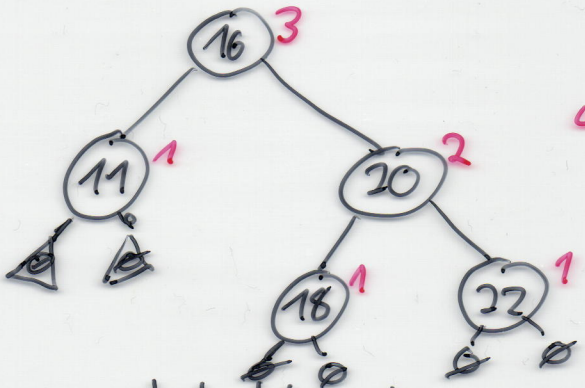
AVL-Bäume

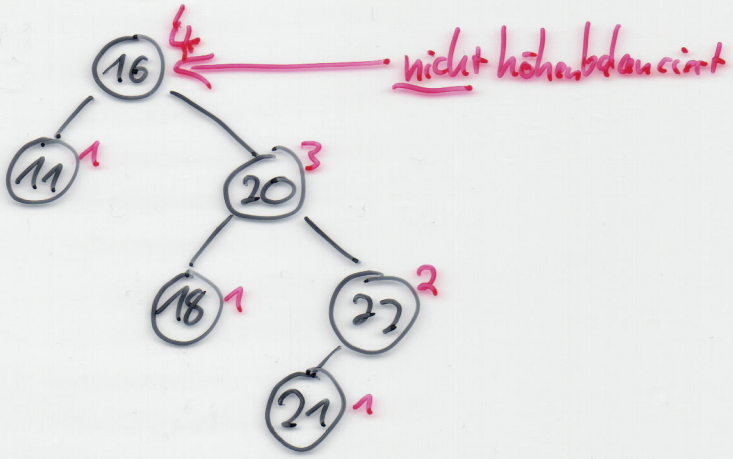
Höhe (v) = # Knoten auf einem längsten Pfad von v zu einem Blatt.

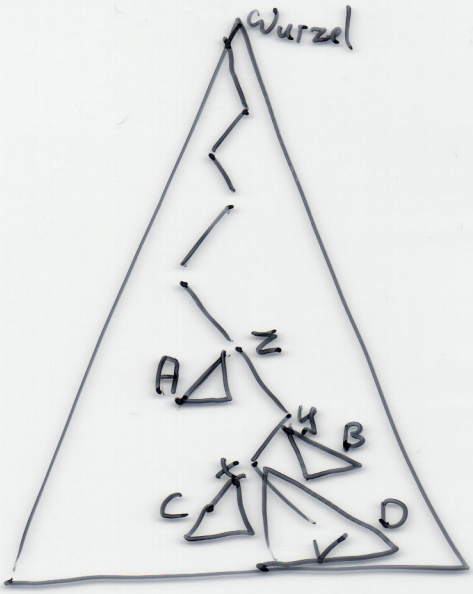
Bsp:



Ein binärer Suchbaum ist höhenbalanciert, wenn sich für jeden inneren Knoten die Höhe seiner Kinder um höchstens 1 unterscheidet.

21 Einfügen:





Allgemein

v: eingefügter Knoten

z: niedrigster unbalancierter Vorfahre von v

y: Kind von z und Vorfahre von v

x: Kind von y " " " "

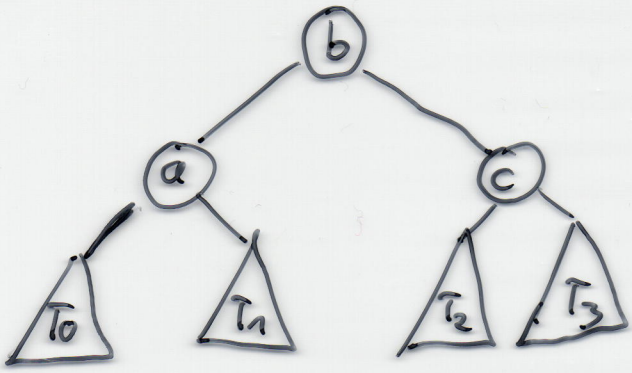
A,B,C,D: „Die anderen" Teilbäume unter z/y sowie beide unter x.

Achtung, die Teilbäume können leer sein

Sei $a<b<c$ die Sortierung von x,y,z

$T_0<\dots<T_3$ " " " A,B,C,D

Jetzt restrukturiere unter z:

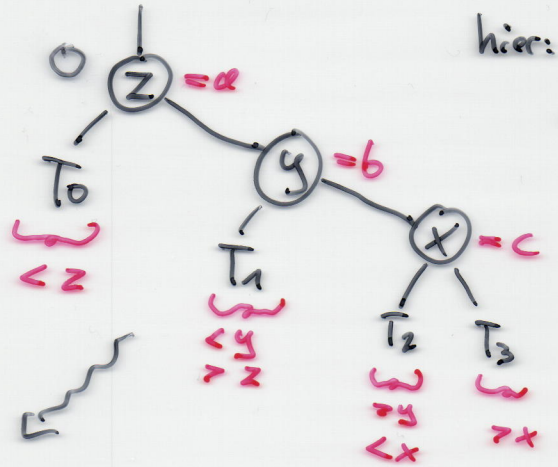


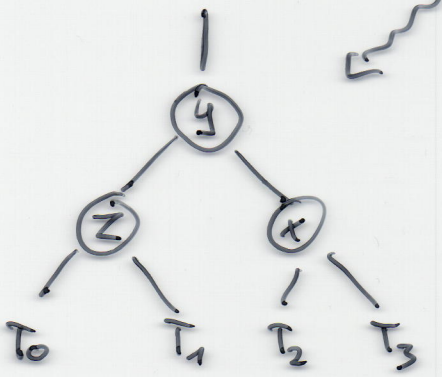
Bsp 1:

hier: $z<y<x$

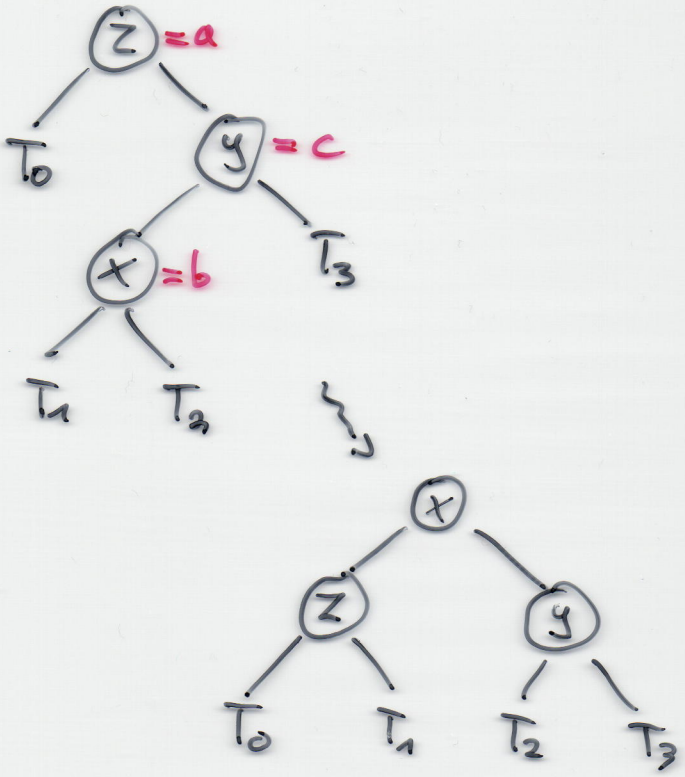
$\|\quad\|\quad\|$

$a\quad b\quad c$

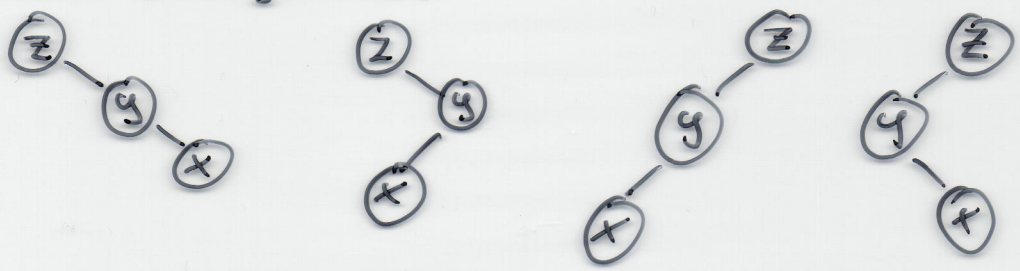




Bsp 2:

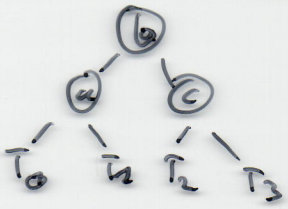


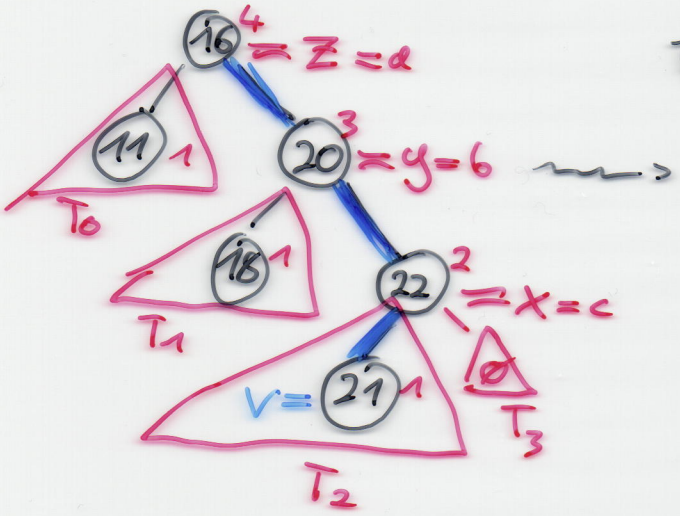
Wie viele Fälle gibt es?

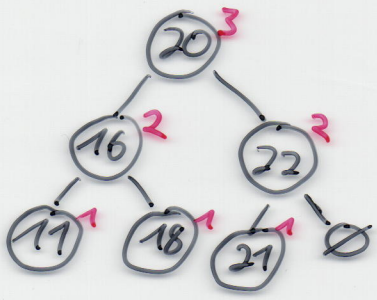


= 4

Zurück zum Anfangsbsp.:







Rekursionen

Funktionen dürfen andere Funktionen aufrufen, auch sich selbst!

Bsp.:

```
function printTree (T: Tree) begin
  if T = (L, x, R) then // T ist nicht leer:
    print(x)
    printTree(L)
    printTree(R)
  fi
end
```

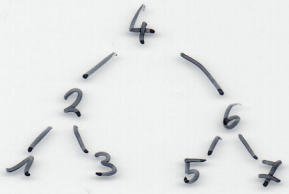
preorder

```
printTree (L)
print(x)
printTree (R)
```

inorder

```
printTree (L)
printTree (R)
print(x)
```

postorder



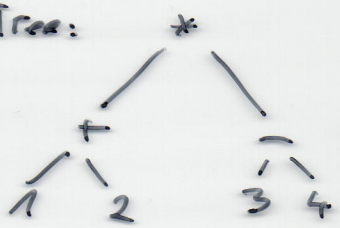
4(2 1 3)(6 5 7)

(1 2 3)4(5 6 7)

(1 3 2)(5 7 6)4

Wozu inorder? Sortierte Ausgabe/Traversion eines Binärbaums

" pre-order? Expression Tree:



*(+(1,2), -(3,4))

mult(add(1,2), diff(3,4))

Wozu postorder? Umgekehrte polnische Notation! (1+2)·3

1 ↵ 2 ↵ + 3 ↵ ·

Häufiges Muster bei Rekursion

(a) Kleine Probleminstanz → Direkt lösen

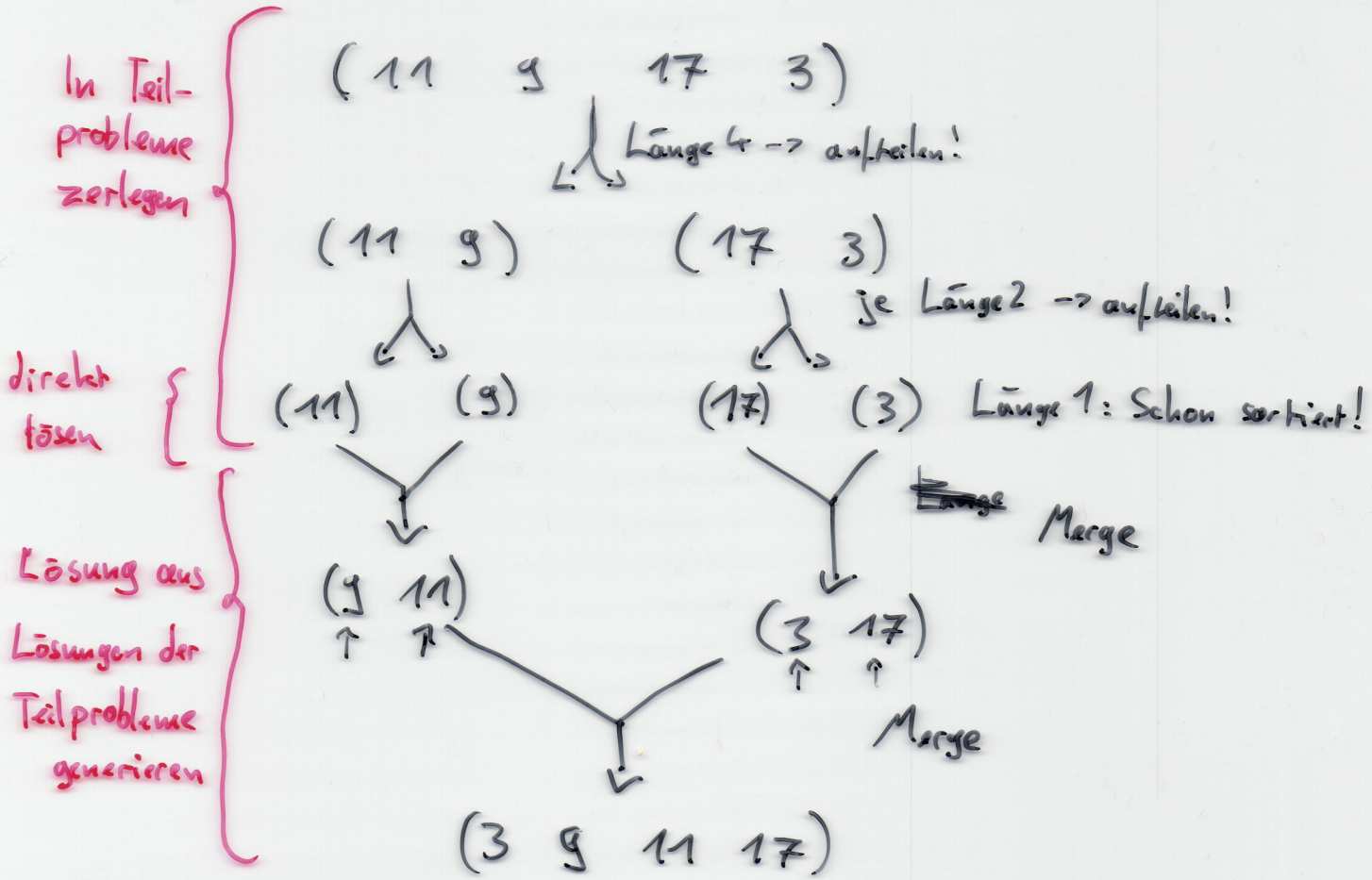
(b) Große " → Aufteilen in kleine (divide)
Rekursiv lösen (conquer)
Zusammenfügen (merge)

} Divide & Conquer

Bsp. Mergesort

(a) Felder der Länge 0 oder 1 → sind schon sortiert

(b) " " " ≥ 2 → Aufteilen in 2 Felder halber Länge
Rekursiv Teilfelder sortieren
Zusammenfügen sortierter Teilfelder



Bsp: Prüfe, ob Baum Binärbaum ist!

```
function checkTree(T:Tree) begin
    return checkTree(T, -∞, ∞)
end

function checkTree(T:Tree, lb, ub) begin
    if T = (L, x, R) then
        if x < lb ∨ x > ub then
            return FALSE
        fi
        return checkTree(L, lb, x) ∧ checkTree(R, x, ub)
    else
        return TRUE        // T ist leer
    fi
end
```

→ alles in T muss in]lb, ub[liegen