

# *Kapitel 4: Dynamische Datenstrukturen*

*Algorithmen und Datenstrukturen  
WS 2017/18*

**Prof. Dr. Sándor Fekete**



# 4.1 Grundoperationen

## Aufgabenstellung:



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*

## Im Folgenden:



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*

## Im Folgenden:

**$S$  Menge von Objekten**



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*

## Im Folgenden:

<b>S</b>	<b>Menge von Objekten</b>
<b>k</b>	<b>Wert eines Elements (“Schlüssel”)</b>



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*

## Im Folgenden:

<b>S</b>	<b>Menge von Objekten</b>
<b>k</b>	<b>Wert eines Elements (“Schlüssel”)</b>
<b>x</b>	<b>Zeiger auf Element</b>



# 4.1 Grundoperationen

## Aufgabenstellung:

- *Verwalten einer Menge  $S$  von Objekten*
- *Ausführen von verschiedenen Operationen (s.u.)*

## Im Folgenden:

<b>S</b>	<b>Menge von Objekten</b>
<b>k</b>	<b>Wert eines Elements (“Schlüssel”)</b>
<b>x</b>	<b>Zeiger auf Element</b>
<b>NIL</b>	<b>spezieller, “leerer” Zeiger</b>



## 4.1 Grundoperationen



## 4.1 Grundoperationen

**SEARCH(S,k): “Suche in S nach k”**



## 4.1 Grundoperationen

**SEARCH(S,k): “Suche in S nach k”**



## 4.1 Grundoperationen

**SEARCH(S,k): “Suche in S nach k”**

**Durchsuche die Menge S nach einem Element von Wert k.**



## 4.1 Grundoperationen

**SEARCH(S,k): “Suche in S nach k”**

**Durchsuche die Menge S nach einem Element von Wert k.**



## 4.1 Grundoperationen

**SEARCH(S,k): “Suche in S nach k”**

**Durchsuche die Menge S nach einem Element von Wert k.**

**Ausgabe: Zeiger x, falls x existent**



## 4.1 Grundoperationen

**SEARCH(S,k): “Suche in S nach k”**

**Durchsuche die Menge S nach einem Element von Wert k.**

**Ausgabe: Zeiger x, falls x existent  
NIL, falls kein Element Wert k hat.**



## 4.1 Grundoperationen

**SEARCH(S,k): “Suche in S nach k”**

**Durchsuche die Menge S nach einem Element von Wert k.**

**Ausgabe: Zeiger x, falls x existent  
NIL, falls kein Element Wert k hat.**



## 4.1 Grundoperationen



## 4.1 Grundoperationen

**INSERT(S,x):** “Füge x in S ein”



## 4.1 Grundoperationen

**INSERT(S,x): “Füge x in S ein”**



## 4.1 Grundoperationen

**INSERT(S,x): “Füge x in S ein”**

**Erweitere S um das Element, das unter der Adresse x steht.**



## 4.1 Grundoperationen

**INSERT(S,x): “Füge x in S ein”**

**Erweitere S um das Element, das unter der Adresse x steht.**



## 4.1 Grundoperationen



## 4.1 Grundoperationen

**DELETE(S,x):** “Entferne x aus S”



## 4.1 Grundoperationen

**DELETE(S,x): “Entferne x aus S”**



## 4.1 Grundoperationen

**DELETE(S,x): “Entferne x aus S”**

**Lösche das unter der Adresse x stehende Element aus der Menge S.**



## 4.1 Grundoperationen

**DELETE(S,x): “Entferne x aus S”**

**Lösche das unter der Adresse x stehende Element aus der Menge S.**



## 4.1 Grundoperationen



## 4.1 Grundoperationen

**MINIMUM(S):** “Suche das Minimum in S”



## 4.1 Grundoperationen

**MINIMUM(S):** “Suche das Minimum in S”



## 4.1 Grundoperationen

**MINIMUM(S): “Suche das Minimum in S”**

**Finde in S ein Element von kleinstem Wert.**



## 4.1 Grundoperationen

**MINIMUM(S): “Suche das Minimum in S”**

**Finde in S ein Element von kleinstem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**



## 4.1 Grundoperationen

**MINIMUM(S): “Suche das Minimum in S”**

**Finde in S ein Element von kleinstem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**



## 4.1 Grundoperationen

**MINIMUM(S): “Suche das Minimum in S”**

**Finde in S ein Element von kleinstem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**

**Ausgabe: Zeiger x auf solch ein Element**



## 4.1 Grundoperationen

**MINIMUM(S): “Suche das Minimum in S”**

**Finde in S ein Element von kleinstem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**

**Ausgabe: Zeiger x auf solch ein Element**



## 4.1 Grundoperationen



## 4.1 Grundoperationen

**MAXIMUM(S):** “Suche das Maximum in S”



## 4.1 Grundoperationen

**MAXIMUM(S):** “Suche das Maximum in S”



## 4.1 Grundoperationen

**MAXIMUM(S):** “Suche das Maximum in S”

**Finde in S ein Element von größtem Wert.**



## 4.1 Grundoperationen

**MAXIMUM(S):** “Suche das Maximum in S”

**Finde in S ein Element von größtem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**



## 4.1 Grundoperationen

**MAXIMUM(S): “Suche das Maximum in S”**

**Finde in S ein Element von größtem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**



## 4.1 Grundoperationen

**MAXIMUM(S):** “Suche das Maximum in S”

**Finde in S ein Element von größtem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**

**Ausgabe: Zeiger x auf solch ein Element**



## 4.1 Grundoperationen

**MAXIMUM(S): “Suche das Maximum in S”**

**Finde in S ein Element von größtem Wert.  
(Annahme: Die Werte lassen sich vollständig  
vergleichen!)**

**Ausgabe: Zeiger x auf solch ein Element**



## 4.1 Grundoperationen



## 4.1 Grundoperationen

**PREDECESSOR(S,x):**

**“Finde das nächstkleinere Element”**



## 4.1 Grundoperationen

**PREDECESSOR(S,x):**

**“Finde das nächstkleinere Element”**



## 4.1 Grundoperationen

**PREDECESSOR(S,x):**

**“Finde das nächstkleinere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstkleinerem  
Wert in S.**



## 4.1 Grundoperationen

**PREDECESSOR(S,x):**

**“Finde das nächstkleinere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstkleinerem  
Wert in S.**



## 4.1 Grundoperationen

**PREDECESSOR(S,x):**

**“Finde das nächstkleinere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstkleinerem  
Wert in S.**

**Ausgabe: Zeiger y auf Element**



## 4.1 Grundoperationen

**PREDECESSOR(S,x):**

**“Finde das nächstkleinere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstkleinerem  
Wert in S.**

**Ausgabe: Zeiger y auf Element  
NIL, falls x Minimum von S angibt**



## 4.1 Grundoperationen

**PREDECESSOR(S,x):**

**“Finde das nächstkleinere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstkleinerem  
Wert in S.**

**Ausgabe: Zeiger y auf Element  
NIL, falls x Minimum von S angibt**



## 4.1 Grundoperationen



## 4.1 Grundoperationen

**SUCCESSOR(S,x):**

**“Finde das nächstgrößere Element”**



## 4.1 Grundoperationen

**SUCCESSOR(S,x):**

**“Finde das nächstgrößere Element”**



## 4.1 Grundoperationen

**SUCCESSOR(S,x):**

**“Finde das nächstgrößere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstgrößerem  
Wert in S.**



## 4.1 Grundoperationen

**SUCCESSOR(S,x):**

**“Finde das nächstgrößere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstgrößerem  
Wert in S.**



## 4.1 Grundoperationen

**SUCCESSOR(S,x):**

**“Finde das nächstgrößere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstgrößerem  
Wert in S.**

**Ausgabe: Zeiger y auf Element**



## 4.1 Grundoperationen

**SUCCESSOR(S,x):**

**“Finde das nächstgrößere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstgrößerem  
Wert in S.**

**Ausgabe: Zeiger y auf Element  
NIL, falls x Maximum von S angibt**



## 4.1 Grundoperationen

**SUCCESSOR(S,x):**

**“Finde das nächstgrößere Element”**

**Für ein in x stehendes Element in S,  
bestimme ein Element von nächstgrößerem  
Wert in S.**

**Ausgabe: Zeiger y auf Element  
NIL, falls x Maximum von S angibt**



# 4.1 Grundoperationen



# 4.1 Grundoperationen

**Wie nimmt man das vor?**



# 4.1 Grundoperationen

**Wie nimmt man das vor?**



# 4.1 Grundoperationen

**Wie nimmt man das vor?**

**Wie lange dauert das,  
in Abhängigkeit von der Größe von  $S$ ?**



# 4.1 Grundoperationen

**Wie nimmt man das vor?**

**Wie lange dauert das,  
in Abhängigkeit von der Größe von  $S$ ?**



# 4.1 Grundoperationen

**Wie nimmt man das vor?**

**Wie lange dauert das,  
in Abhängigkeit von der Größe von S?**

**Unsortierte Unterlagen:**



# 4.1 Grundoperationen

Wie nimmt man das vor?

Wie lange dauert das,  
in Abhängigkeit von der Größe von S?

**Unsortierte Unterlagen:**

**Immer alles durchgehen, also:  $O(n)$**



# 4.1 Grundoperationen

Wie nimmt man das vor?

Wie lange dauert das,  
in Abhängigkeit von der Größe von S?

**Unsortierte Unterlagen:**

**Immer alles durchgehen, also:  $O(n)$**



# 4.1 Grundoperationen

Wie nimmt man das vor?

Wie lange dauert das,  
in Abhängigkeit von der Größe von S?

**Unsortierte Unterlagen:**

**Immer alles durchgehen, also:  $O(n)$**

**Sortierte Unterlagen: Geht schneller!**



# 4.1 Grundoperationen



# 4.1 Grundoperationen

**Langsam:**



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*  
Alle Objekte anschauen



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*

Alle Objekte anschauen

## Sehr schnell:



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*

Alle Objekte anschauen

## Sehr schnell:

- $O(1)$ : *konstante Zeit*



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*

Alle Objekte anschauen

## Sehr schnell:

- $O(1)$ : *konstante Zeit*

Immer gleich schnell, egal wie groß S ist.



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*

Alle Objekte anschauen

## Sehr schnell:

- $O(1)$ : *konstante Zeit*

Immer gleich schnell, egal wie groß S ist.

## Schnell:



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*

Alle Objekte anschauen

## Sehr schnell:

- $O(1)$ : *konstante Zeit*

Immer gleich schnell, egal wie groß S ist.

## Schnell:

- $O(\log n)$ : *logarithmische Zeit*



# 4.1 Grundoperationen

## Langsam:

- $O(n)$ : *lineare Zeit*

Alle Objekte anschauen

## Sehr schnell:

- $O(1)$ : *konstante Zeit*

Immer gleich schnell, egal wie groß S ist.

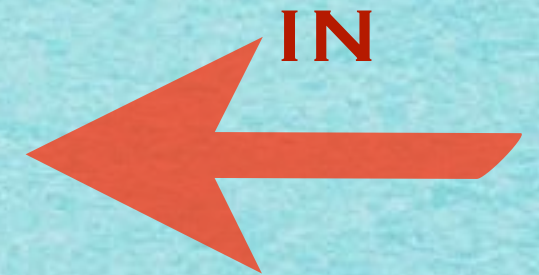
## Schnell:

- $O(\log n)$ : *logarithmische Zeit*

Wiederholtes Halbieren

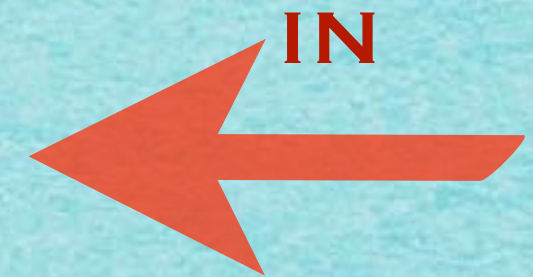
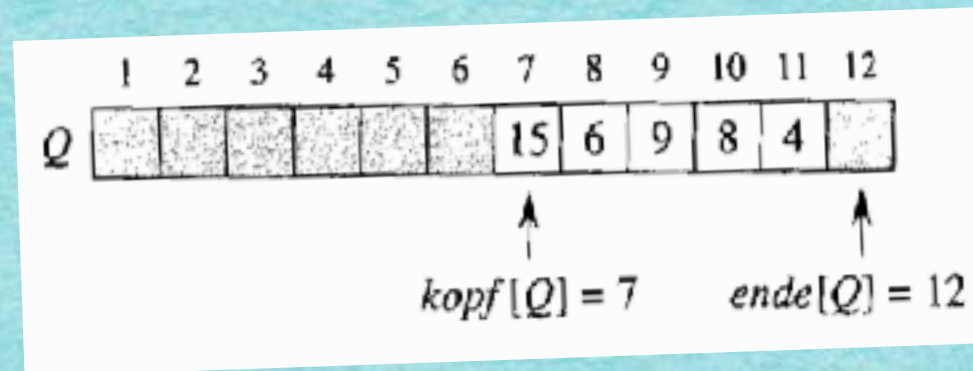
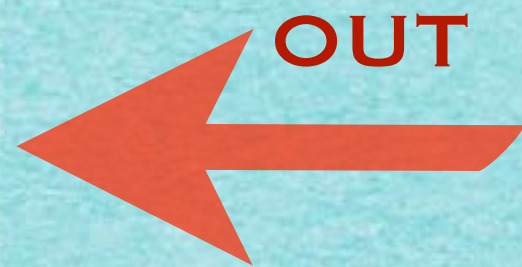


## 4.2 Stapel und Warteschlange



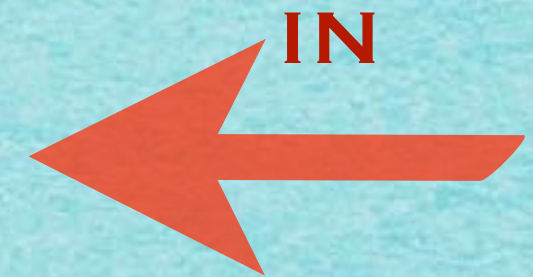
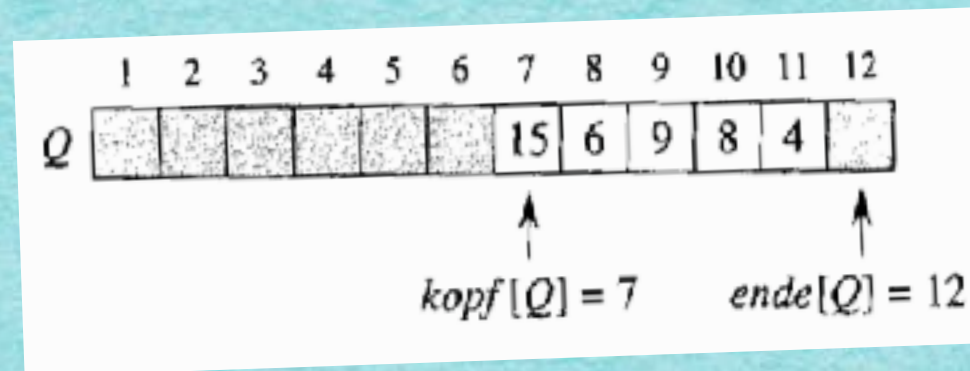
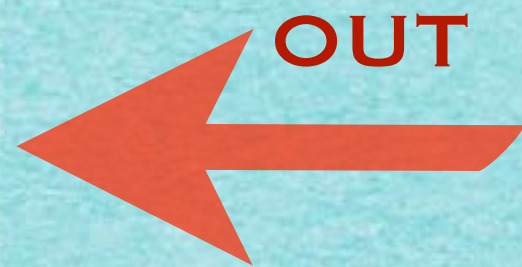


## 4.2 Stapel und Warteschlange





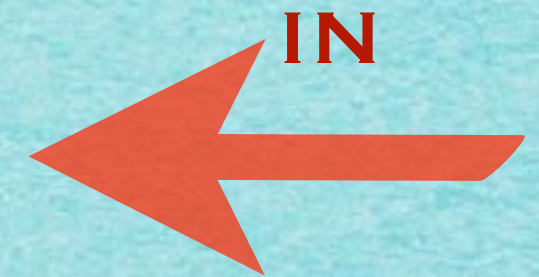
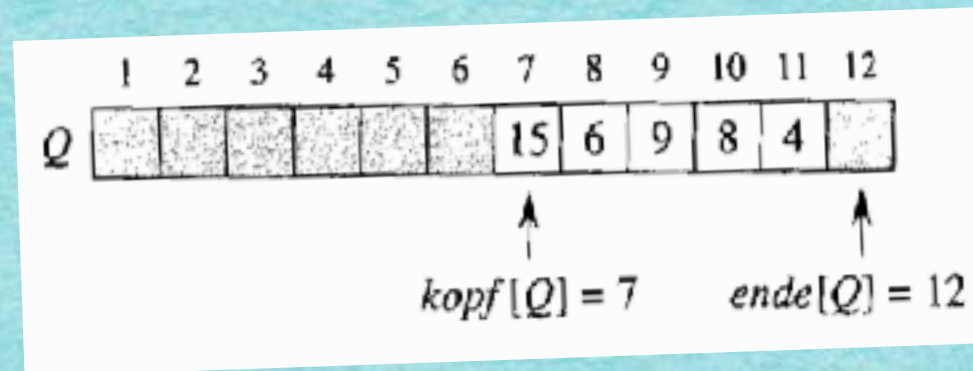
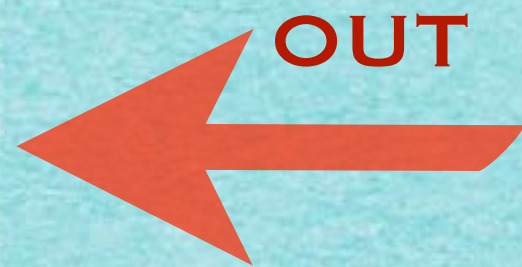
## 4.2 Stapel und Warteschlange



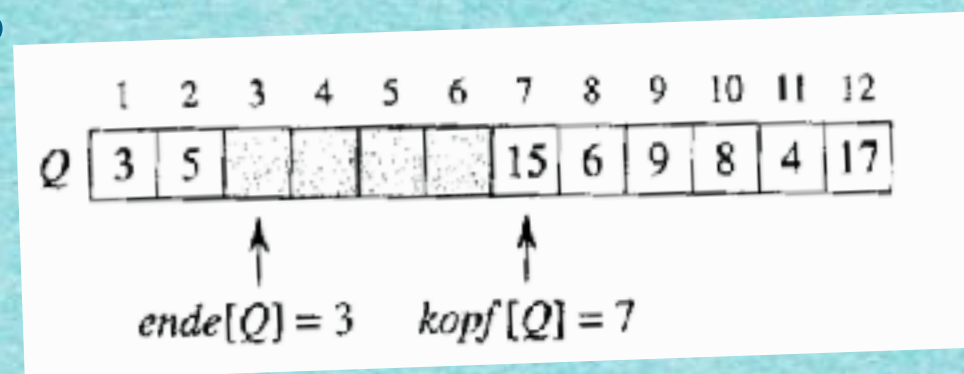
ENQUEUE: 17, 3, 5



# 4.2 Stapel und Warteschlange

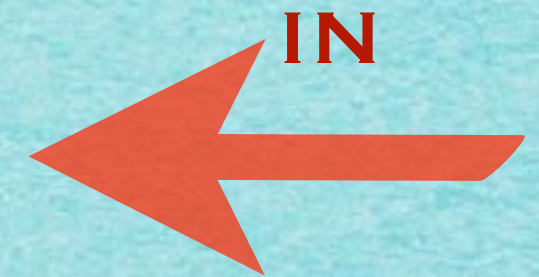
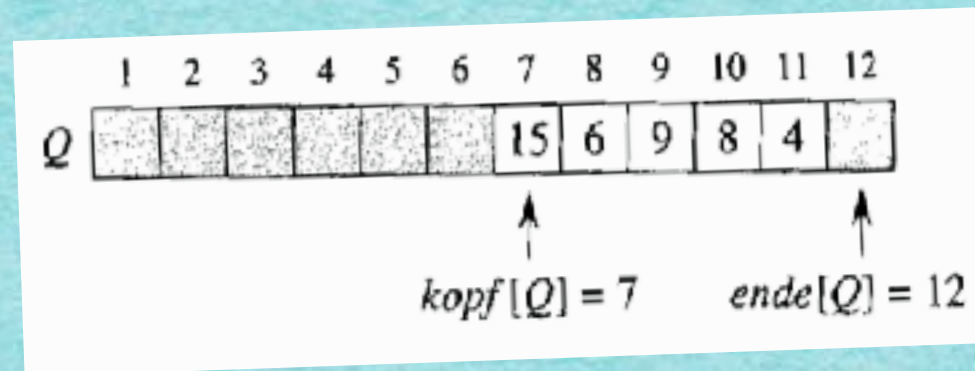
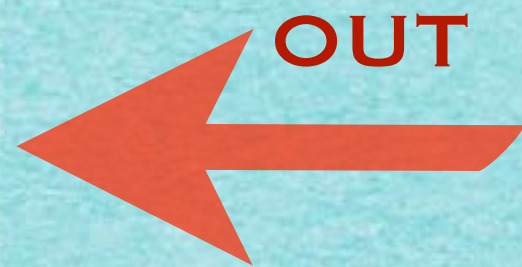


ENQUEUE: 17, 3, 5

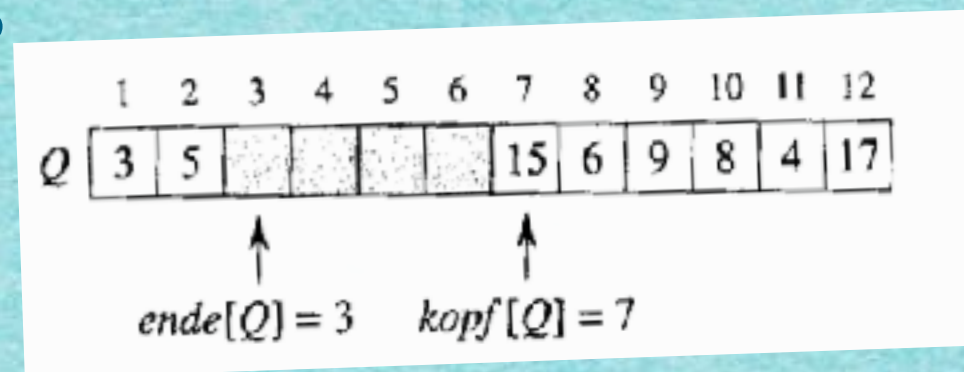




## 4.2 Stapel und Warteschlange



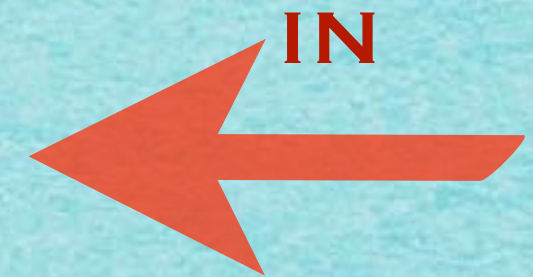
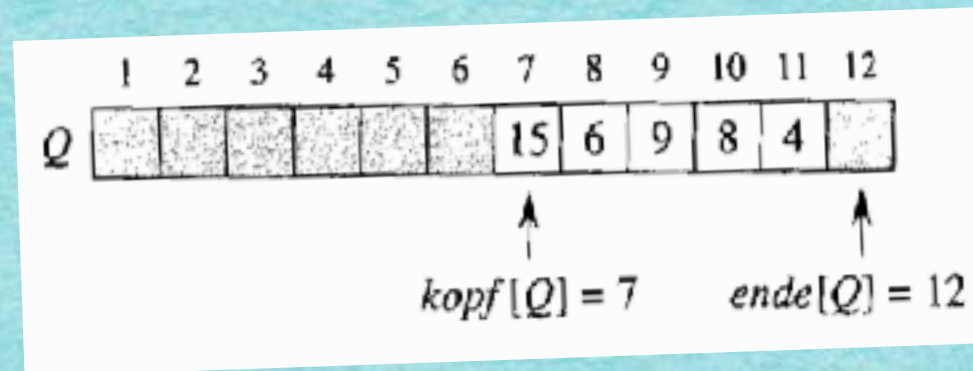
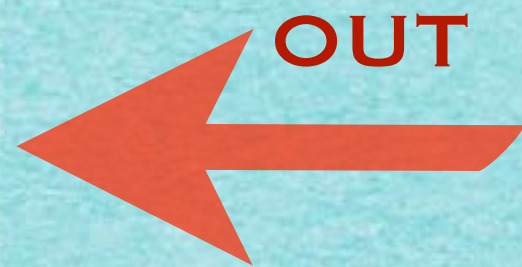
ENQUEUE: 17, 3, 5



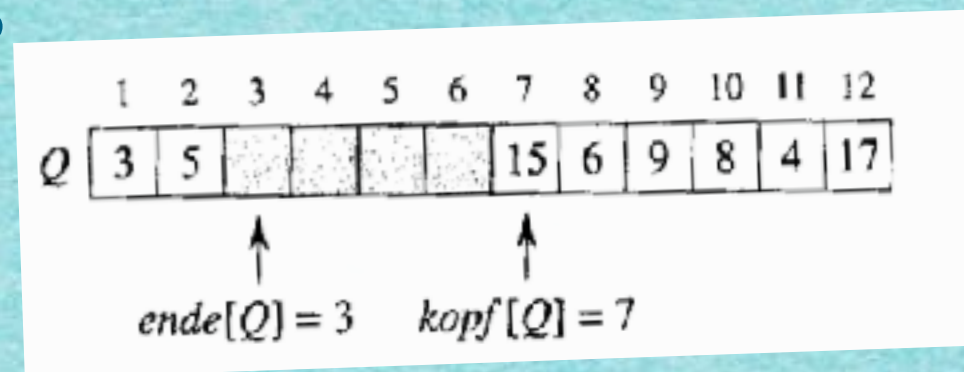
DEQUEUE:



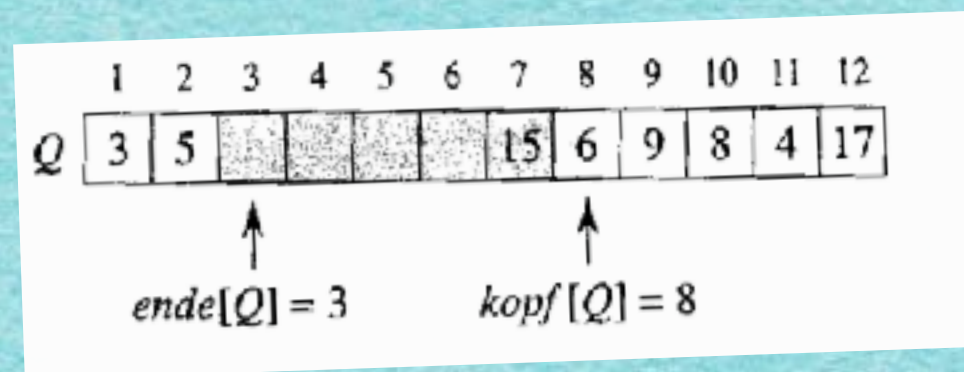
## 4.2 Stapel und Warteschlange



ENQUEUE: 17, 3, 5

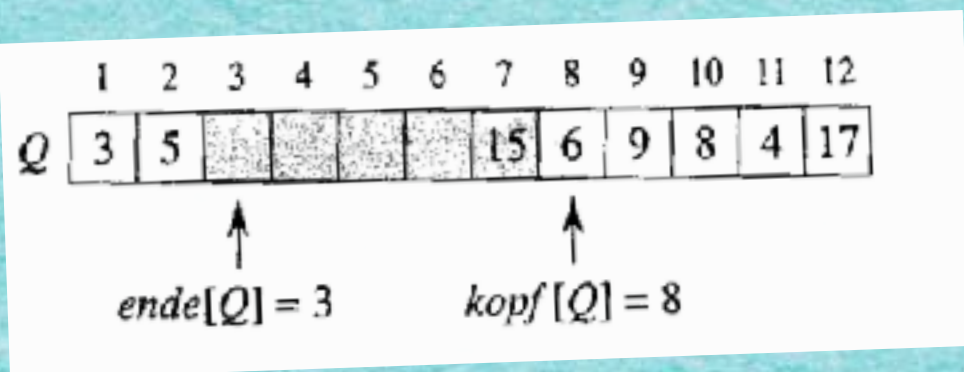
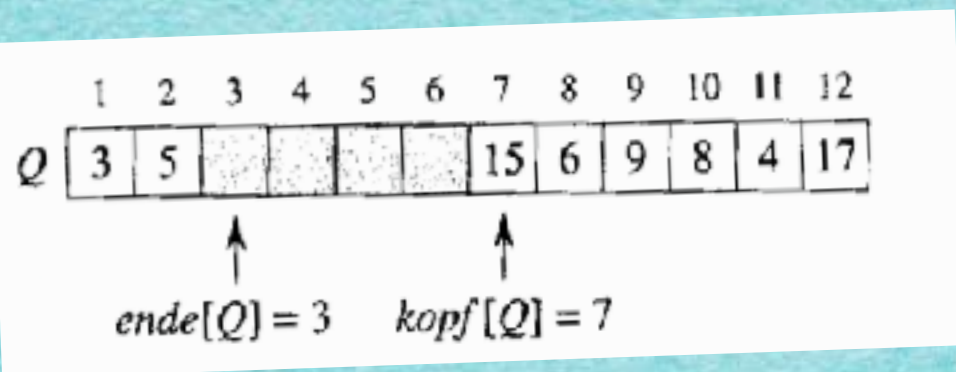
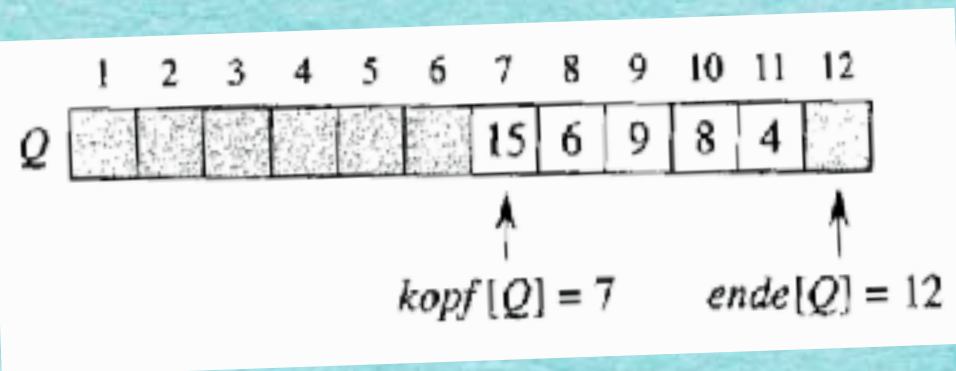


DEQUEUE:



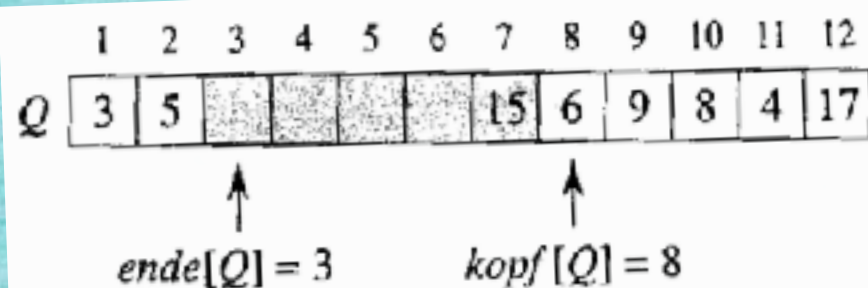
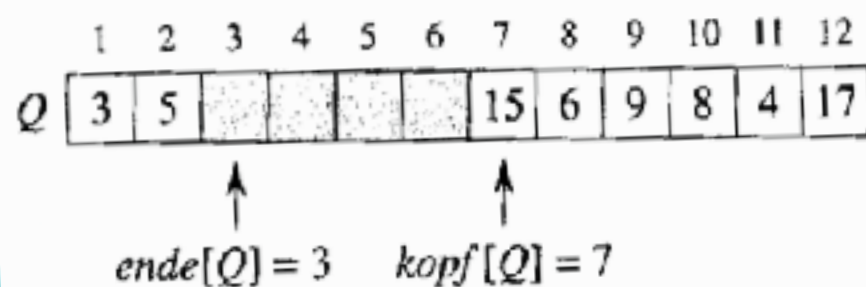
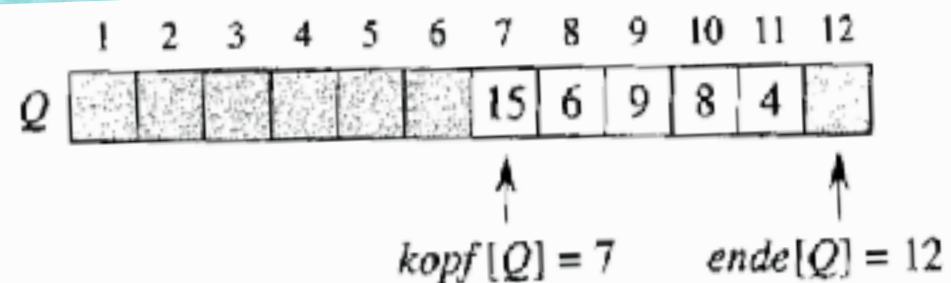


# WARTESCHLANGE AUF ARRAY UMGESETZT





# WARTESCHLANGE AUF ARRAY UMGESETZT



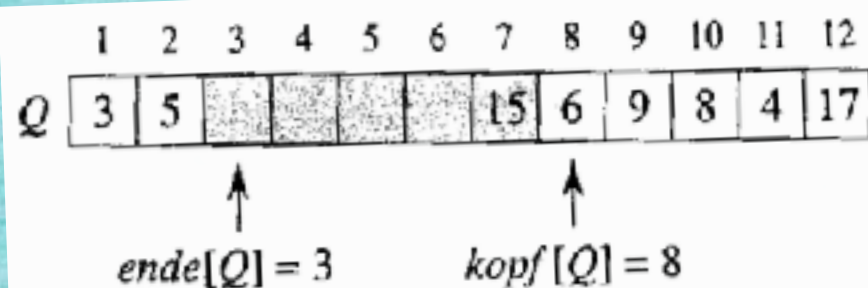
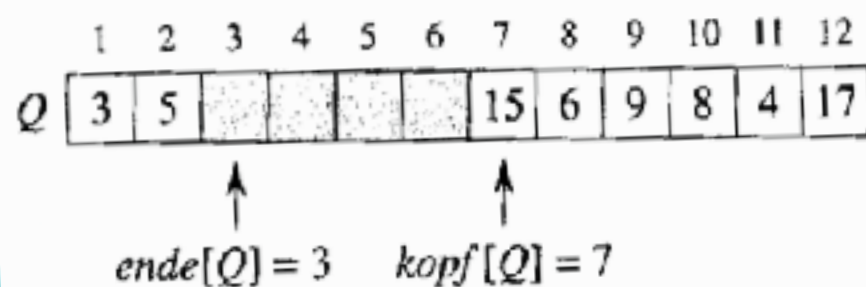
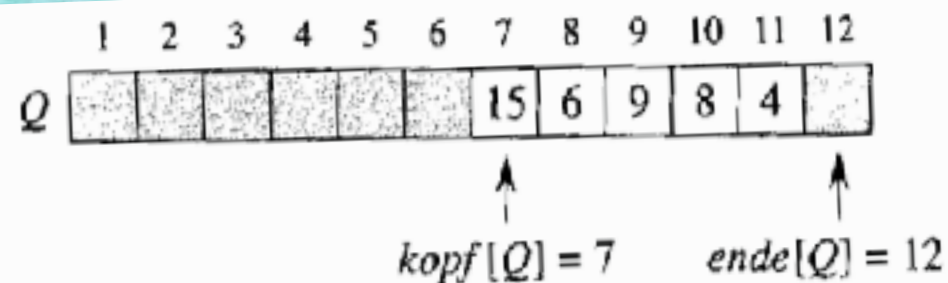
ENQUEUE(Q, x)

```

1  Q[ende[Q]] ← x
2  if ende[Q] = länge[Q]
3      then ende[Q] ← 1
4      else ende[Q] ← ende[Q] + 1
    
```



# WARTESCHLANGE AUF ARRAY UMGESETZT



ENQUEUE(Q, x)

```

1  Q[ende[Q]] ← x
2  if ende[Q] = länge[Q]
3      then ende[Q] ← 1
4      else ende[Q] ← ende[Q] + 1
    
```

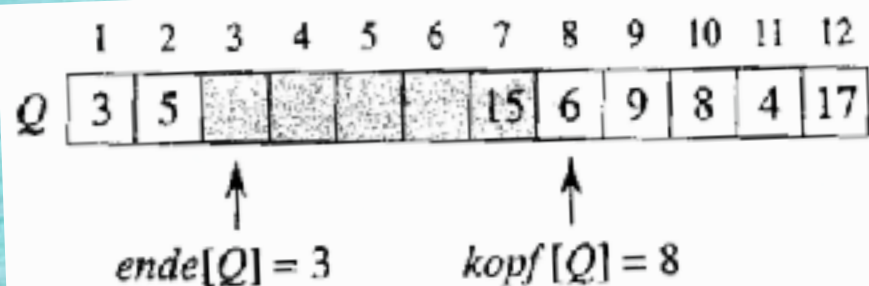
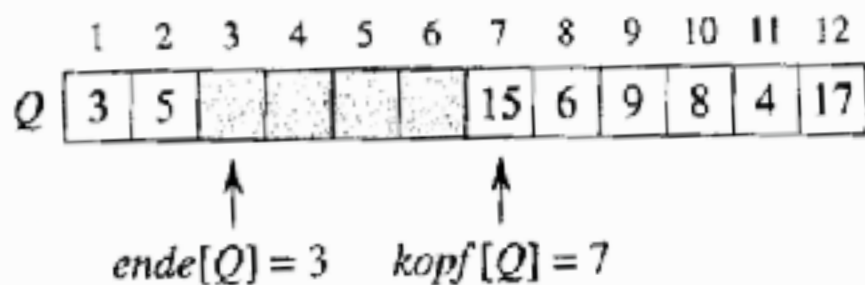
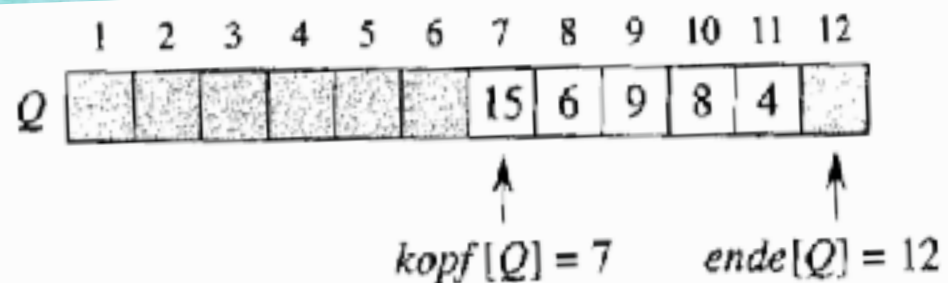
DEQUEUE(Q)

```

1  x ← Q[kopf[Q]]
2  if kopf[Q] = länge[Q]
3      then kopf[Q] ← 1
4      else kopf[Q] ← kopf[Q] + 1
5  return x
    
```



# WARTESCHLANGE AUF ARRAY UMGESETZT

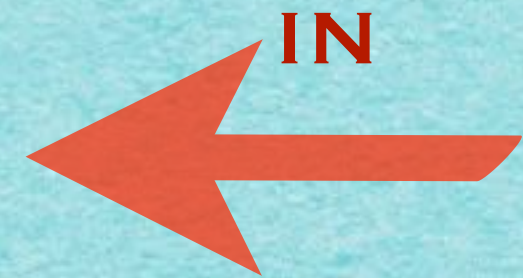




# STACK AUF ARRAY UMGESETZT



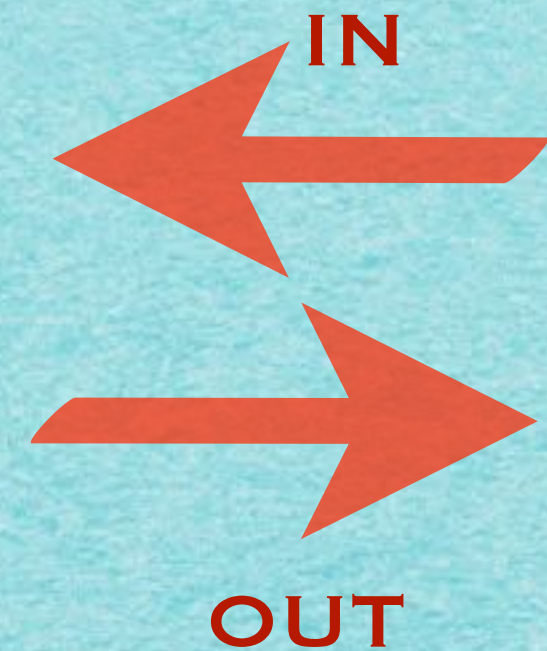
# STACK AUF ARRAY UMGESETZT





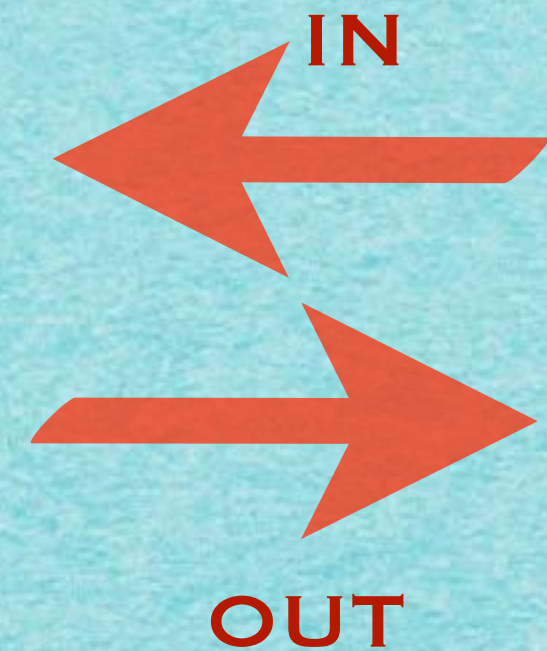
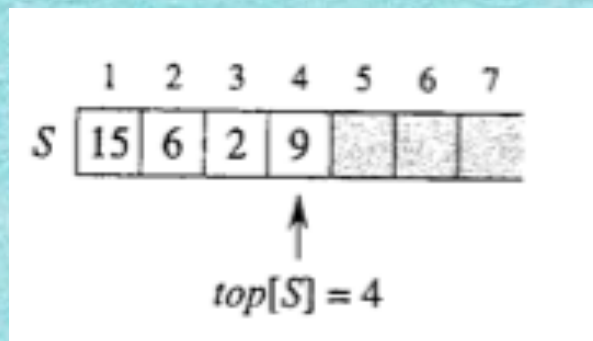
# STACK

## AUF ARRAY UMGESETZT



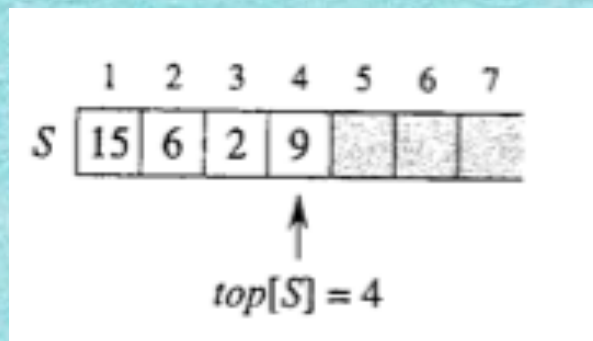


# STACK AUF ARRAY UMGESETZT

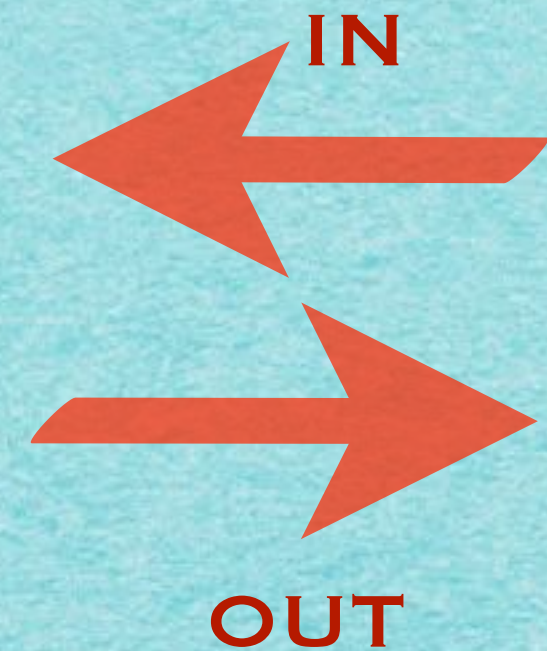




# STACK AUF ARRAY UMGESETZT

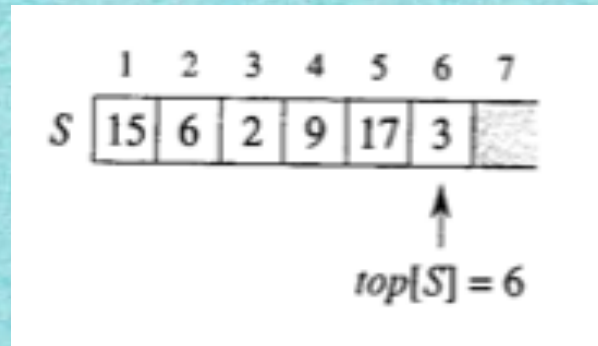
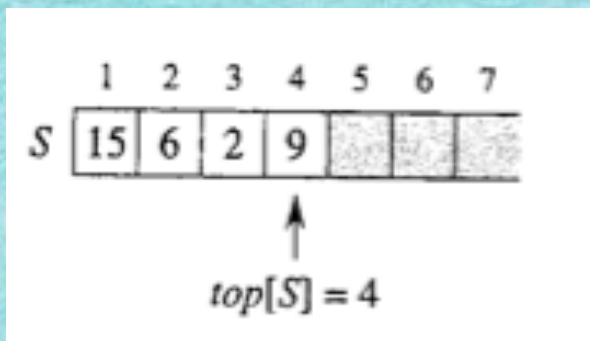


PUSH: 17, 3

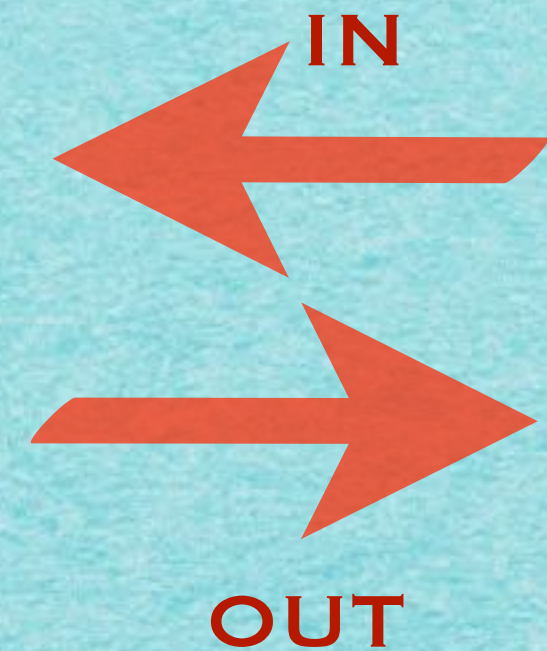




# STACK AUF ARRAY UMGESETZT



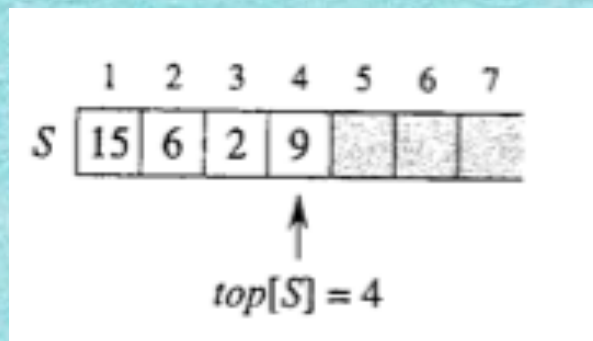
PUSH: 17, 3



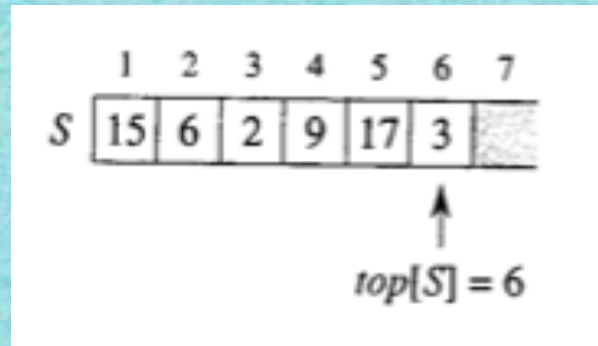


# STACK

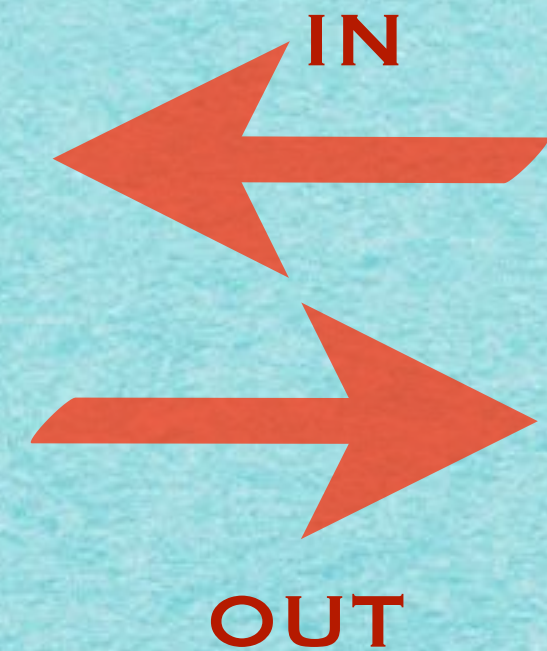
## AUF ARRAY UMGESETZT



PUSH: 17, 3



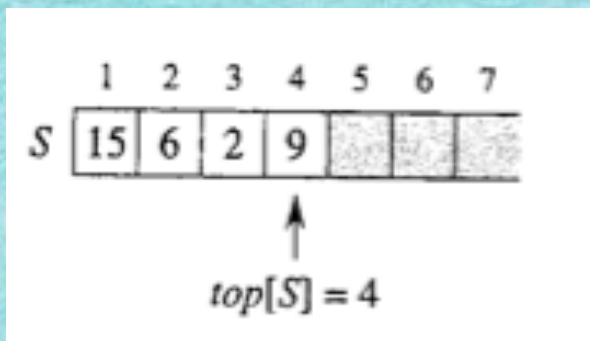
POP



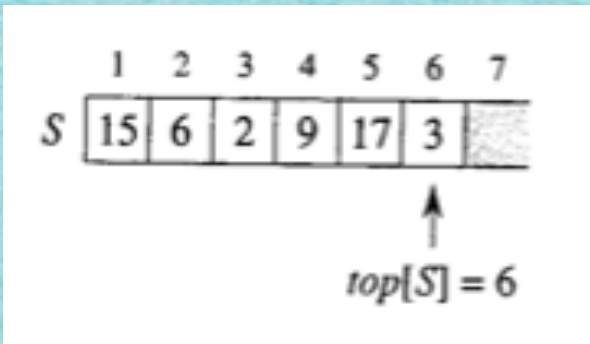


# STACK

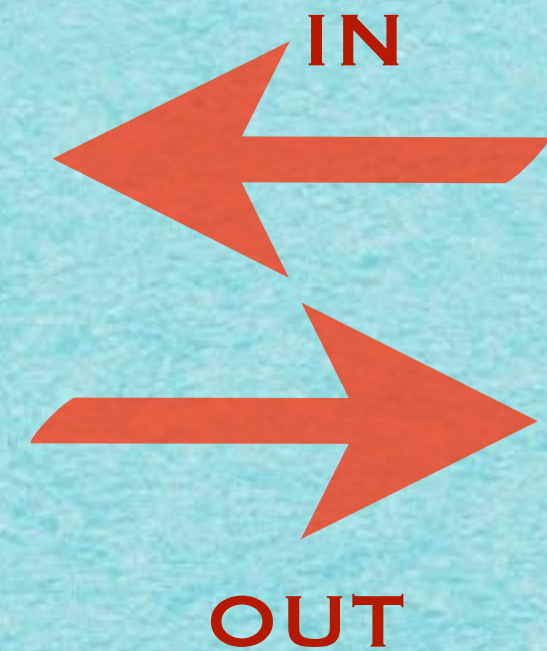
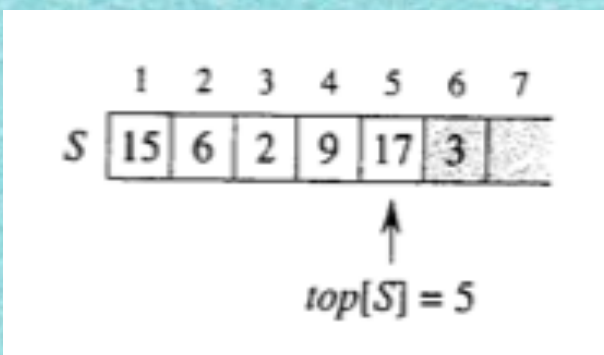
## AUF ARRAY UMGESETZT



PUSH: 17, 3



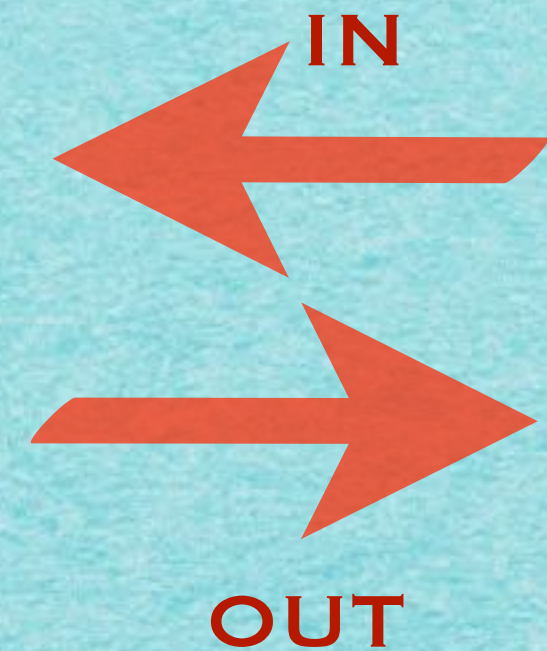
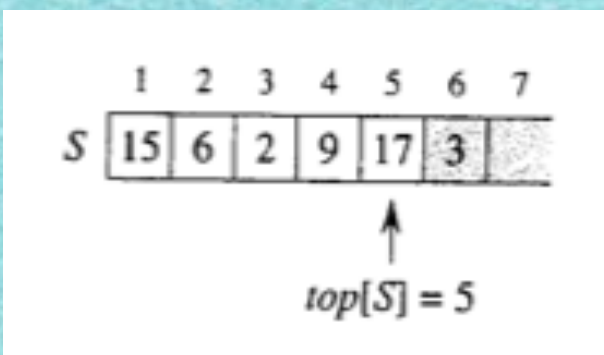
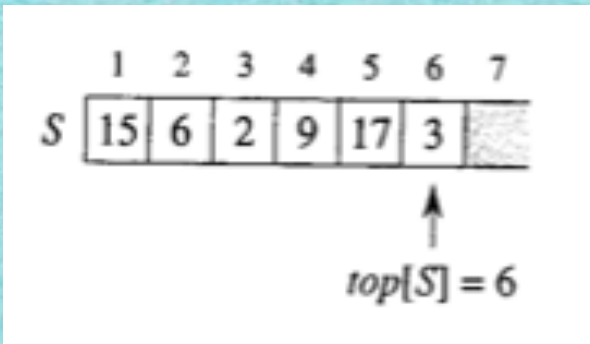
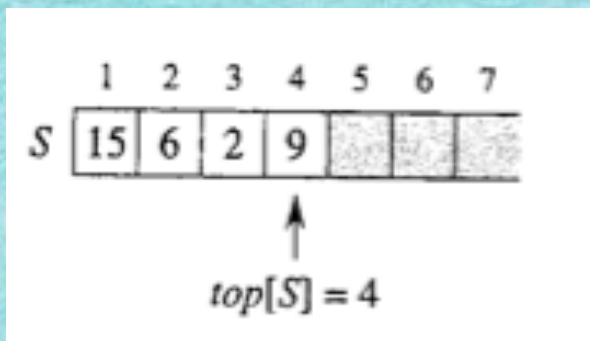
POP





# STACK

## AUF ARRAY UMGESETZT



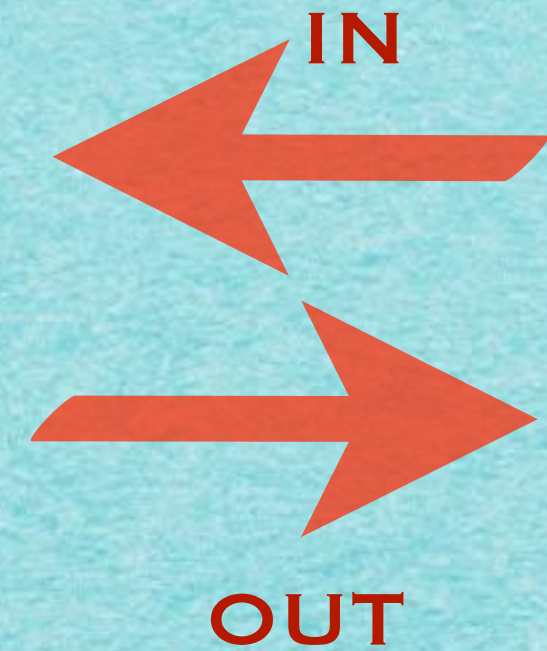
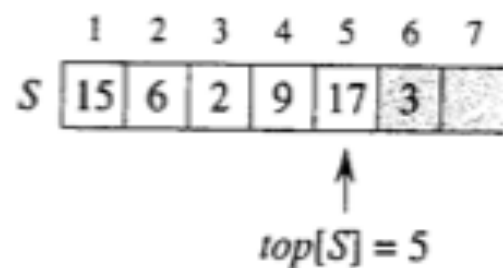
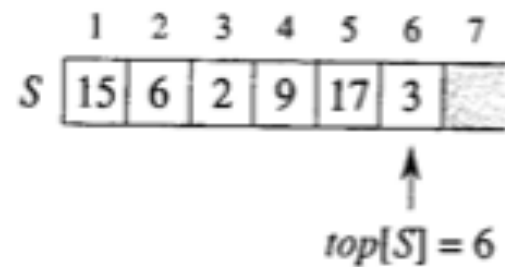


# STACK

## AUF ARRAY UMGESETZT



```
STACK-EMPTY( $S$ )  
1  if  $top[S] = 0$   
2    then return WAHR  
3    else return FALSCH
```



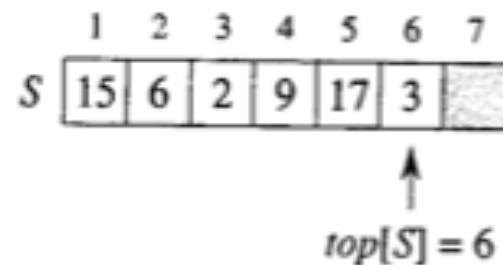


# STACK

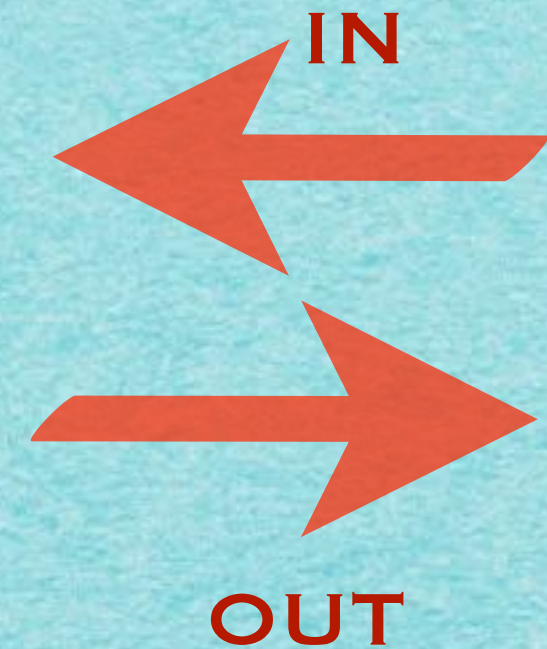
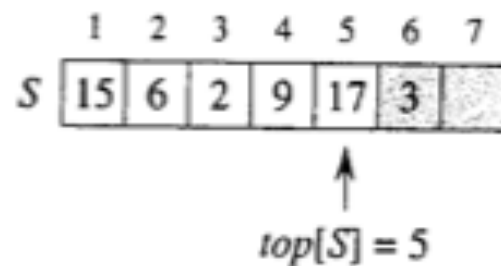
## AUF ARRAY UMGESETZT



```
STACK-EMPTY(S)  
1  if  $top[S] = 0$   
2    then return WAHR  
3    else return FALSCH
```



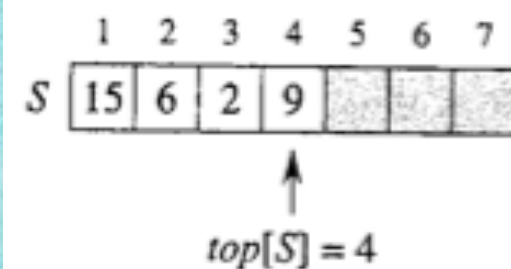
```
PUSH(S, x)  
1   $top[S] \leftarrow top[S] + 1$   
2   $S[top[S]] \leftarrow x$ 
```





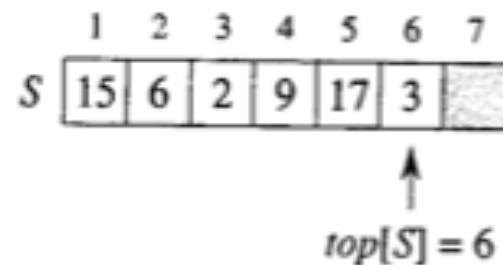
# STACK

## AUF ARRAY UMGESETZT



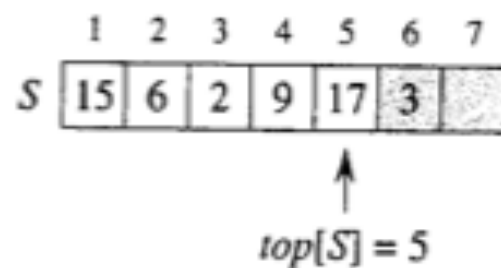
```

STACK-EMPTY(S)
1  if  $top[S] = 0$ 
2      then return WAHR
3      else return FALSCH
    
```



```

PUSH(S, x)
1   $top[S] \leftarrow top[S] + 1$ 
2   $S[top[S]] \leftarrow x$ 
    
```



```

POP(S)
1  if STACK-EMPTY(S)
2      then error "Unterlauf"
3      else  $top[S] \leftarrow top[S] - 1$ 
4            return  $S[top[S] + 1]$ 
    
```

IN

OUT



## 4.3 Verkettete Listen



## 4.3 Verkettete Listen

**Idee:**



# 4.3 Verkettete Listen

Idee:





## 4.3 Verkettete Listen

**Idee:**



**Ordne Objekte nicht explizit in aufeinanderfolgenden Speicherzellen an, sondern gib jeweils Vorgänger und Nachfolger an.**



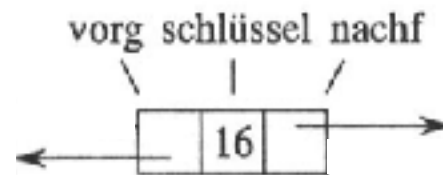




# Struktur einer doppelt verketteten Liste

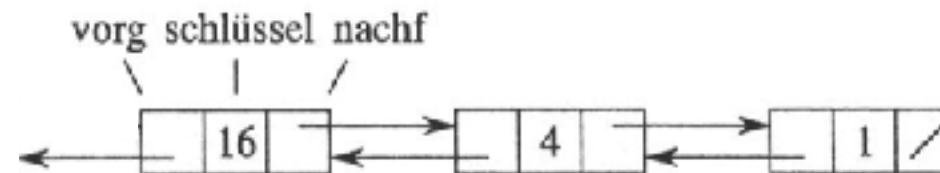


## Struktur einer doppelt verketteten Liste



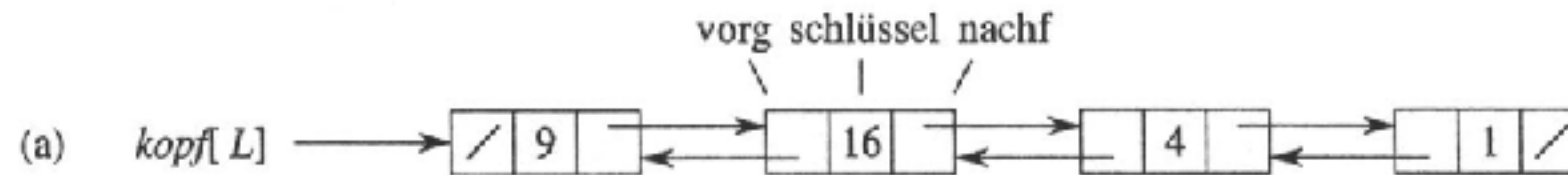


## Struktur einer doppelt verketteten Liste



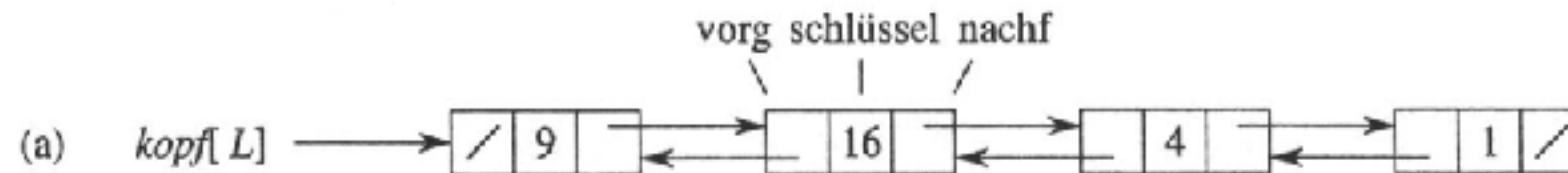


## Struktur einer doppelt verketteten Liste





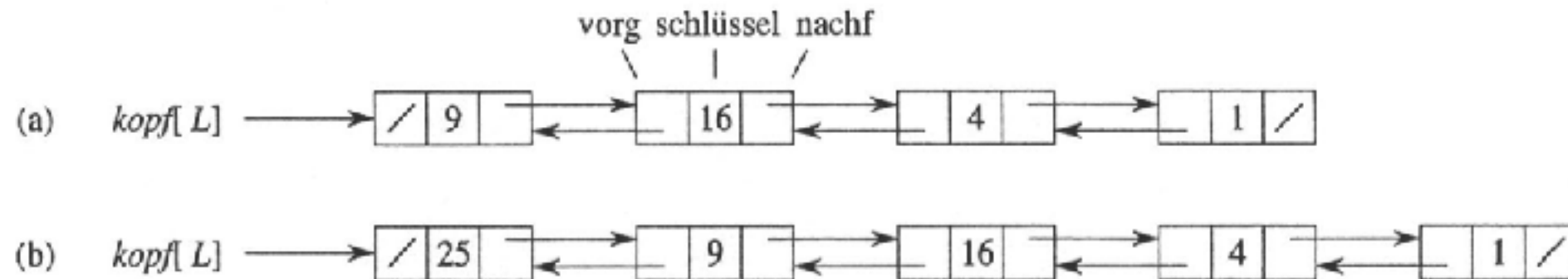
## Struktur einer doppelt verketteten Liste



- Füge vorne das Element mit Schlüssel 25 ein.



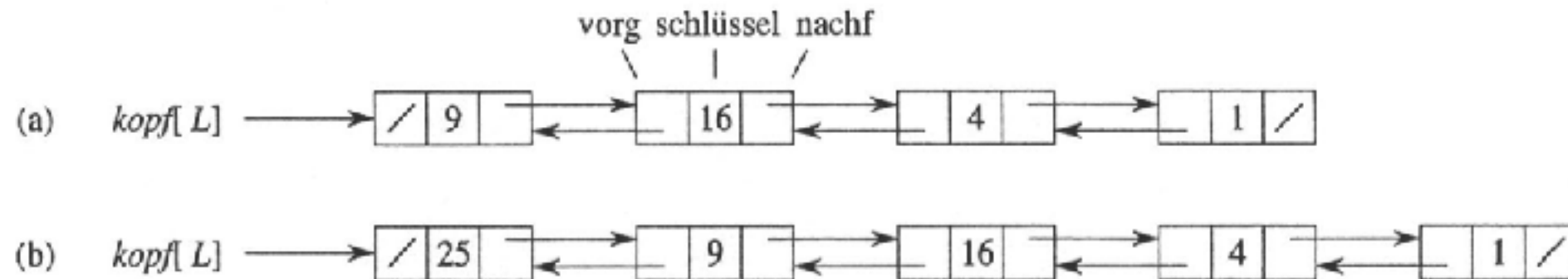
## Struktur einer doppelt verketteten Liste



- Füge vorne das Element mit Schlüssel 25 ein.



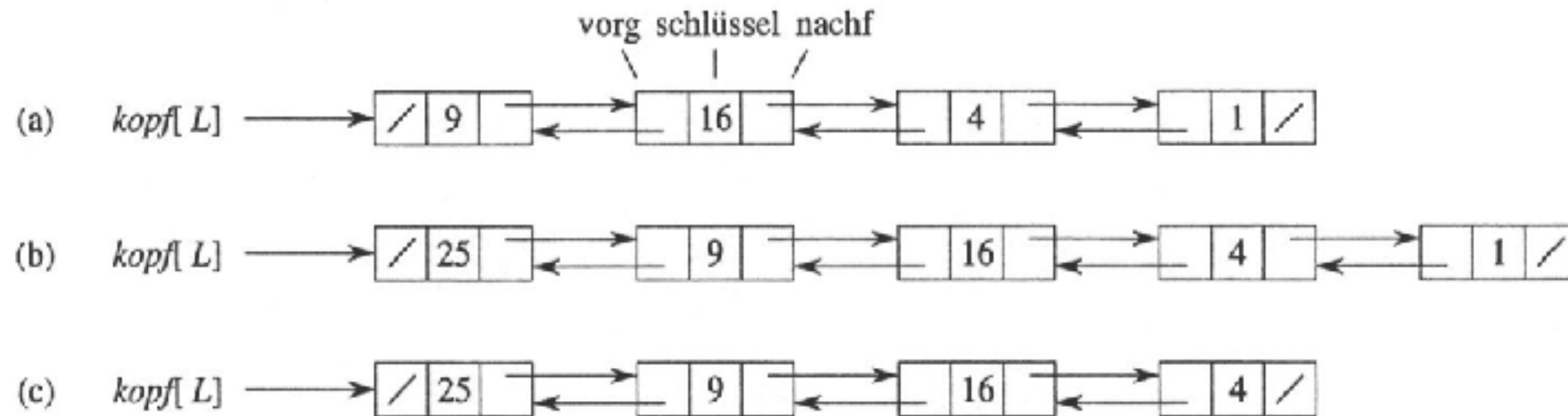
## Struktur einer doppelt verketteten Liste



- Füge vorne das Element mit Schlüssel 25 ein.
- Finde ein Element mit Schlüssel 1 und lösche es.



## Struktur einer doppelt verketteten Liste



- Füge vorne das Element mit Schlüssel 25 ein.
- Finde ein Element mit Schlüssel 1 und lösche es.



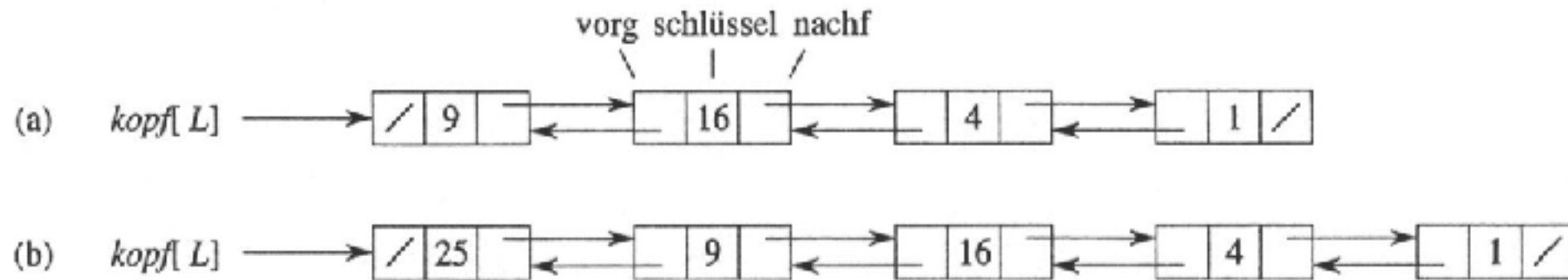




## Einfügen in eine doppelt verkettete Liste

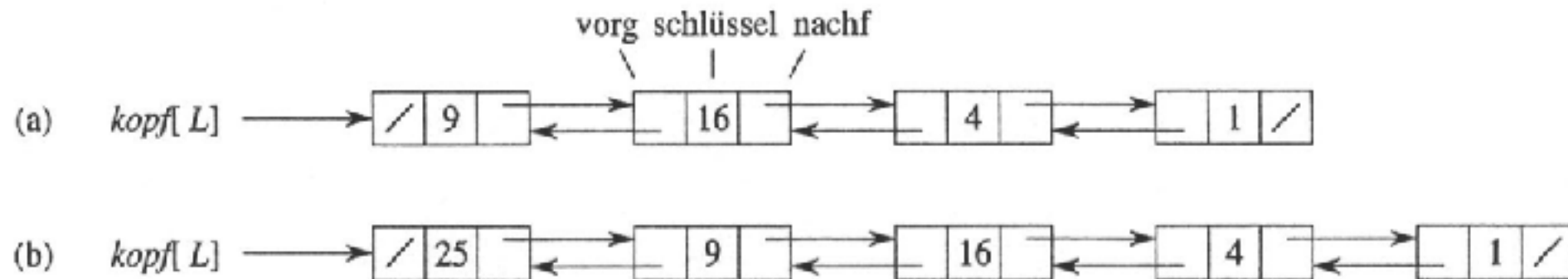


## Einfügen in eine doppelt verkettete Liste





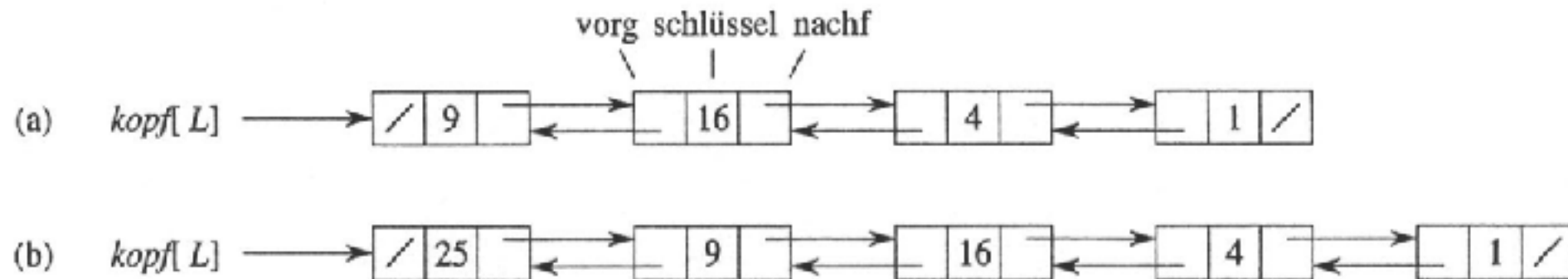
## Einfügen in eine doppelt verkettete Liste



```
LIST-INSERT( $L, x$ )  
1   $nachf[x] \leftarrow kopf[L]$   
2  if  $kopf[L] \neq NIL$   
3      then  $vorg[kopf[L]] \leftarrow x$   
4   $kopf[L] \leftarrow x$   
5   $vorg[x] \leftarrow NIL$ 
```



## Einfügen in eine doppelt verkettete Liste



```
LIST-INSERT(L, x)  
1  nachf[x] ← kopf[L]  
2  if kopf[L] ≠ NIL  
3    then vorg[kopf[L]] ← x  
4  kopf[L] ← x  
5  vorg[x] ← NIL
```

**Laufzeit:  $O(1)$**



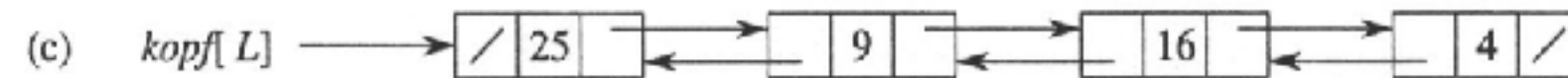




## Löschen aus einer doppelt verketteten Liste

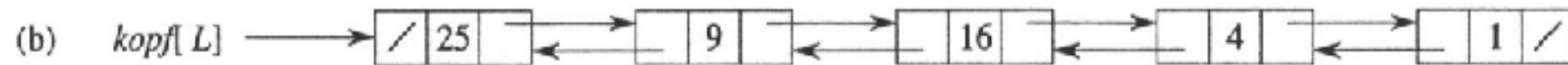


## Löschen aus einer doppelt verketteten Liste





## Löschen aus einer doppelt verketteten Liste

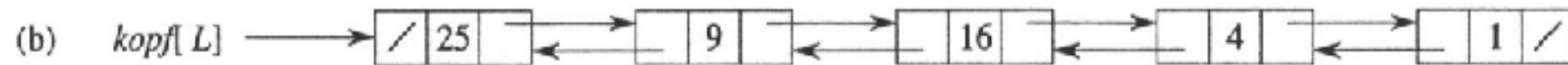


LIST-SEARCH( $L, k$ )

```
1  $x \leftarrow \text{kopf}[L]$ 
2 while  $x \neq \text{NIL}$  und  $\text{schlüssel}[x] \neq k$ 
3     do  $x \leftarrow \text{nachf}[x]$ 
4 return  $x$ 
```



## Löschen aus einer doppelt verketteten Liste



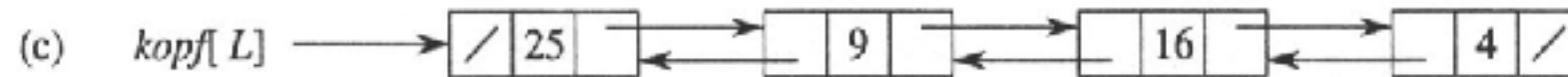
LIST-SEARCH( $L, k$ )

```
1  $x \leftarrow \text{kopf}[L]$ 
2 while  $x \neq \text{NIL}$  und  $\text{schlüssel}[x] \neq k$ 
3     do  $x \leftarrow \text{nachf}[x]$ 
4 return  $x$ 
```

**Laufzeit:  $O(n)$**



## Löschen aus einer doppelt verketteten Liste



**Laufzeit:  $O(n)$**

LIST-SEARCH( $L, k$ )

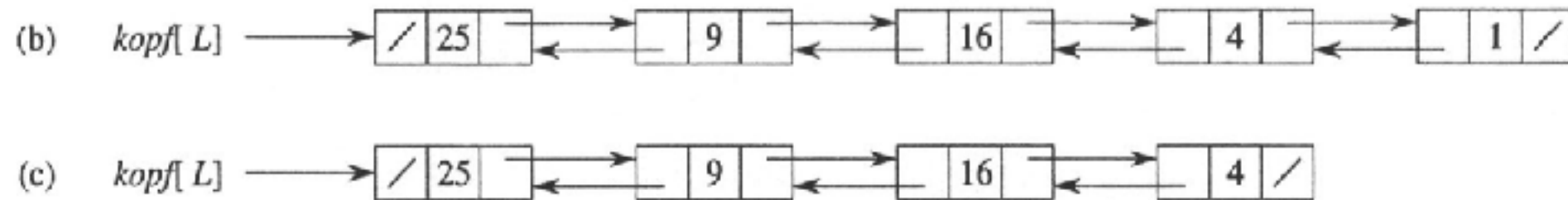
```
1  $x \leftarrow \text{kopf}[L]$ 
2 while  $x \neq \text{NIL}$  und  $\text{schlüssel}[x] \neq k$ 
3     do  $x \leftarrow \text{nachf}[x]$ 
4 return  $x$ 
```

LIST-DELETE( $L, x$ )

```
1 if  $\text{vorg}[x] \neq \text{NIL}$ 
2     then  $\text{nachf}[\text{vorg}[x]] \leftarrow \text{nachf}[x]$ 
3     else  $\text{kopf}[L] \leftarrow \text{nachf}[x]$ 
4 if  $\text{nachf}[x] \neq \text{NIL}$ 
5     then  $\text{vorg}[\text{nachf}[x]] \leftarrow \text{vorg}[x]$ 
```



## Löschen aus einer doppelt verketteten Liste



Laufzeit:  $O(n)$

LIST-SEARCH( $L, k$ )

```
1  $x \leftarrow \text{kopf}[L]$ 
2 while  $x \neq \text{NIL}$  und  $\text{schlüssel}[x] \neq k$ 
3     do  $x \leftarrow \text{nachf}[x]$ 
4 return  $x$ 
```

LIST-DELETE( $L, x$ )

```
1 if  $\text{vorg}[x] \neq \text{NIL}$ 
2     then  $\text{nachf}[\text{vorg}[x]] \leftarrow \text{nachf}[x]$ 
3     else  $\text{kopf}[L] \leftarrow \text{nachf}[x]$ 
4 if  $\text{nachf}[x] \neq \text{NIL}$ 
5     then  $\text{vorg}[\text{nachf}[x]] \leftarrow \text{vorg}[x]$ 
```

Laufzeit:  $O(1)$



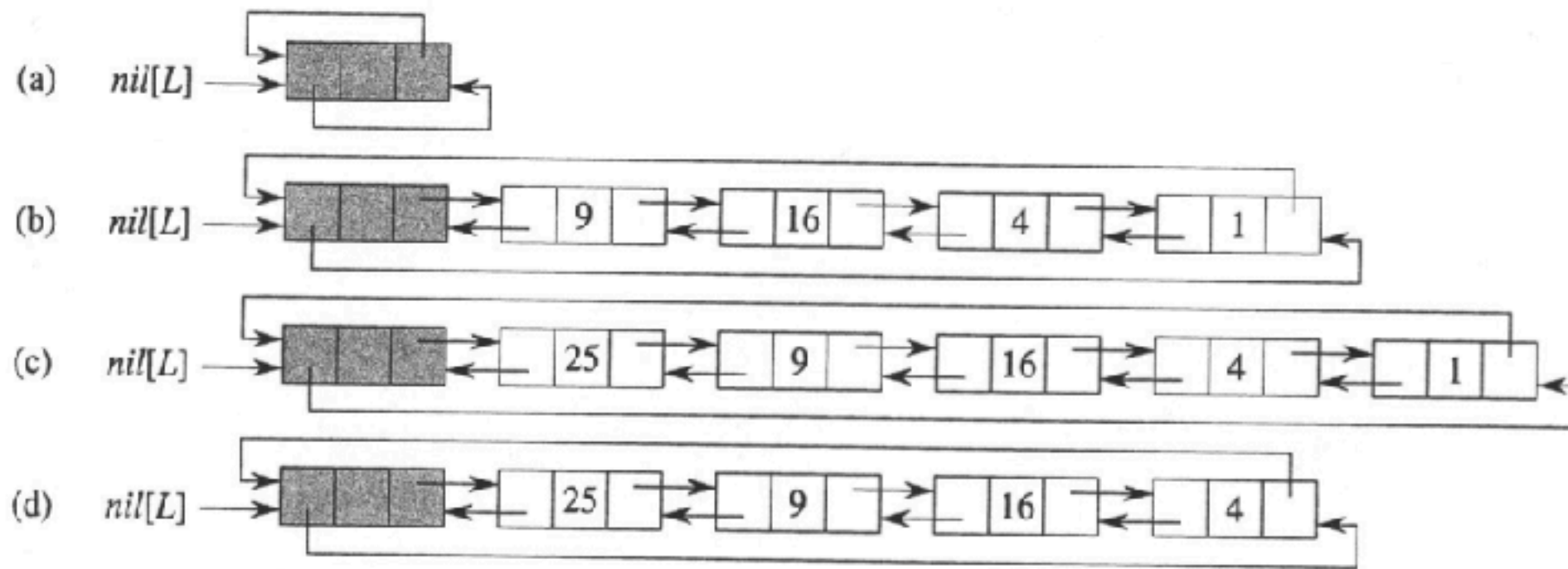




## Alternative: Zyklische Struktur mit "Wächter" nil[L]

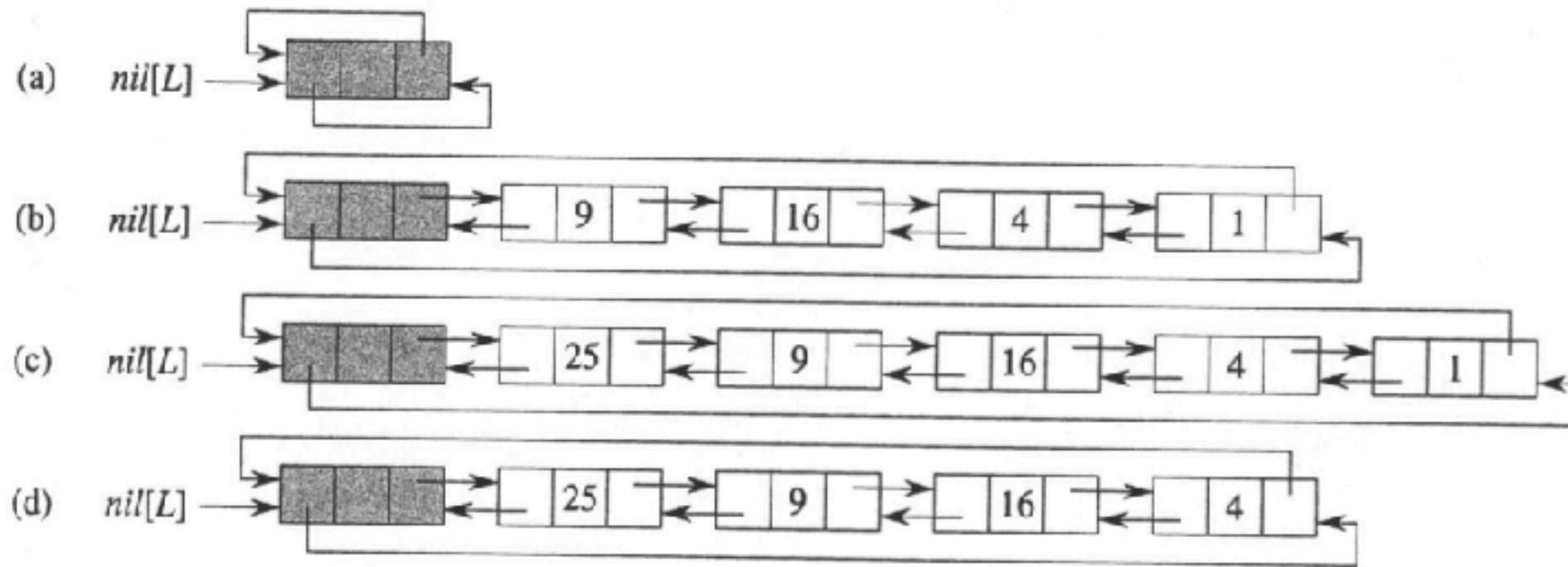


## Alternative: Zyklische Struktur mit "Wächter" nil[L]





## Alternative: Zyklische Struktur mit "Wächter" $nil[L]$



LIST-INSERT'( $L, x$ )

- 1  $nachf[x] \leftarrow nachf[nil[L]]$
- 2  $vorg[nachf[nil[L]]] \leftarrow x$
- 3  $nachf[nil[L]] \leftarrow x$
- 4  $vorg[x] \leftarrow nil[L]$



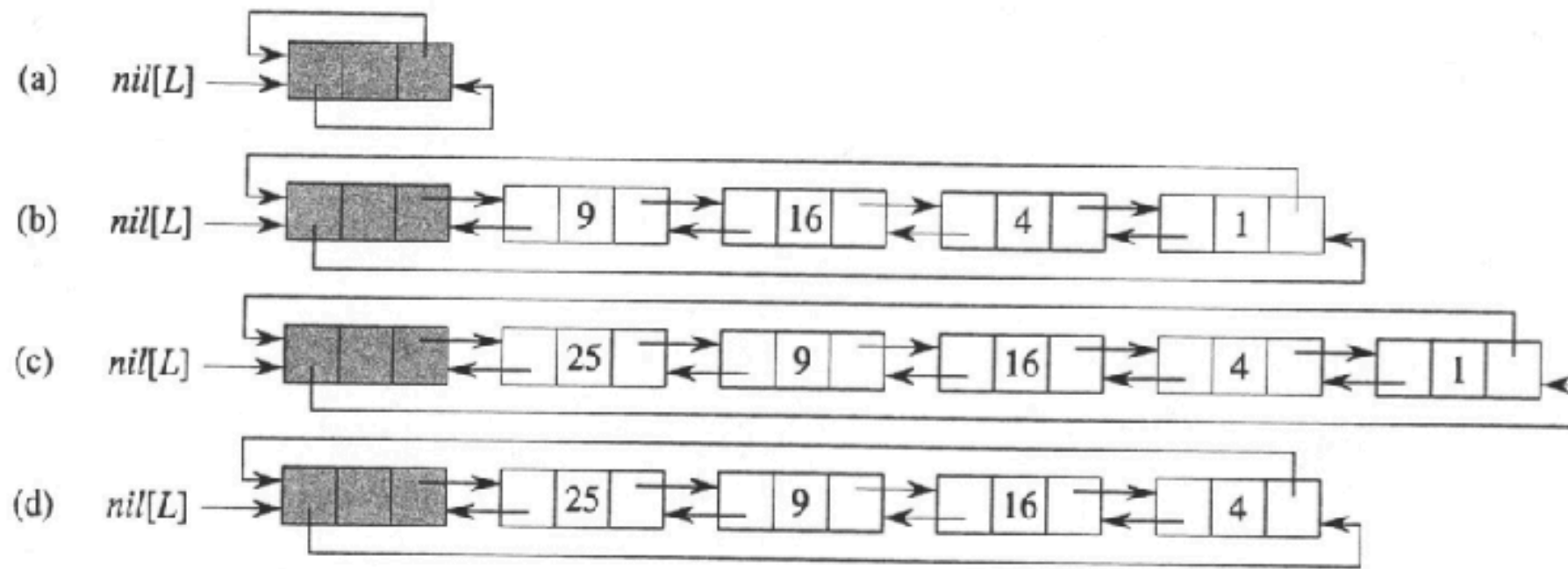




## Alternative: Zyklische Struktur mit Wächter "nil[L]"

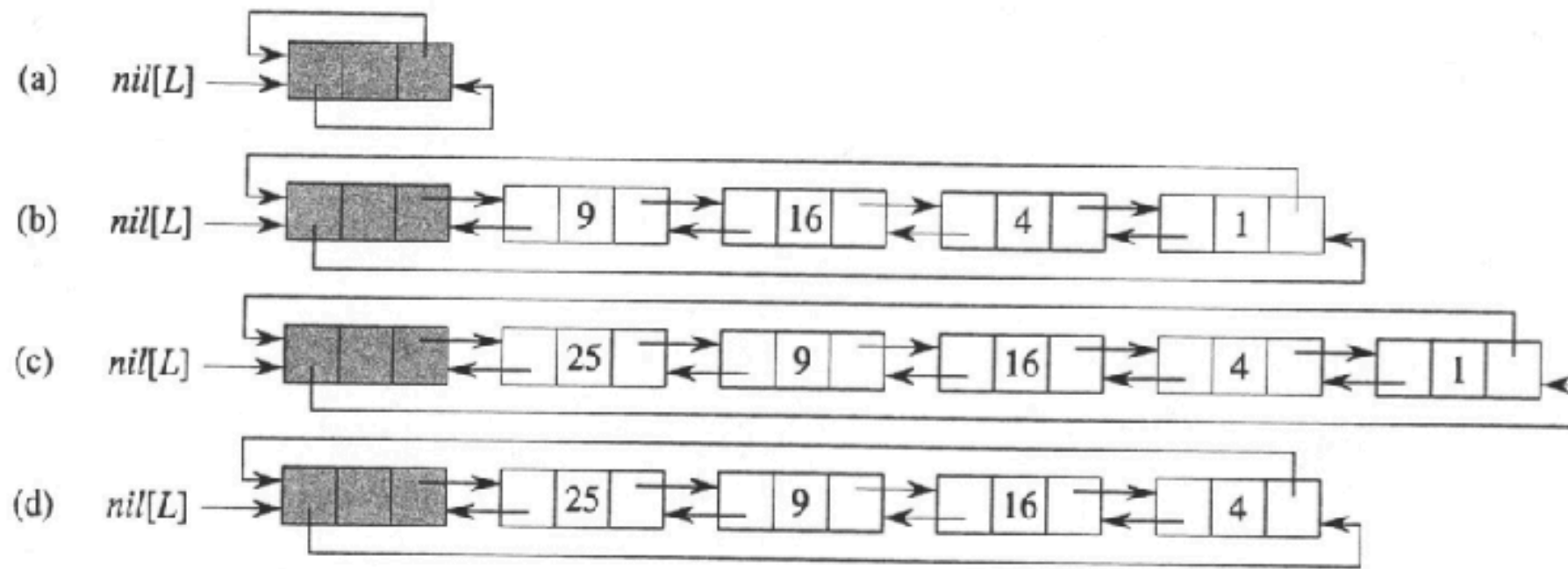


## Alternative: Zyklische Struktur mit Wächter "nil[L]"





## Alternative: Zyklische Struktur mit Wächter "nil[L]"

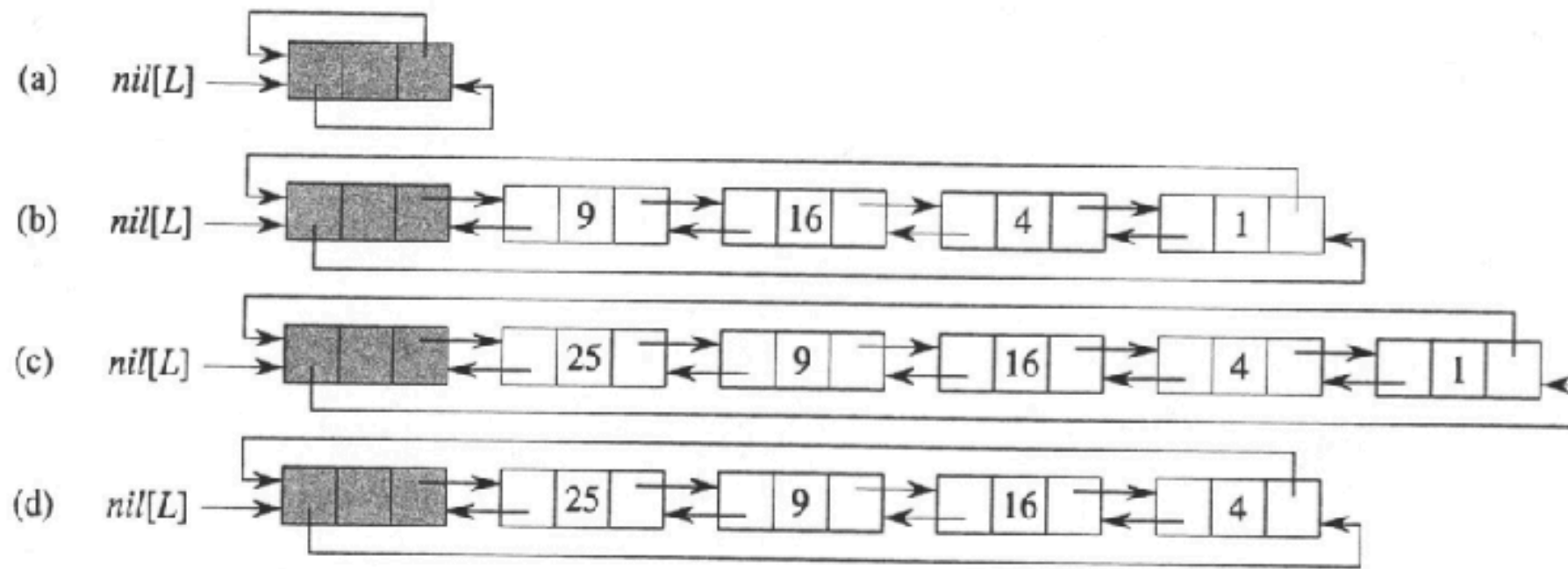


LIST-SEARCH'(L, k)

```
1   $x \leftarrow nachf[nil[L]]$   
2  while  $x \neq nil[L]$  und  $schlüssel[x] \neq k$   
3      do  $x \leftarrow nachf[x]$   
4  return x
```



## Alternative: Zyklische Struktur mit Wächter "nil[L]"



LIST-SEARCH'( $L, k$ )

```

1   $x \leftarrow nachf[nil[L]]$ 
2  while  $x \neq nil[L]$  und  $schlüssel[x] \neq k$ 
3      do  $x \leftarrow nachf[x]$ 
4  return  $x$ 

```

LIST-DELETE'( $L, x$ )

```

1   $nachf[vorg[x]] \leftarrow nachf[x]$ 
2   $vorg[nachf[x]] \leftarrow vorg[x]$ 

```



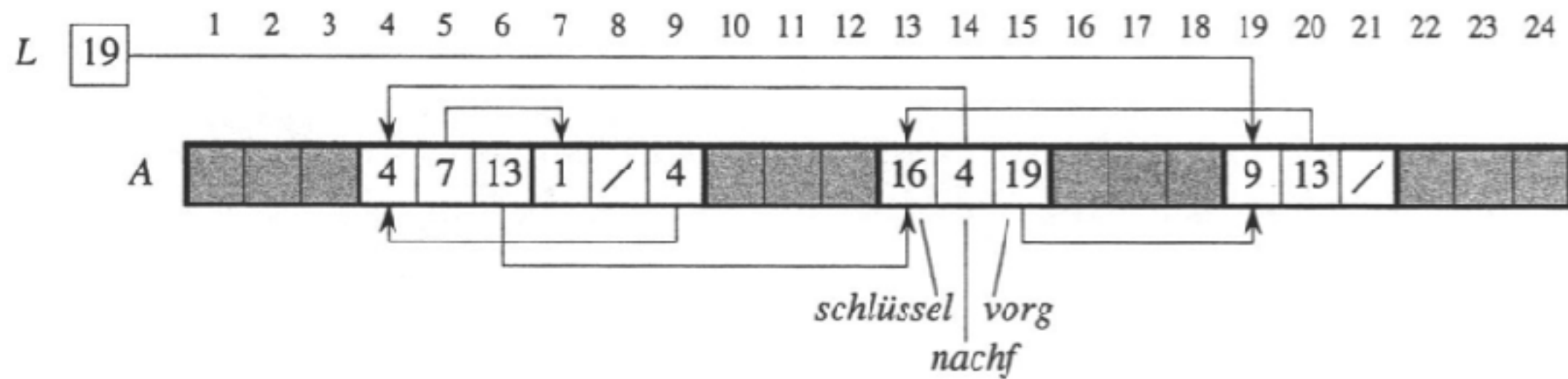




**Speicherung kann irgendwo erfolgen!**

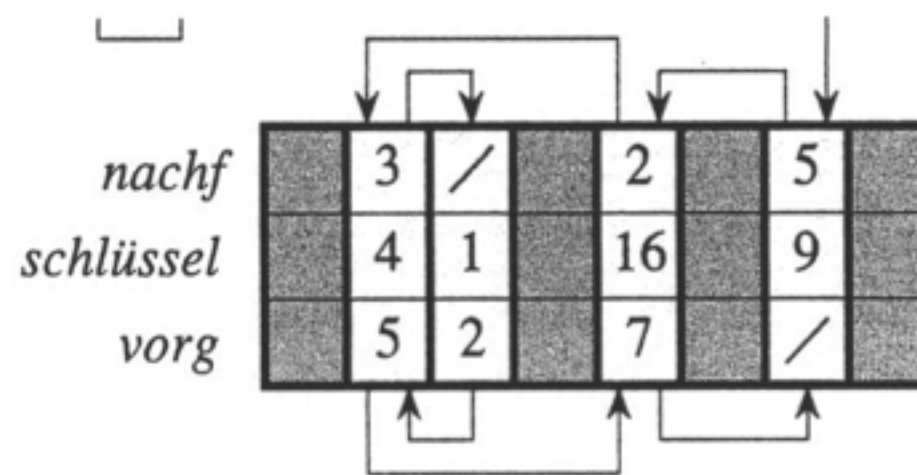
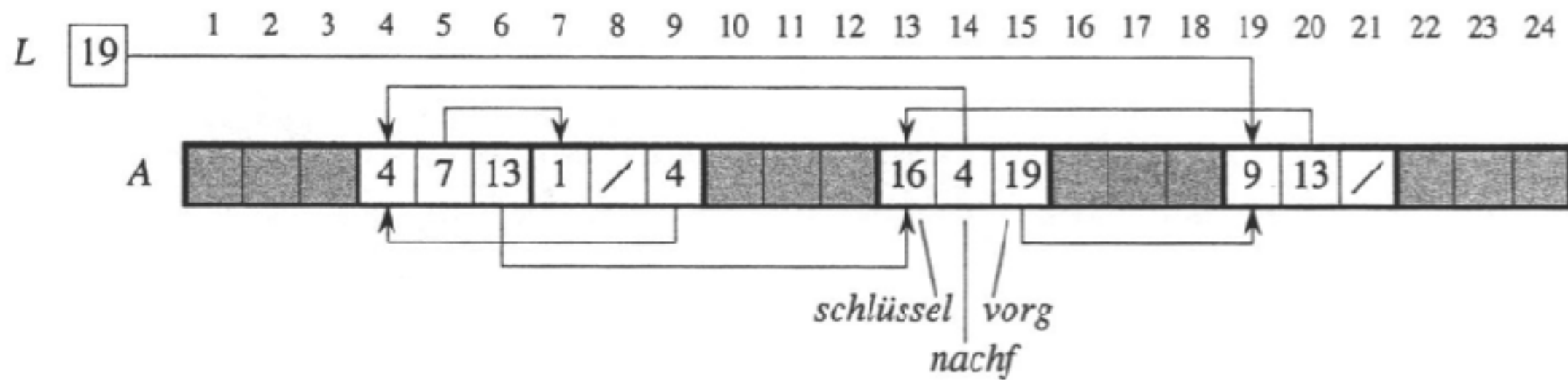


## Speicherung kann irgendwo erfolgen!



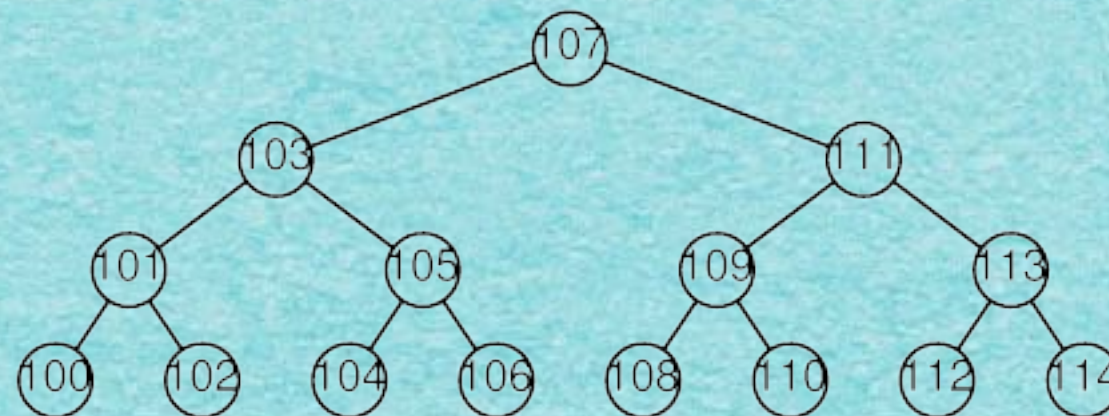


# Speicherung kann irgendwo erfolgen!





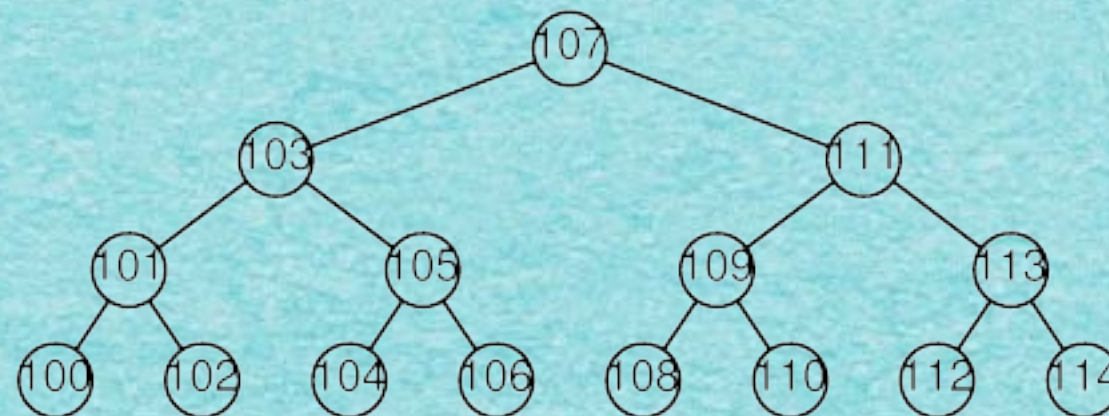
# 4.4 Binäre Suche





# 4.4 Binäre Suche

## Aufgabenstellung:

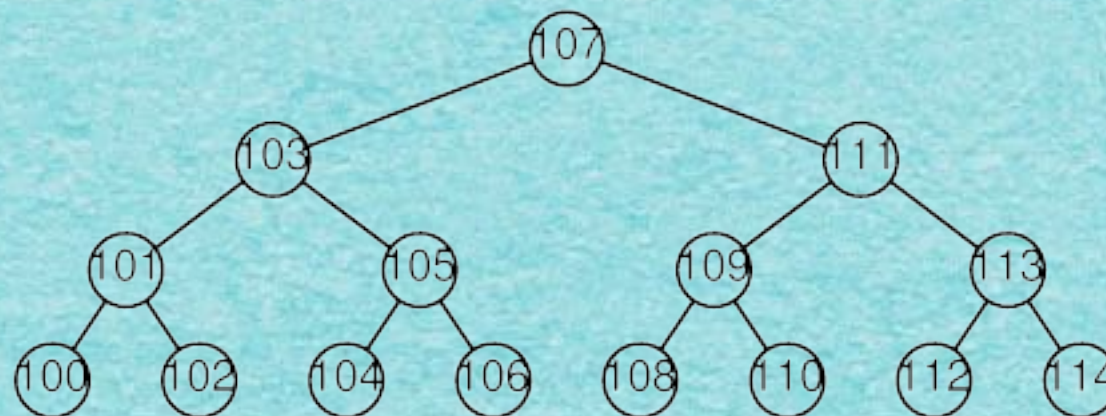




# 4.4 Binäre Suche

## Aufgabenstellung:

- *Rate eine Zahl zwischen 100 und 114!*

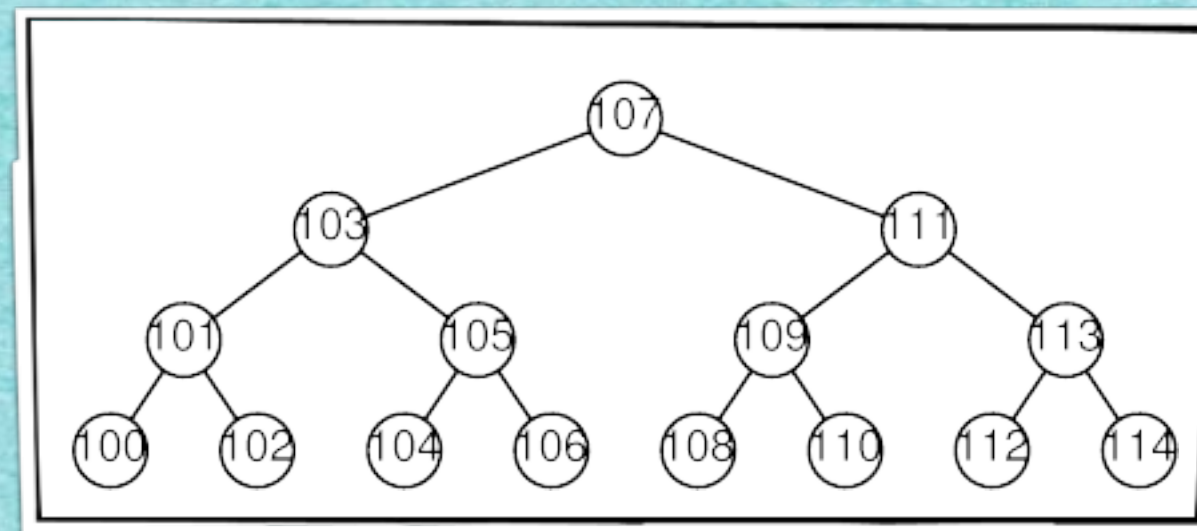




# 4.4 Binäre Suche

## Aufgabenstellung:

- *Rate eine Zahl zwischen 100 und 114!*





# Algorithmus 4.1





# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS  $\leq$  RECHTS) DO {



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS  $\leq$  RECHTS) DO {

1.1. MITTE :=  $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS  $\leq$  RECHTS) DO {

1.1. MITTE :=  $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$

1.2. IF (S[MITTE] = WERT) THEN

1.2.1. RETURN MITTE



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS ≤ RECHTS) DO {
  - 1.1. MITTE :=  $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$
  - 1.2. IF (S[MITTE] = WERT) THEN
    - 1.2.1. RETURN MITTE
  - 1.3. ELSEIF (S[MITTE] > WERT) THEN
    - 1.3.1. RECHTS := MITTE - 1}



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS ≤ RECHTS) DO {
  - 1.1. MITTE :=  $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$
  - 1.2. IF (S[MITTE] = WERT) THEN
    - 1.2.1. RETURN MITTE
  - 1.3. ELSEIF (S[MITTE] > WERT) THEN
    - 1.3.1. RECHTS := MITTE - 1
  - 1.4. ELSEIF
    - 1.4.1. LINKS := MITTE + 1}



# Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE :=  $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$   
    1.2. IF (S[MITTE] = WERT) THEN  
        1.2.1. RETURN MITTE  
    1.3. ELSEIF (S[MITTE] > WERT) THEN  
        1.3.1. RECHTS := MITTE - 1  
    1.4. ELSEIF  
        1.4.1. LINKS := MITTE + 1  
}
```



# Algorithmus 4.1

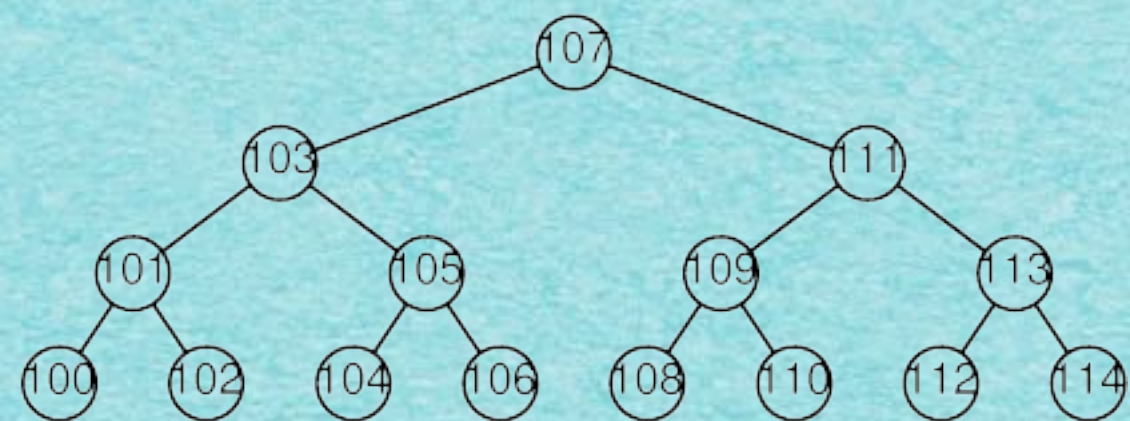
INPUT: Sortierter Array mit Einträgen  $S[I]$ , Suchwert WERT,  
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

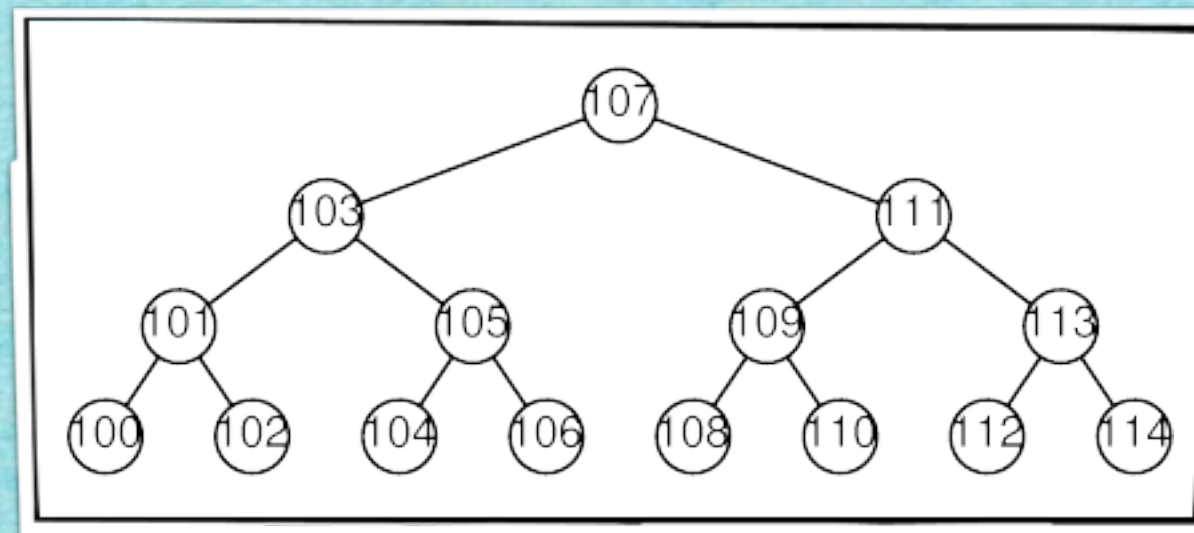
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS ≤ RECHTS) DO {
  - 1.1. MITTE :=  $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$
  - 1.2. IF (S[MITTE] = WERT) THEN
    - 1.2.1. RETURN MITTE
  - 1.3. ELSEIF (S[MITTE] > WERT) THEN
    - 1.3.1. RECHTS := MITTE - 1
  - 1.4. ELSEIF
    - 1.4.1. LINKS := MITTE + 1}
2. RETURN "WERT nicht gefunden!"











## 4.4 Binäre Suche



## 4.4 Binäre Suche

### Satz 4.2



## 4.4 Binäre Suche

### **Satz 4.2**

*Die binäre Suche terminiert in  $O(\log(\text{RECHTS-LINKS}))$  Schritten*



## 4.4 Binäre Suche

### **Satz 4.2**

*Die binäre Suche terminiert in  $O(\log(\text{RECHTS}-\text{LINKS}))$  Schritten (für  $\text{RECHTS} > \text{LINKS}$ ).*



## 4.4 Binäre Suche

### **Satz 4.2**

*Die binäre Suche terminiert in  $O(\log(\text{RECHTS}-\text{LINKS}))$  Schritten (für  $\text{RECHTS} > \text{LINKS}$ ).*

### **Beweis:**

**Selbst!**



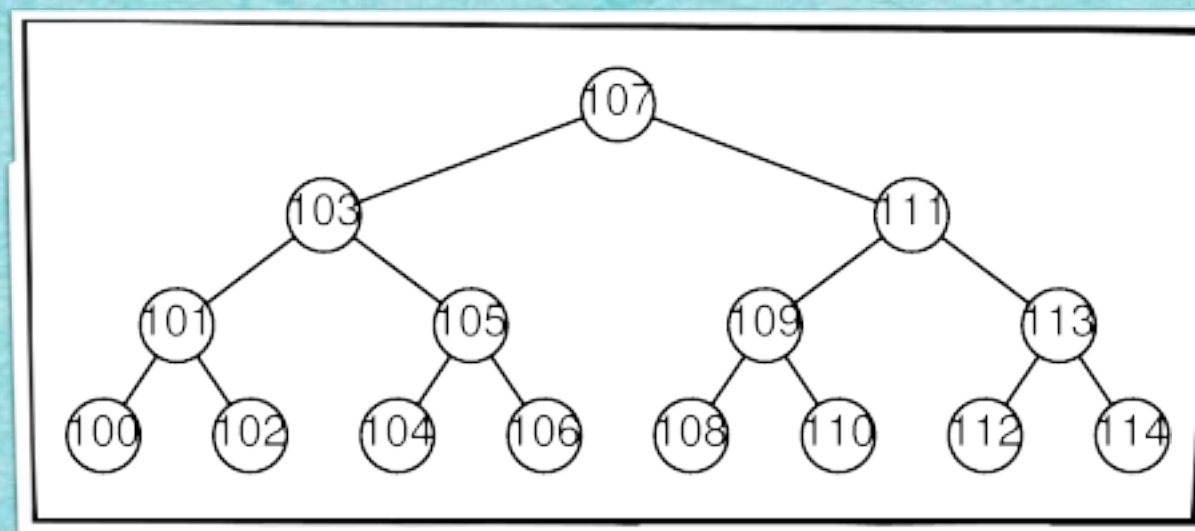
## 4.4 Binäre Suche

### Satz 4.2

*Die binäre Suche terminiert in  $O(\log(\text{RECHTS-LINKS}))$  Schritten (für  $\text{RECHTS} > \text{LINKS}$ ).*

### Beweis:

Selbst!

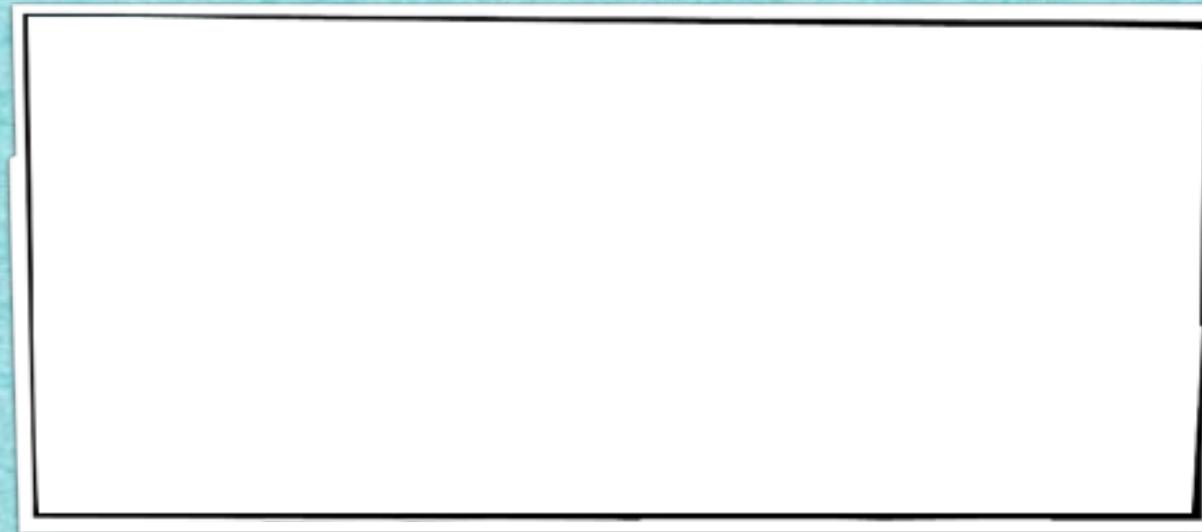




# 4.5 Binäre Suchbäume

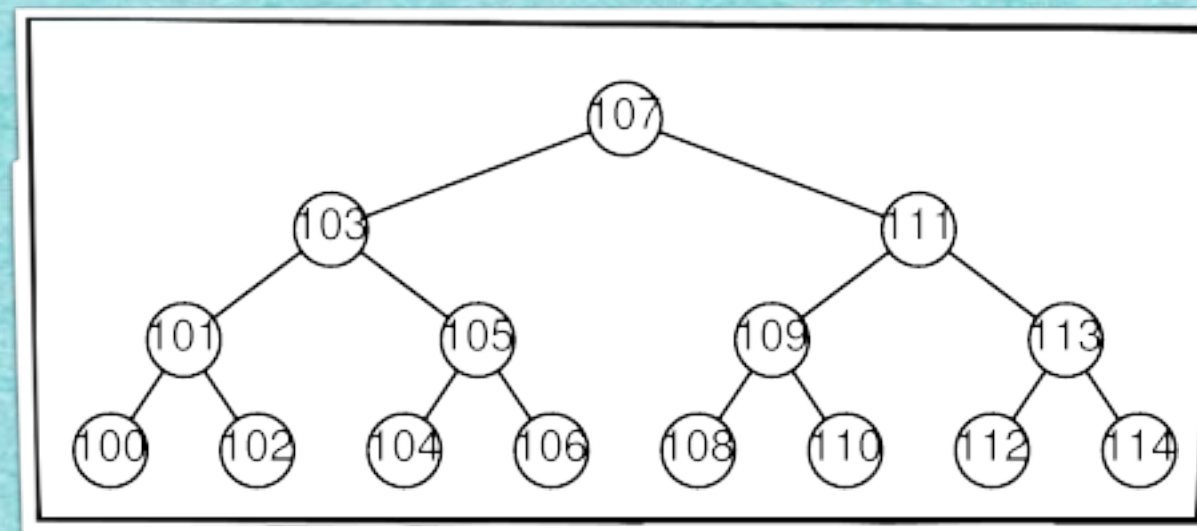


# 4.5 Binäre Suchbäume





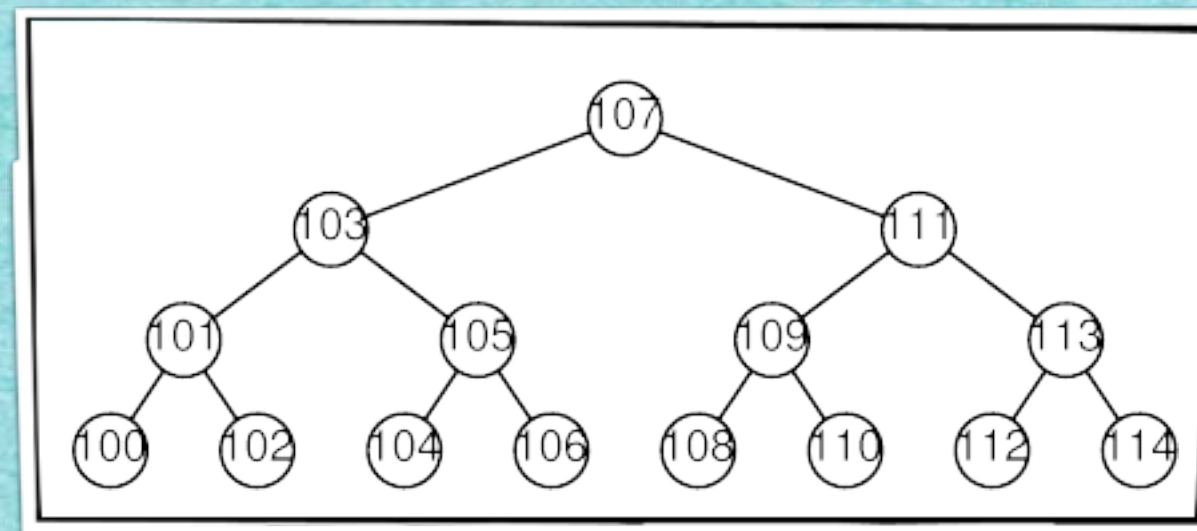
# 4.5 Binäre Suchbäume





# 4.5 Binäre Suchbäume

Ideen:

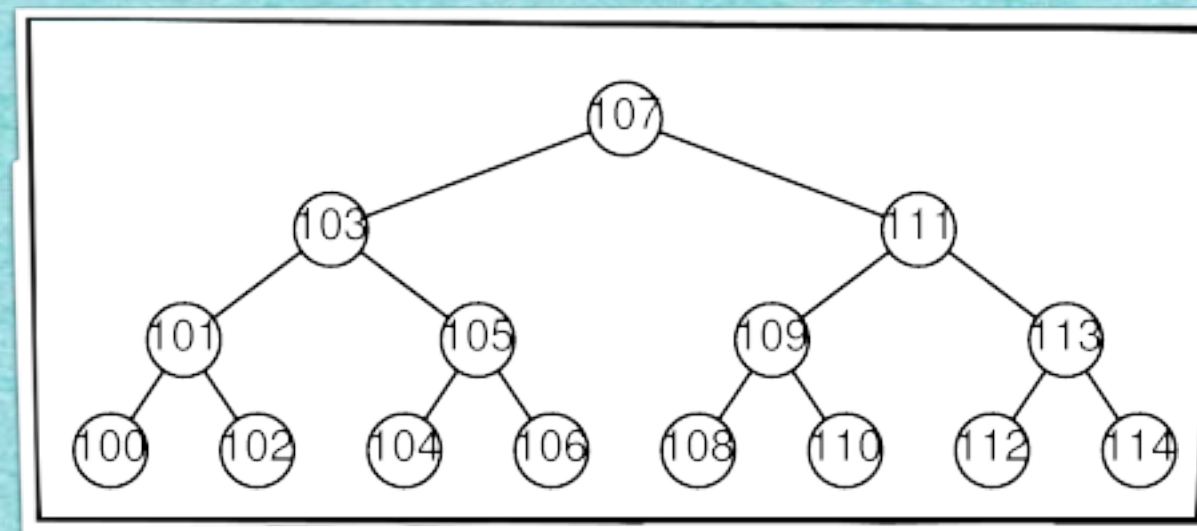




# 4.5 Binäre Suchbäume

## Ideen:

- *Strukturiere Daten wie im möglichen Ablauf einer binären Suche!*

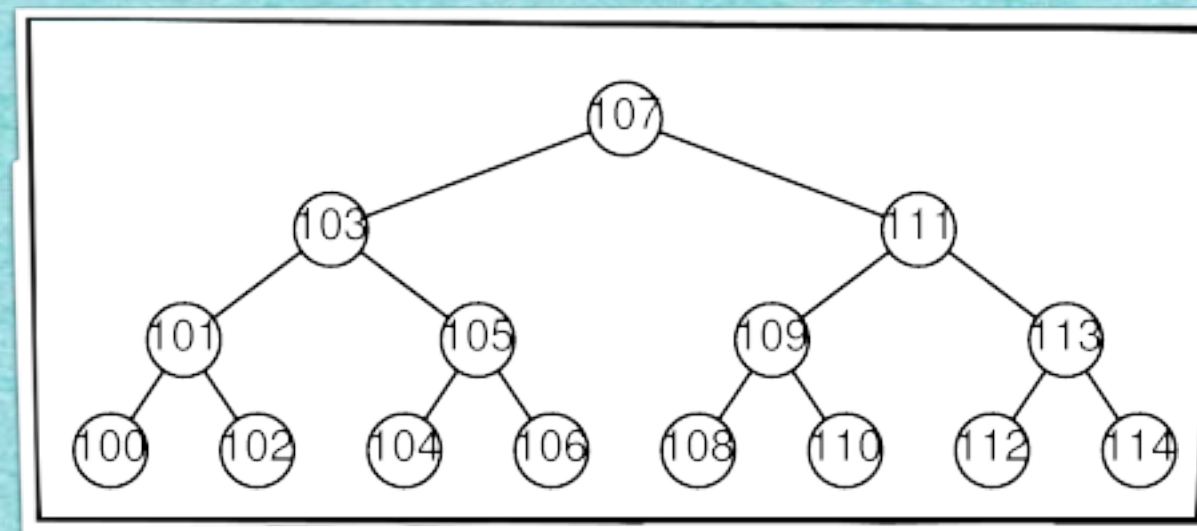




# 4.5 Binäre Suchbäume

## Ideen:

- *Strukturiere Daten wie im möglichen Ablauf einer binären Suche!*
- *Erziele logarithmische Zeiten!*





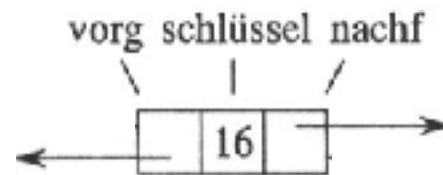




# Struktur einer doppelt verketteten Liste

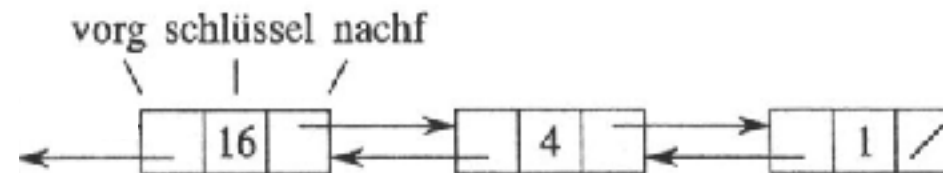


## Struktur einer doppelt verketteten Liste



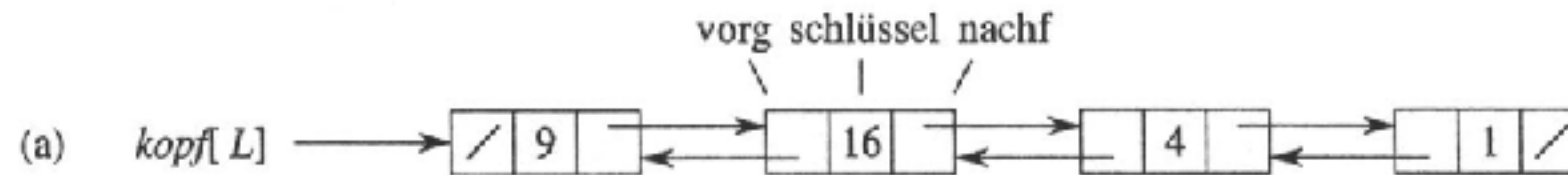


## Struktur einer doppelt verketteten Liste



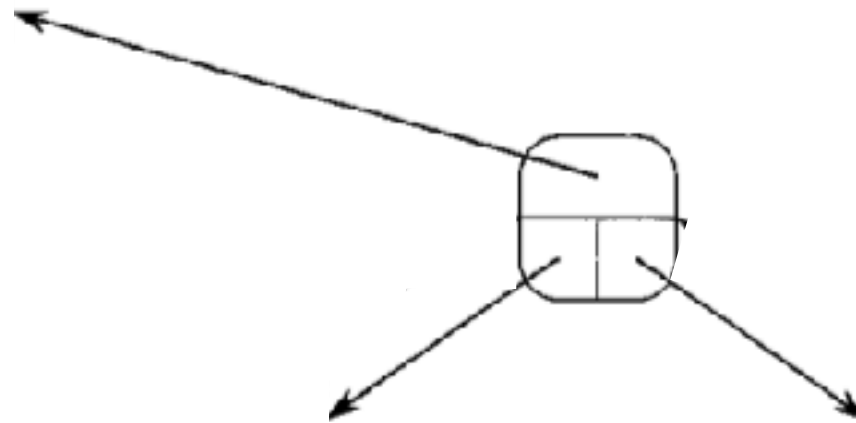


## Struktur einer doppelt verketteten Liste



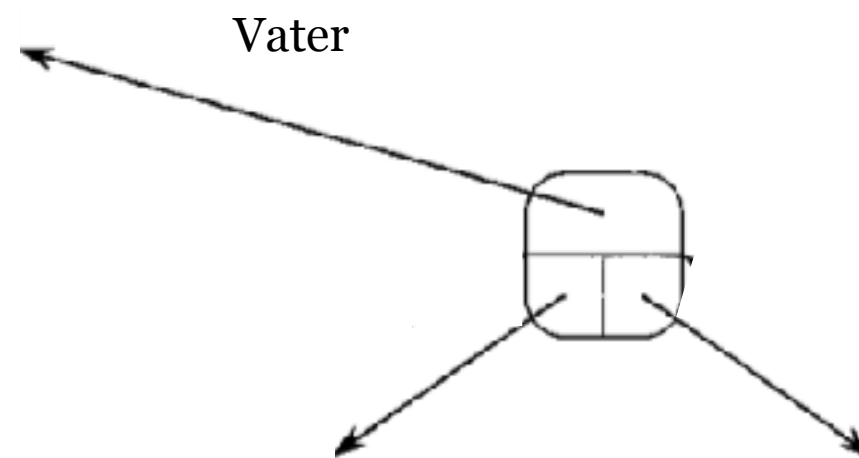


# Binärer Suchbaum



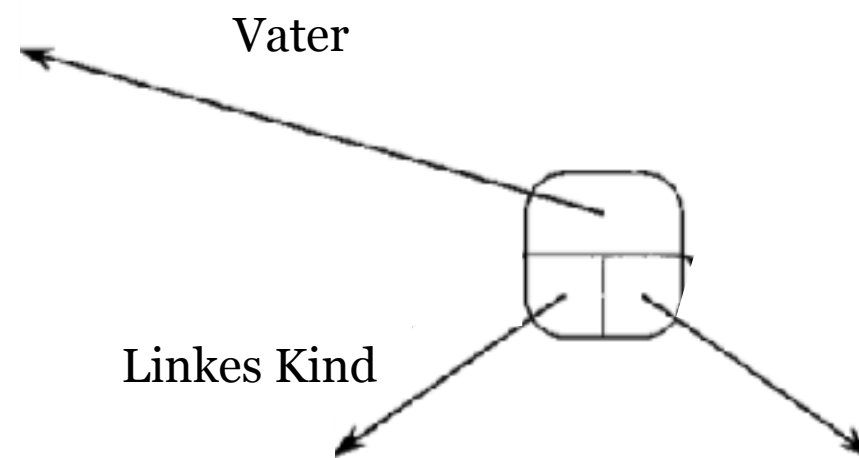


# Binärer Suchbaum



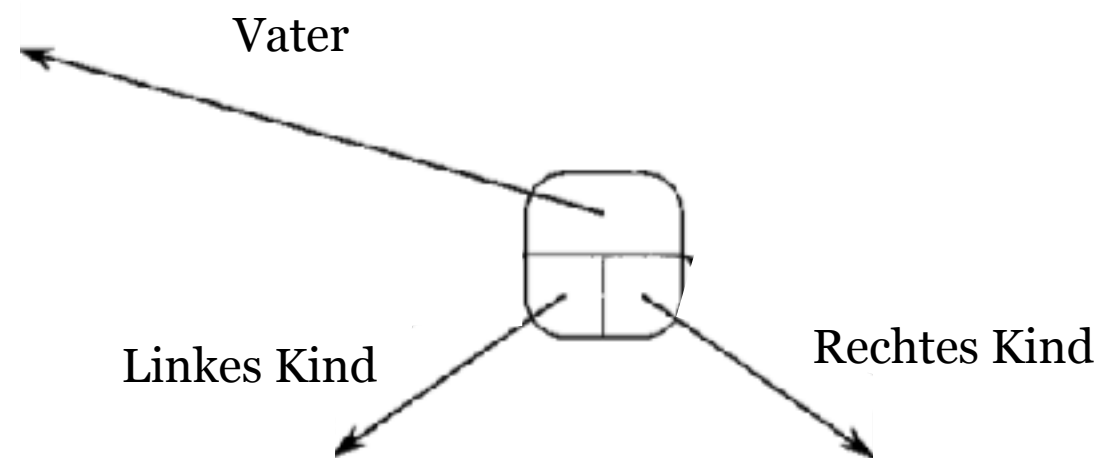


# Binärer Suchbaum



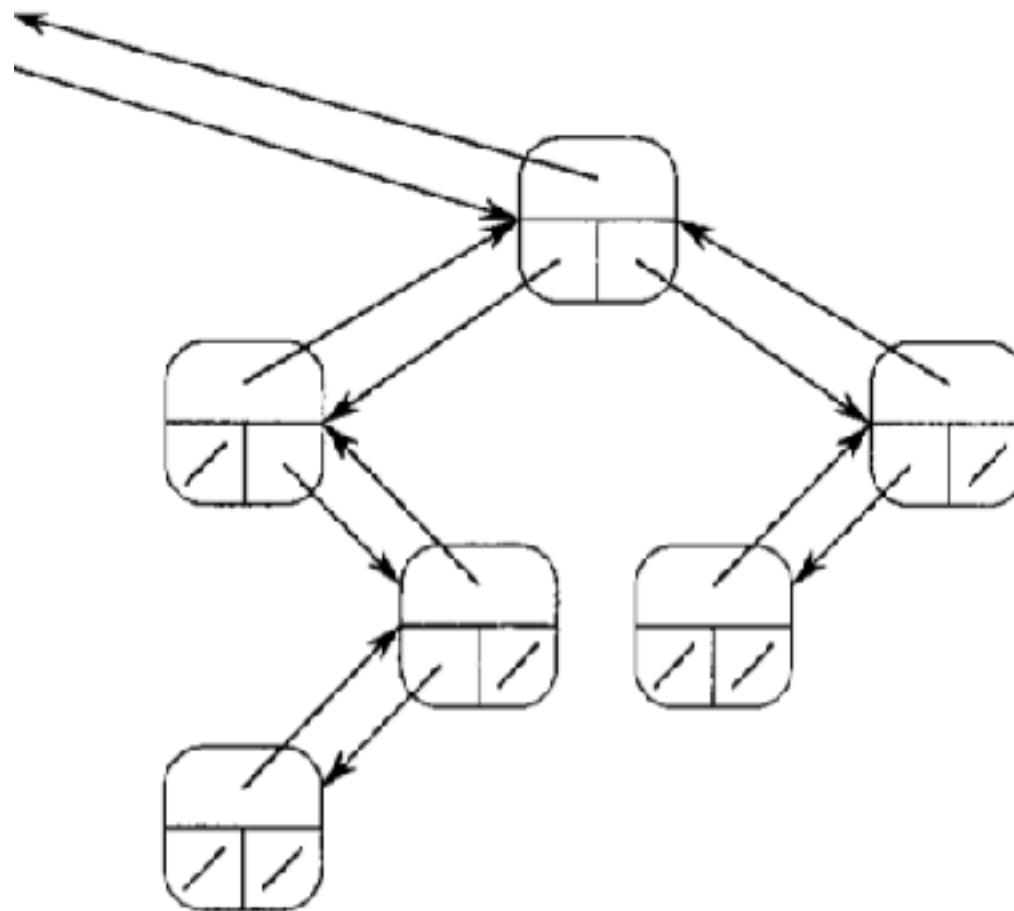


# Binärer Suchbaum



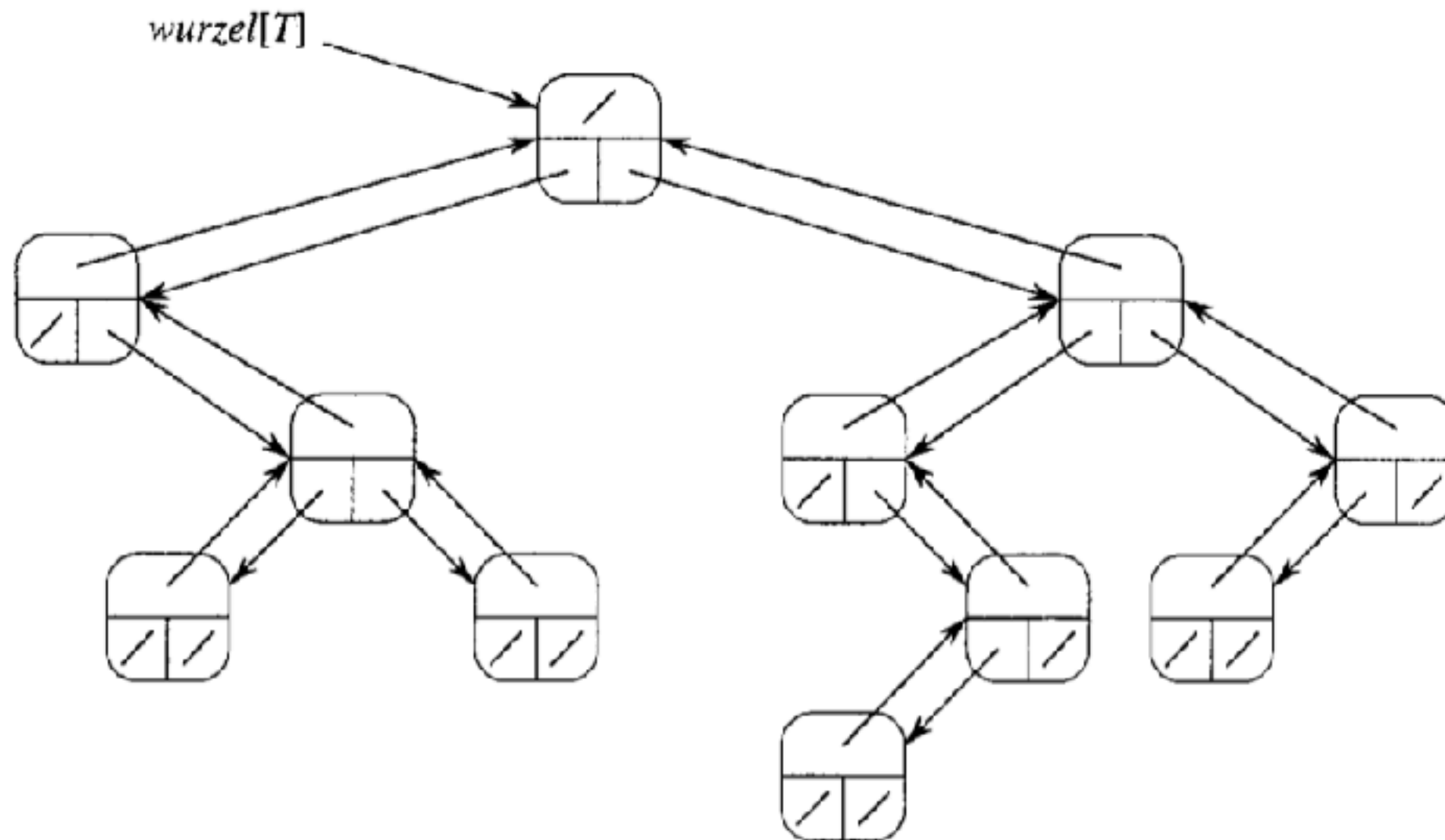


# Binärer Suchbaum



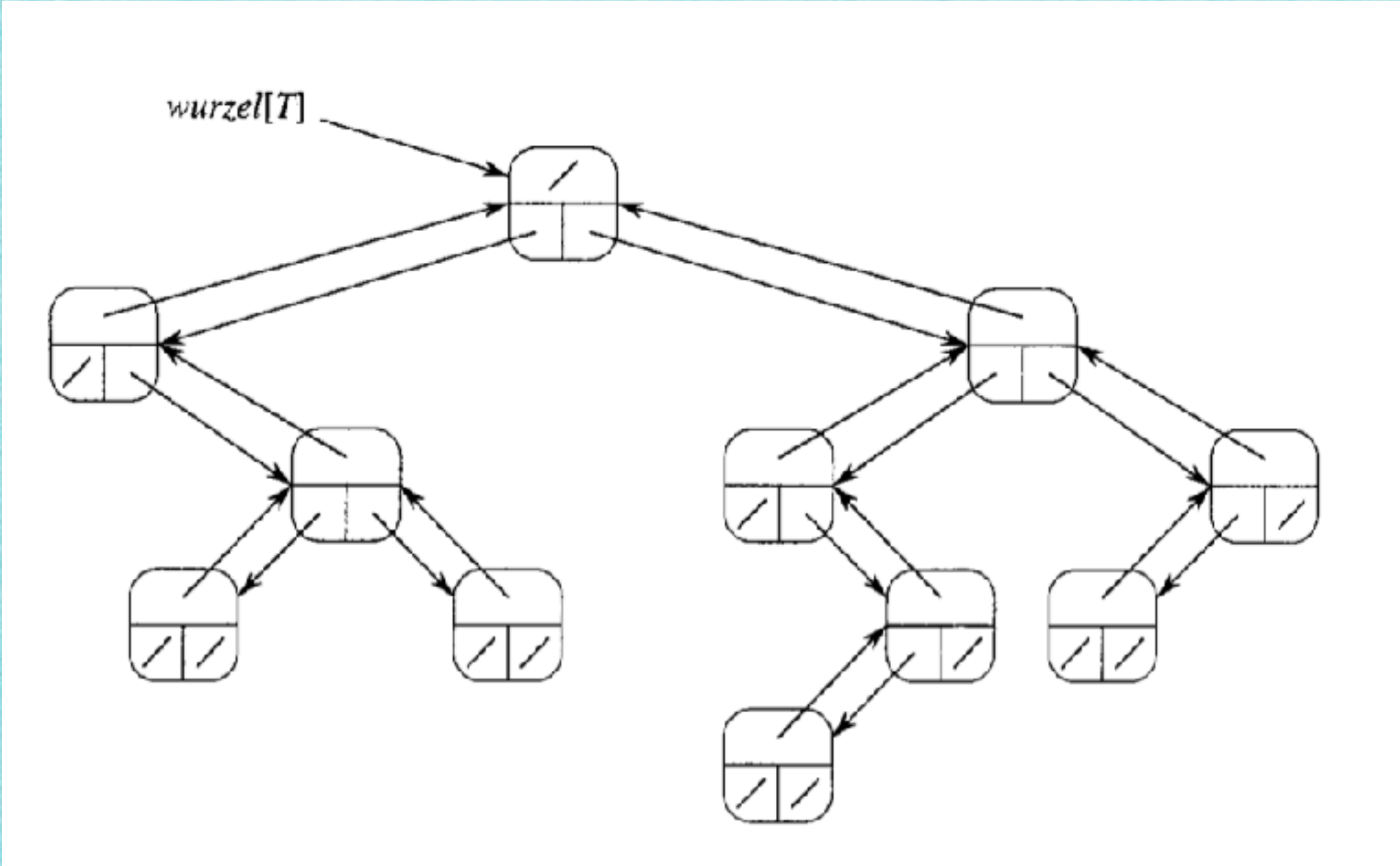


# Binärer Suchbaum





# Binärer Suchbaum



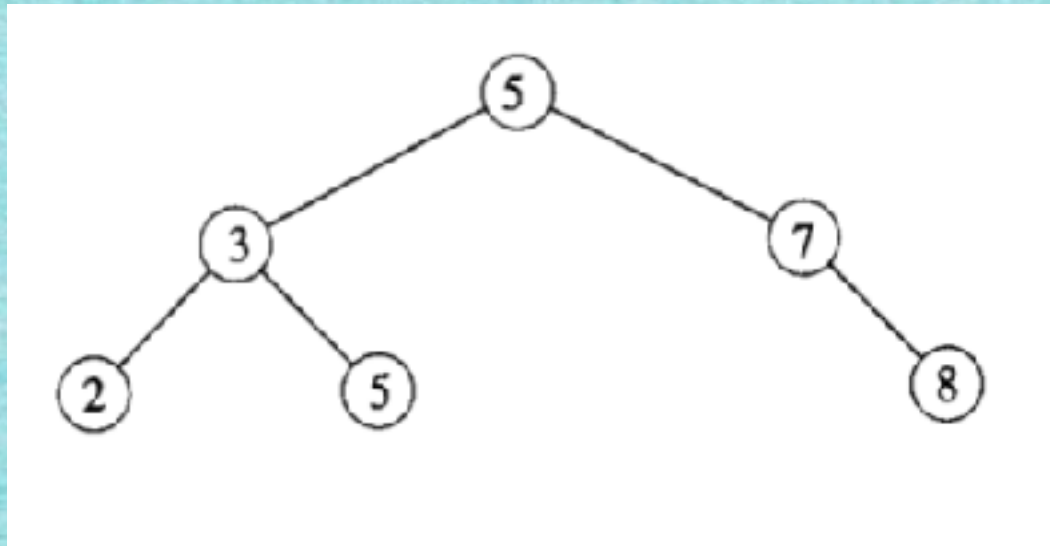
**Außerdem wichtig: Struktur der Schlüsselwerte!**



# Ordnungsstruktur

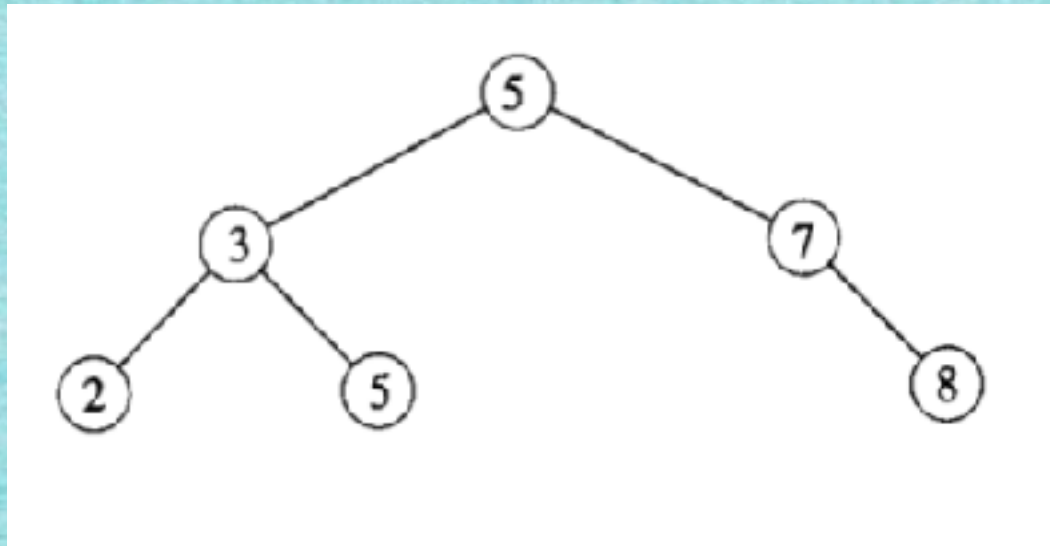


# Ordnungsstruktur





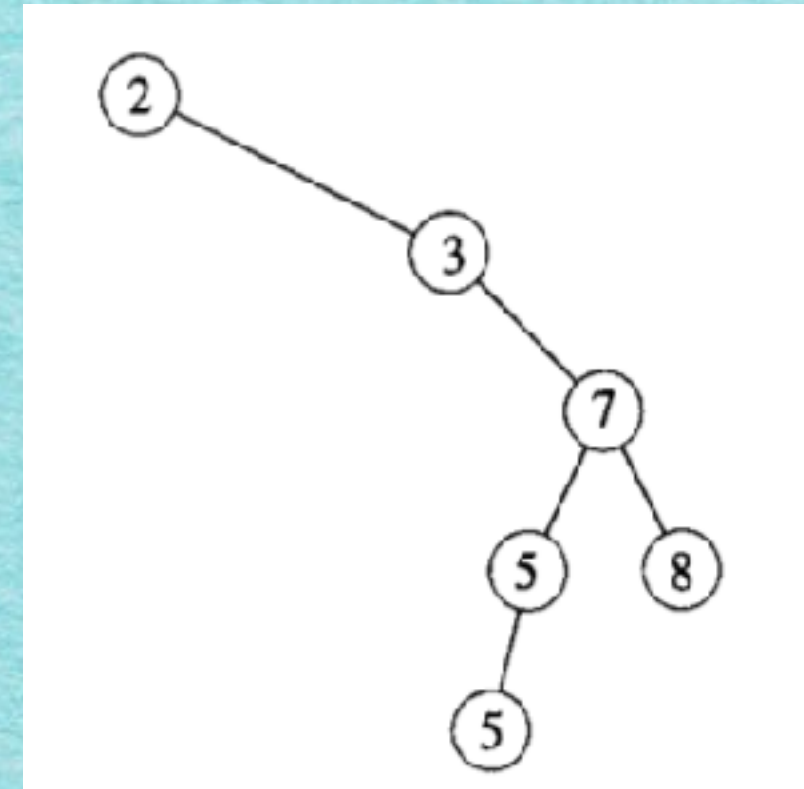
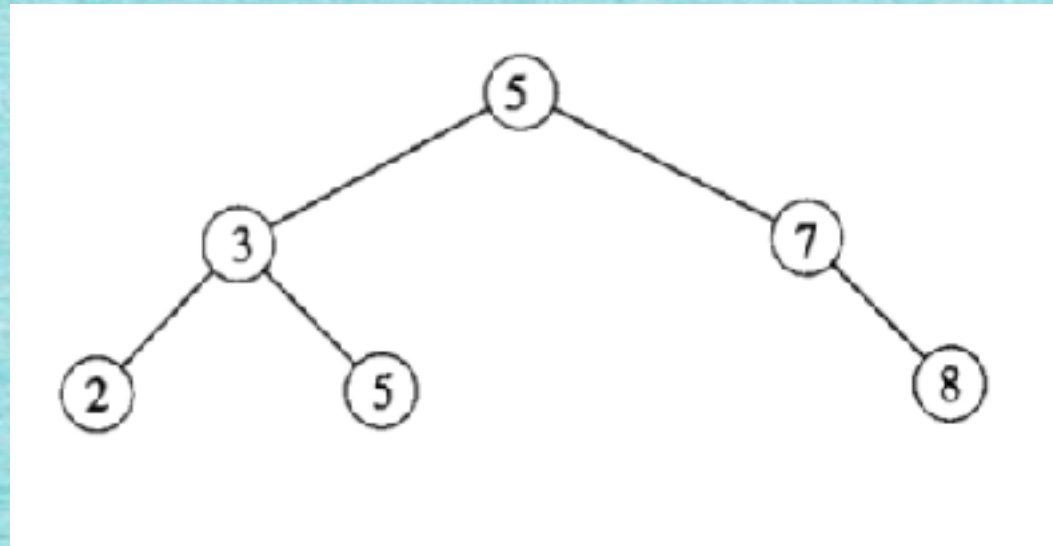
# Ordnungsstruktur



**Linker Teilbaum: Kleinere (bzw. nicht größere) Zahlen**  
**Rechter Teilbaum: Größere Zahlen**



# Ordnungsstruktur



**Linker Teilbaum: Kleinere (bzw. nicht größere) Zahlen**  
**Rechter Teilbaum: Größere Zahlen**



*Mehr demnächst!*



*Mehr demnächst!*

*s.fekete@tu-bs.de*