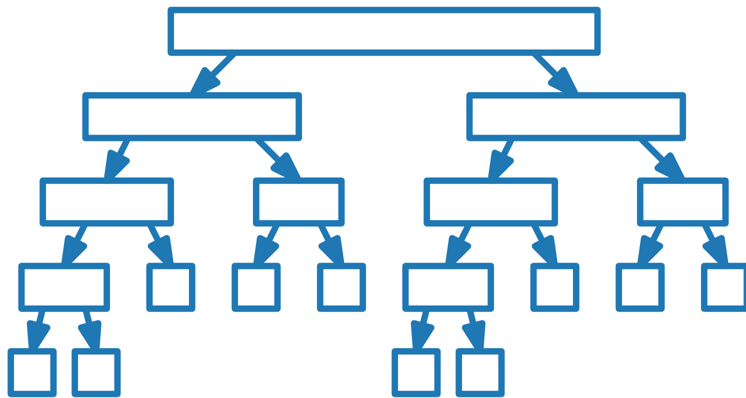
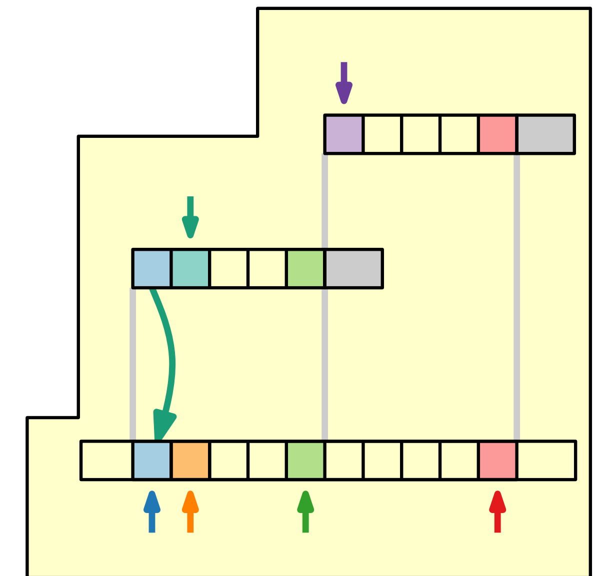
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Algorithmen und Datenstrukturen

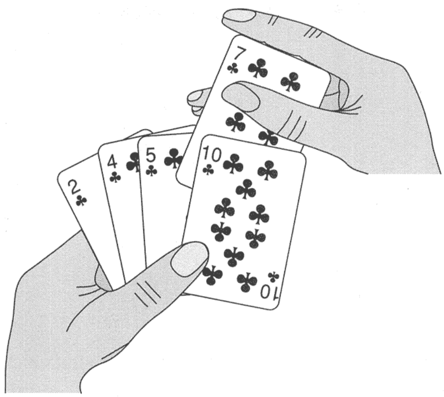
Vorlesung 2:
Teile und Herrsche



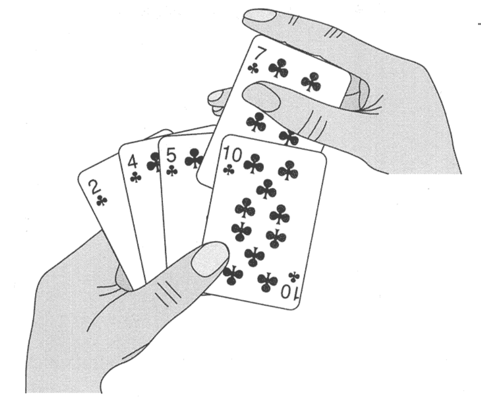


Alexander Wolff

Wintersemester 2024



Teile und Herrsche



Idee.

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

Allgemein.

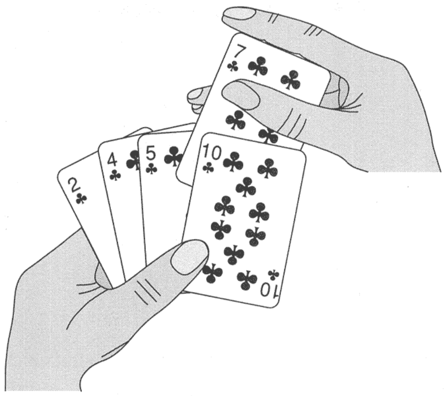
Teile . . . eine Instanz in kleinere Instanzen **desselben** Problems.

Aufruf einer Funktion durch sich selbst

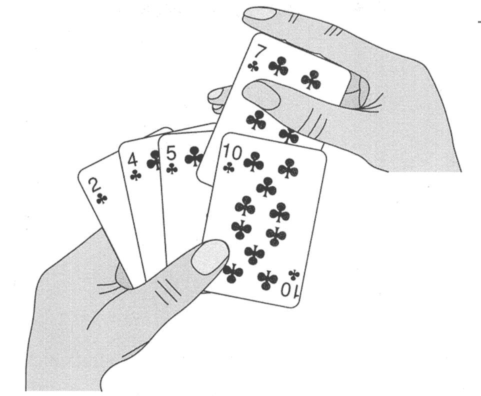
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

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

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



Teile und Herrsche



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

Defaultwerte

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

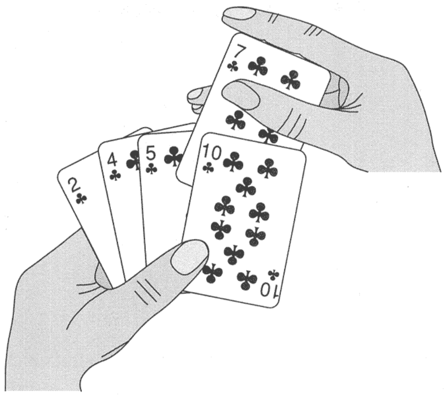
Allgemein.

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

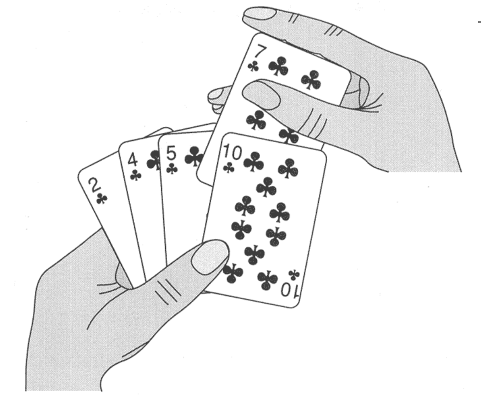
Herrsche . . . durch **rekursives** Lösen von Teilinstanzen –
nur falls diese sehr klein sind, löse sie direkt.

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

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



Teile und Herrsche



```
MergeSort(int[] A, int ℓ = 1, int r = A.length)
if ℓ < r then
    m = ⌊(ℓ + r)/2⌋            } teile
    MergeSort(A, ℓ, m)         } herrsche
    MergeSort(A, m + 1, r)     }
    Merge(A, ℓ, m, r)          } kombiniere
```

TODO!

Allgemein.

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

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

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

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

Kombiniere

MERGE(int[] A, int ℓ, int m, int r)

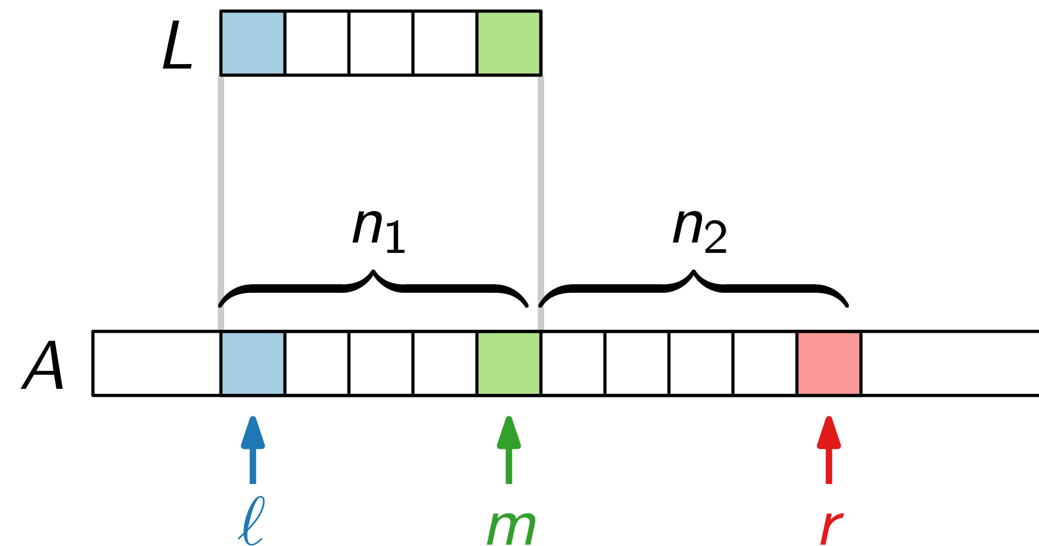
$n_1 = m - \ell + 1$; $n_2 = r - m$

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

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

for $i = 1$ **to** n_1 **do**

$L[i] = A[(\ell - 1) + i]$



Kombiniere

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

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

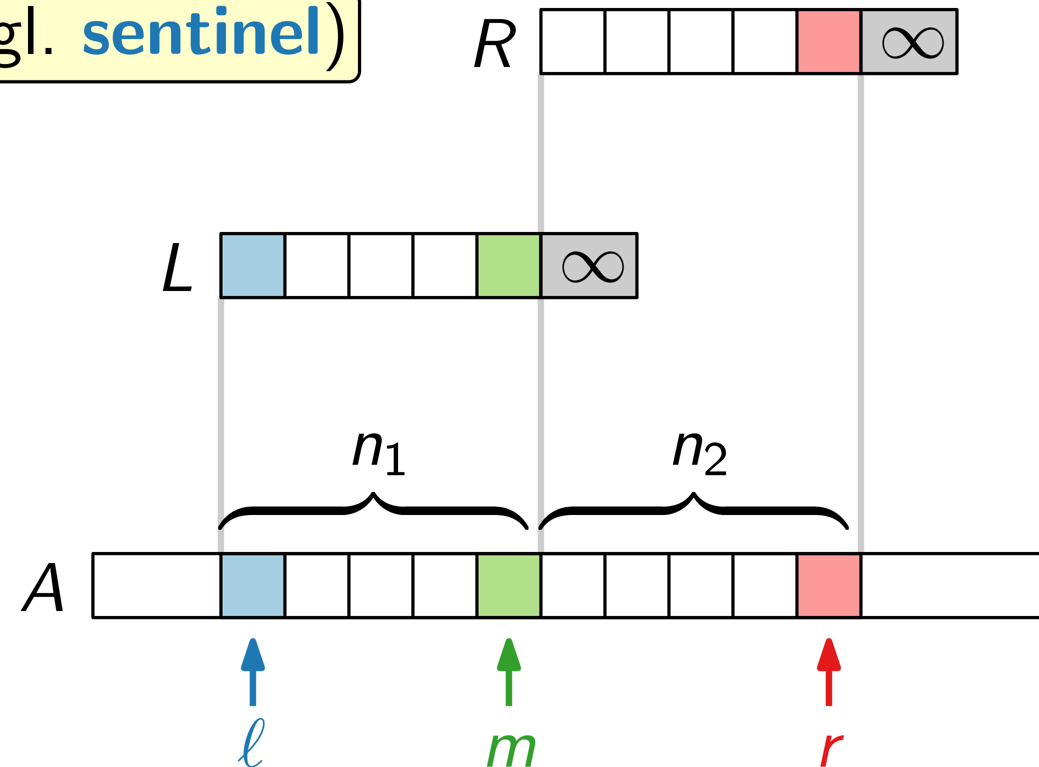
$L = \textbf{new}\ \text{int}[1 \dots n_1 + 1];\ R = \textbf{new}\ \text{int}[1 \dots n_2 + 1]$

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

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

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

Stopper (engl. **sentinel**)



Kombiniere

Merge(int[] A, int ℓ, int m, int r)

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

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

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

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

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

$i = j = 1$

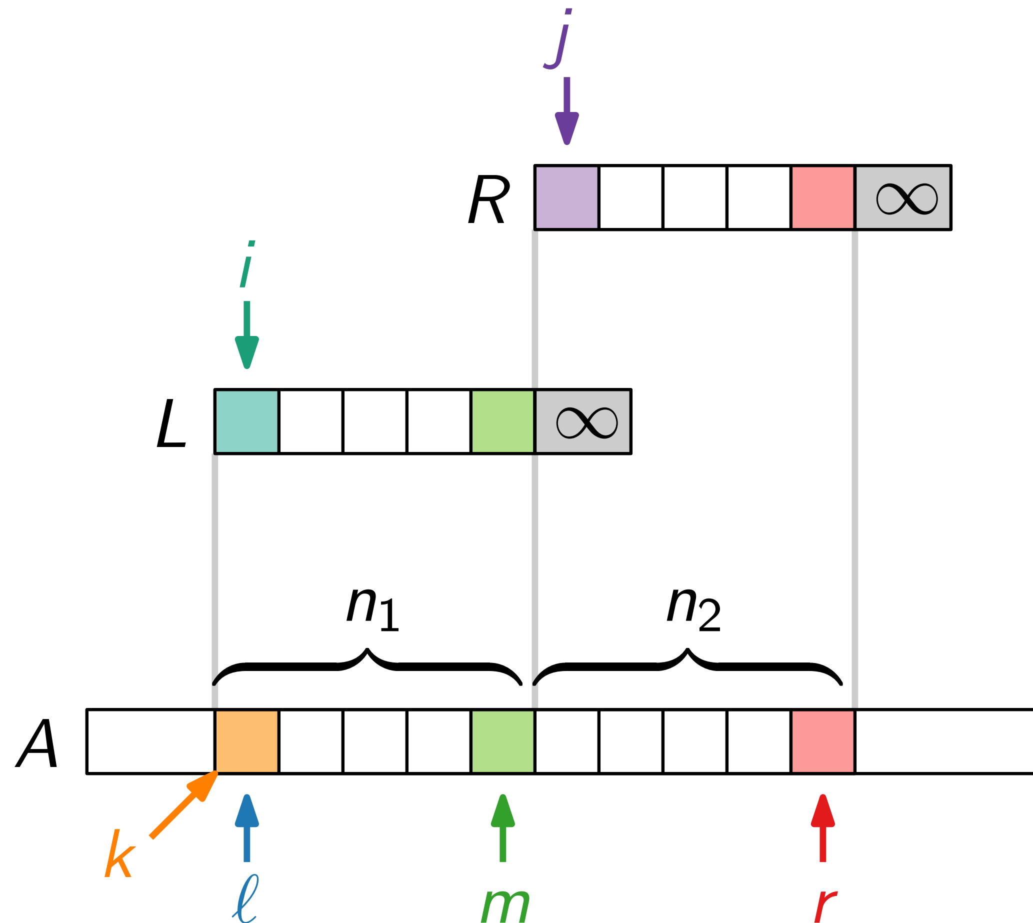
for $k = \ell$ **to** r **do**

Aufgabe.

Schreiben Sie den Rest der Routine auf ein Stück Papier!

Benutzen Sie dazu L und R.

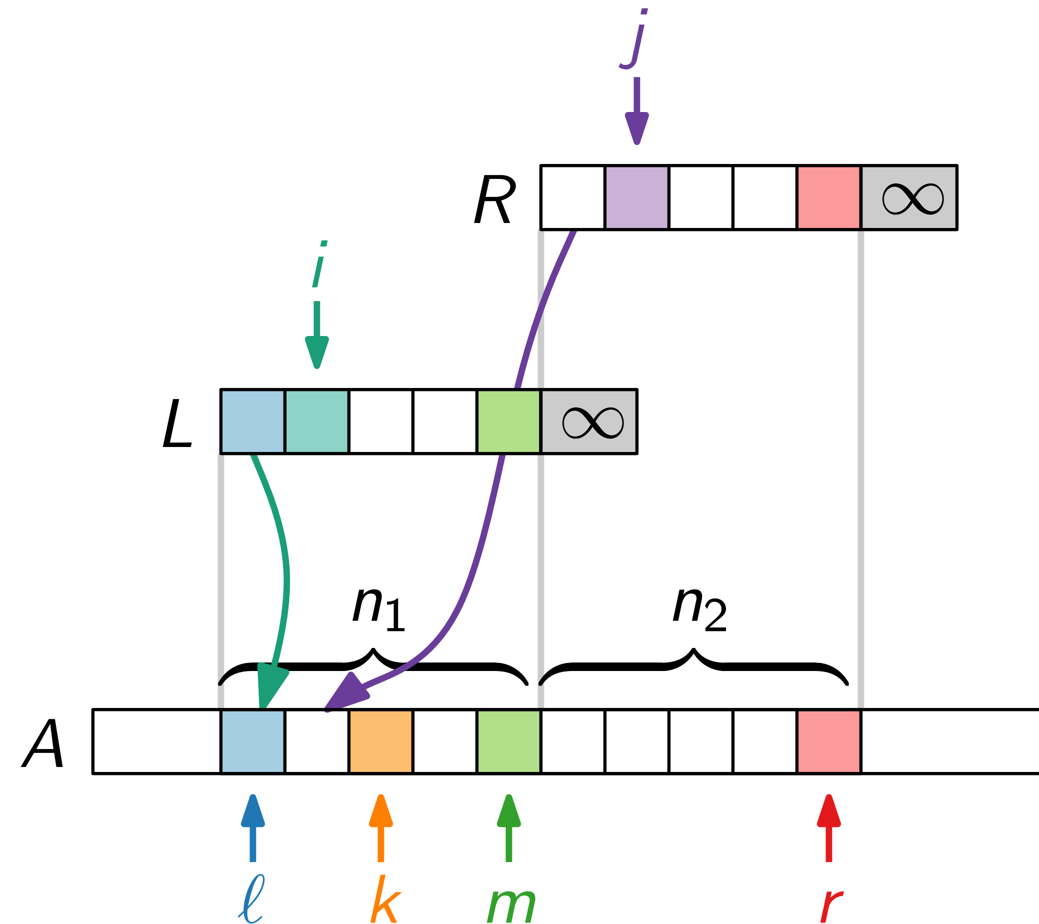
[5 Minuten]



Kombiniere

```
Merge(int[] A, int ℓ, int m, int r)
  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
```

Aber... stimmt das denn alles?



MergeSort – ein Beispiel

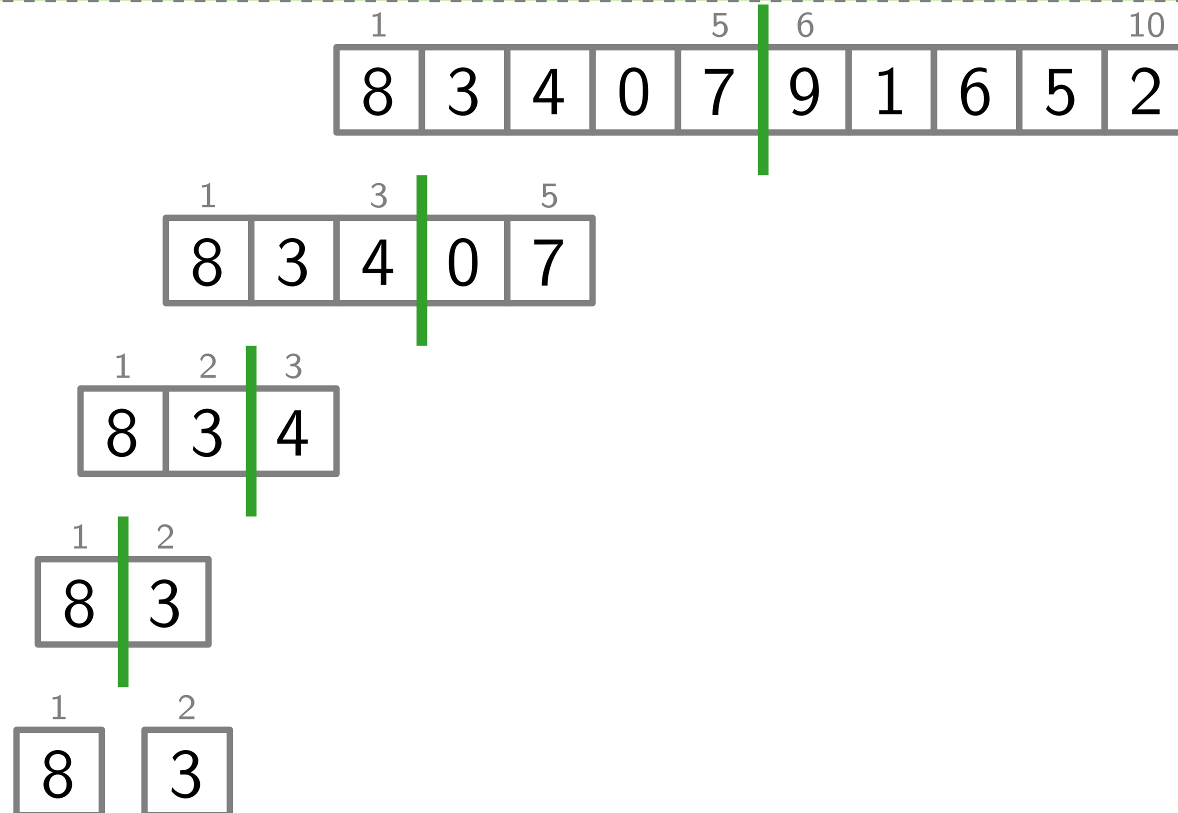
```
MergeSort(int[] A, int ℓ = 1, int r = A.length)
if ℓ < r then
    m = ⌊(ℓ + r)/2⌋                 } teile
    MergeSort(A, ℓ, m)              } herrsche
    MergeSort(A, m + 1, r)          }
    Merge(A, ℓ, m, r)               } kombiniere
```

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



MergeSort – ein Beispiel

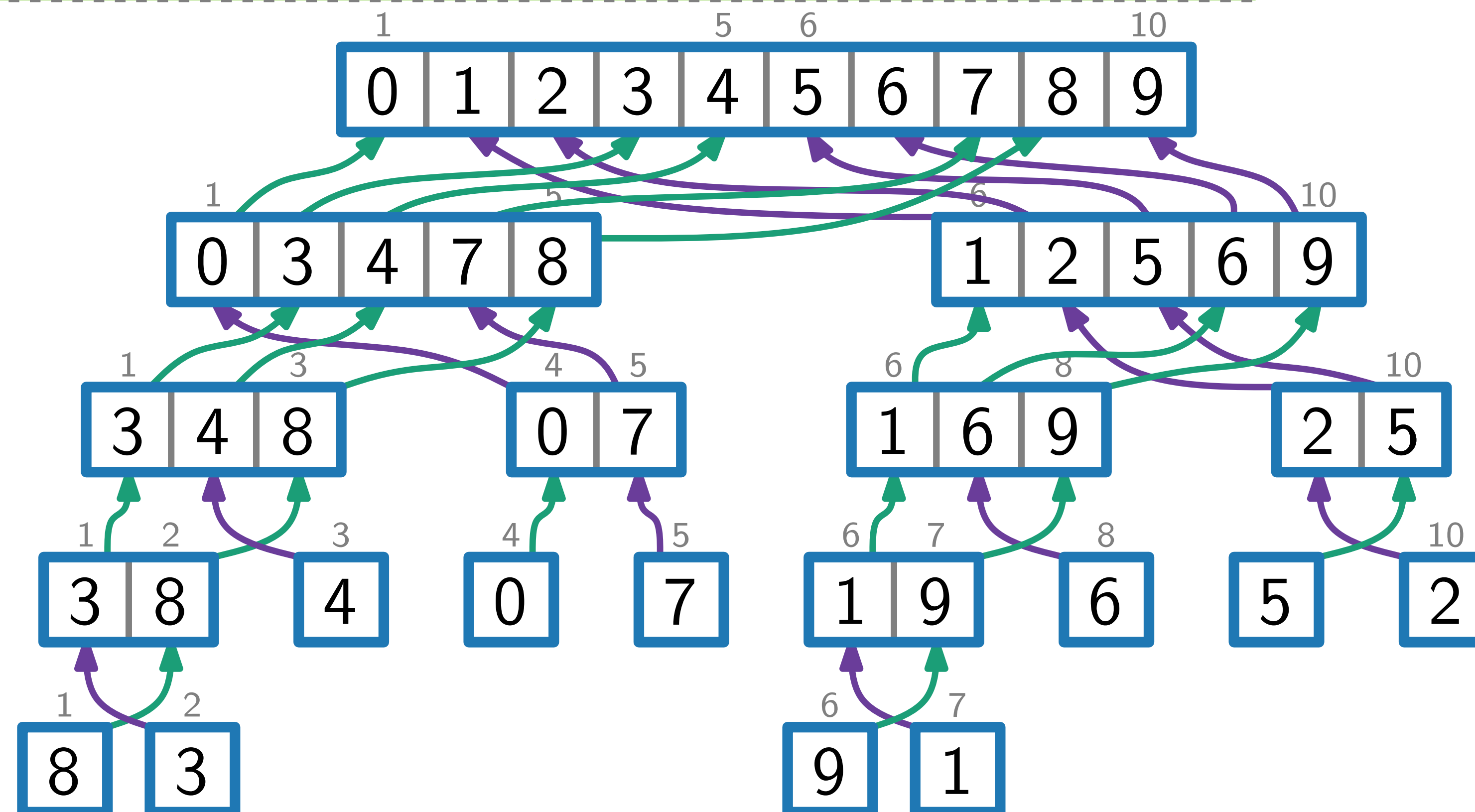
MergeSort(int[] A, int $\ell = 1$, int $r = A.length$)

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

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

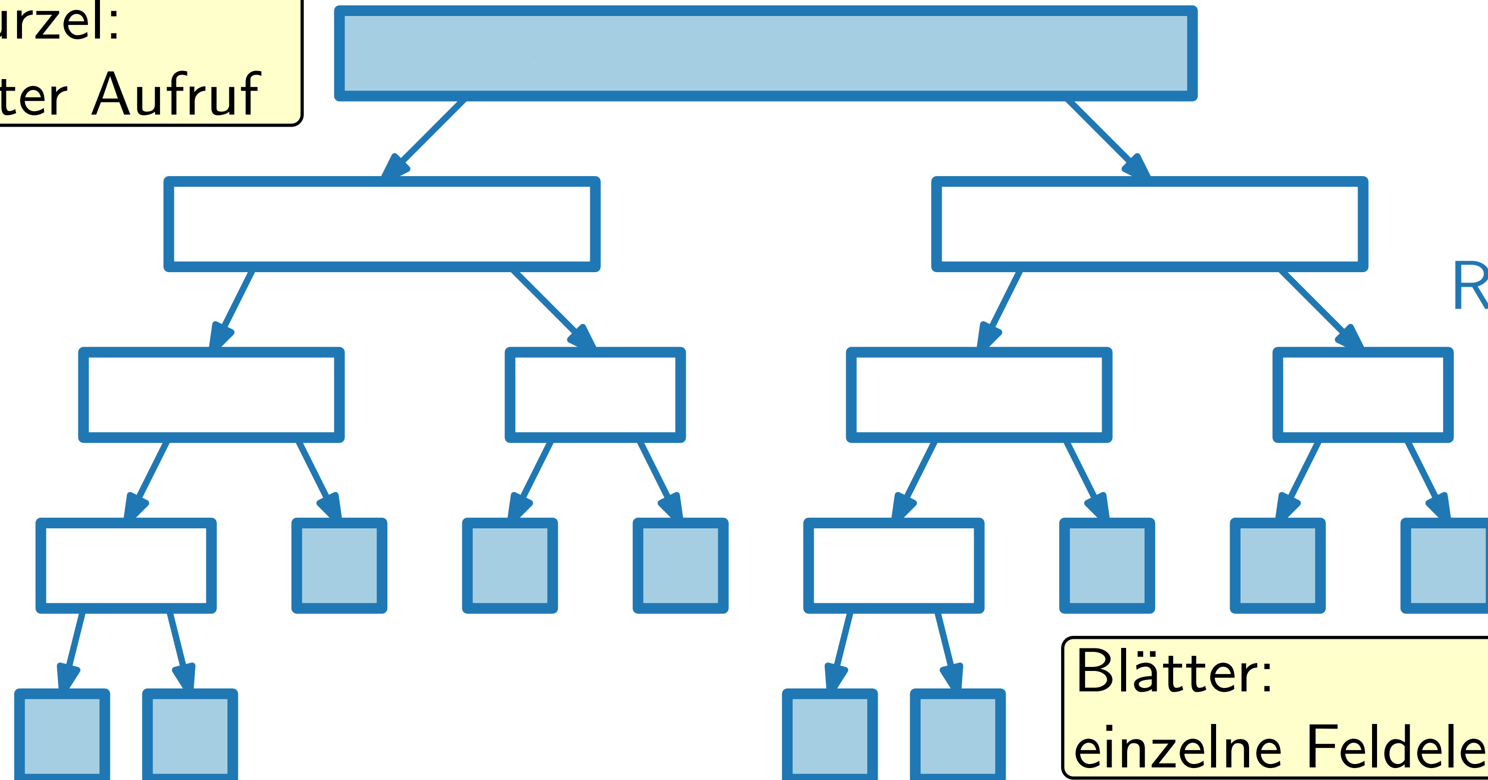
MergeSort(A, ℓ, m)
MergeSort($A, m + 1, r$) } **herrsche**

Merge(A, ℓ, m, r) } **kombiniere**



MergeSort – ein Beispiel

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



Rekursionsbaum von MergeSort

Baum der rekursiven Aufrufe

Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

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

1. Initialisierung ✓

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


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

Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

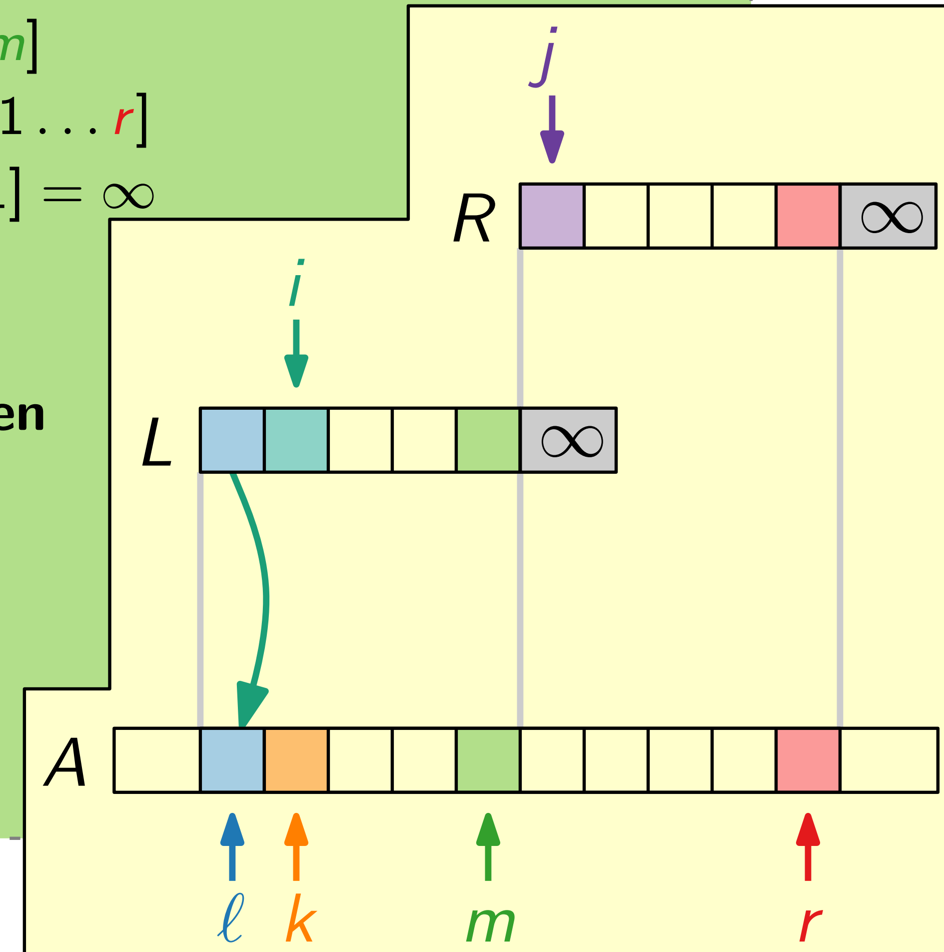
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

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
```



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

dank Invariante

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

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

$\Rightarrow$ Invariante!

Korrektheit von Merge

... nach Schema „F"!

0. Schleifeninvariante

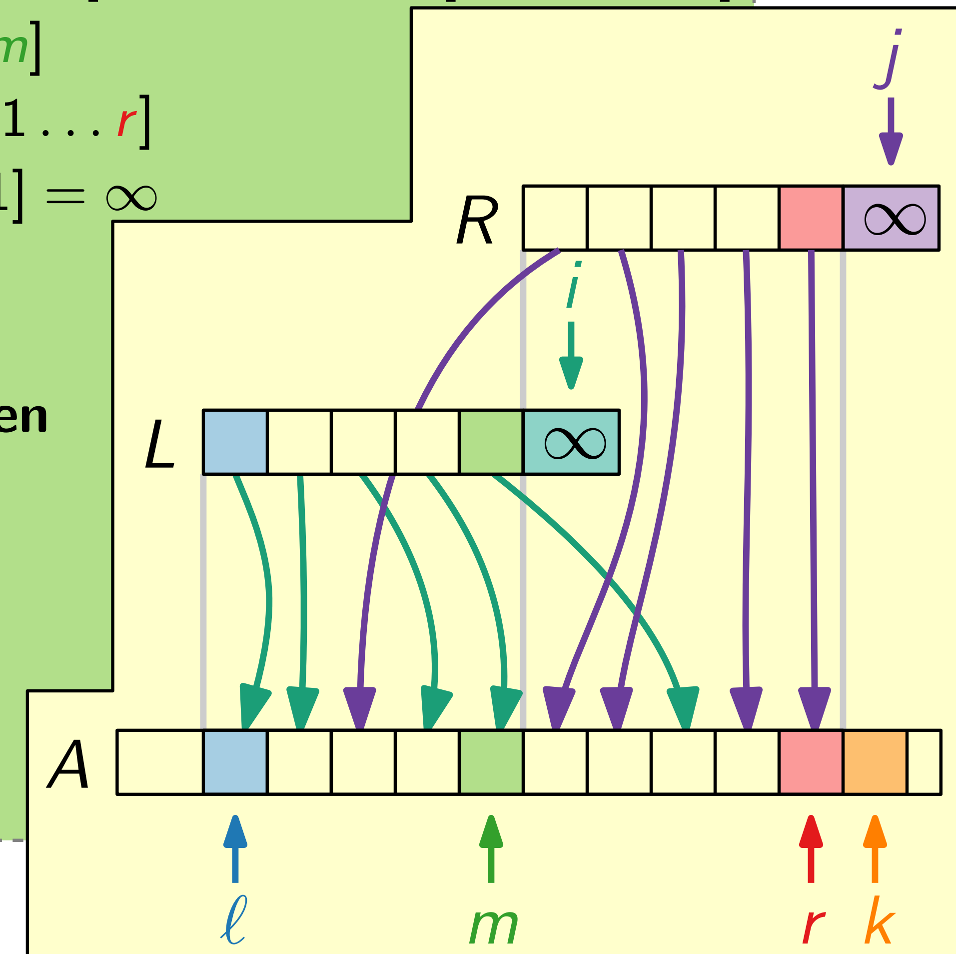
- $A[\ell \dots k-1]$ enthält die $k-\ell$ kleinsten Elemente von $L \cup R$ **sortiert**.
- $L[i]$ und $R[j]$ sind die kleinsten Elemente in L bzw. R, die noch nicht in A kopiert wurden.

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

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



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

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

+2 Stopper

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

Korrektheit von Merge

... nach Schema „F“!

0. Schleifeninvariante

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

1. Initialisierung ✓

2. Aufrechterhaltung ✓

3. Terminierung ✓

Also ist Merge korrekt! □

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

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

Und MergeSort? Korrekt? Effizient?

Korrektheit von MergeSort

```
MergeSort(int[] A, int ℓ = 1, int r = A.length)
if ℓ < r then
    m = ⌊(ℓ + r)/2⌋                         } teile
    MergeSort(A, ℓ, m)                      } herrsche
    MergeSort(A, m + 1, r)                  }
    Merge(A, ℓ, m, r)                       } kombiniere
```

Korrekt? Welche Beweistechnik? Hm, MergeSort ist **rekursiv**...

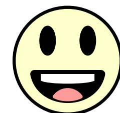
Vollständige Induktion über $n = r - \ell + 1$ ($= A[\ell \dots r].length$):

$n = 1$: **Induktionsanfang**

Dann ist $\ell = r$.

$\Rightarrow$ **if**-Block wird nicht betreten.

D.h. nichts passiert.

OK, da $A[\ell \dots \ell]$ schon sortiert. 

Korrektheit von MergeSort

```
MergeSort(int[] A, int ℓ = 1, int r = A.length)
 if ℓ < r then
   m = ⌊(ℓ + r)/2⌋                          } teile
   MergeSort(A, ℓ, m)                       } herrsche
   MergeSort(A, m + 1, r)                   }
   Merge(A, ℓ, m, r)                        } kombiniere
```

$n > 1$: **Induktionsschritt**

Induktionsannahme: MergeSort korrekt für Felder der Länge $< n$.

Wegen $n > 1$ ist $\ell < r$. $\Rightarrow$ **if**-Block wird betreten.

Nach Wahl von m gilt $\ell \leq m < r$.

$\Rightarrow A[\ell \dots m]$ und $A[m+1 \dots r]$ sind **kürzer** als $A[\ell \dots r]$.

$\underset{\text{I.A.}}{\Rightarrow}$ MergeSort(A, ℓ, m) ist korrekt und
MergeSort$(A, m+1, r)$ ist korrekt.
Schon bewiesen: Merge ist korrekt.

$\Big\}$ MergeSort(A, ℓ, r) ist korrekt. $\square$

Übersicht

Techniken für Korrektheitsbeweise

- iterative Algorithmen (à la InsertionSort, Factorial, Merge)

  per **Schleifeninvariante** (Schema „F“)

- rekursive Algorithmen (à la MergeSort)

  per **vollständige Induktion**

Laufzeit?