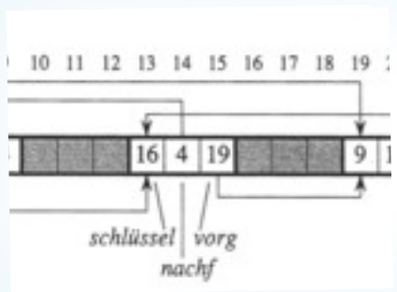
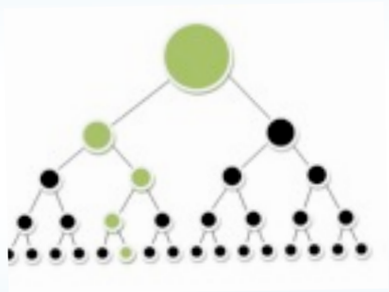




Kapitel 4: Dynamische Datenstrukturen

Algorithmen und Datenstrukturen
WS 2015/16

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

<u>SEARCH(S,k):</u> **“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

<u>INSERT(S,x):</u> **“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

<u>MINIMUM(S):</u> **“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

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

<u>PREDECESSOR(S,x):</u>
“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

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

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

Langsam:

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

  Alle Objekte anschauen

Sehr schnell:

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

  Immer gleich schnell, egal wie groß S ist.

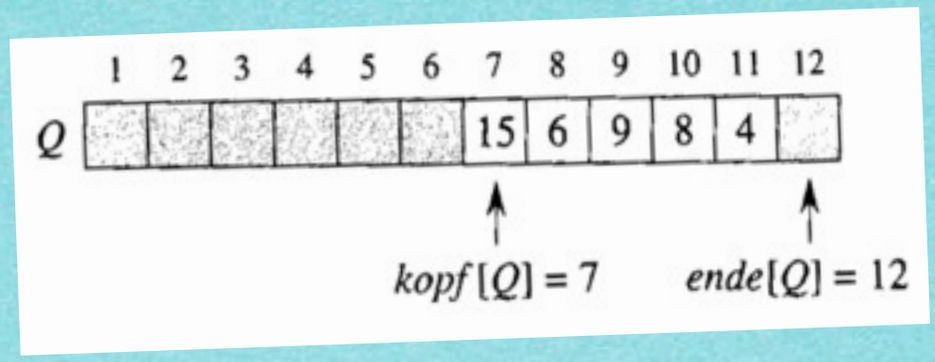
Schnell:

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

  Wiederholtes Halbieren

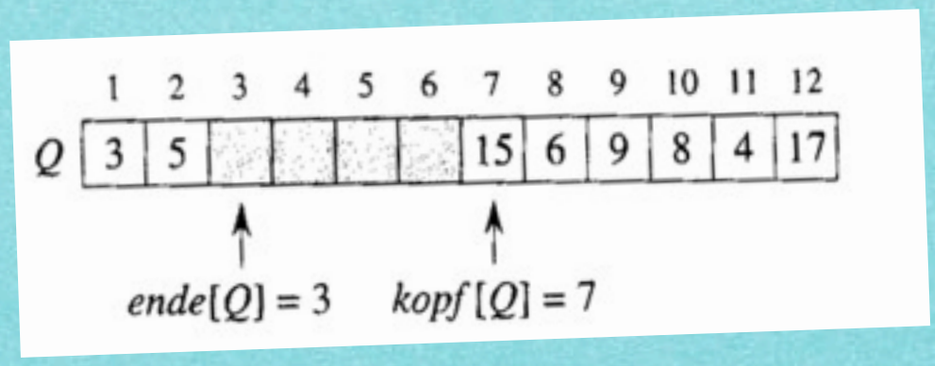
4.2 Stapel und Warteschlange

OUT

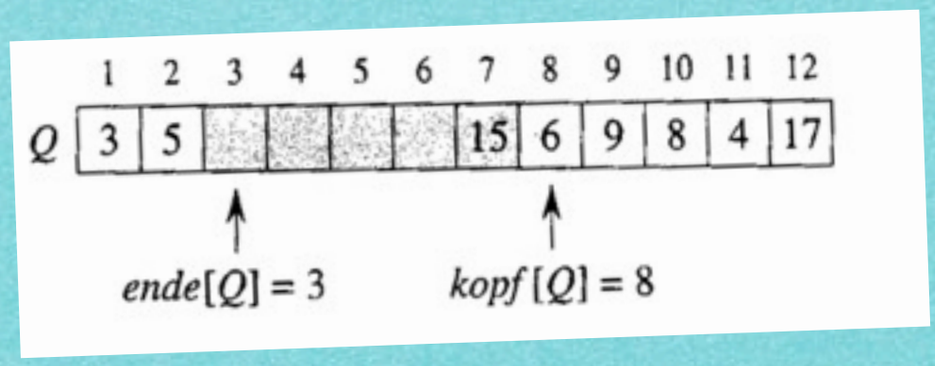


IN

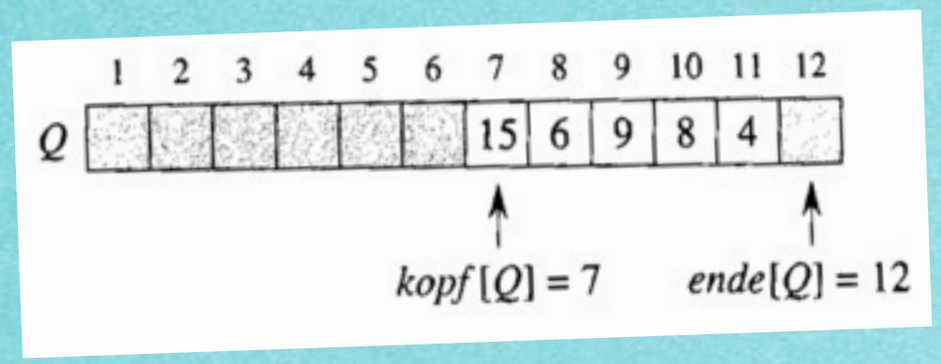
Enqueue: 17, 3, 5

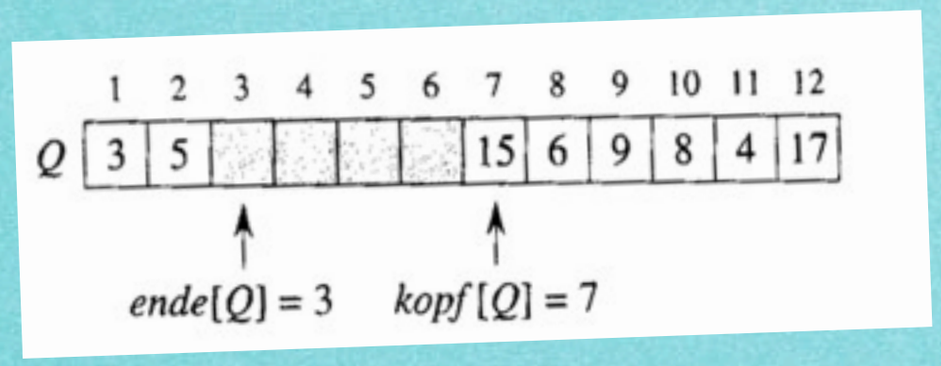


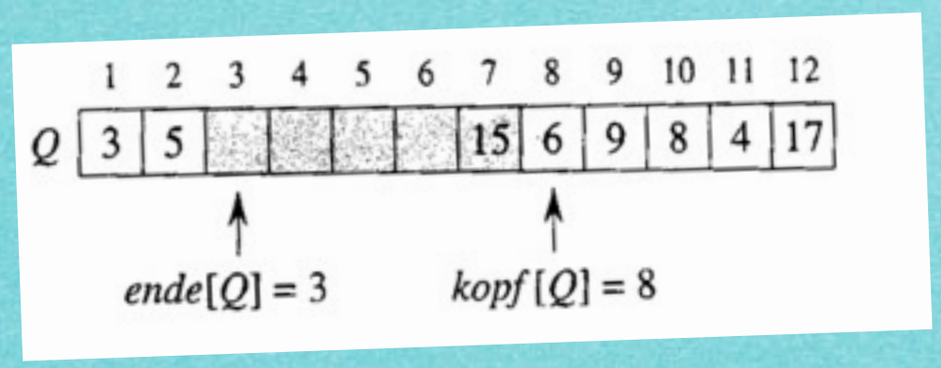
Dequeue:



Warteschlange auf Array umgesetzt



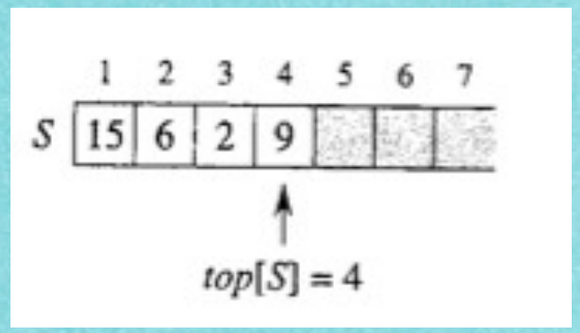


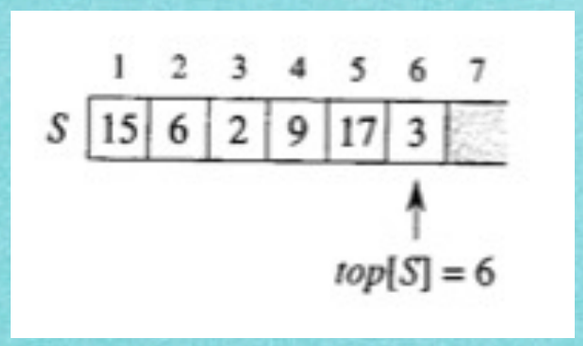


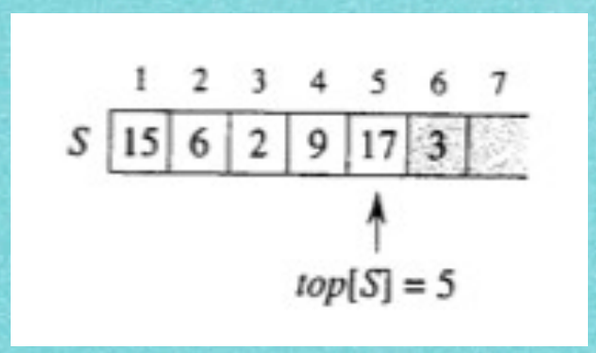

```
ENQUEUE(Q, x)
1  Q[ende[Q]] ← x
2  if ende[Q] = länge[Q]
3     then ende[Q] ← 1
4     else ende[Q] ← ende[Q] + 1
```

```
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
```

Stack
auf Array umgesetzt

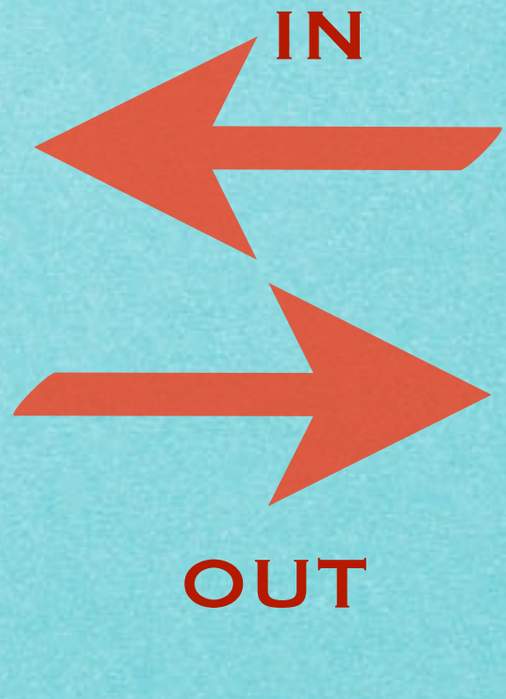




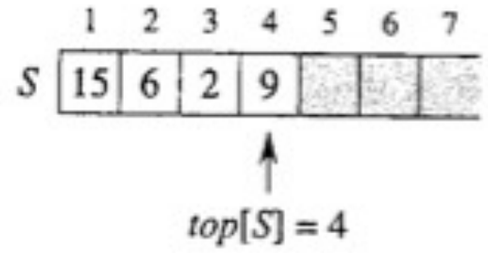


Push: 17, 3

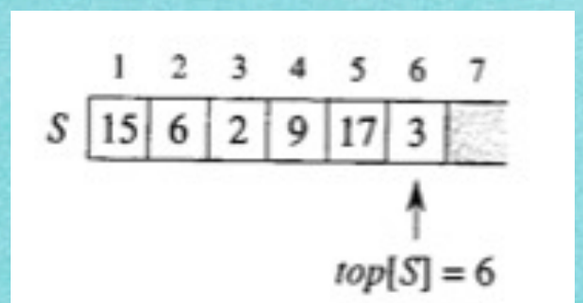
Pop



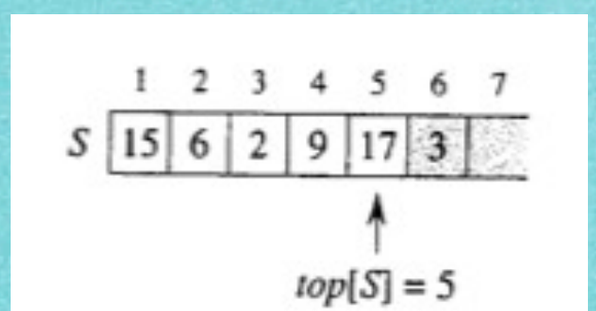
Stack
auf Array umgesetzt



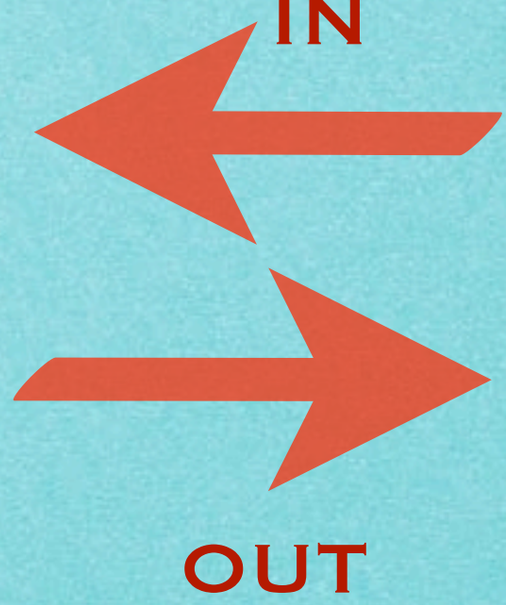

```
STACK-EMPTY(S)
1  if top[S] = 0
2      then return WAHR
3      else  return FALSCH
```




```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```




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

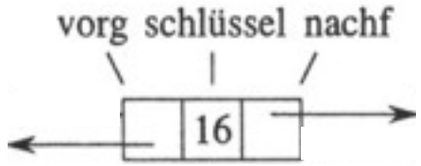


4.3 Verkettete Listen

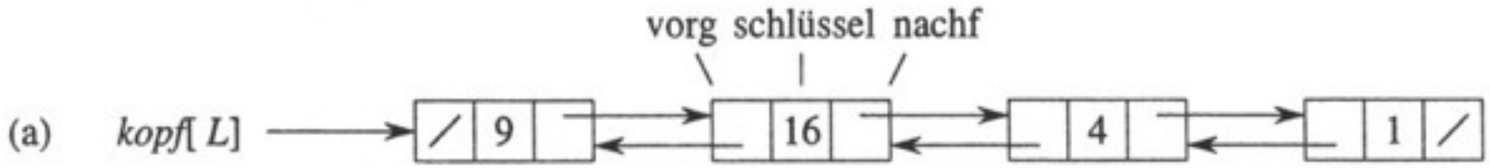
Idee:

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

Struktur einer doppelt verketteten Liste

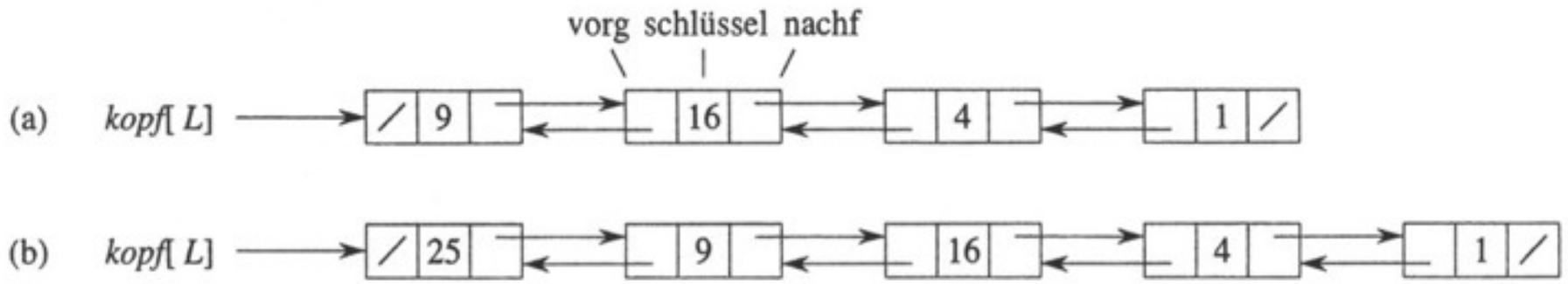


Struktur einer doppelt verketteten Liste



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

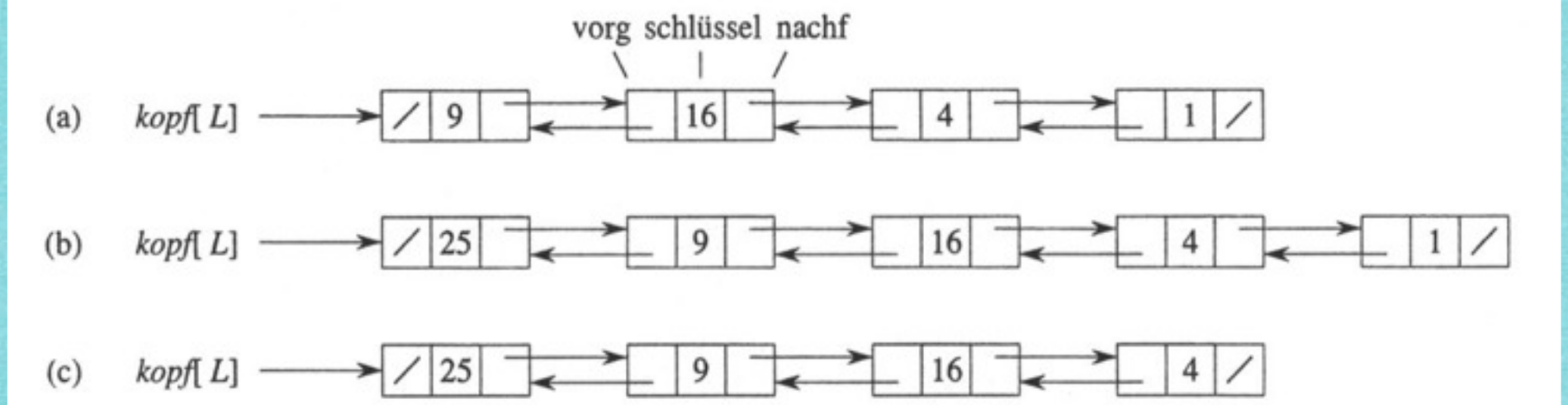
Struktur einer doppelt verketteten Liste



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

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

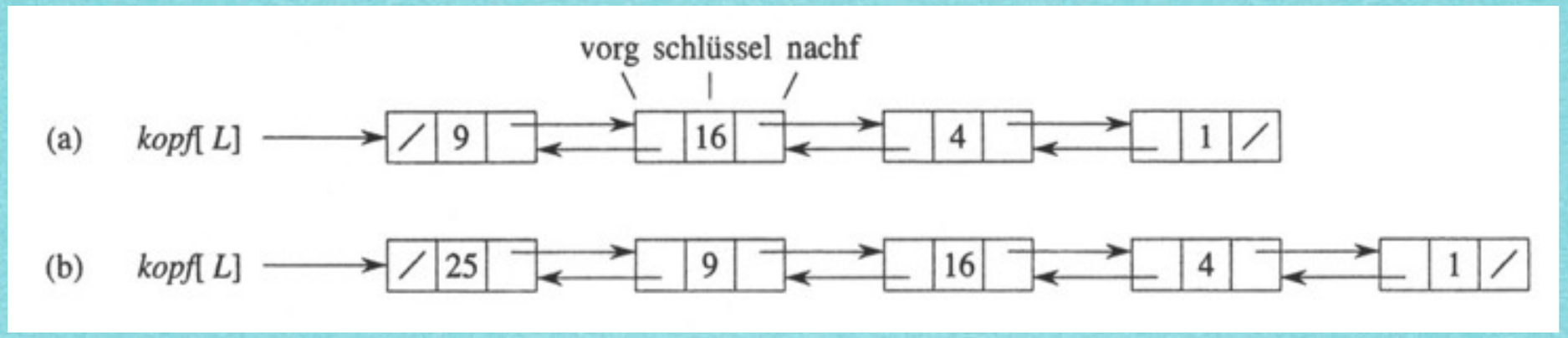
Struktur einer doppelt verketteten Liste



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

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

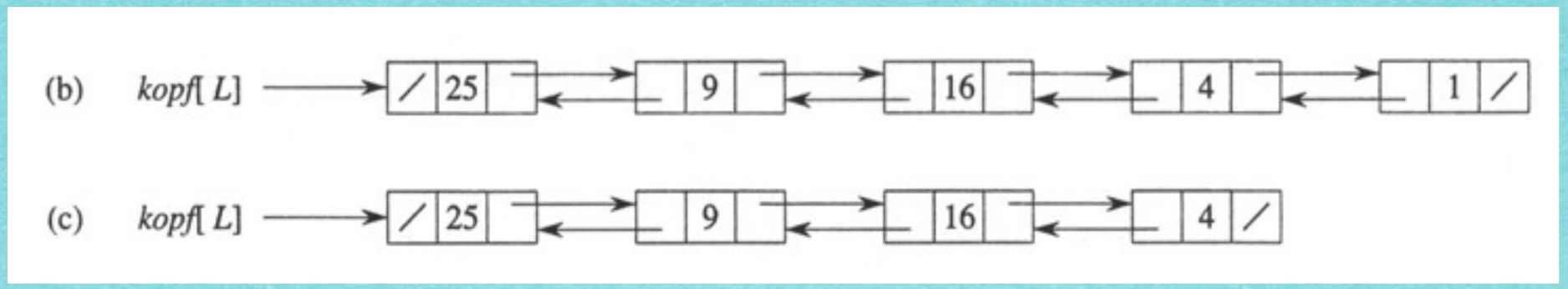
Einfügen in eine doppelt verkettete Liste




```
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



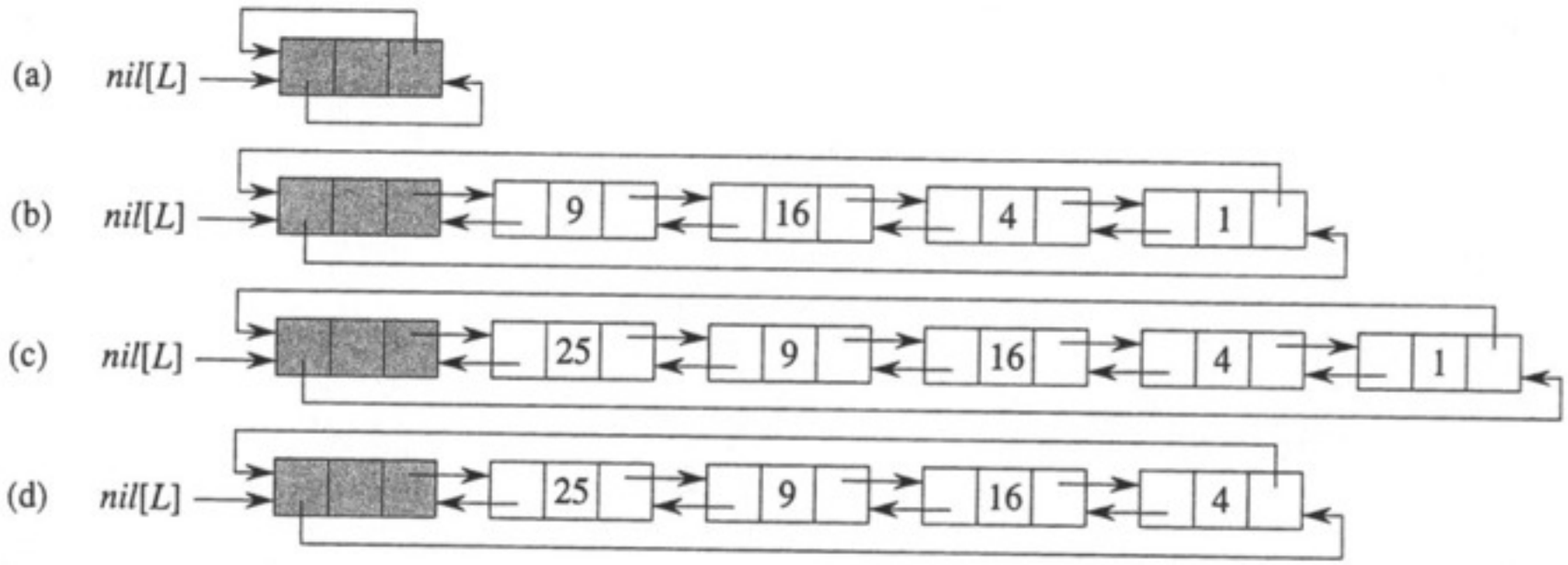
Laufzeit: O(n)

```
LIST-SEARCH(L, k)
1  x ← kopf[L]
2  while x ≠ NIL und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

Laufzeit: O(1)

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

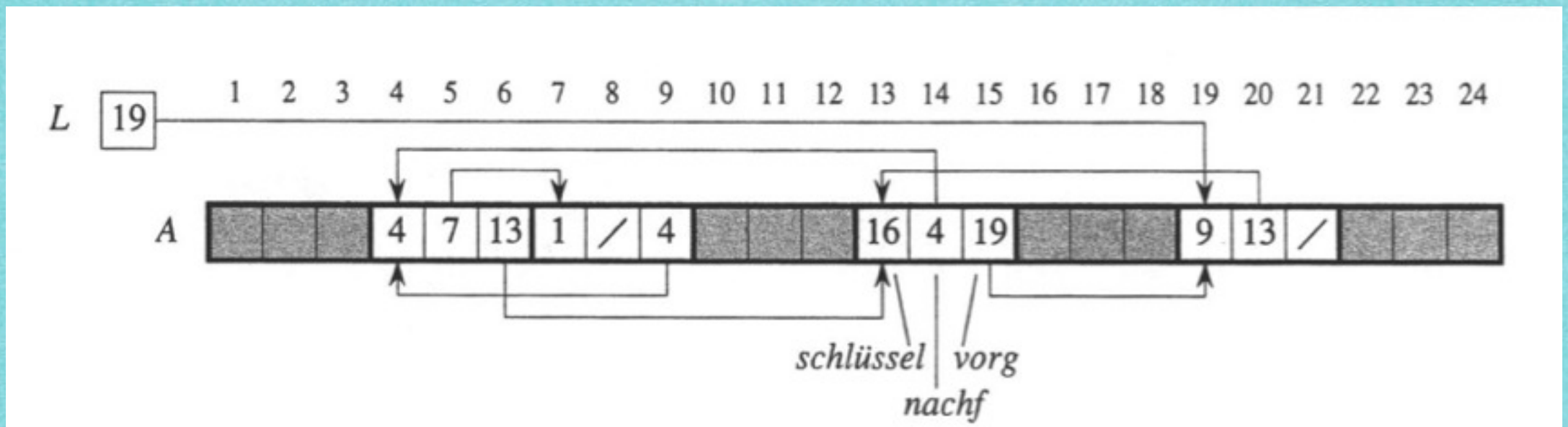
Alternative: Zyklische Struktur mit Wächter "nil[L]"

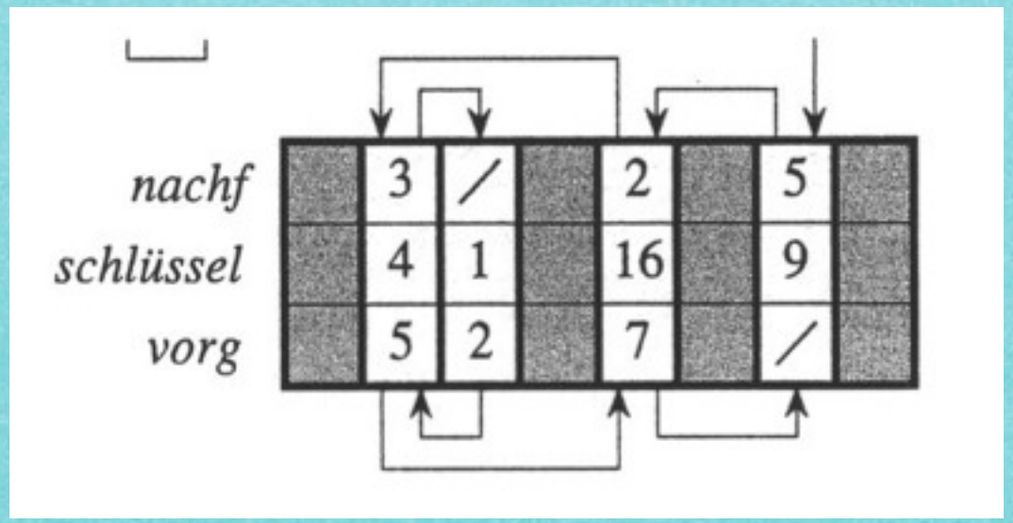



```
LIST-SEARCH'(L, k)
1  x ← nachf[nil[L]]
2  while x ≠ nil[L] und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

```
LIST-DELETE'(L, x)
1  nachf[vorg[x]] ← nachf[x]
2  vorg[nachf[x]] ← vorg[x]
```

Speicherung kann irgendwo erfolgen!

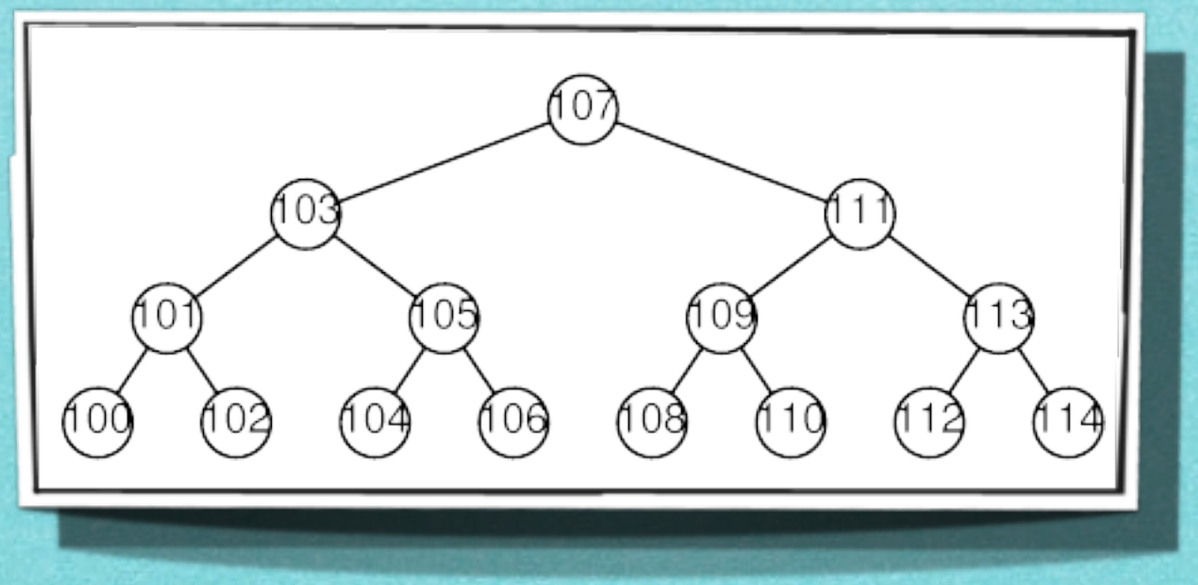




4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***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

    }

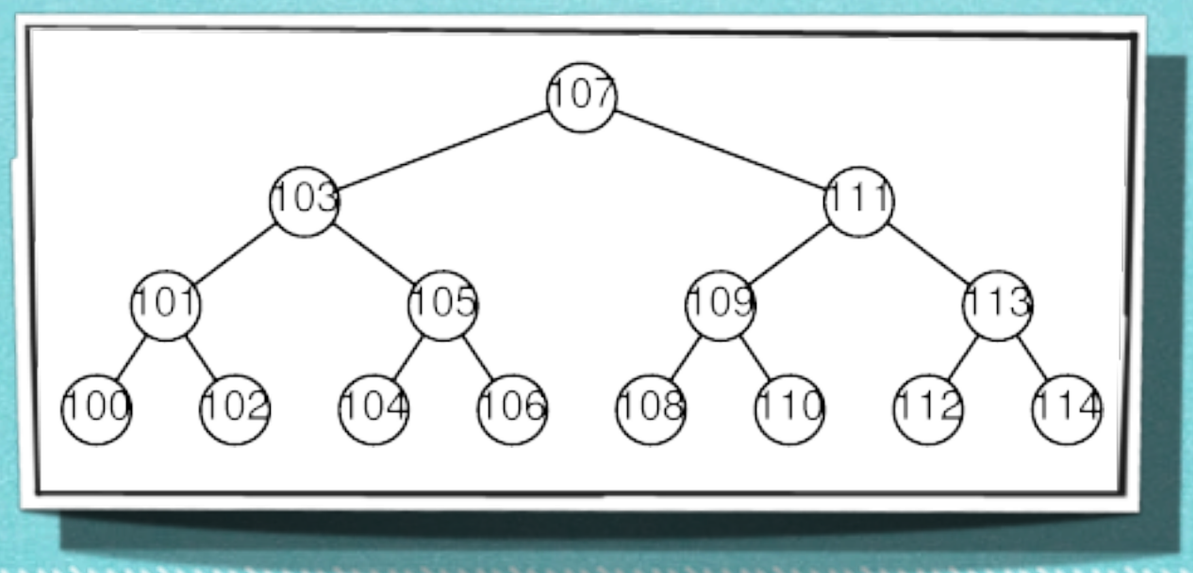
2. RETURN "WERT nicht gefunden!"

Satz 4.2

Die binäre Suche terminiert in O(log(RECHTS-LINKS)) Schritten (für RECHTS>LINKS).

Beweis:

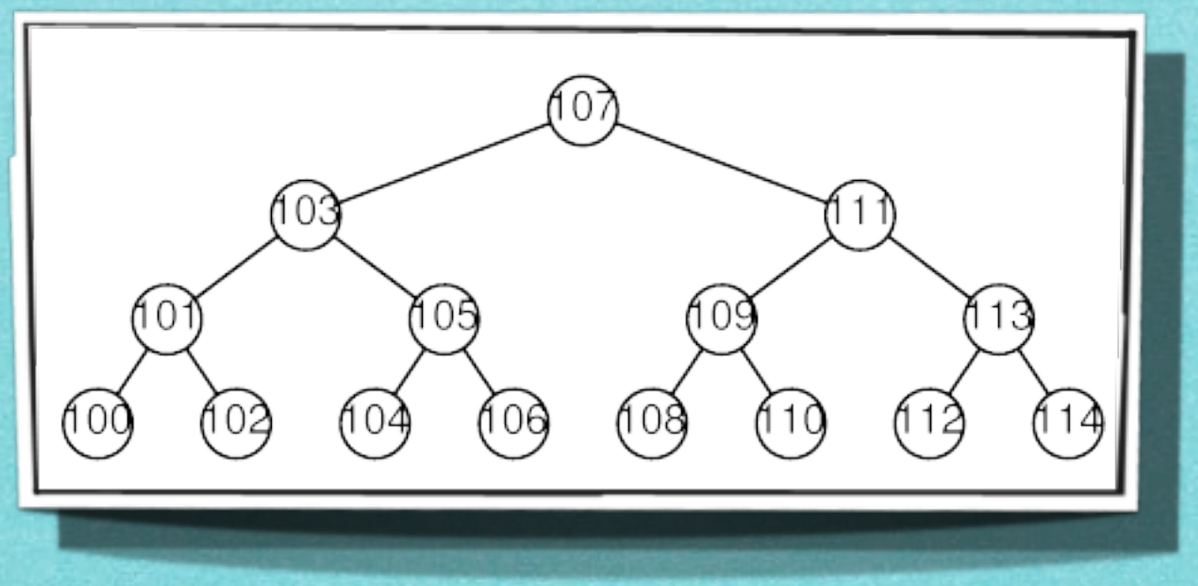
Selbst!



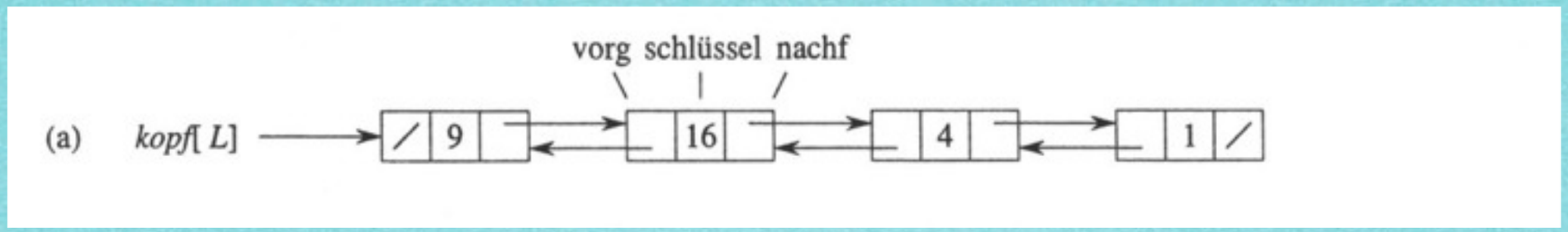
4.5 Binäre Suchbäume

Ideen:

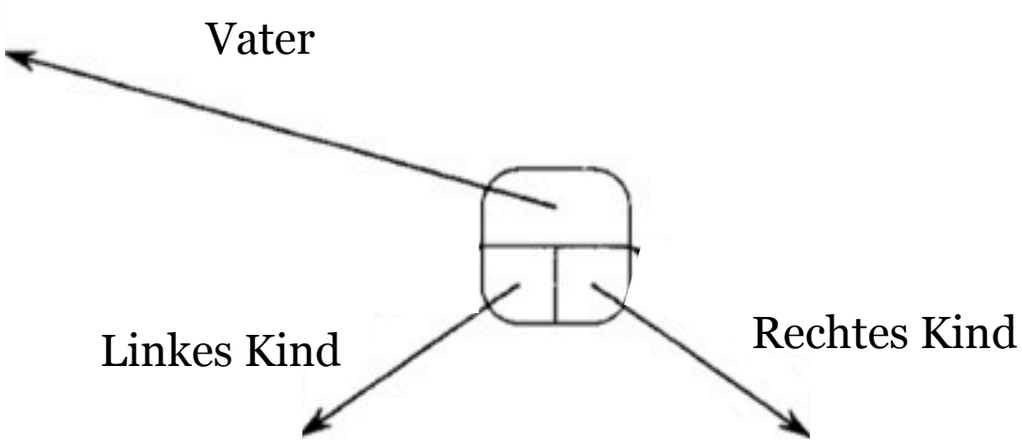
- ***Strukturiere Daten wie im möglichen Ablauf einer binären Suche!***
- ***Erziele logarithmische Zeiten!***



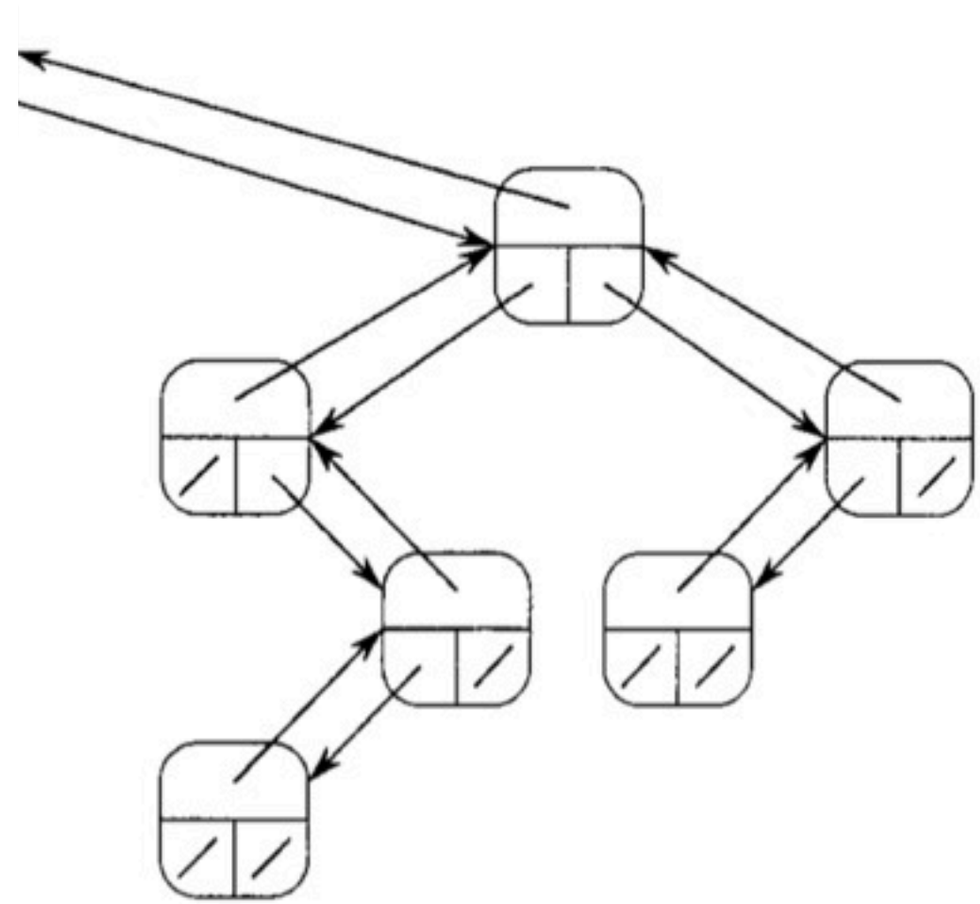
Struktur einer doppelt verketteten Liste



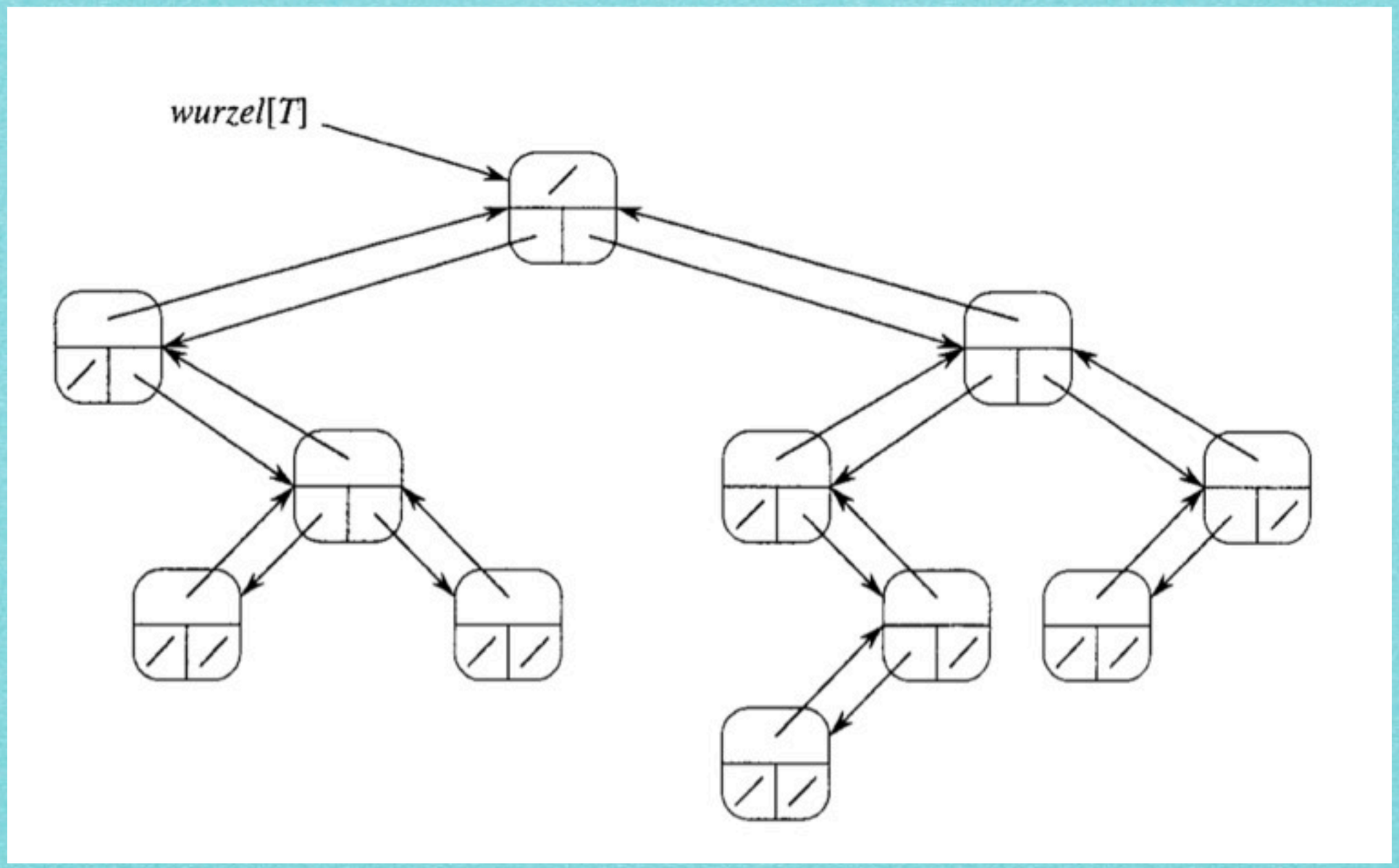
Binärer Suchbaum



Binärer Suchbaum

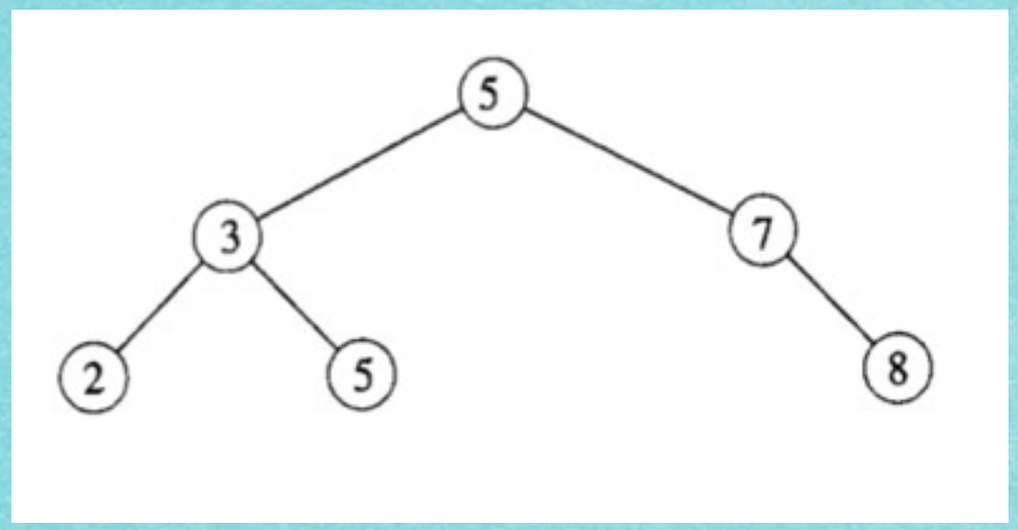


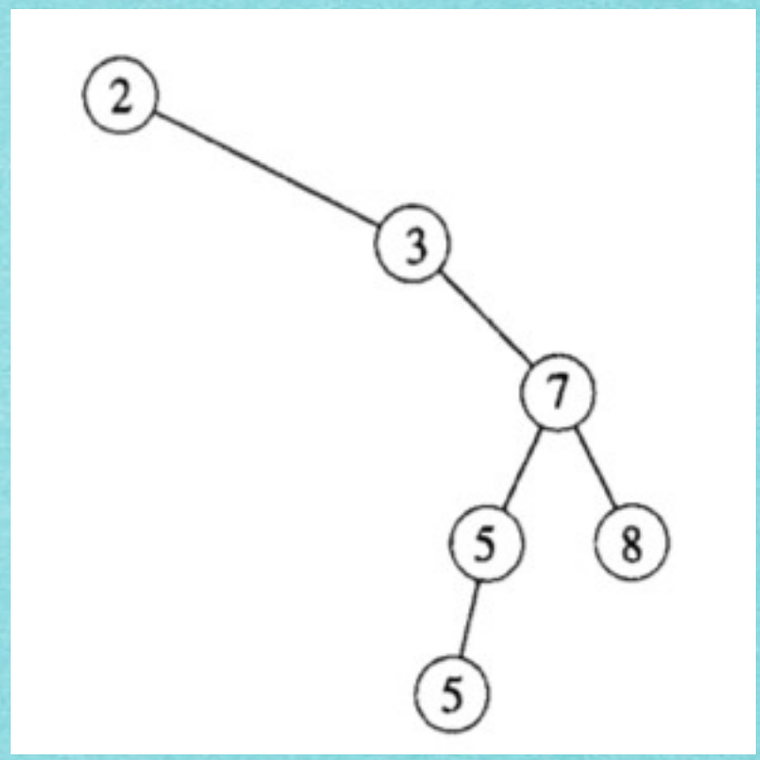
Binärer Suchbaum



Außerdem wichtig: Struktur der Schlüsselwerte!

Ordnungsstruktur





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

Mehr demnächst!

s.fekete@tu-bs.de