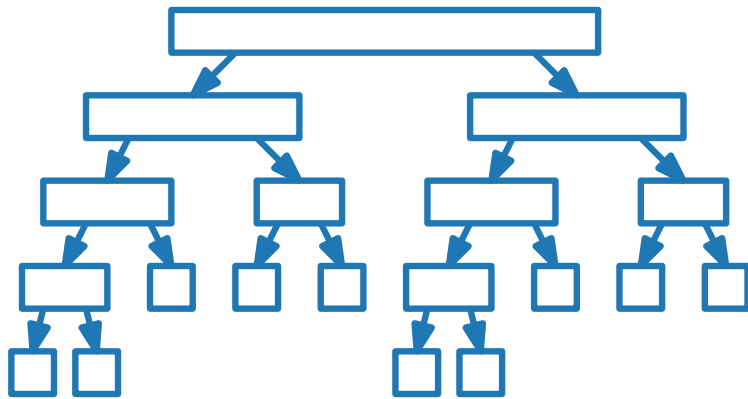
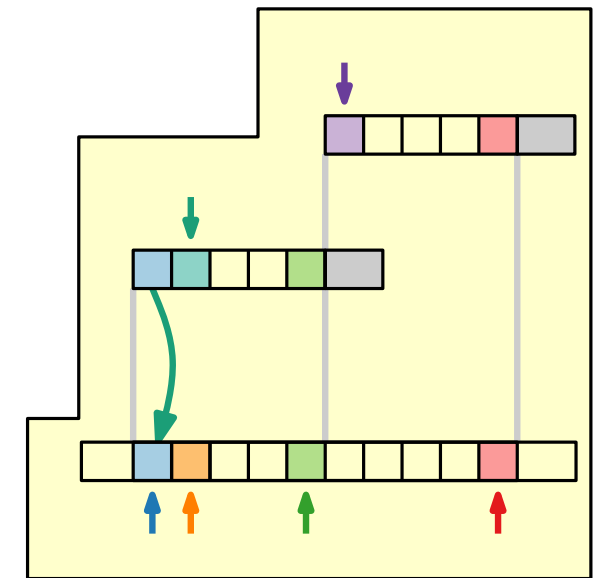


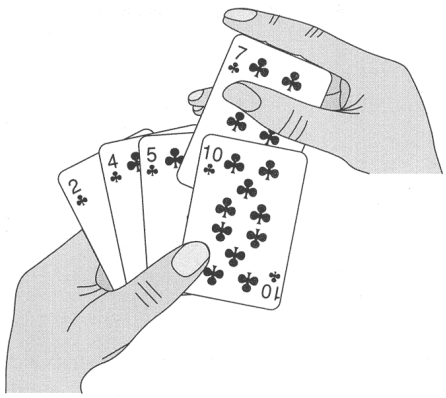
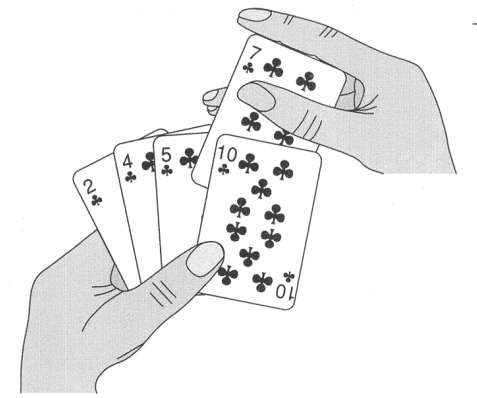


Algorithmen und Datenstrukturen

Vorlesung 2: Teile und 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.

Aufruf einer Funktion durch sich selbst

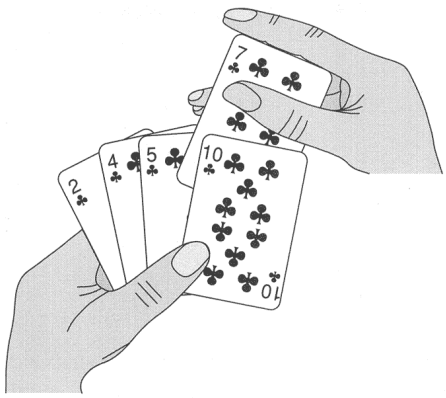
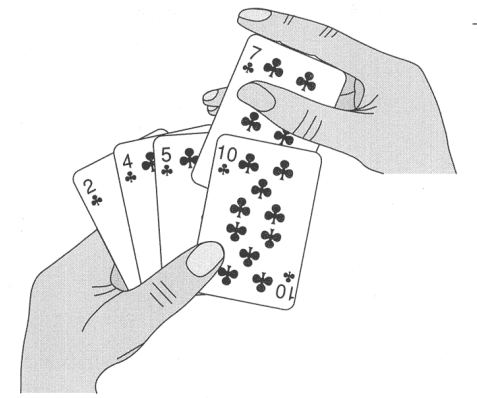
Herrsche ...

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  $\ell = 1$ , int  $r = A.length$ )`

Defaultwerte

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

Allgemein.

Teile ...

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

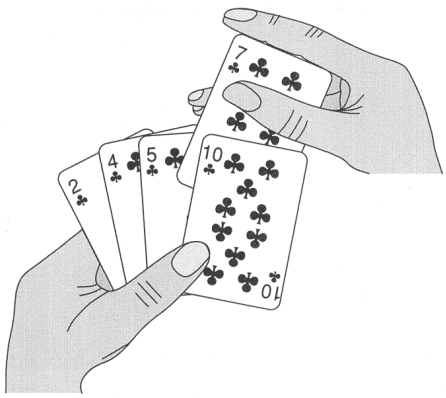
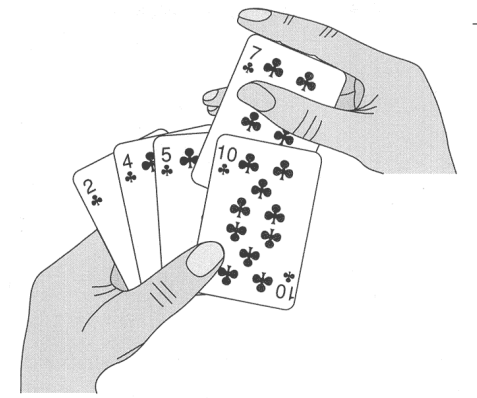
Herrsche ...

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  $\ell = 1$ , 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
  
```

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

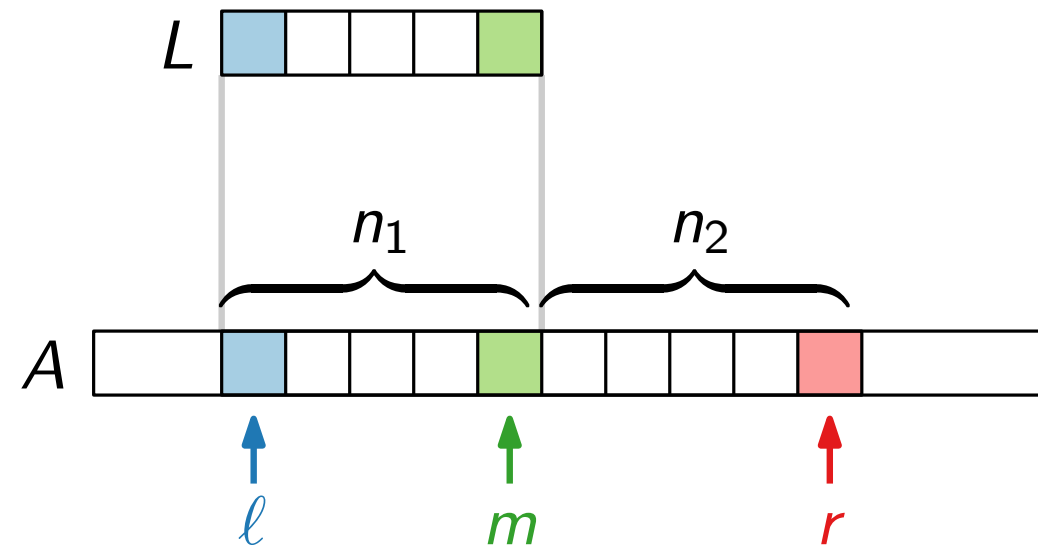
```
MERGE(int[] A, int  $\ell$ , int  $m$ , int  $r$ )
```

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

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

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

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



Kombiniere

MERGE(int[] A, int ℓ , int m , int r)

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

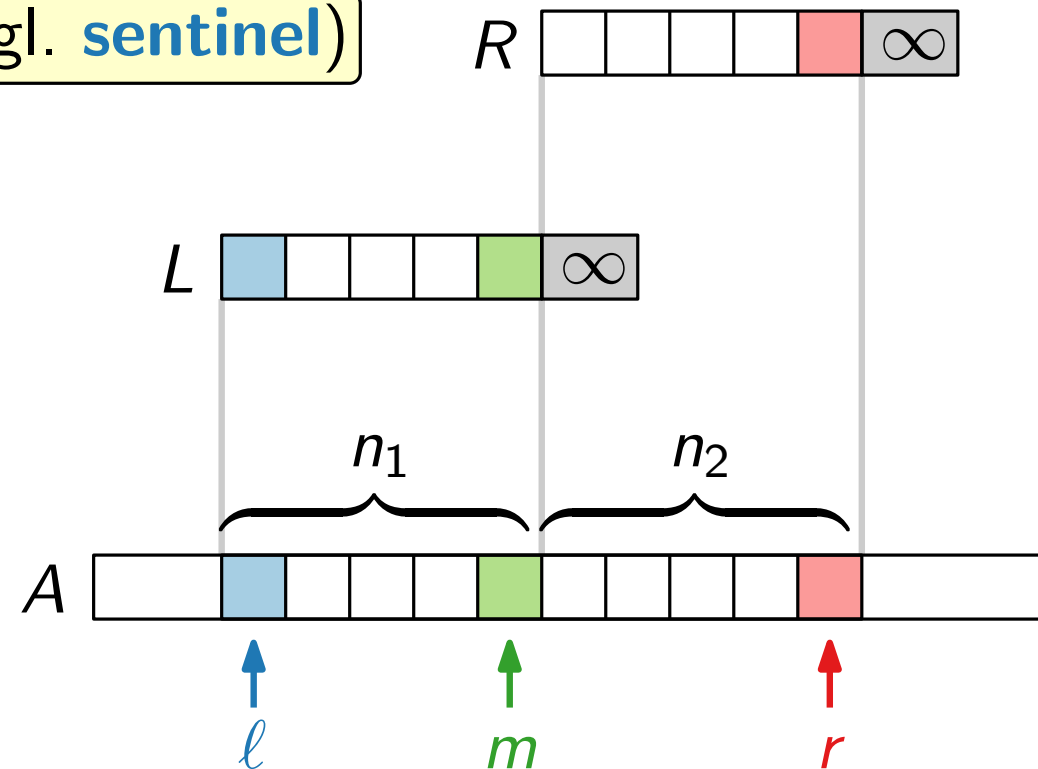
$L = \text{new int}[1 \dots n_1 + 1]$; $R = \text{new int}[1 \dots n_2 + 1]$

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

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

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

Stopper (engl. **sentinel**)



Kombiniere

MERGE(int[] A, int ℓ , int m , int r)

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

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

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

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

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

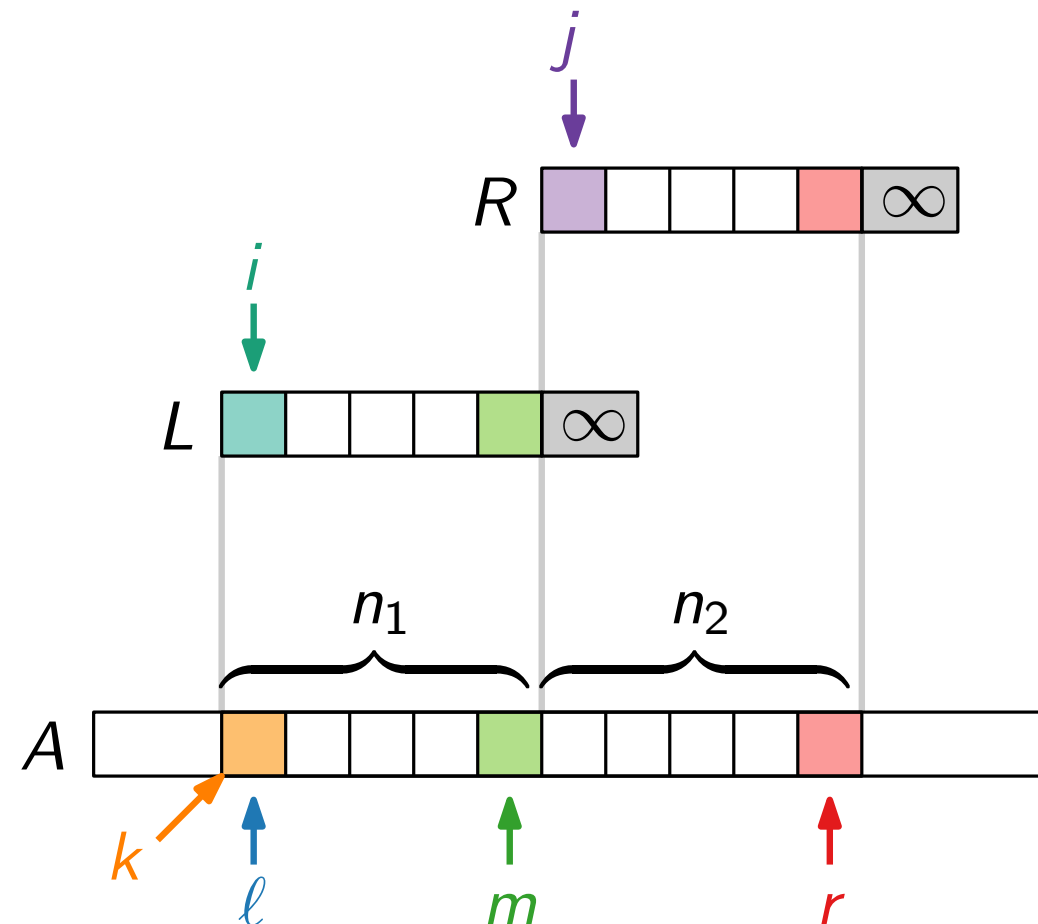
$i = j = 1$

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)

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

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

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

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

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

$i = j = 1$

for $k = \ell$ to r do

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

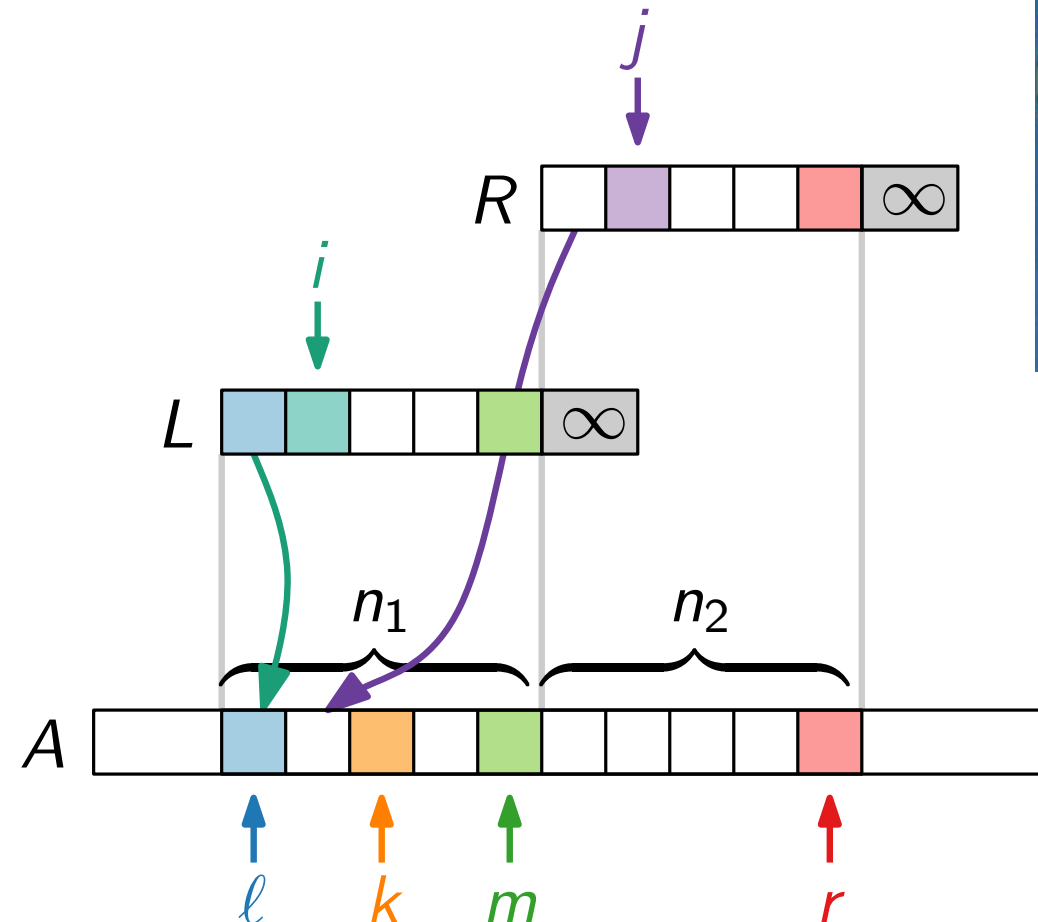
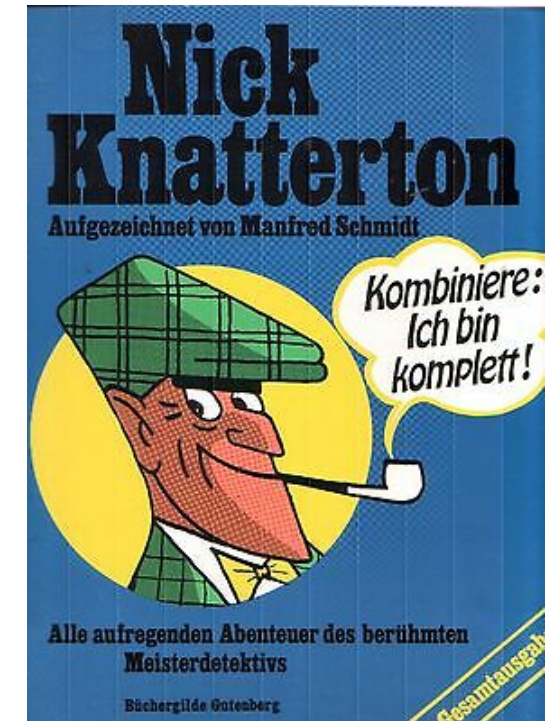
$A[k] = L[i]$

$i = i + 1$

 else

$A[k] = R[j]$

$j = j + 1$



MergeSort – ein Beispiel

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

Demo.

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

MergeSort – ein Beispiel

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

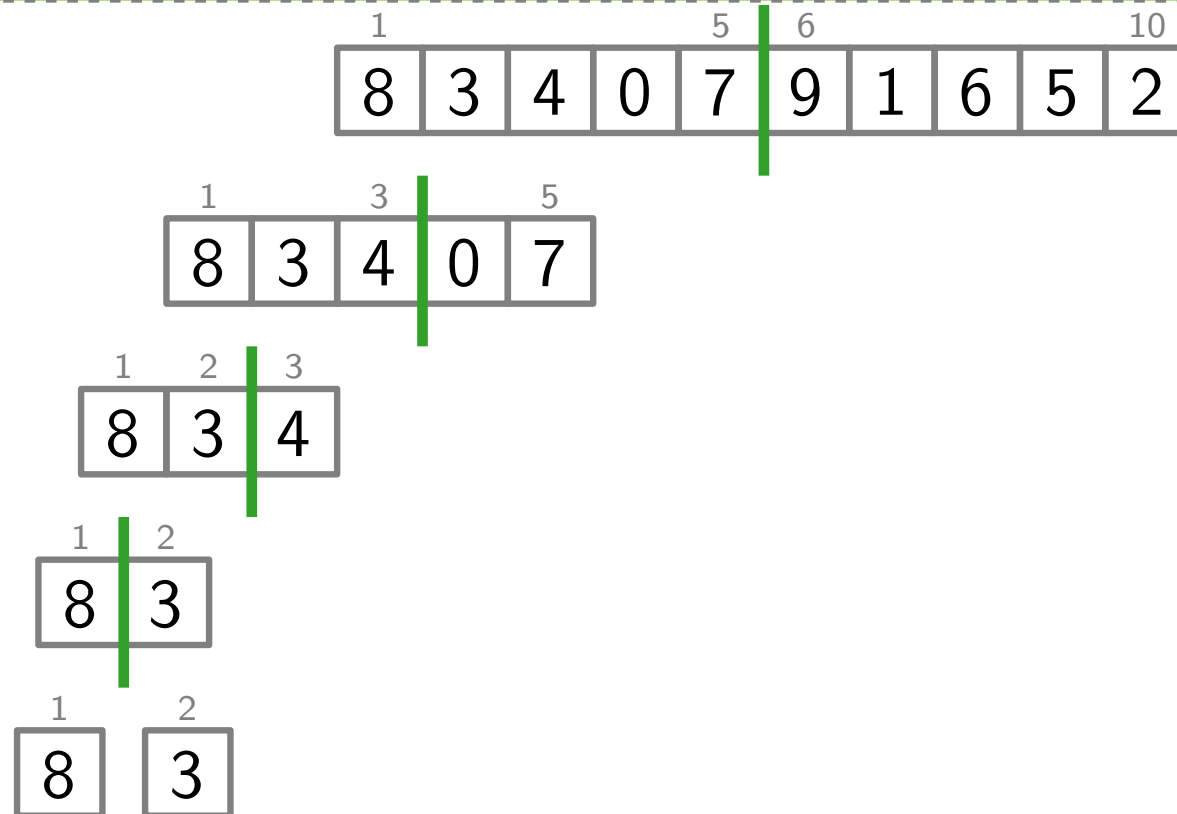
if $\ell < r$ **then**

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

 MERGE_SORT(A, ℓ , m) } **herrsche**

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

 MERGE(A, ℓ , m , r) } **kombiniere**



MergeSort – ein Beispiel

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

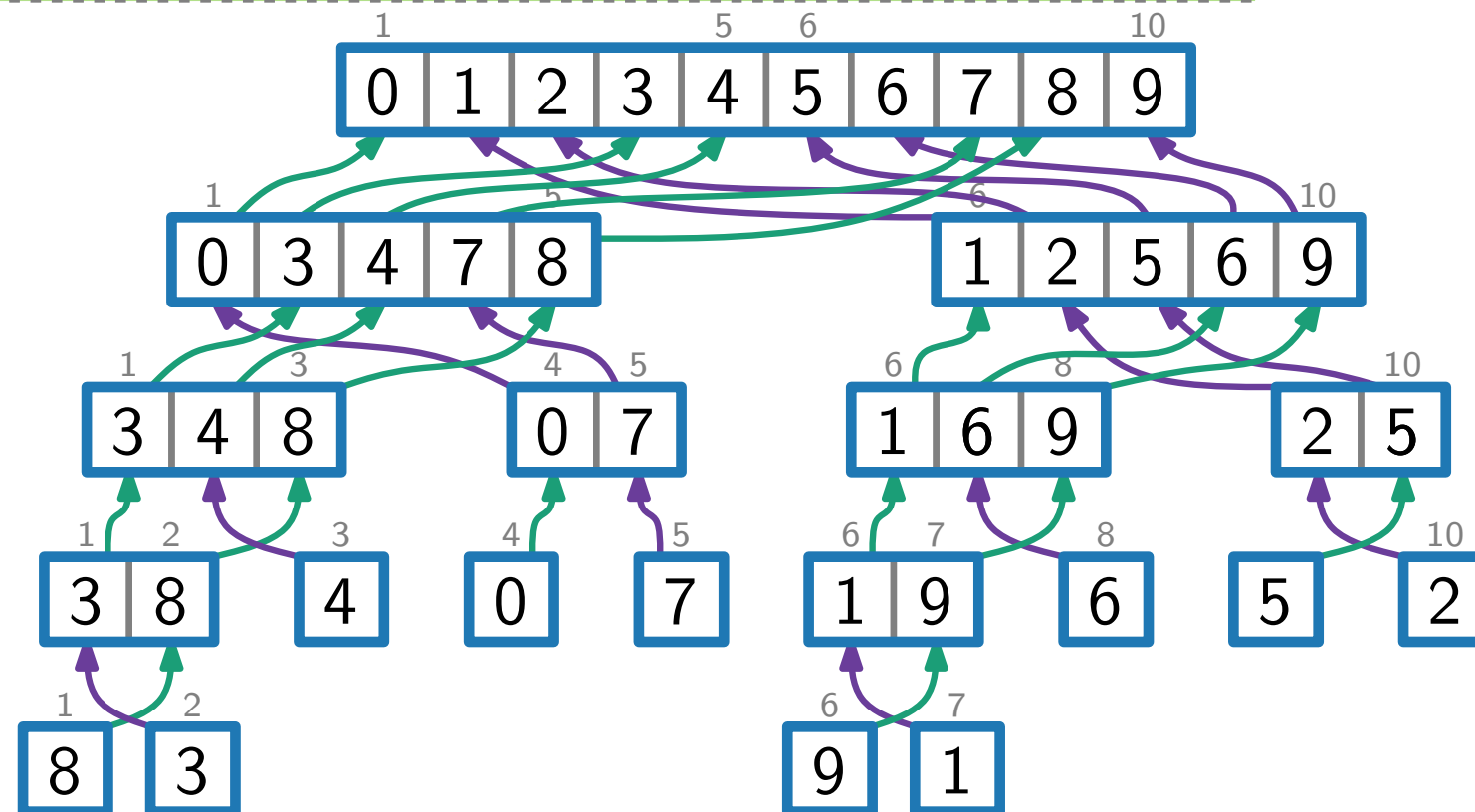
if $\ell < r$ **then**

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

 MERGESORT(A, ℓ , m) } **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
```

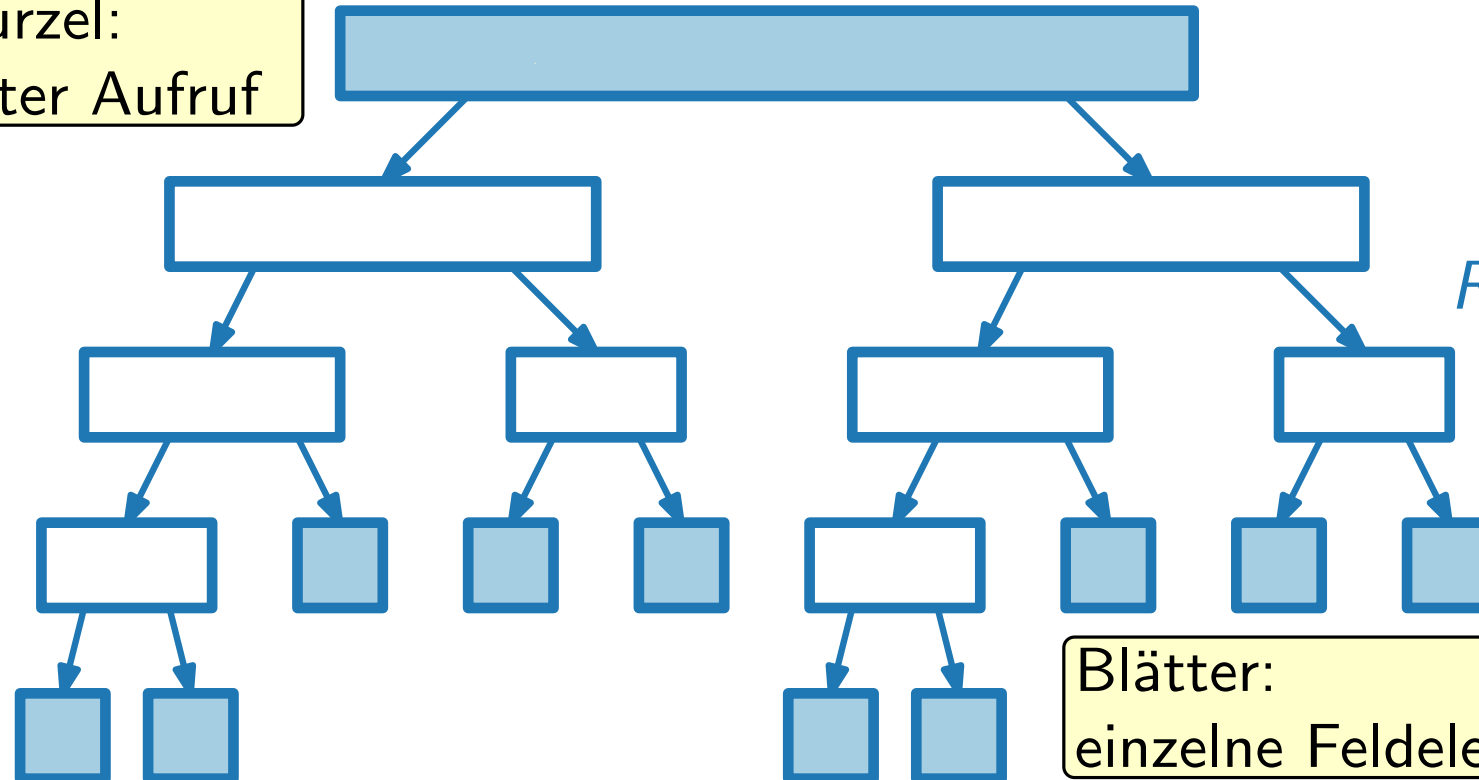
```
     $m = \lfloor (\ell + r) / 2 \rfloor$  } teile
```

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

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

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

Wurzel:
erster Aufruf



Rekursionsbaum von MERGESORT

Baum der
rekursiven
Aufrufe

Blätter:
einzelne Feldelemente

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
```

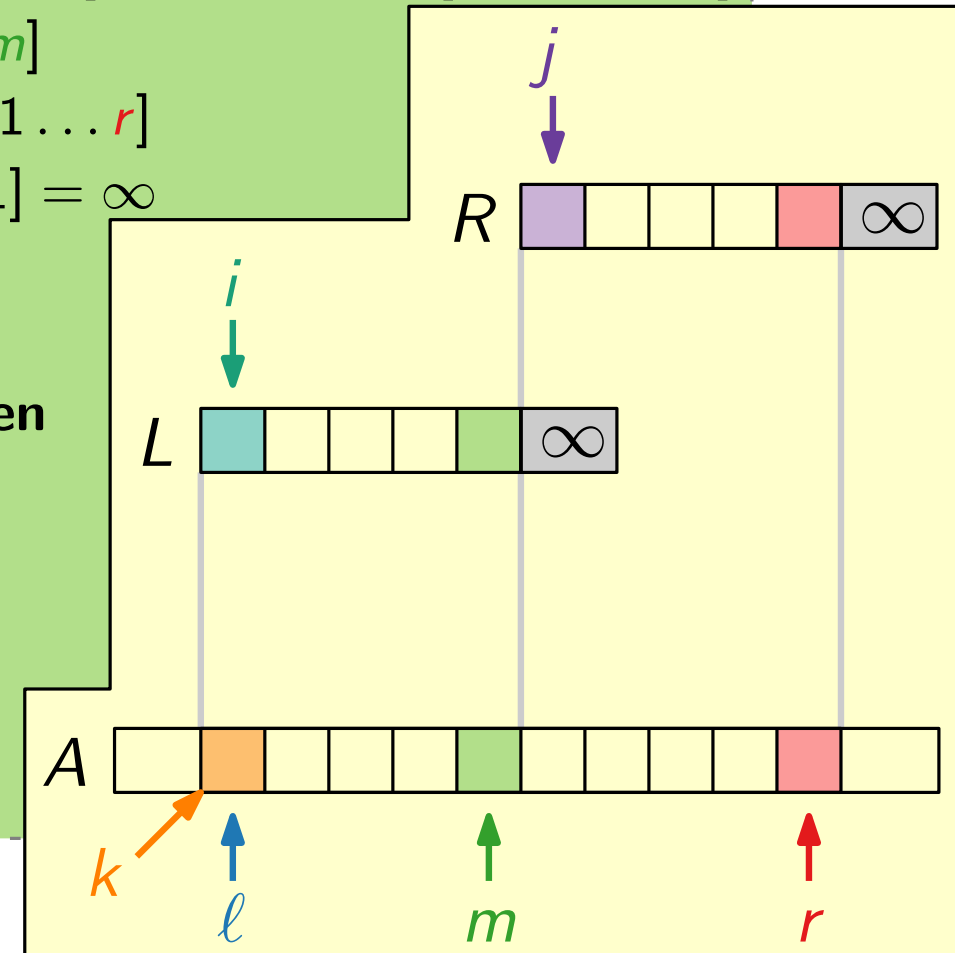
```
     $A[k] = L[i]$ 
```

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

```
  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 ✓

2. Aufrechterhaltung ✓

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

- Nun gilt:

dank Invariante

erhöhe $i \Rightarrow$

erhöhe $k \Rightarrow$

- $A[\ell \dots k]$ 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 .

$A[\ell \dots k - 1]$ 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
```

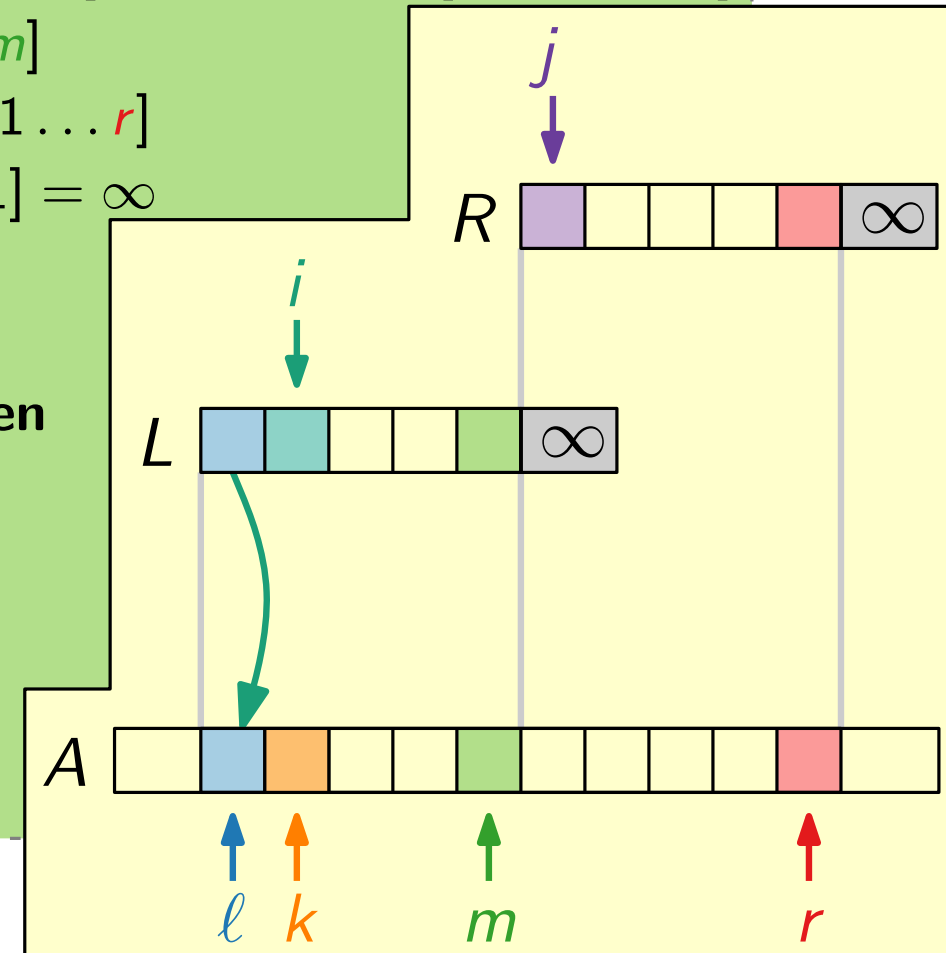
```
     $A[k] = L[i]$ 
```

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

```
  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.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

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

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

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

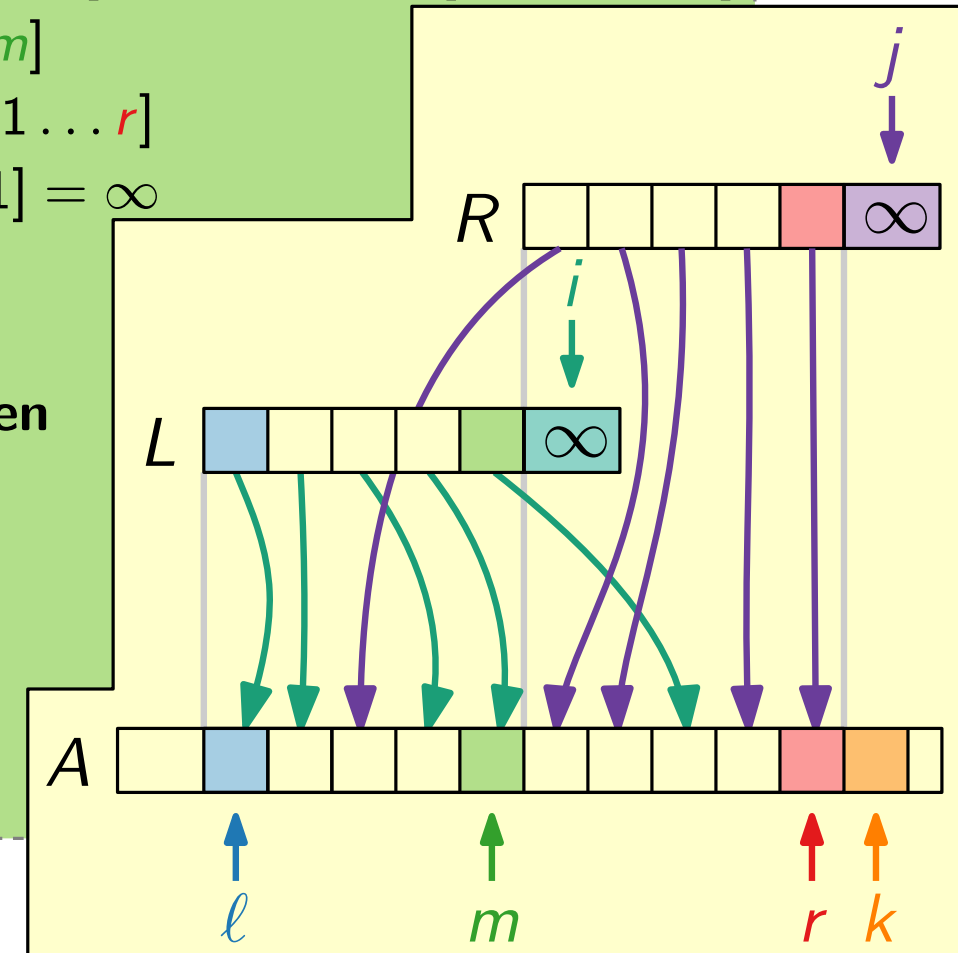
```
     $A[k] = L[i]$ 
```

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

```
  else
```

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

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



+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 ✓

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 $A[\ell \dots r].length = r - \ell + 1$ Vergleiche.

Und MERGESORT? Korrekt? Effizient?

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**

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 $A[\ell \dots r]$.

$\Rightarrow$ $\left. \begin{array}{l} \text{MERGESORT}(A, \ell, m) \text{ ist korrekt und} \\ \text{MERGESORT}(A, m + 1, r) \text{ ist korrekt.} \end{array} \right\} \text{MERGESORT}(A, \ell, r) \text{ ist korrekt.} \quad \square$

Schon bewiesen: **MERGE** ist korrekt.

Übersicht

Techniken für Korrektheitsbeweise

- iterative Algorithmen (à la INSERTIONSORT, FACTORIAL, MERGE)
per **Schleifeninvariante** (Schema „F“)
- rekursive Algorithmen (à la MERGESORT)
per **vollständige Induktion**

Laufzeit?