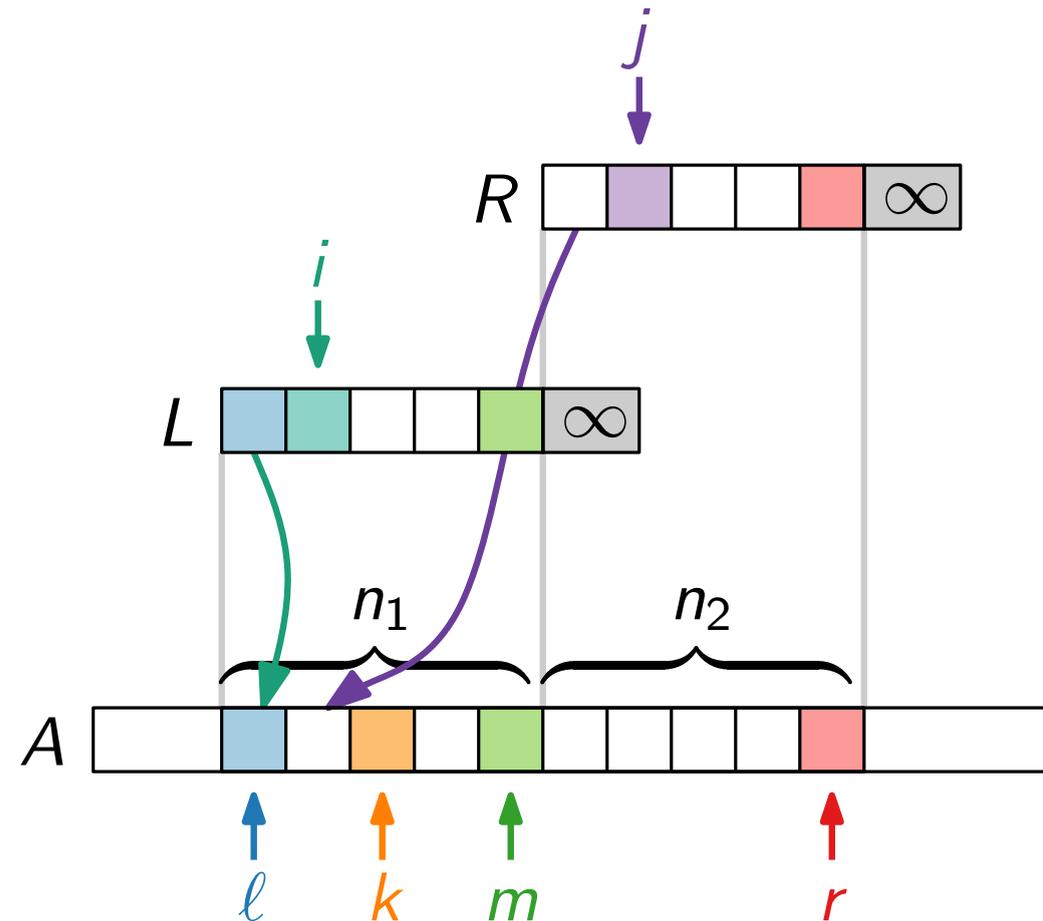
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

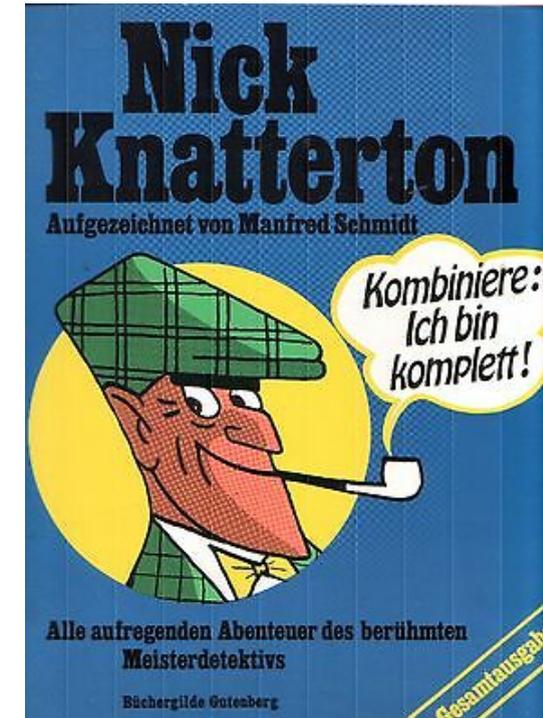
```
  else
```

```
     $A[k] = R[j]$ 
```

```
     $j = j + 1$ 
```



Aber... stimmt das denn alles?



# MergeSort – ein Beispiel

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

```
if  $\ell < r$  then
```

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

```
    MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    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,  $\ell$ ,  $m$ ) } herrsche  
    MERGESORT(A,  $m + 1$ ,  $r$ ) }  
    MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```

## Demo.

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

# MergeSort – ein Beispiel

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

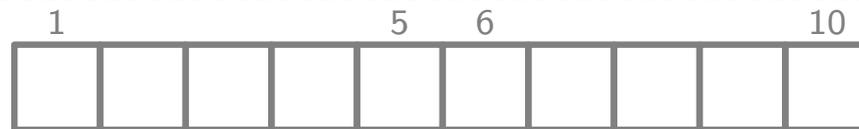
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  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,  $\ell$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    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  $r = A.length$ )
```

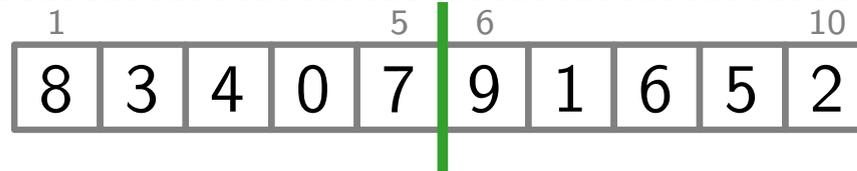
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

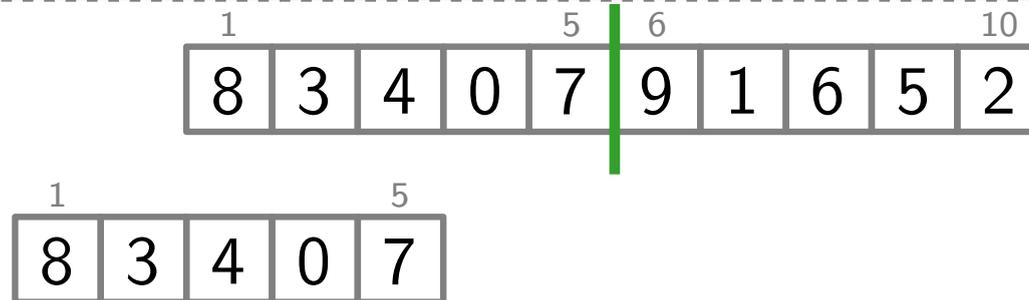
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

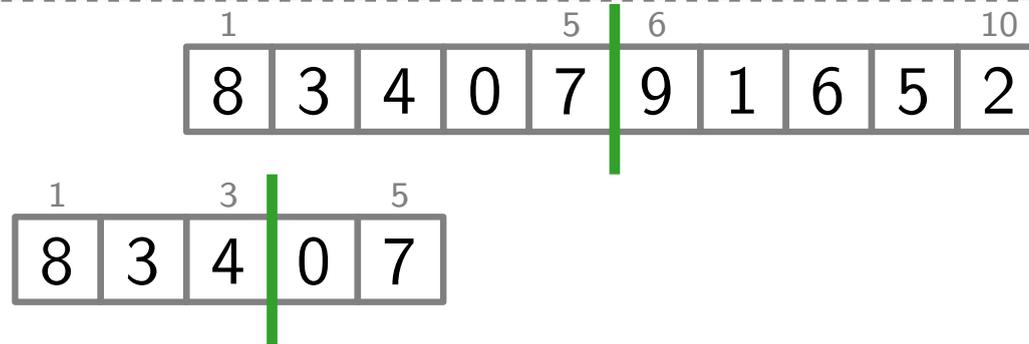
```
if  $\ell < r$  then
```

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

```
    MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

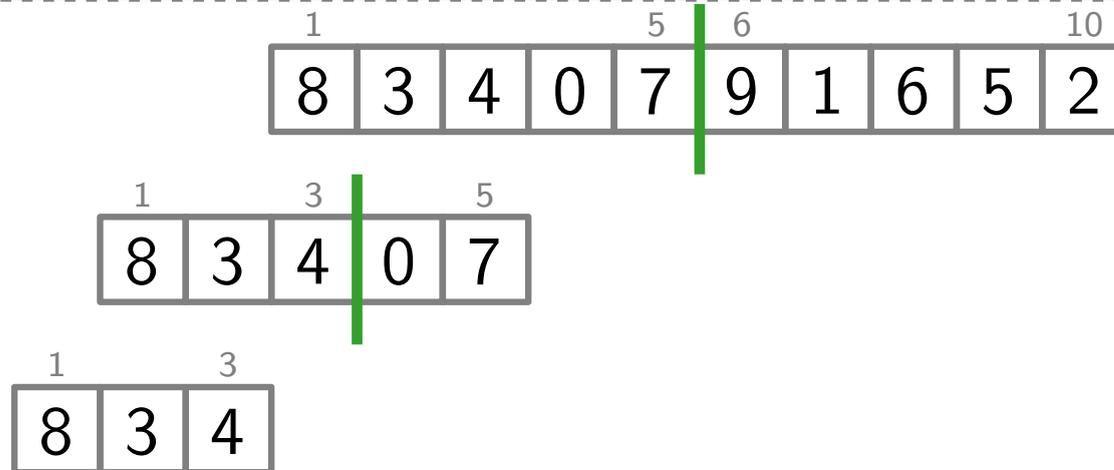
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

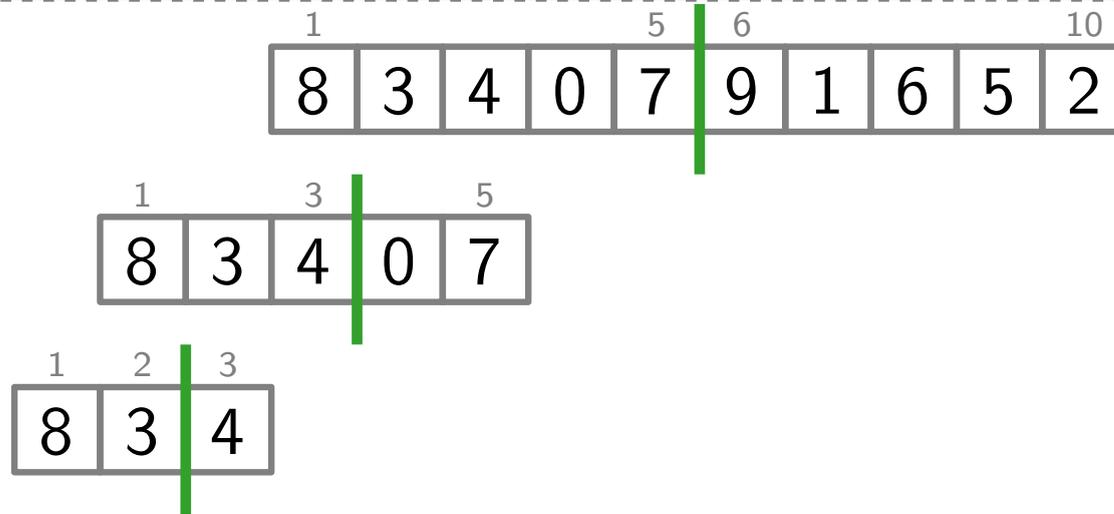
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

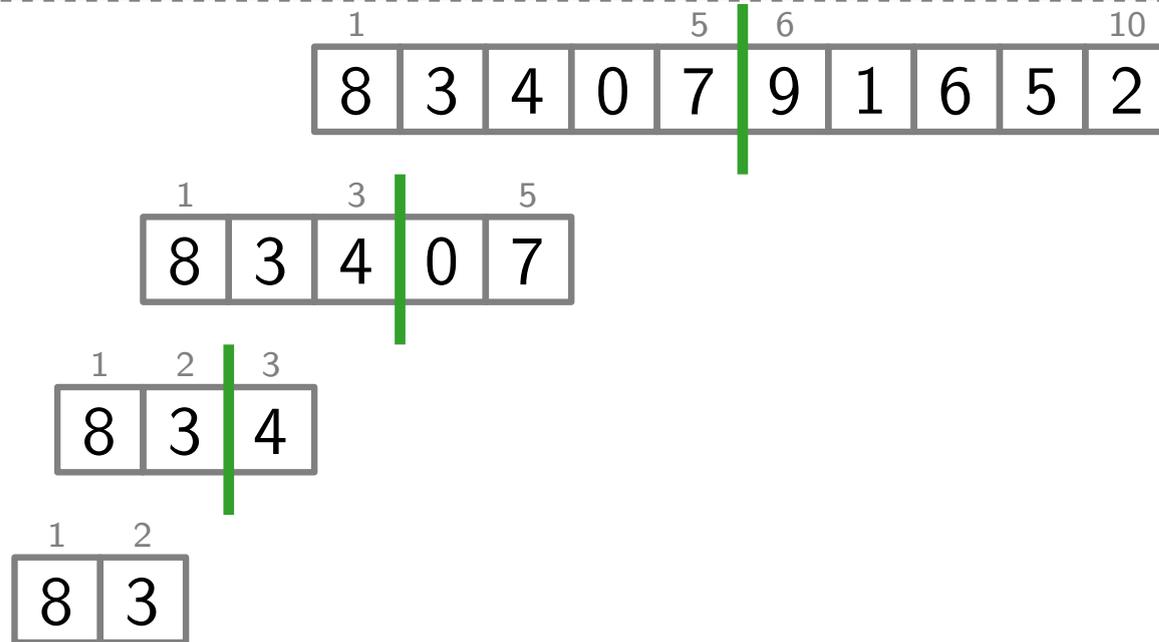
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

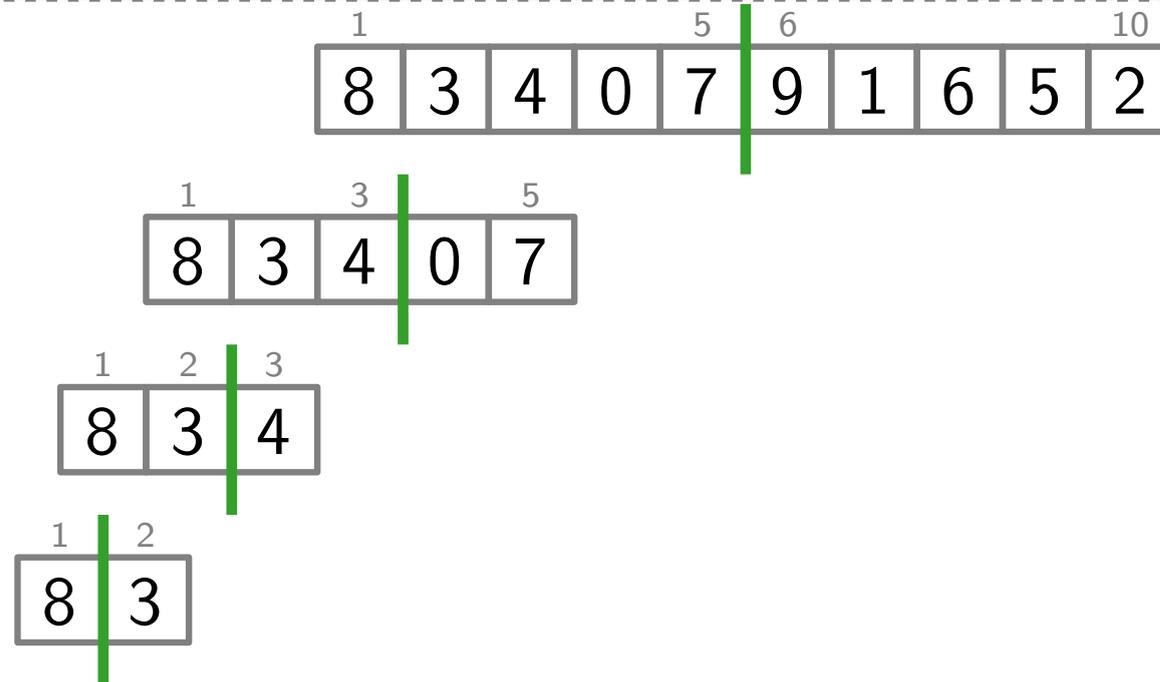
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

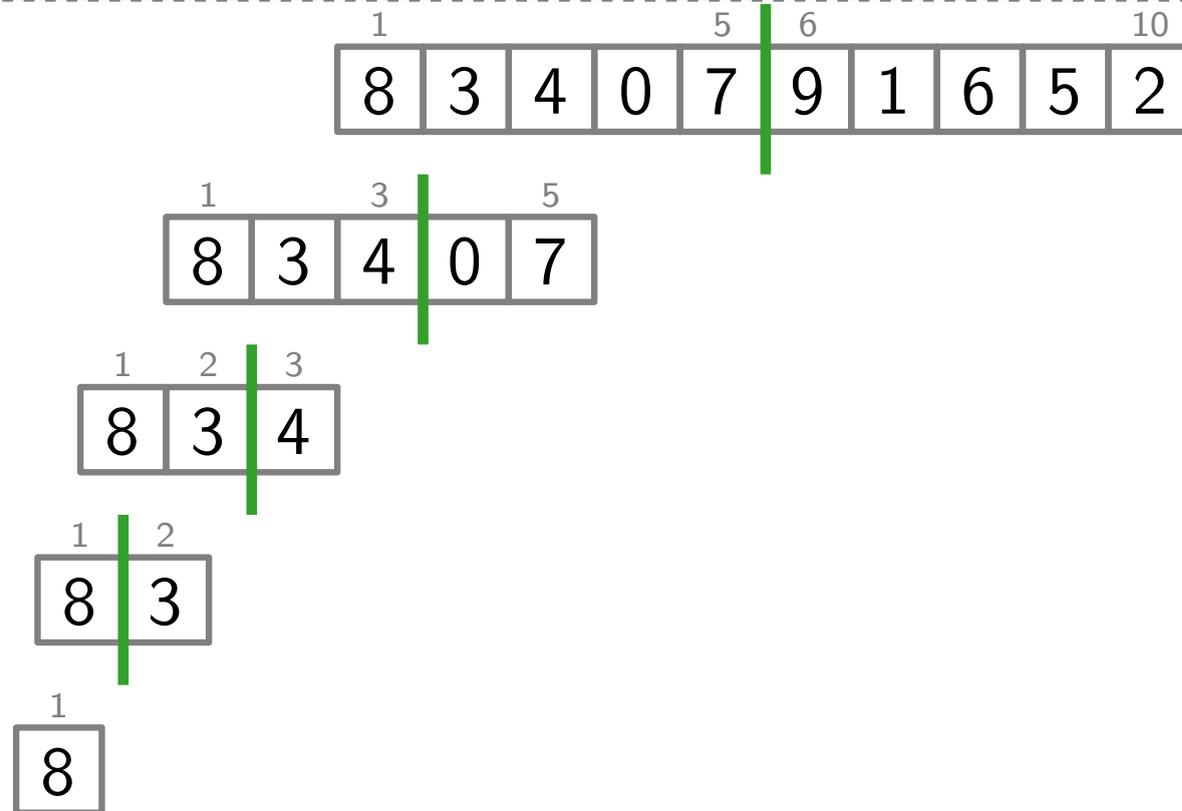
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

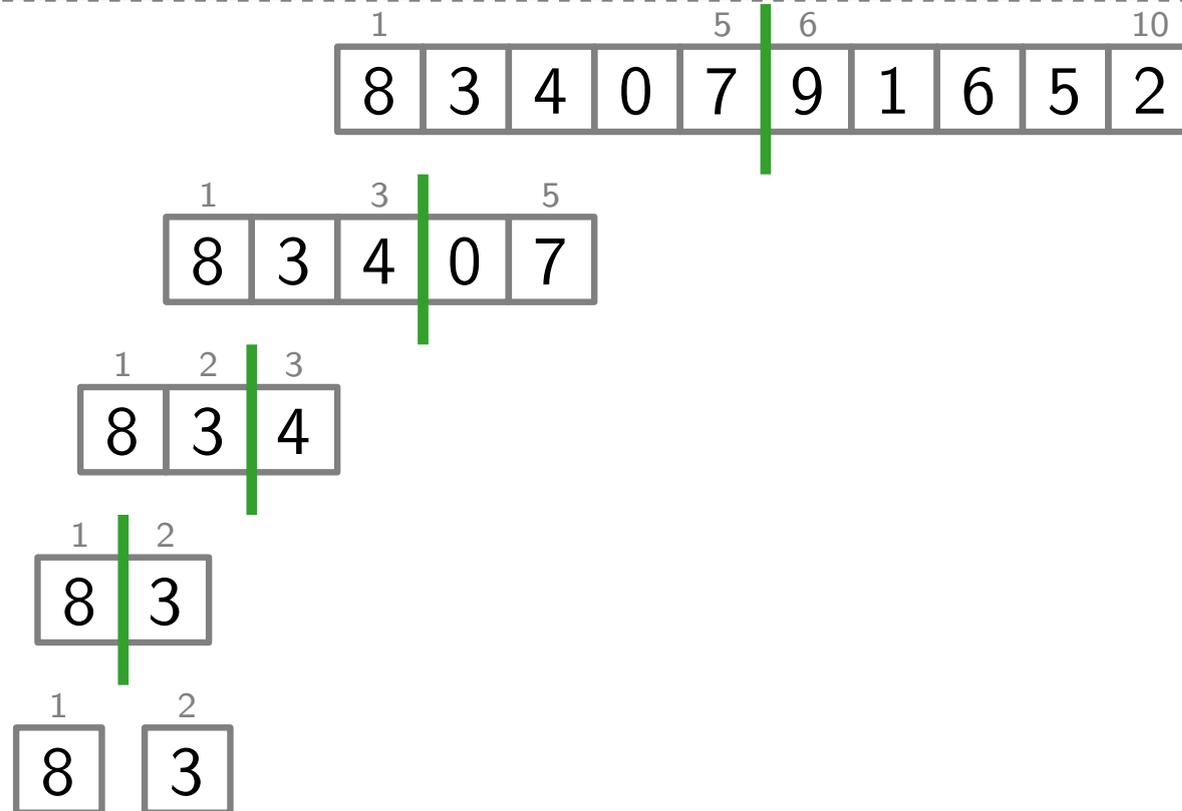
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

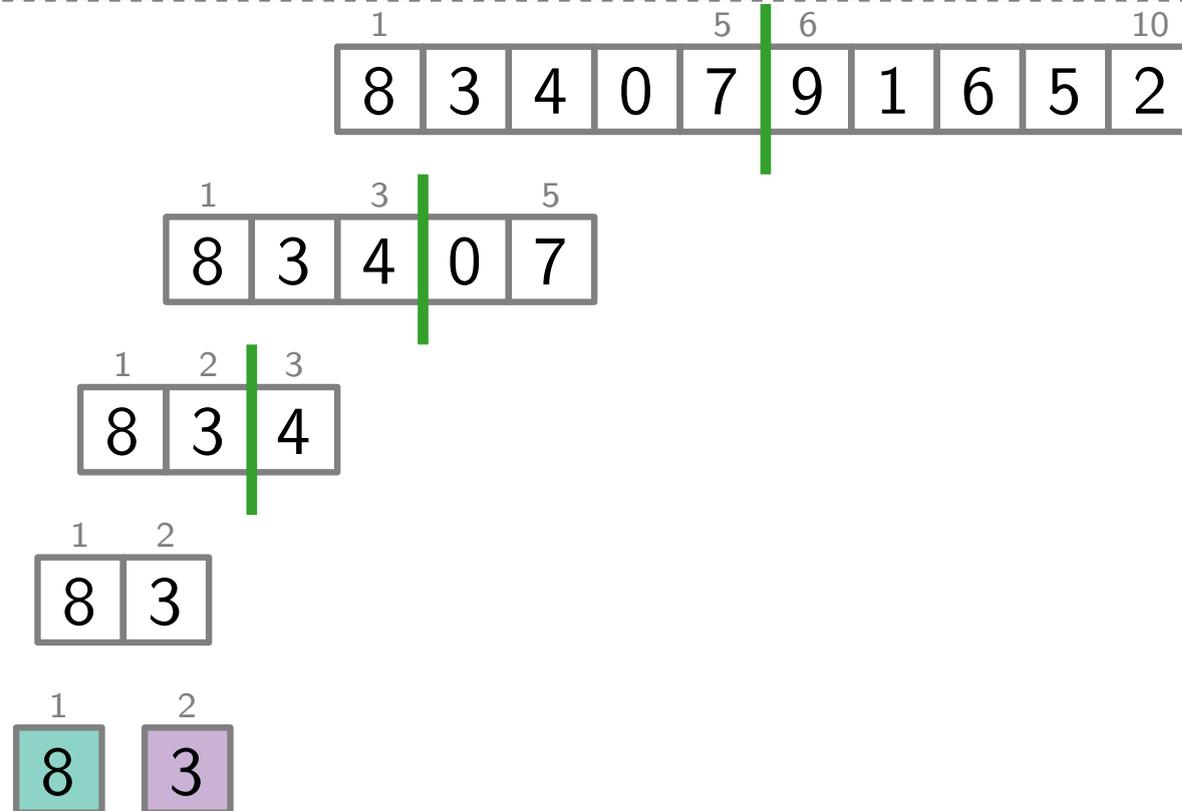
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

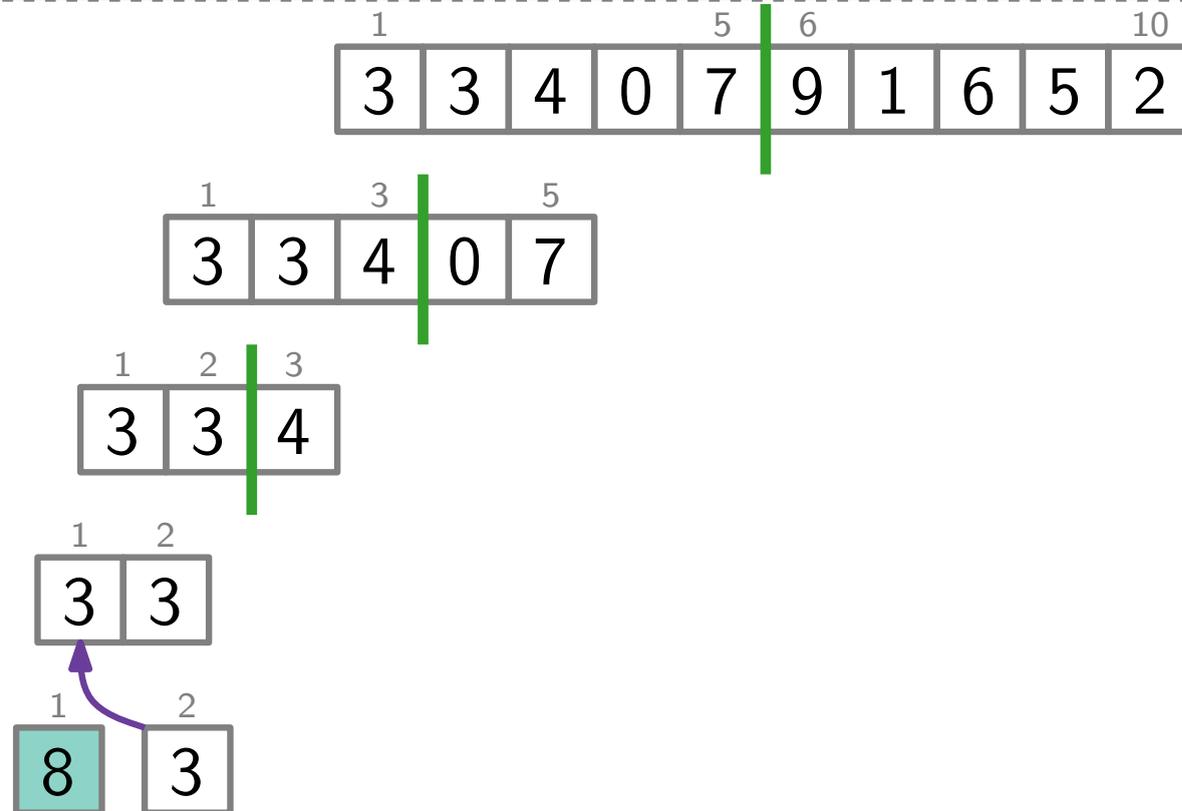
```
if  $\ell < r$  then
```

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

```
    MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

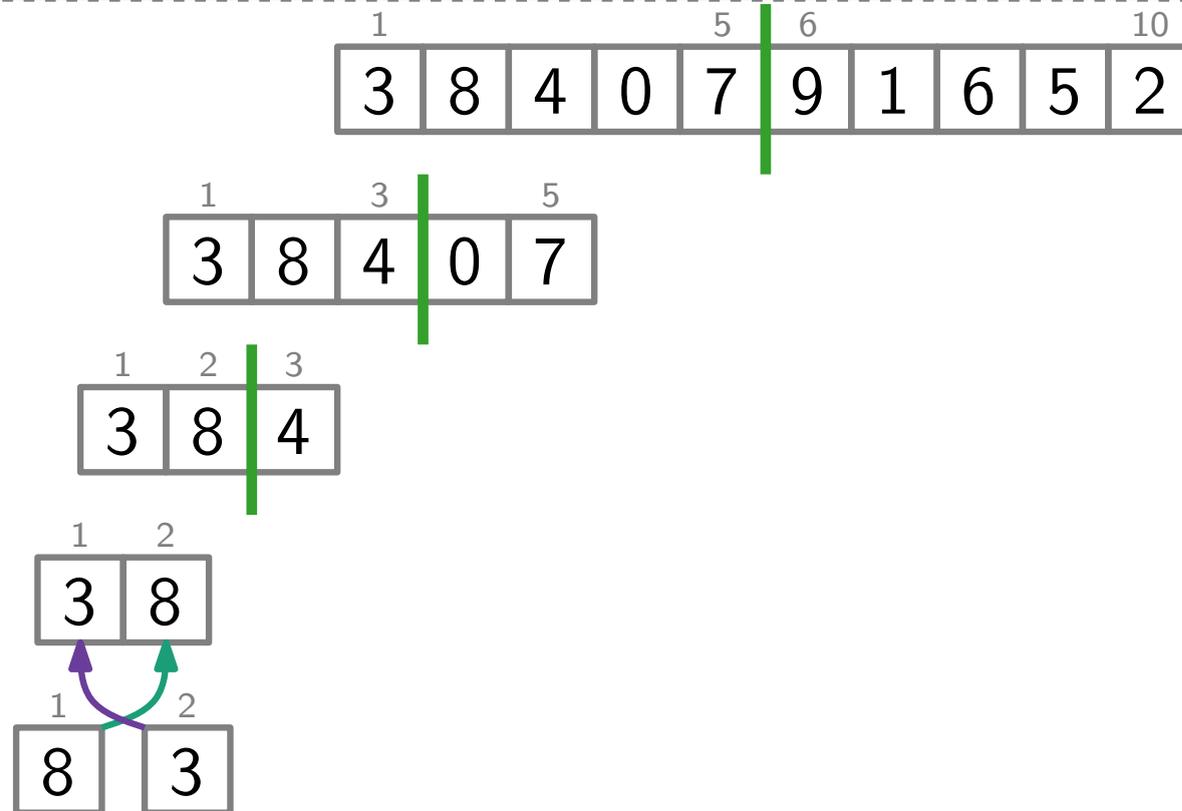
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

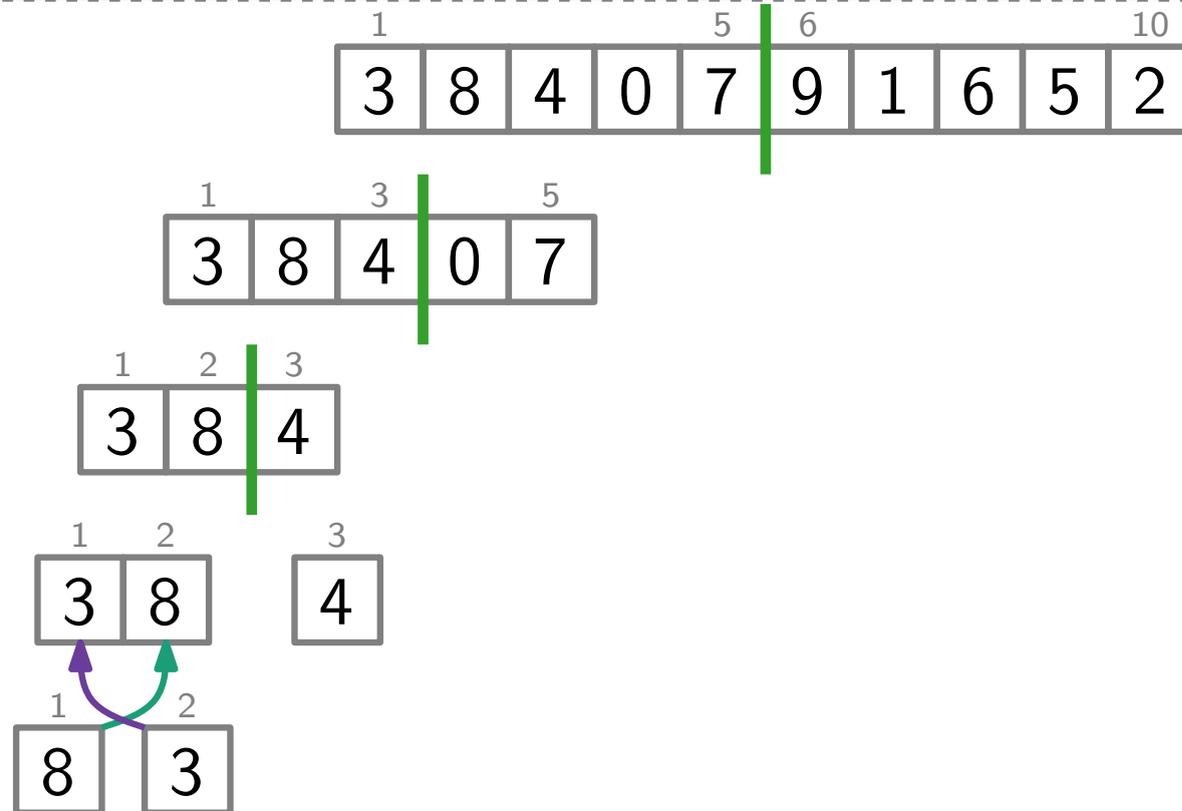
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

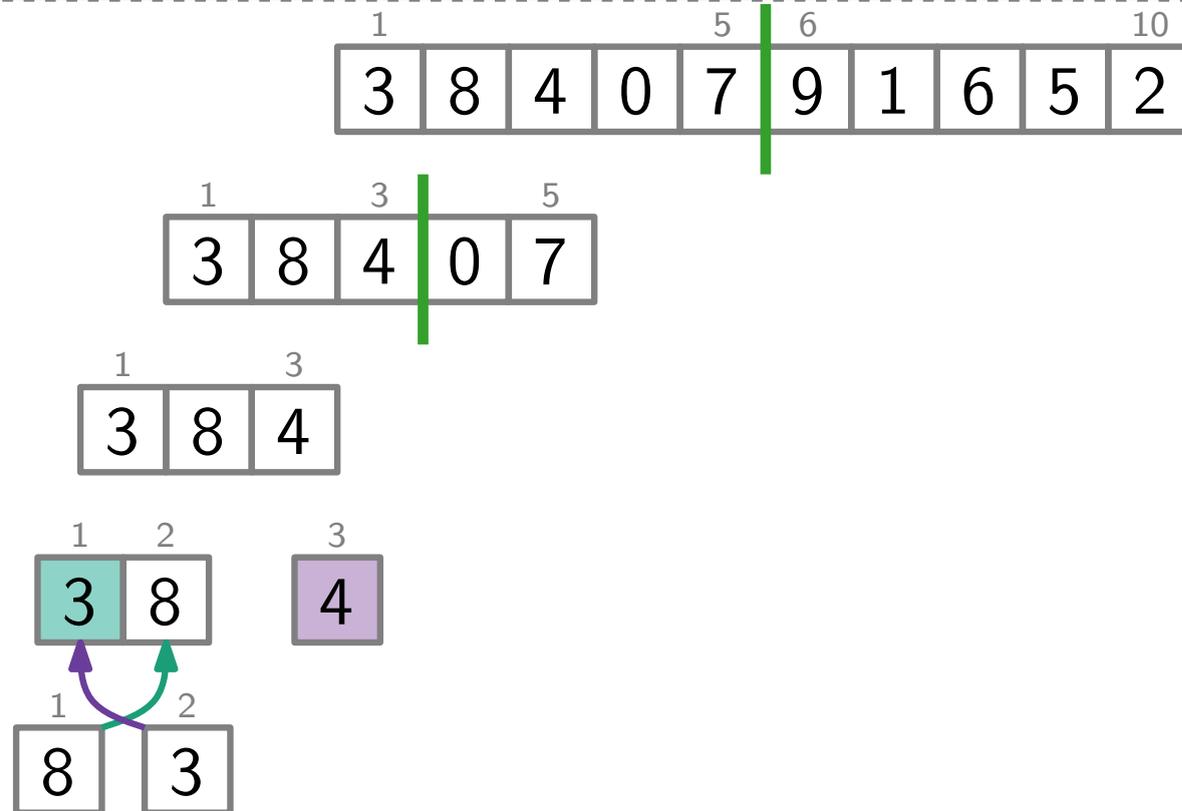
```
if  $l < r$  then
```

```
   $m = \lfloor (l + r) / 2 \rfloor$  } teile
```

```
  MERGESORT(A,  $l$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

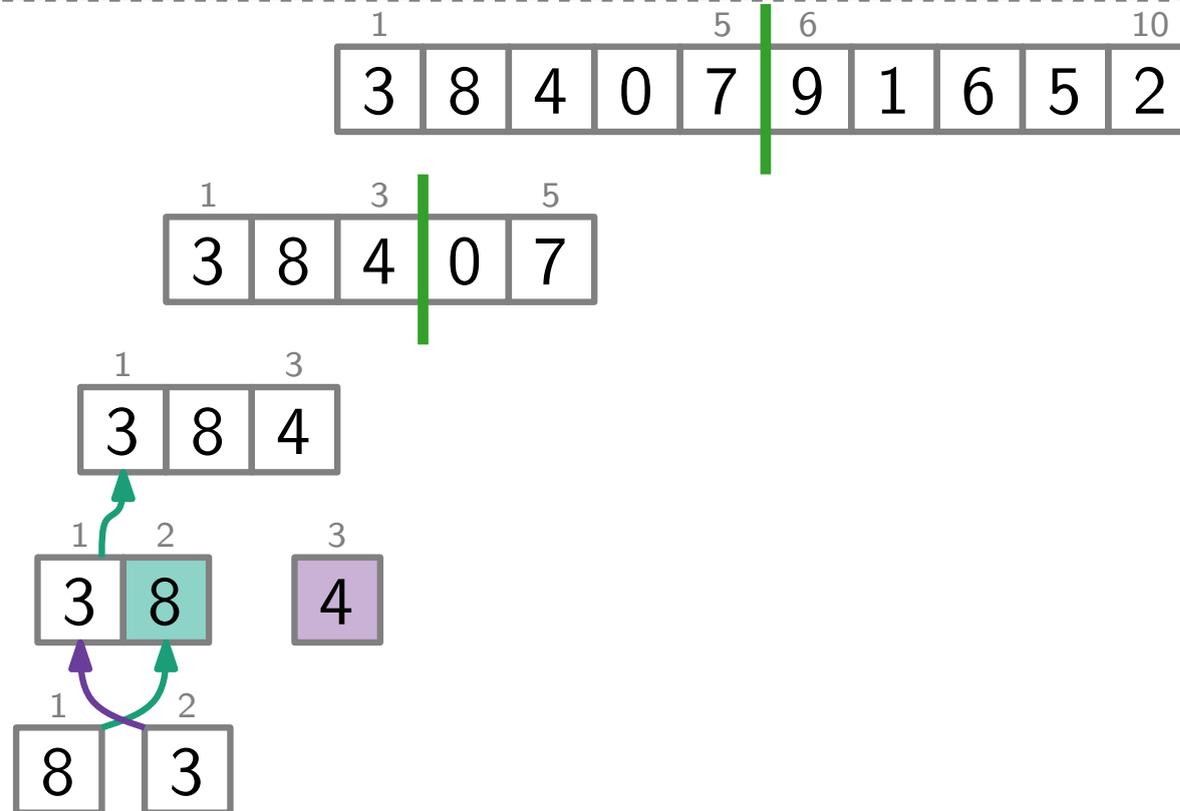
```
if  $l < r$  then
```

```
   $m = \lfloor (l + r) / 2 \rfloor$  } 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  $r = A.length$ )
```

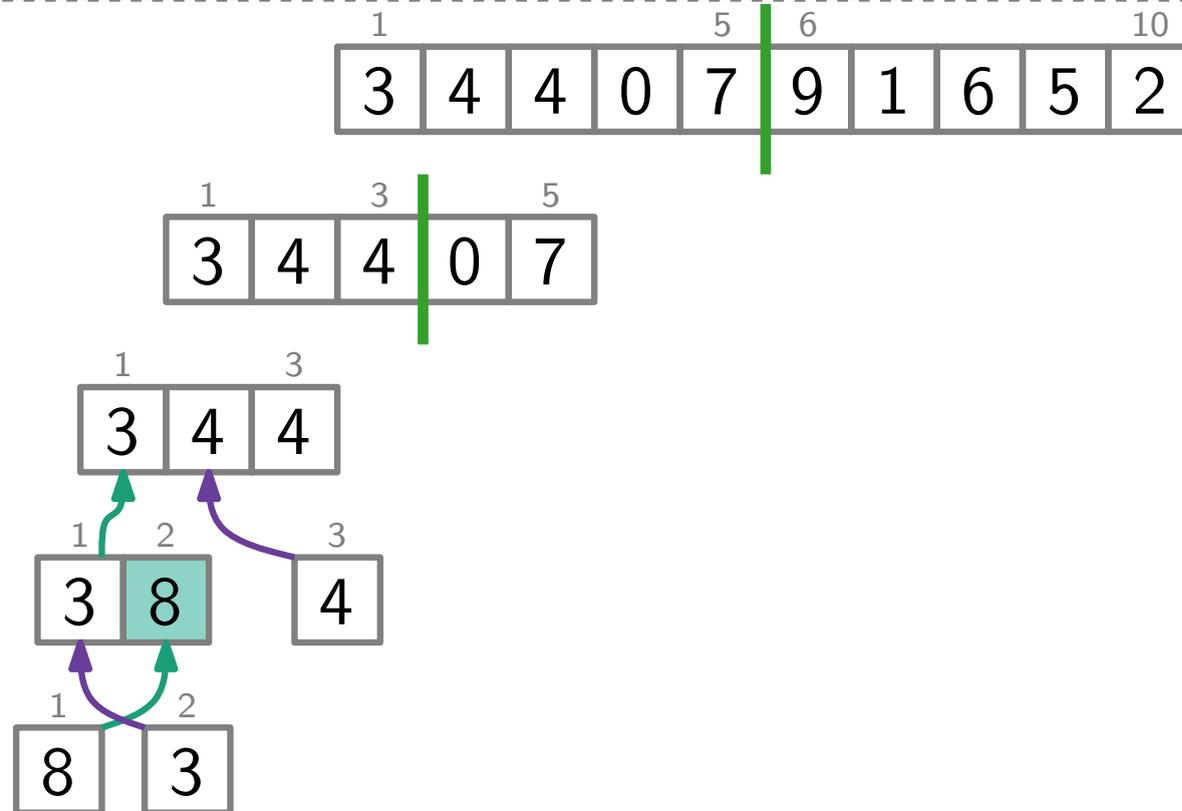
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

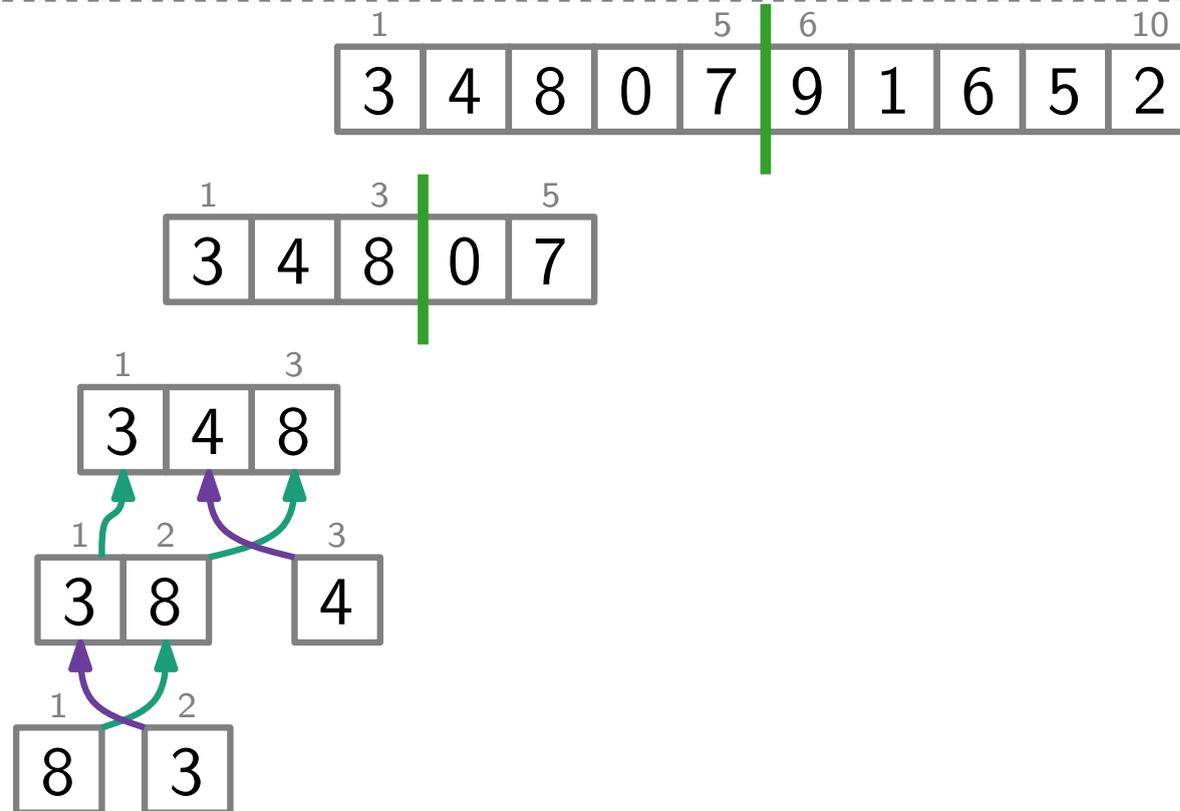
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

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

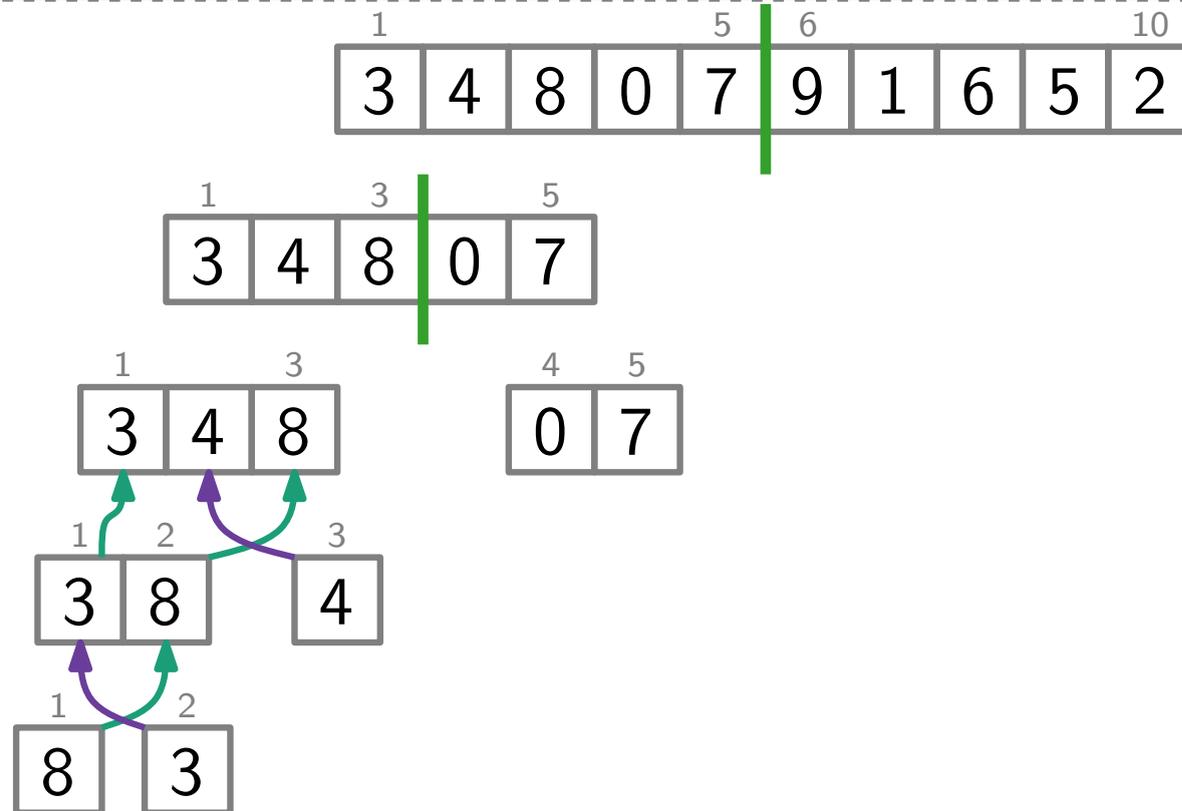
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

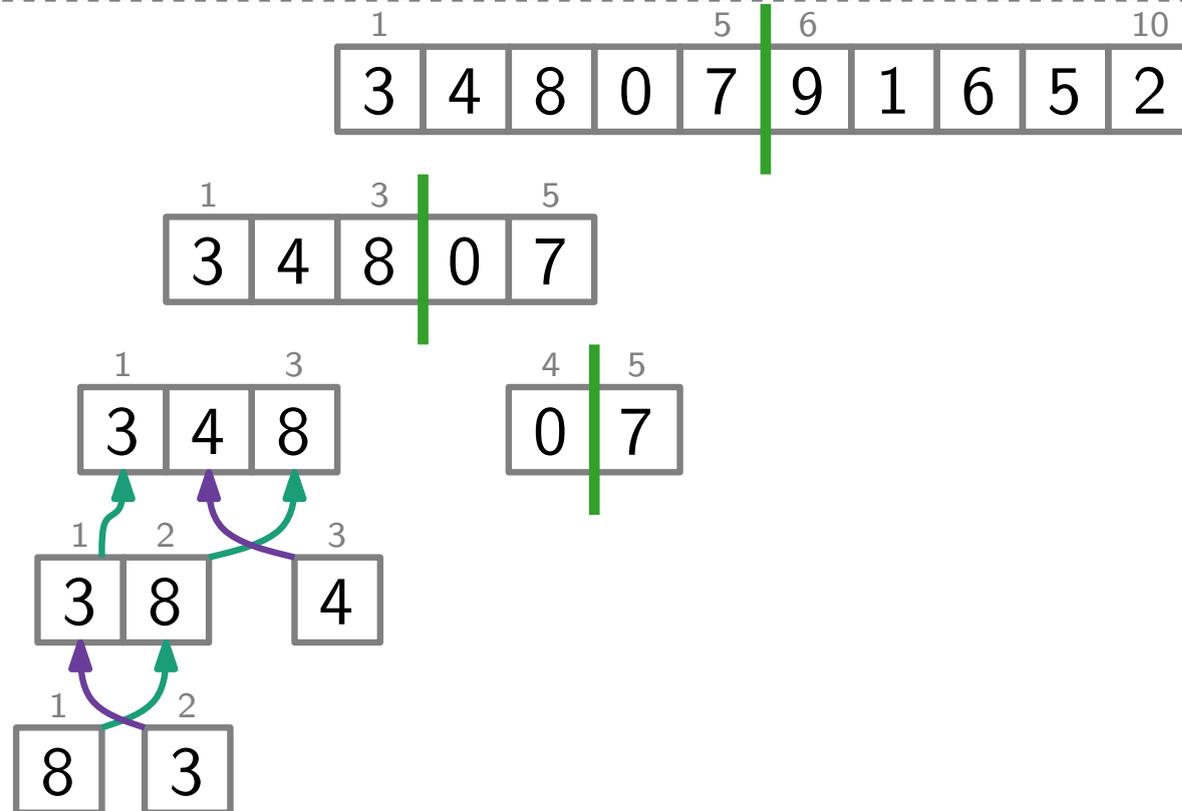
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

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

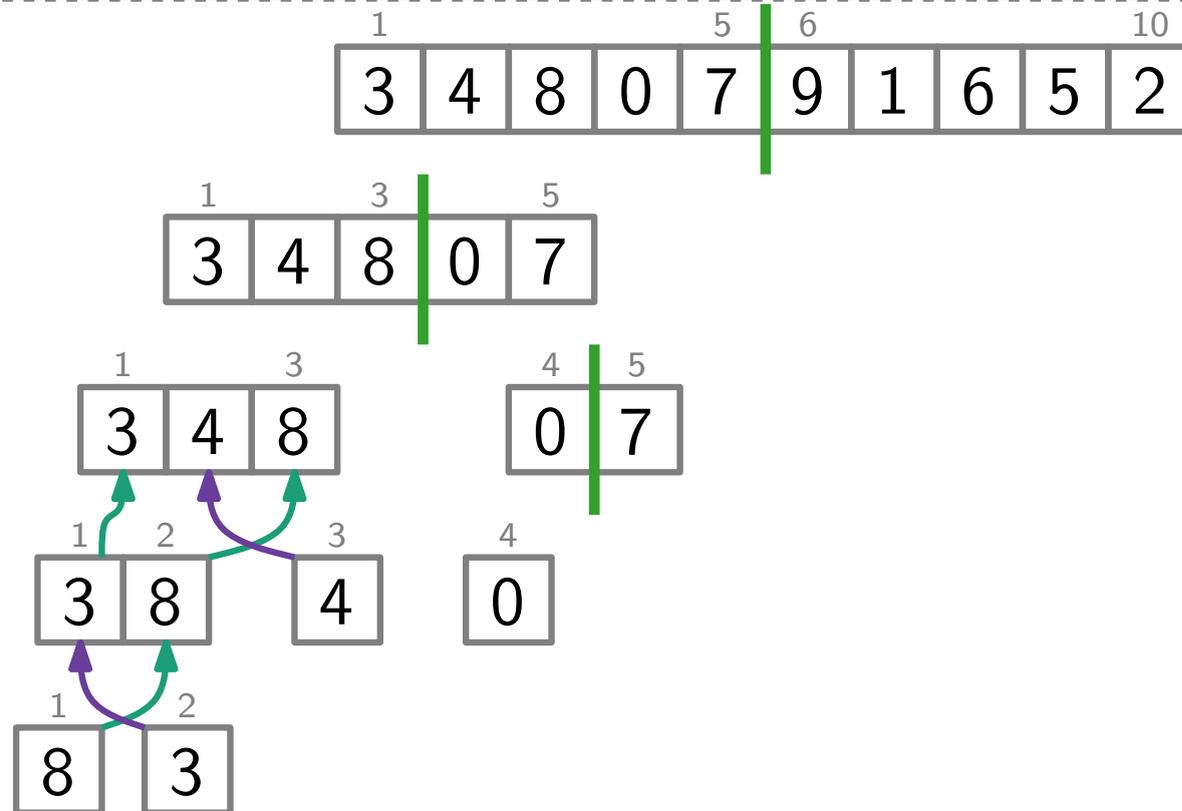
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

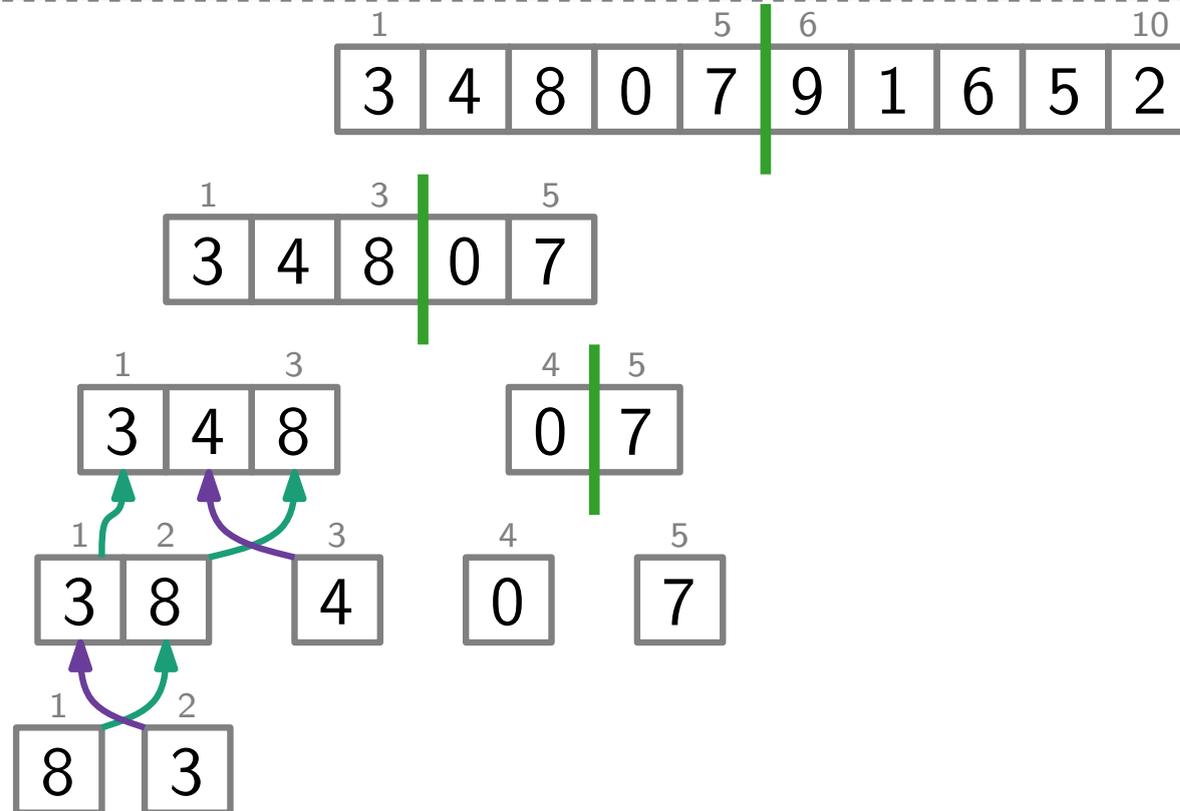
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

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

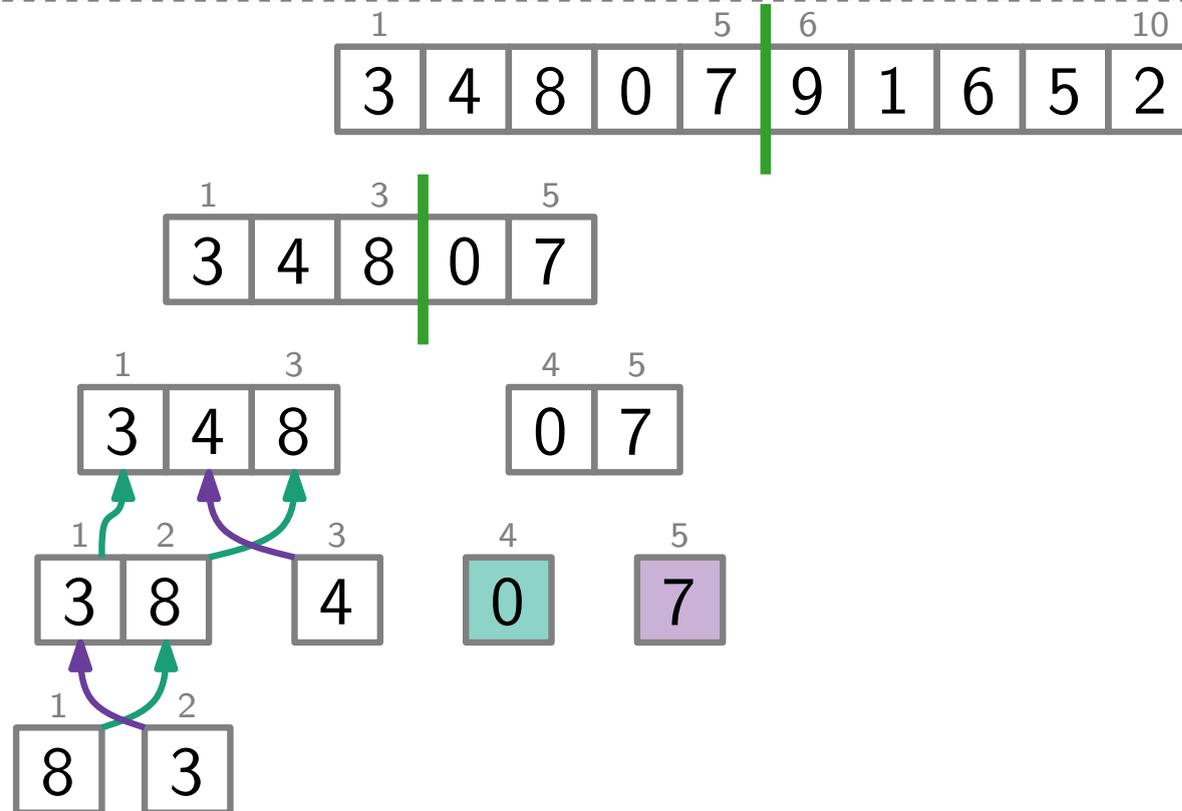
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

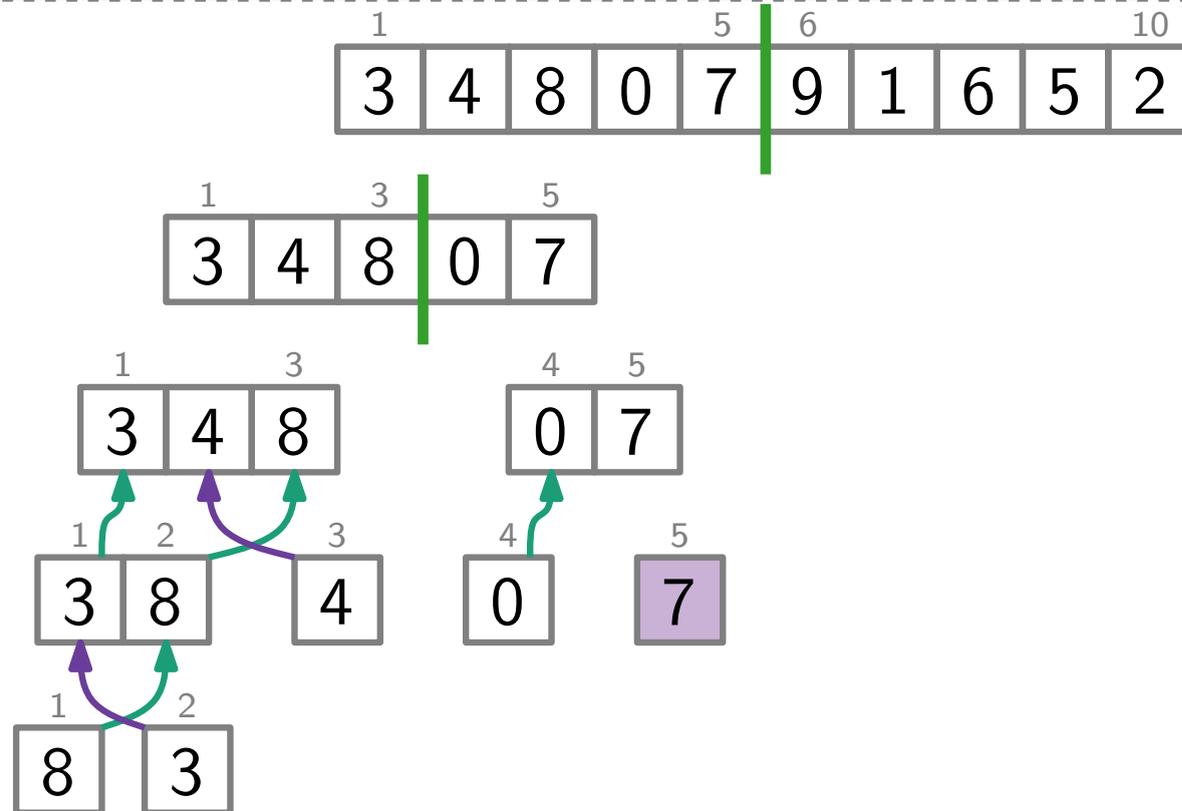
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

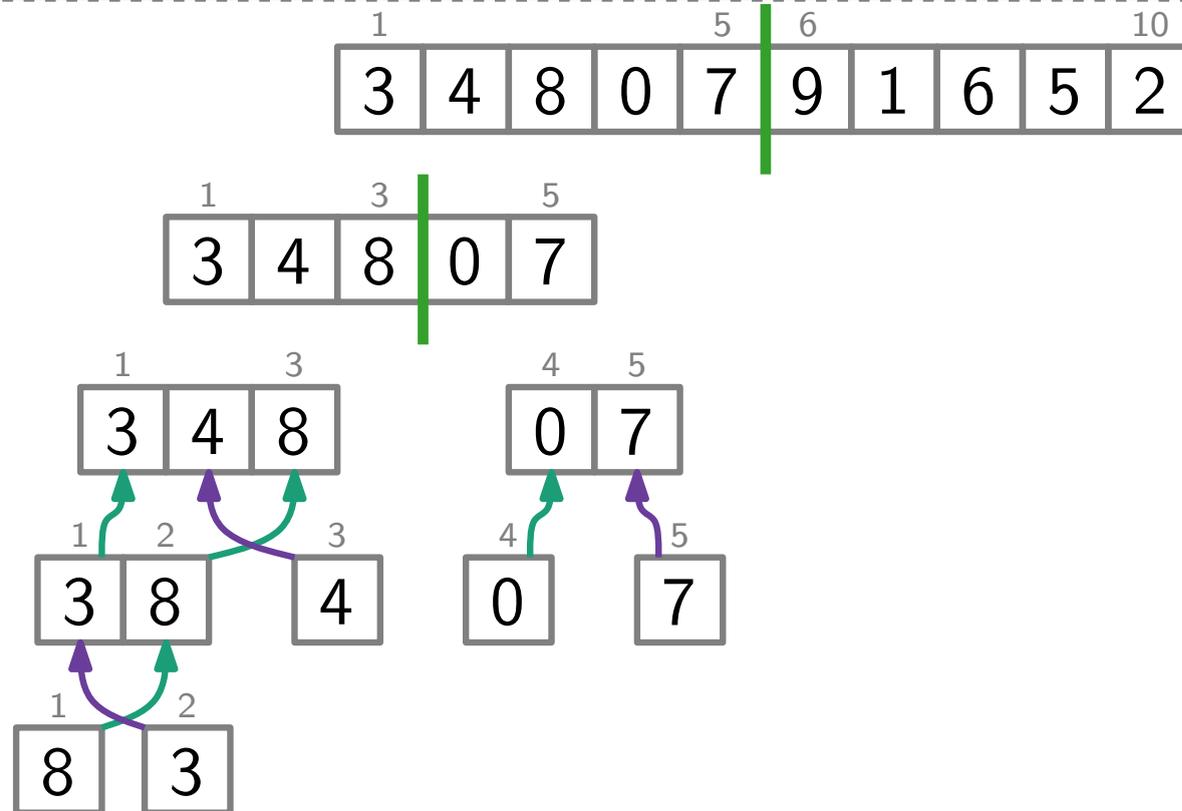
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

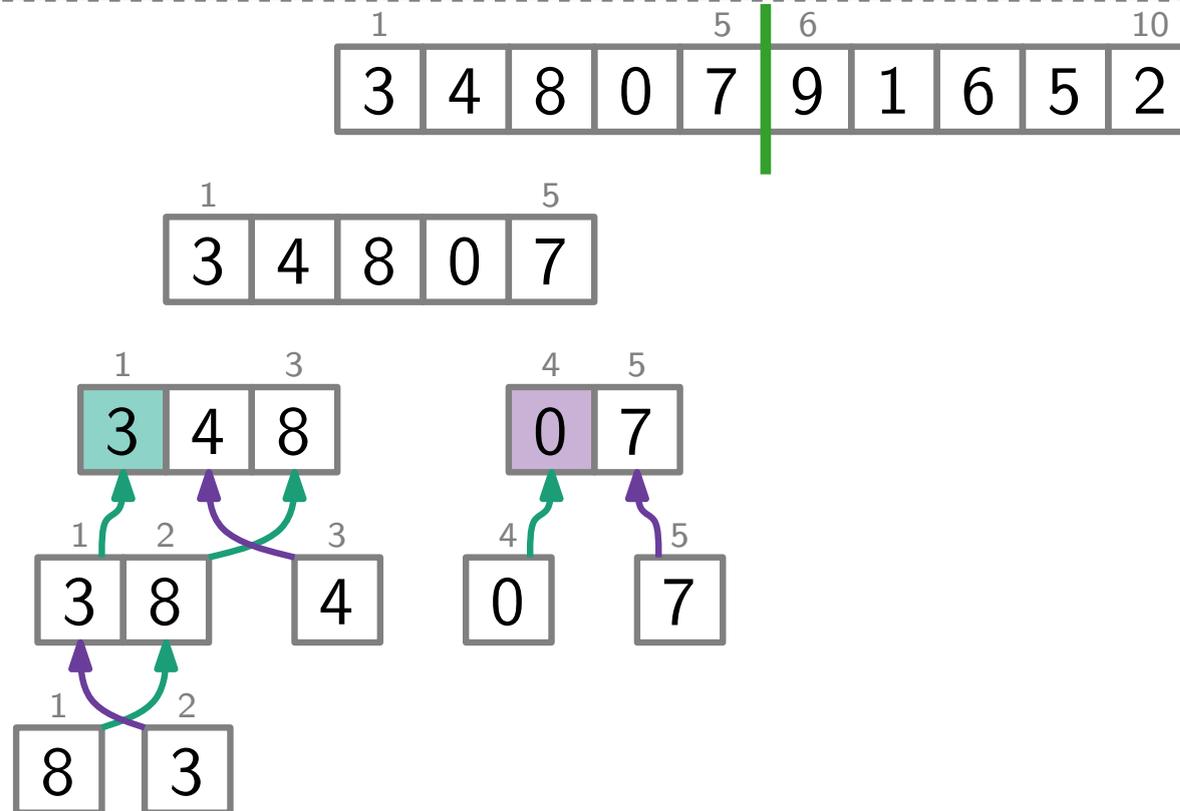
```
if  $l < r$  then
```

```
     $m = \lfloor (l + r) / 2 \rfloor$  } teile
```

```
    MERGESORT(A,  $l$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

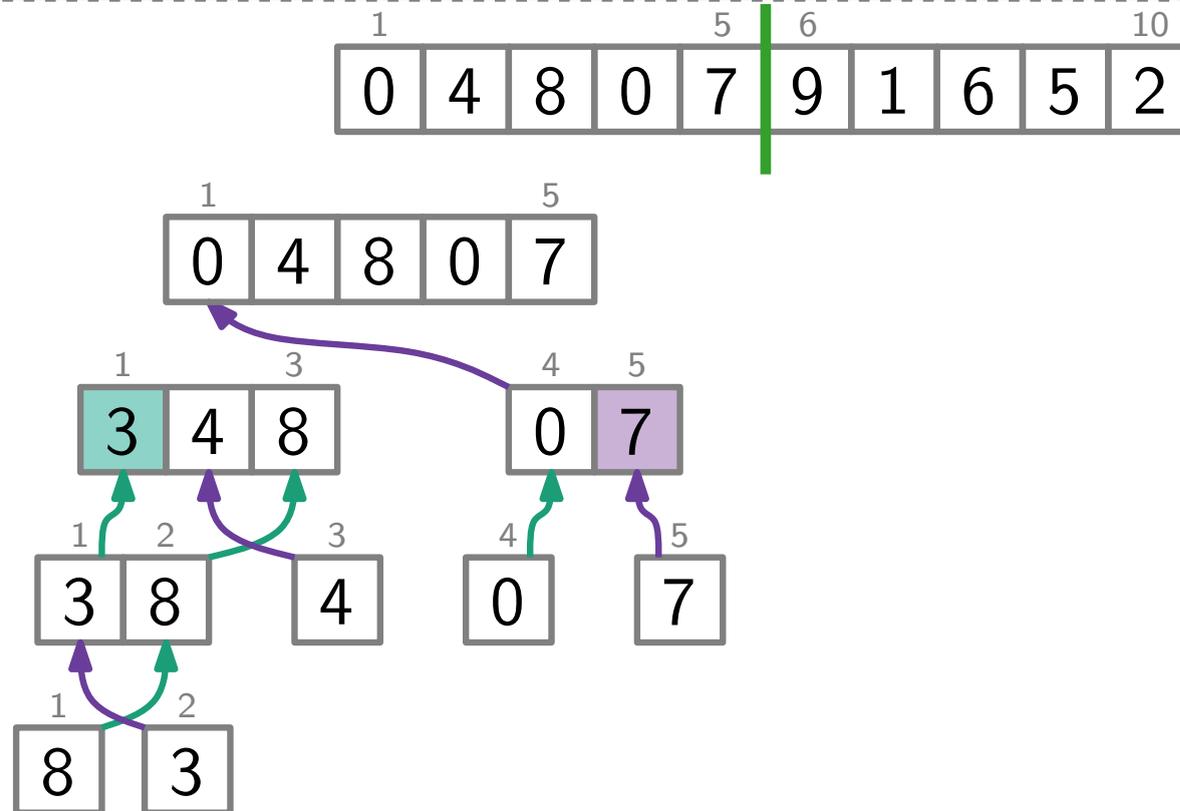
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

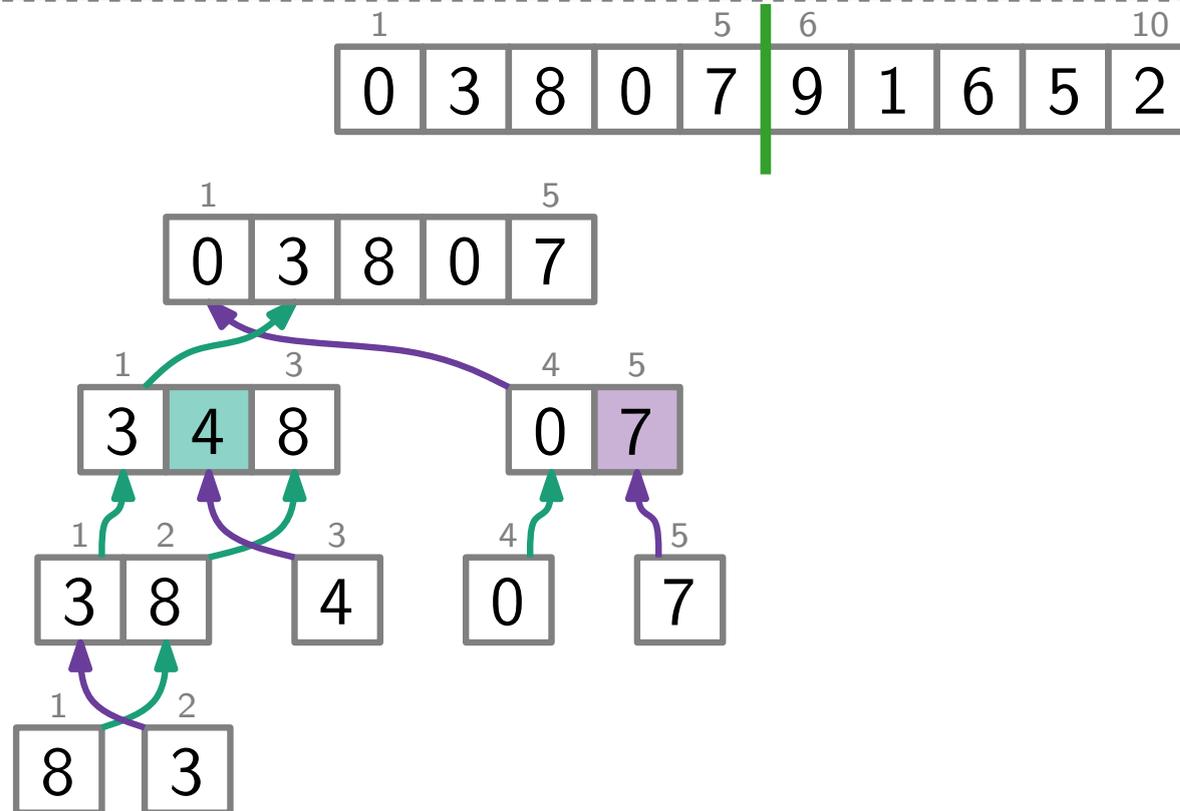
```
if  $l < r$  then
```

```
     $m = \lfloor (l + r) / 2 \rfloor$  } teile
```

```
    MERGESORT(A,  $l$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

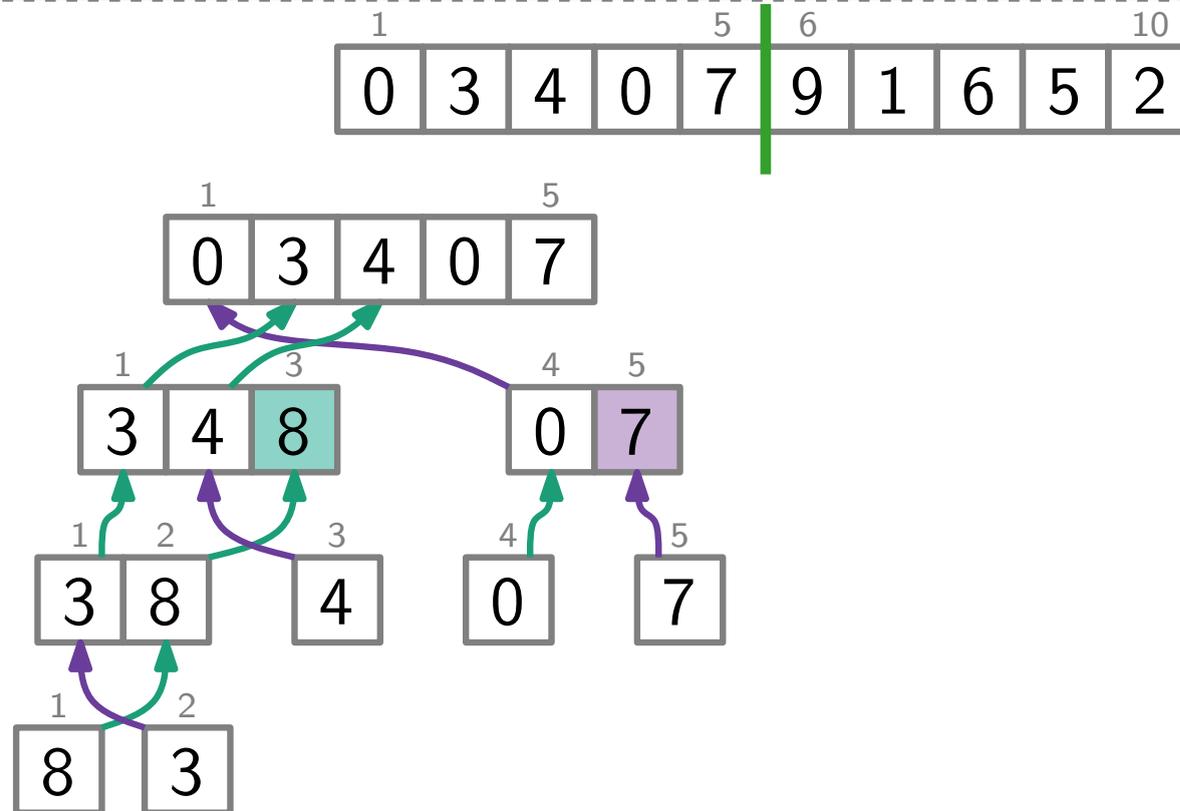
```
if  $l < r$  then
```

```
   $m = \lfloor (l + r) / 2 \rfloor$  } teile
```

```
  MERGESORT(A,  $l$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

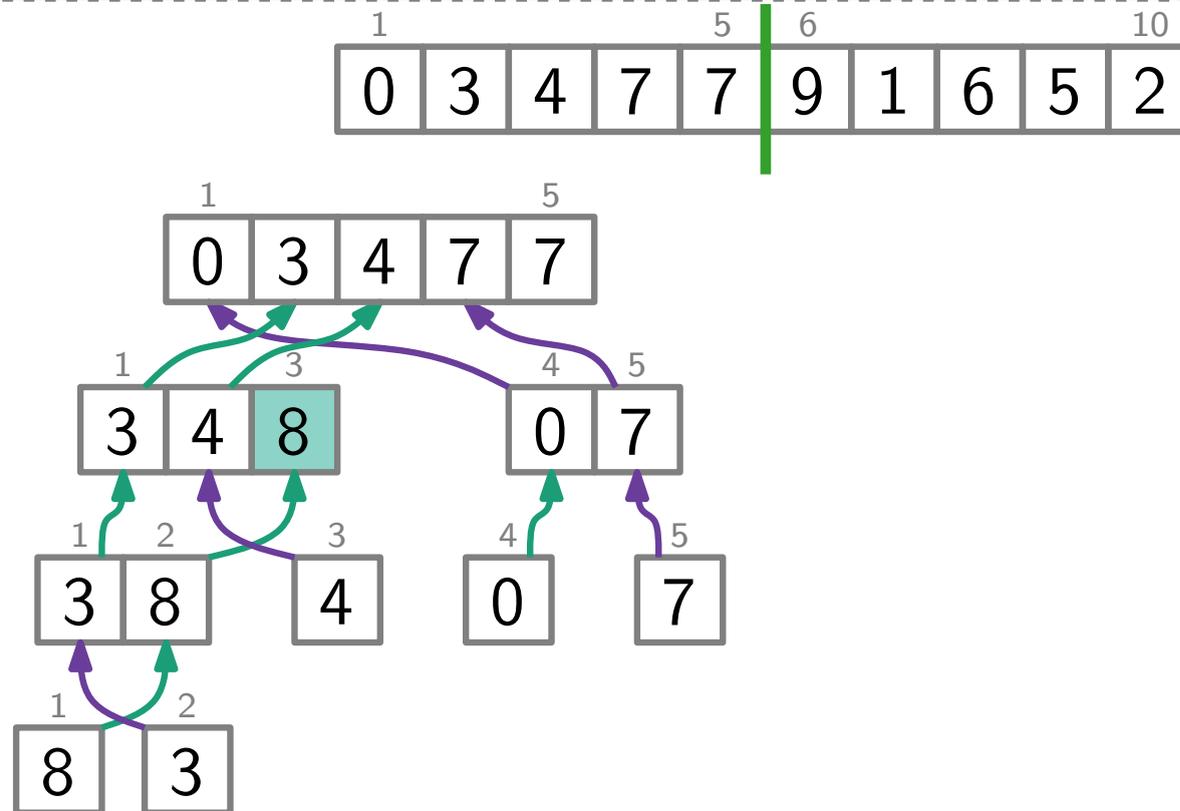
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

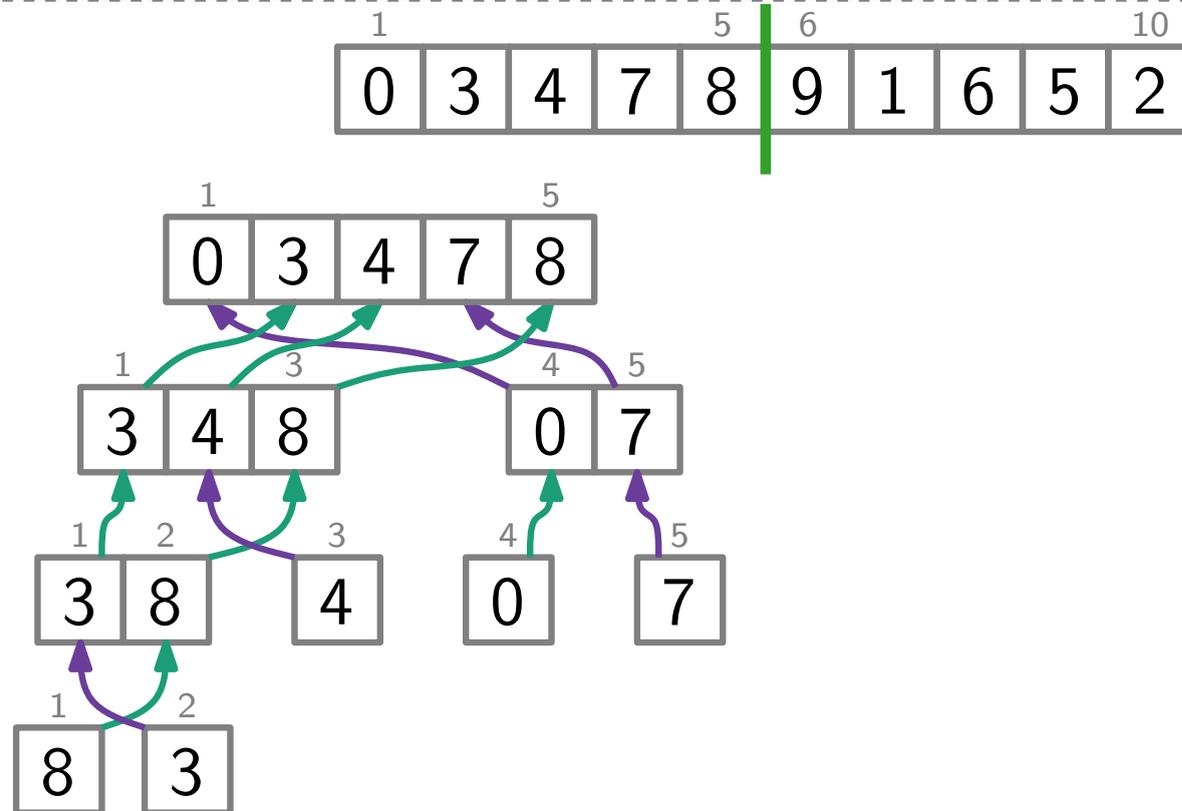
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

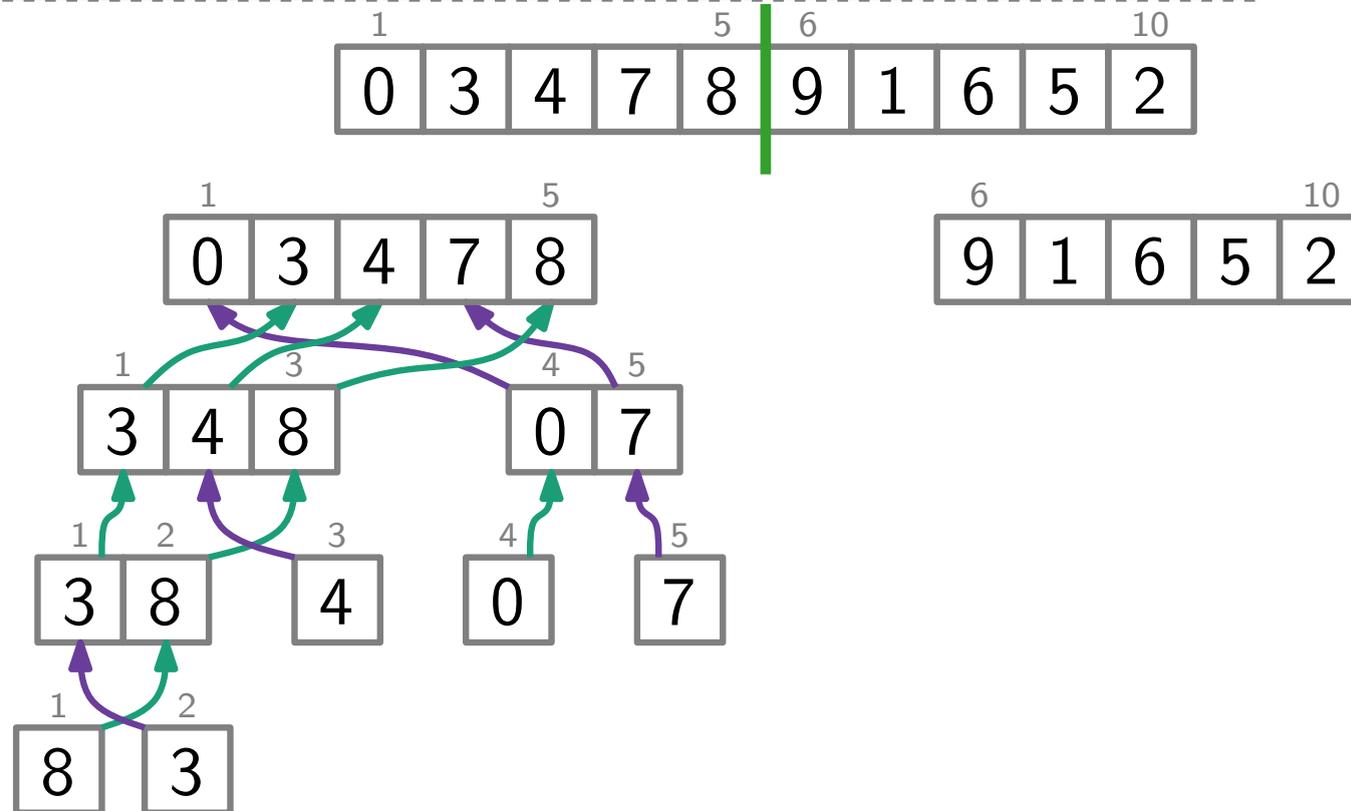
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

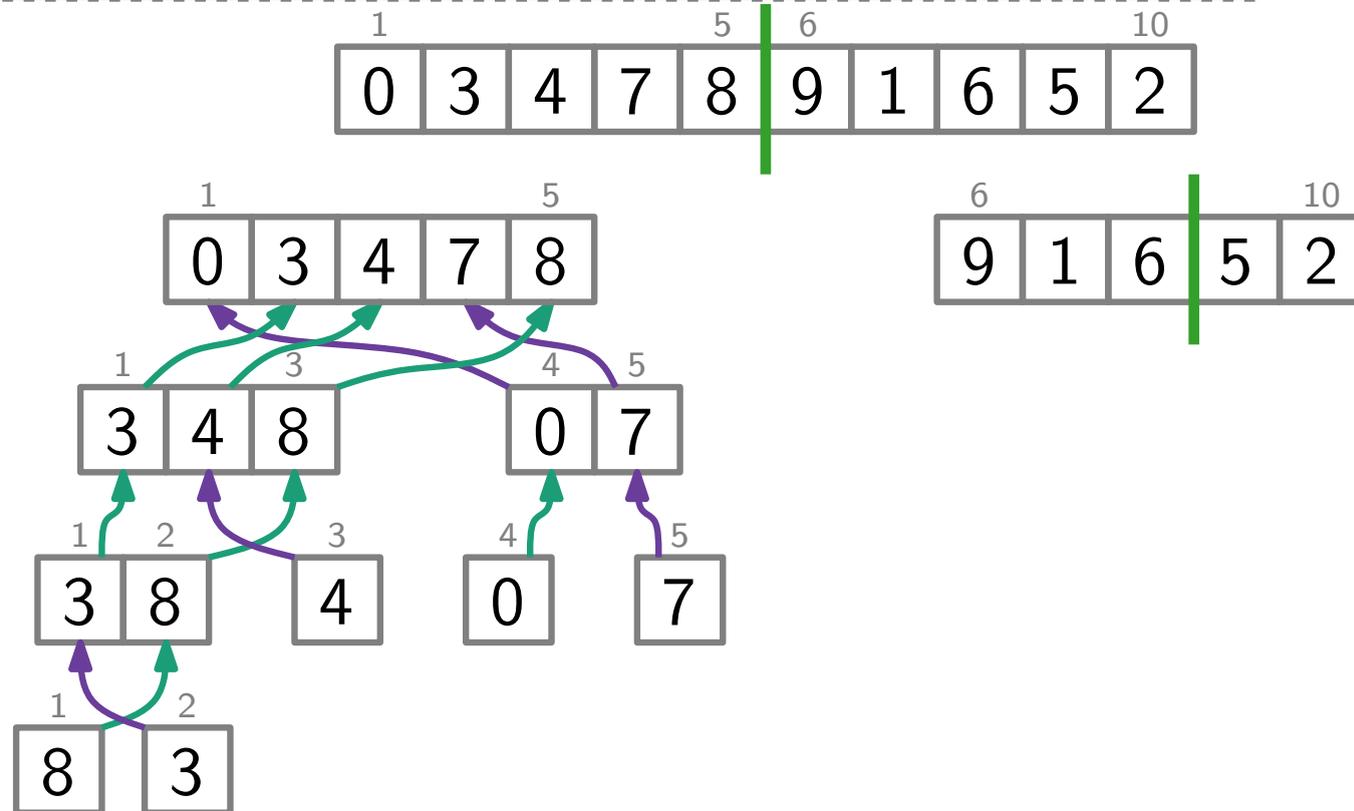
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

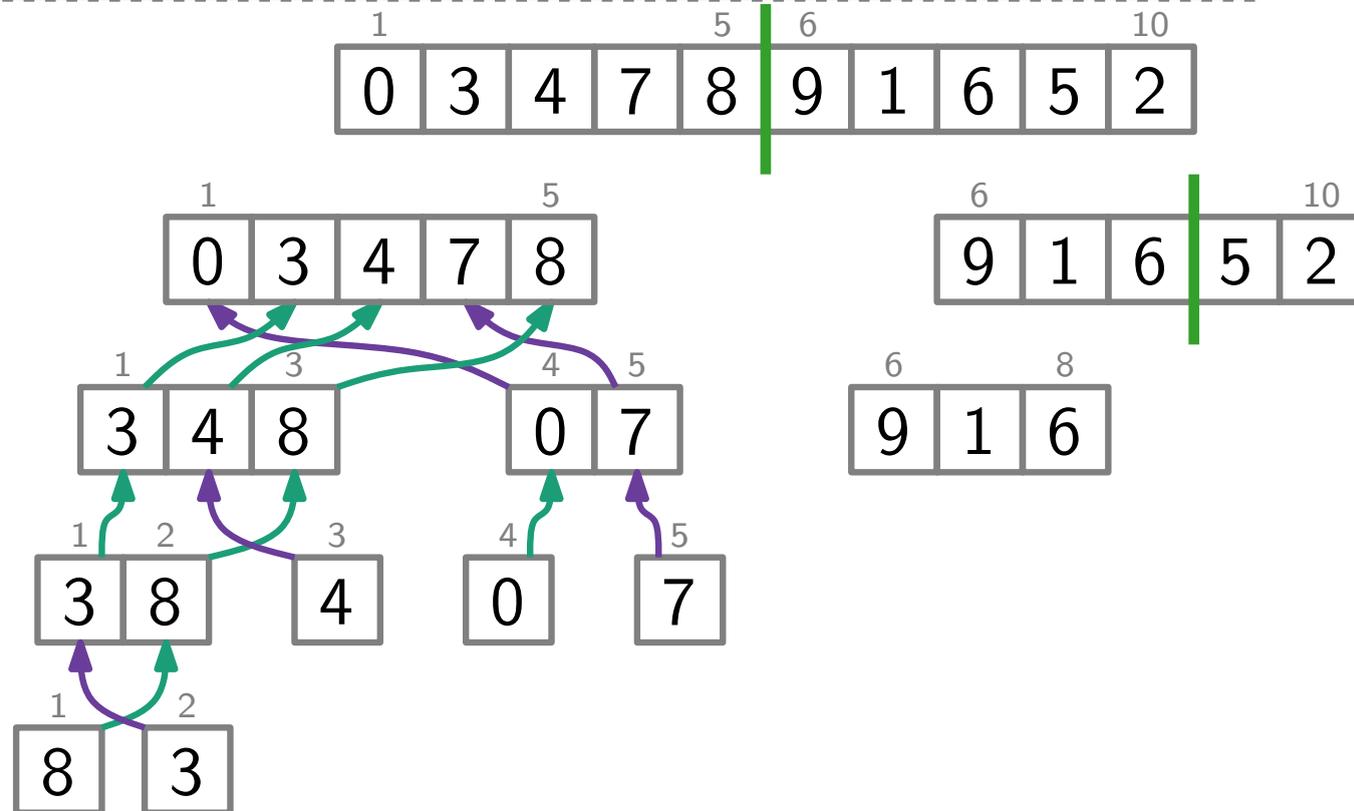
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

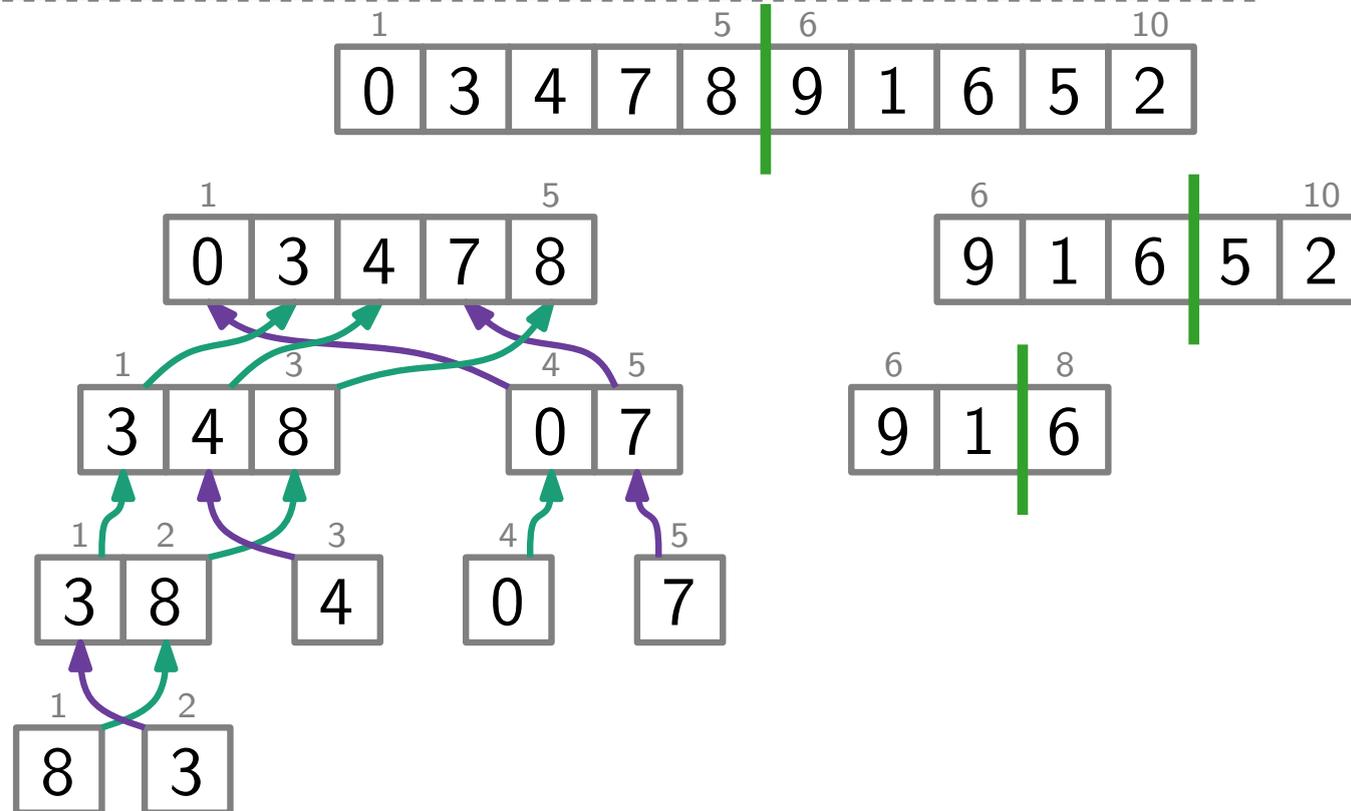
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

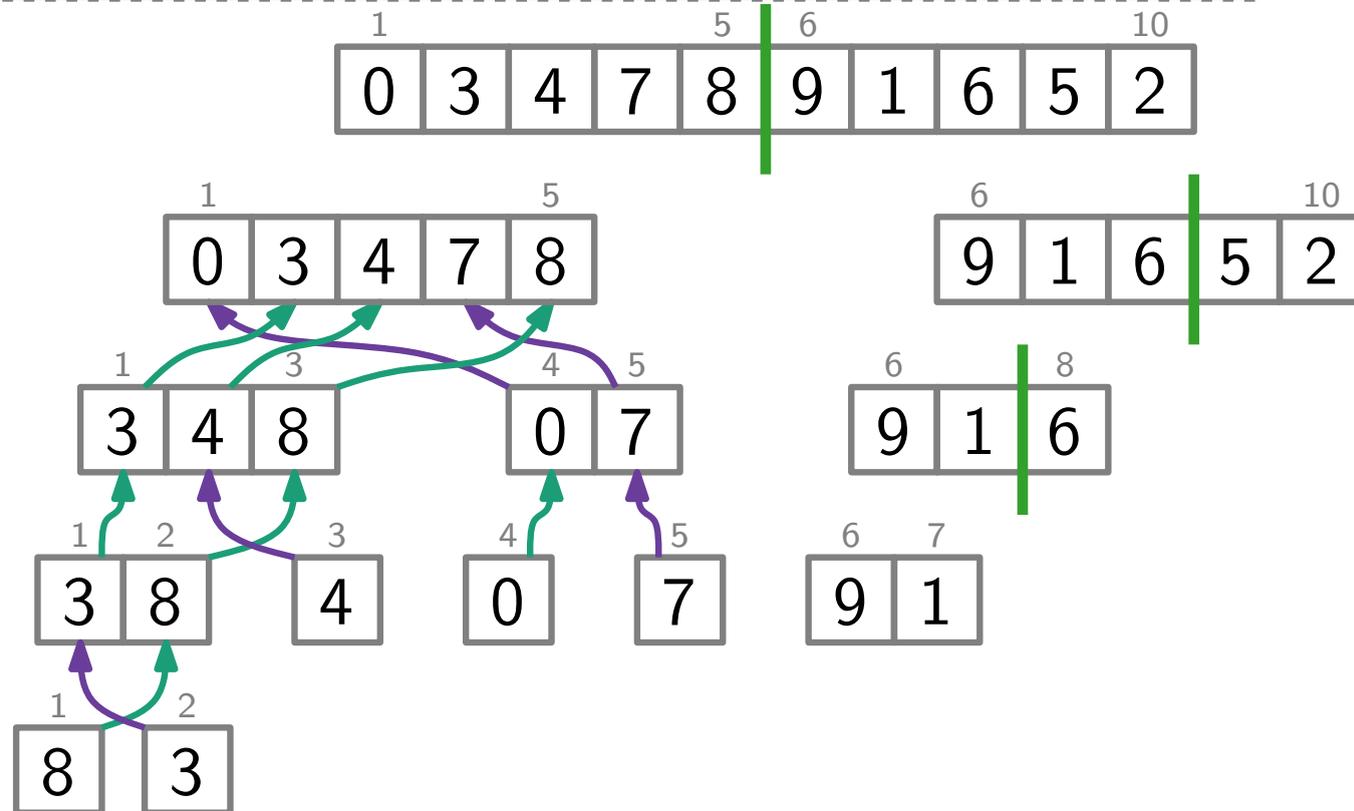
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

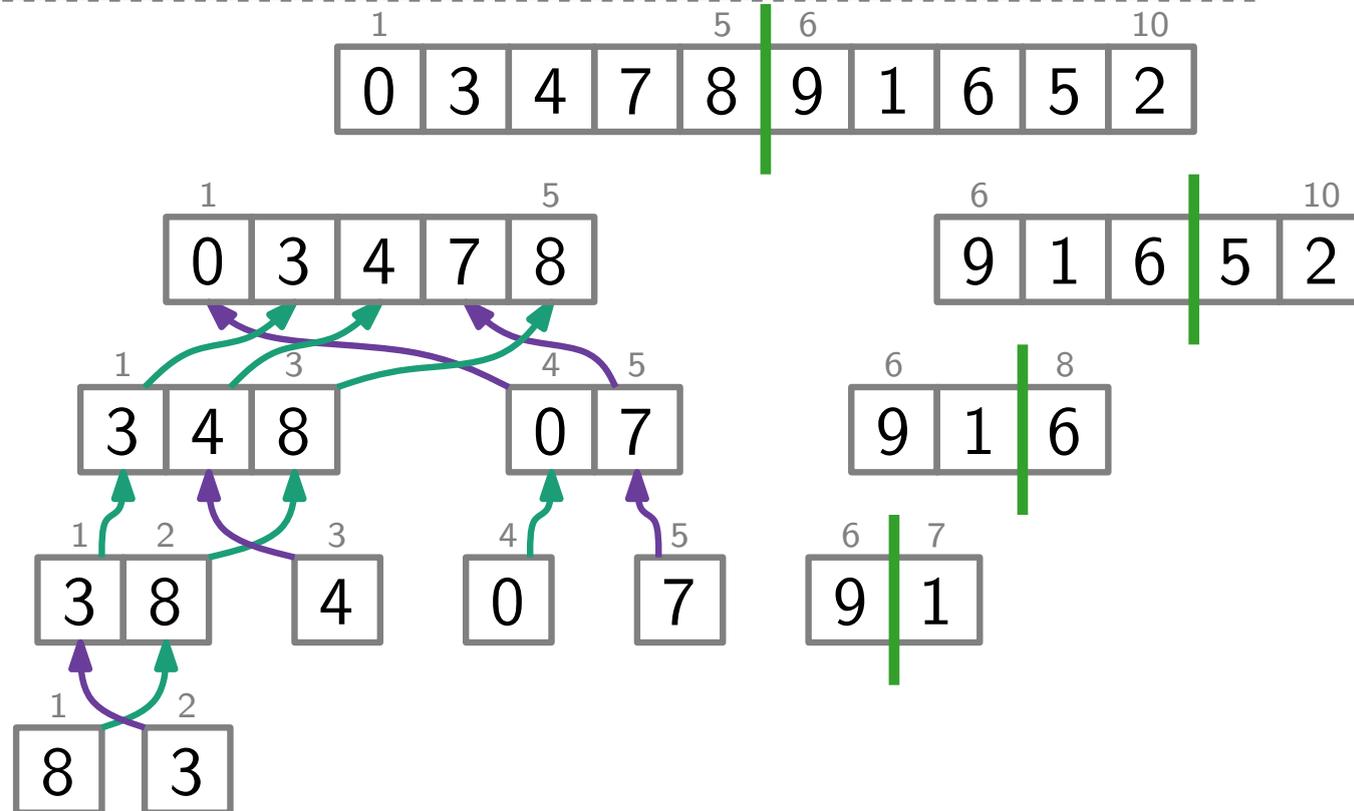
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

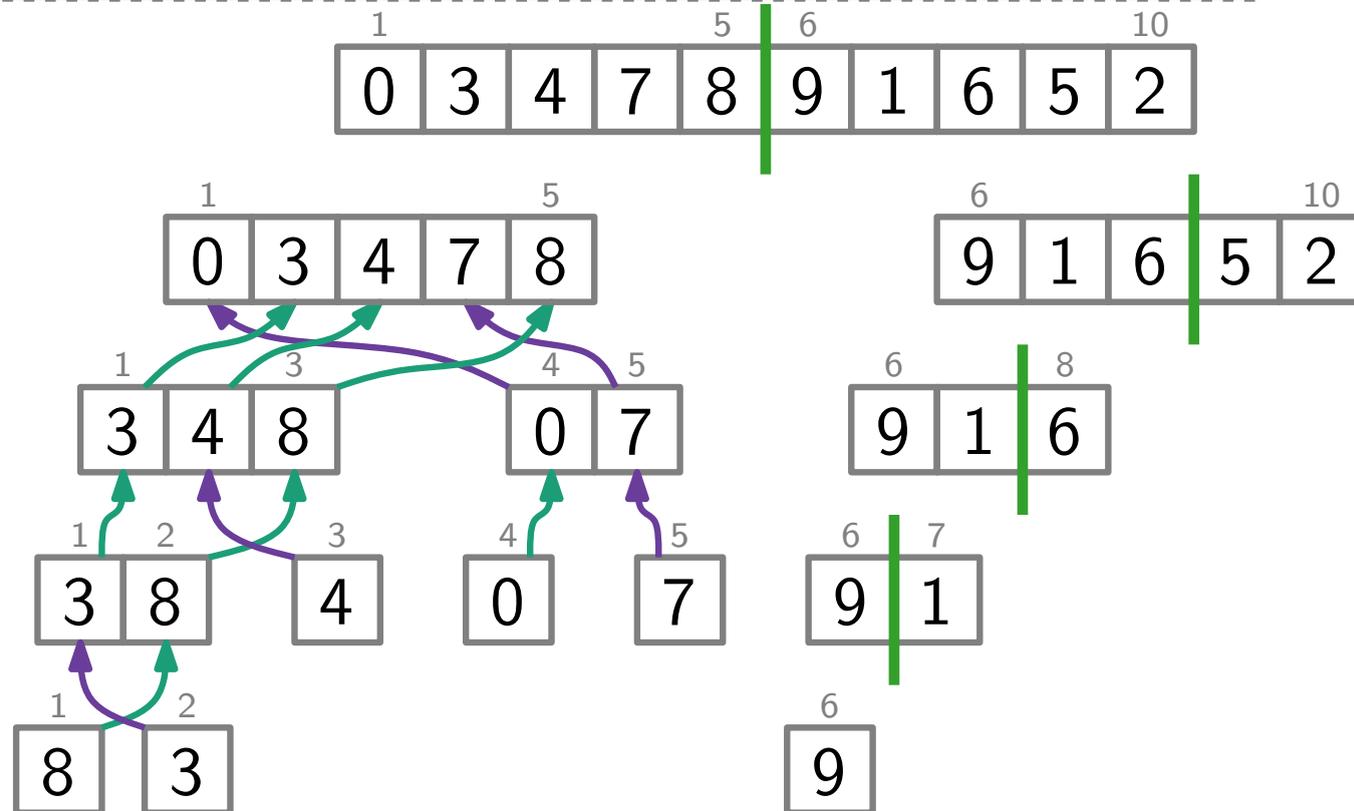
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere

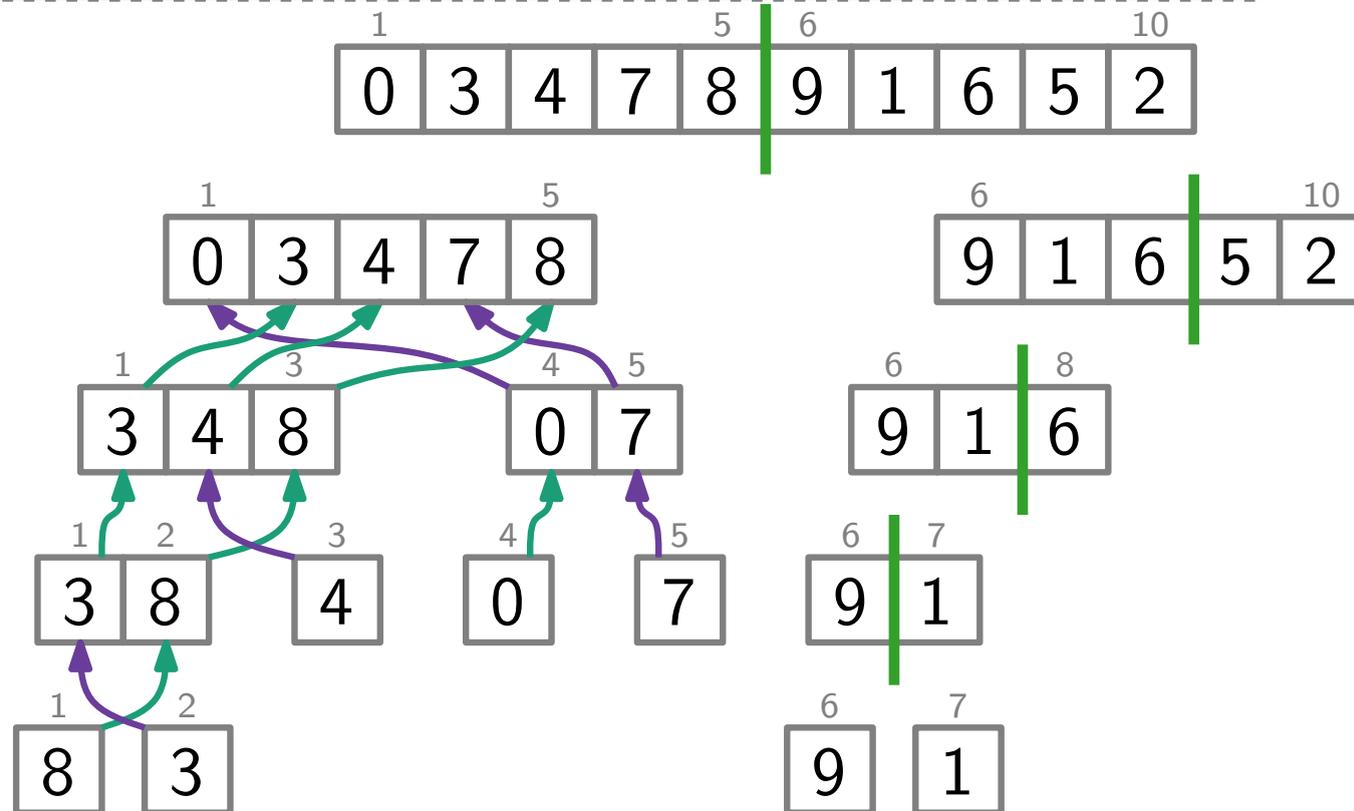


# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

if  $l < r$  then

$m = \lfloor (l + r) / 2 \rfloor$	} teile
MERGESORT(A, $l$ , $m$ )	} herrsche
MERGESORT(A, $m + 1$ , $r$ )	
MERGE(A, $l$ , $m$ , $r$ )	} kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

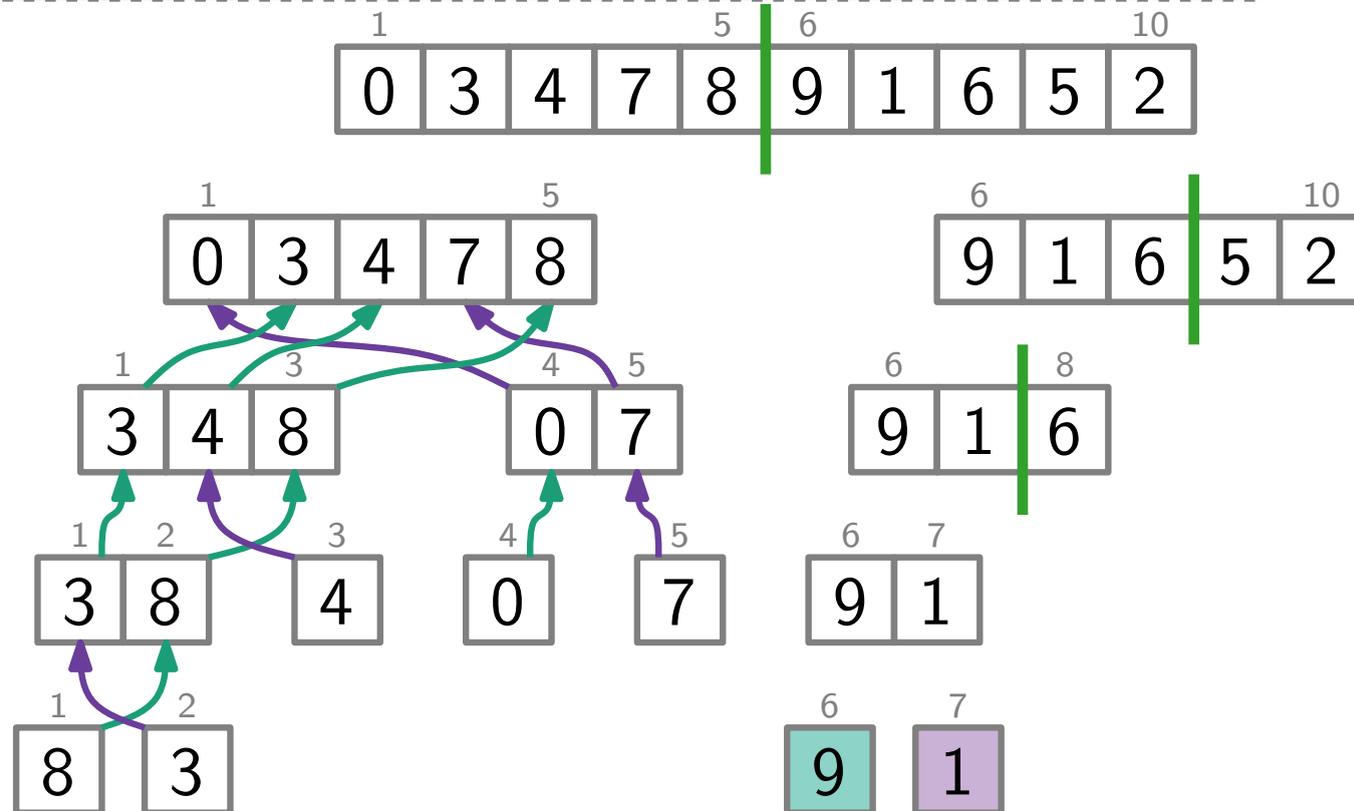
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

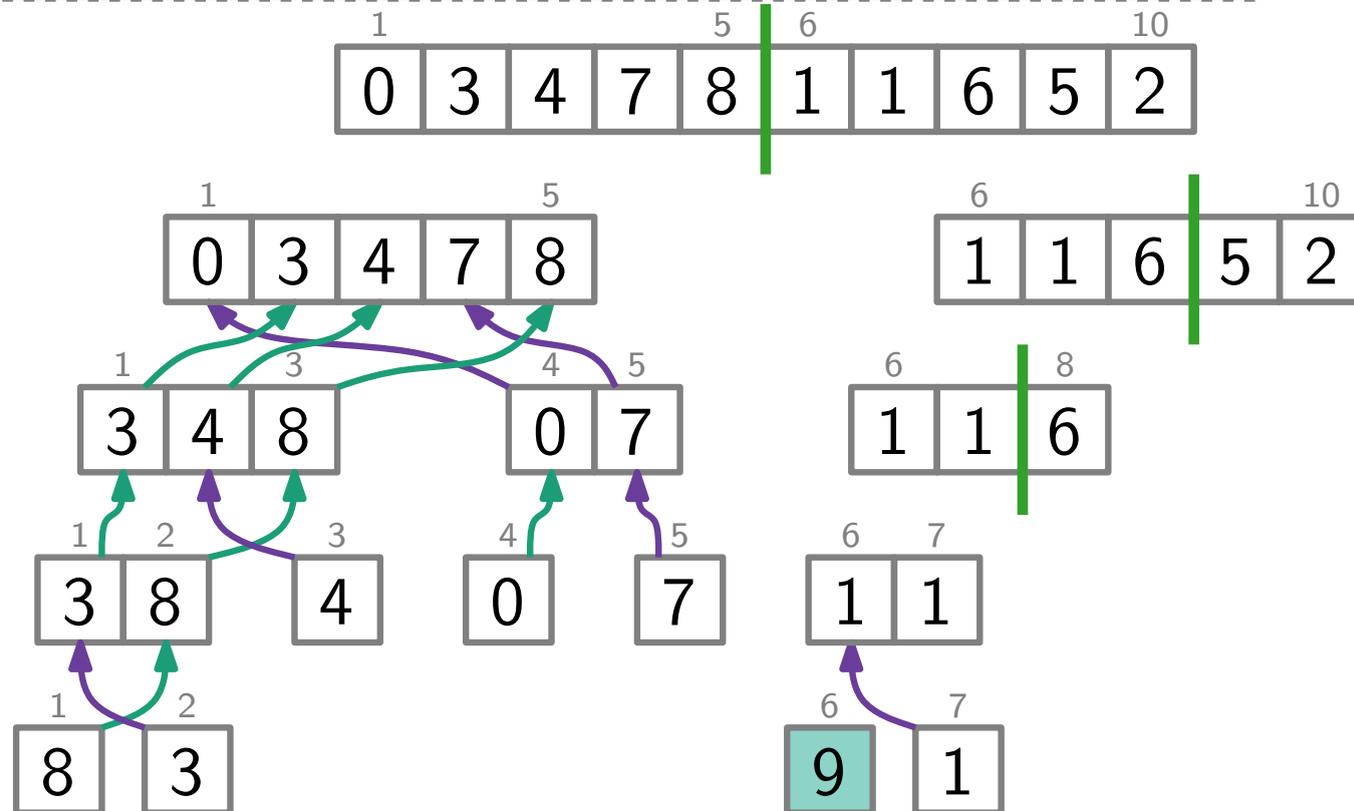
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

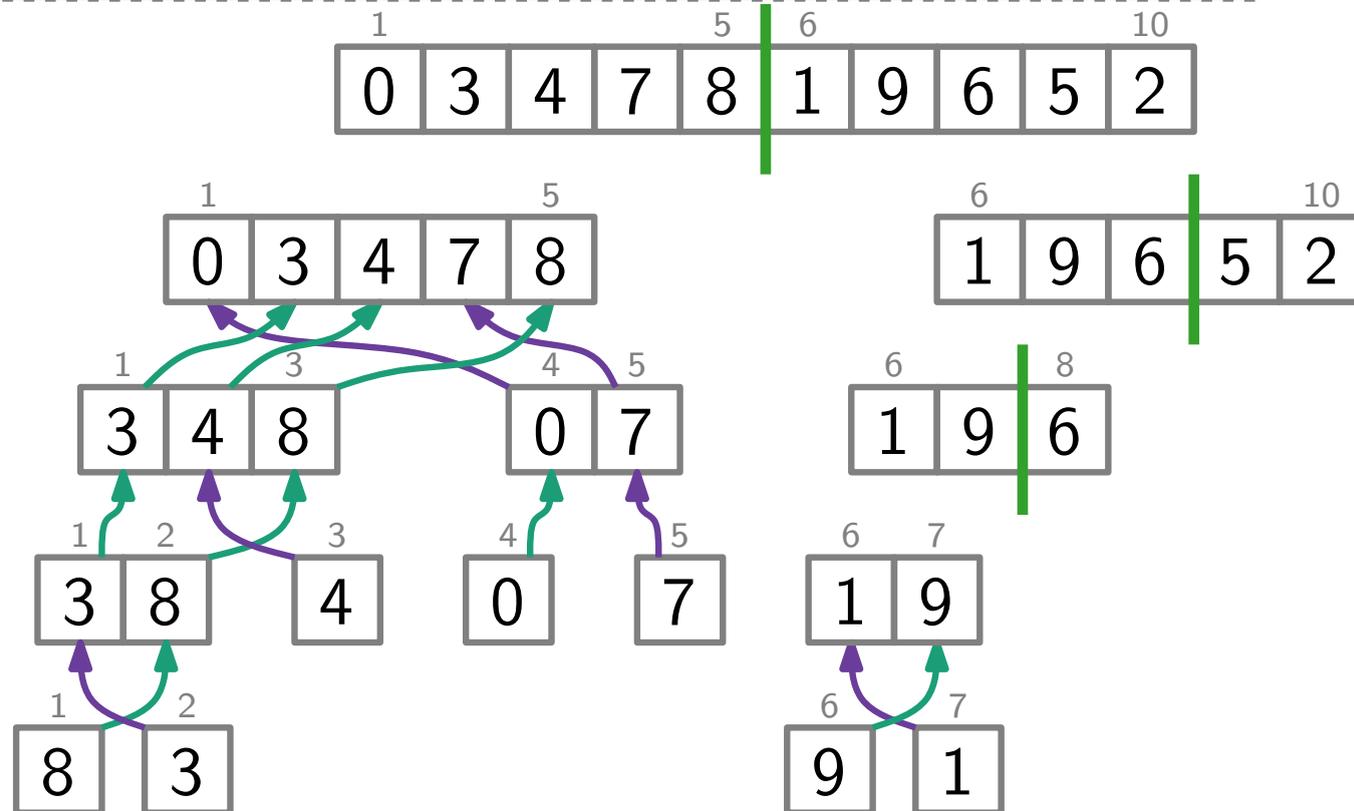
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

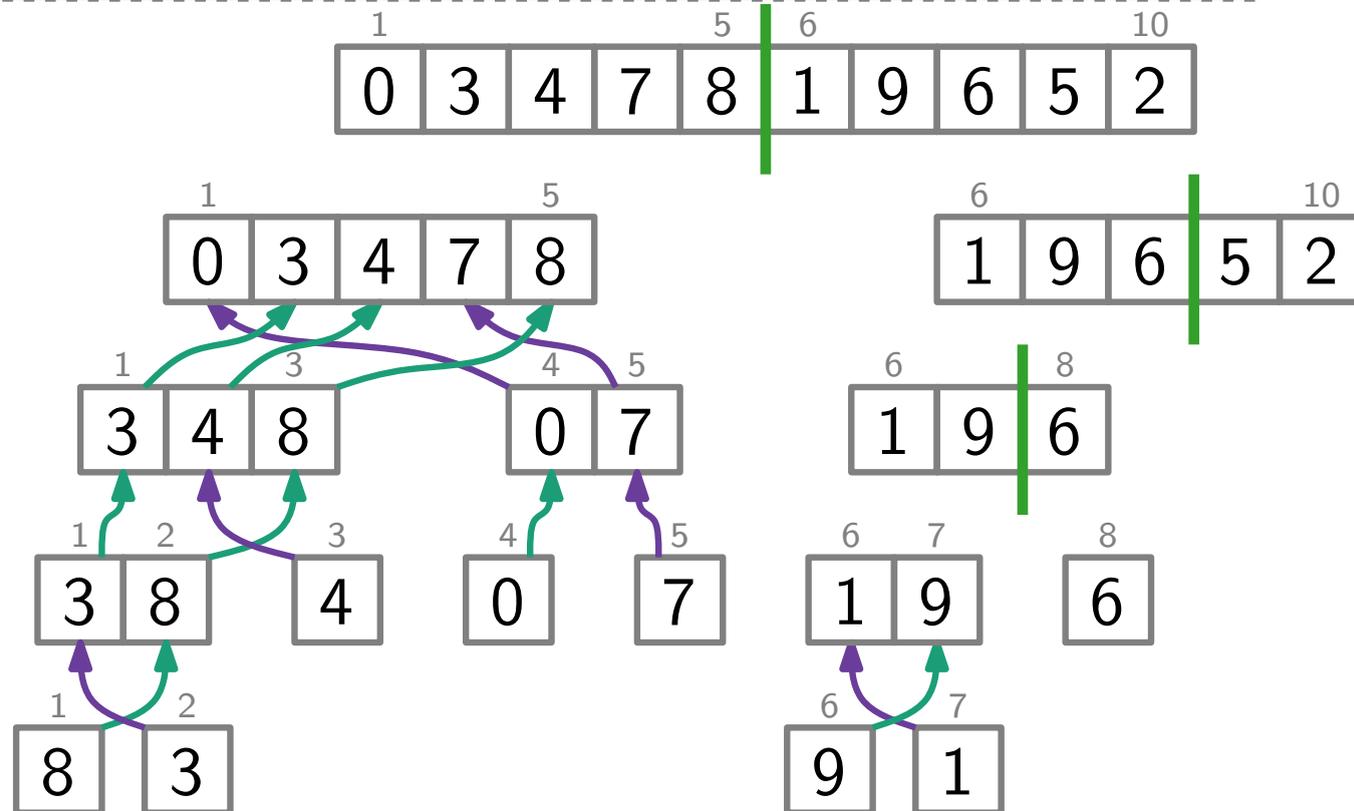
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere

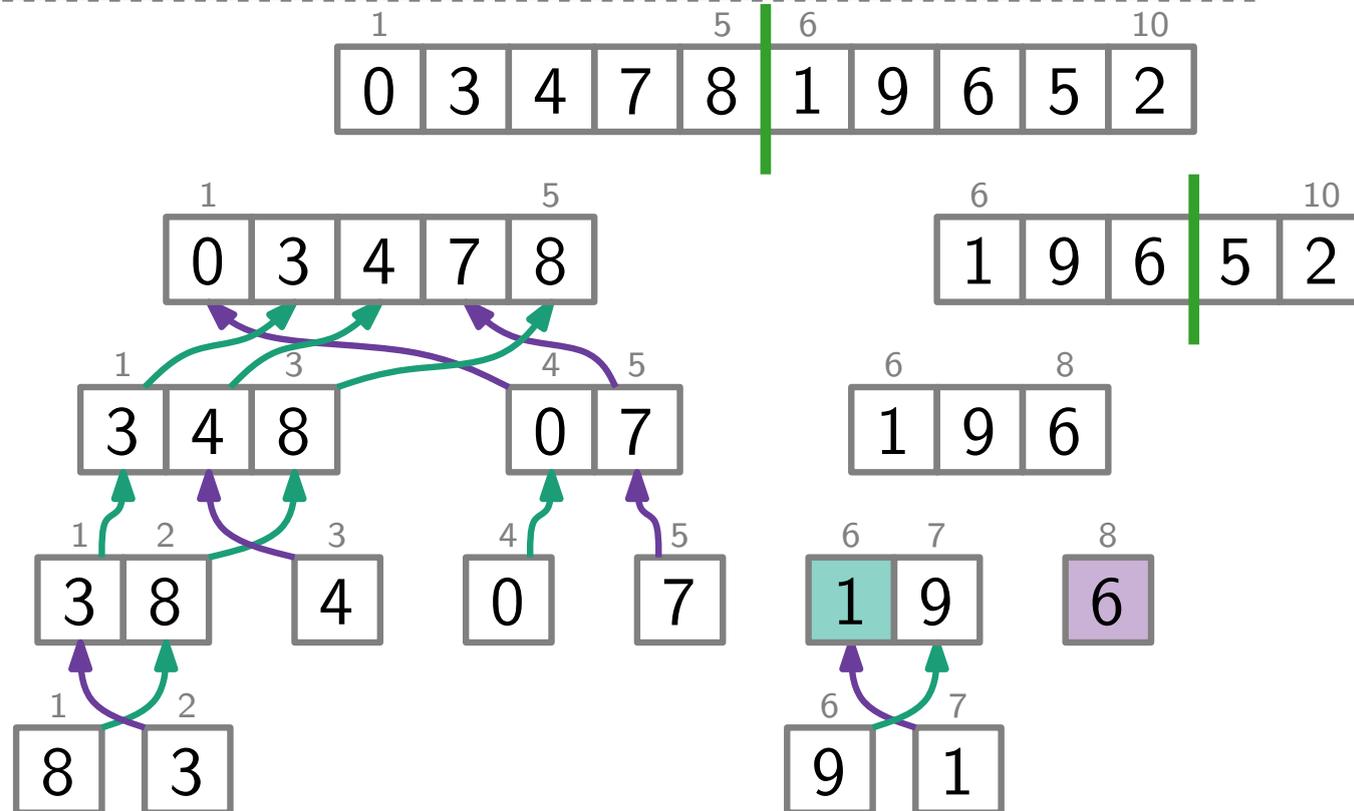


# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

if  $l < r$  then

$m = \lfloor (l + r) / 2 \rfloor$	} teile
MERGESORT(A, $l$ , $m$ )	} herrsche
MERGESORT(A, $m + 1$ , $r$ )	
MERGE(A, $l$ , $m$ , $r$ )	} kombiniere

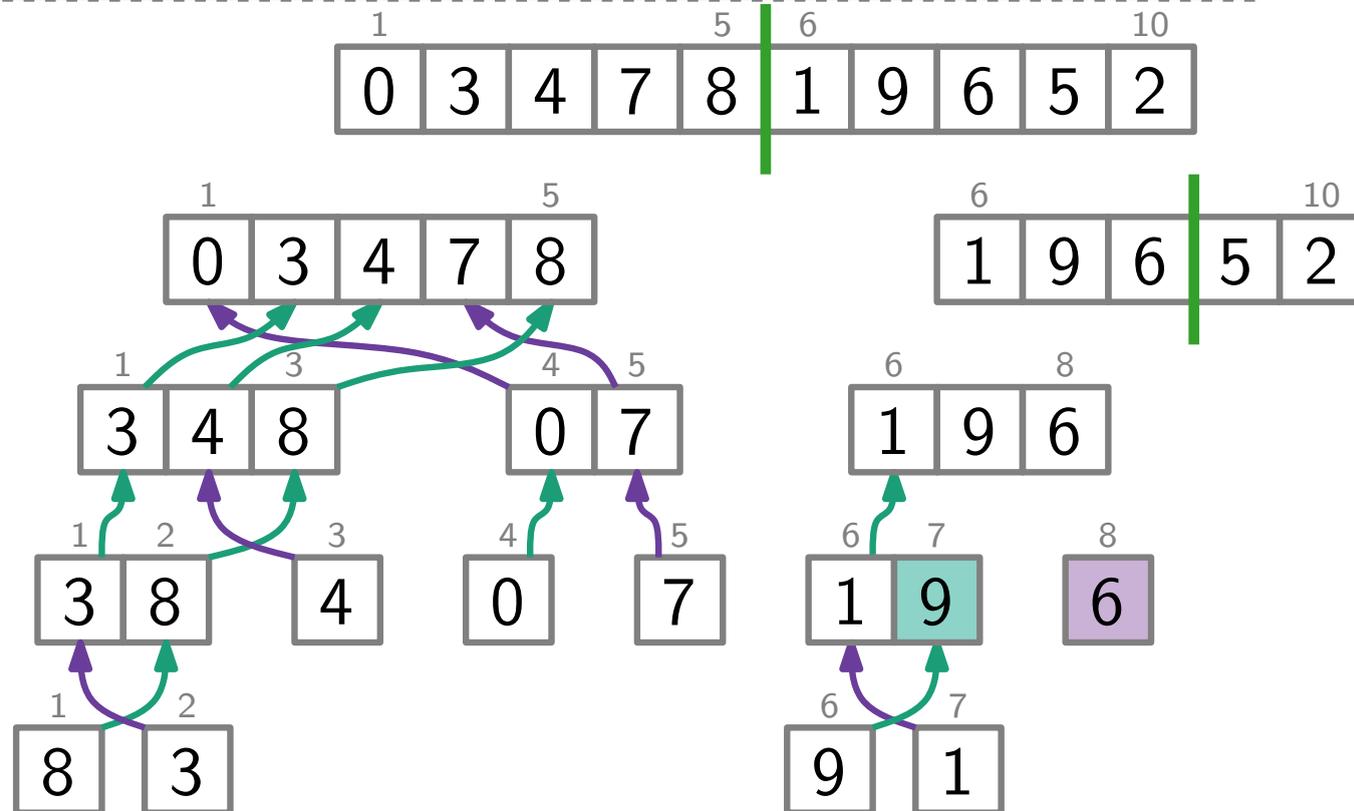


# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

if  $l < r$  then

$m = \lfloor (l + r) / 2 \rfloor$	} teile
MERGESORT(A, $l$ , $m$ )	} herrsche
MERGESORT(A, $m + 1$ , $r$ )	
MERGE(A, $l$ , $m$ , $r$ )	} kombiniere

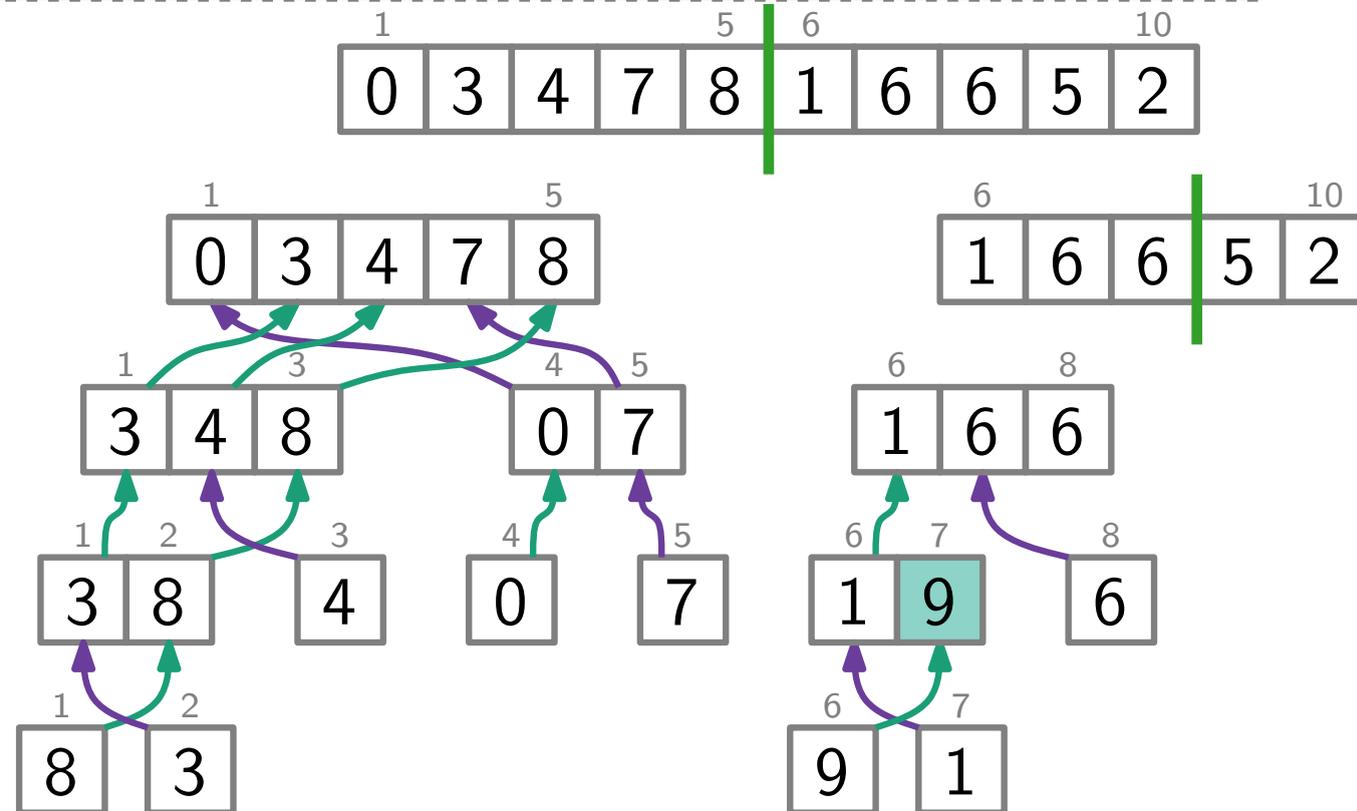


# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

if  $l < r$  then

$m = \lfloor (l + r) / 2 \rfloor$	} teile
MERGESORT(A, $l$ , $m$ )	} herrsche
MERGESORT(A, $m + 1$ , $r$ )	
MERGE(A, $l$ , $m$ , $r$ )	} kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

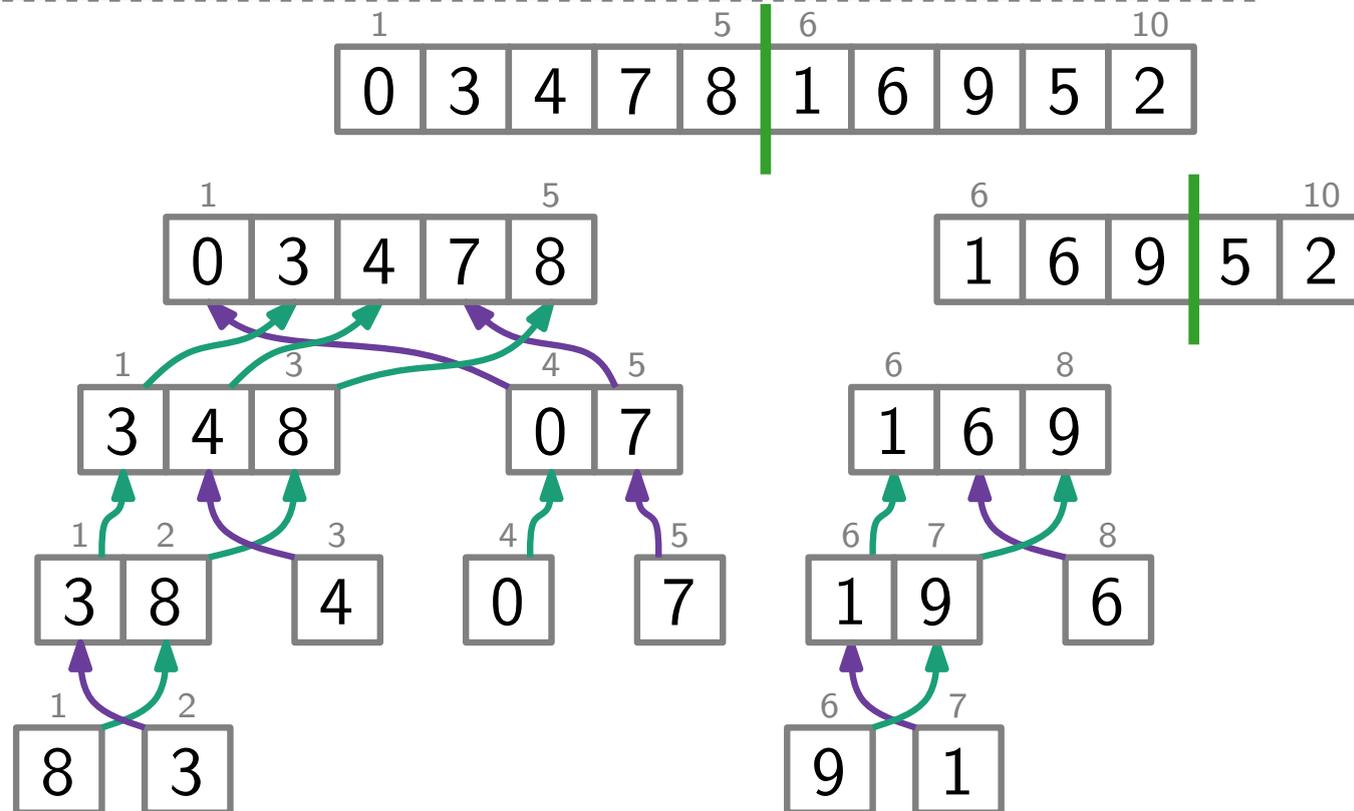
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere

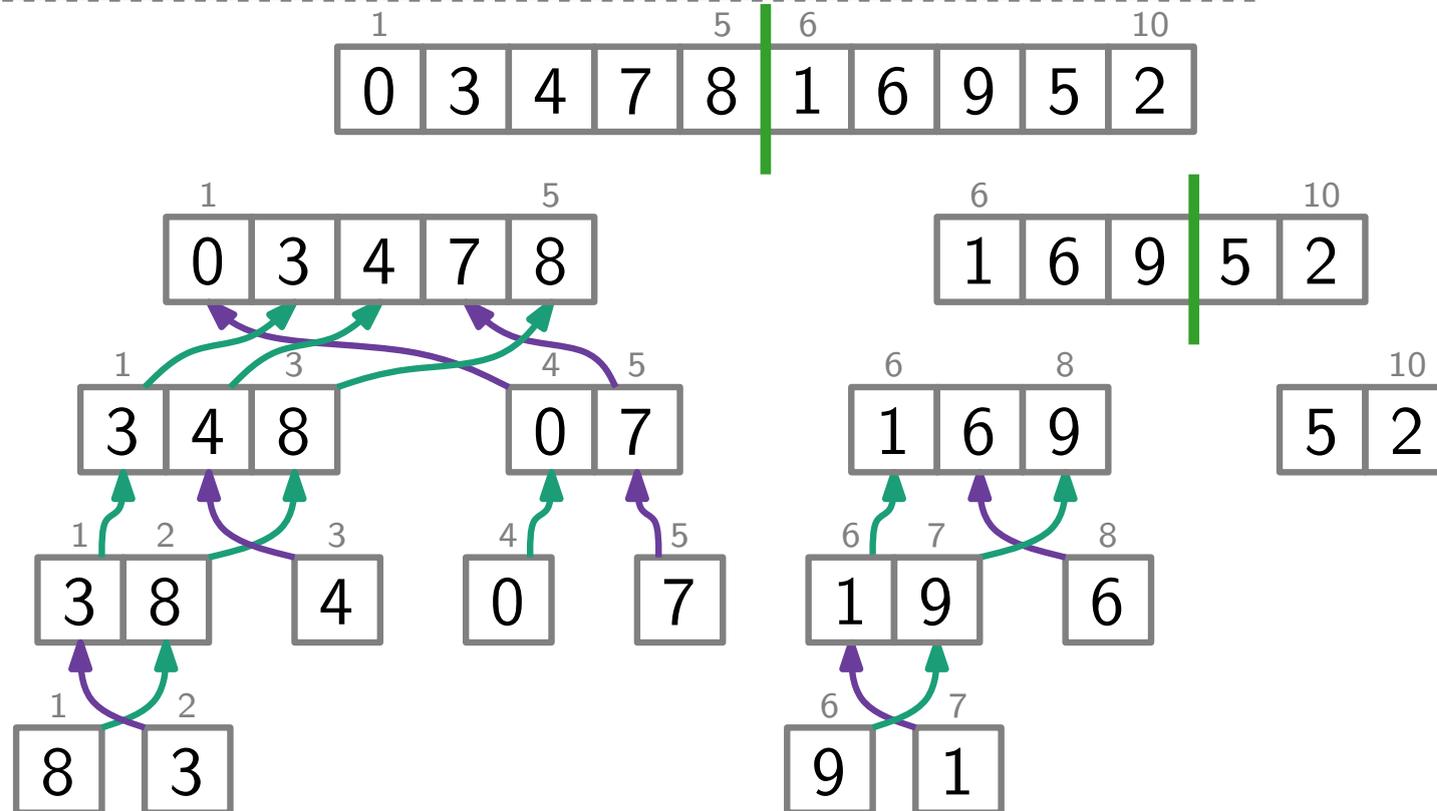


# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

if  $l < r$  then

$m = \lfloor (l + r) / 2 \rfloor$	} teile
MERGESORT(A, $l$ , $m$ )	} herrsche
MERGESORT(A, $m + 1$ , $r$ )	
MERGE(A, $l$ , $m$ , $r$ )	} kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

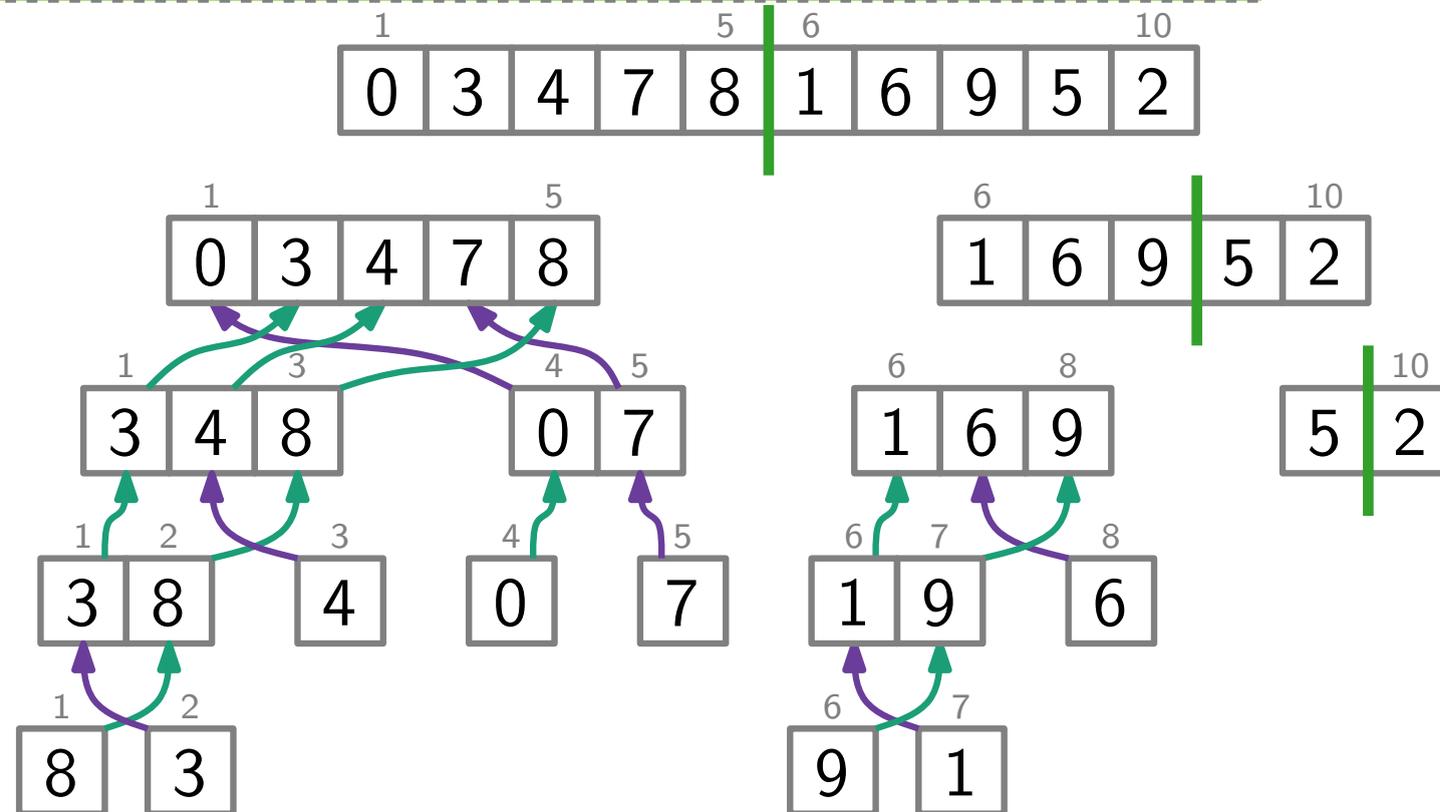
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere

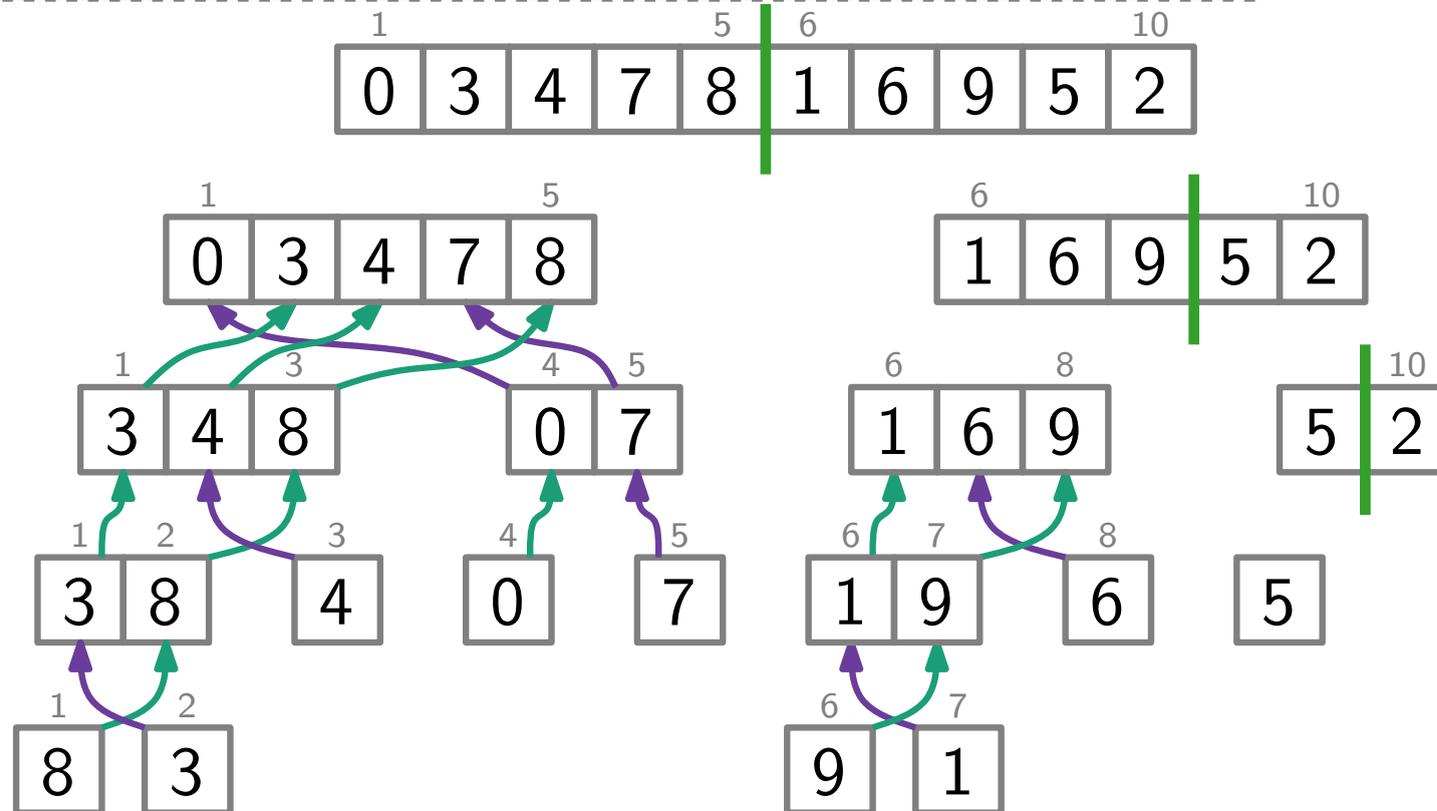


# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

if  $l < r$  then

$m = \lfloor (l + r) / 2 \rfloor$	} teile
MERGESORT(A, $l$ , $m$ )	} herrsche
MERGESORT(A, $m + 1$ , $r$ )	
MERGE(A, $l$ , $m$ , $r$ )	} kombiniere



# MergeSort – ein Beispiel

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

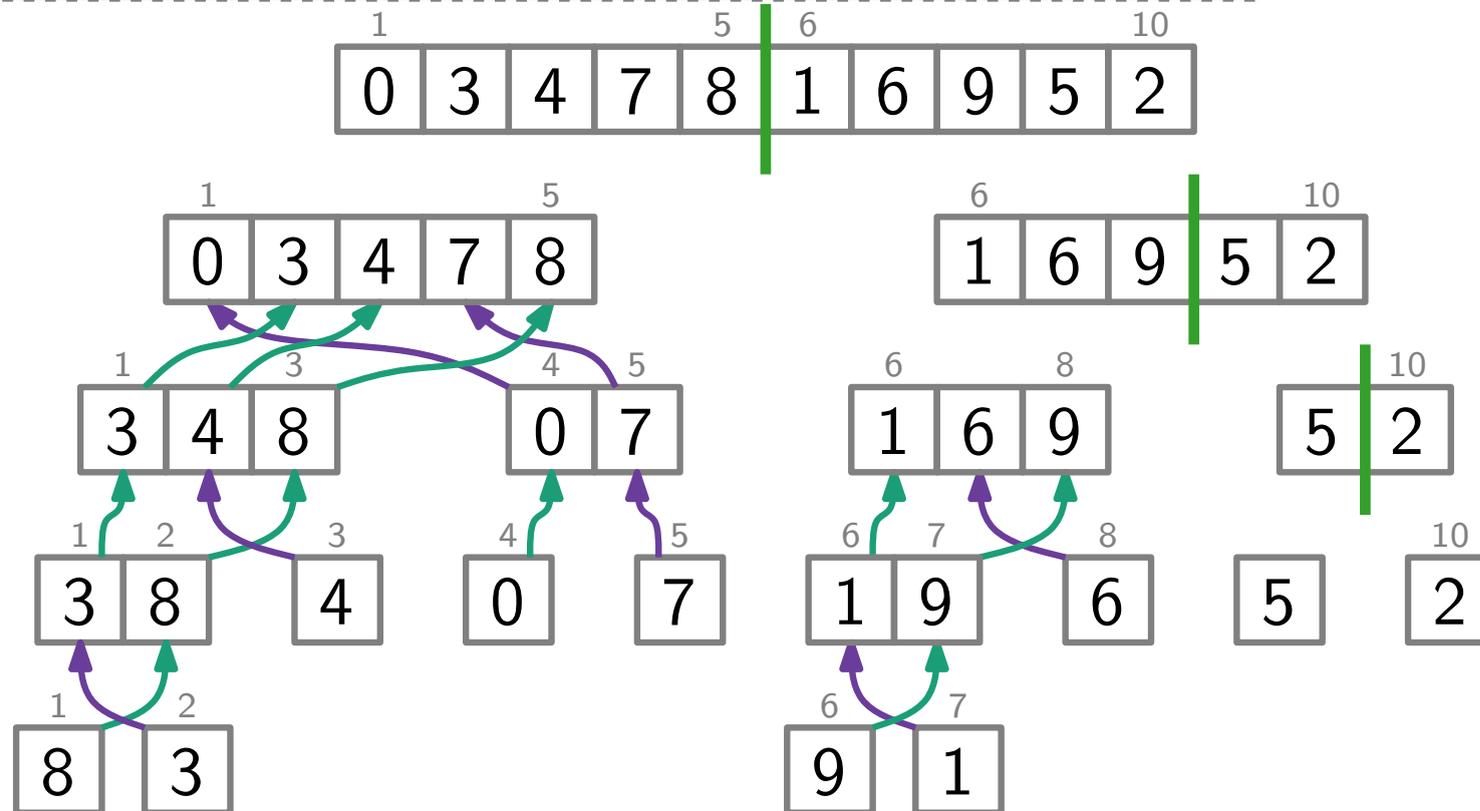
```
if  $l < r$  then
```

```
     $m = \lfloor (l + r) / 2 \rfloor$  } teile
```

```
    MERGESORT(A,  $l$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

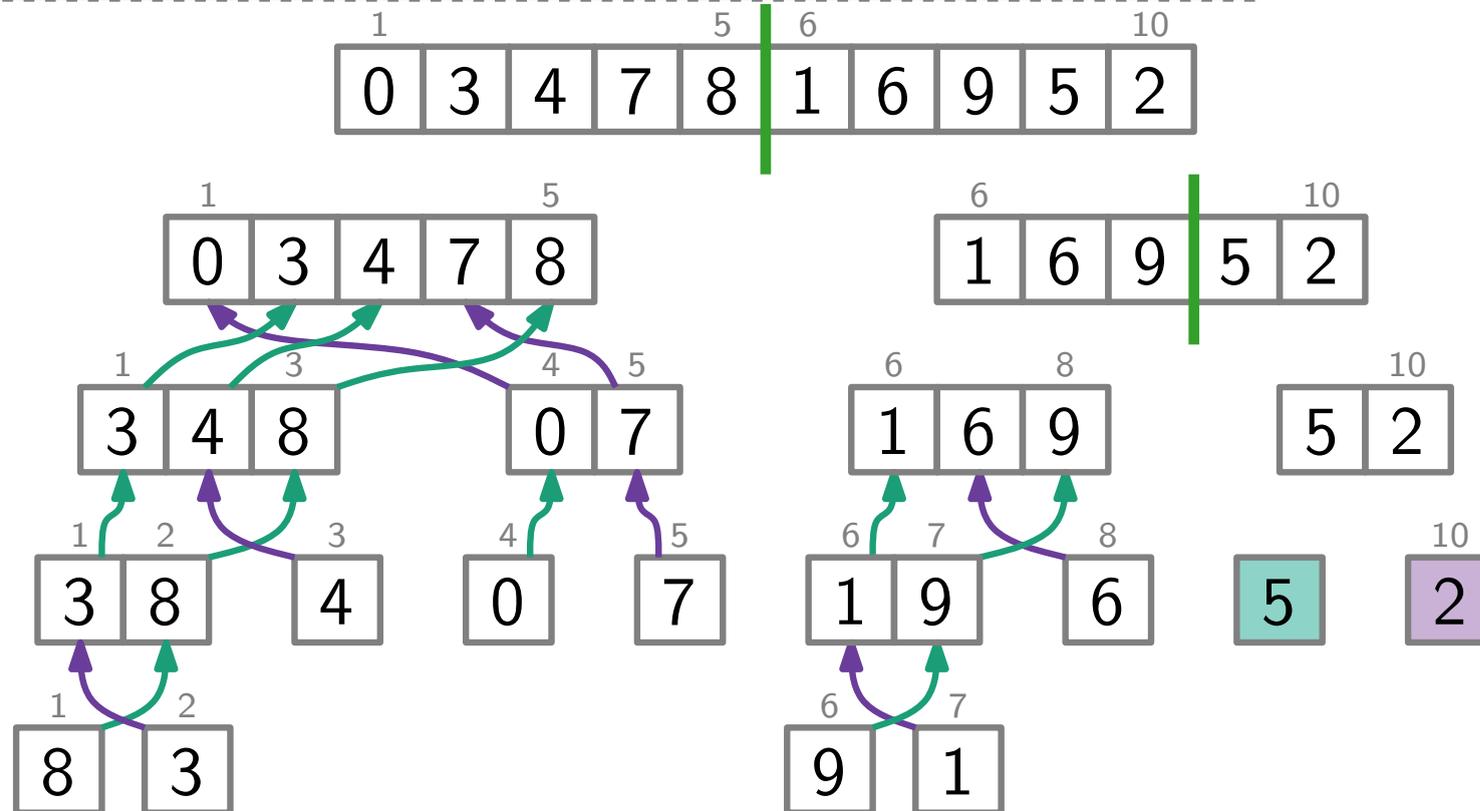
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

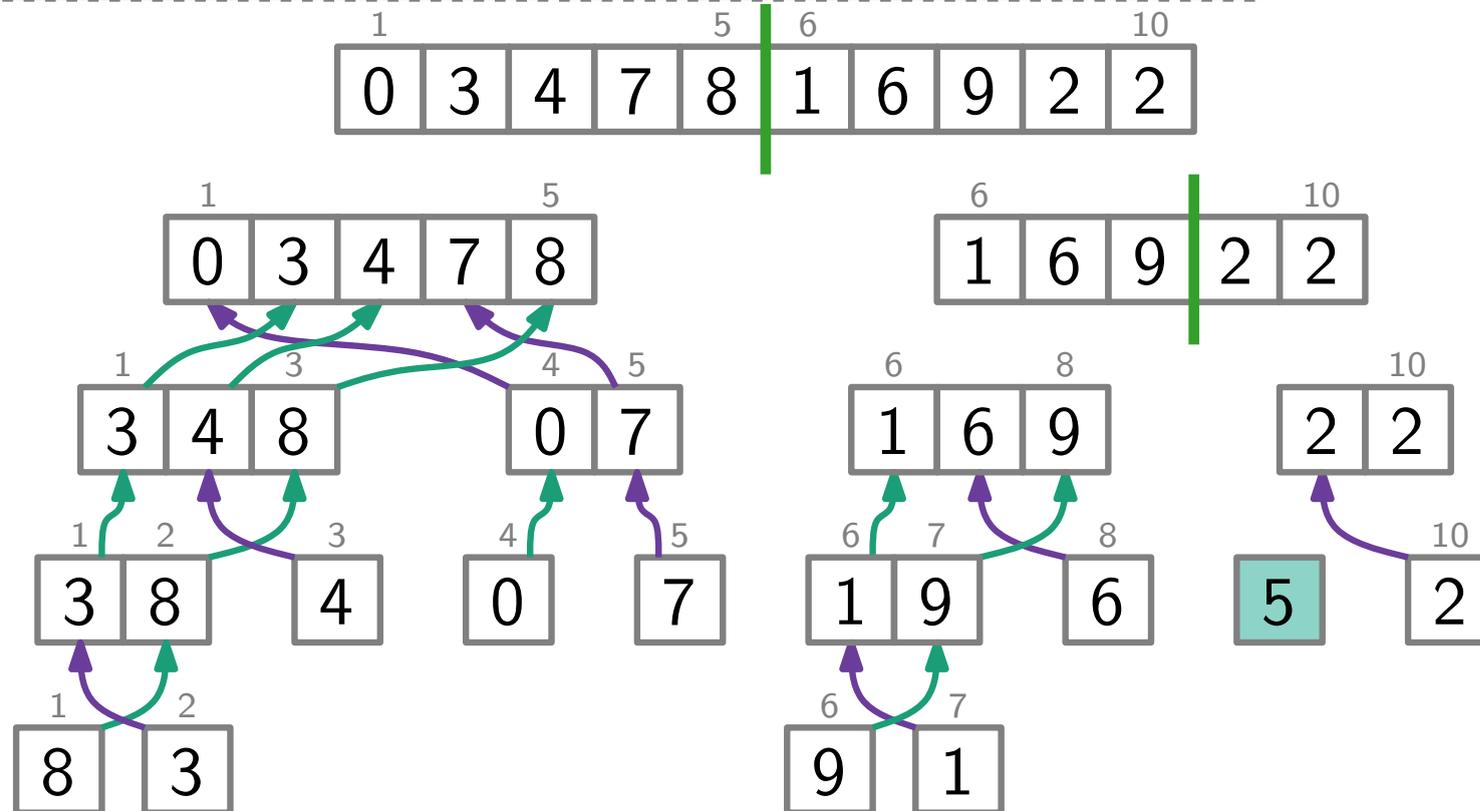
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

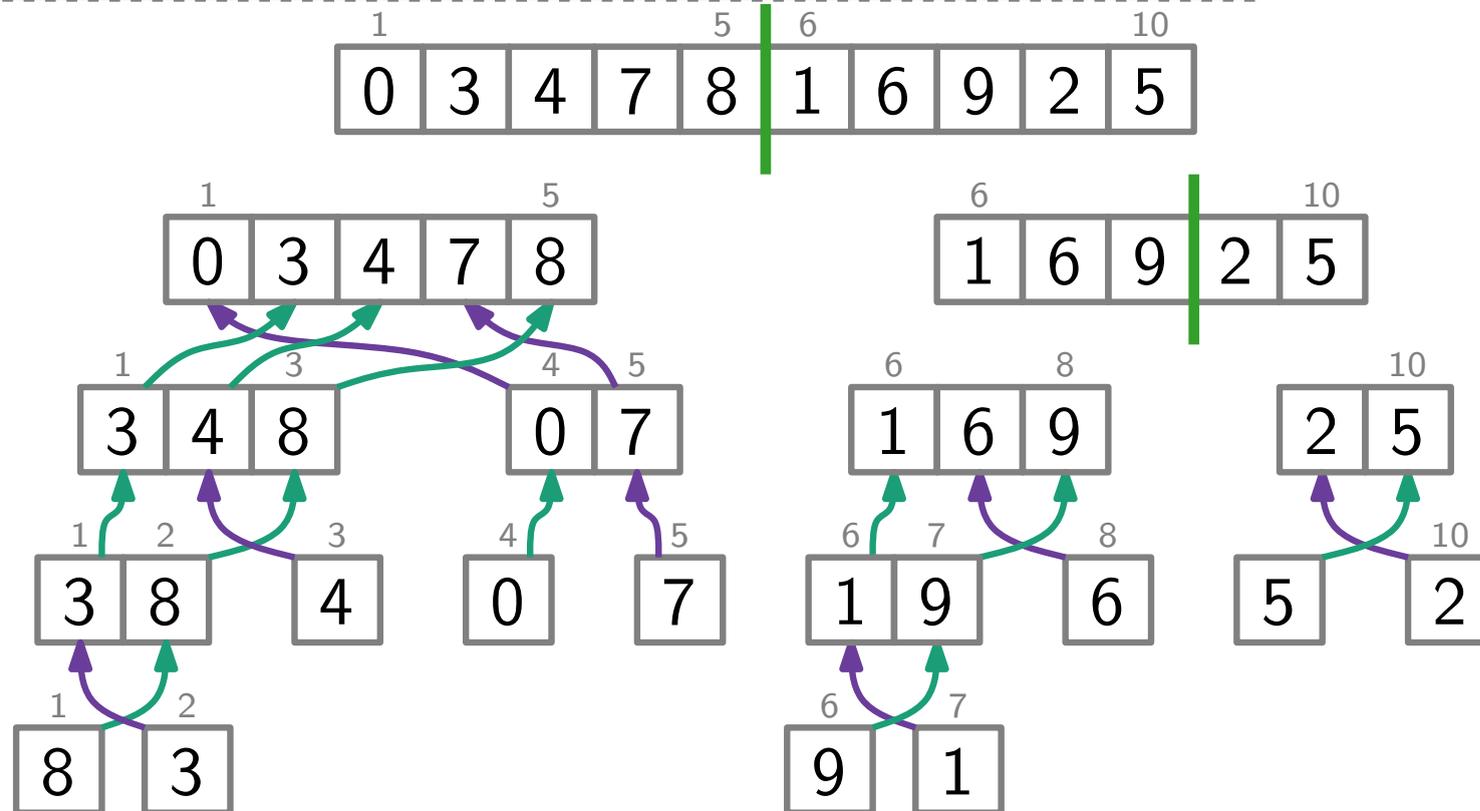
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

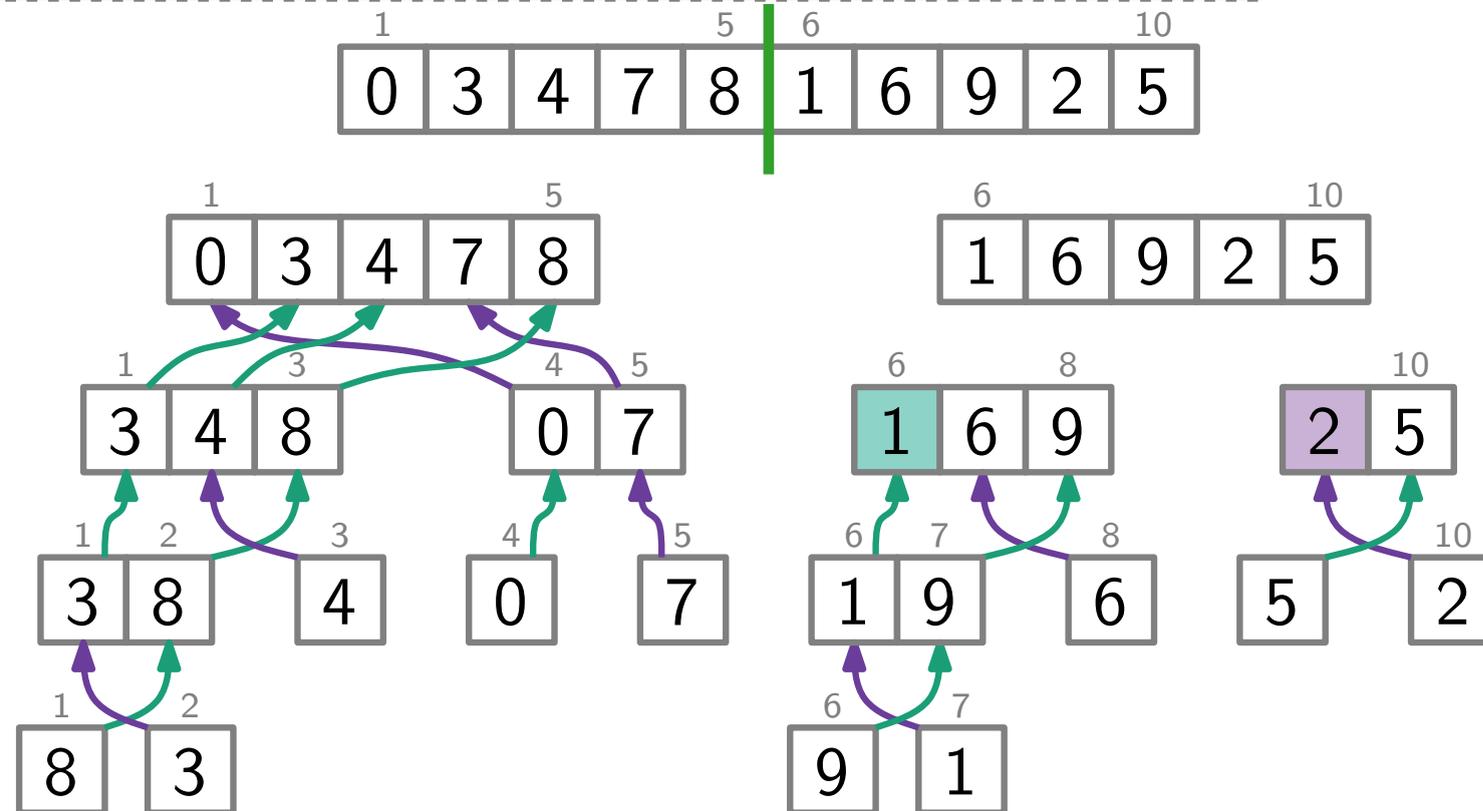
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

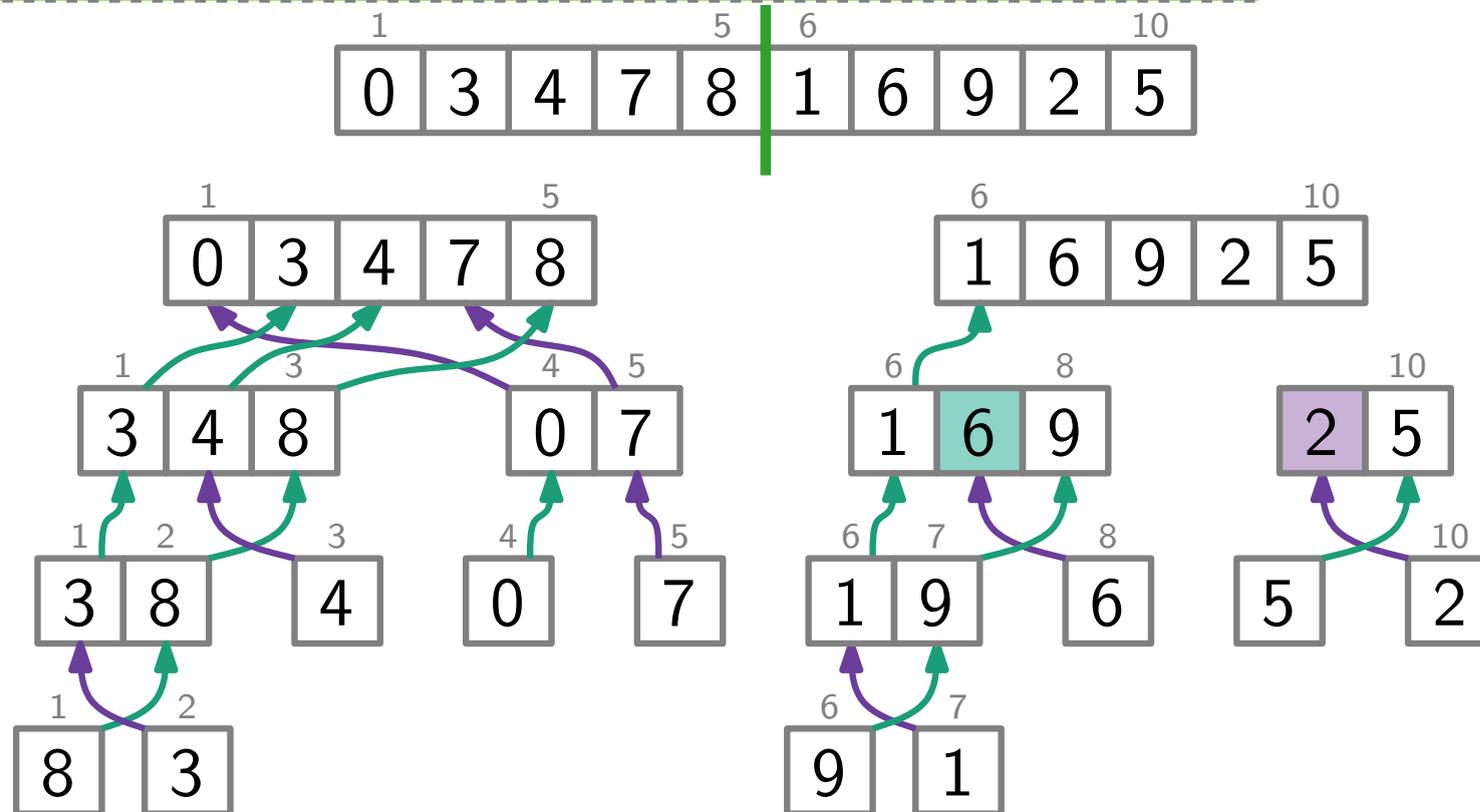
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

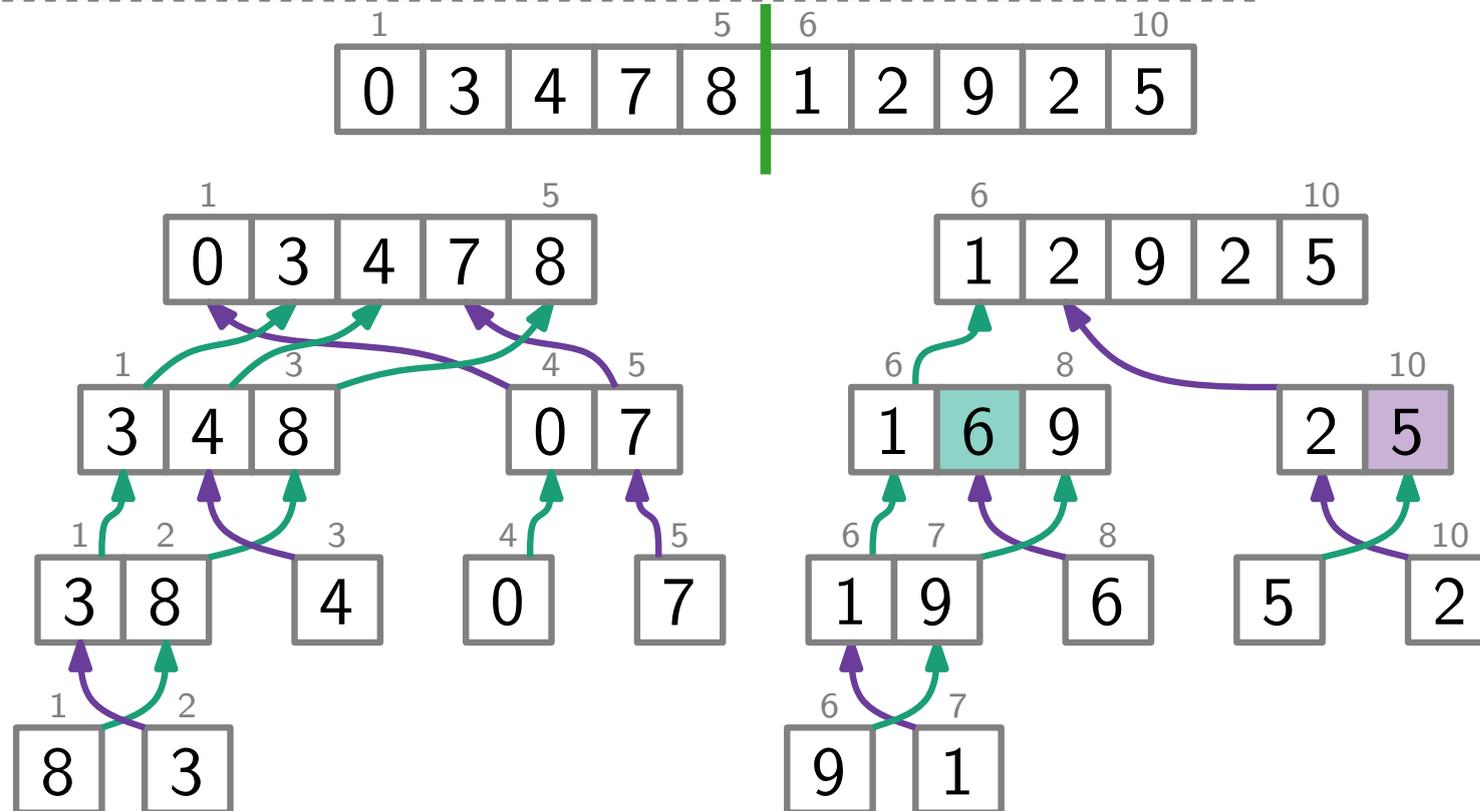
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

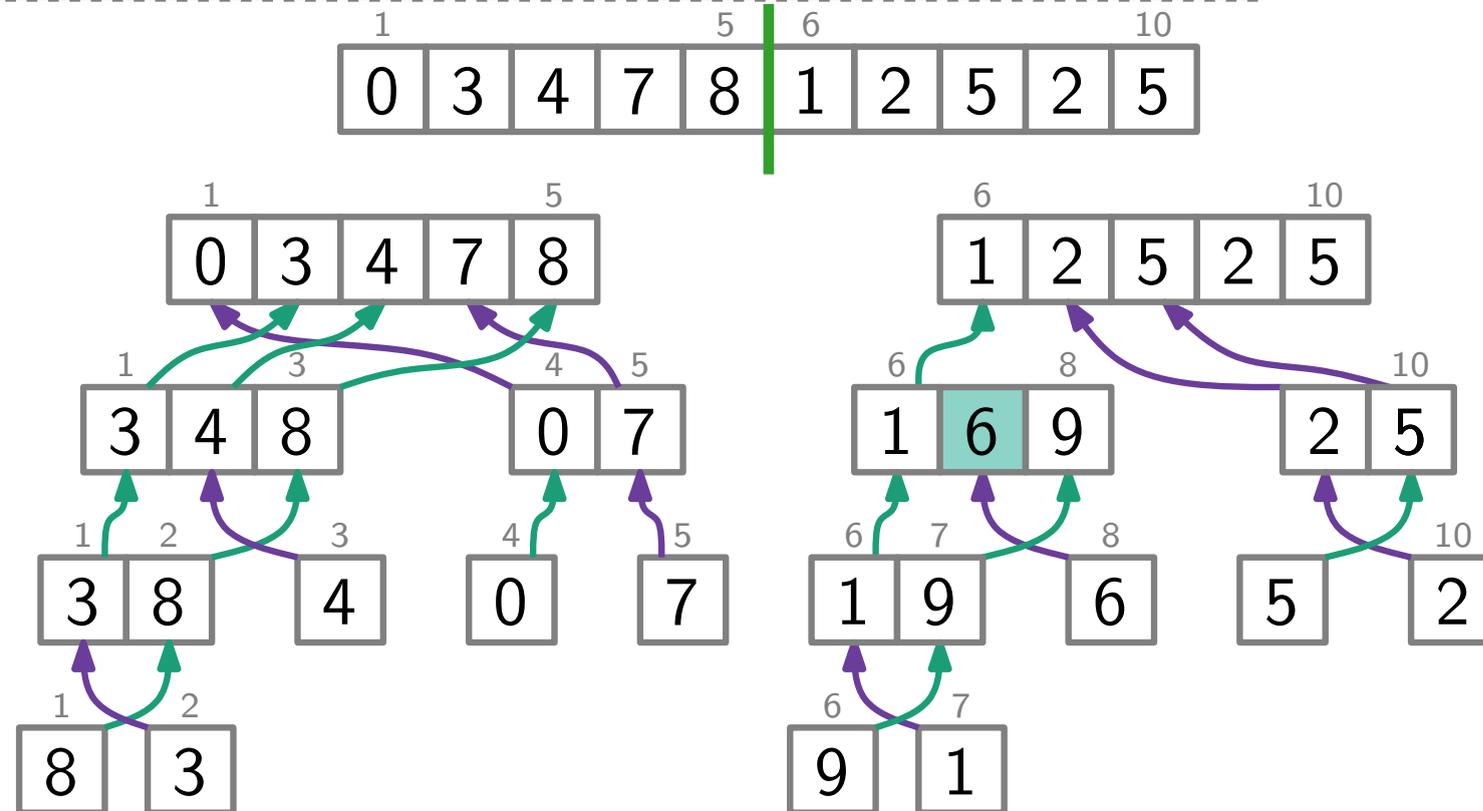
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

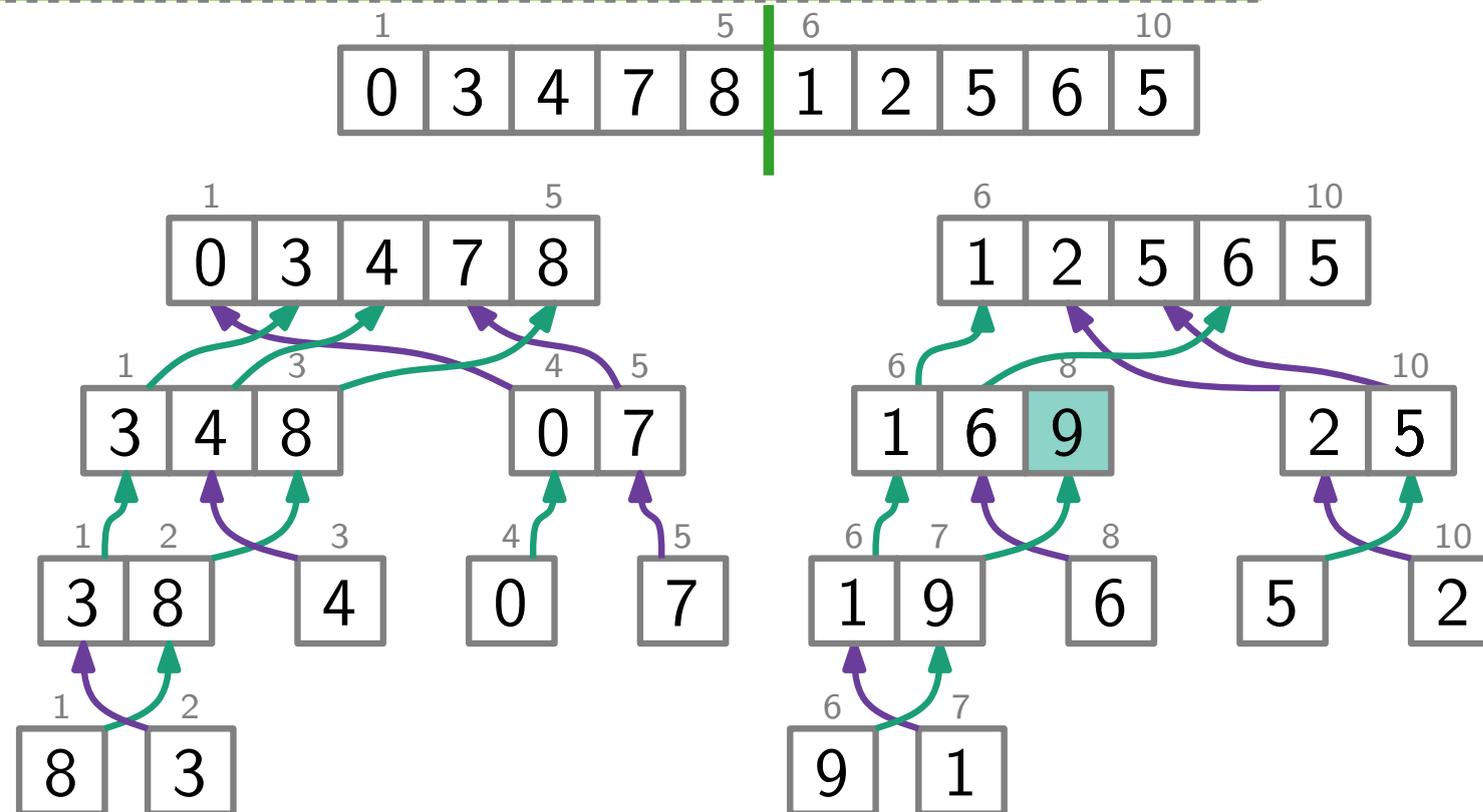
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

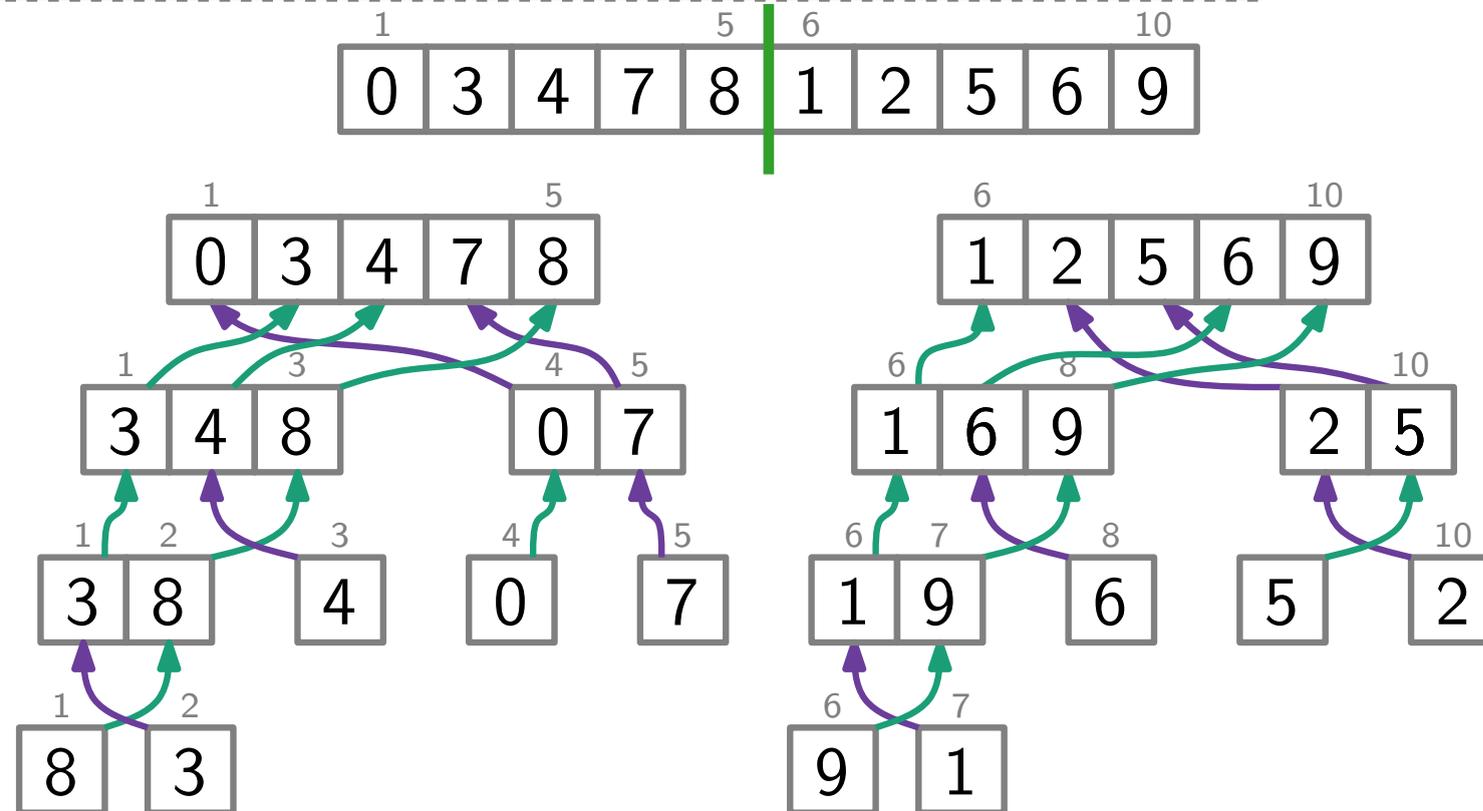
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

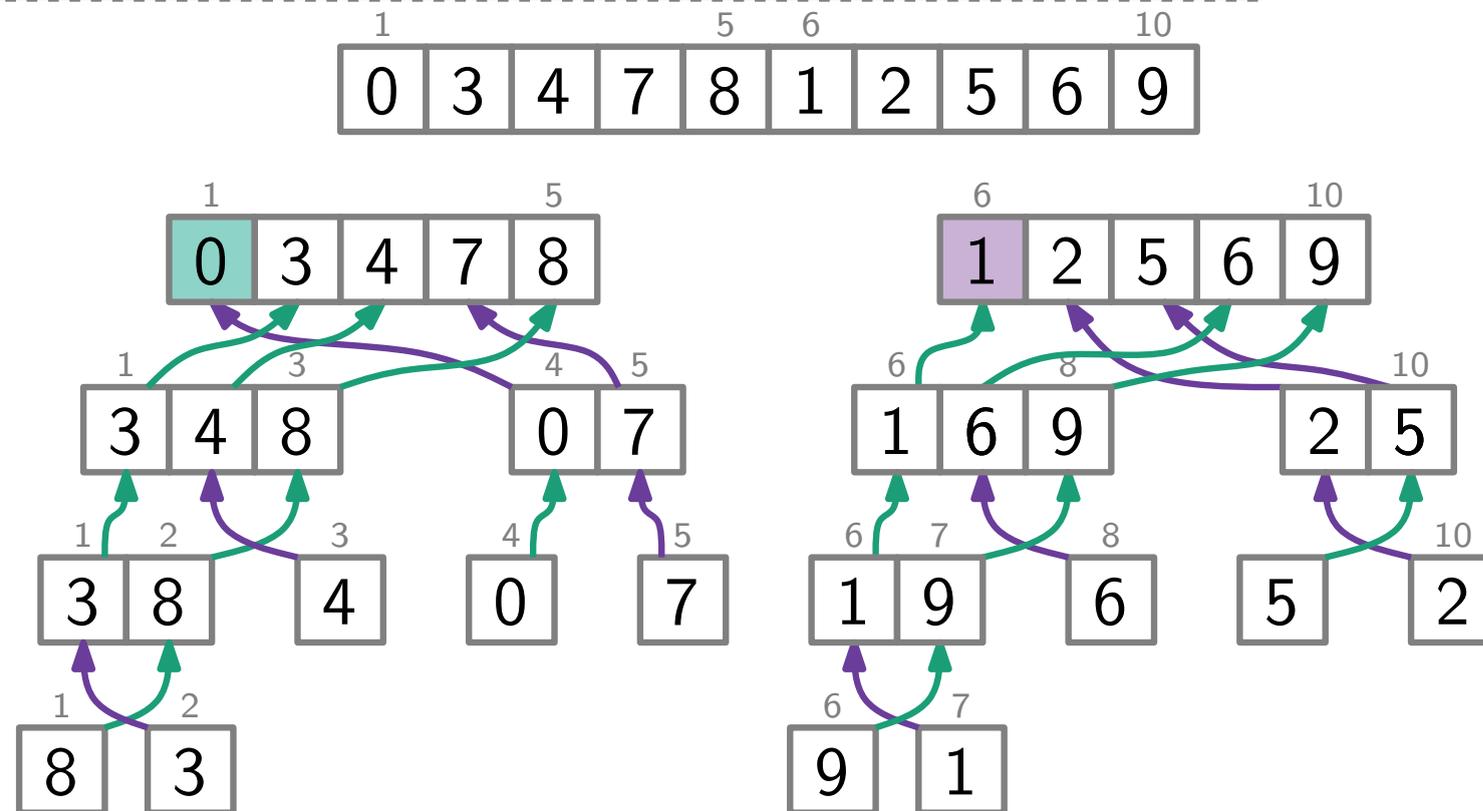
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

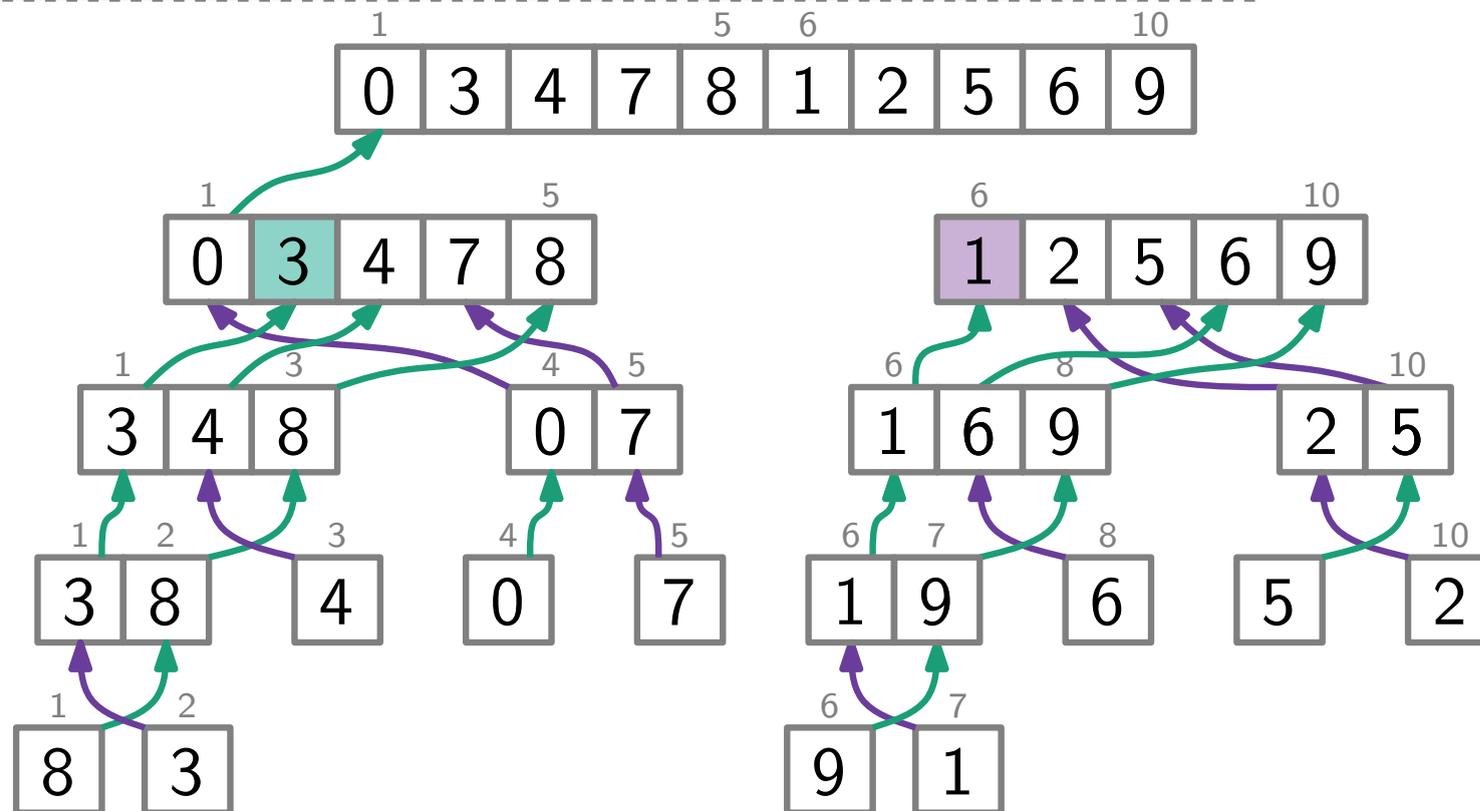
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

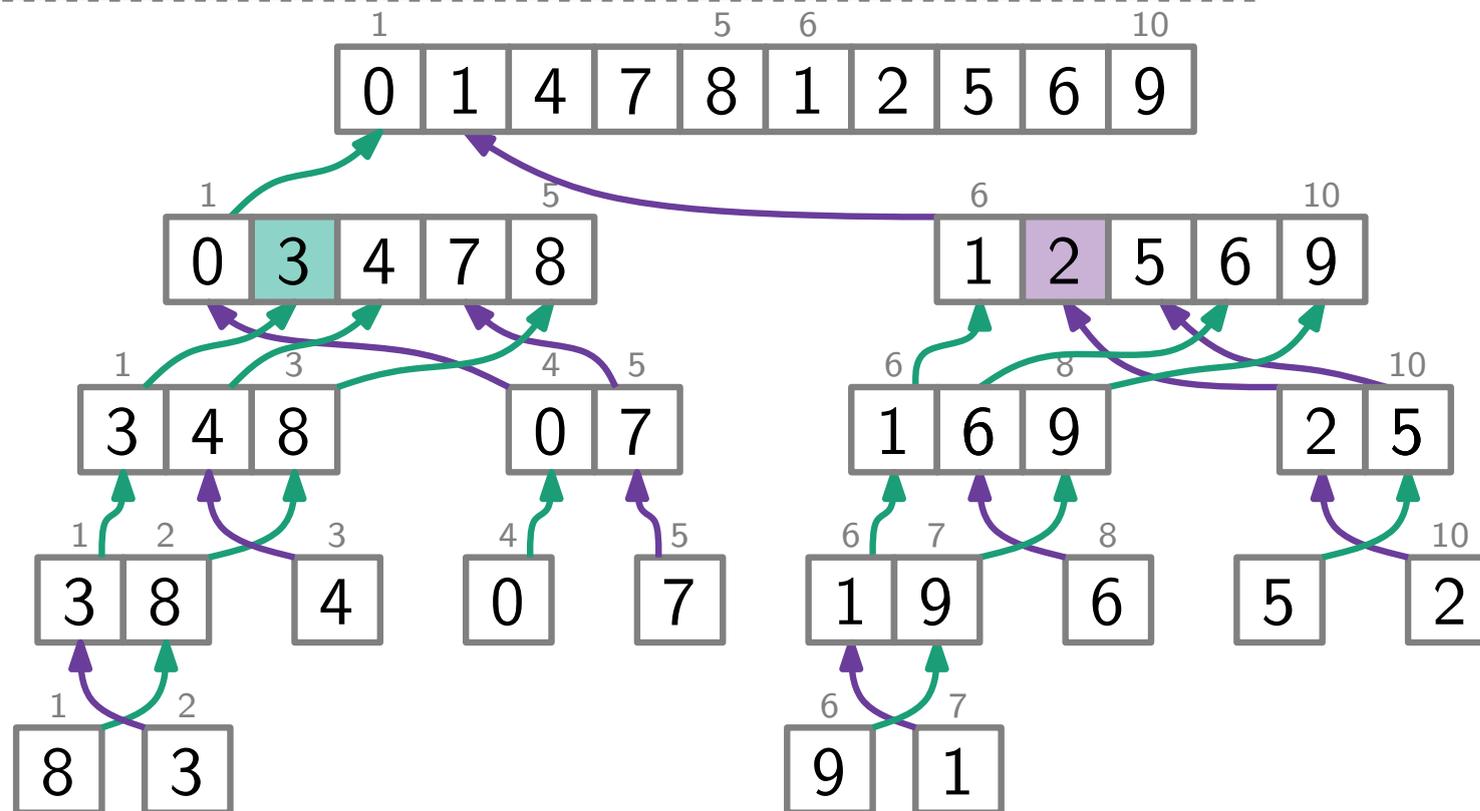
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

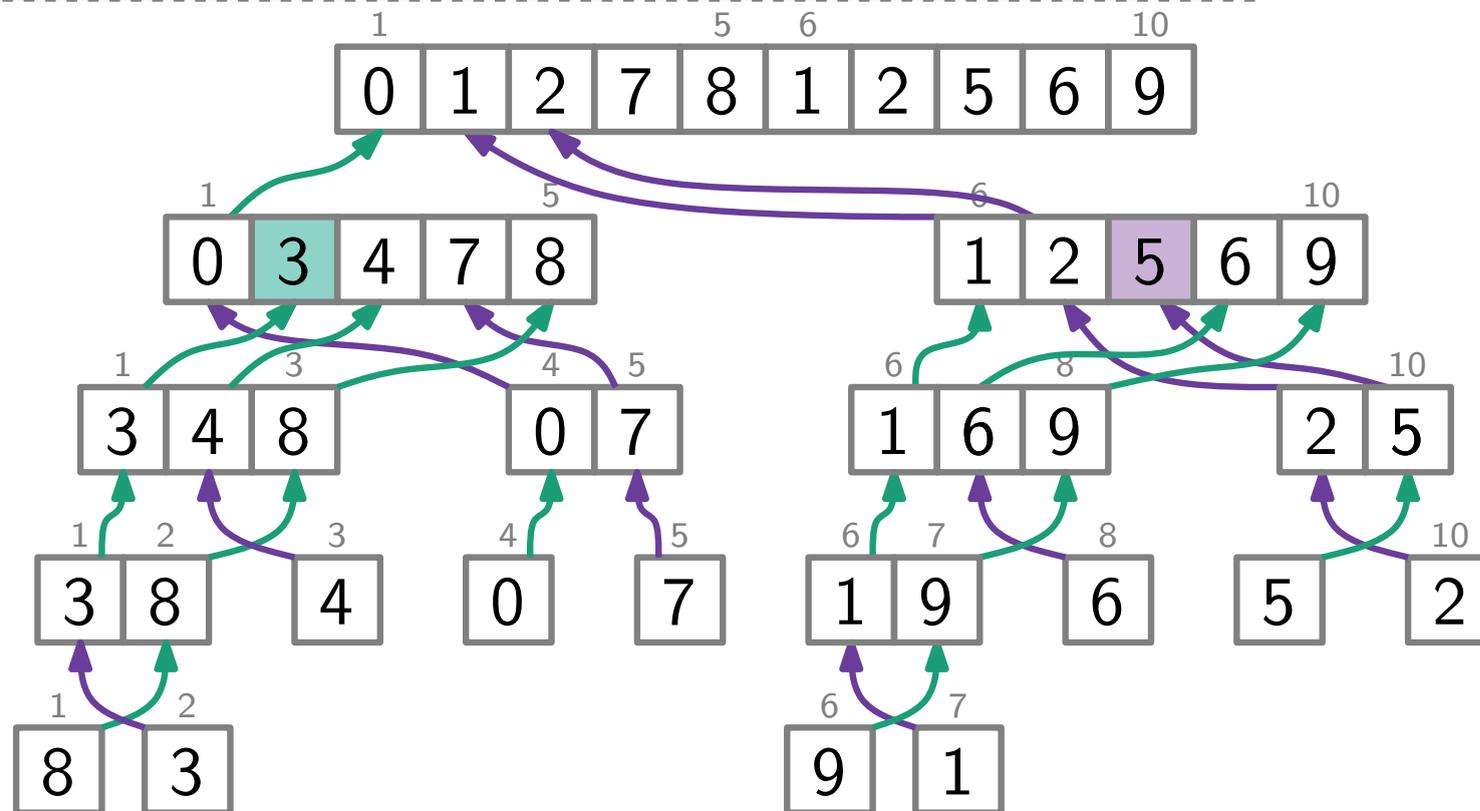
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

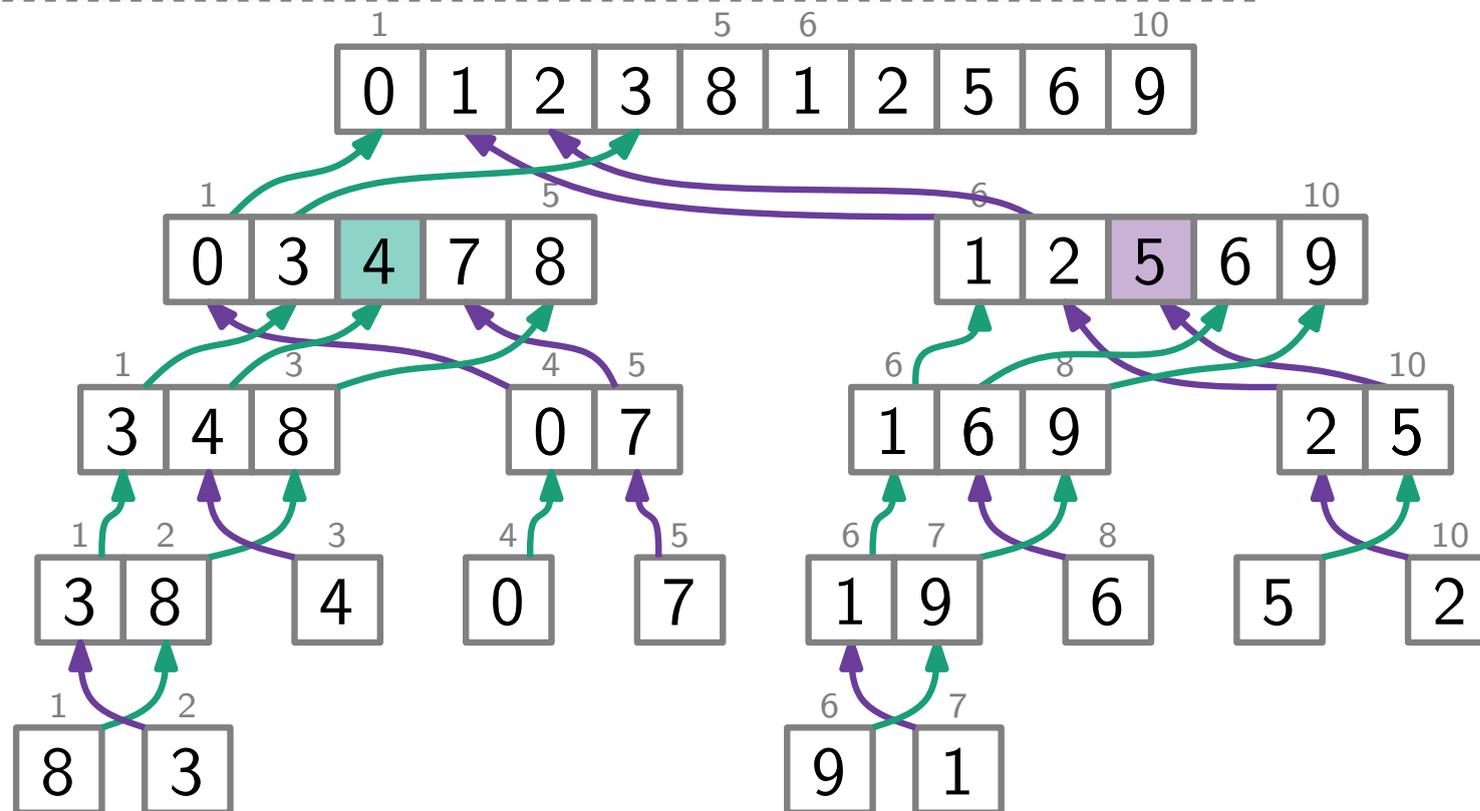
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

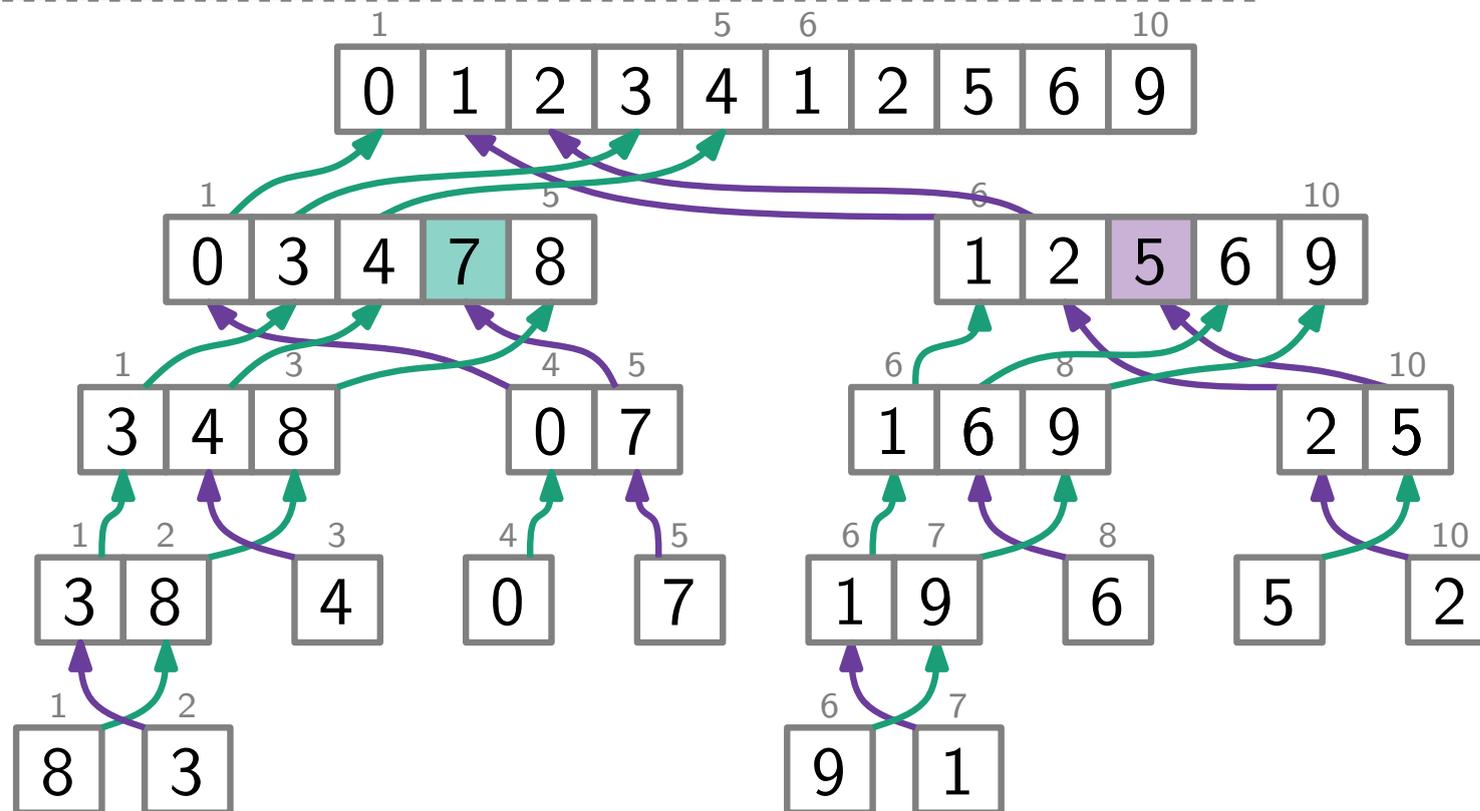
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

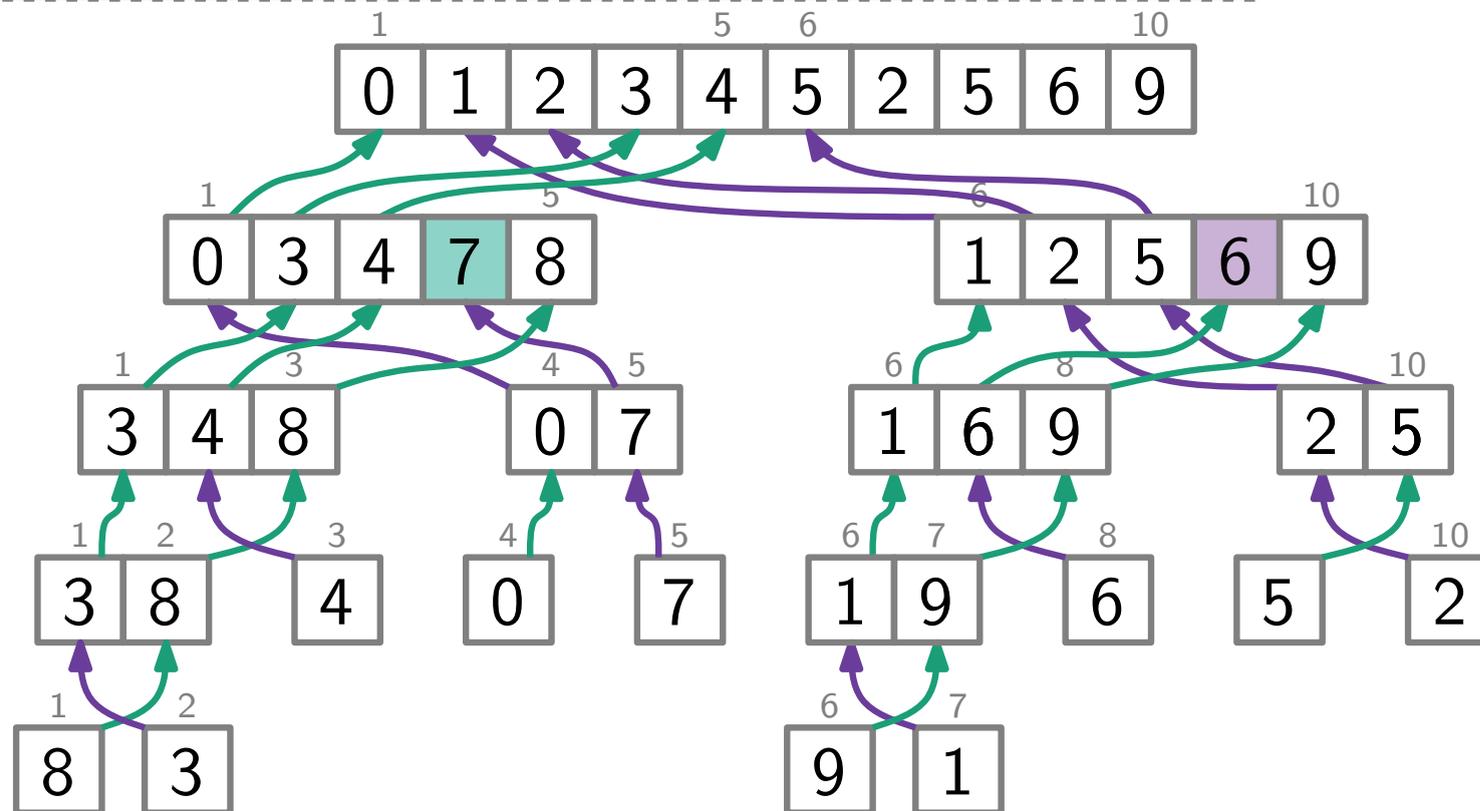
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

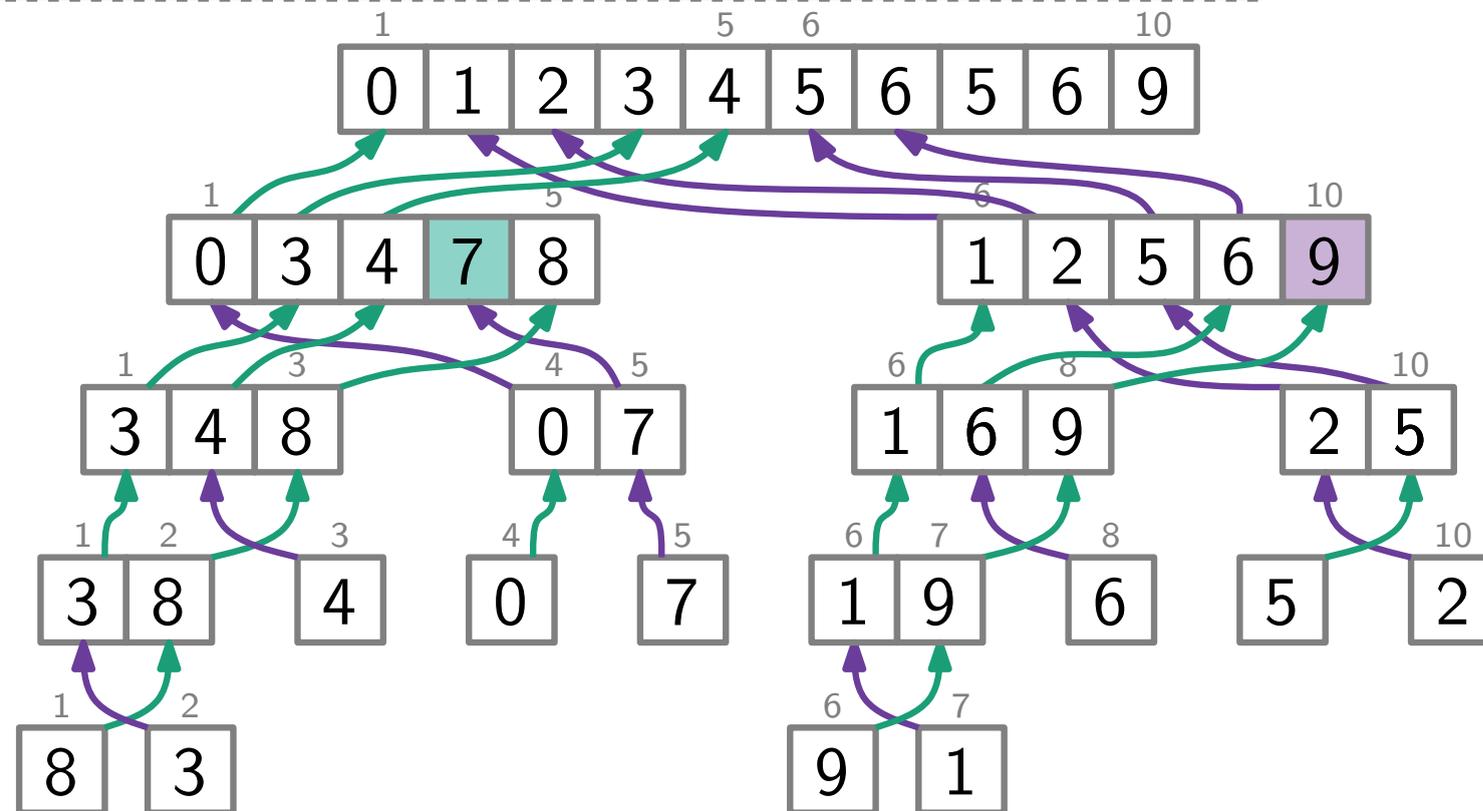
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

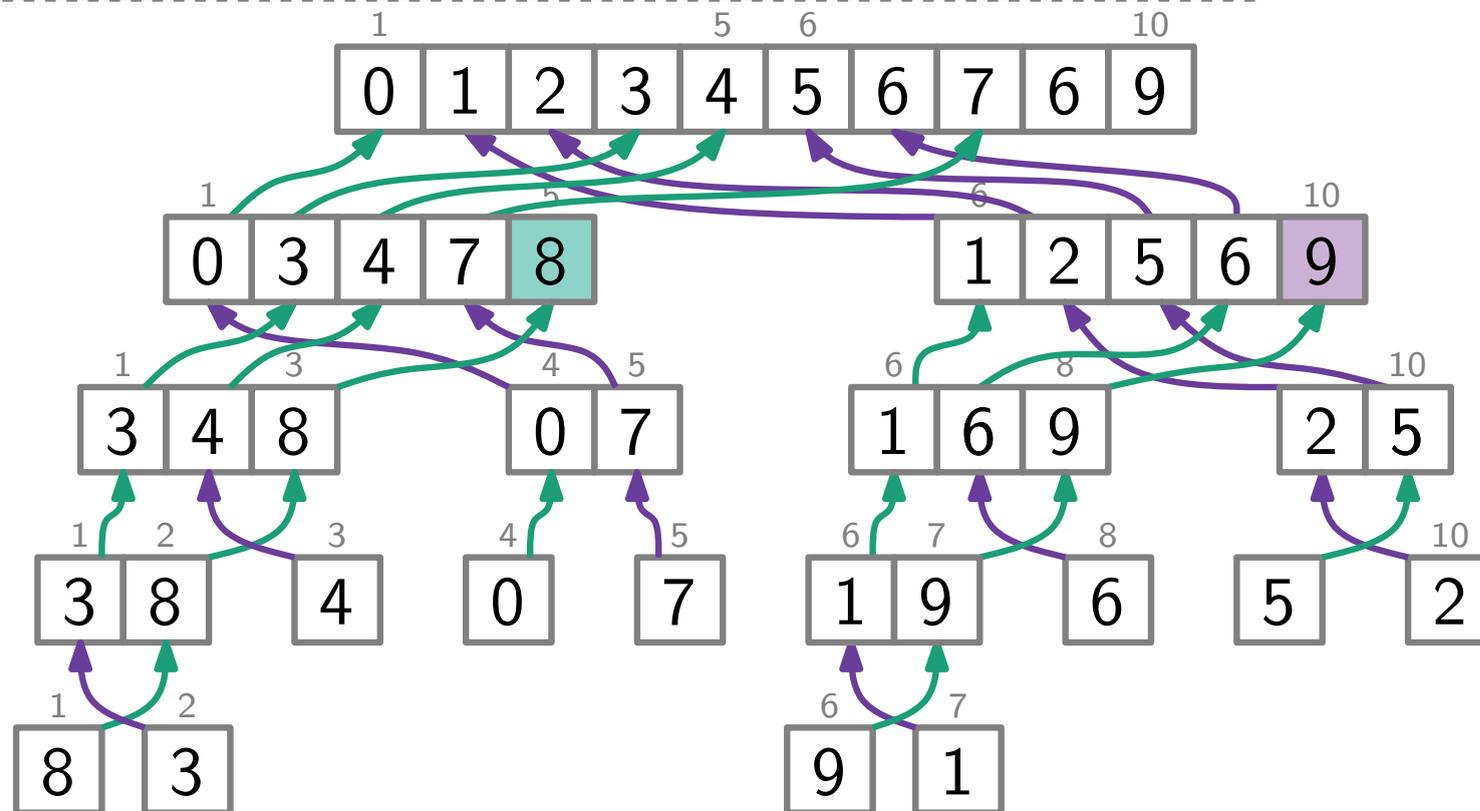
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

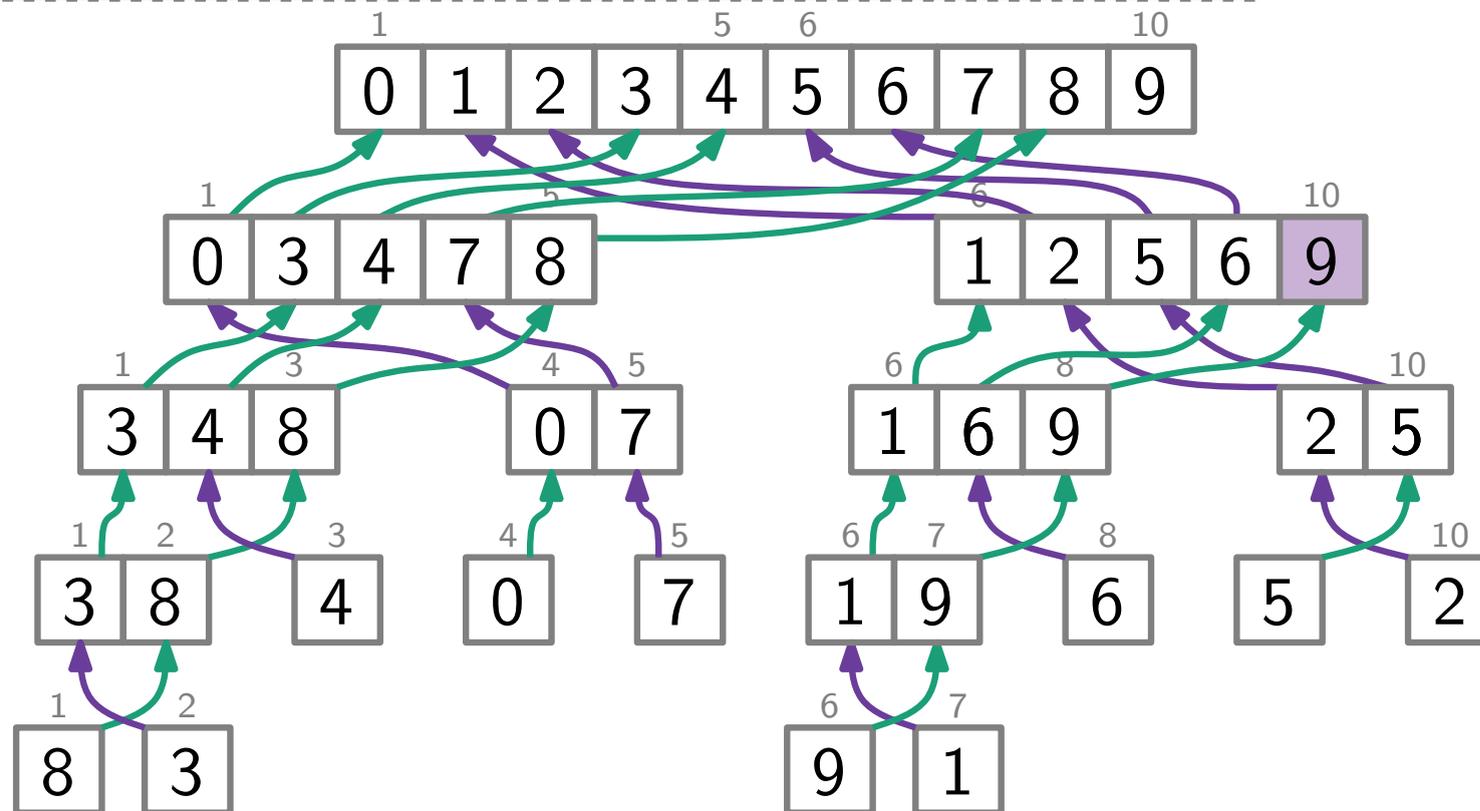
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

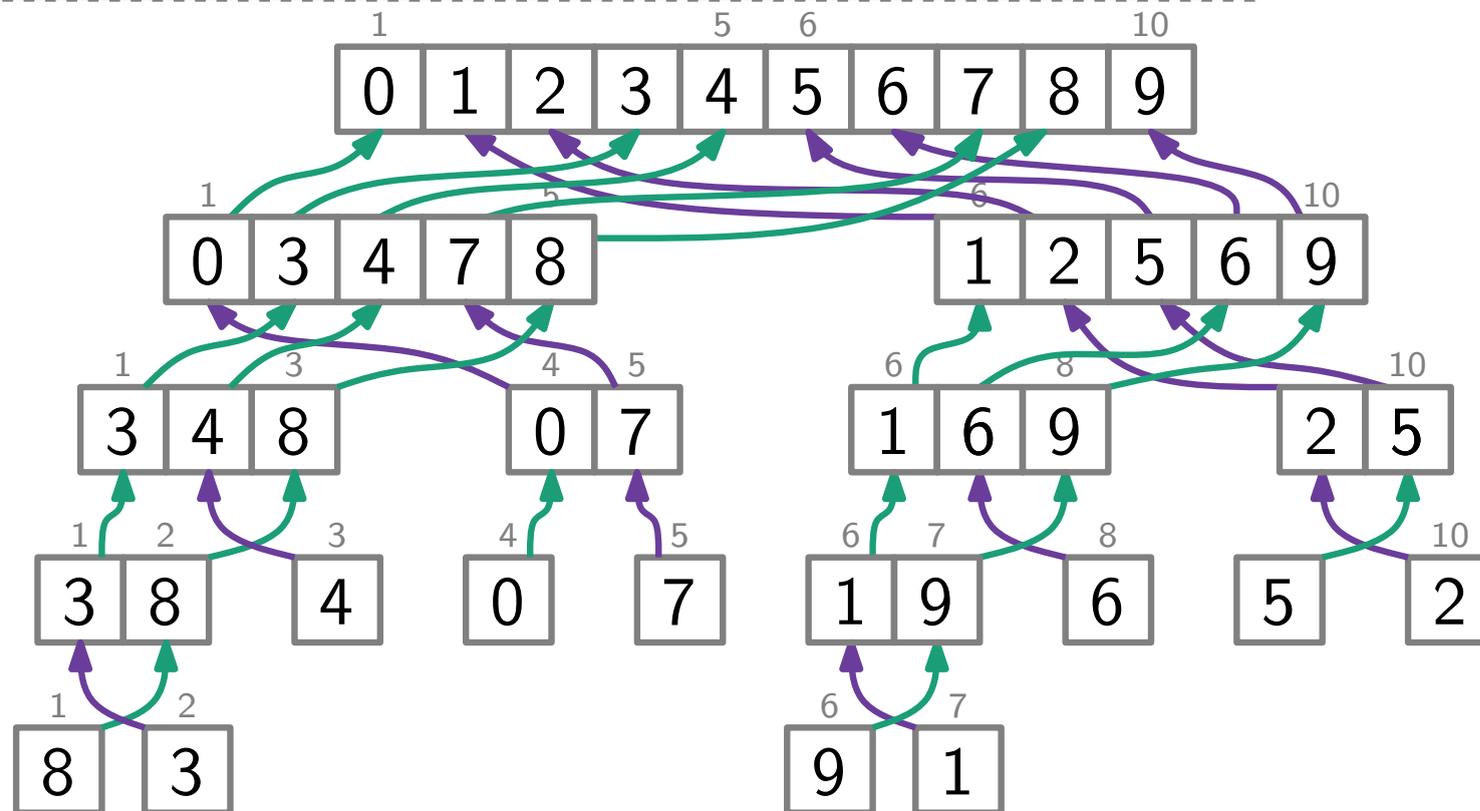
if  $l < r$  then

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

MERGESORT(A,  $l$ ,  $m$ ) } herrsche

MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } kombiniere



# MergeSort – ein Beispiel

MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )

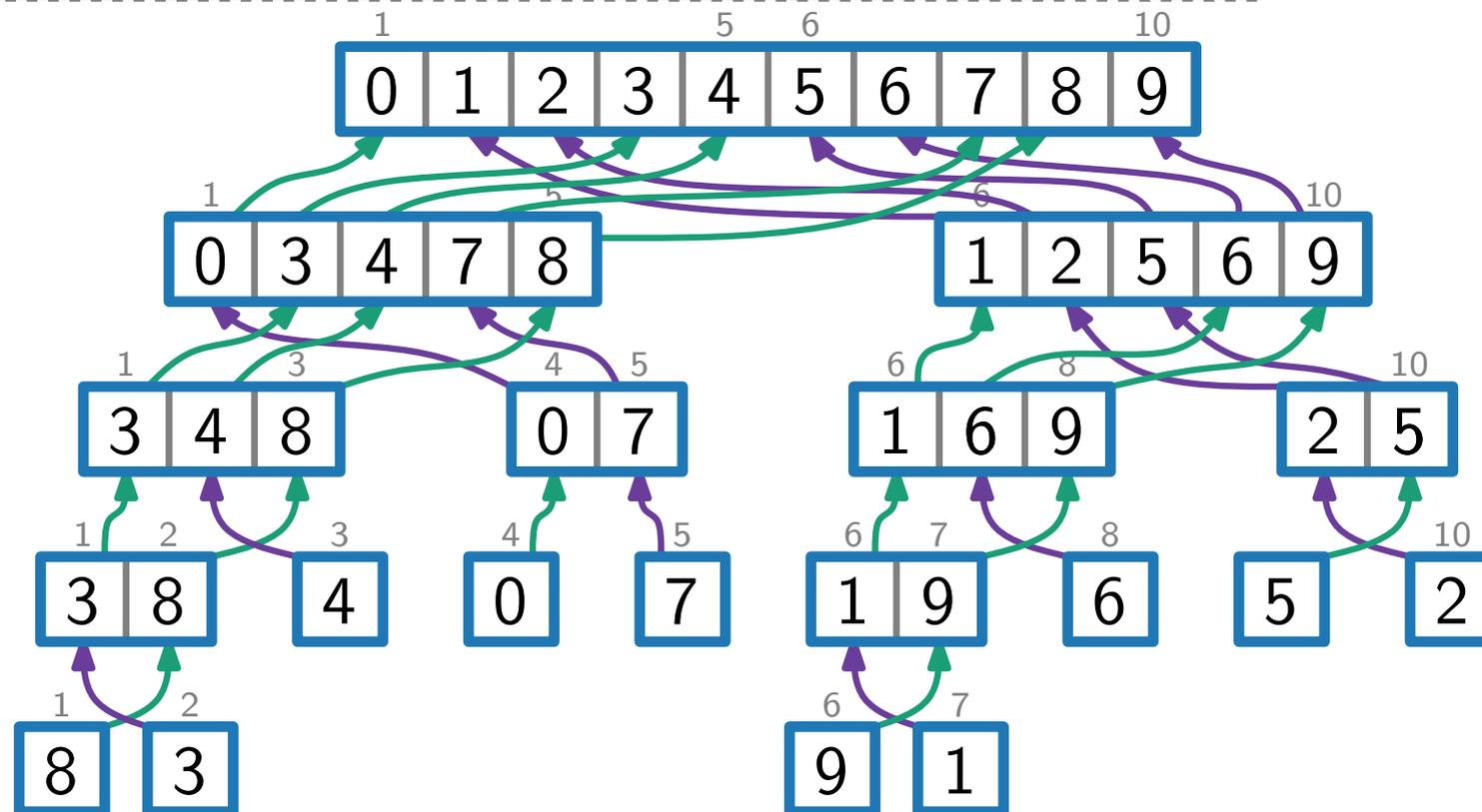
**if**  $l < r$  **then**

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

MERGESORT(A,  $l$ ,  $m$ ) } **herrsche**

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

MERGE(A,  $l$ ,  $m$ ,  $r$ ) } **kombiniere**



# MergeSort – ein Beispiel

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

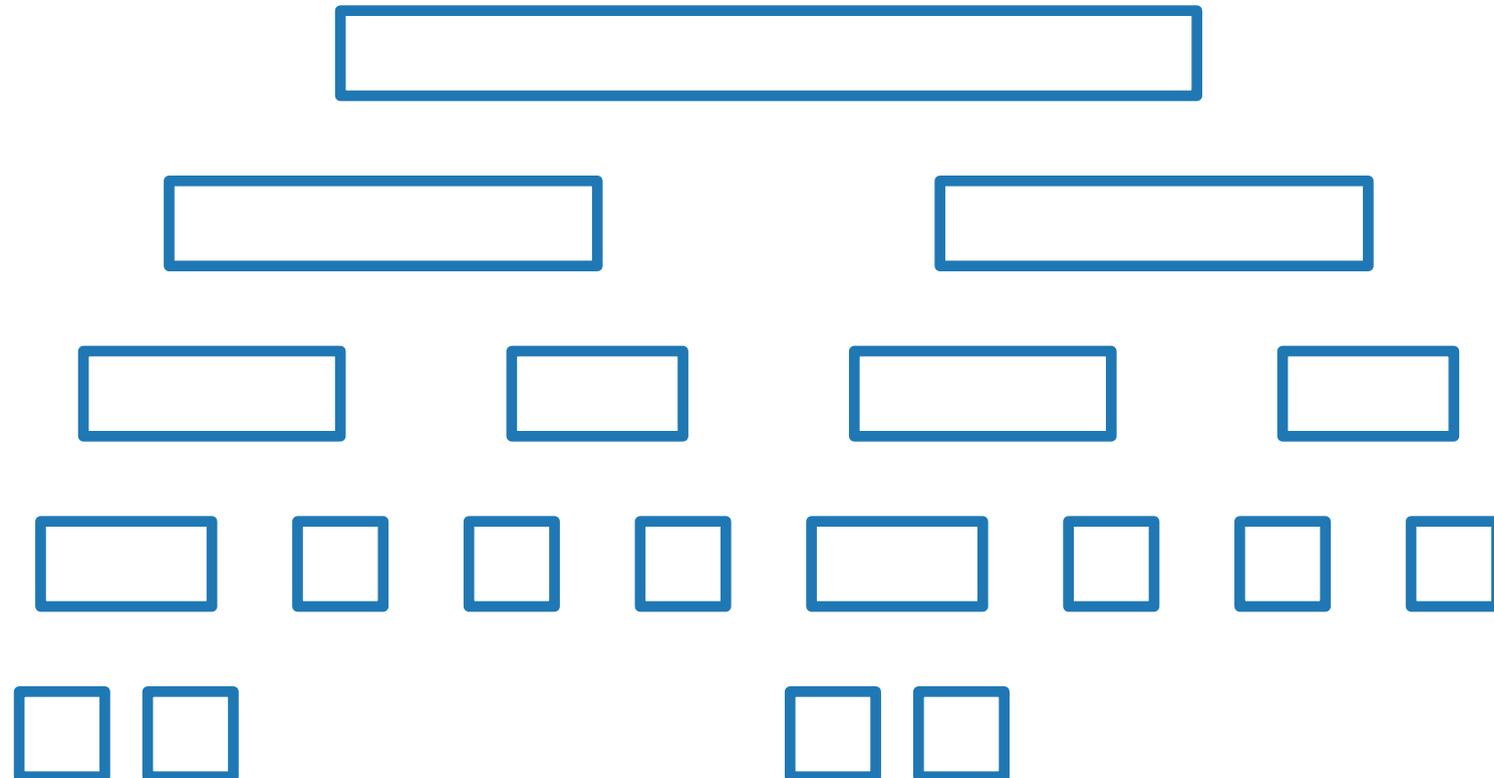
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



# MergeSort – ein Beispiel

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

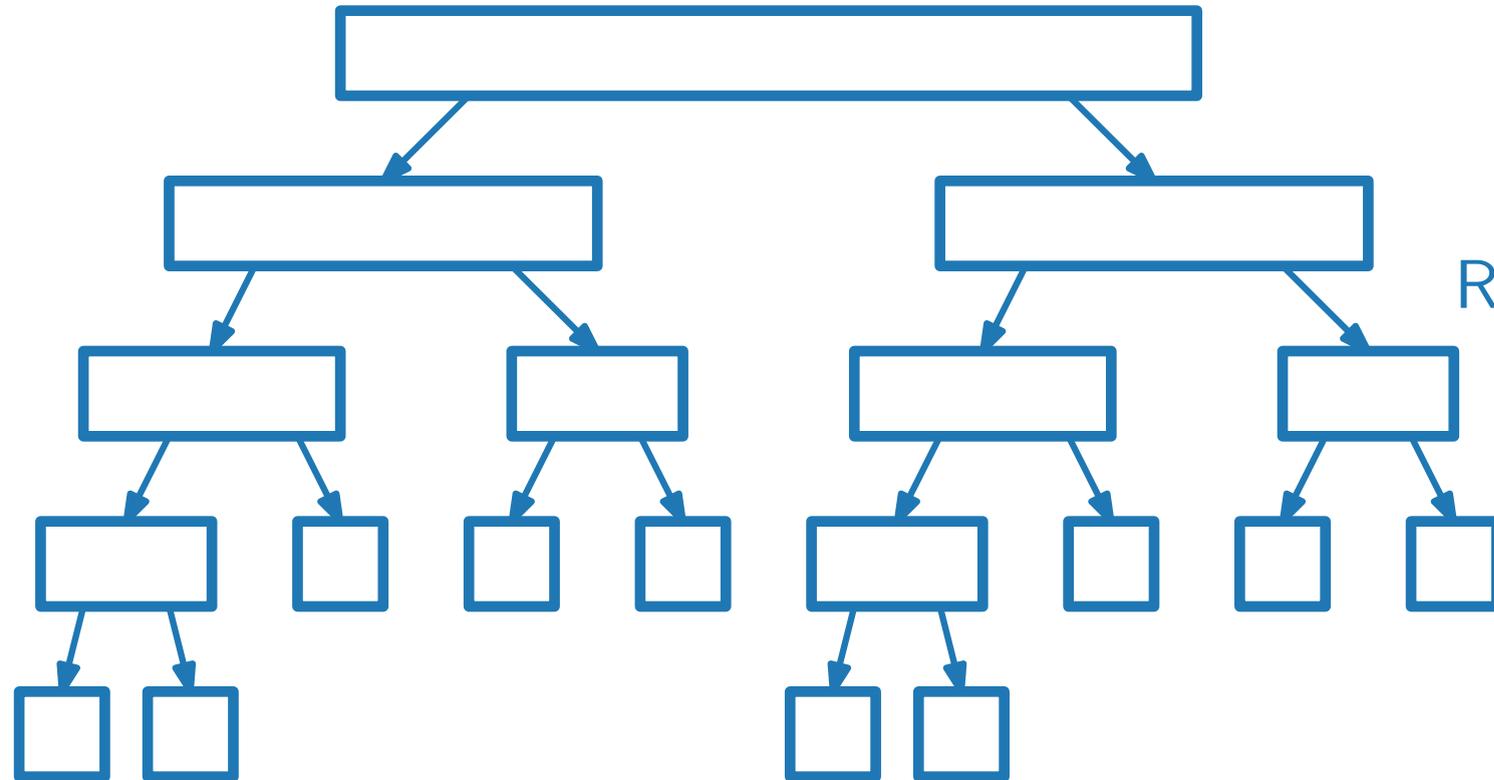
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



Rekursionsbaum von MERGESORT

# MergeSort – ein Beispiel

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

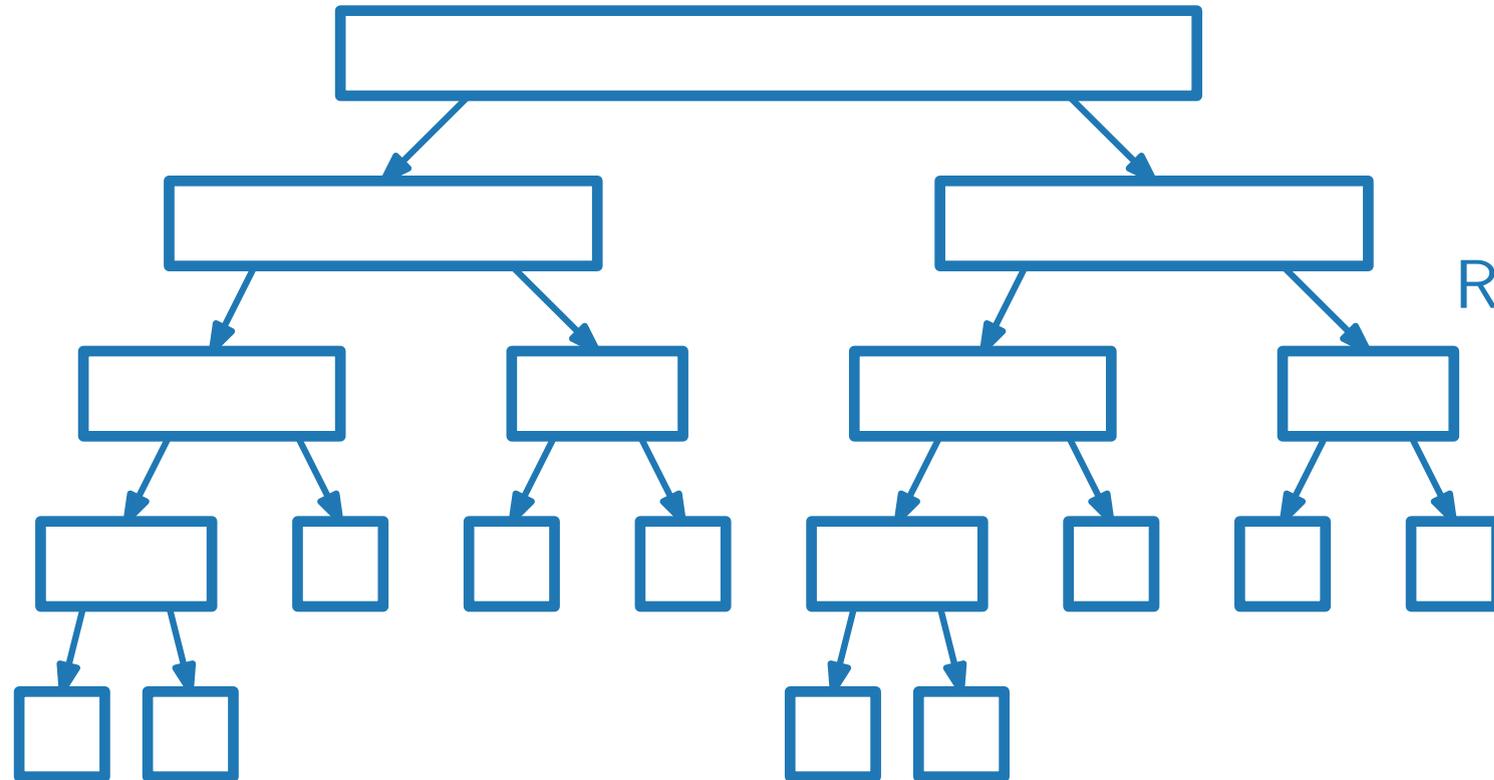
```
if  $\ell < r$  then
```

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

```
  MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
  MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```



Rekursionsbaum von MERGESORT

Baum der  
rekursiven  
Aufrufe

# MergeSort – ein Beispiel

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

```
if  $l < r$  then
```

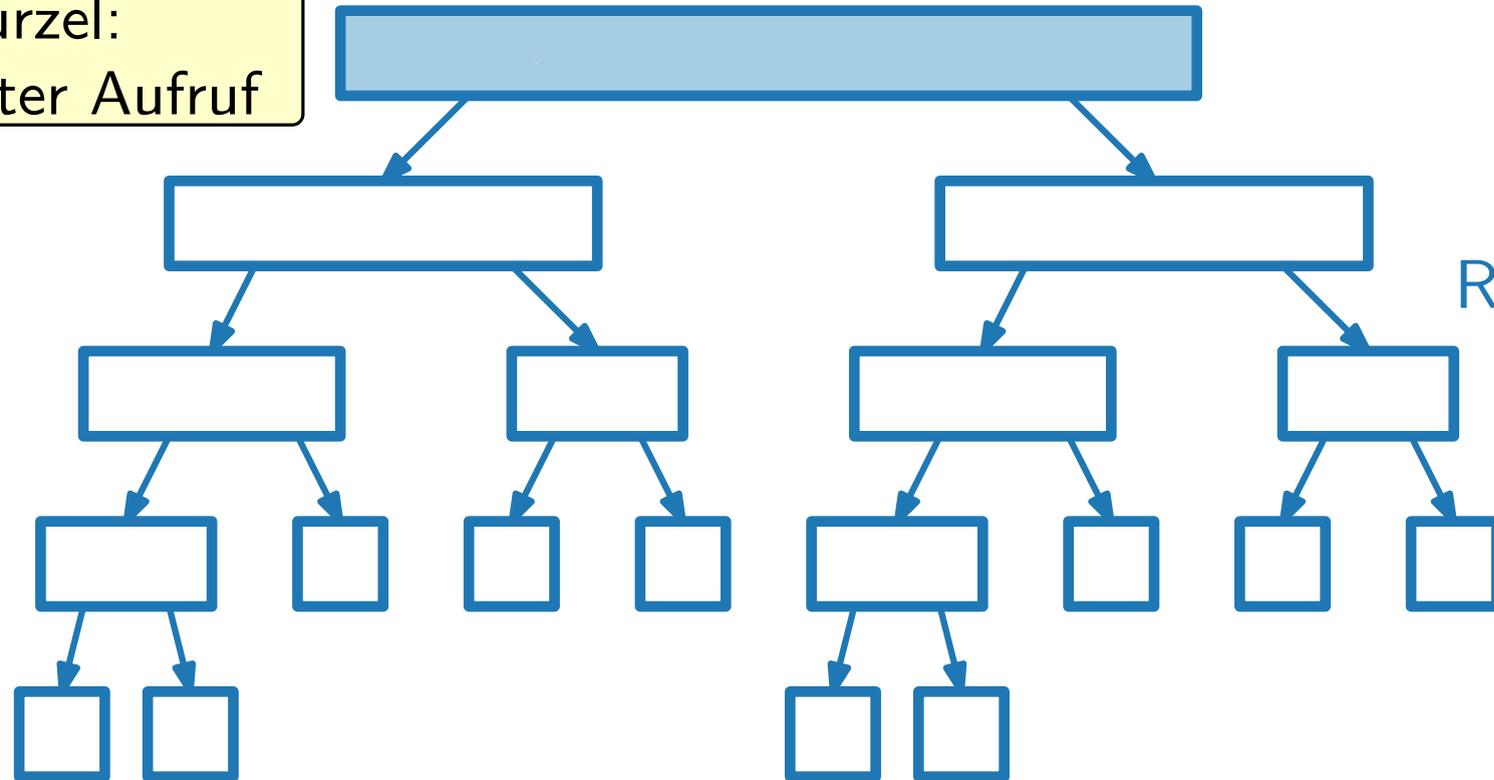
```
   $m = \lfloor (l + r) / 2 \rfloor$  } teile
```

```
  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

```
MERGESORT(int[] A, int  $l = 1$ , int  $r = A.length$ )
```

```
if  $l < r$  then
```

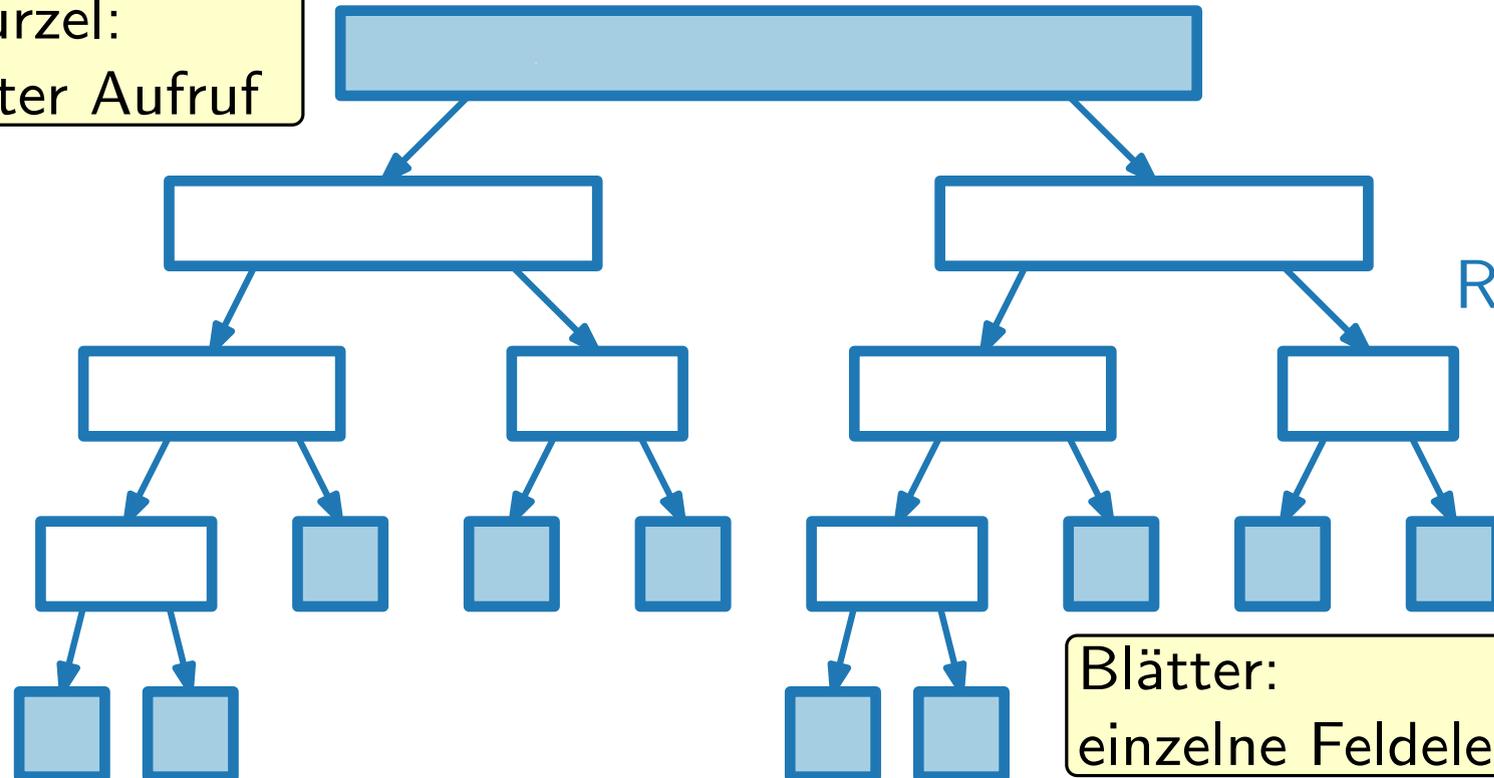
```
   $m = \lfloor (l + r) / 2 \rfloor$  } teile
```

```
  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

Blätter:  
einzelne Feldelemente

# Korrektheit von MERGE

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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“!

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

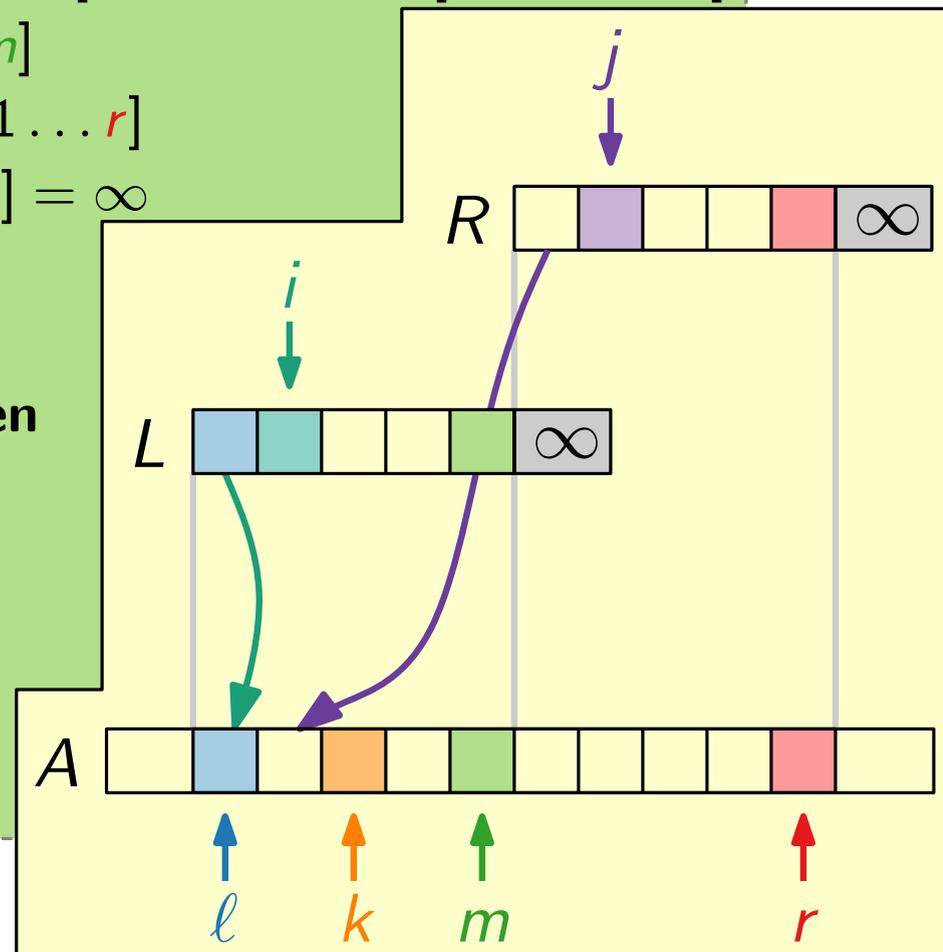
$A[k] = L[i]$

$i = i + 1$

  else

$A[k] = R[j]$

$j = j + 1$



# Korrektheit von MERGE

... nach Schema „F“!

## 0. Schleifeninvariante

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

```
MERGE(int[] A, int  $l$ , int  $m$ , int  $r$ )
```

```
 $n_1 = m - l + 1$ ;  $n_2 = r - m$ 
```

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

```
 $L[1 \dots n_1] = A[l \dots m]$ 
```

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

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

```
 $i = j = 1$ 
```

```
for  $k = l$  to  $r$  do
```

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

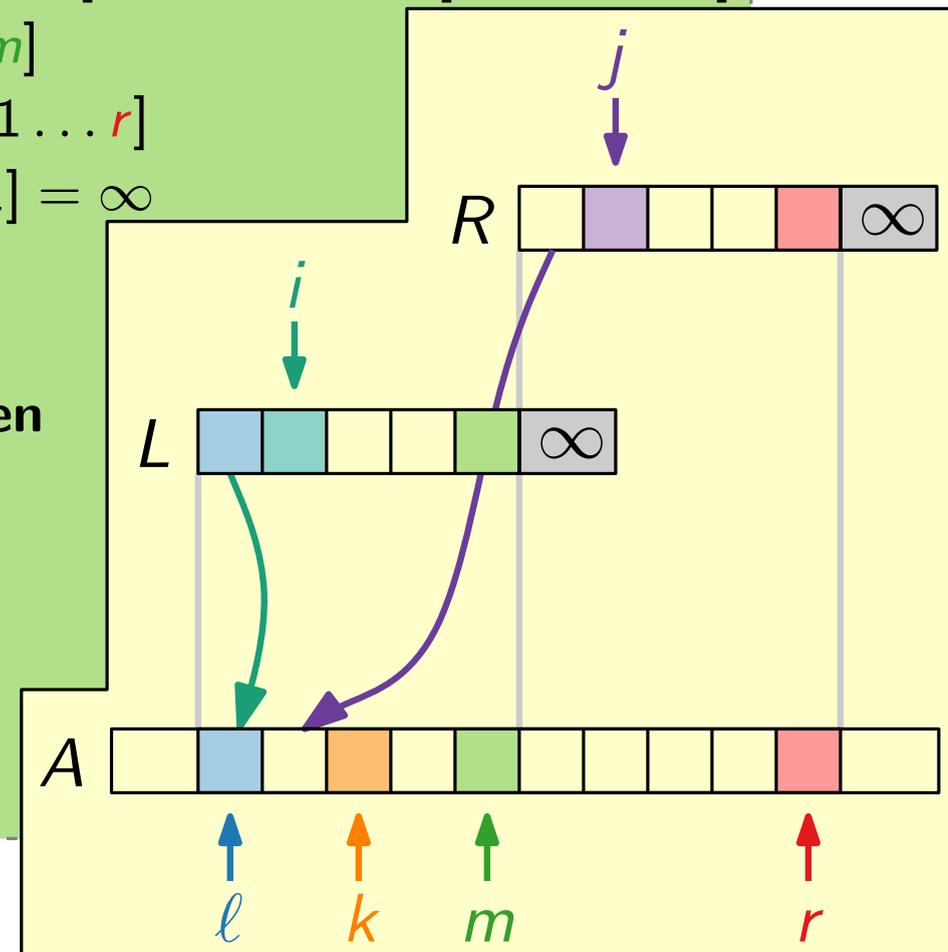
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  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  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

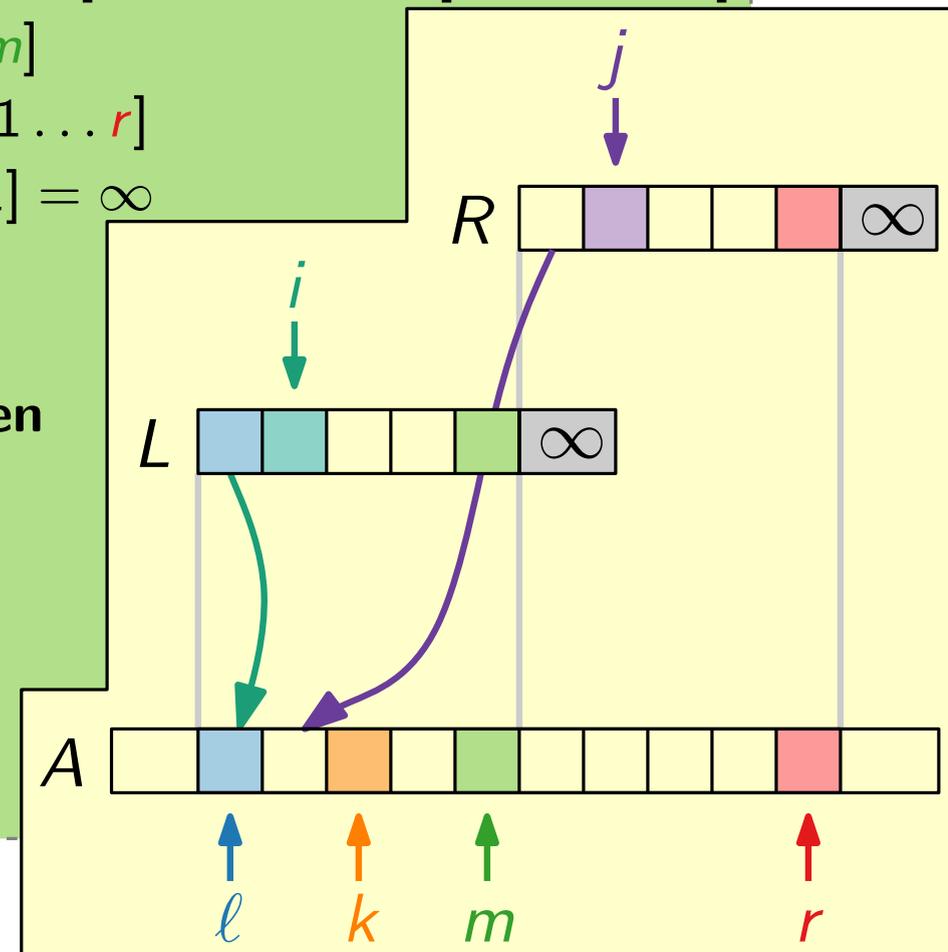
$A[k] = L[i]$

$i = i + 1$

  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  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

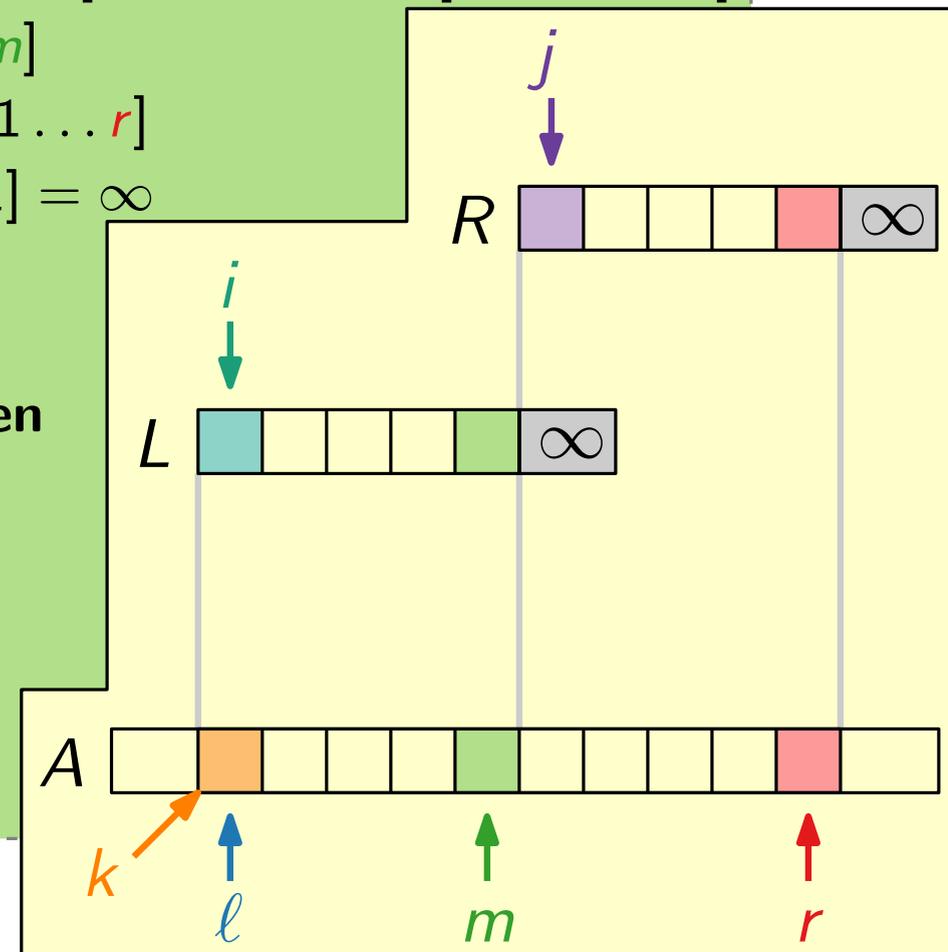
$A[k] = L[i]$

$i = i + 1$

  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  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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  $L \cup R$ .
- Da  $i = j = 1$ , sind  $L[i]$  und  $R[j]$  die kleinsten noch nicht kopierten Elemente

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

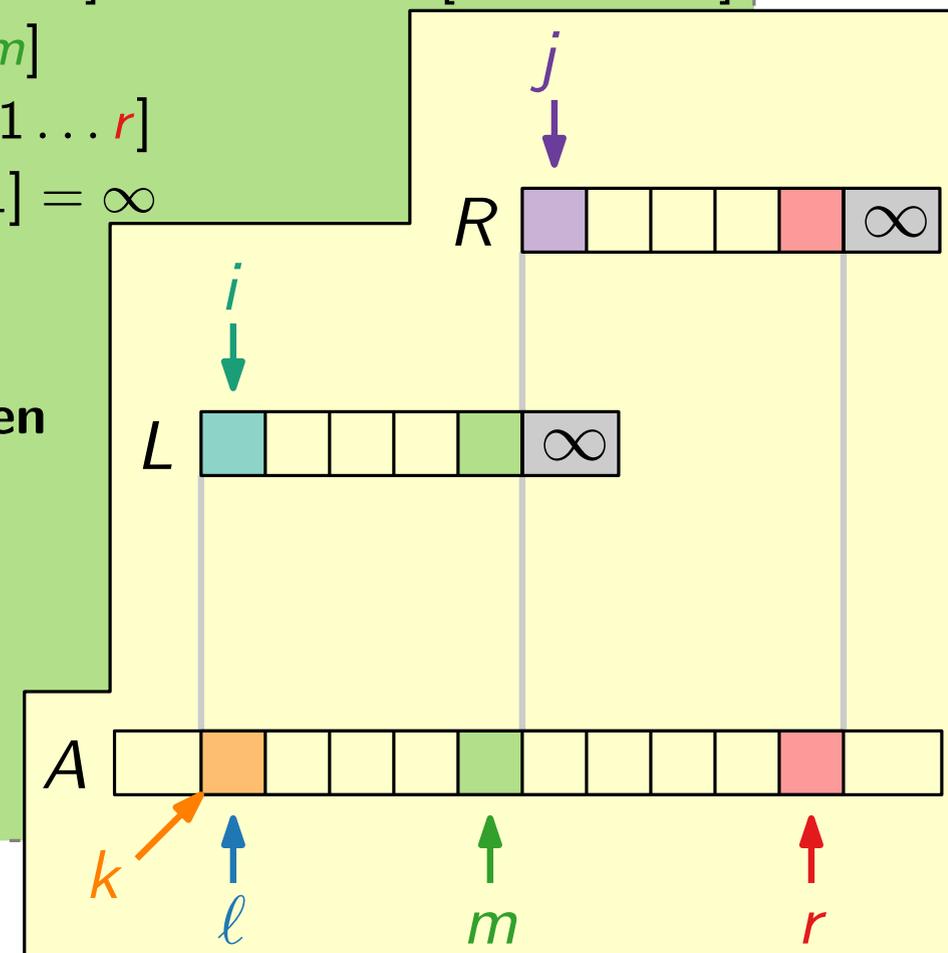
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  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  $L \cup R$ .
- Da  $i = j = 1$ , sind  $L[i]$  und  $R[j]$  die kleinsten noch nicht kopierten Elemente

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

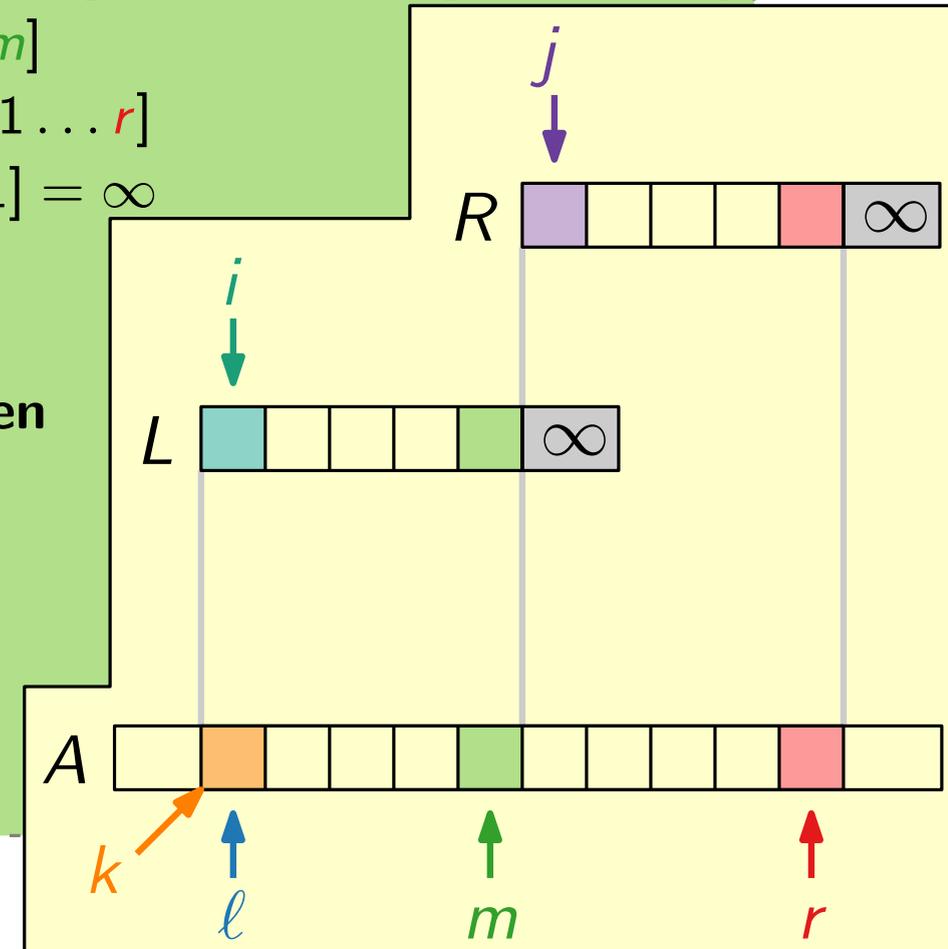
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  else
```

```
     $A[k] = R[j]$ 
```

```
     $j = j + 1$ 
```



# Korrektheit von MERGE

... nach Schema „F“!

## 0. Schleifeninvariante

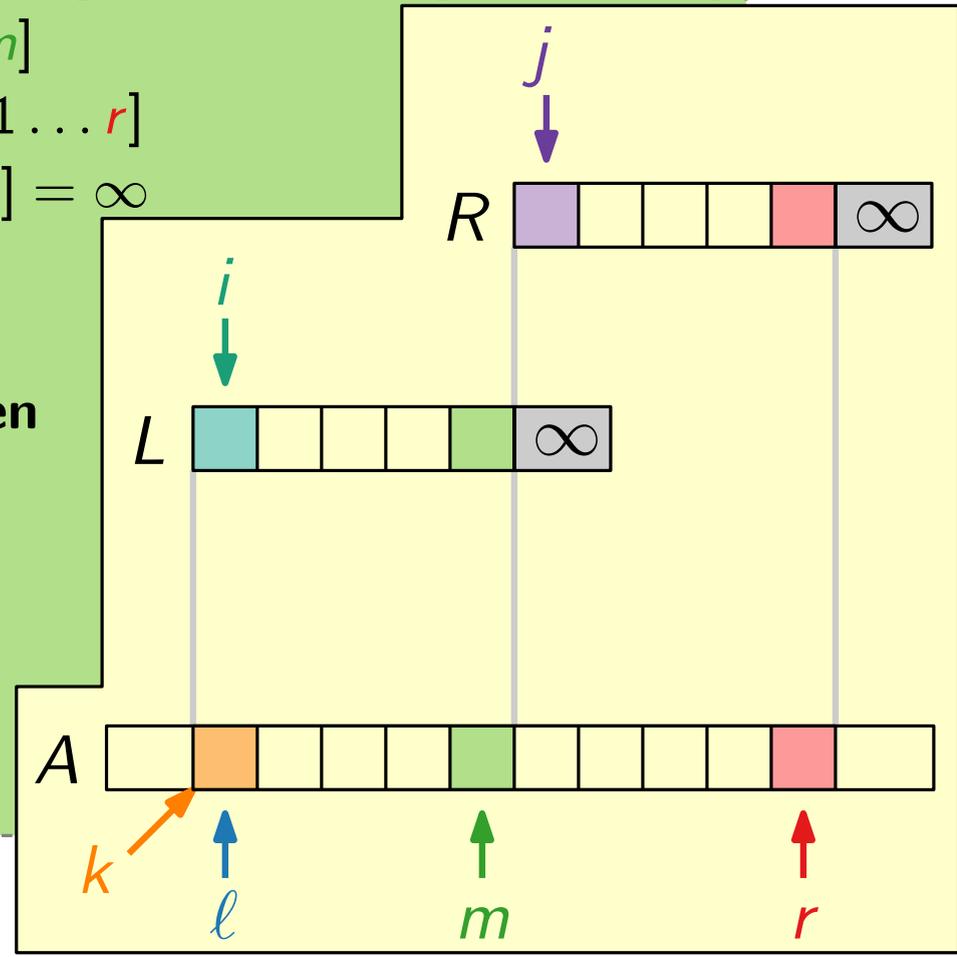
- $A[\ell \dots k - 1]$  enthält die  $k - \ell$  kleinsten Elemente von  $L \cup R$  **sortiert**.
- $L[i]$  und  $R[j]$  sind die kleinsten Elemente in  $L$  bzw.  $R$ , die noch nicht in  $A$  kopiert wurden.

## 1. Initialisierung ✓

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

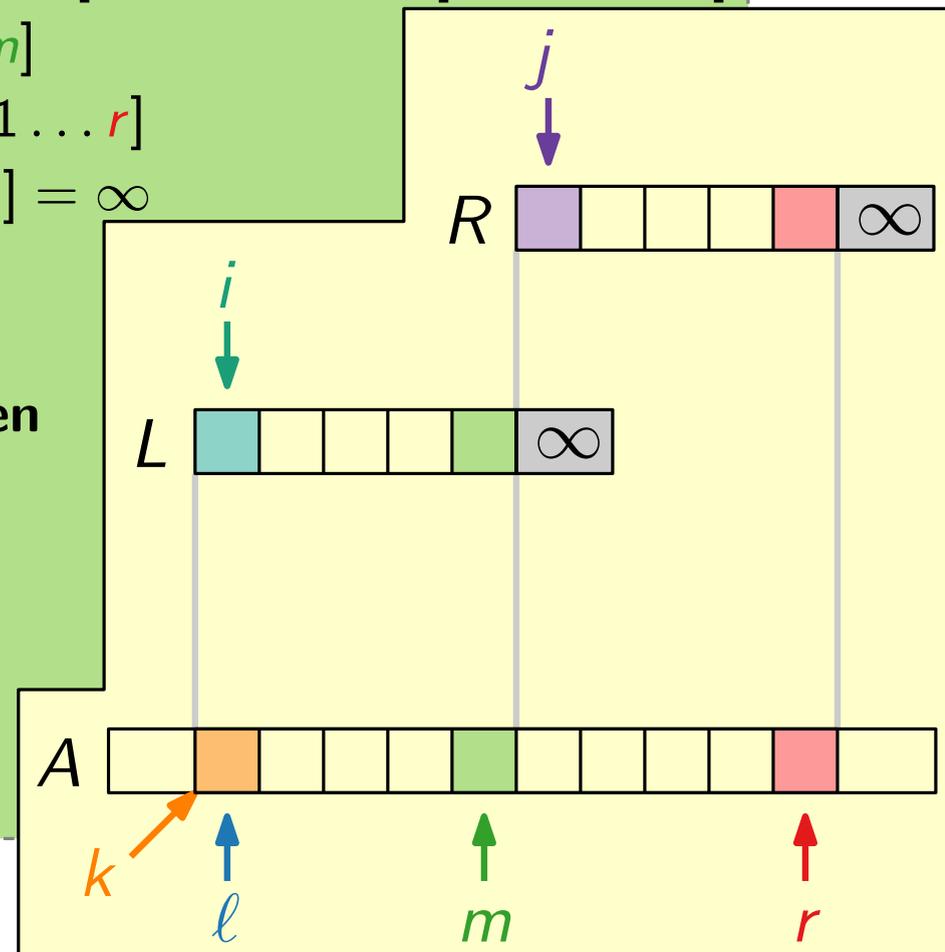
$A[k] = L[i]$

$i = i + 1$

  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

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

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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

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

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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

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

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

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

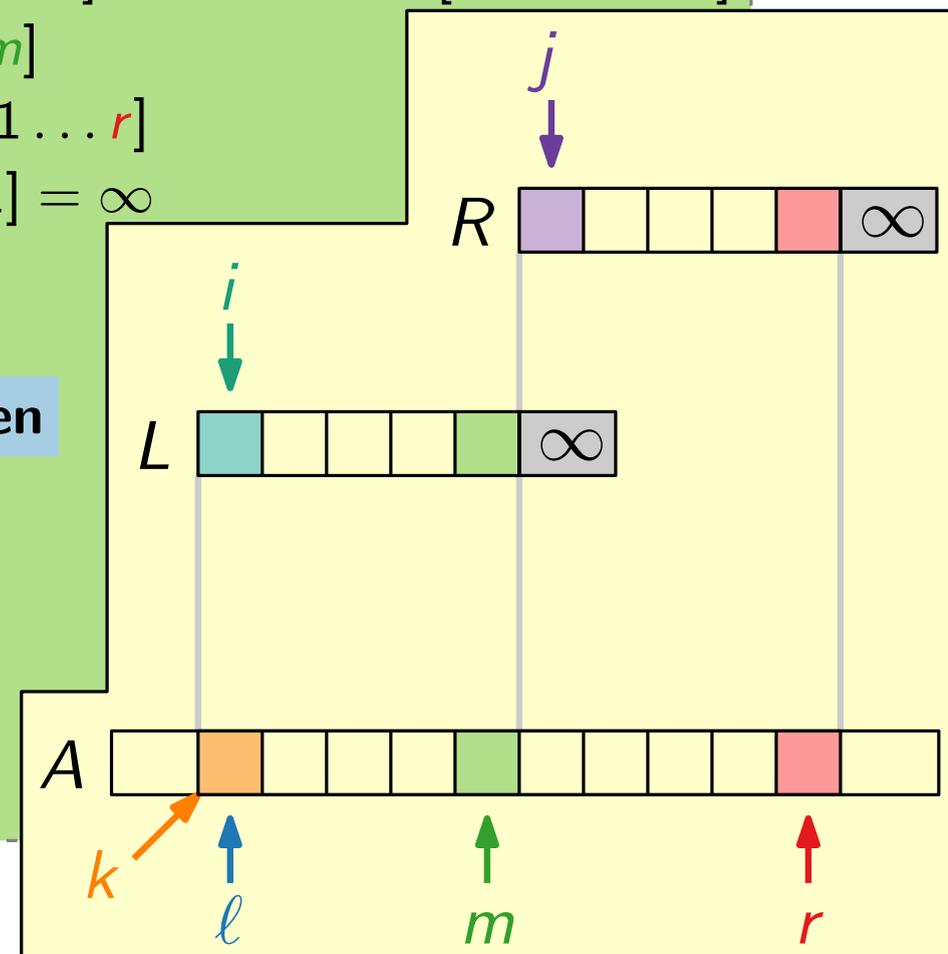
$A[k] = L[i]$

$i = i + 1$

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

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

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

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

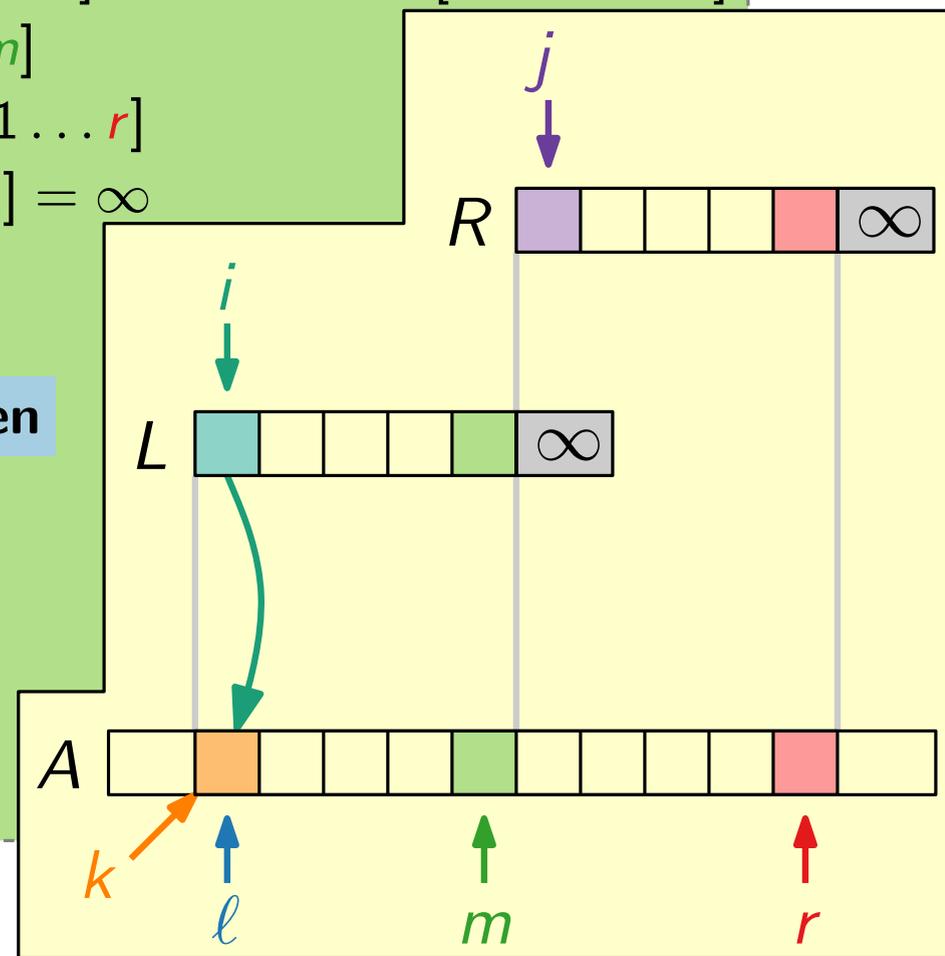
$A[k] = L[i]$

$i = i + 1$

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

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

■ Nun gilt:

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

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

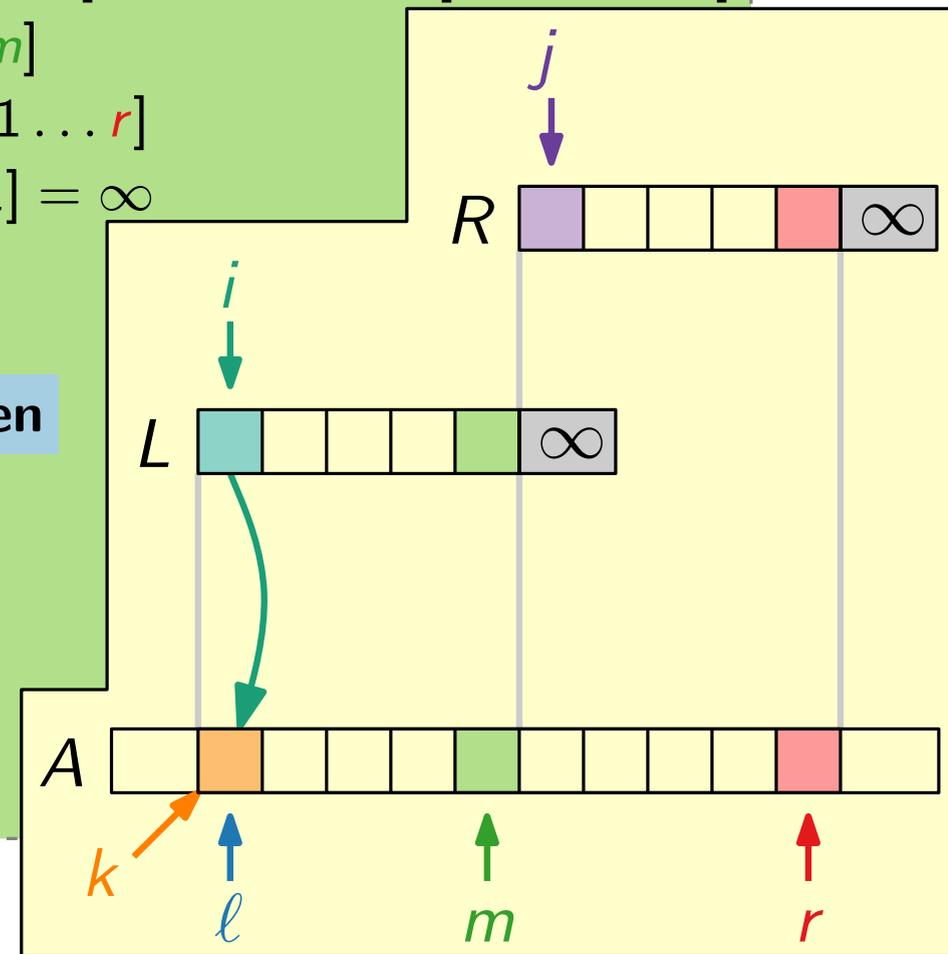
$A[k] = L[i]$

$i = i + 1$

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

- 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.

dank Invariante

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

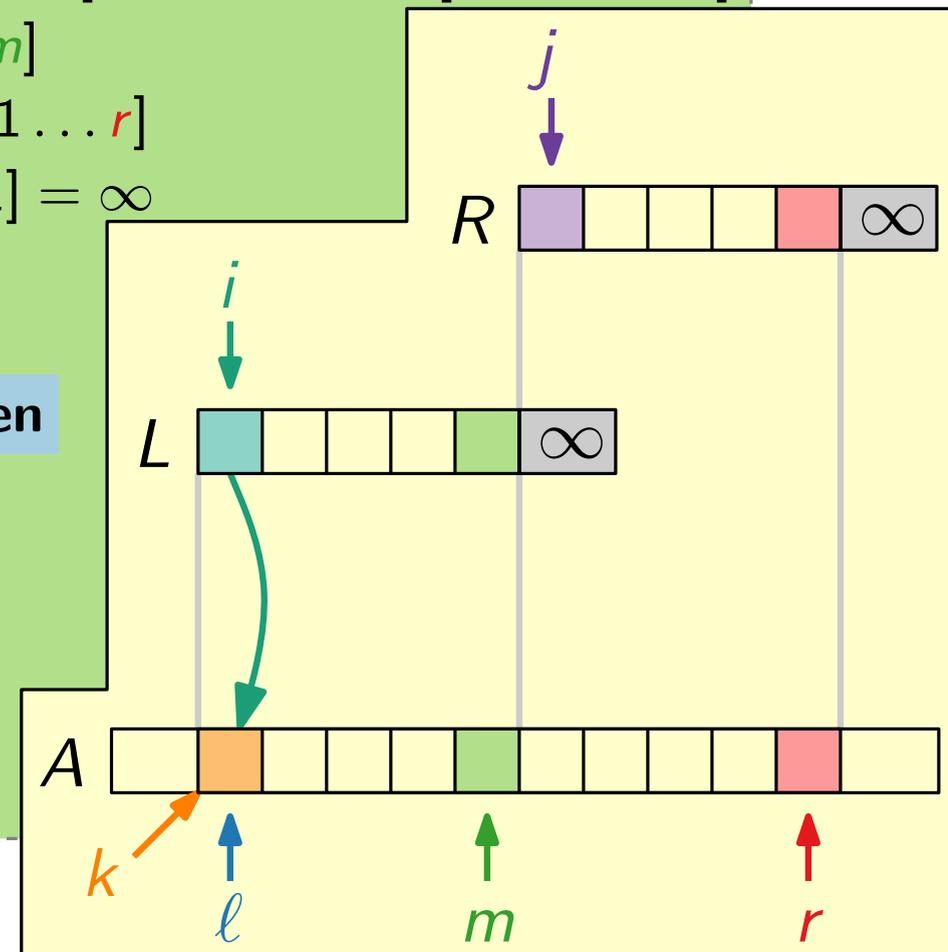
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  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

- 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

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

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

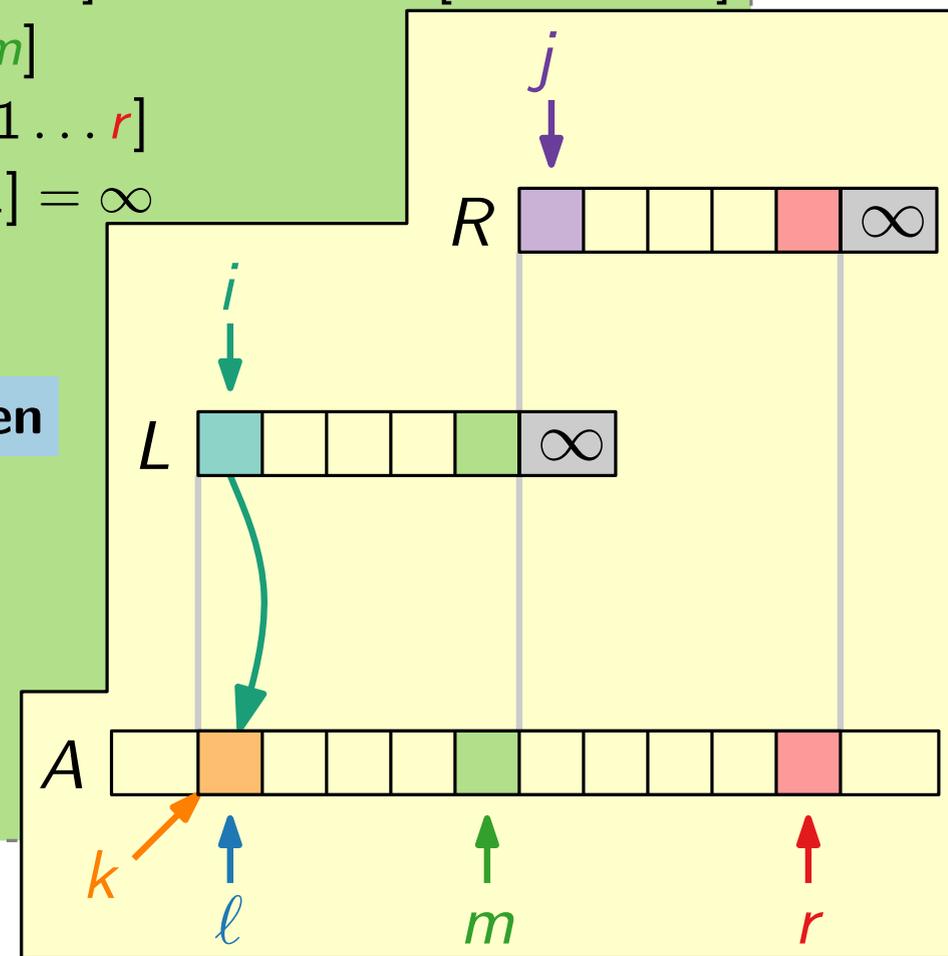
$A[k] = L[i]$

$i = i + 1$

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

- 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  $i$

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

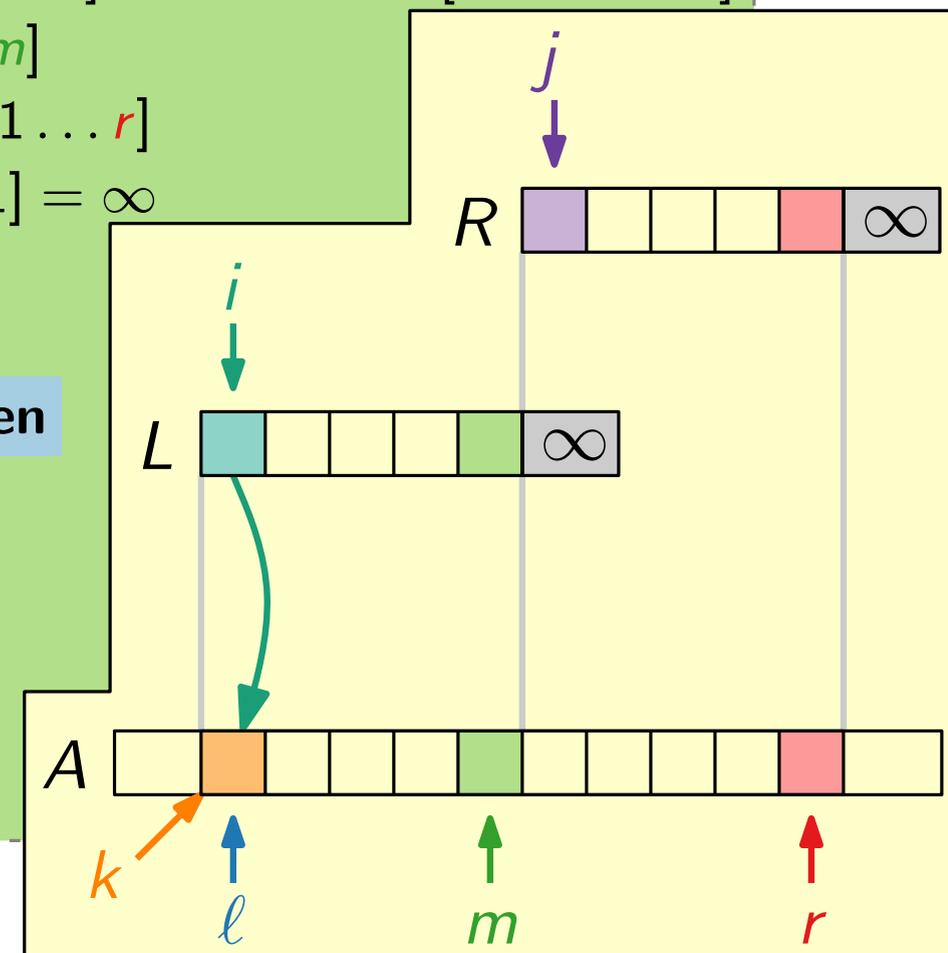
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  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.

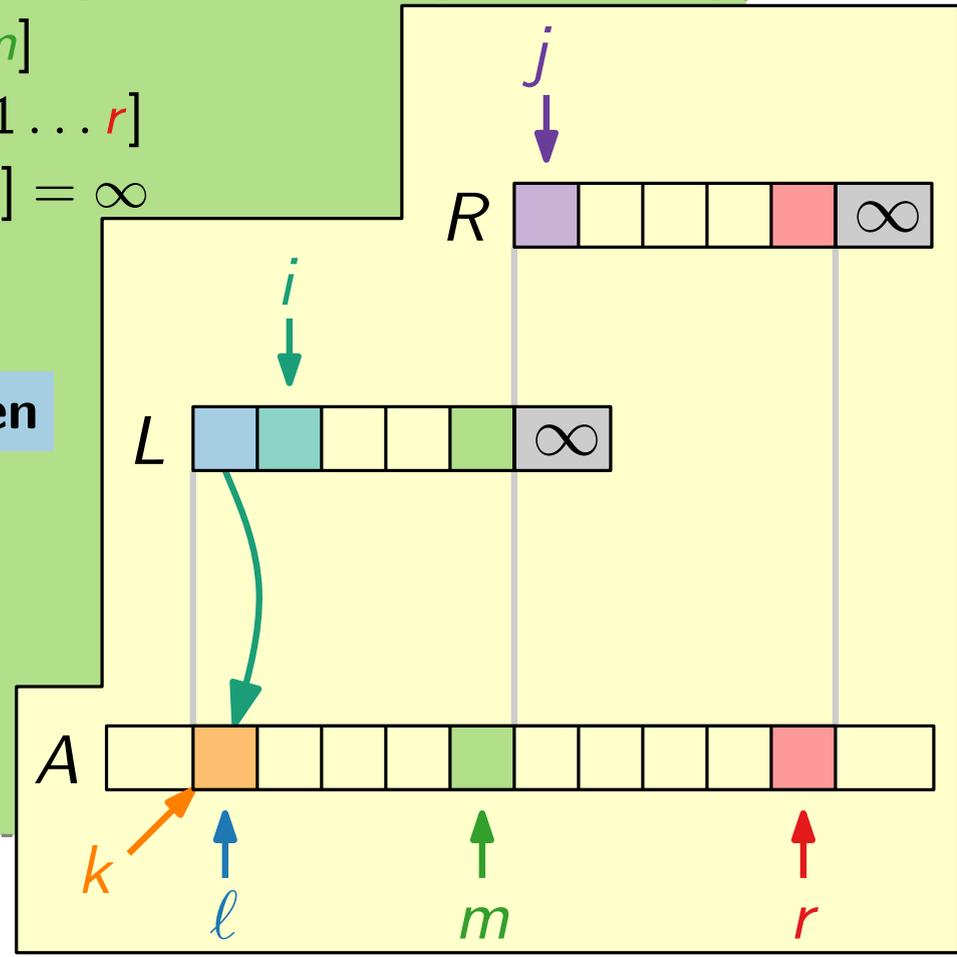
## 1. Initialisierung ✓

## 2. Aufrechterhaltung

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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.

dank Invariante

- $L[i + 1]$  ist kleinstes noch nicht kopiertes Element in  $L$ .

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

- 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  $i \Rightarrow$

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

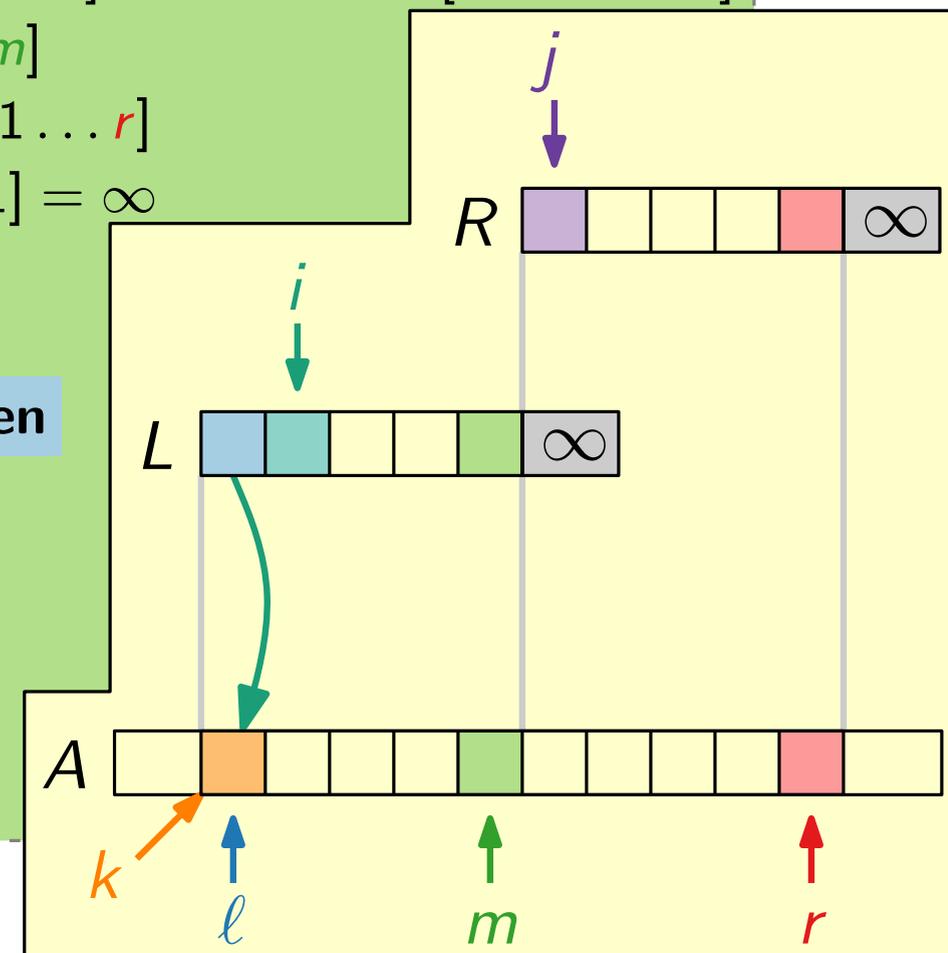
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  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

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

Nun gilt:

dank Invariante

erhöhe  $i \Rightarrow$

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

- $L[i + 1]$  ist kleinstes noch nicht kopiertes Element in  $L$ .

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

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

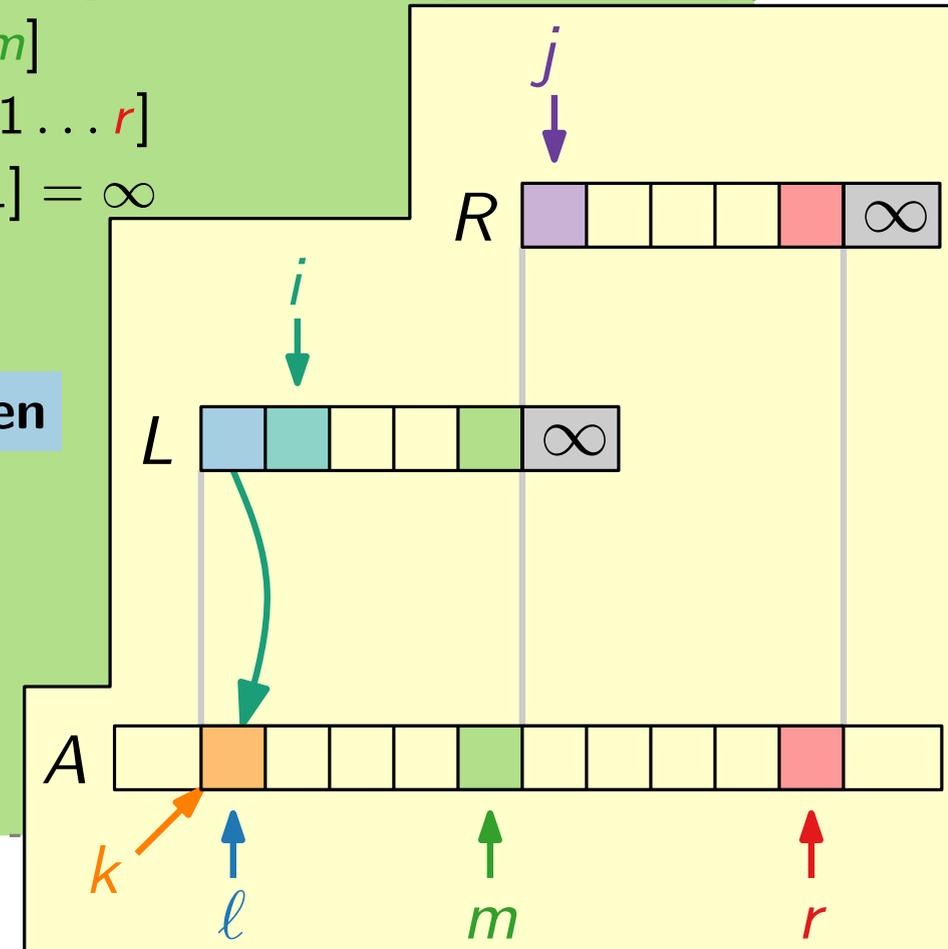
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  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.

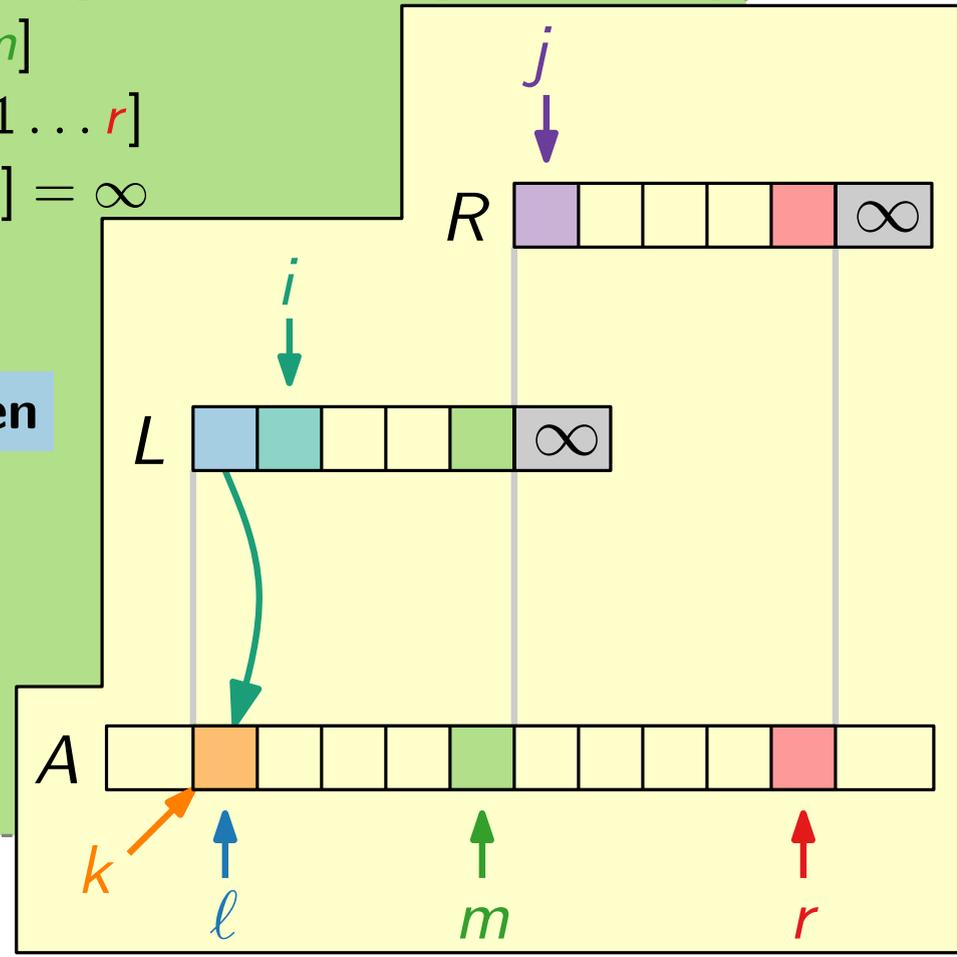
## 1. Initialisierung ✓

## 2. Aufrechterhaltung

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
   $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
   $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
   $L[1 \dots n_1] = A[\ell \dots m]$ 
   $R[1 \dots n_2] = A[m + 1 \dots r]$ 
   $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
   $i = j = 1$ 
  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  $i \Rightarrow$   
 erhöhe  $k$

$L[i]$  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.

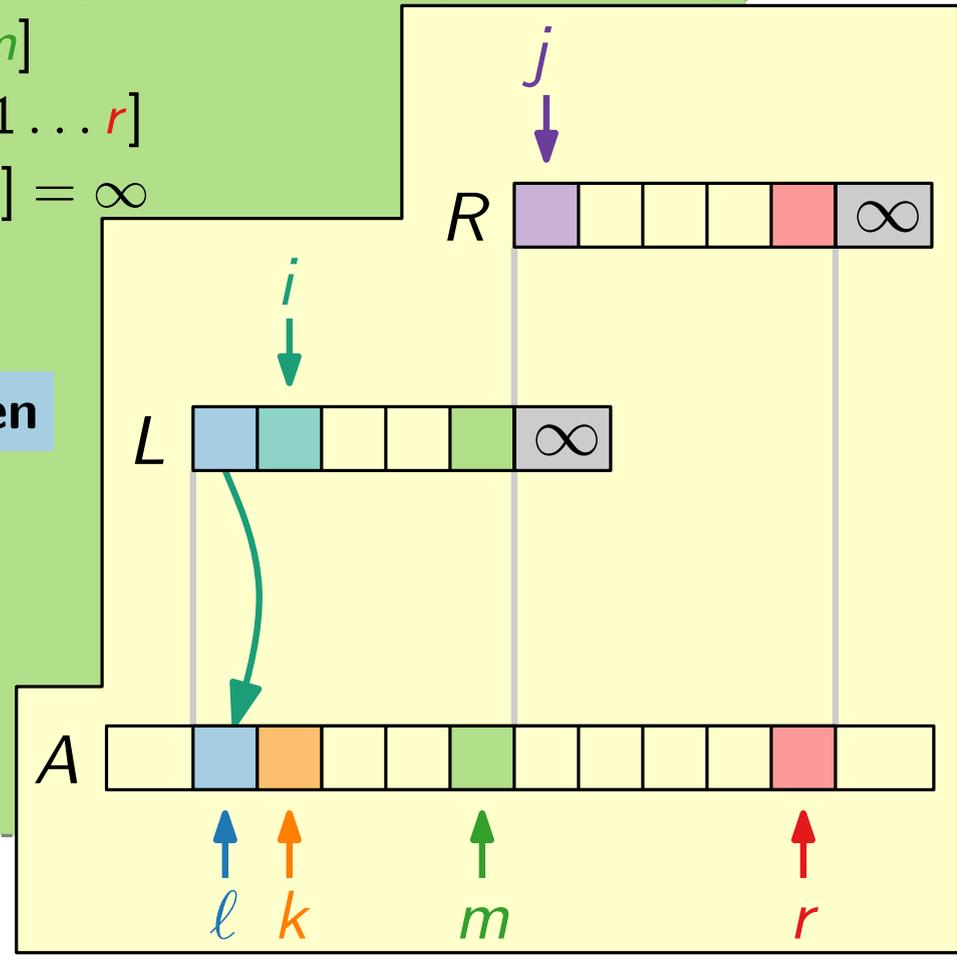
## 1. Initialisierung ✓

## 2. Aufrechterhaltung

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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

- 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  $k \Rightarrow$

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

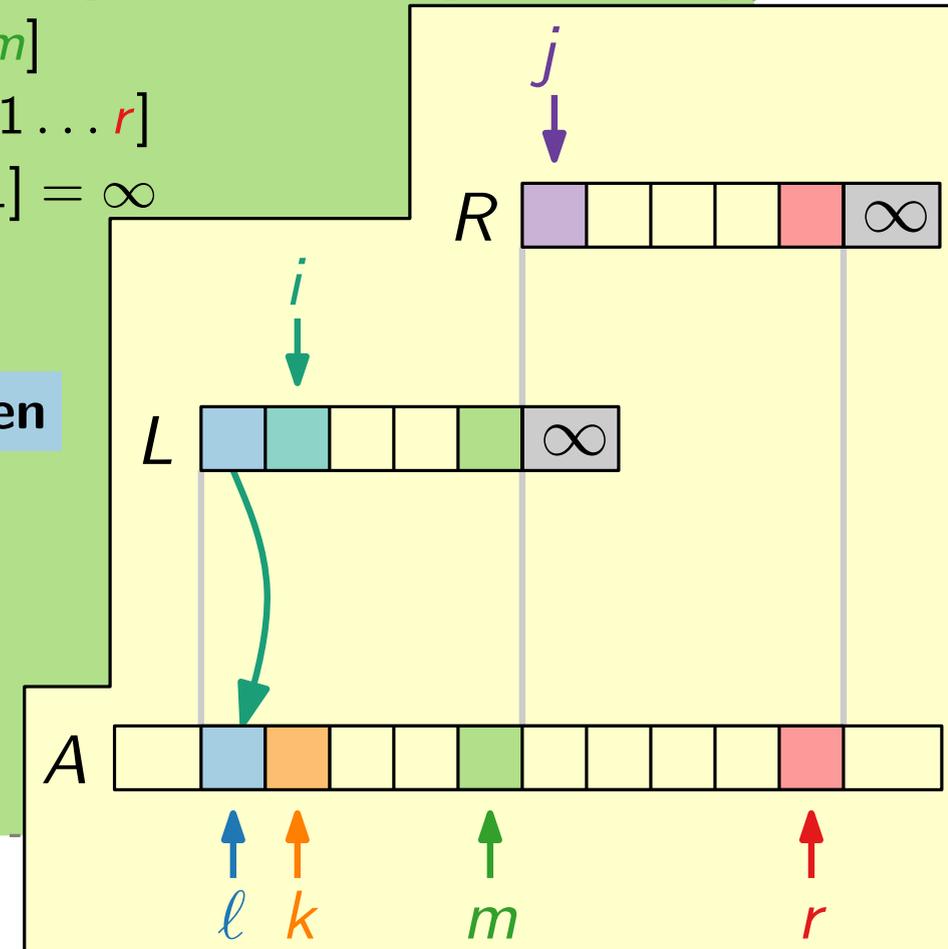
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  else
```

```
     $A[k] = R[j]$ 
```

```
     $j = j + 1$ 
```



# Korrektheit von MERGE

... nach Schema „F“!

## 0. Schleifeninvariante

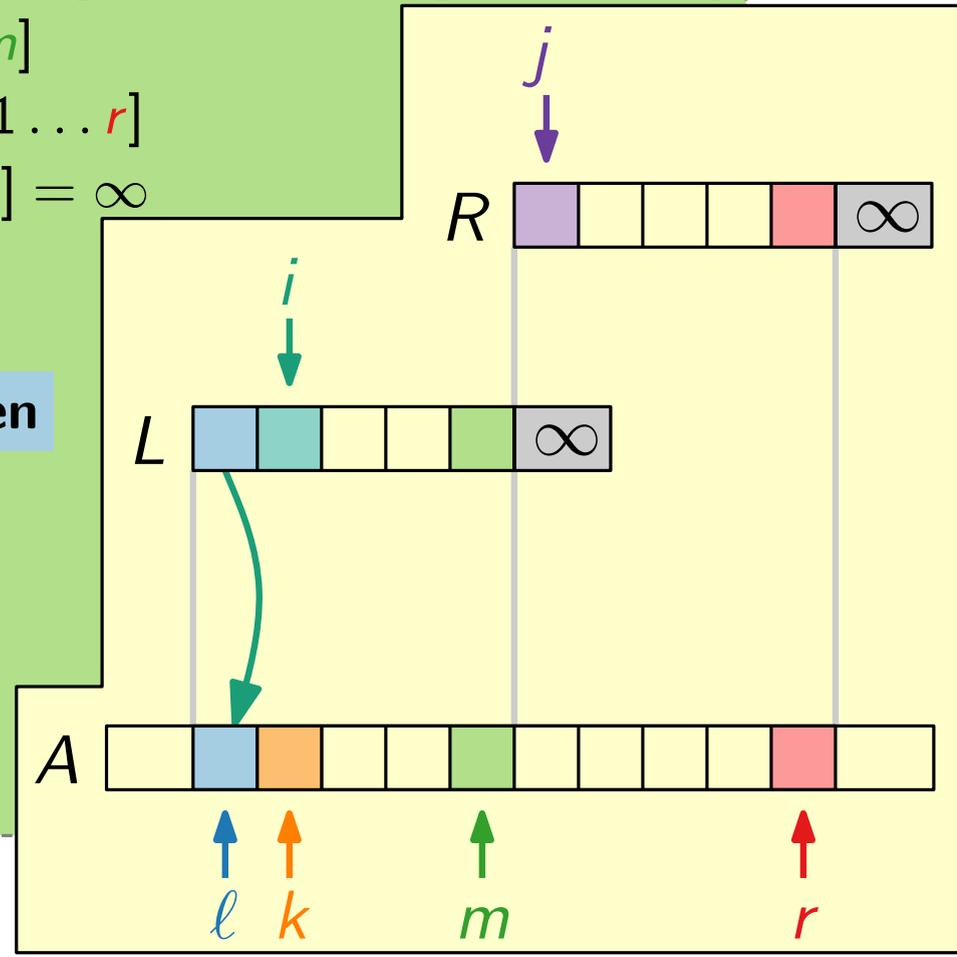
- $A[\ell \dots k - 1]$  enthält die  $k - \ell$  kleinsten Elemente von  $L \cup R$  **sortiert**.
- $L[i]$  und  $R[j]$  sind die kleinsten Elemente in  $L$  bzw.  $R$ , die noch nicht in  $A$  kopiert wurden.

## 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)  $R[j] < L[i]$ .

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.

# 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.

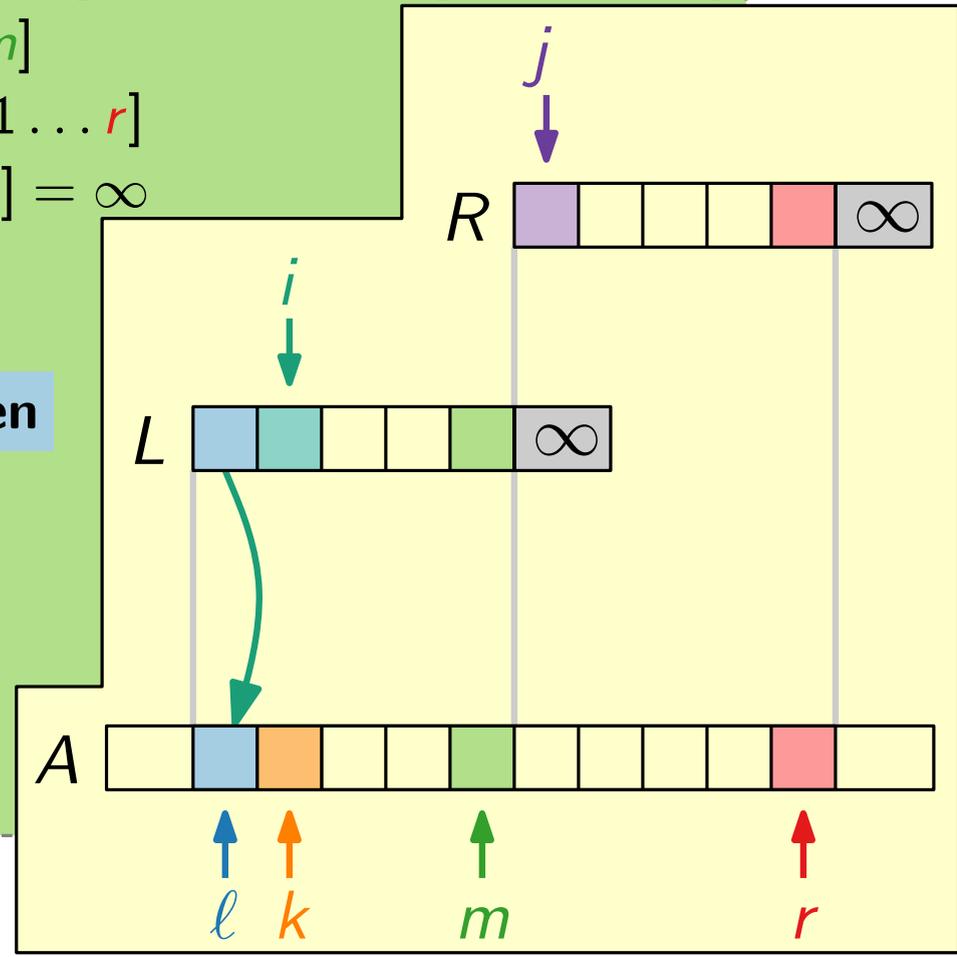
## 1. Initialisierung ✓

## 2. Aufrechterhaltung

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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  $L$ .

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.

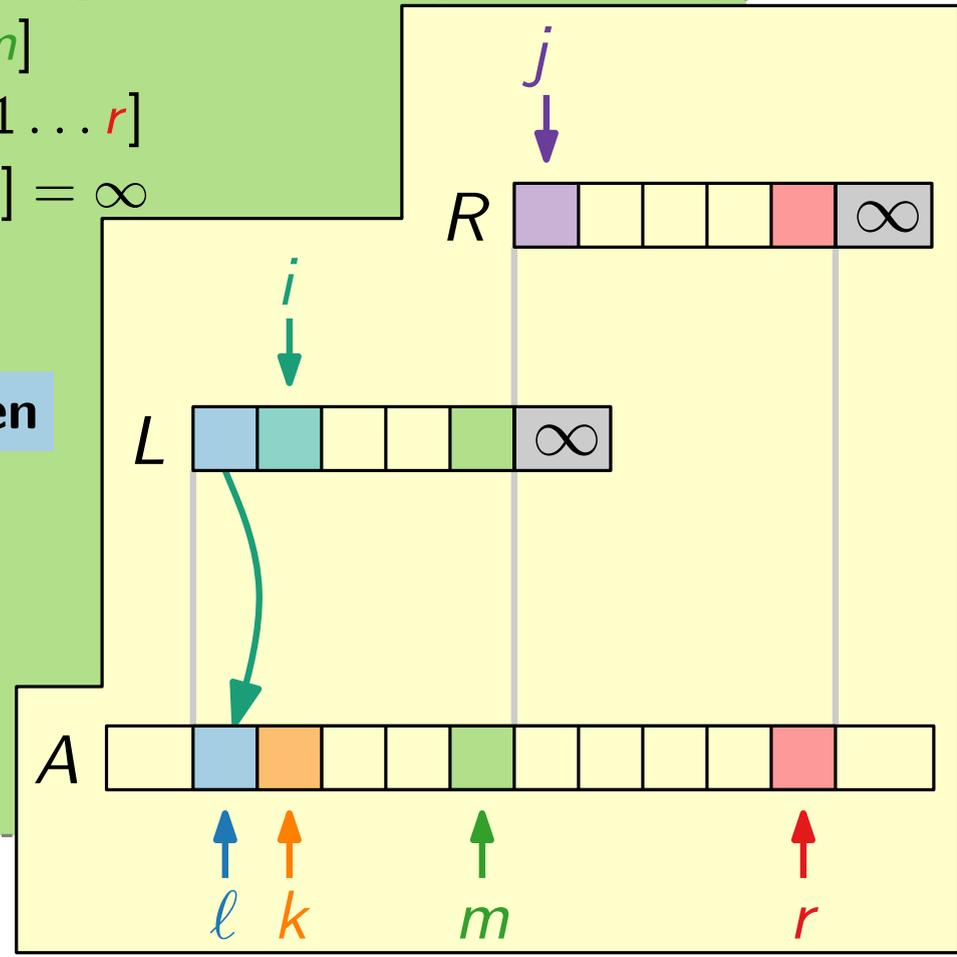
## 1. Initialisierung ✓

## 2. Aufrechterhaltung

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
   $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
   $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
   $L[1 \dots n_1] = A[\ell \dots m]$ 
   $R[1 \dots n_2] = A[m + 1 \dots r]$ 
   $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
   $i = j = 1$ 
  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  $i \Rightarrow$   
 erhöhe  $k \Rightarrow$

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

$\Rightarrow$  Invariante!

# Korrektheit von MERGE

... nach Schema „F“!

## 0. Schleifeninvariante

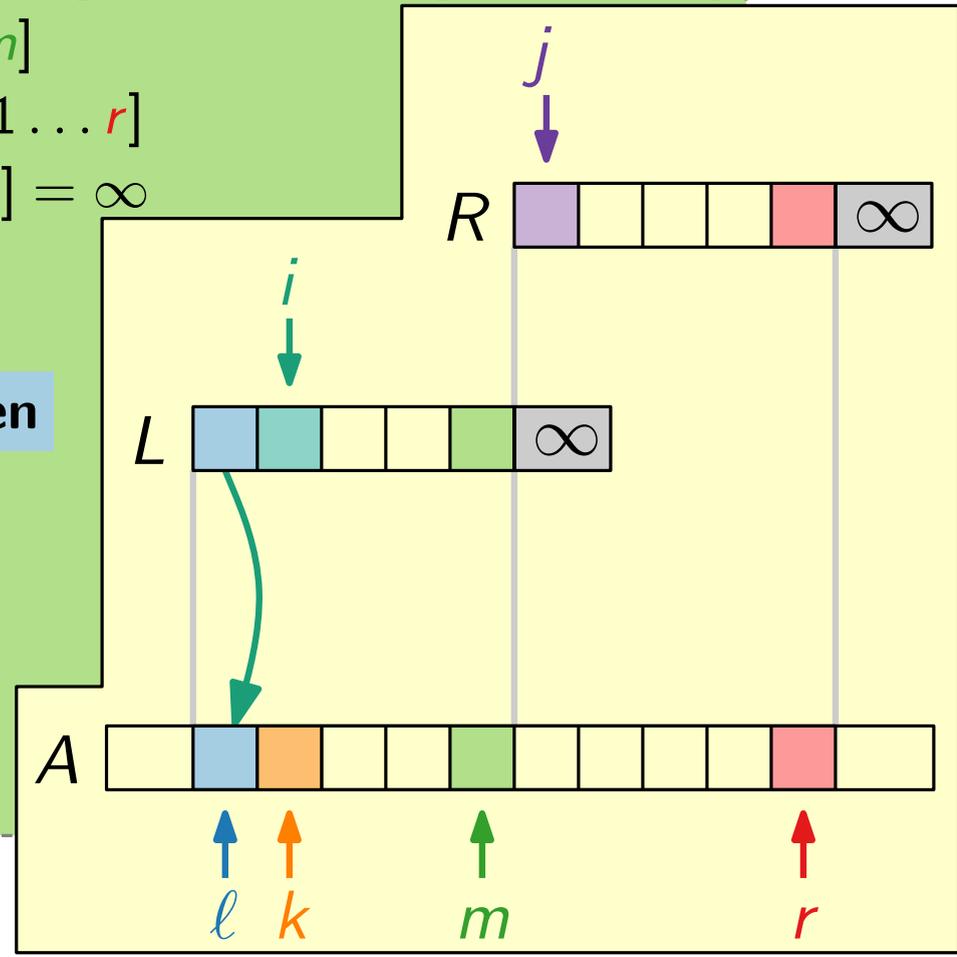
- $A[\ell \dots k - 1]$  enthält die  $k - \ell$  kleinsten Elemente von  $L \cup R$  **sortiert**.
- $L[i]$  und  $R[j]$  sind die kleinsten Elemente in  $L$  bzw.  $R$ , die noch nicht in  $A$  kopiert wurden.

## 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)  $L[i] \leq R[j]$ , ✓ (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  $i \Rightarrow$   
 erhöhe  $k \Rightarrow$

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

⇒ 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

- 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  $i \Rightarrow$   
erhöhe  $k \Rightarrow$

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

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

$\Rightarrow$  Invariante!

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

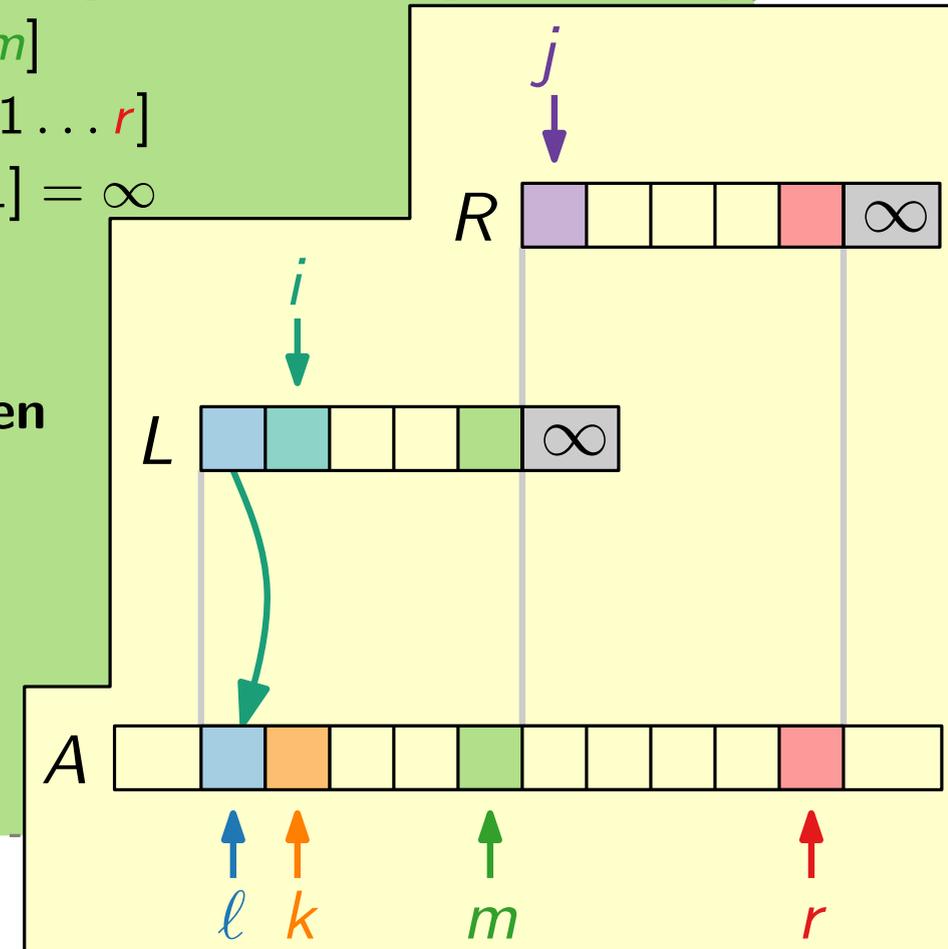
```
     $A[k] = L[i]$ 
```

```
     $i = i + 1$ 
```

```
  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

- 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.

dank Invariante

- $L[i + 1]$  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.

⇒ Invariante!

Fall (b) symmetrisch.

```

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

dank Invariante

erhöhe  $i \Rightarrow$   
erhöhe  $k \Rightarrow$

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

$\Rightarrow$  Invariante!

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

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

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

```
     $A[k] = L[i]$ 
```

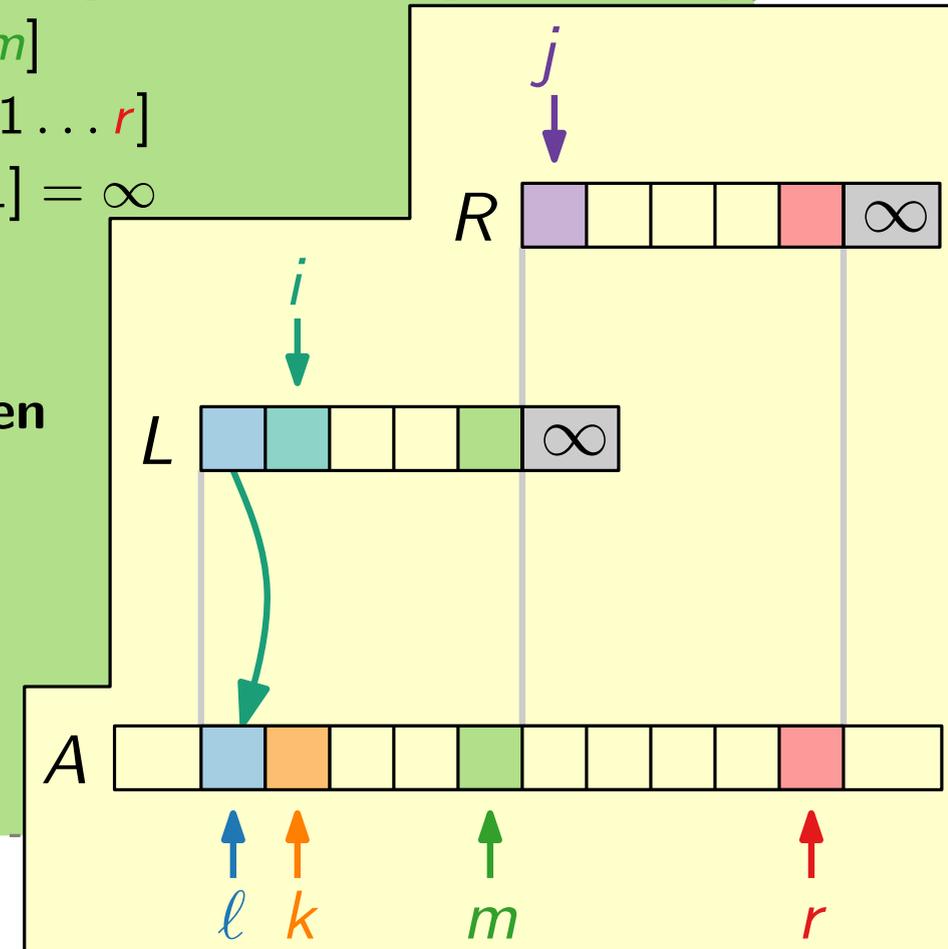
```
     $i = i + 1$ 
```

```
  else
```

```
     $A[k] = R[j]$ 
```

```
     $j = j + 1$ 
```

Fall (b) symmetrisch.



# 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.

Fall (b) symmetrisch.

## 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!

```
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
```

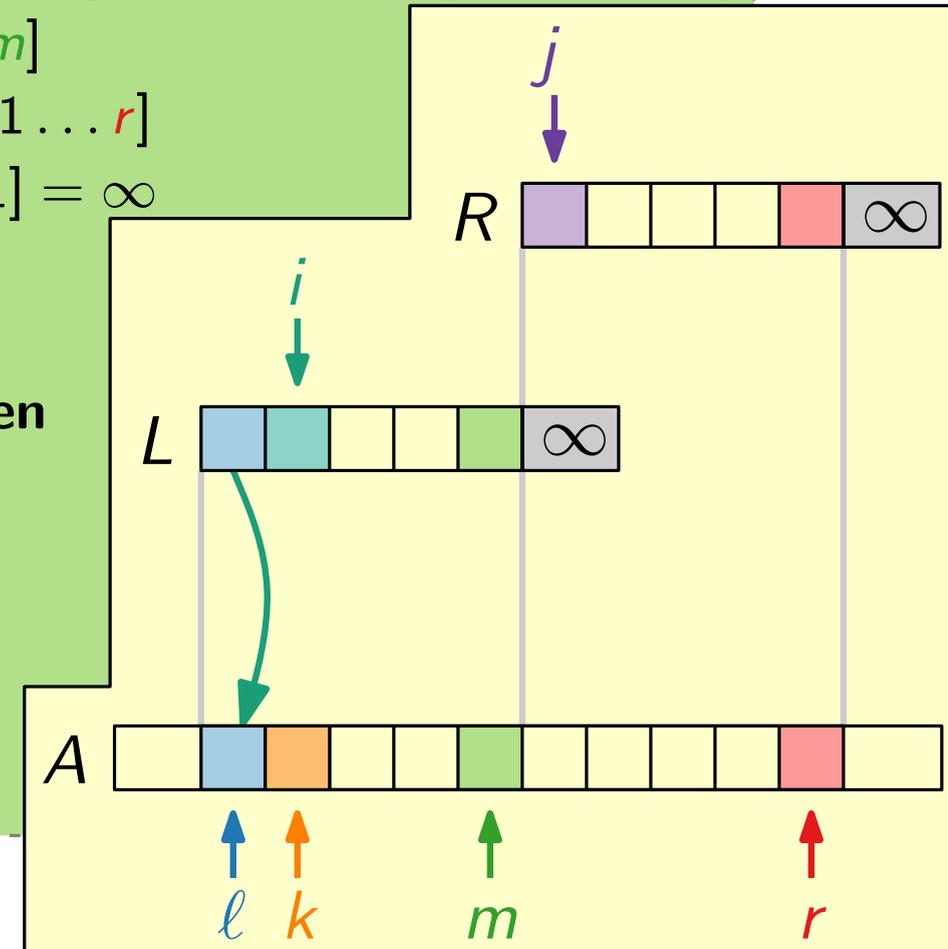
```
      A[k] = L[i]
```

```
      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  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

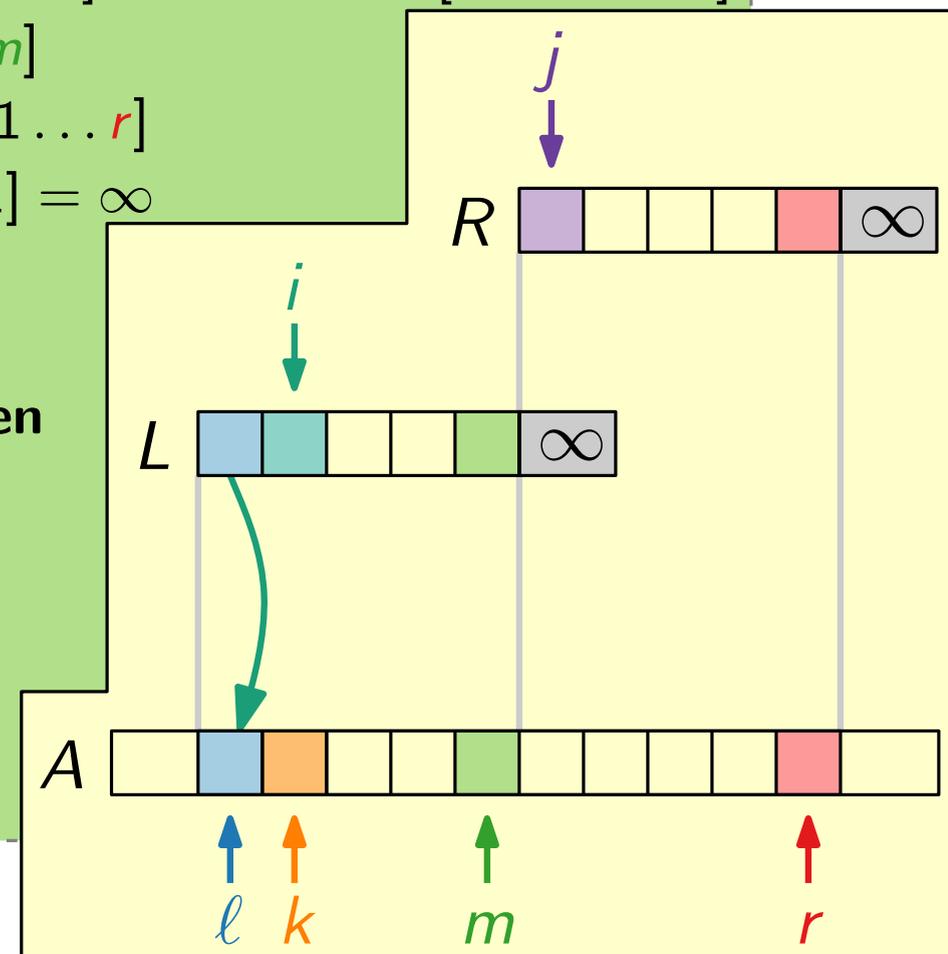
$A[k] = L[i]$

$i = i + 1$

  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  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

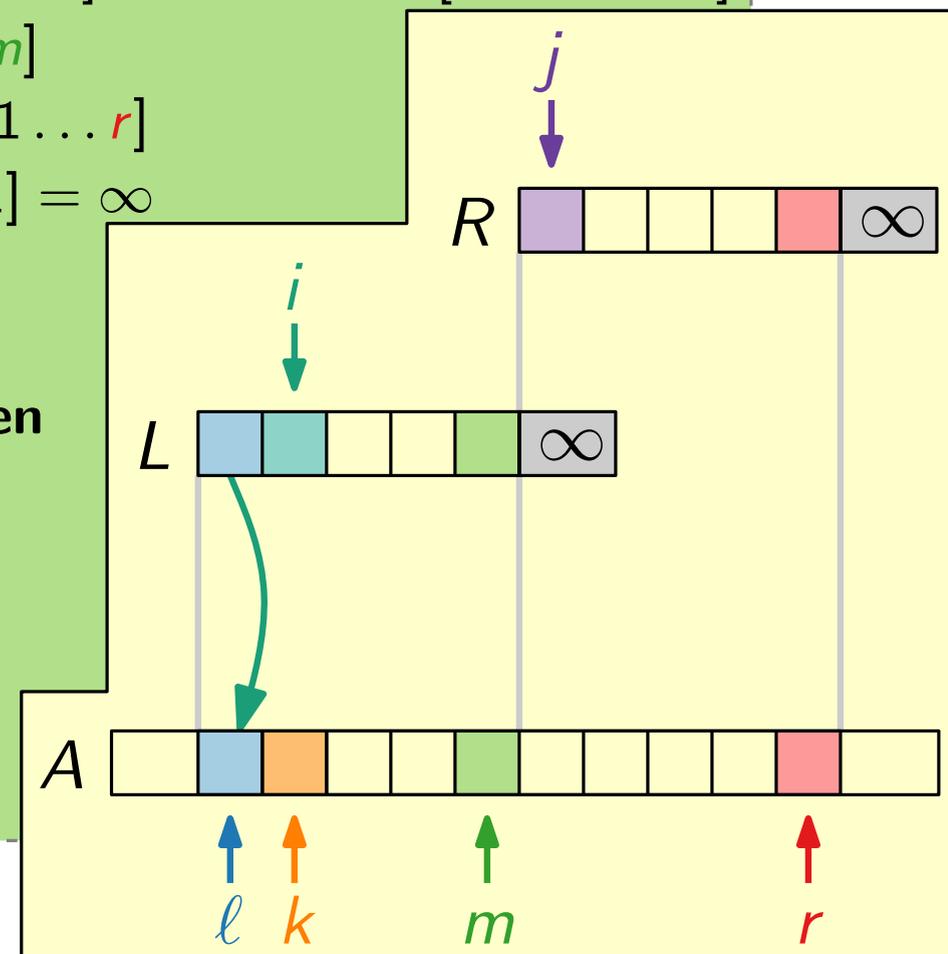
$A[k] = L[i]$

$i = i + 1$

  else

$A[k] = R[j]$

$j = j + 1$



# Korrektheit von MERGE

... nach Schema „F“!

## 0. Schleifeninvariante

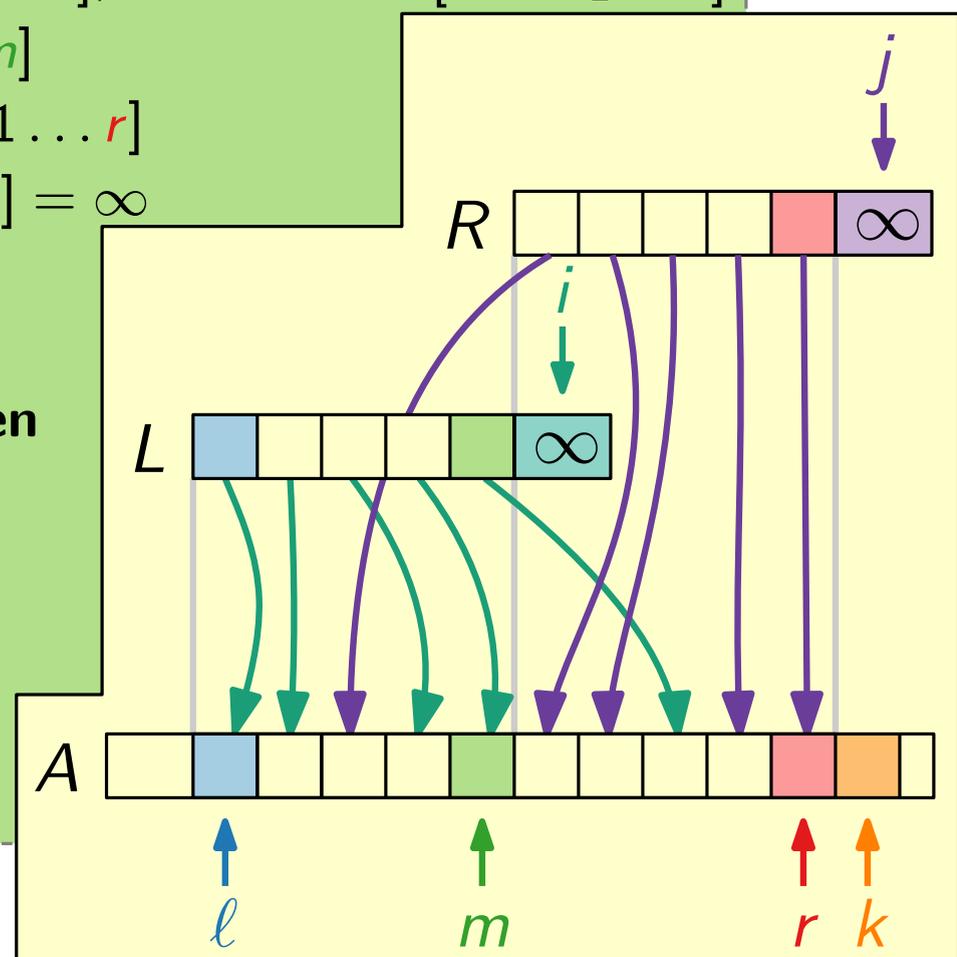
- $A[\ell \dots k - 1]$  enthält die  $k - \ell$  kleinsten Elemente von  $L \cup R$  **sortiert**.
- $L[i]$  und  $R[j]$  sind die kleinsten Elemente in  $L$  bzw.  $R$ , die noch nicht in  $A$  kopiert wurden.

1. Initialisierung ✓
2. Aufrechterhaltung ✓
3. Terminierung

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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  $k = r + 1$ .

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )

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

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

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

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

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

$i = j = 1$

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

  if  $L[i] \leq R[j]$  then

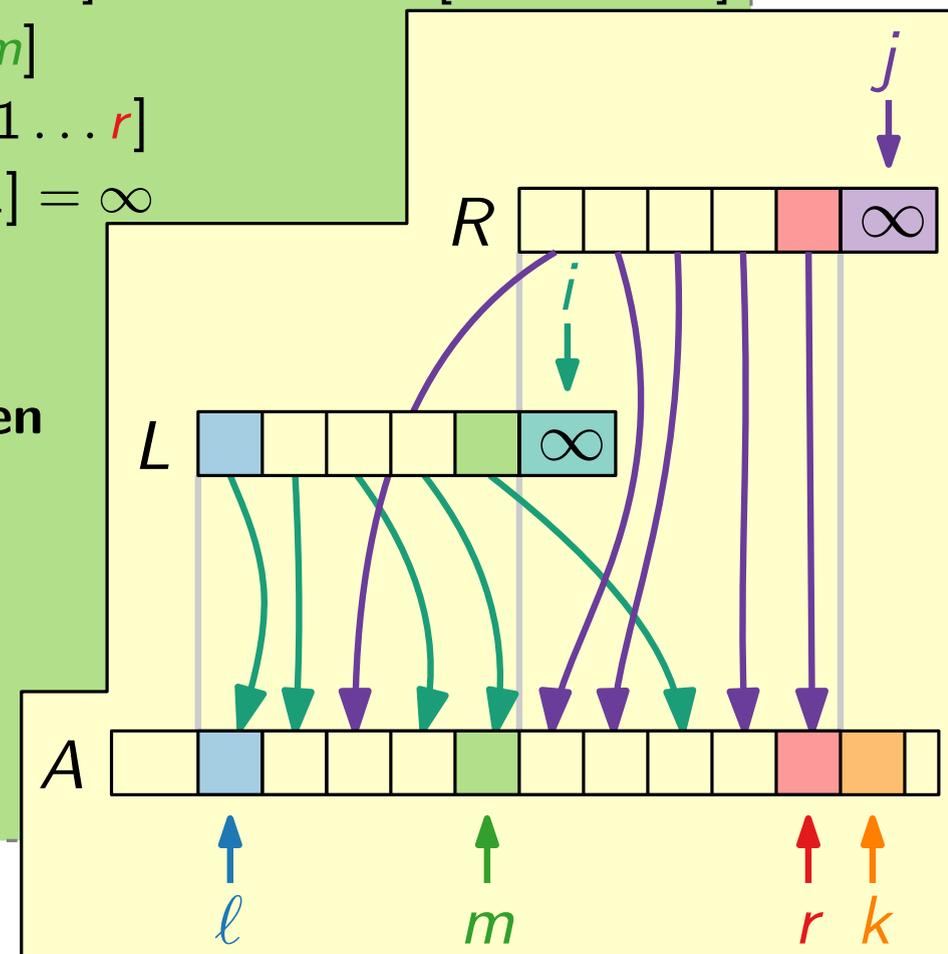
$A[k] = L[i]$

$i = i + 1$

  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.

```

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

```

The diagram illustrates the merge process. It shows three arrays: L, R, and A. L and R are temporary arrays containing elements from A. A is the target array being filled. Arrows show elements being copied from L and R into A. The diagram also shows the indices i, j, and k moving through the arrays.

# 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 ✓

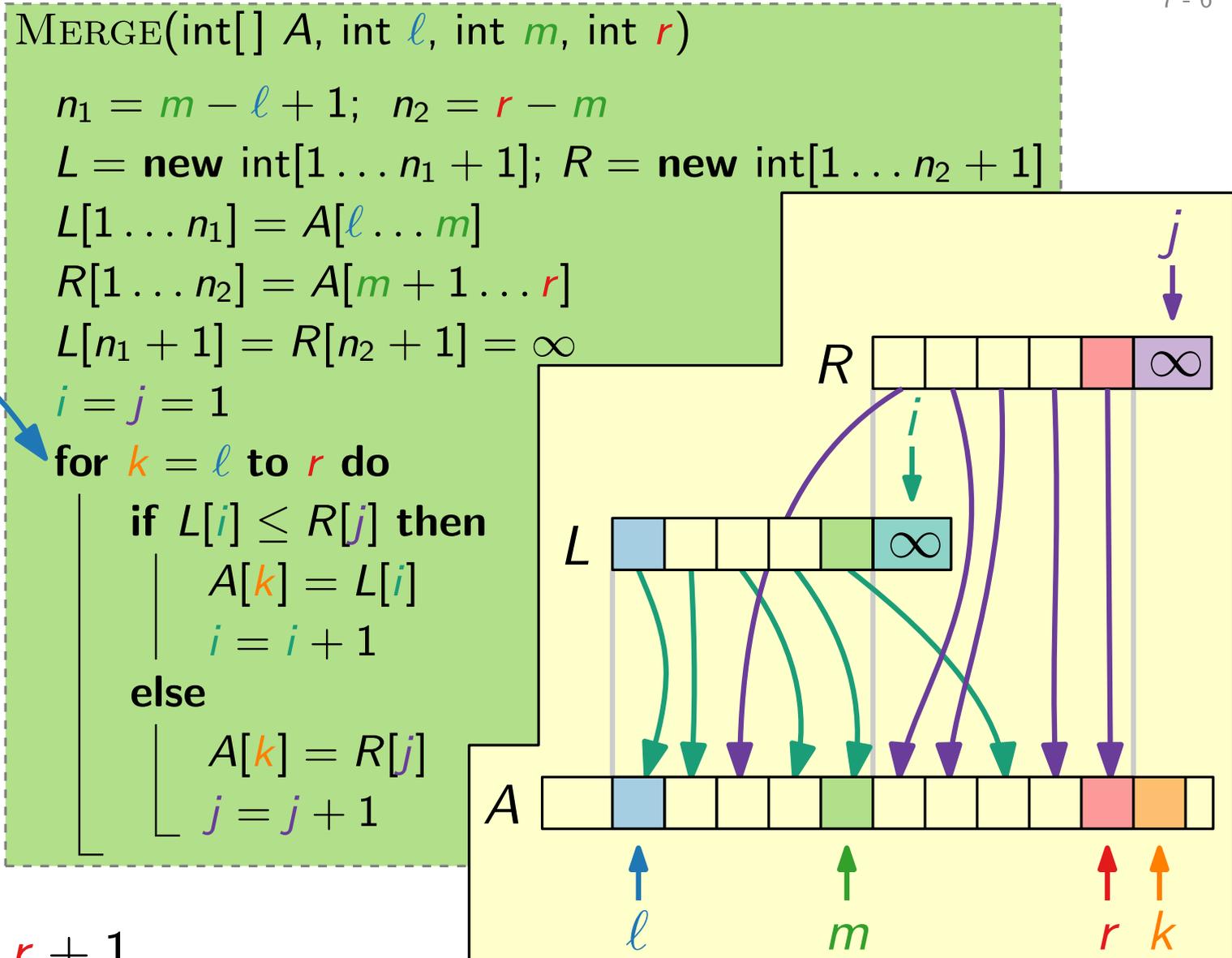
## 2. Aufrechterhaltung ✓

## 3. Terminierung

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

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

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



# 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  $k = r + 1$ .

⇒  $A[\ell \dots k - 1] = A[\ell \dots r]$  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  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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  $k = r + 1$ .

⇒  $A[\ell \dots k - 1] = A[\ell \dots r]$  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  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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  $k = r + 1$ .

⇒  $A[\ell \dots k - 1] = A[\ell \dots r]$  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  $\ell$ , int  $m$ , int  $r$ )
 $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
 $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
 $L[1 \dots n_1] = A[\ell \dots m]$ 
 $R[1 \dots n_2] = A[m + 1 \dots r]$ 
 $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
 $i = j = 1$ 
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  $k = r + 1$ .

⇒  $A[\ell \dots k - 1] = A[\ell \dots r]$  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
  
```

The diagram illustrates the merge process. It shows three arrays: L, R, and A. L and R are temporary arrays containing elements from A. A is the target array being filled. Arrows show elements being copied from L and R into A. The diagram also shows the indices i, j, and k moving through the arrays.

+2 Stopper

# 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  $k = r + 1$ .

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

```

MERGE(int[] A, int l, int m, int r)
  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
  
```

+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 ✓

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

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

```

MERGE(int[] A, int l, int m, int r)
  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

```

+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  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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 ✓

Also ist MERGE korrekt!

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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 ✓

Also ist MERGE korrekt!

**Laufzeit?**

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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 ✓

Also ist MERGE korrekt!

**Laufzeit?**

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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 ✓

Also ist MERGE korrekt!

### Laufzeit?

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
   $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
   $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
   $L[1 \dots n_1] = A[\ell \dots m]$ 
   $R[1 \dots n_2] = A[m + 1 \dots r]$ 
   $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
   $i = j = 1$ 
  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 ✓

Also ist MERGE korrekt!

```

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

```

□

**Laufzeit?**

MERGE macht genau

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!

```

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

```

□

**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  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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 ✓

Also ist MERGE korrekt!



**Laufzeit?**

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

Und MERGESORT?

Korrekt?

```

MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
   $n_1 = m - \ell + 1$ ;  $n_2 = r - m$ 
   $L = \text{new int}[1 \dots n_1 + 1]$ ;  $R = \text{new int}[1 \dots n_2 + 1]$ 
   $L[1 \dots n_1] = A[\ell \dots m]$ 
   $R[1 \dots n_2] = A[m + 1 \dots r]$ 
   $L[n_1 + 1] = R[n_2 + 1] = \infty$ 
   $i = j = 1$ 
  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 ✓

Also ist MERGE korrekt! □

**Laufzeit?**

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

Und MERGESORT?

Korrekt? Effizient?

```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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

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

```
 $i = j = 1$ 
```

```
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 MERGESORT

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

```
if  $\ell < r$  then
```

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

```
    MERGESORT(A,  $\ell$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```

# 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,  $\ell$ ,  $m$ ) }  
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche  
  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,  $\ell$ ,  $m$ ) }  
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche  
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } 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,  $\ell$ ,  $m$ ) }
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } 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,  $\ell$ ,  $m$ ) }
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
  MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere

```

Korrekt? Welche Beweistechnik? Hm, MERGESORT ist **rekursiv**...

**Vollständige Induktion** über  $n = r - \ell + 1$  ( $= A[\ell \dots r].length$ ):

# 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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**

# 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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  $\ell = r$ .

# 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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.

D.h. nichts passiert.

# 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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.

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,  $\ell$ ,  $m$ ) } herrsche
```

```
    MERGESORT(A,  $m + 1$ ,  $r$ ) }
```

```
    MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } kombiniere
```

$n > 1$ : **Induktionsschritt**

# 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,  $\ell$ ,  $m$ )             }
      MERGESORT(A,  $m + 1$ ,  $r$ )          } herrsche
    ] MERGE(A,  $\ell$ ,  $m$ ,  $r$ )           } 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    MERGE(A,  $\ell$ ,  $m$ ,  $r$ ) } 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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.

# 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,  $\ell$ ,  $m$ ) }
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
  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  $\ell \leq m < r$ .

# 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,  $\ell$ ,  $m$ ) }
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
  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  $A[\ell \dots r]$ .

# 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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  $A[\ell \dots r]$ .

$\Rightarrow$   
I.A.

# 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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  $A[\ell \dots r]$ .

$\Rightarrow$  **MERGESORT(A,  $\ell$ ,  $m$ )** ist korrekt und  
I.A.

# 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,  $\ell$ ,  $m$ ) }
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
  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  $A[\ell \dots r]$ .

$\Rightarrow$  MERGESORT(A,  $\ell$ ,  $m$ ) ist korrekt und

**I.A.**

MERGESORT(A,  $m + 1$ ,  $r$ ) ist 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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  $A[\ell \dots r]$ .

$\Rightarrow$  MERGESORT(A,  $\ell$ ,  $m$ ) ist korrekt und

**I.A.** 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,  $\ell$ ,  $m$ ) }
    MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
    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  $A[\ell \dots r]$ .

$\Rightarrow$  MERGESORT(A,  $\ell$ ,  $m$ ) ist korrekt und

**I.A.**

MERGESORT(A,  $m + 1$ ,  $r$ ) ist korrekt.

Schon bewiesen: MERGE ist 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,  $\ell$ ,  $m$ ) }
  MERGESORT(A,  $m + 1$ ,  $r$ ) } herrsche
  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  $A[\ell \dots r]$ .

$\Rightarrow$   $\left. \begin{array}{l} \text{I.A. } \text{MERGESORT}(A, \ell, m) \text{ ist korrekt und} \\ \text{MERGESORT}(A, m + 1, r) \text{ ist korrekt.} \end{array} \right\} \text{MERGESORT}(A, \ell, r) \text{ ist korrekt. } \square$

Schon bewiesen: **MERGE** 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?