AVL TREES, AVL SORT

BALANCED BSTs

- the importance of being balanced
- AVL trees
  - definition
  - rotations
  - insert
- other balanced trees
- data structures in general

(- lower bounds)

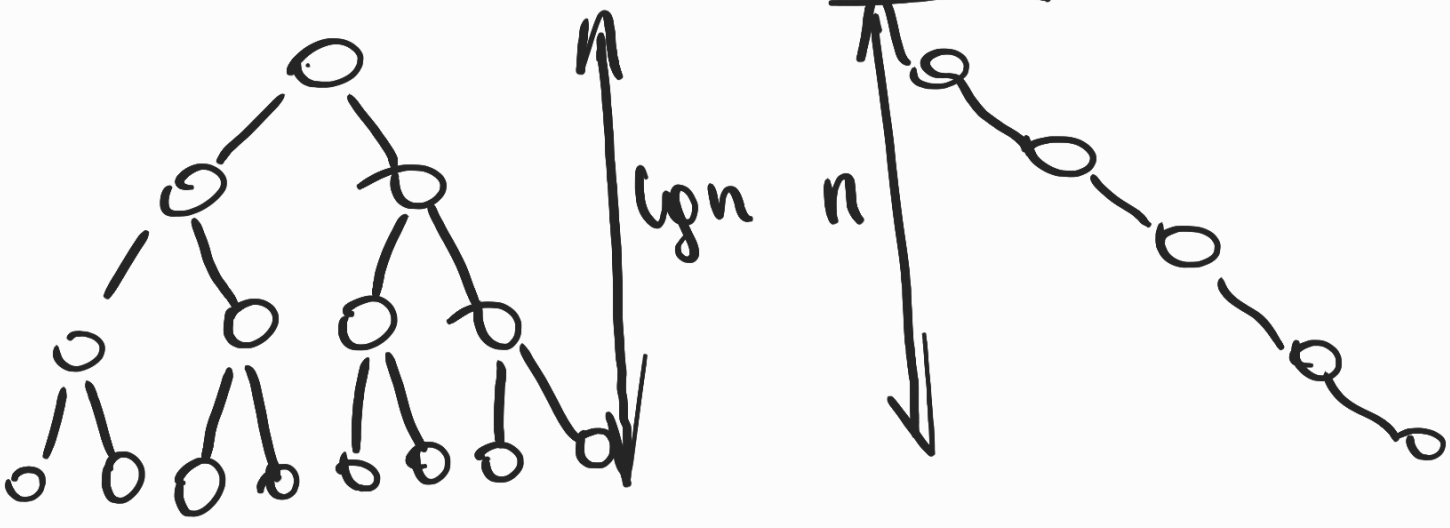
Recall:

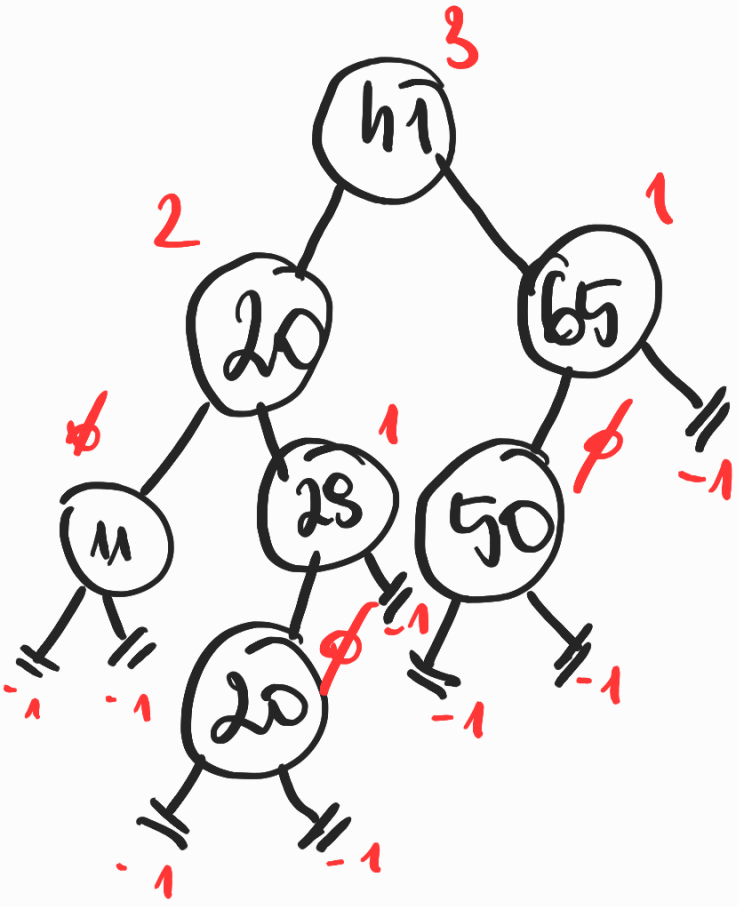
Binary Search Trees (BSTs)

- rooted binary tree
- each node has
  - key
  - left pointer
  - right pointer
  - parent pointer

- BST property

Balanced or not :





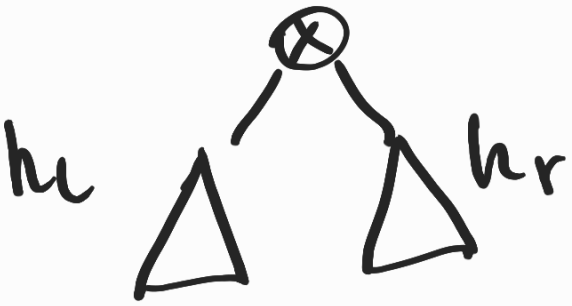
height of a node
= length of longest path from it down to a leaf
= max (height left child, height right child)

AVL TREES

require heights of left &

might children of every node to differ by at most ±1

$$|h_L - h_r| \leq 1$$



AVL TREES ARE BALANCED

worst case is when right subtree has height one more than left for every node.

$N_{O(1)} = O(1)$

$N_h = 1 + N_{h-1} + N_{h-2}$

right left

almost Fibonacci

$N_h > F_h$

$F_h = \varphi^h / \sqrt{5}$

$\varphi > 1$

$\frac{\varphi^h}{\sqrt{5}} < n$

$h - - < \log_\varphi n$

$\approx 1.440 \log n$

$N_h = 1 + N_{h-1} + N_{h-2}$

$N_h > 1 + 2N_{h-2}$

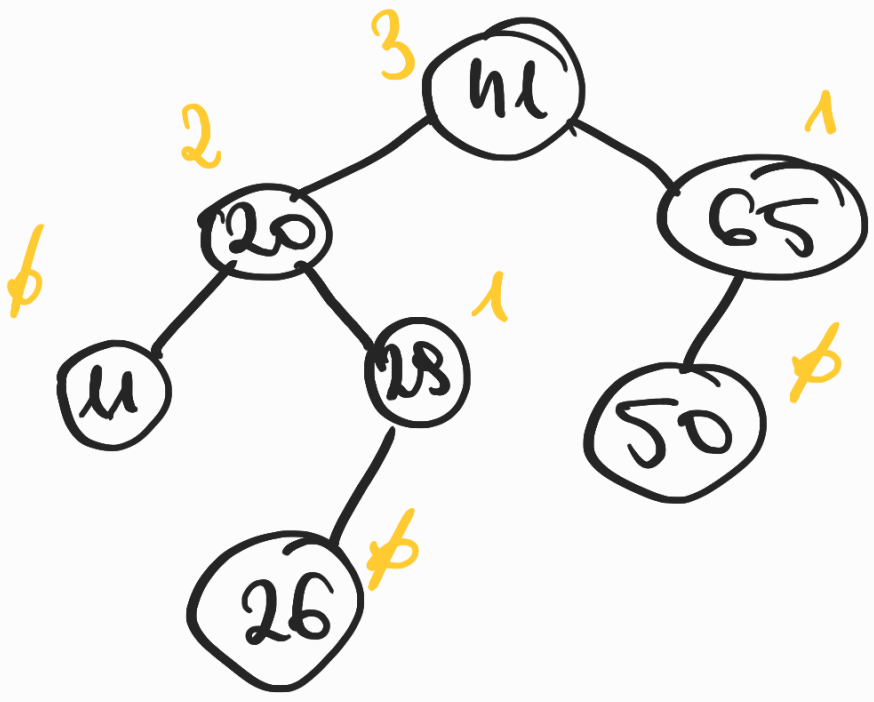
$N_h \geq 2N_{h-2}$

$= \Theta(2^{h/2})$

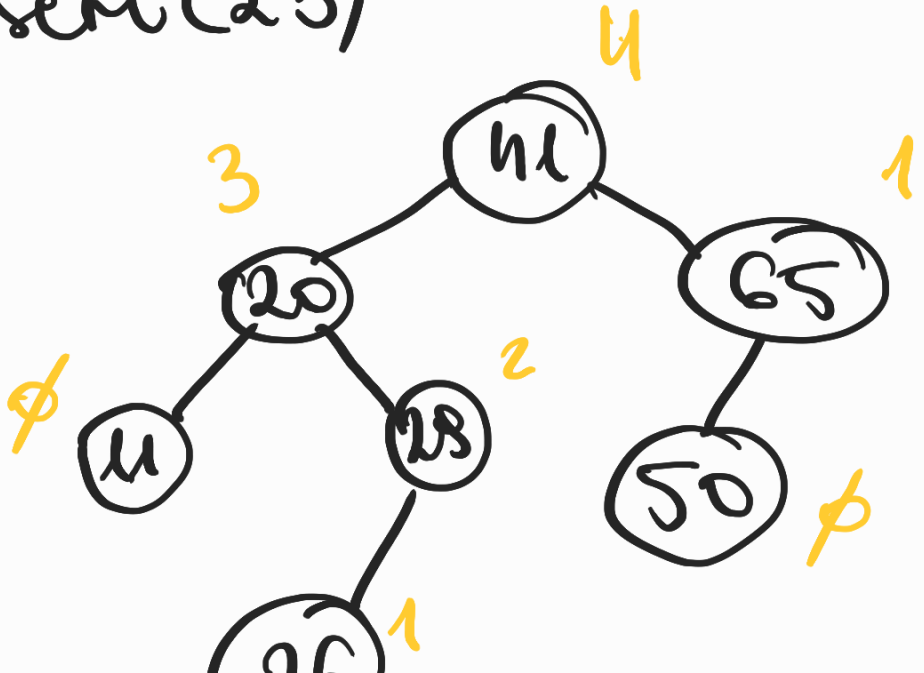
$h < 2 \lg n$

AVL INSERT

1) Simple BST insert
2) fix AVL property

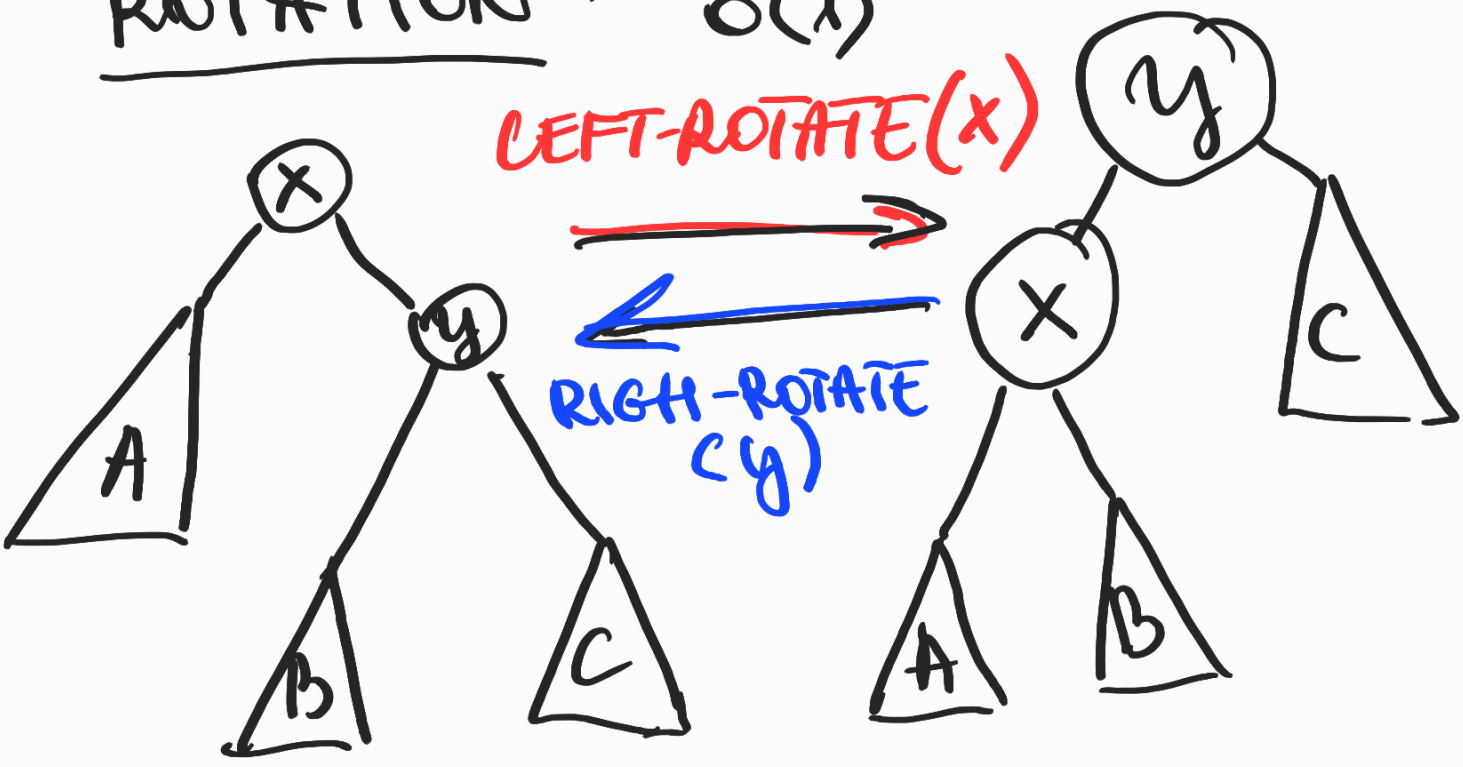


Insert(23)

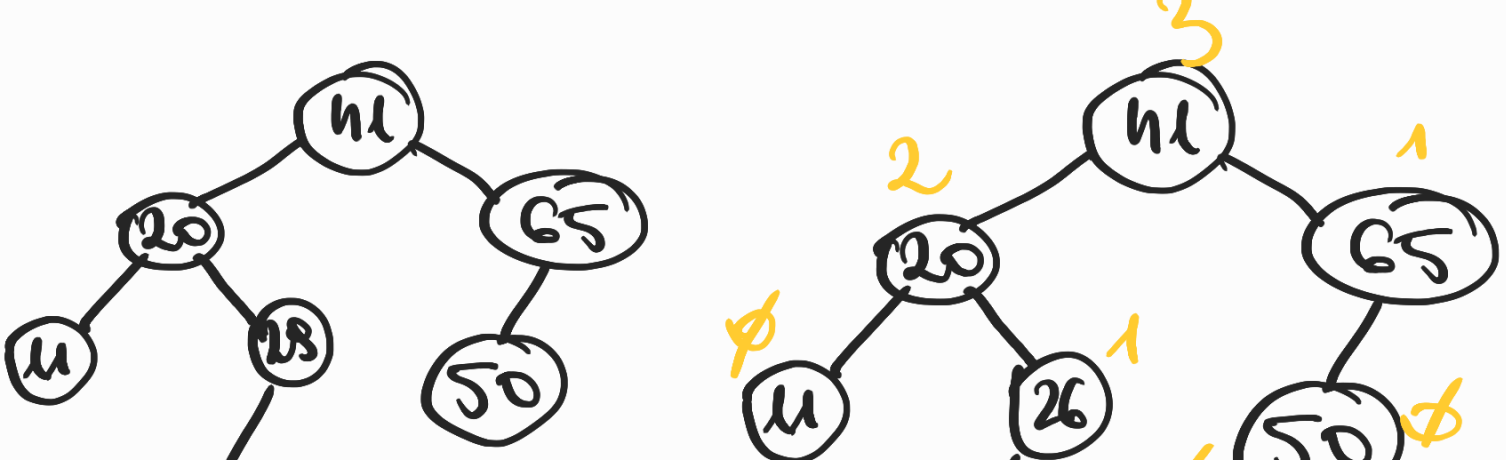



26
23
∅

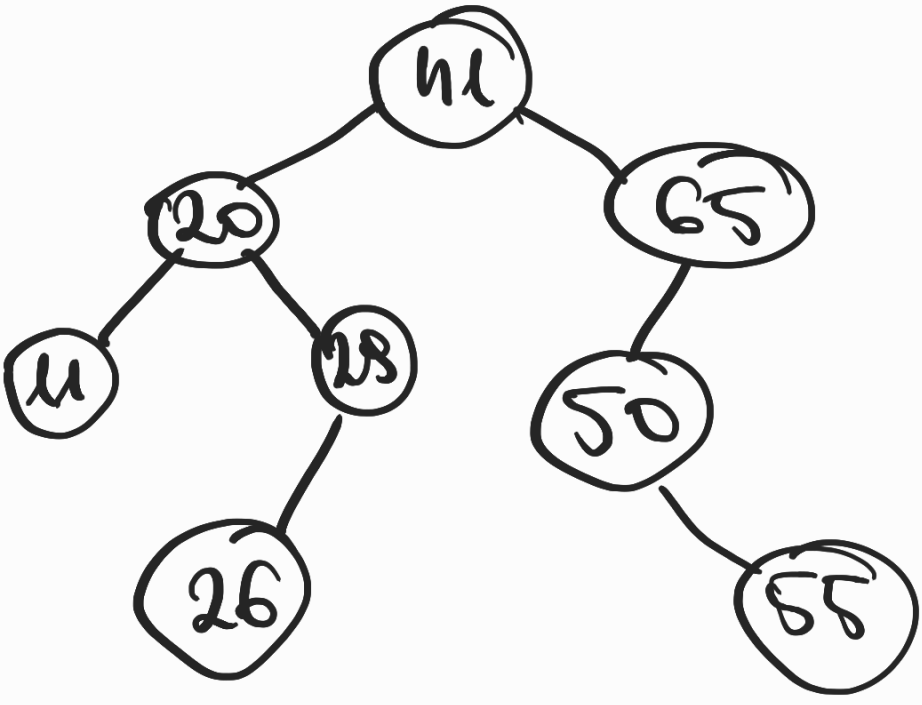
ROTATION : O(1)

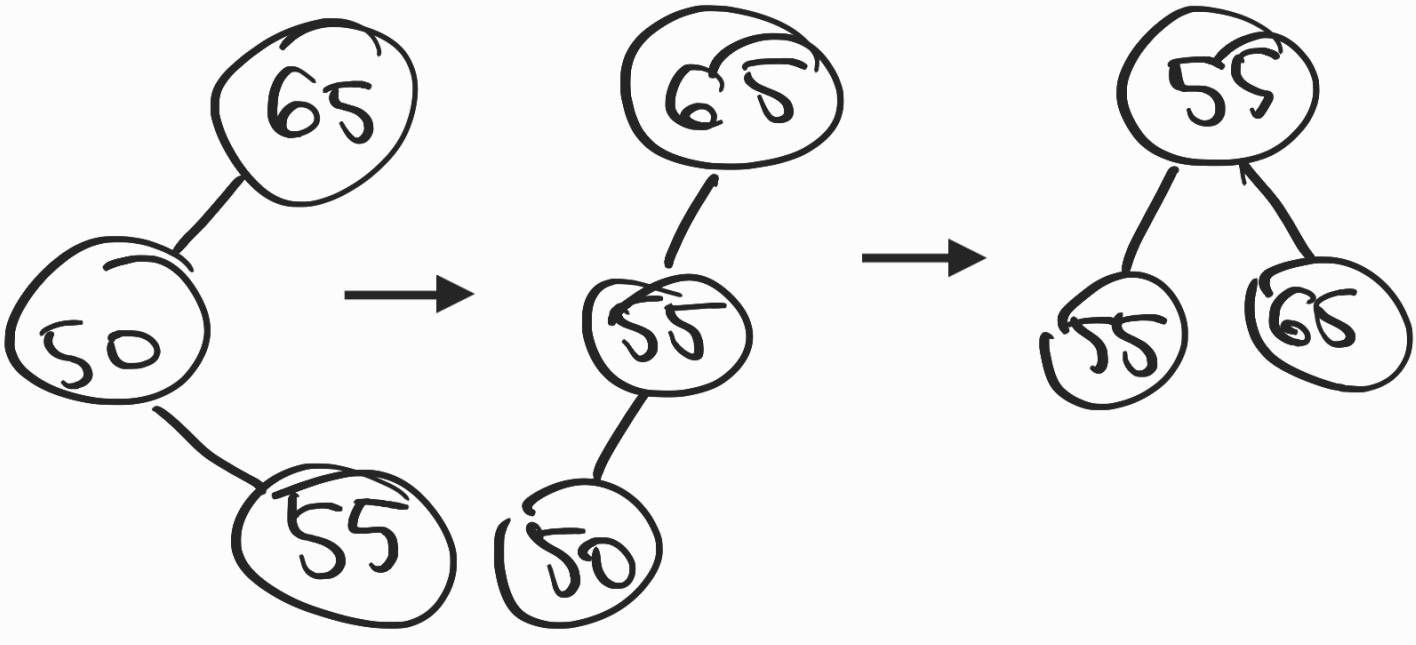

LEFT-ROTATE(x)
RIGH-ROTATE (y)
x
y
A
B
C
y
x
A
B
C


28
26
23
right-rotate(28)
26
23
28


41
20
65
11
23
50
3
41
2
20
1
65
∅
11
1
26
∅
50

Insert(55)



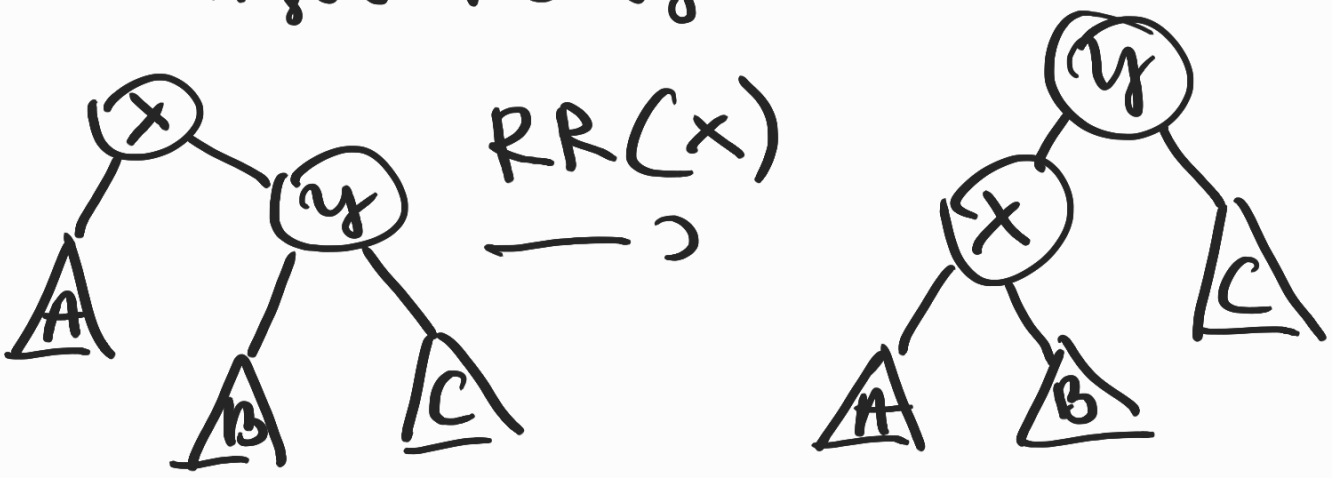


2*) fix AVL property from changed node up

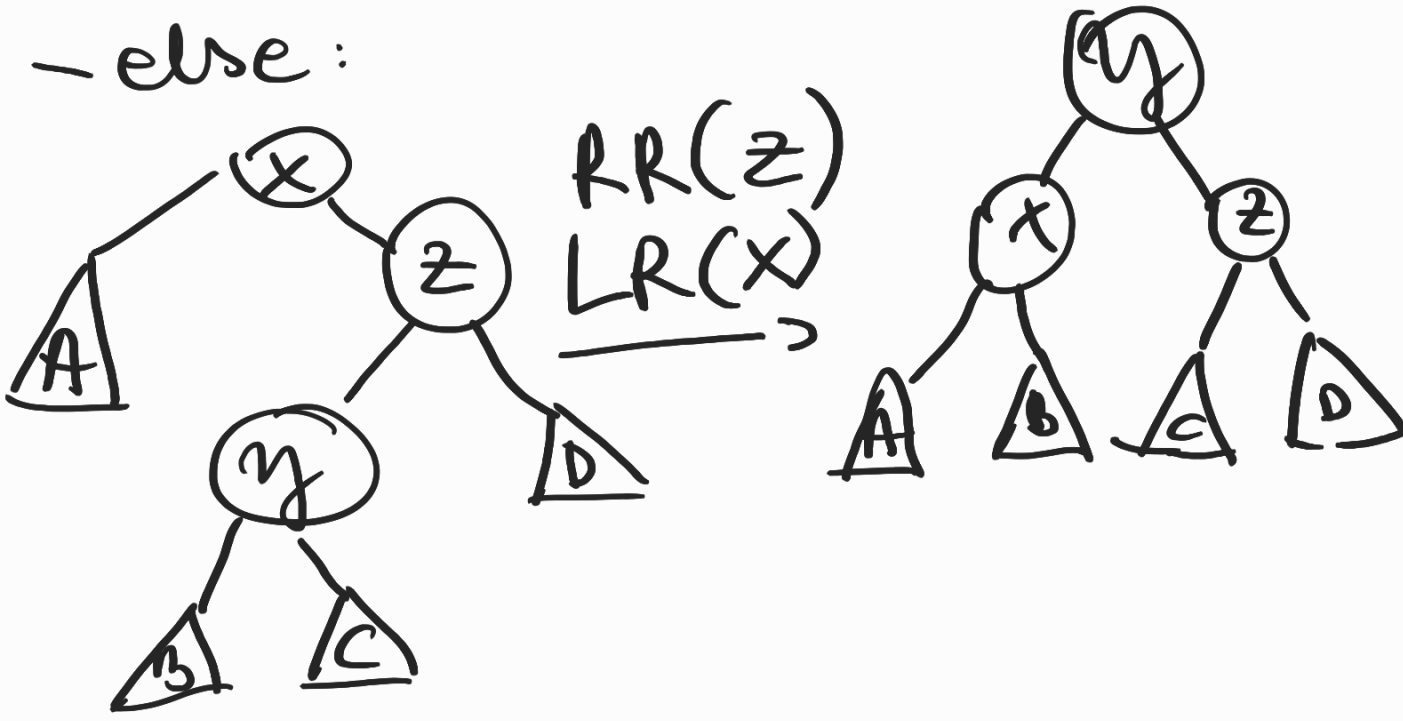
- suppose x is lowest

- assume right(x) higher (x.right)

- if x is right child is right-heavy or balanced



- else:



AVL SORT

- insert n items - $\Theta(n \log n)$
- in-order traversal - $\Theta(n)$

ABSTACT DATE TYPE:

- insert & delete } priority queeu
- min
- successor / pred.

↓

DATA STRUCTURE

- heap
- AVL
- balanced BST