

Kapitel 4: Dynamische Datenstrukturen

*Algorithmen und Datenstrukturen
WS 2023/24*

Prof. Dr. Sándor Fekete

Wie verwalten wir dynamische Mengen von Objekten?



Waschkorb

4.1 Grundoperationen

Aufgabenstellung:

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

Im Folgenden:

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

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

INSERT(S,x): “Füge x in S ein”

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

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

Langsam:

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

Alle Objekte anschauen

Sehr schnell:

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

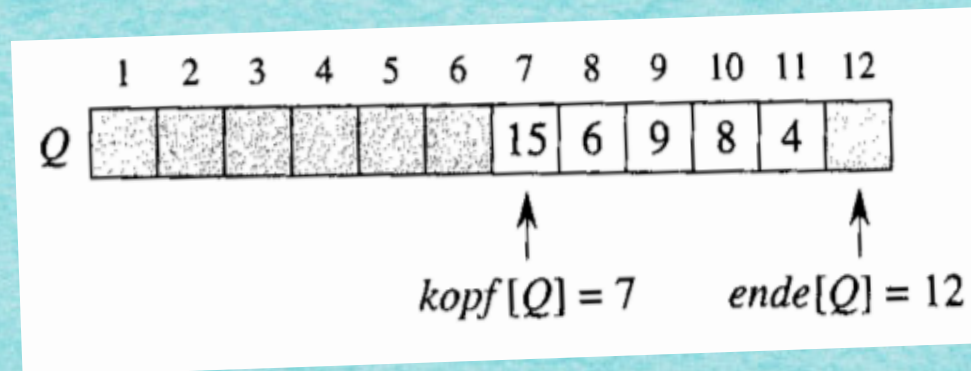
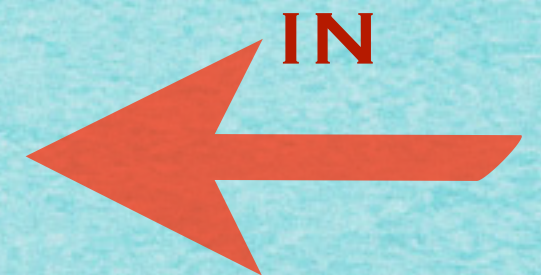
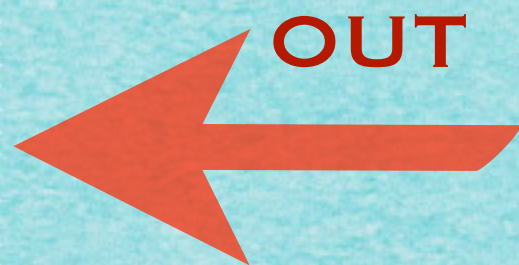
Immer gleich schnell, egal wie groß S ist.

Schnell:

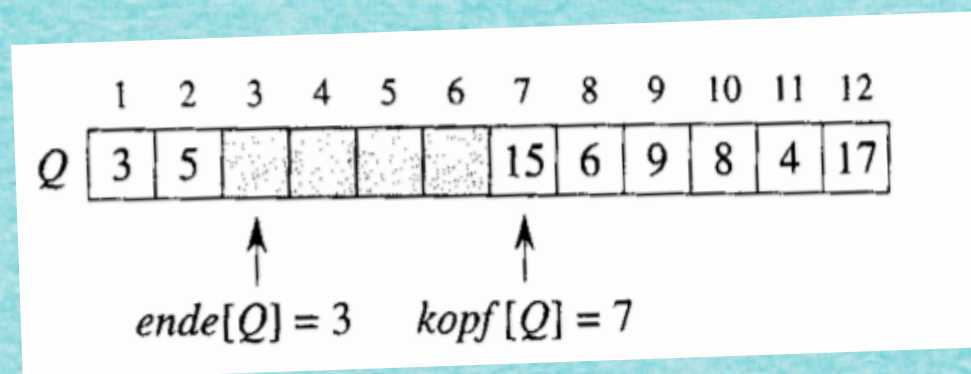
- $O(\log n)$: *logarithmische Zeit*

Wiederholtes Halbieren

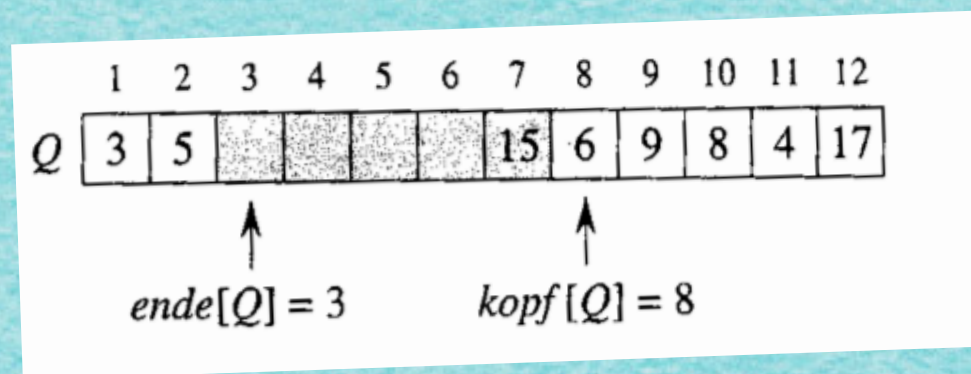
4.2 Stapel und Warteschlange



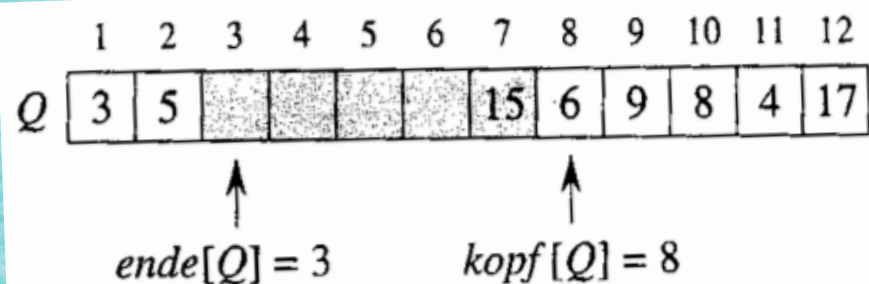
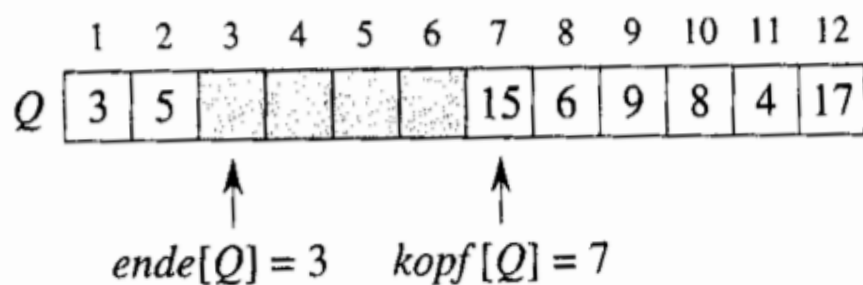
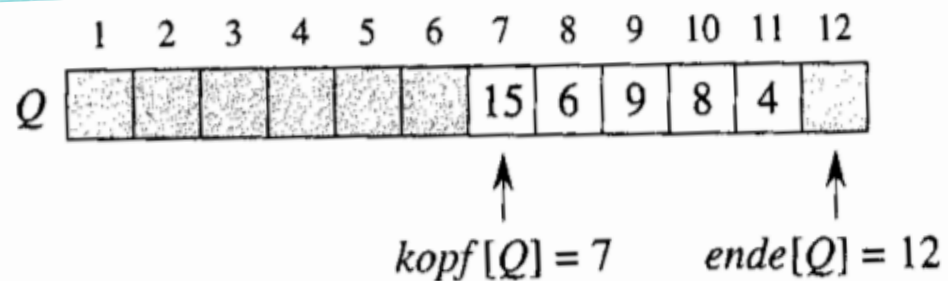
ENQUEUE: 17, 3, 5



DEQUEUE:



WARTESCHLANGE AUF ARRAY UMGESETZT



ENQUEUE(Q, x)

```

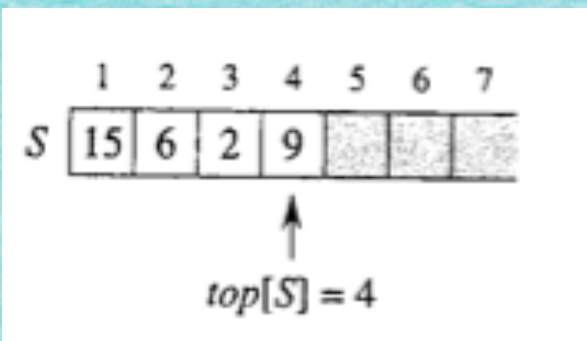
1   $Q[ende[Q]] \leftarrow x$ 
2  if  $ende[Q] = länge[Q]$ 
3     then  $ende[Q] \leftarrow 1$ 
4     else  $ende[Q] \leftarrow ende[Q] + 1$ 
```

DEQUEUE(Q)

```

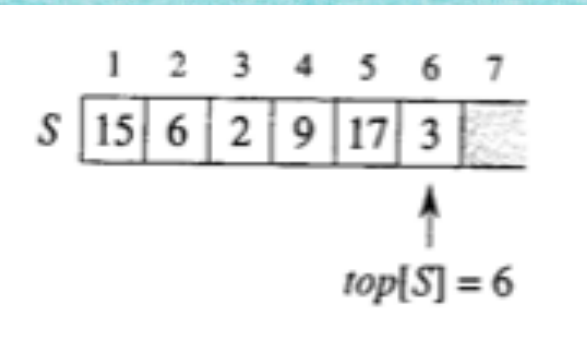
1   $x \leftarrow Q[kopf[Q]]$ 
2  if  $kopf[Q] = länge[Q]$ 
3     then  $kopf[Q] \leftarrow 1$ 
4     else  $kopf[Q] \leftarrow kopf[Q] + 1$ 
5  return  $x$ 
```


STACK AUF ARRAY UMGESETZT



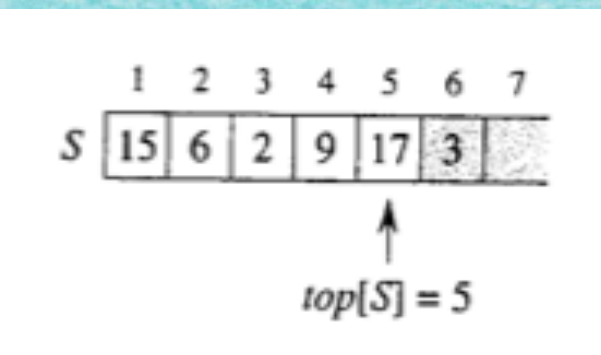
```
STACK-EMPTY(S)
1  if  $top[S] = 0$ 
2     then return WAHR
3     else return FALSCH
```

PUSH: 17, 3

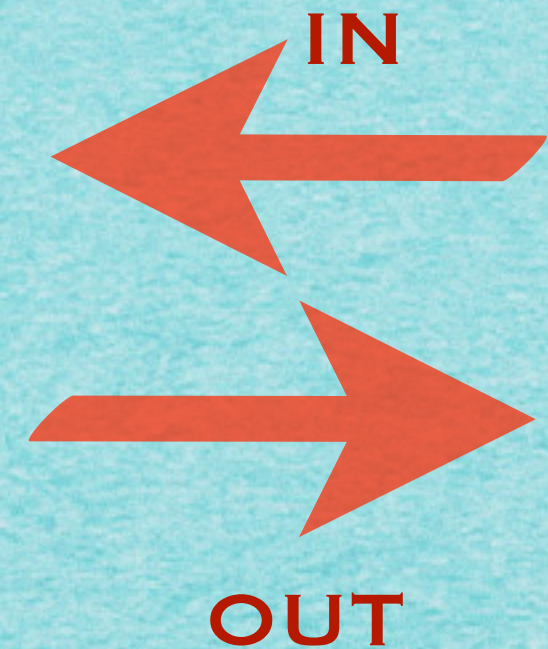


```
PUSH(S, x)
1   $top[S] \leftarrow top[S] + 1$ 
2   $S[top[S]] \leftarrow x$ 
```

POP



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



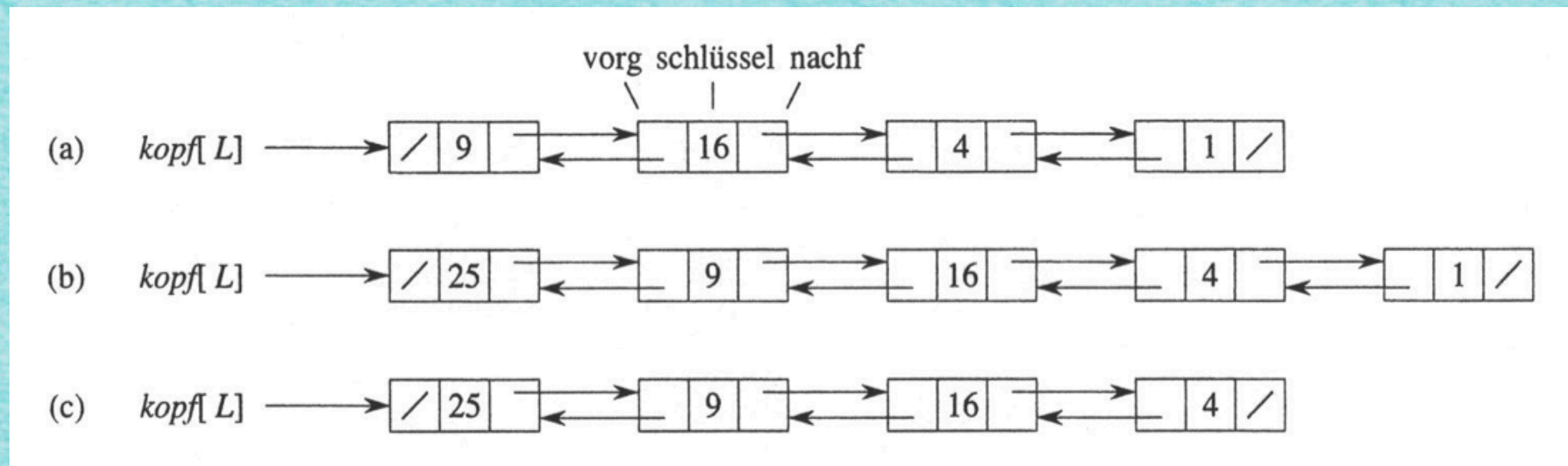
4.3 Verkettete Listen

Idee:



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

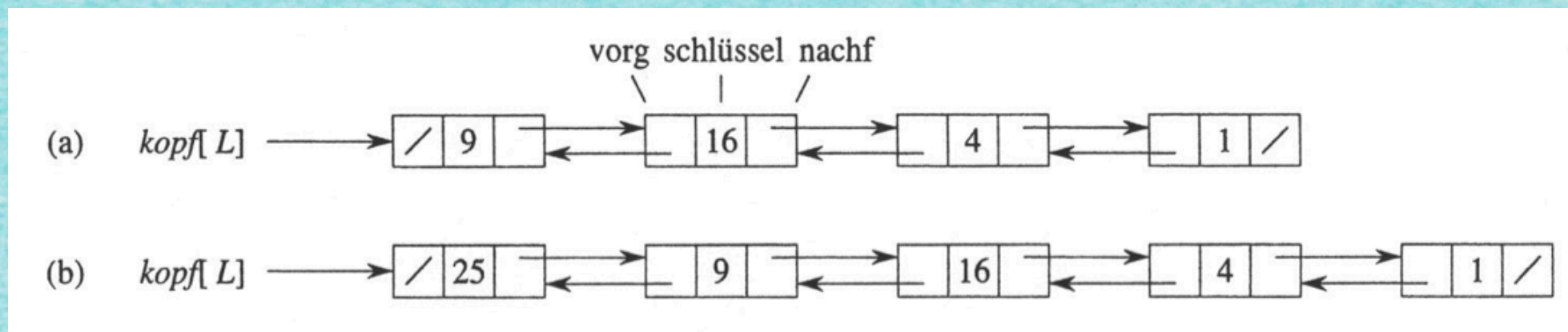
Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

- **Finde ein Element mit Schlüssel 1 und lösche es.**

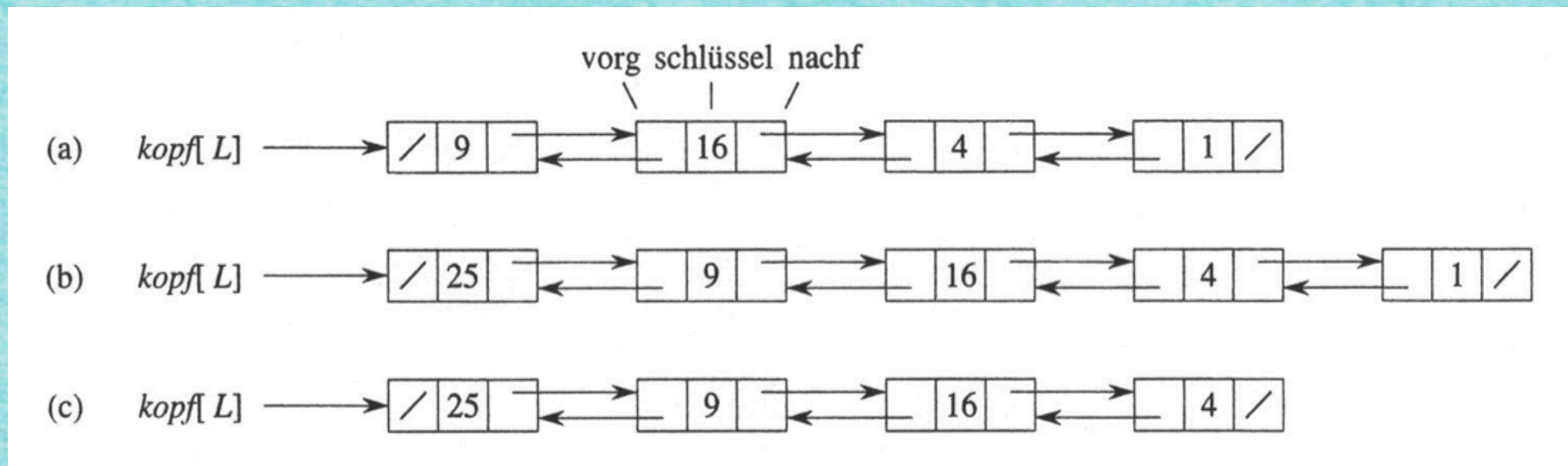
Einfügen in eine doppelt verkettete Liste



```
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)

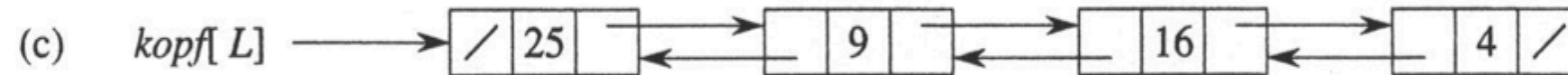
Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

- **Finde ein Element mit Schlüssel 1 und lösche es.**

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  $x$ 
```

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: $O(n)$

Laufzeit: $O(1)$

4.4 Binäre Suche

Aufgabenstellung:

- *Rate eine Zahl zwischen 100 und 114!*

<



>

100 101 102 103 104 105 106

107

108 109 110 111 112 113 114

4.4 Binäre Suche

Aufgabenstellung:

- *Rate eine Zahl zwischen 100 und 114!*

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114



4.4 Binäre Suche

Aufgabenstellung:

- *Rate eine Zahl zwischen 100 und 114!*

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114



4.4 Binäre Suche

Aufgabenstellung:

- *Rate eine Zahl zwischen 100 und 114!*

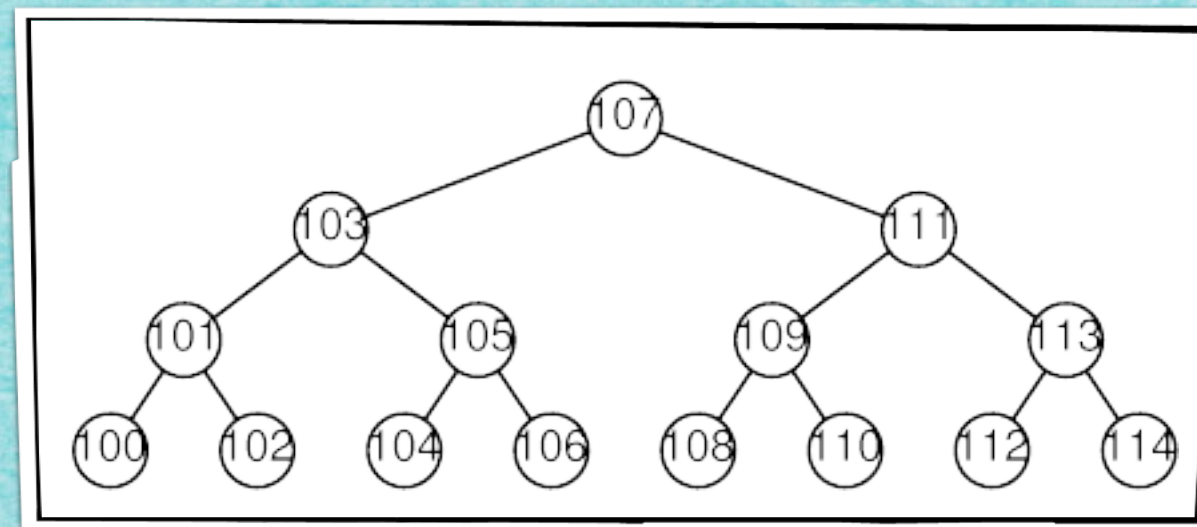
100 101 102 103 104 105 106 107 108 109 **110** 111 112 113 114

!

4.4 Binäre Suche

Aufgabenstellung:

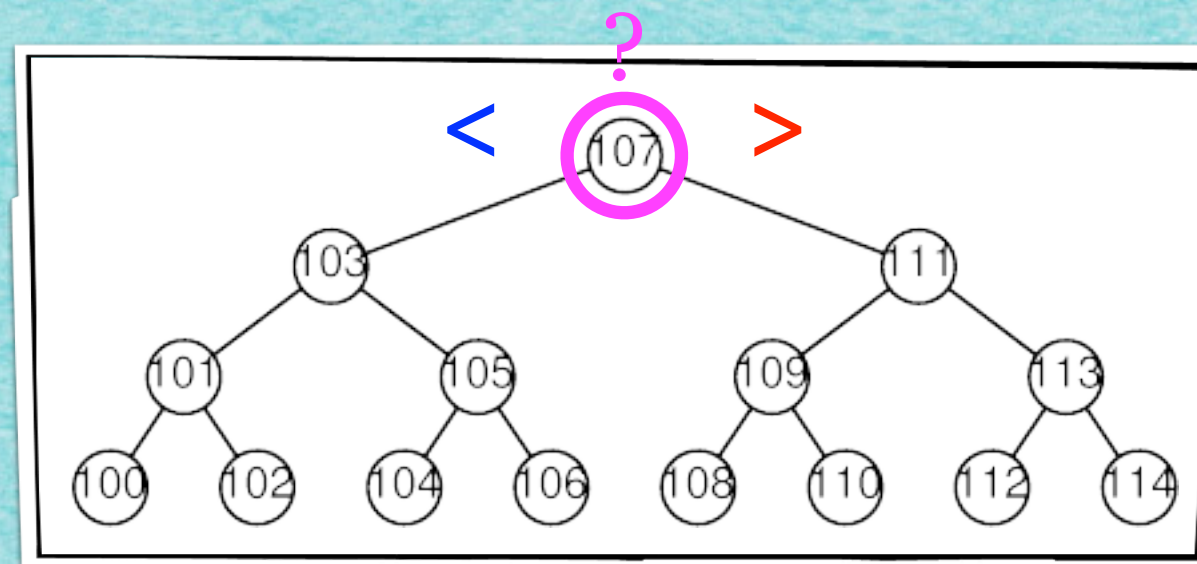
- *Rate eine Zahl zwischen 100 und 114!*



4.4 Binäre Suche

Aufgabenstellung:

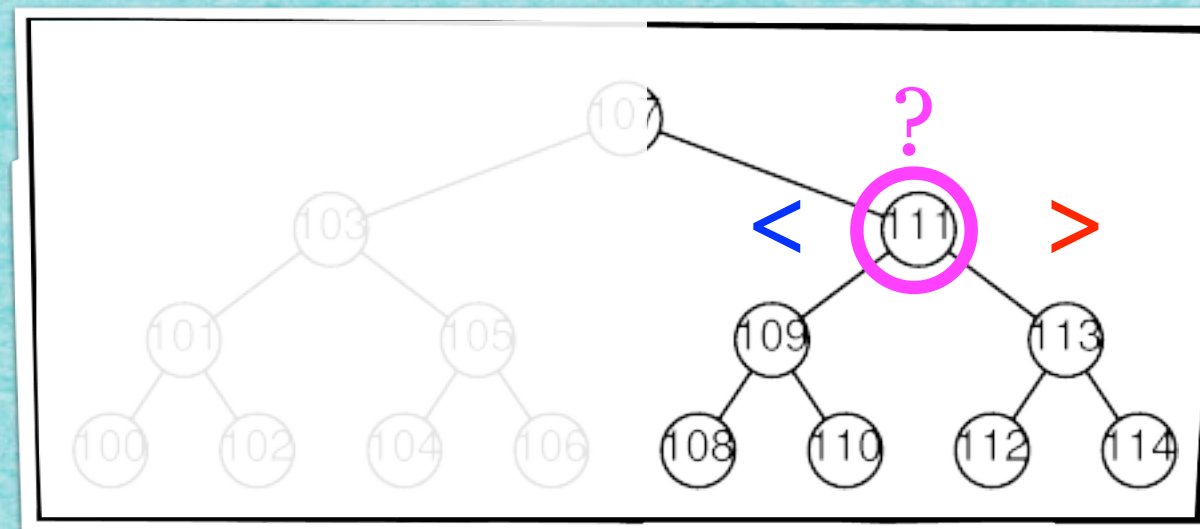
- *Rate eine Zahl zwischen 100 und 114!*



4.4 Binäre Suche

Aufgabenstellung:

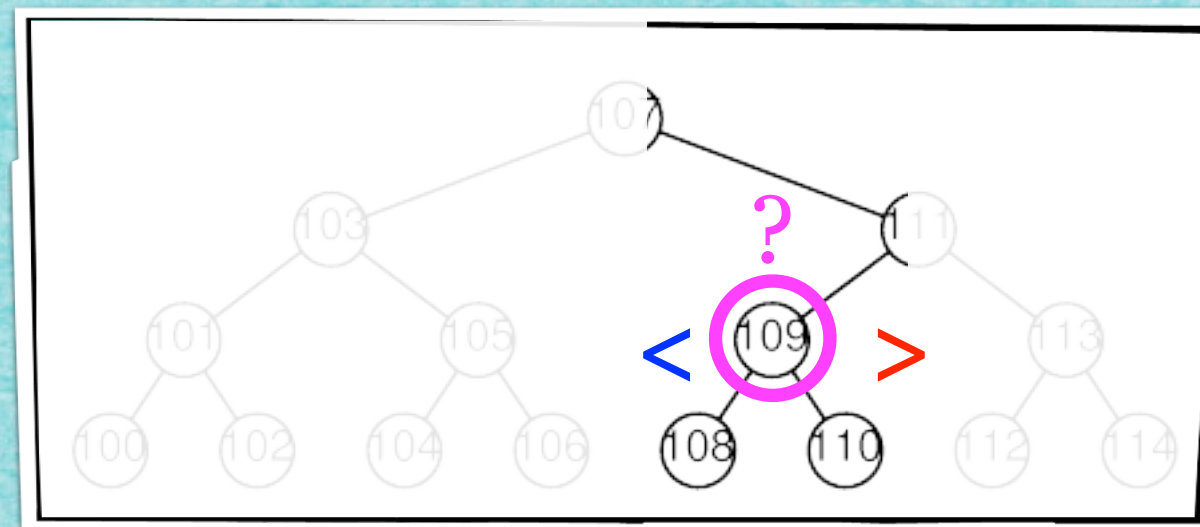
- *Rate eine Zahl zwischen 100 und 114!*



4.4 Binäre Suche

Aufgabenstellung:

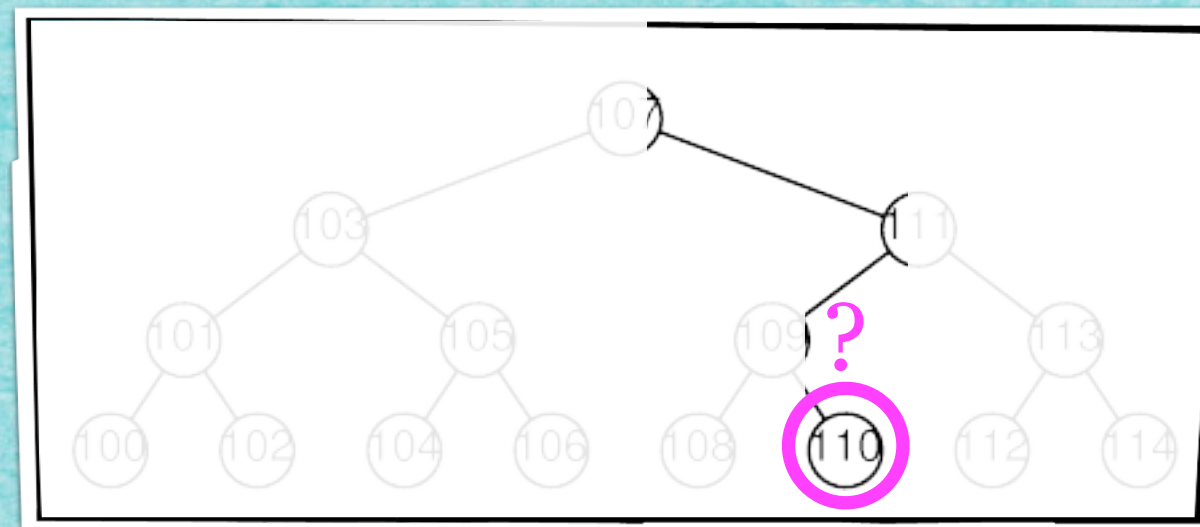
- *Rate eine Zahl zwischen 100 und 114!*



4.4 Binäre Suche

Aufgabenstellung:

- *Rate eine Zahl zwischen 100 und 114!*



Algorithmus 4.1

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!"

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

	LINKS							MITTE								RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14		
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96		

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

	WERT=42													
							MITTE	LINKS						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

		WERT=42														
								LINKS			MITTE				RECHTS	
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14		
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96		

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

RETURN 9

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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

	LINKS																RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14			
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96			

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

	LINKS		WERT=7				RECHTS	MITTE							
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96	

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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

	LINKS		MITTE			RECHTS								
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

			WERT=7	LINKS	MITTE	RECHTS									
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96	

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊  $\frac{LINKS + RECHTS}{2}$  ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

				WERT=7													
				RECHTS													
				MITTE													
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14			
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96			

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

				WERT=7										
				LINKS										
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

```
1. WHILE (LINKS ≤ RECHTS) DO {  
    1.1. MITTE := ⌊ (LINKS + RECHTS) / 2 ⌋  
    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!"
```

Binäre Suche

Aufgabenstellung:

- *Finde eine gesuchte Zahl in der gegebenen sortierten Menge!*

				WERT=7											
				RECHTES											
				LINKS											
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96	

WERT nicht gefunden!

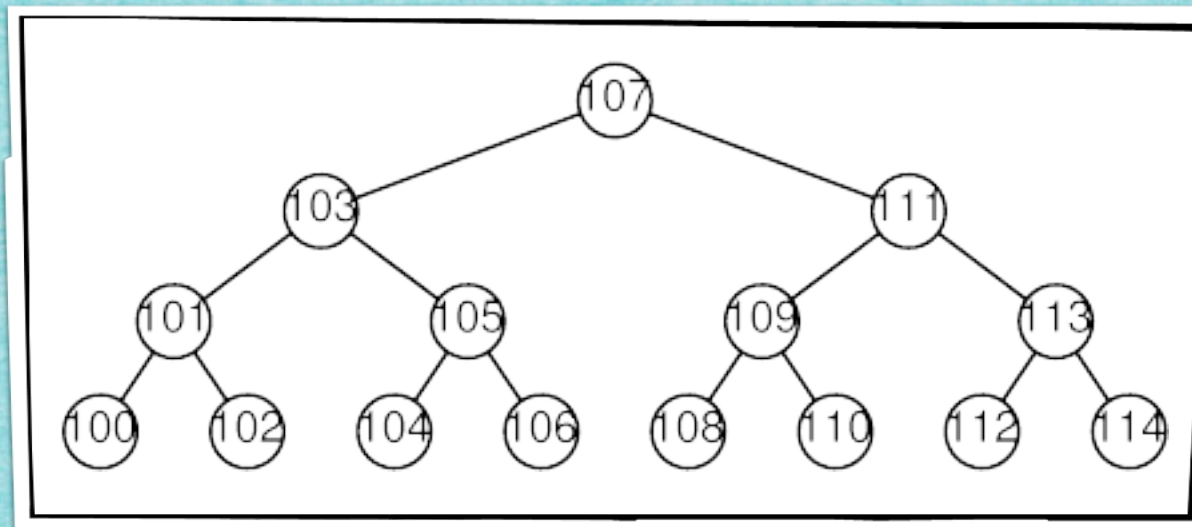
4.4 Binäre Suche

Satz 4.2

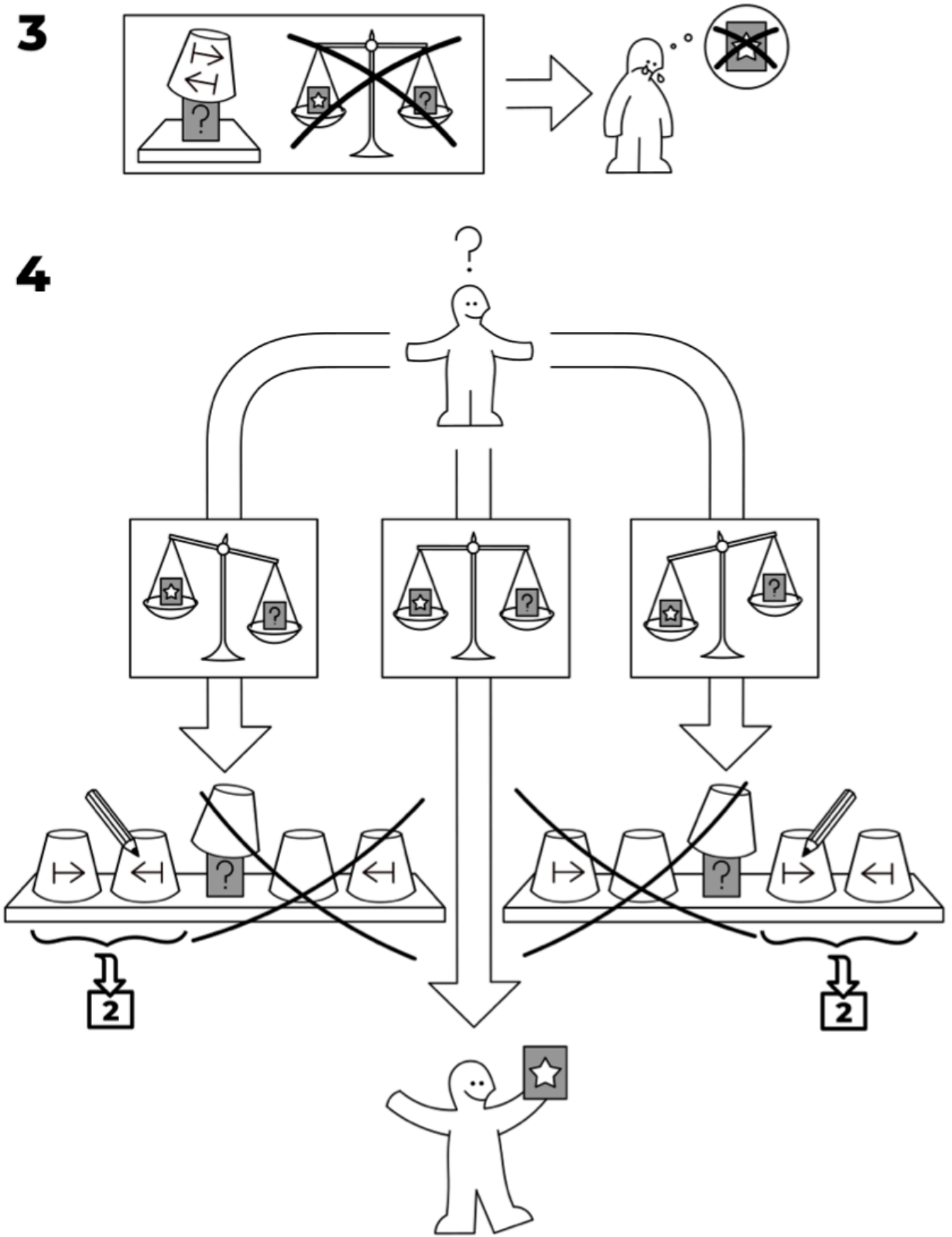
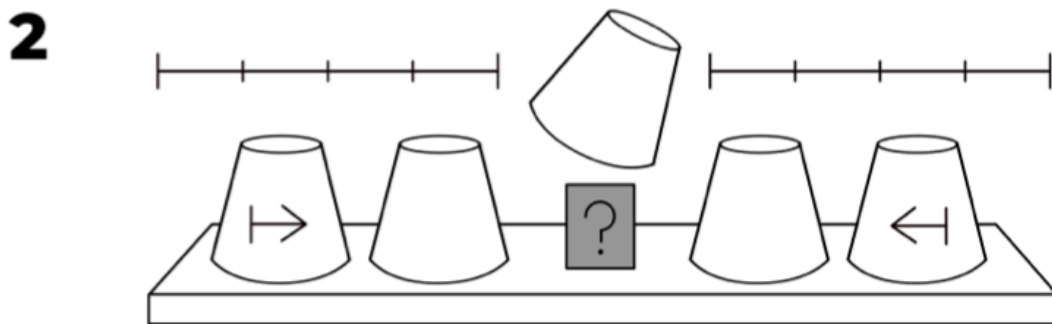
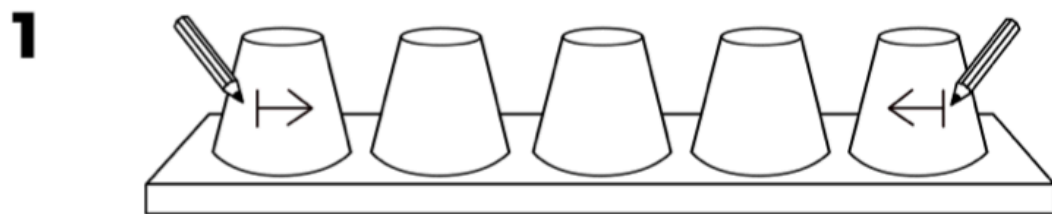
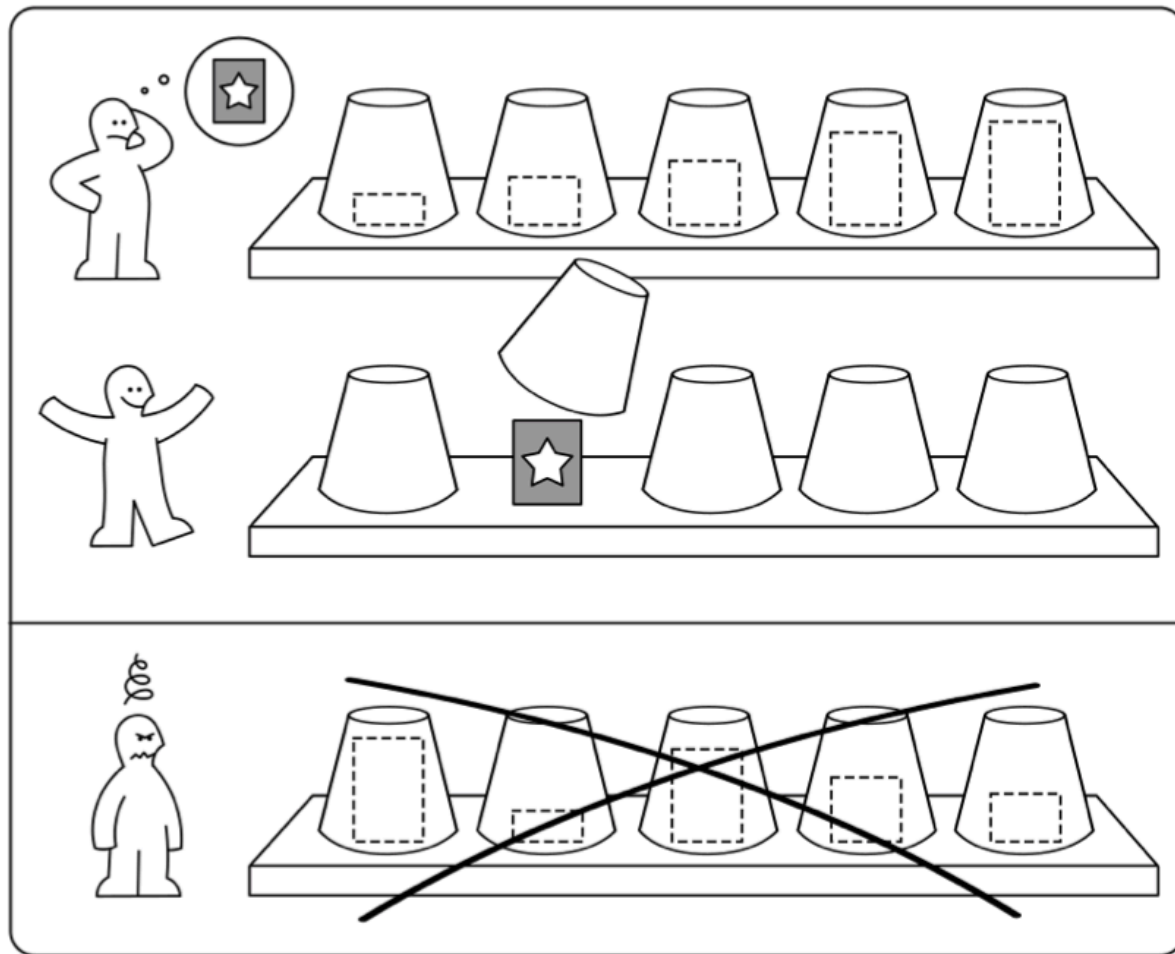
Die binäre Suche terminiert in $O(\log(\text{RECHTS}-\text{LINKS}))$ Schritten (für $\text{RECHTS} > \text{LINKS}$).

Beweis:

Selbst!



BINÄRY SEARCH



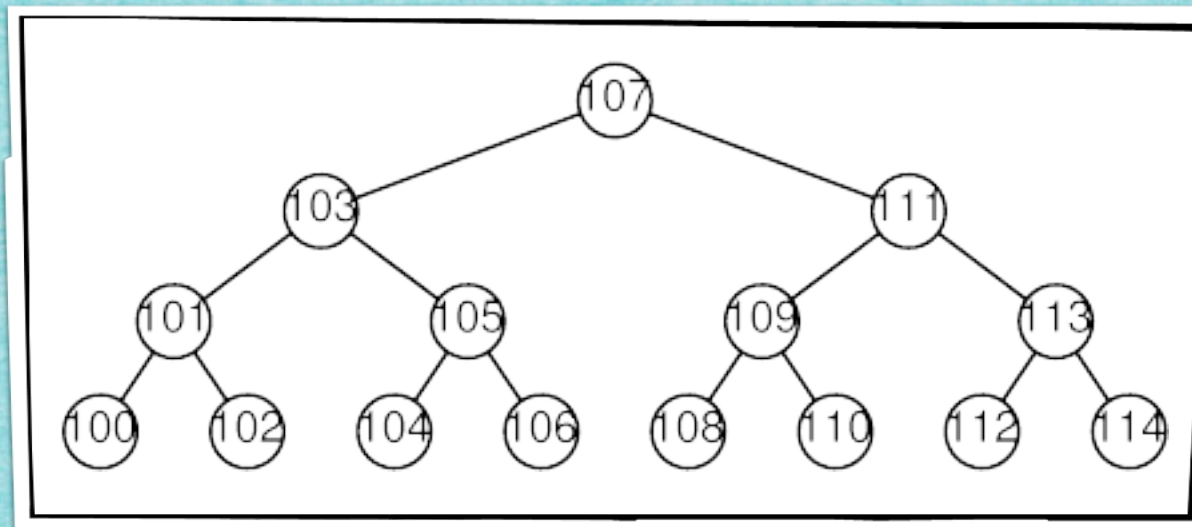
4.4 Binäre Suche

Satz 4.2

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

Beweis:

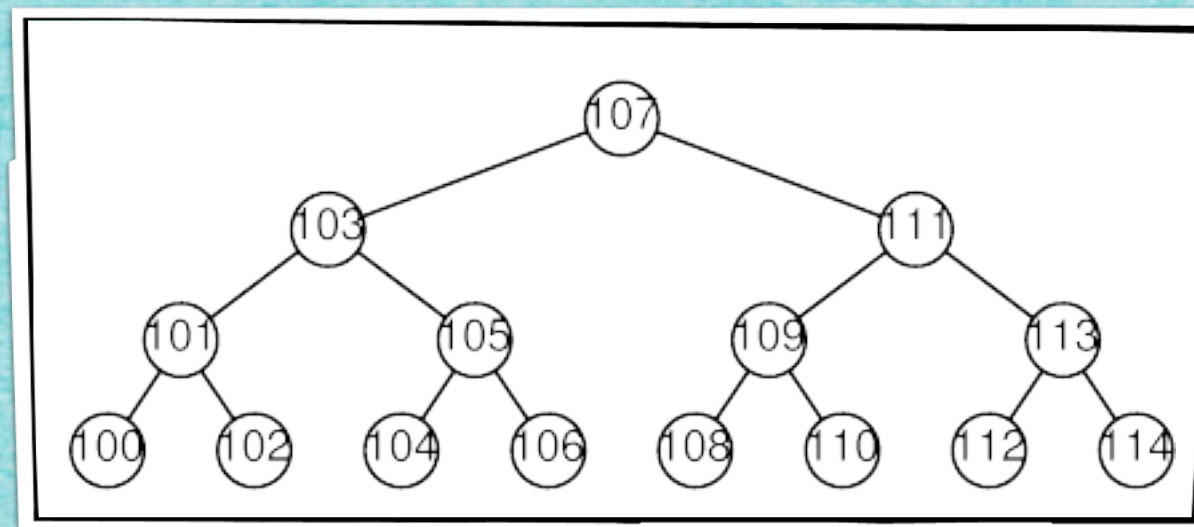
Selbst!



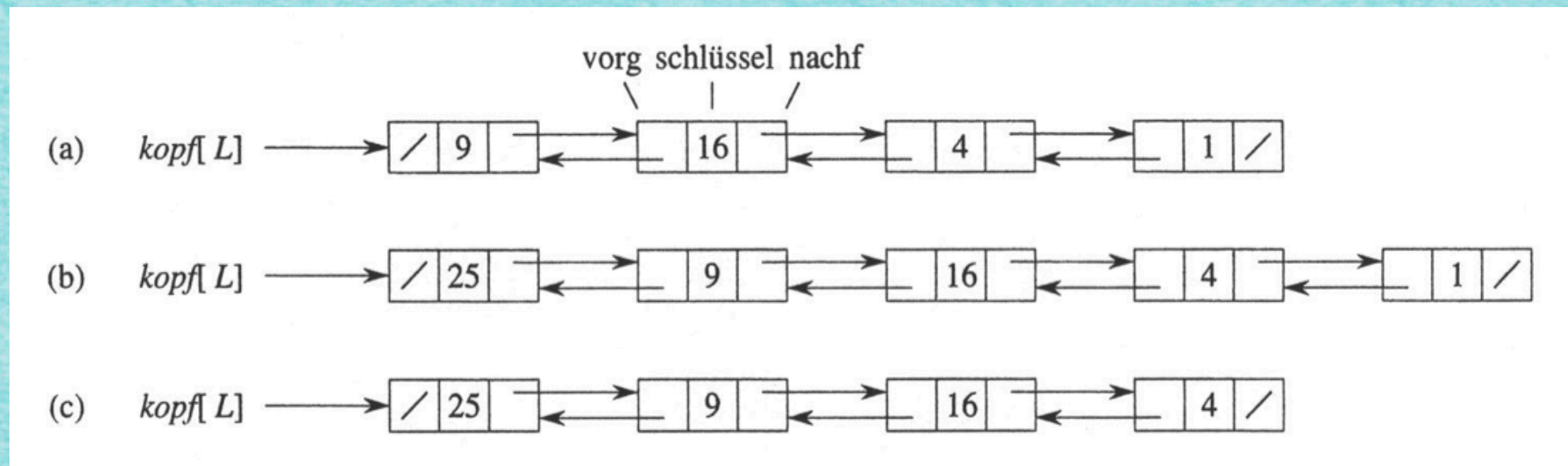
4.5 Binäre Suchbäume

Ideen:

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



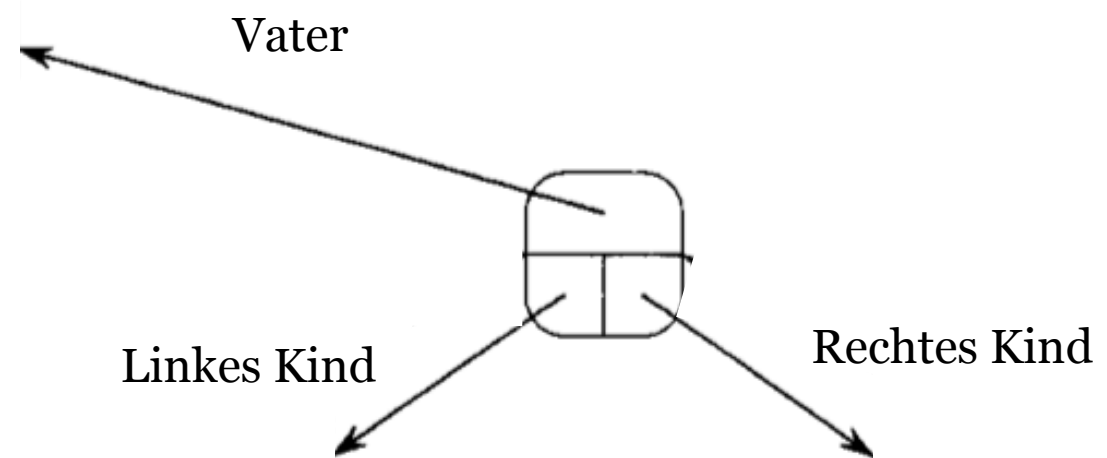
Struktur einer doppelt verketteten Liste



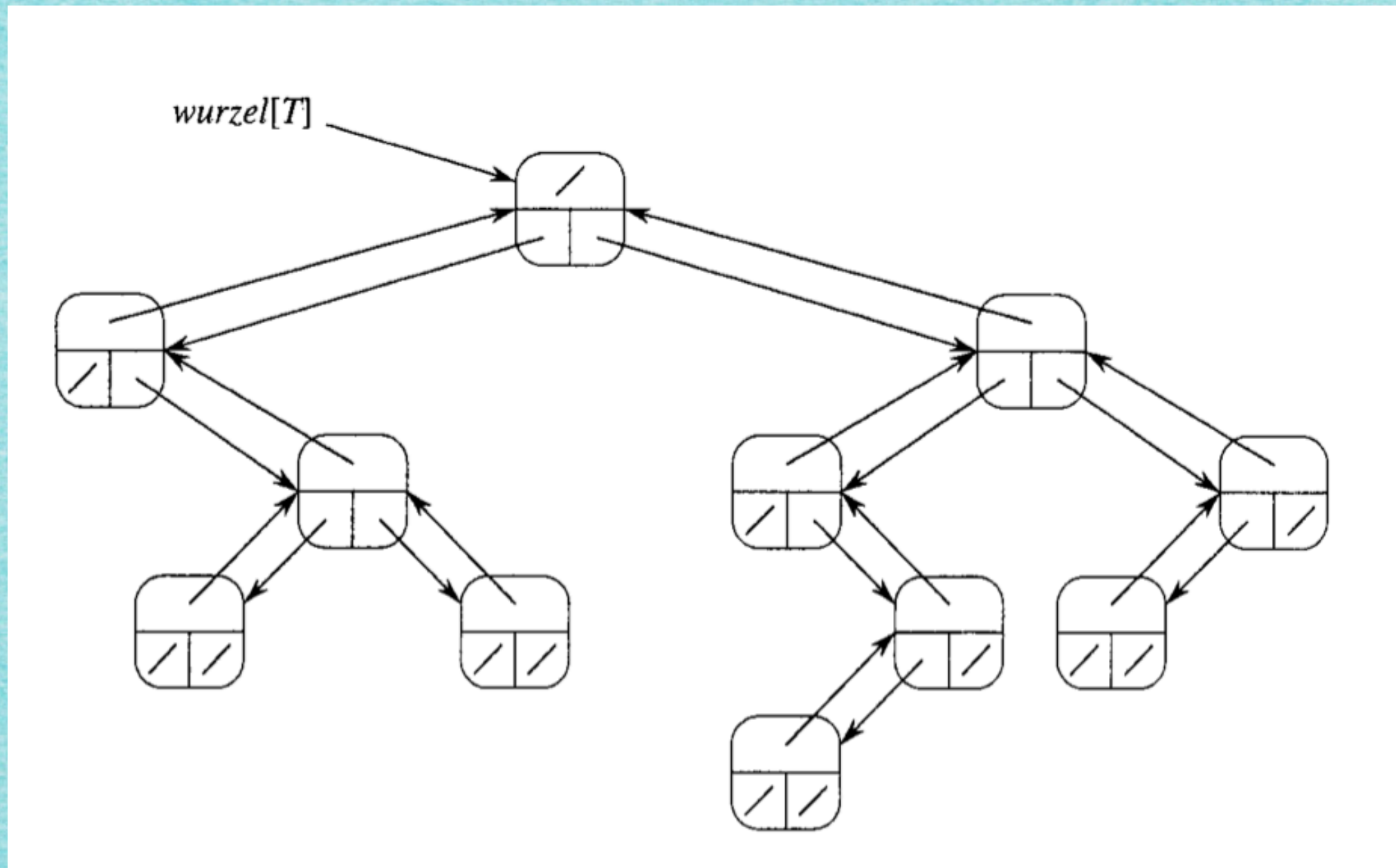
- **Füge vorne das Element mit Schlüssel 25 ein.**

- **Finde ein Element mit Schlüssel 1 und lösche es.**

Binärer Suchbaum

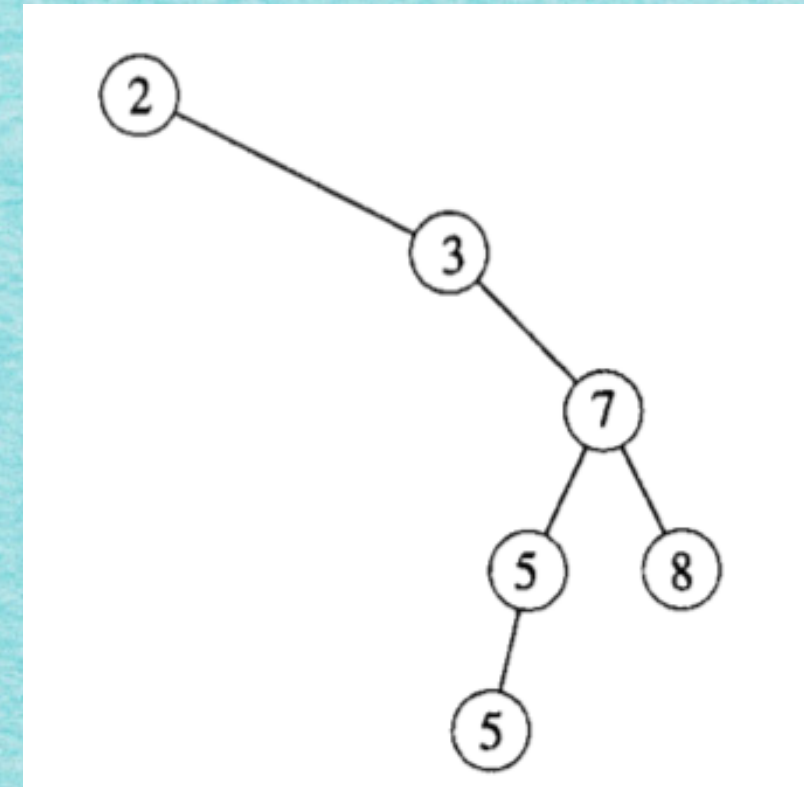
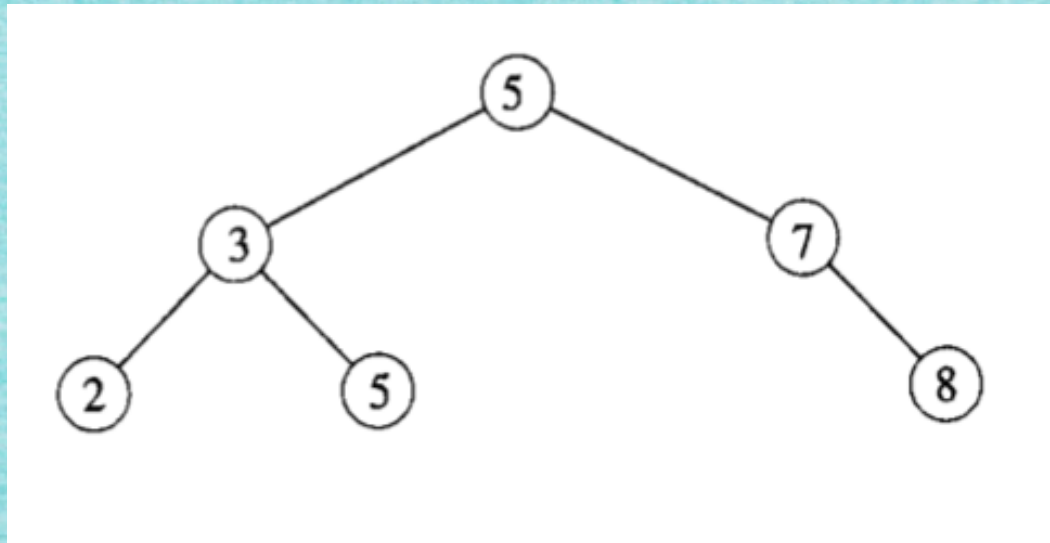


Binärer Suchbaum



Außerdem wichtig: Struktur der Schlüsselwerte!

Ordnungsstruktur

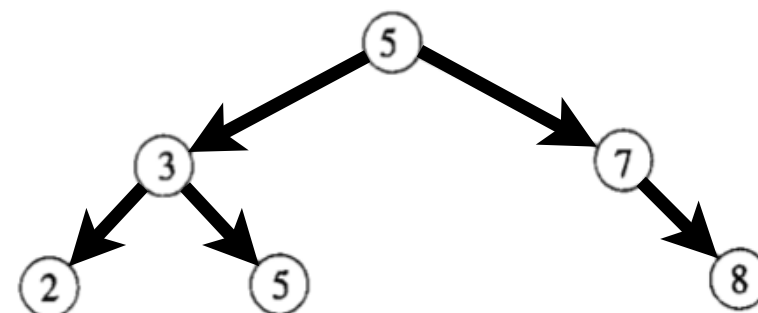


Linker Teilbaum: Kleinere (bzw. nicht größere) Zahlen
Rechter Teilbaum: Größere Zahlen

4.5 Binäre Suchbäume

Definition 4.3

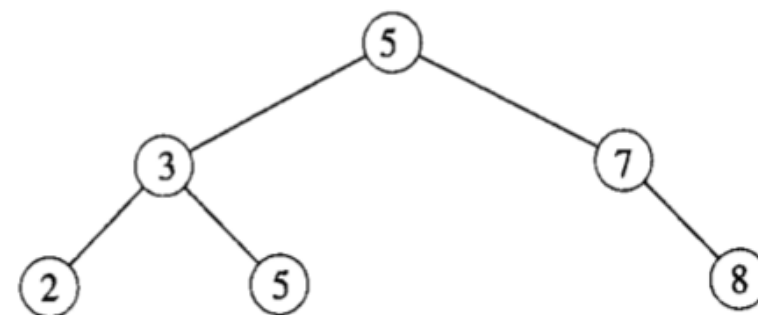
- (1)** Ein gerichteter Graph $D=(V,A)$ besteht aus einer endlichen Menge V von Knoten v und einer endlichen Menge von gerichteten Kanten, $a=(v,w)$. (v ist Vorgänger von w .)
- (2)** Ein gerichteter Baum $B=(V,T)$ hat folgende Eigenschaften:
 - (i) Es gibt einen eindeutigen Knoten w ohne Vorgänger.
 - (ii) Jeder Knoten außer w ist auf einem eindeutigen Weg von w aus erreichbar; das heißt insbesondere, dass v einen eindeutigen Vorgänger (Vaterknoten) hat.
- (3)** Die Höhe eines gerichteten Baumes ist die maximale Länge eines gerichteten Weges von der Wurzel.
- (4)** Ein binärer Baum ist ein gerichteter Baum, in dem jeder Knoten höchstens zwei Nachfolger (“Kindknoten”) hat. Einer ist der “linke” $l[v]$, der andere der “rechte”, $r[v]$.



4.5 Binäre Suchbäume

Definition 4.3

- (1)** Ein gerichteter Graph $D=(V,A)$ besteht aus einer endlichen Menge V von Knoten v und einer endlichen Menge von gerichteten Kanten, $a=(v,w)$. (v ist Vorgänger von w .)
- (2)** Ein gerichteter Baum $B=(V,T)$ hat folgende Eigenschaften:
 - (i) Es gibt einen eindeutigen Knoten w ohne Vorgänger.
 - (ii) Jeder Knoten außer w ist auf einem eindeutigen Weg von w aus erreichbar; das heißt insbesondere, dass v einen eindeutigen Vorgänger (Vaterknoten) hat.
- (3)** Die Höhe eines gerichteten Baumes ist die maximale Länge eines gerichteten Weges von der Wurzel.
- (4)** Ein binärer Baum ist ein gerichteter Baum, in dem jeder Knoten höchstens zwei Nachfolger (“Kindknoten”) hat. Einer ist der “linke” $l[v]$, der andere der “rechte”, $r[v]$.



4.5 Binäre Suchbäume

Definition 4.3 (Forts.)

(5) Ein binärer Baum heißt voll, wenn jeder Knoten zwei oder keinen Kindknoten hat.

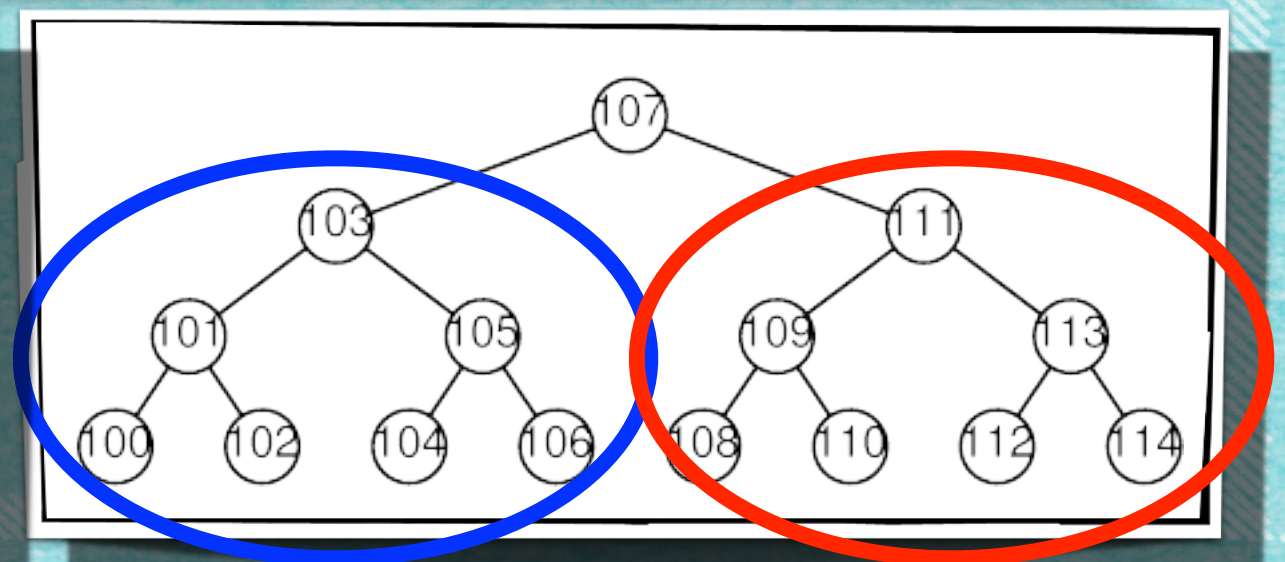
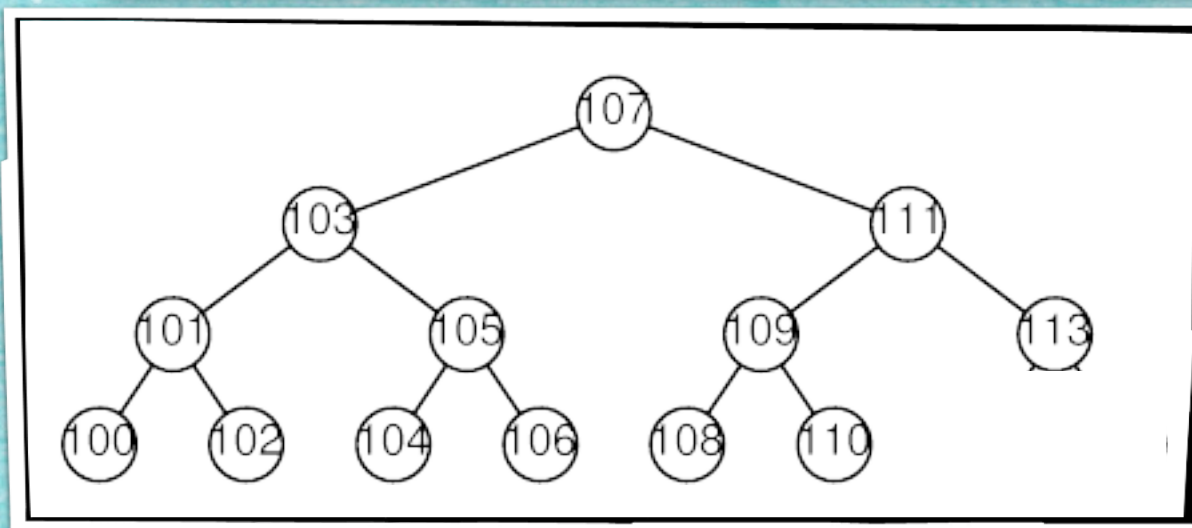
(6) Ein binärer Baum heißt vollständig, wenn zusätzlich alle Blätter gleichen Abstand zur Wurzel haben.

(7) Ein Knoten ohne Kindknoten heißt Blatt.

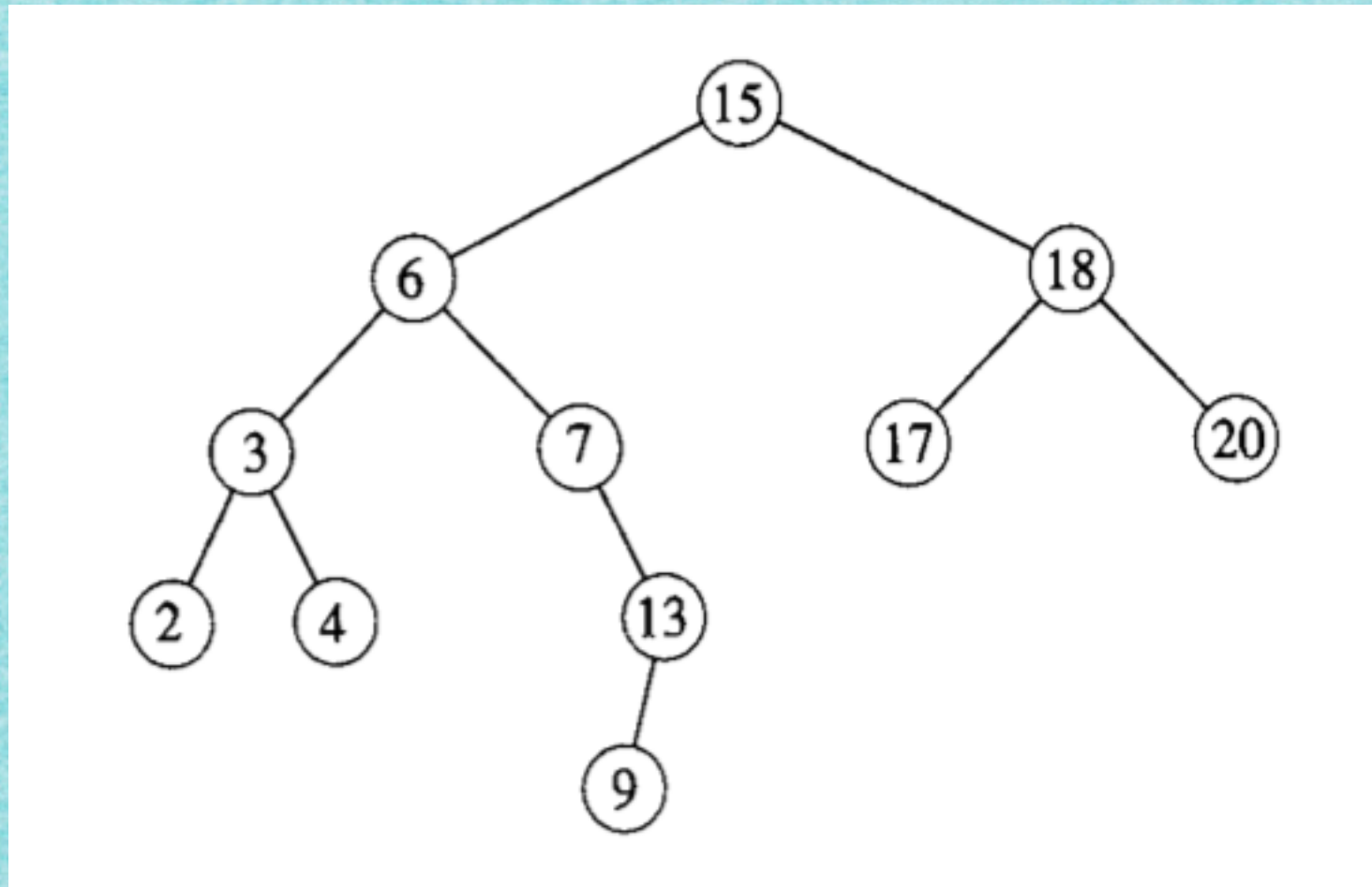
(8) Der Teilbaum eines Knotens v ist durch die Menge der von v erreichbaren Knoten und der dabei verwendeten Kanten definiert; der linke Teilbaum ist der Teilbaum von $l[v]$.

(9) In einem binären Suchbaum hat jeder Knoten v einen Schlüsselwert $S[v]$, und es gilt:

- $S[u] \leq S[v]$ für Knoten u im linken Teilbaum von v
- $S[u] > S[v]$ für Knoten u im rechten Teilbaum von v



Minimum und Maximum



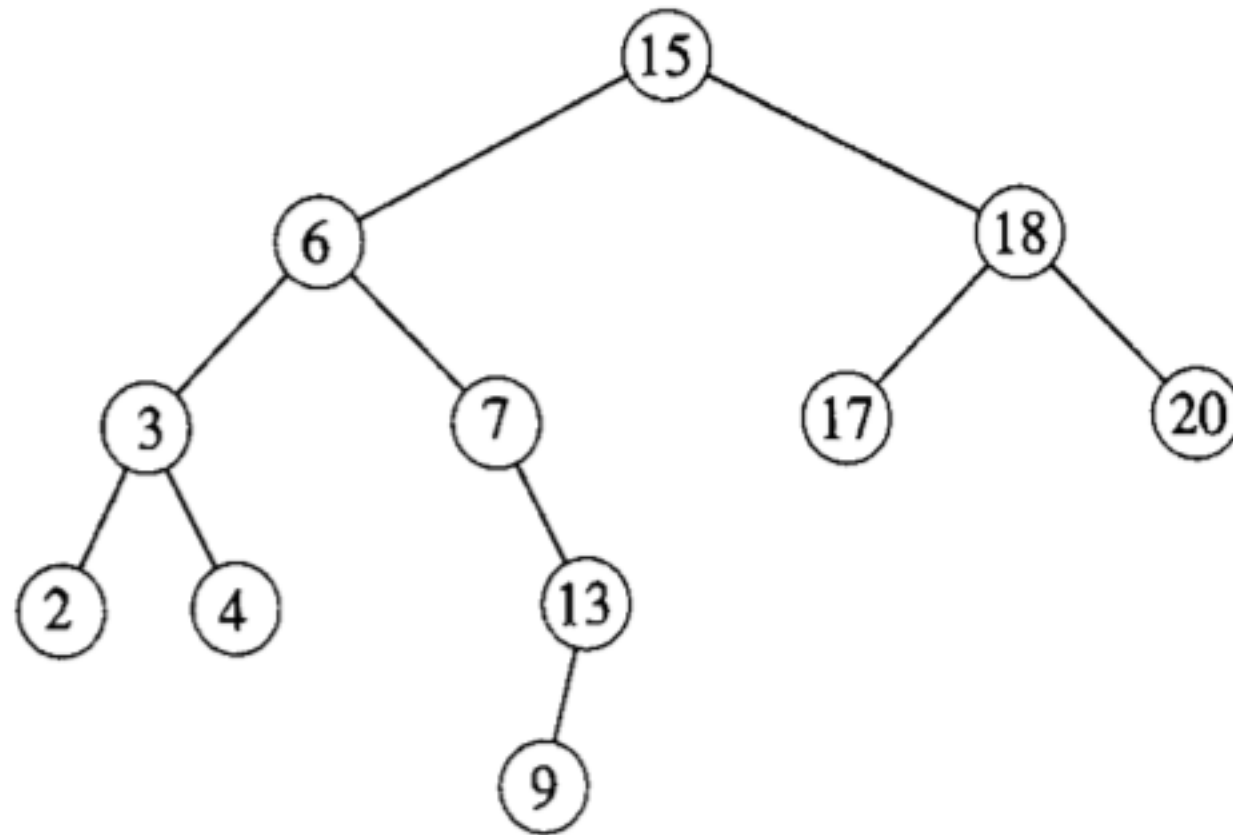
TREE-MINIMUM(x)

```
1 while  $links[x] \neq NIL$ 
2   do  $x \leftarrow links[x]$ 
3 return  $x$ 
```

TREE-MAXIMUM(x)

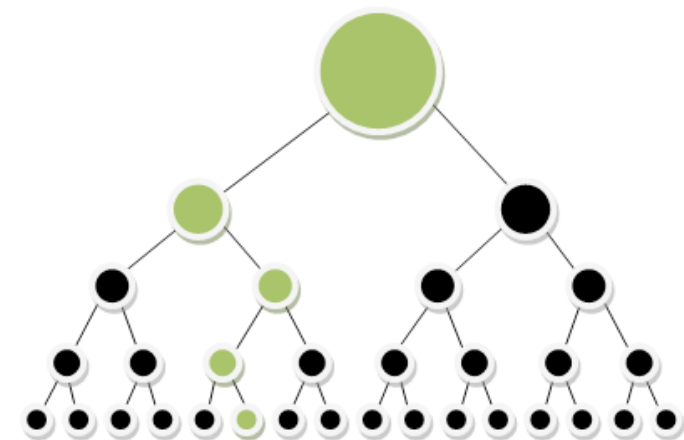
```
1 while  $rechts[x] \neq NIL$ 
2   do  $x \leftarrow rechts[x]$ 
3 return  $x$ 
```


Suche im Suchbaum



ITERATIVE-TREE-SEARCH(x, k)

```
1 while  $x \neq \text{NIL}$  und  $k \neq \text{schlüssel}[x]$ 
2   do if  $k < \text{schlüssel}[x]$ 
3     then  $x \leftarrow \text{links}[x]$ 
4     else  $x \leftarrow \text{rechts}[x]$ 
5 return  $x$ 
```



4.1 Grundoperationen

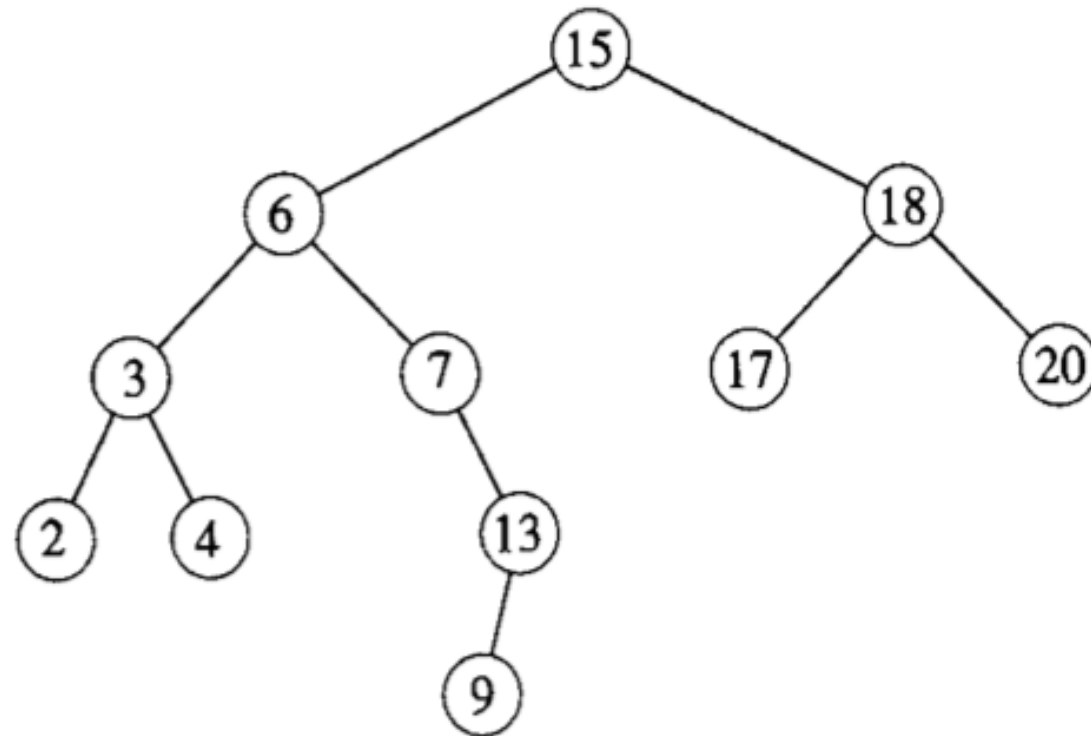
SUCCESSOR(S,x):

“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**

Nachfolger im Suchbaum



TREE-SUCCESSOR(x)

```
1  if  $rechts[x] \neq \text{NIL}$ 
2    then return TREE-MINIMUM( $rechts[x]$ )
3   $y \leftarrow p[x]$ 
4  while  $y \neq \text{NIL}$  und  $x = rechts[y]$ 
5    do  $x \leftarrow y$ 
6     $y \leftarrow p[y]$ 
7  return  $y$ 
```

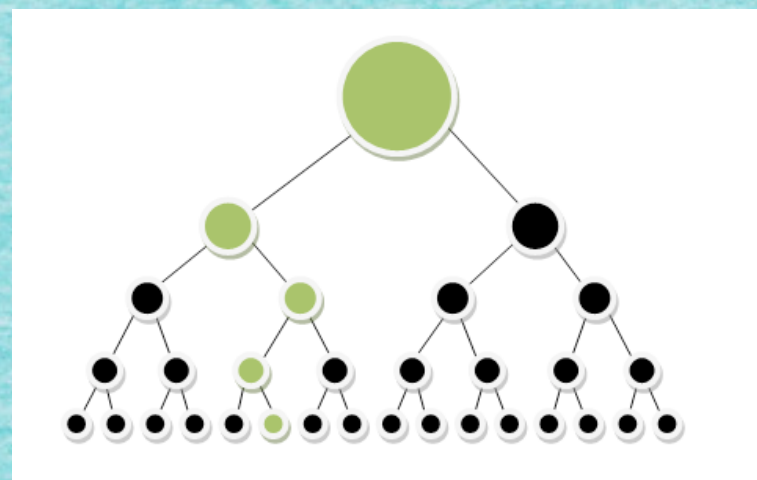

4.5 Binäre Suchbäume

Satz 4.4

Suchen, Minimum, Maximum, Nachfolger, Vorgänger können in einem binären Suchbaum der Höhe h in Zeit $O(h)$ durchlaufen werden.

Beweis:

Klar, der Baum wird nur einmal vertikal durchlaufen!

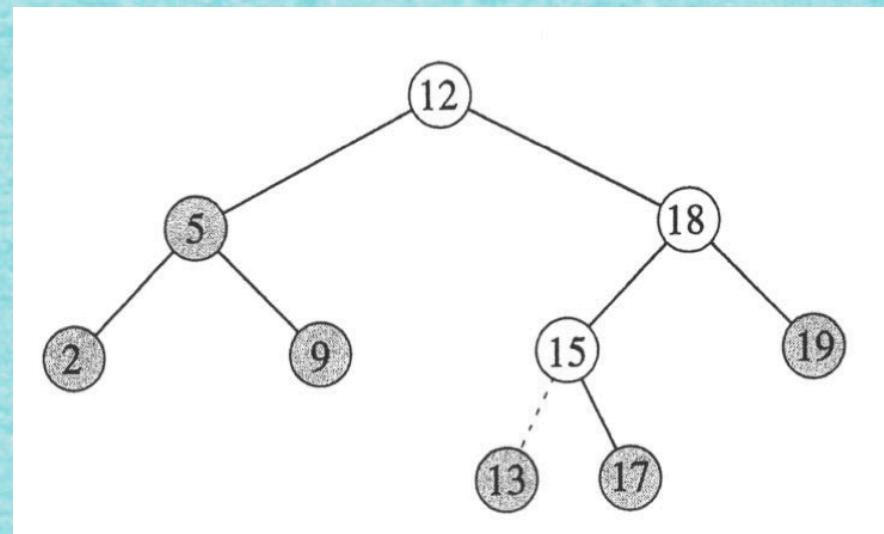


4.1 Grundoperationen

INSERT(S,x): “Füge x in S ein”

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

Einfügen im Suchbaum



TREE-INSERT(T, z)

```
2  $x \leftarrow \text{wurzel}[T]$ 
3 while  $x \neq \text{NIL}$ 
4     do  $y \leftarrow x$ 
5         if  $\text{schlüssel}[z] < \text{schlüssel}[x]$ 
6             then  $x \leftarrow \text{links}[x]$ 
7             else  $x \leftarrow \text{rechts}[x]$ 
8  $p[z] \leftarrow y$ 
9 if  $y = \text{NIL}$ 
10     then  $\text{wurzel}[T] \leftarrow z$ 
11     else if  $\text{schlüssel}[z] < \text{schlüssel}[y]$ 
12         then  $\text{links}[y] \leftarrow z$ 
13         else  $\text{rechts}[y] \leftarrow z$ 
```

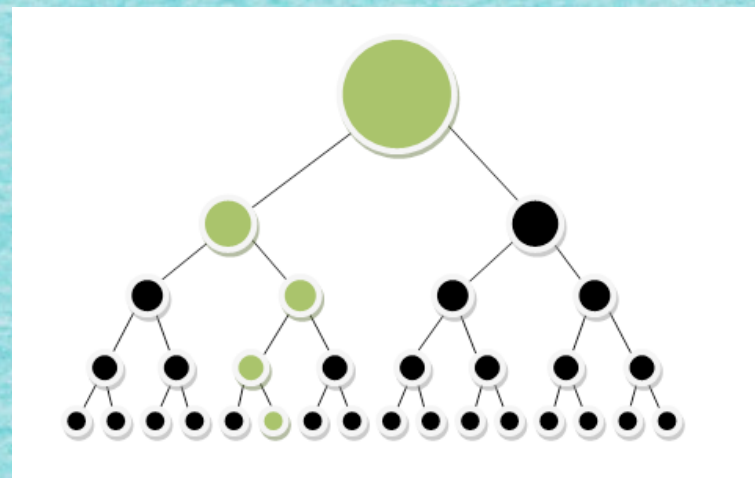

4.5 Binäre Suchbäume

Satz 4.5

Einfügen benötigt $O(h)$ für einen binären Suchbaum der Höhe h .

Beweis:

Klar, der Baum wird nur vertikal abwärts durchlaufen!

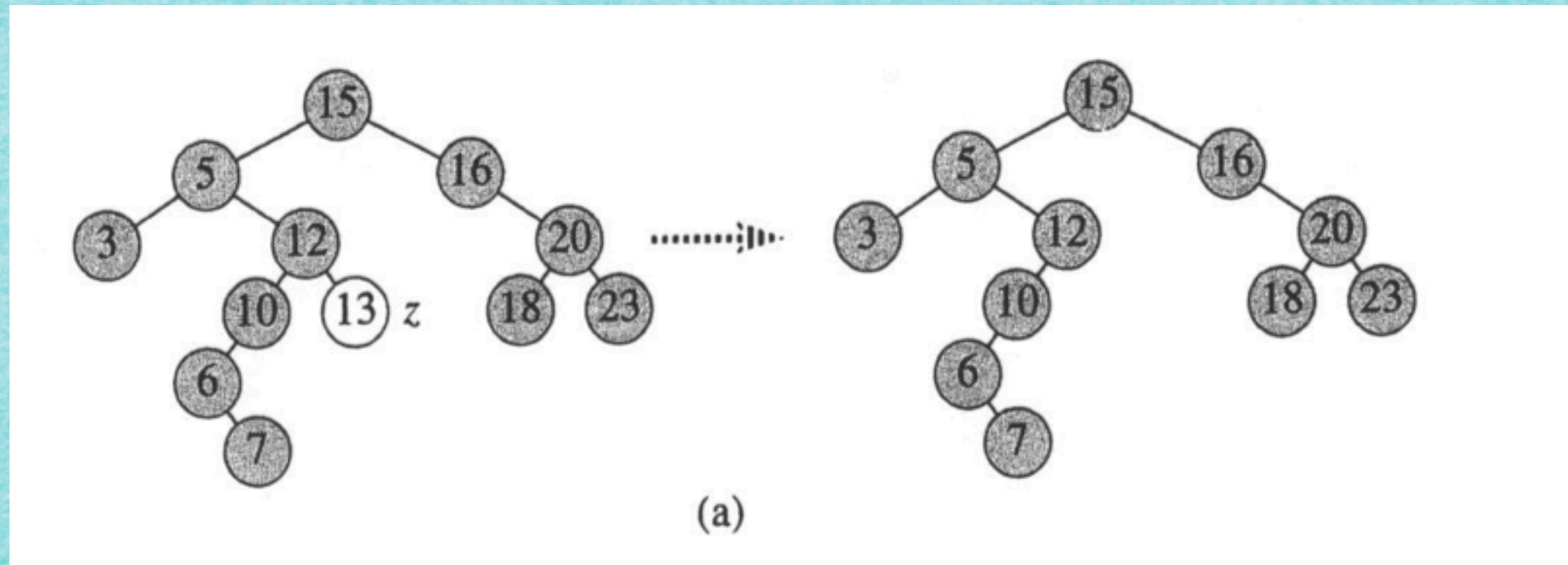


4.1 Grundoperationen

DELETE(S,x): “Entferne x aus S”

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

Löschen im Suchbaum

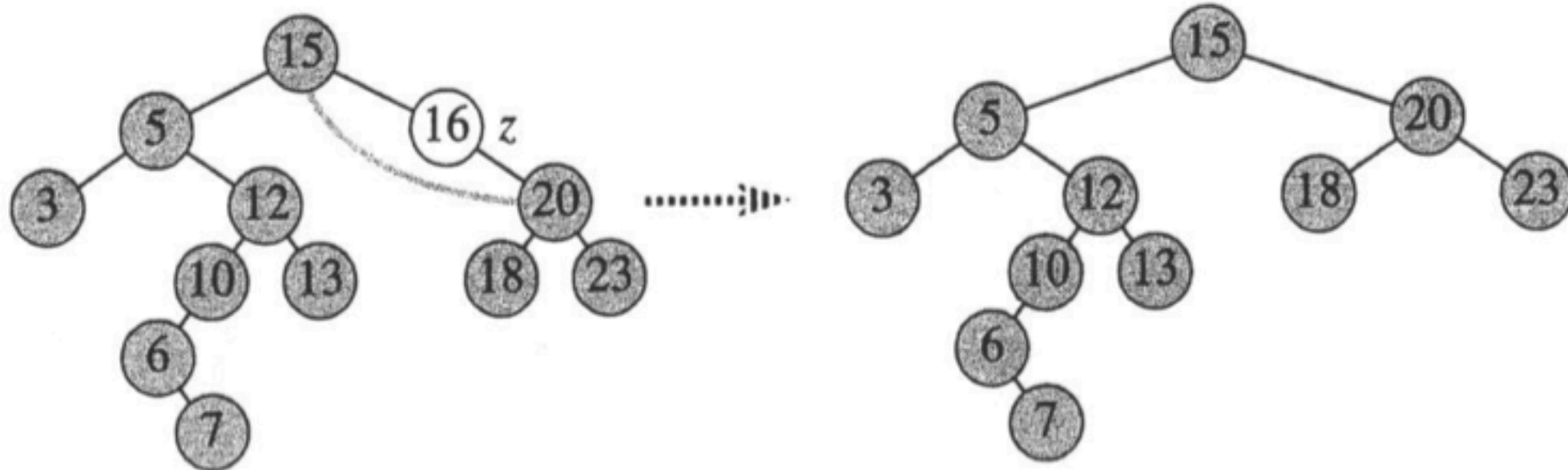


Lösche 13!

(a) Keine Kinder:

Einfach entfernen

Löschen im Suchbaum



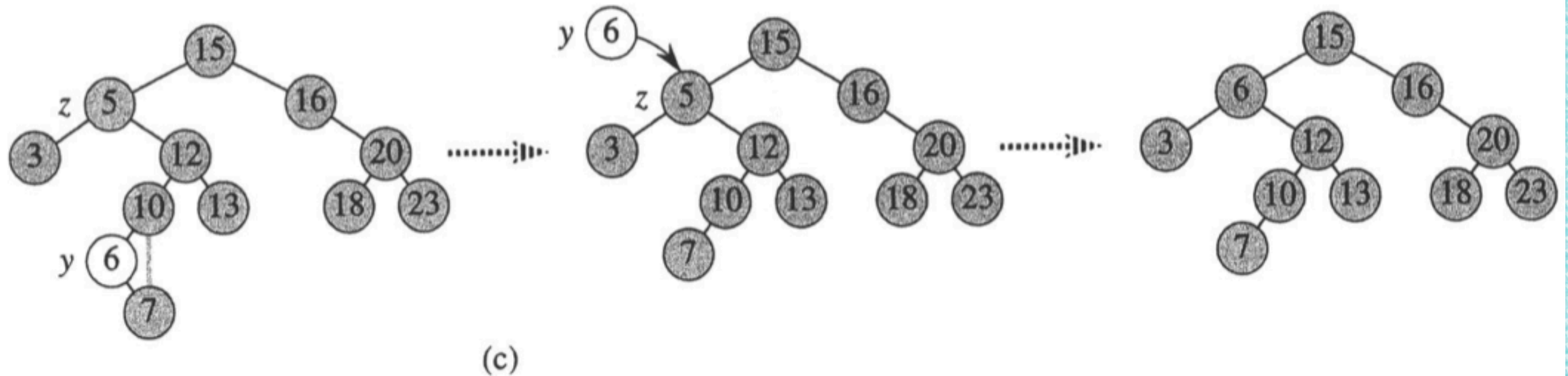
(b)

Lösche 16!

(b) Ein Kind:

“Ausschneiden”

Löschen im Suchbaum



Lösche 5!

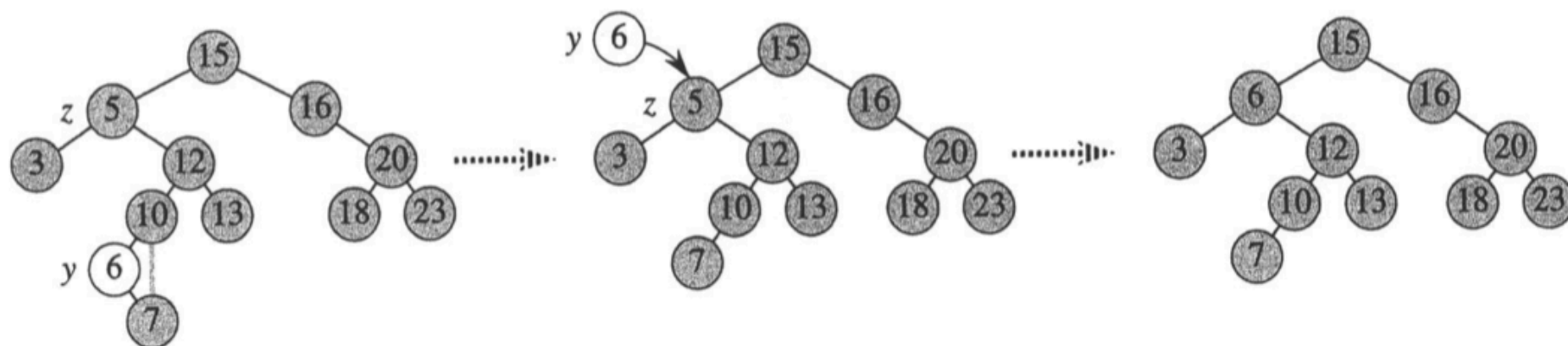
(c) Zwei Kinder:

“Nachfolger verpflanzen”

Löschen im Suchbaum

TREE-DELETE(T, z)

```
1  if  $links[z] = NIL$  oder  $rechts[z] = NIL$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow TREE-SUCCESSOR(z)$ 
4  if  $links[y] \neq NIL$ 
5    then  $x \leftarrow links[y]$ 
6    else  $x \leftarrow rechts[y]$ 
7  if  $x \neq NIL$ 
8    then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = NIL$ 
10   then  $wurzel[T] \leftarrow x$ 
11   else if  $y = links[p[y]]$ 
12         then  $links[p[y]] \leftarrow x$ 
13         else  $rechts[p[y]] \leftarrow x$ 
14  if  $y \neq z$ 
15   then  $schlüssel[z] \leftarrow schlüssel[y]$ 
16         kopiere die Satellitendaten von  $y$  in  $z$ 
17  return  $y$ 
```



(c)

Löschen im Suchbaum

```
TREE-DELETE( $T, z$ )
1  if  $links[z] = NIL$  oder  $rechts[z] = NIL$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow TREE-SUCCESSOR(z)$ 
4  if  $links[y] \neq NIL$ 
5    then  $x \leftarrow links[y]$ 
6    else  $x \leftarrow rechts[y]$ 
7  if  $x \neq NIL$ 
8    then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = NIL$ 
10   then  $wurzel[T] \leftarrow x$ 
11   else if  $y = links[p[y]]$ 
12         then  $links[p[y]] \leftarrow x$ 
13         else  $rechts[p[y]] \leftarrow x$ 
14  if  $y \neq z$ 
15    then  $schlüssel[z] \leftarrow schlüssel[y]$ 
16         kopiere die Satellitendaten von  $y$  in  $z$ 
17  return  $y$ 
```


4.5 Binäre Suchbäume

Satz 4.6

Löschen benötigt $O(h)$ für einen binären Suchbaum der Höhe h .

Beweis:

Klar, der Baum wird i.W. nur einmal durchlaufen!

4.5 Binäre Suchbäume

Schnell:

- $O(\log n)$: *logarithmische Zeit*
- $O(h)$: *Tiefe des Baumes*

Also: Wie können wir die Tiefe des Baumes auf $O(\log n)$ beschränken?

Ab an die Tafel!

s.fekete@tu-bs.de

Mehr demnächst!

s.fekete@tu-bs.de