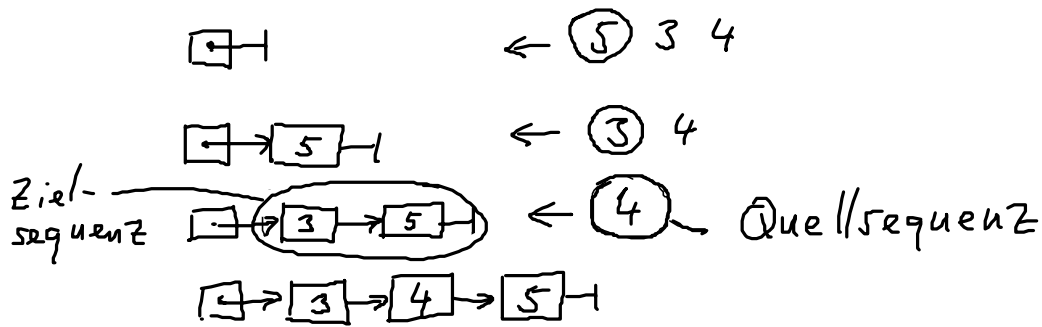


## 10.1.3 Insertion Sort

Idee  $\rightarrow$  siehe Aufgabe 44



jetzt im Array

- $\rightarrow$  andere Suche
- $\rightarrow$  anderes Einfügen

Informale Beschreibung:

1. geg. Array  $vec[n]$ . Zu Anfang sei  $vec[0]$  die Zielsequenz (sortiert) und  $vec[1], \dots, vec[n-1]$  die Quellsequenz (unsortiert)

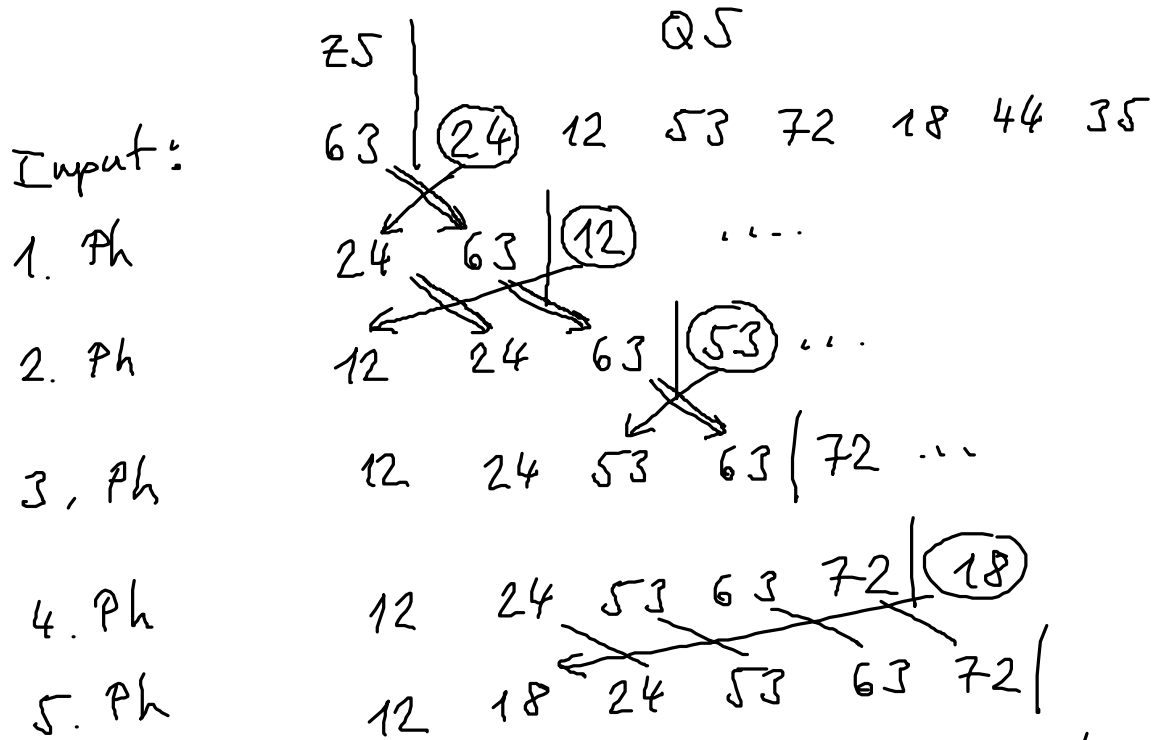
2.  $n-1$  Phasen

3. in Phase  $i$ : nehme  $vec[i]$  aus QS und füge in ZS an richtiger Stelle ein.

Invariante: nach Phase  $i$  gilt für die akt. ZS

$$vec[0].key \leq vec[1].key \leq \dots \leq vec[i].key$$

Zustandsverlauf:



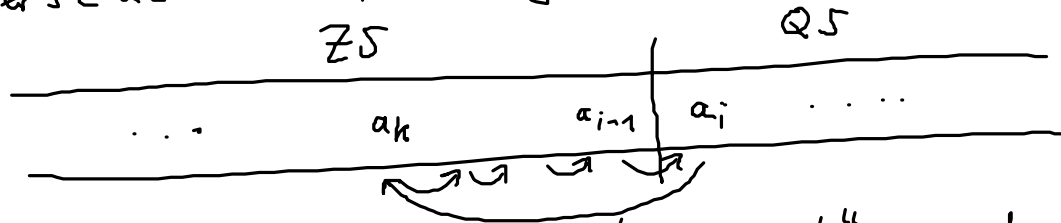
Anzahl Vergleiche: kann ZS binär durchsuchen

$\Rightarrow \lfloor \log i \rfloor + 1$  Vergleiche in Phase  $i$   
 $C(n) = \sum_{i=1}^{n-1} (\lfloor \log i \rfloor + 1) \leq \sum_{i=1}^{n-1} (\log(n-1) + 1)$

$= (n-1) [\log(n-1) + 1]$   
 $\in O(n \log n)$

(in lin. Liste  $\Rightarrow C(n) = \sum_{i=1}^{n-1} i \in O(n^2)$ )

Anzahl Zuweisungen: muss Teil der ZS nach rechts verschieben  $\rightarrow$  Ringtausch im Array



max. Länge der Verschiebung = Länge der ZS =  $i$

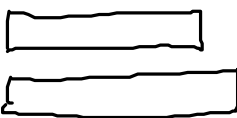
$\Rightarrow A(n) \leq \sum_{i=1}^{n-1} i \in O(n^2)$

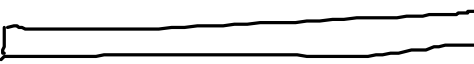
(in lin. Liste je 2 Zuweisung je Phase  $\Rightarrow A(n) \in O(n)$ )

# 10.2 Mergesort

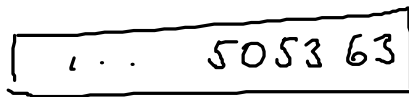
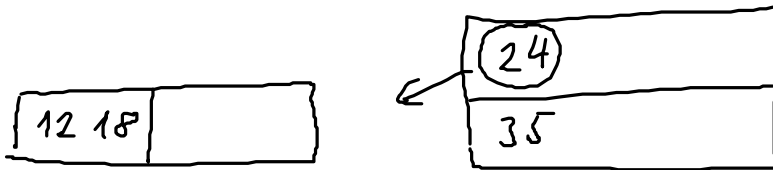
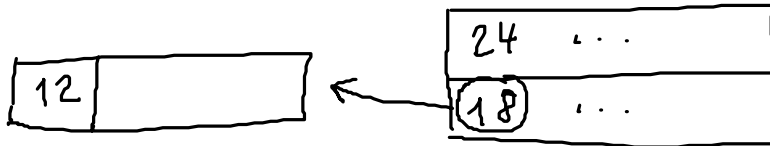
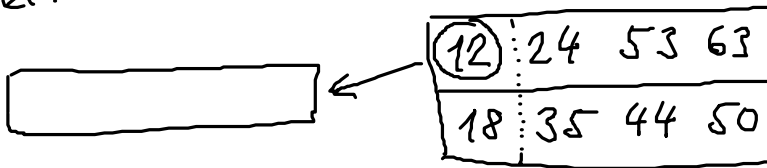
Sortieren durch Mischen ("merge")

## 10.2.1 Mischen sortierter Arrays

Input:   $n$  } nichtabsteigend "≤"  
 $m$  } sortierte Arrays

Output:   $m+n$  nichtabst. sortiert

Beispiel:



merge (vec1, vec2)

Input: nichtabst. sort. Arrays vec1, vec2  
mit Längen  $m, n$  (Indizes ab 0)

Output: nichtabst. sort. Array der Länge  $m+n$   
mit den Werten aus vec1, vec2

init. Array  $vec[]$  der Länge  $m+n$

$i := j := k := 0$

while  $i < m$  and  $j < n$  do { mischen }

if  $vec1[i].key < vec2[j].key$  then

$vec[k] := vec1[i]$

$i := i + 1$

else

$vec[k] := vec2[j]$

$j := j + 1$

endif

$k := k + 1$

end while

if  $i = m$  then

$vec[k, \dots, m+n-1] := vec2[j, \dots, n-1]$

else

$vec[k, \dots, m+n-1] := vec1[i, \dots, m-1]$

endif

return  $vec$

---

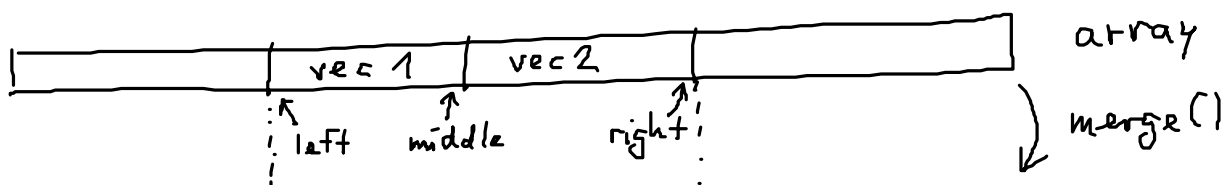
Anzahl Vergleiche: 1 Vergl. pro while-Iteration  
max.  $m+n-1$  Iterationen

$$\Rightarrow C(m, n) \leq m+n-1$$

Anzahl Zuweisungen: müssen Array  $vec$  zuweisen

$$\Rightarrow A(m, n) = |vec| = m+n$$

In der Praxis: muss in situ mischen, d. h.





→ brauche Methode mit Schnittstelle

merge (vec, left, middle, right)

Input: Array vec []

Index left, wo vec1 beginnt

Index middle, wo vec1 endet und bei middle + 1 vec2 beginnt

Index right, wo vec2 endet

Output: vec, zwischen left und right sortiert

### 10.2.2 Mergesort: rekursives Mischen

mergesort (vec)

Input: Array vec [] der Länge n (Indizes ab 0)

Output: vec, nichtabst. sortiert

sortMe (vec, 0, n-1)

{ rek. Erstaufruf }

return vec

sortMe (vec, first, last)

Input: Array vec [], Indizes first, last

Output: vec, zwischen first und last nichtabst. sortiert

if first  $\neq$  last then

middle :=  $\lfloor \frac{1}{2} (first + last) \rfloor$

Aktion 

sortMe (vec, first, middle)
sortMe (vec, middle+1, last)
merge (vec, first, middle, last)

 } Rekursion

endif

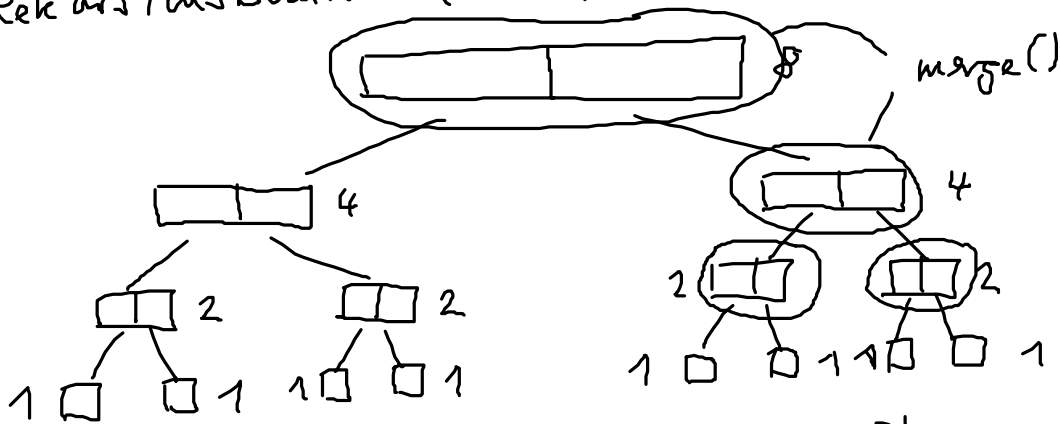
return vec

Korrektheit:  $\text{sortMe}()$  arbeitet korrekt

Bew: (IA)  $n=1 \Rightarrow \text{sortMe}(\text{vec}, 0, 0)$  tut nichts  
und  $\text{vec}[0]$  ist sortiert. ✓

(IS)  $n > 1$ : nach Wahl von  $\text{middle}$  sind die Bereiche  
 $[\text{first}, \text{middle}]$  und  $[\text{middle}+1, \text{last}]$  kürzer  
als  $n$ . (IV)  $\Rightarrow \text{sortMe}()$  sortiert beide Teile  
korrekt.  $\text{merge}()$  korrekt  $\Rightarrow \text{sortMe}()$  korrekt. □

Rekursionsbaum ( $n=8$ ):



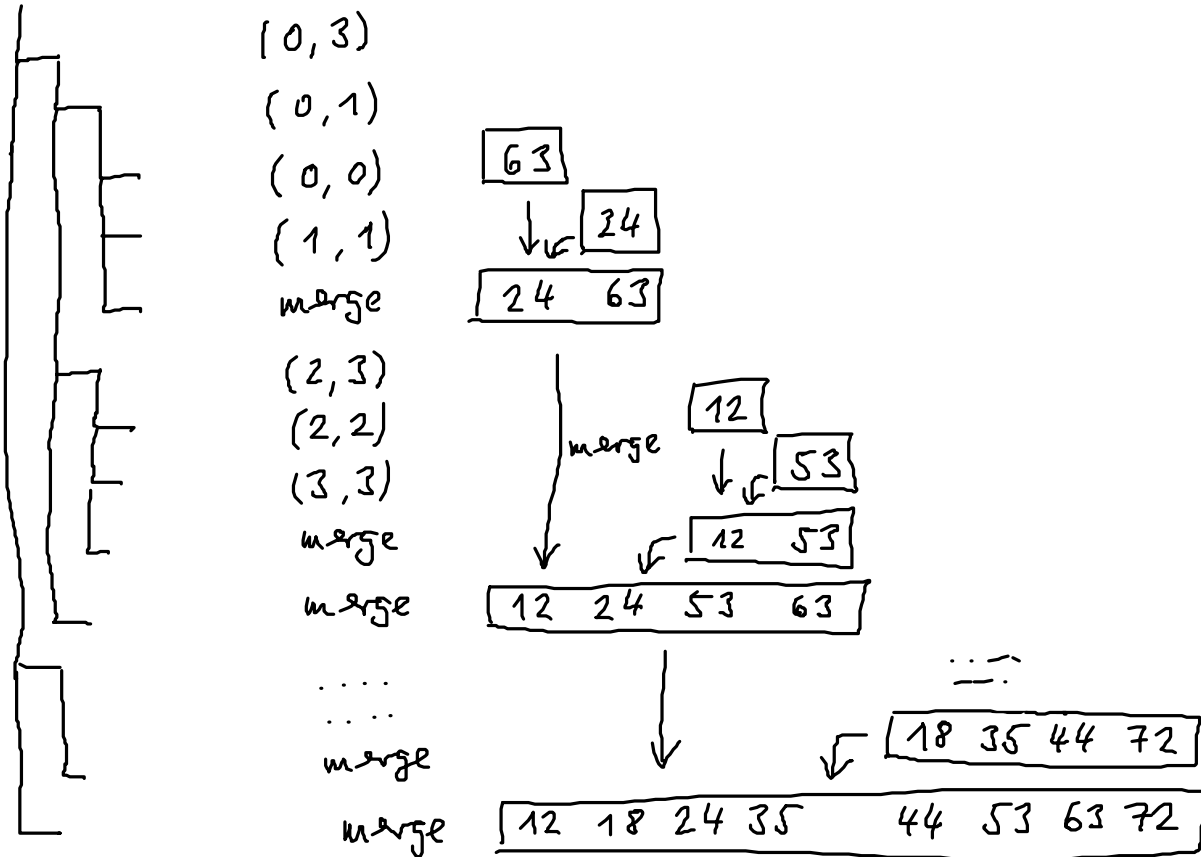
Aus dem lesen wir ab: ( $n=2^9$ )

$$\begin{aligned} R(2^9) &= 9 + 1 \\ T(2^9) &= \sum_{k=0}^9 2^k = 2^{9+1} - 1 \end{aligned}$$

Beispiel:

$\text{sortMe}(\text{vec}, 0, 7)$

63 24 12 53 72 18 44 35 vec



### 10.2.3 Analyse

gesehen: für  $n = 2^q$  ist Rekursionsbaum vollst. bin. Baum.

→ kenne seine Eigenschaften

→ sei bis auf weiteres zunächst  $n = 2^q$

Aufbau von  $\text{sortMe}()$  liefert sofort Rekursionsformeln:

$$1) \quad C(2) = 1, \quad C(2n) = 2C(n) + C(n, n) \quad (n \geq 2)$$

wissen aus 10.2.1:  $C(m, n) = m + n - 1$

$$\Rightarrow \boxed{C(2) = 1, \quad C(2n) = 2C(n) + 2n - 1} \quad (n \geq 2)$$

suche geschlossene Formel für  $C(n)$

(→ Skript für Idee)

$$\text{Beh: } \boxed{C(2^q) = (q-1)2^q + 1}$$

Bew: (IA)  $q=1$ :  $C(2) = 0 \cdot 2 + 1 = 1 \checkmark$

$$(IS) \quad C(2^{q+1}) = 2 C(2^q) + 2 \cdot 2^q - 1$$

$$(IV) = 2 \cdot [(q-1)2^q + 1] + 2^{q+1} - 1 \\ = q 2^{q+1} + 1 \quad \square$$

2)  $A(2) = 4$ ,  $A(2n) = 2 A(n) + \underbrace{A(n, n)}_{10.2.1 \Rightarrow 2n} + \underbrace{\text{Kopien der Arrays}}_{2n}$

$$A(2) = 4, \quad A(2n) = 2 A(n) + 4n \quad (n > 2)$$

$$\Rightarrow \quad A(2^q) = q 2^{q+1}$$

3) Was wenn  $n \neq 2^q$  für jedes  $q$ ?

Setze  $q := \lceil \log_2 n \rceil$

dann:  $2^{q-1} < n \leq 2^q \Leftrightarrow q-1 < \log_2 n \leq q$

$C(n)$ ,  $A(n)$  monoton in  $n$

$$\Rightarrow \quad C(n) \leq C(2^q) = (q-1)2^q + 1 \\ < \log_2 n \cdot 2n + 1 \in O(n \log n)$$
$$\Rightarrow \quad A(n) \leq A(2^q) = q 2^{q+1} \\ < (\log_2 n + 1) \cdot 4n \in O(n \log n)$$

Aussagen gelten immer  $\Rightarrow$  auch im worst case!