6.046 Lecture 10

Red-black trees, Rotations, Insertions, Deletions. ①

Balanced search trees

Search tree data structure maintaining dynamic set of n elts using tree of height $O(\lg n)$.

Examples:

- AVL trees
- 2-3 trees
- 2-3-4 trees
- B-trees
- Red-black trees
- skip lists
- Treaps

Red-black trees:

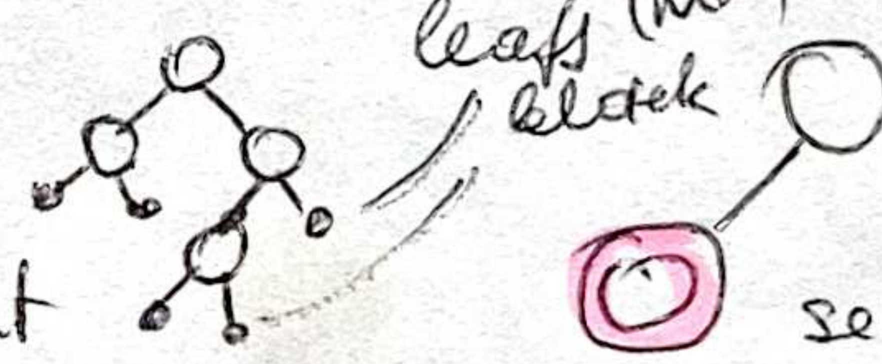
BST data structure with extra color field for each node, satisfying:

Red-black properties

① Every node is either red or black.

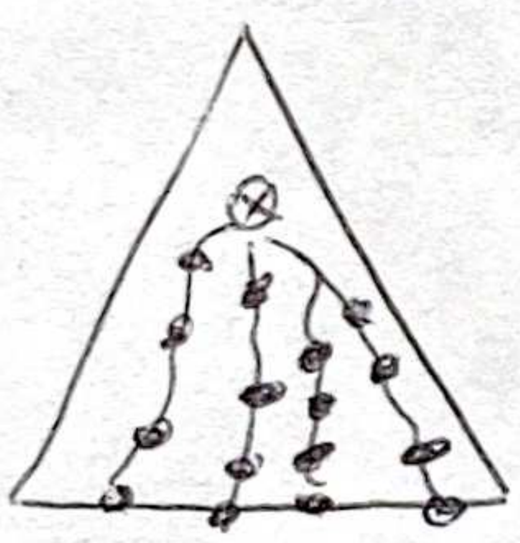
② The root & leaves (nil's) are black

③ Every red node has a black parent



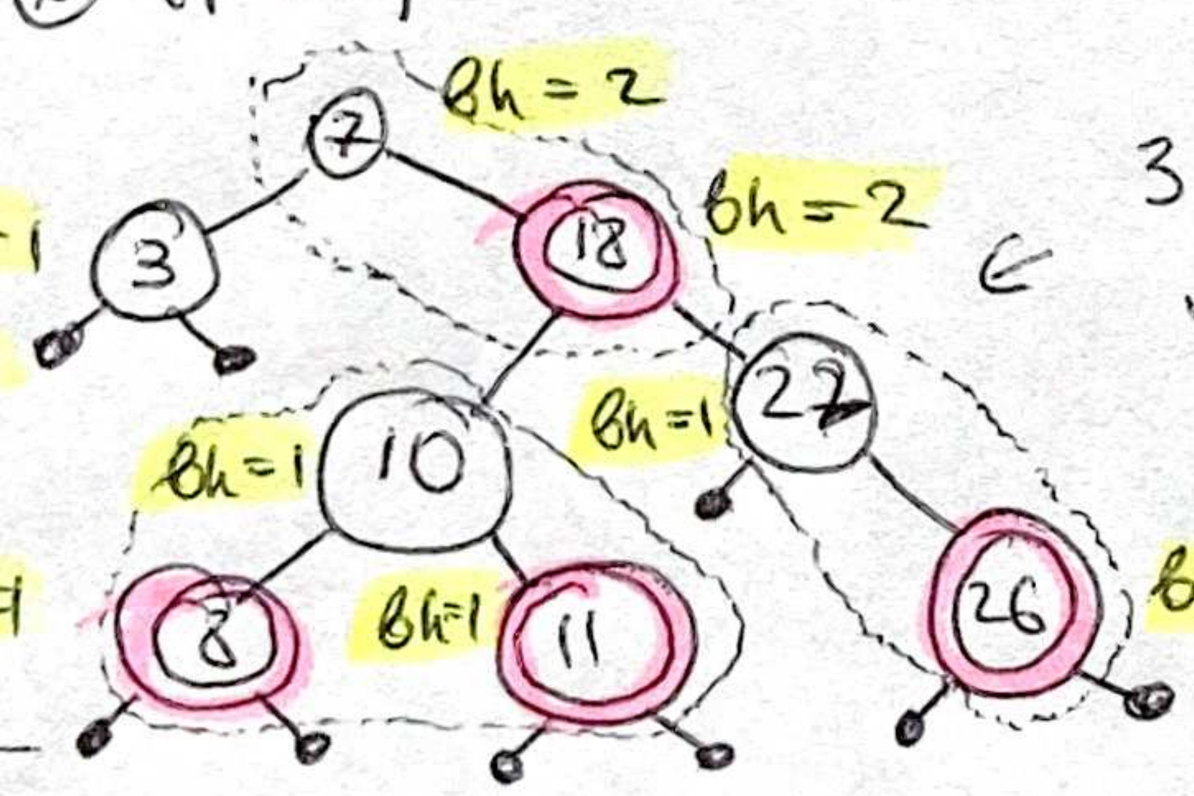
can have several consecutive black nodes, but not red nodes.

④ All simple paths from a node x to a descendant leaf of x have same # black nodes = black height (bh)



black height does not count ⊗ itself

Ex:

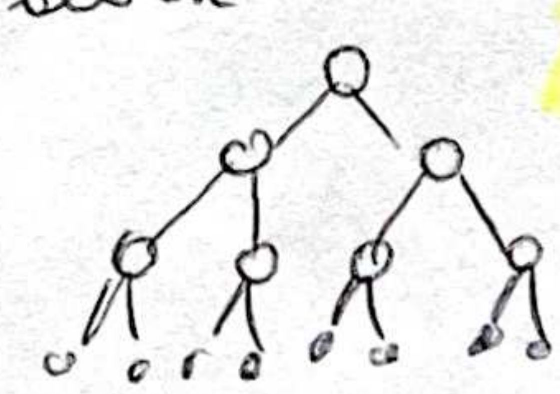


3, 7, 8, 10, 11, 18, 22, 26 ✓ valid BST

a) These properties should force the tree to have $O(\lg n)$ height

b) these properties are easy to maintain in a dynamic setting

easy to start: just build a BST with all black nodes

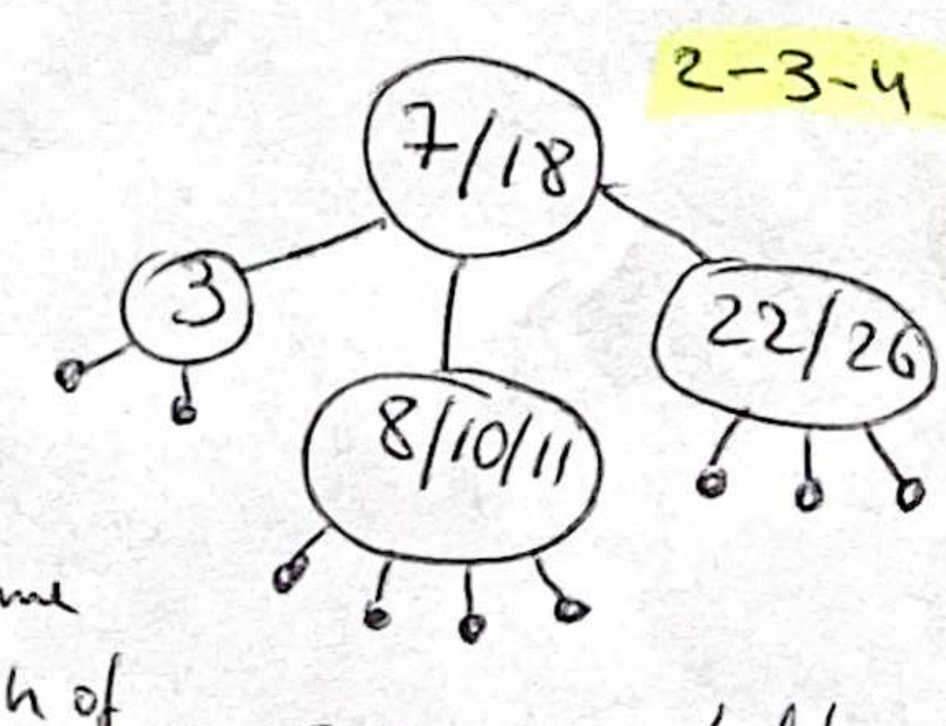


Height of red-black tree:

Red-black tree with n keys has height $h \le 2\lg(n+1) = O(\lg n)$

Proof sketch:

- merge each red node into its black parent node
- every internal node has 2, 3, or 4 children nodes
- every leaf has the same depth = bh of the root (by property 4)

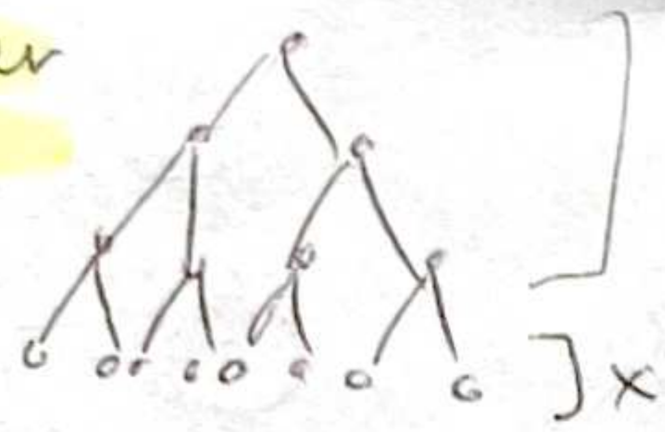


all leafs have the same depth = bh of root

2, 3, or 4 children nodes

leaves = $n+1$ in either tree

for branching factor of 2 in the original tree.



In 2-3-4 tree:

$2^{h'} \leq$ # leaves $\leq 4^{h'}$ $\Rightarrow$ $2^{h'} \leq n+1$

$\Rightarrow h' \leq \lg(n+1)$

$h \leq 2h'$ $\quad (h' \geq \frac{1}{2} h)$

at most one red node for every black node on any path from root to leaf. (property 3)

$\Downarrow$

all red-black trees are balanced ✓ $\Leftarrow$ $h \leq 2 \lg(n+1)$

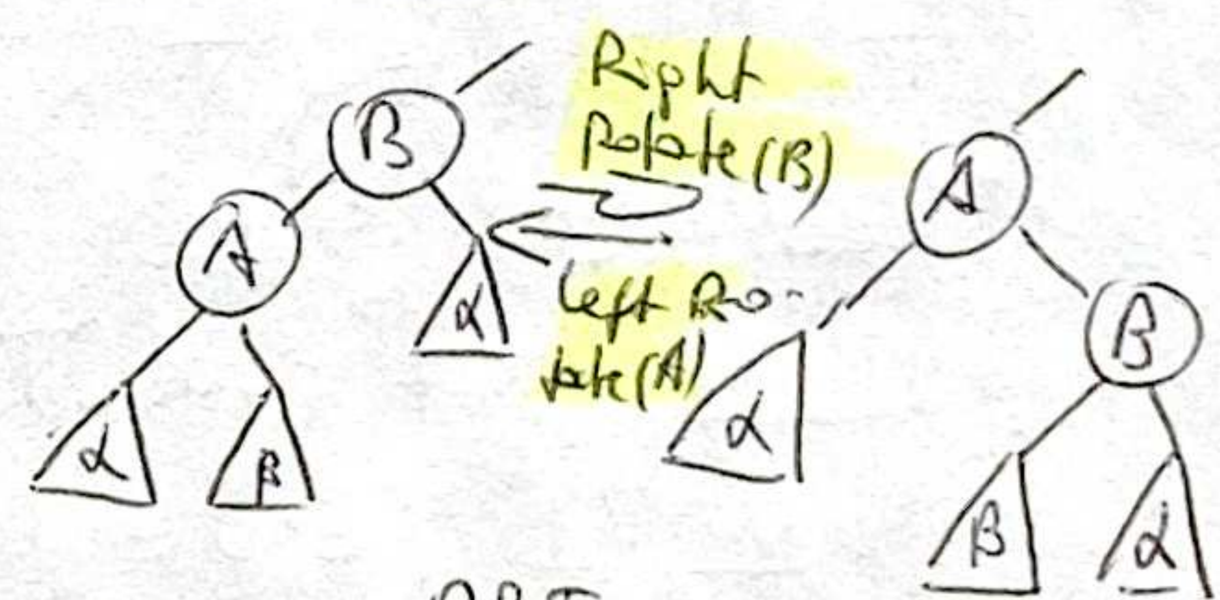
Corollary: Queries (Search, Min, Max, Successor, Predecessor) run in $O(\lg n)$ time in a red-black tree. → easy to query

Updates (Insert & Delete)

must modify the tree.

- BST operation (tree insert, tree delete)
- color changes
- restructuring of links via rotations, constant time operations

Rotations:



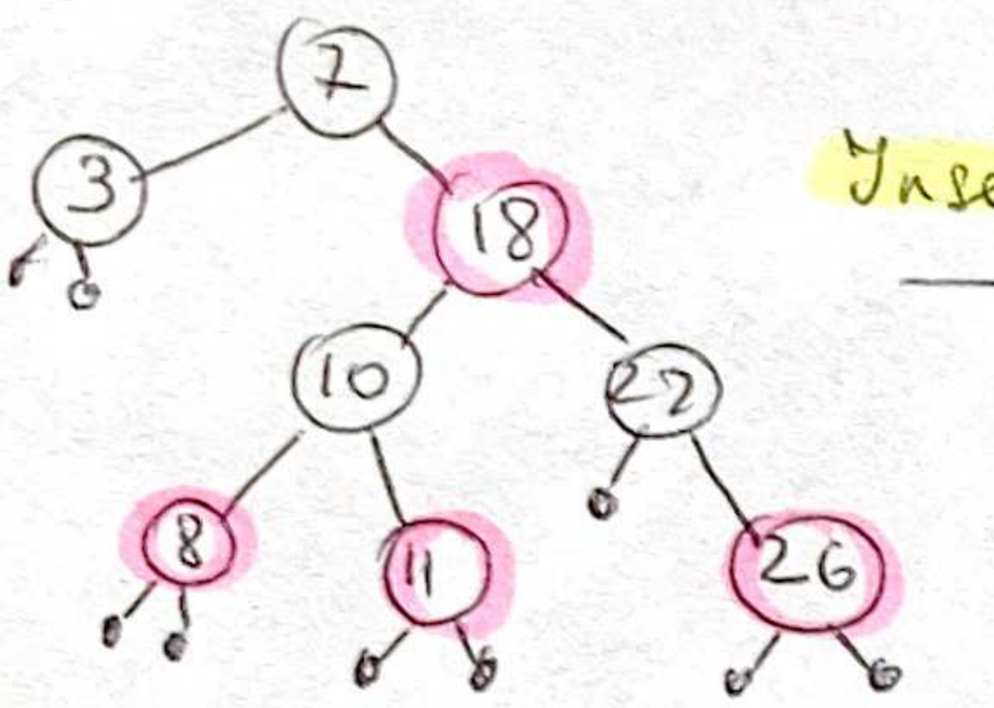
preserves BST property

$\forall a \in \alpha, b \in \beta, c \in \gamma \quad a \leq A \leq b \leq B \leq c$

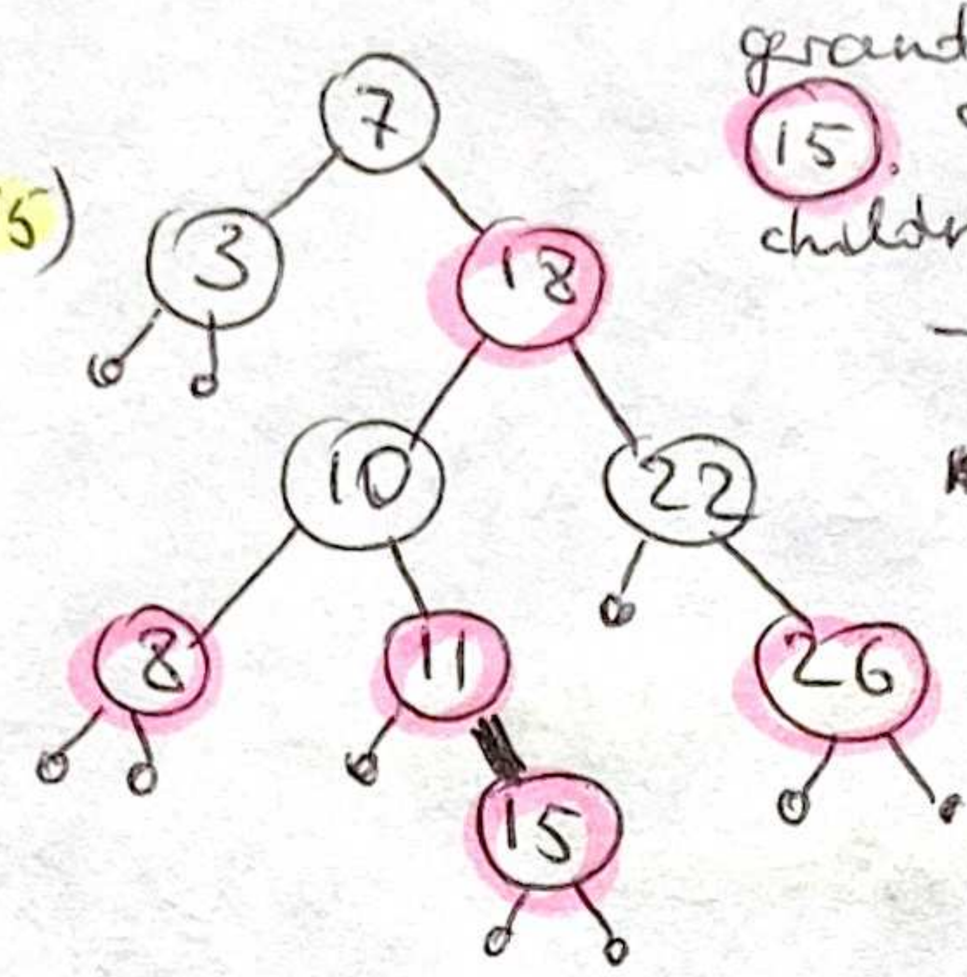
RB-Insert (x): my comment similar to AVL

Idea:
- Tree-Insert (X) → would be a leaf in a BST, but gets two black leafs in red-black tree.
- color node red
- problem if parent is red (violate property ③) but property ④ still holds. [redacted].
- move violation of ③ up the tree via recoloring of nodes until we can fix violation via rotation & recoloring

Ex Insert (15)

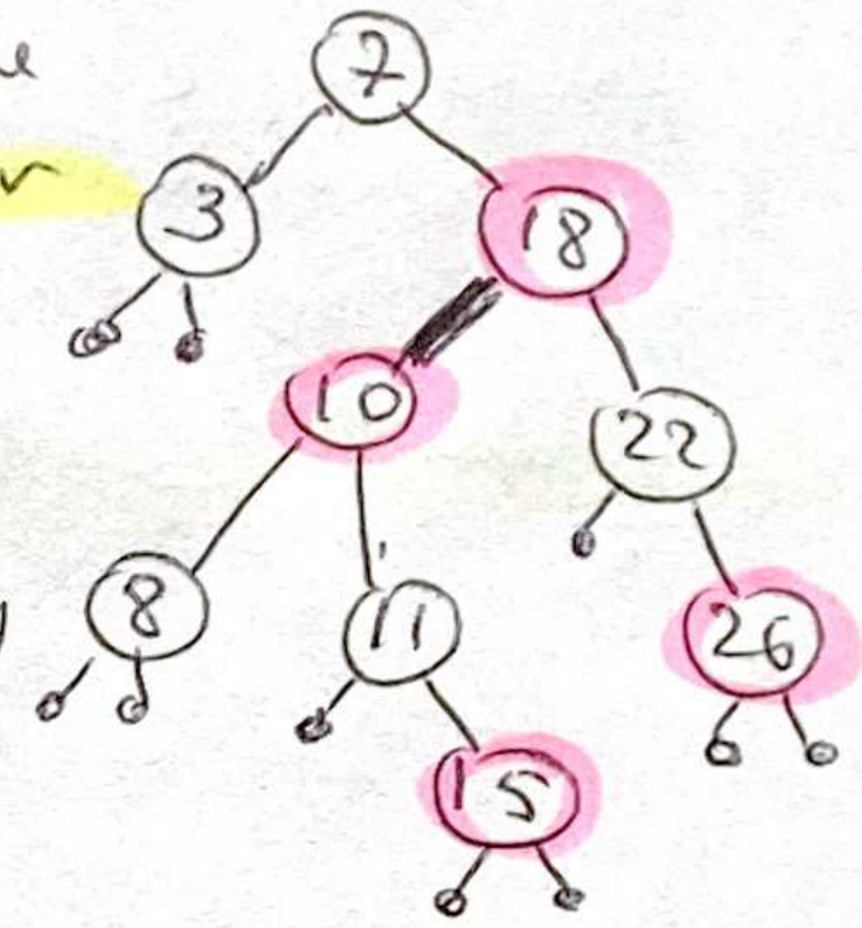


Insert(15) →



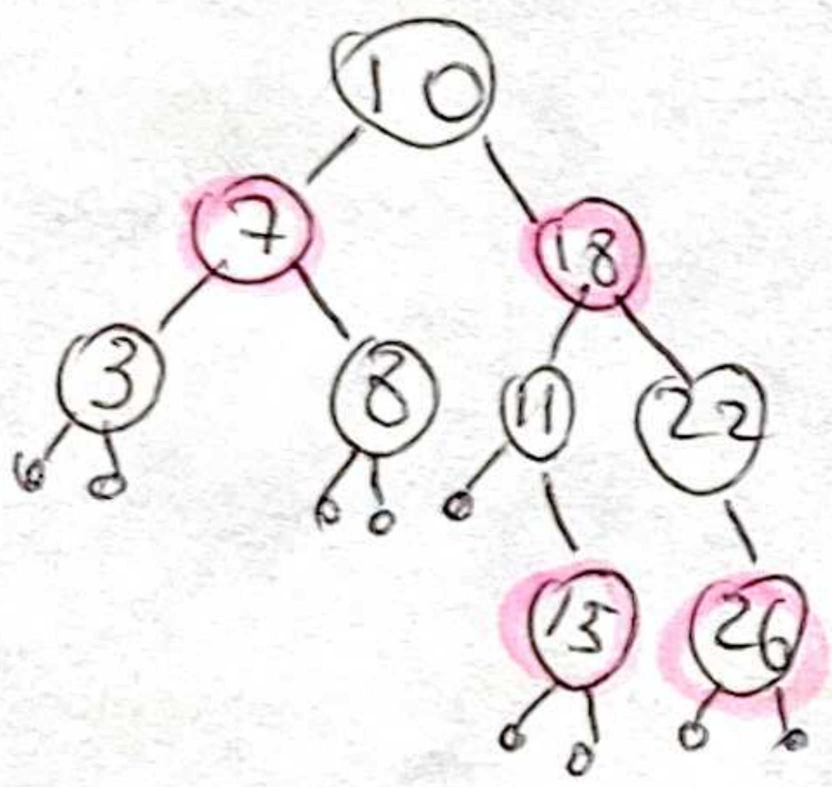
look at the grand-parent of 15. Two red node children → Recolor

[redacted] violation of property 3

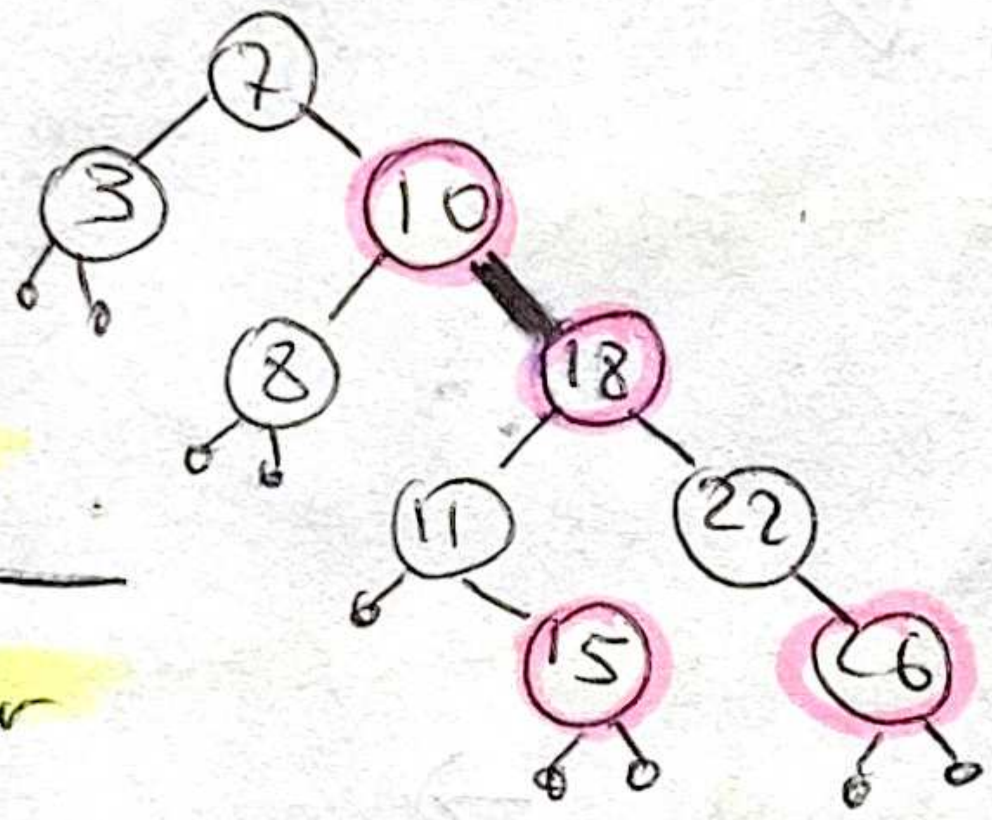


Rotate Right(18)

7, grand-parent of 10 has a red and black child nodes [redacted] cannot recolor 7 and its child [redacted] nodes



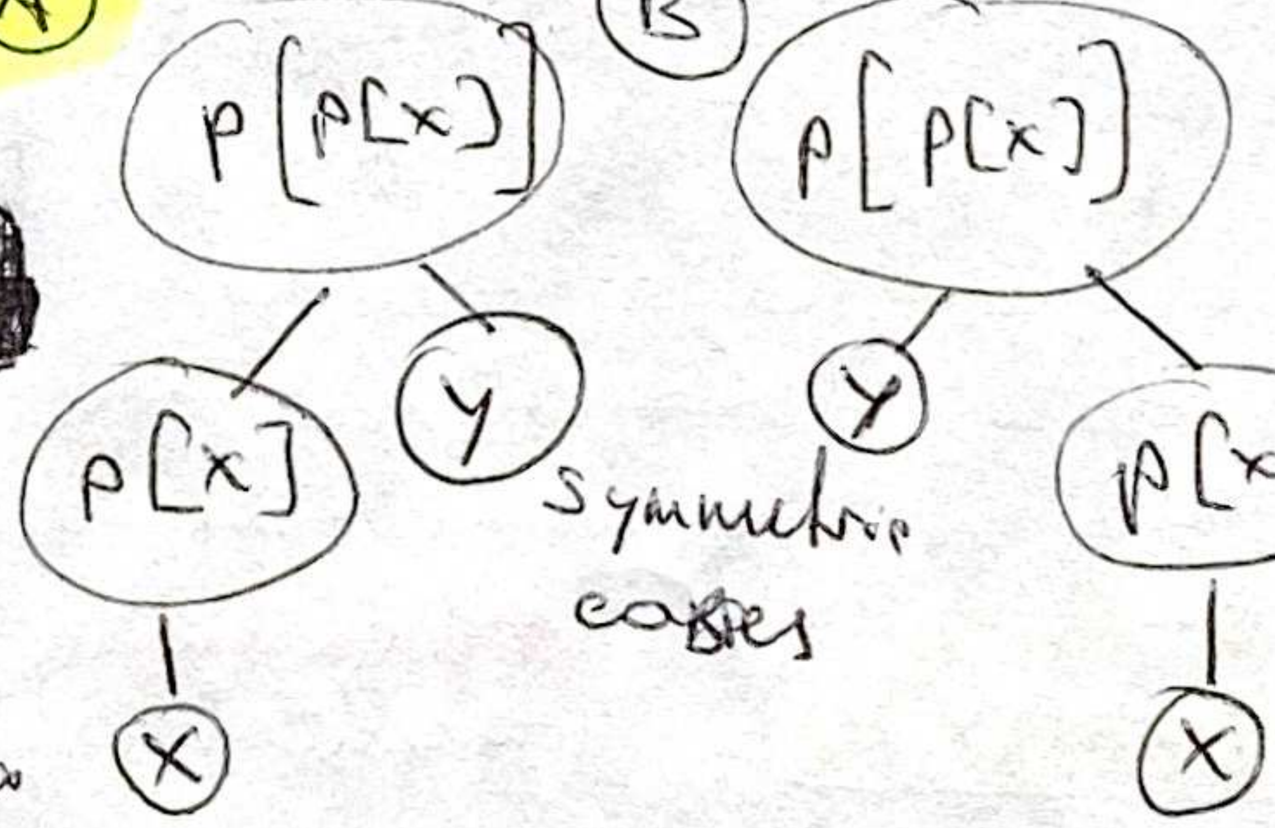
Rotate Left(7) ← Recolor



RB-Insert(T,x):

```
Tree-Insert (T, x)
color [x] ← Red                                  Red
                                                  //
while  x ≠ root [T]  and  color [x] [redacted]
   do if p[x] = left [p[p[x]]]  // A
                                   ↑
                                   B is symmetric
      then y ← right [p[p[x]]]
           if color [y] = Red
           then < Case 1 >
rotations{ else if x = right [p[x]]
                then < case 2 >
           < case 3 >
   else (B)
      same as (A), but reversing
           left ↔ right
color [root [T]] ← Black
```

(A) (B)



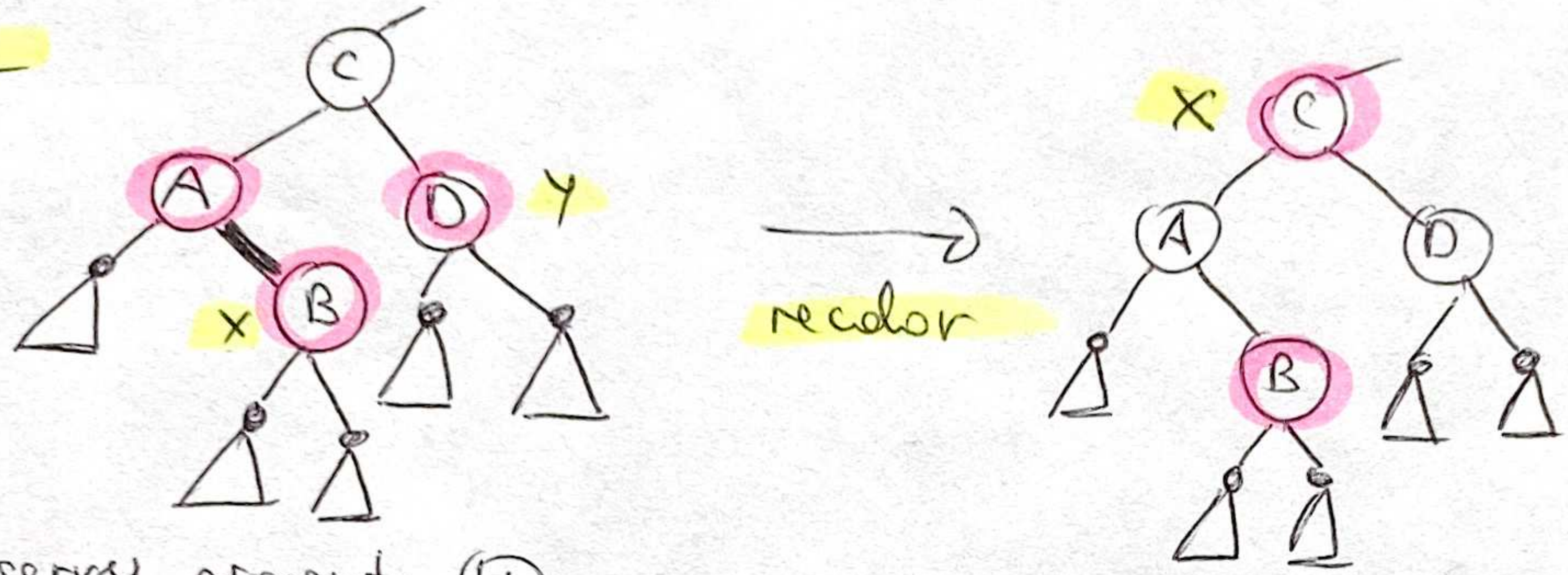
my comment

AVL tree cases also symmetric, and comparisons are two levels up lecture 10

3 cases of A ③

△ has black root & all △ have same bh

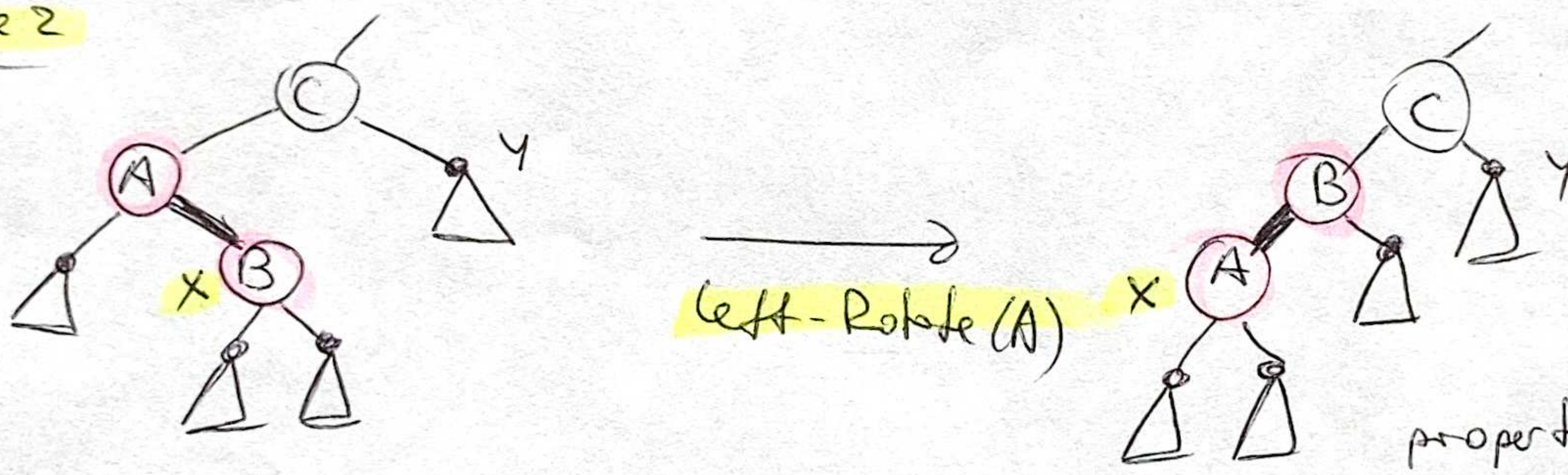
case 1



preserves property ④
fixes property ③ locally,

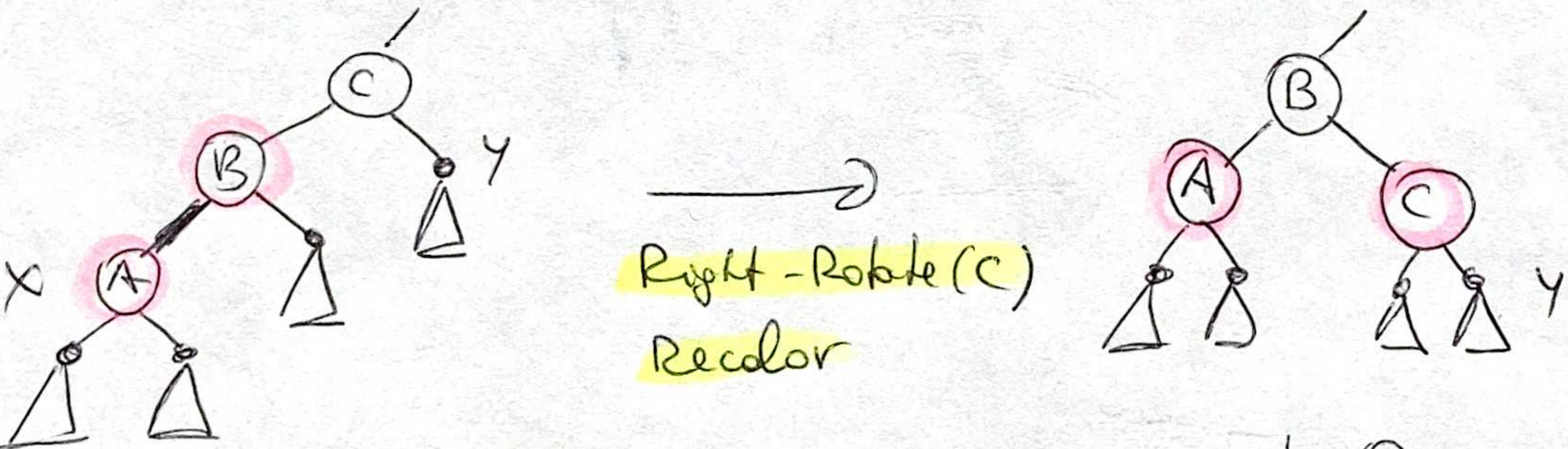
in case1 [redacted] x can be right(A) or left(A)

case 2



property ④ preserved

case 3



property ③ is fixed
property ④ preserved

RB-Insert: adds x to set and preserves the properties of RBTs

Terminates [redacted] after case 2 + 3 and 3, only case 1 can continue upward traversal. O(1)

case 1, does not change tree struct by rotations, only recoloring of nodes and moves x up by 2 levels → O(lg n)

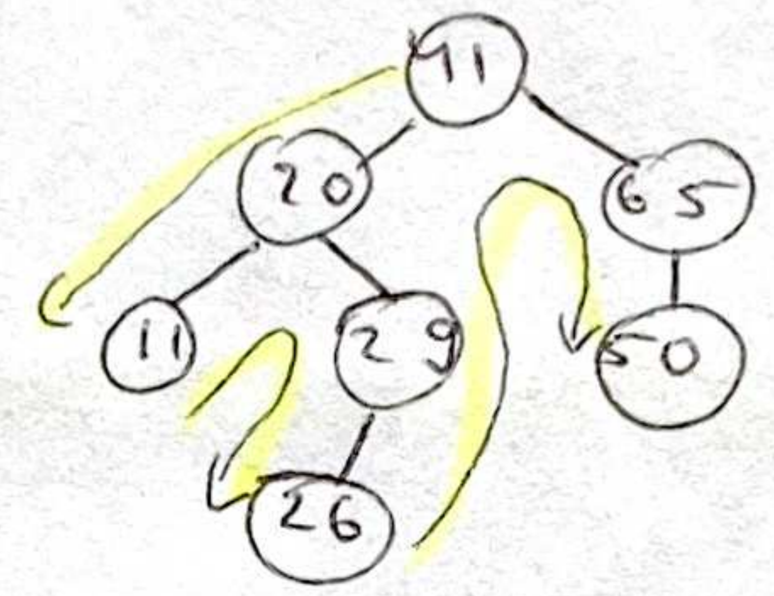
my comment: but in AVLs h < 1.440 lg n + $\log_\varphi \sqrt{5}$ my comment

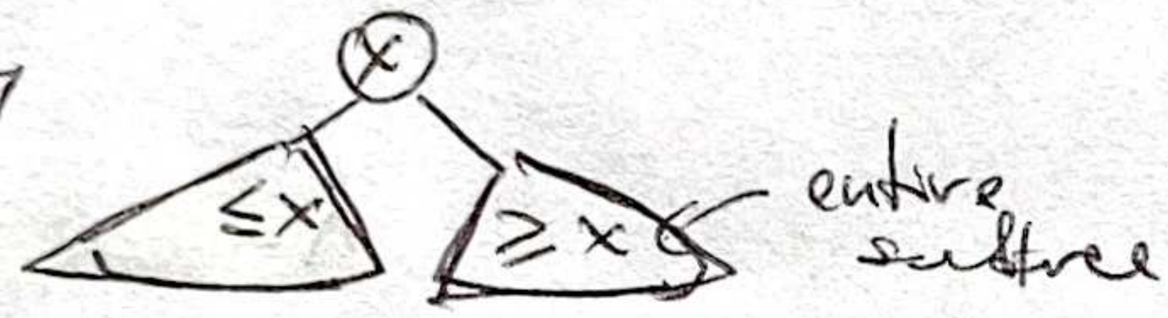
Rotations are more compute-intensive ([redacted] AVL)

Θ(1) Both insert & delete → [redacted] const # in RBTs (a) after insert, but not delete

BSTs

- rooted binary tree
- each node has [redacted]
  - key
  - left pointer
  - right pointer
  - parent pointer
- BST property





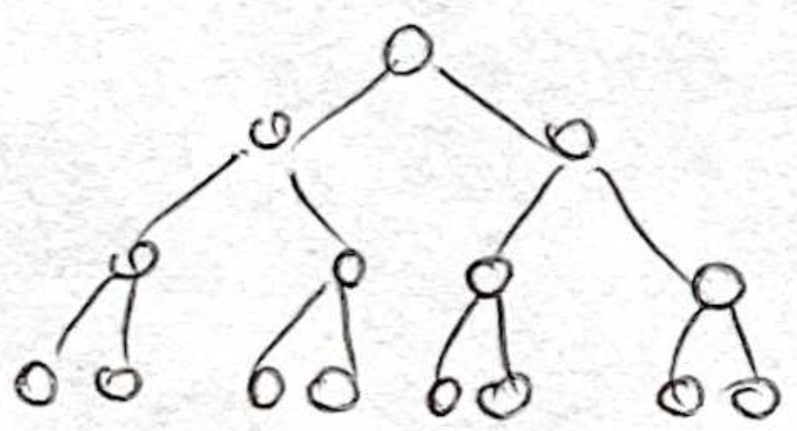
sorted order: in-order traversal

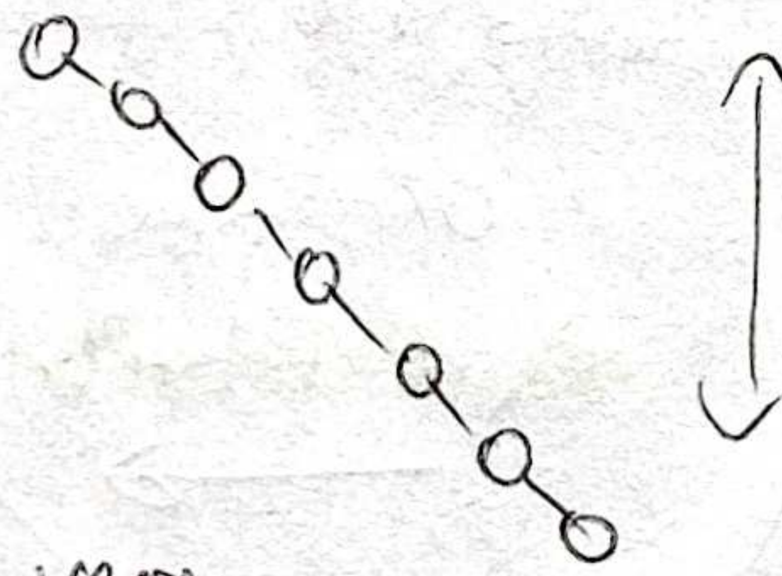
11, 20, 26, 29, 41, 50, 65

BST ops

insert, delete, min, max,
next larger / smaller (successor / predecessor) in $O(h)$ time

Balanced or not:





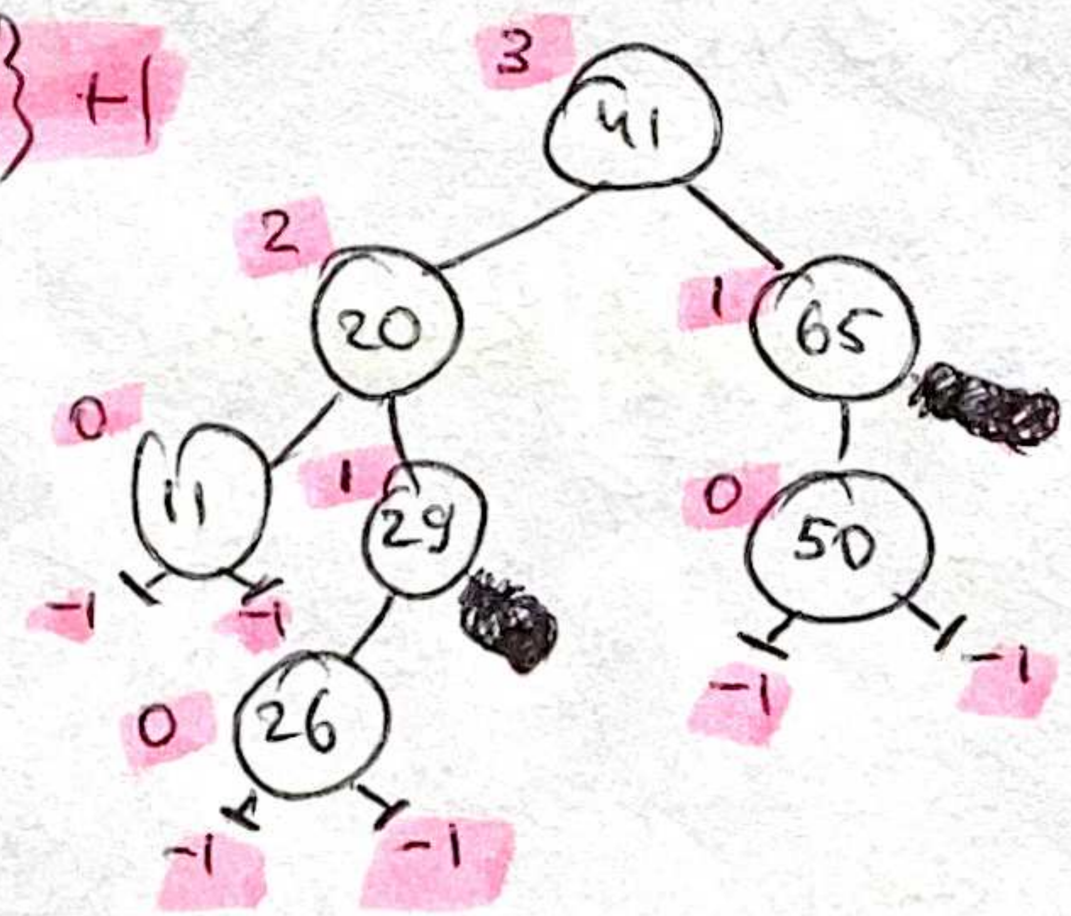
h = length of longest path from root to leaf
down

balanced if
$h = \Theta(\lg n)$

very unbalanced

height of a node: longest path from the node [redacted] down to a leaf

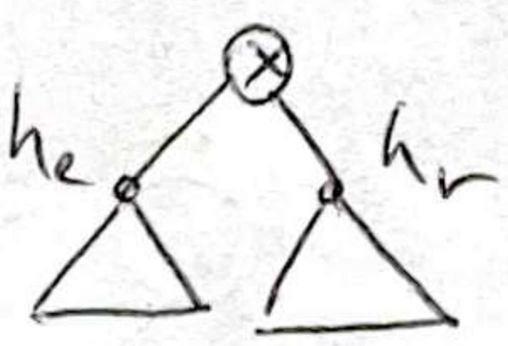
$= \max\{(\text{height left child}), (\text{height right child})\} + 1$



AVL trees

require heights of left and right children of every node to differ by at most ± 1

$|h_\ell - h_r| \leq 1$



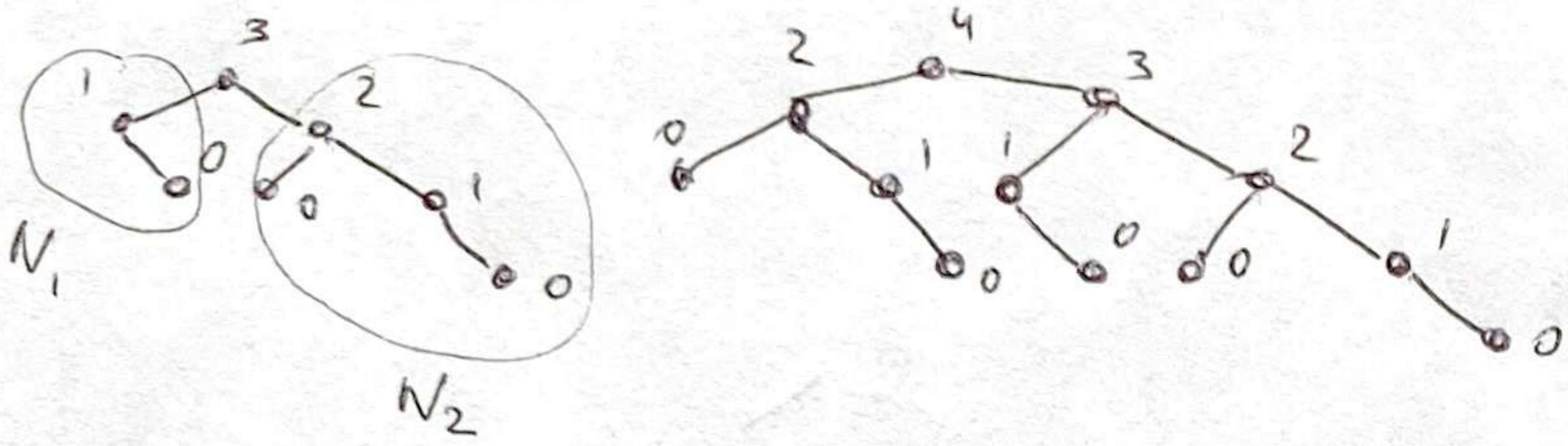
-1 from terminals/leaves $\rightarrow \max\{-1, -1\} + 1 = 0$

AVL trees are balanced:

worst case is when right subtree has height 1 more than left for every node.

N_h = min. # nodes in an AVL tree of height h.

my drawing: AVL worst case



$N_1 = O(1)$

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

right left

$n = N_h > F_h = \frac{\varphi^h}{\sqrt{5}}$ 1.618 $= \varphi > 1$

Fibonacci

$\frac{\varphi^h}{\sqrt{5}} < n \qquad \log_\varphi\left(\frac{\varphi^h}{\sqrt{5}}\right) < \log_\varphi(n)$

monotonically increasing

$h - \log_\varphi \sqrt{5} < \log_\varphi(n) \approx 1.440 \lg n$

close to $\lg n$

Alternative analysis (less tight)

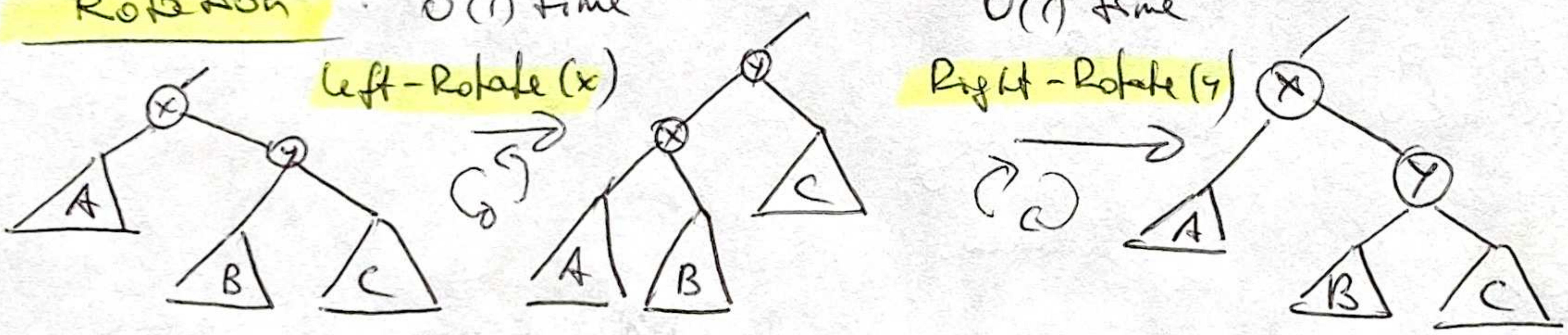
$N_h = 1 + N_{h-1} + N_{h-2}$
$> 1 + 2N_{h-2}$
$> 2N_{h-2}$
$= \Theta(2^{h/2}) \qquad h < 2\lg n$

AVL insert

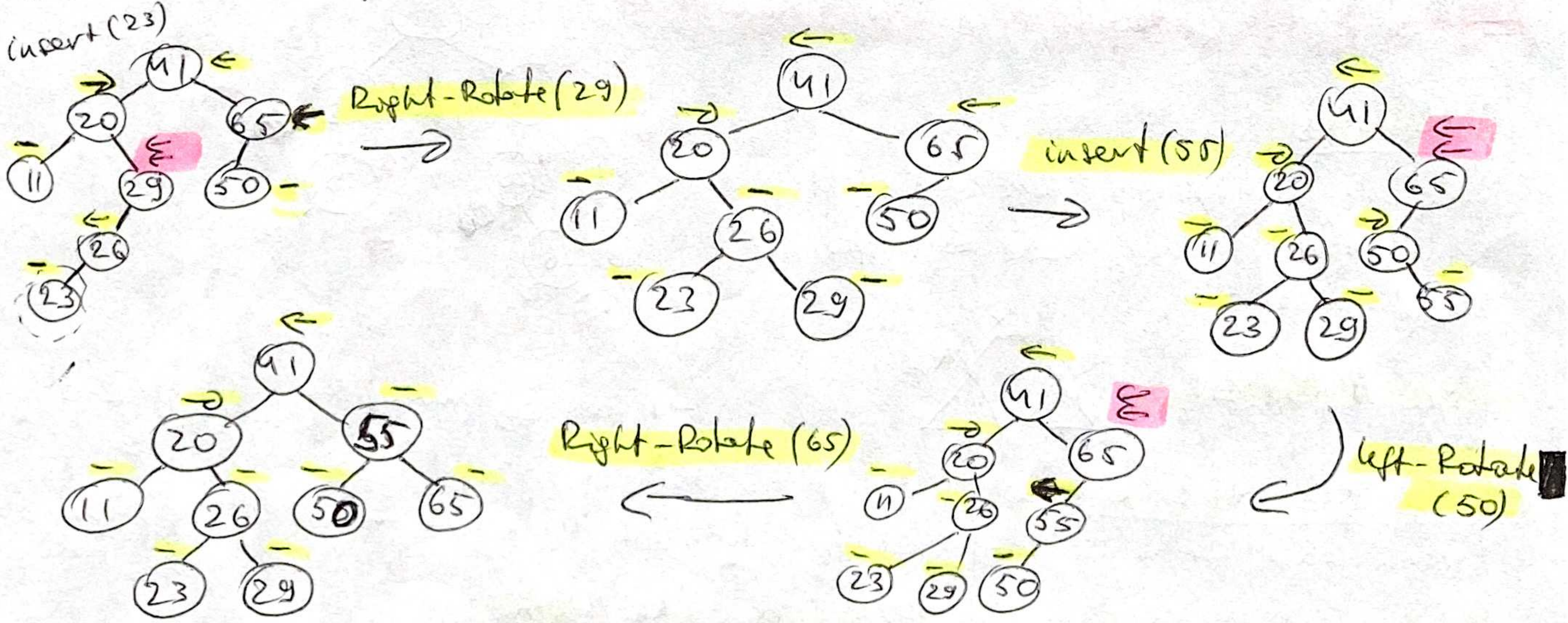
① simple BST insert [redacted] always (leaf insertion)

② fix AVL property from the changed node up

Rotation: $O(1)$ time ← root x goes to left $O(1)$ time

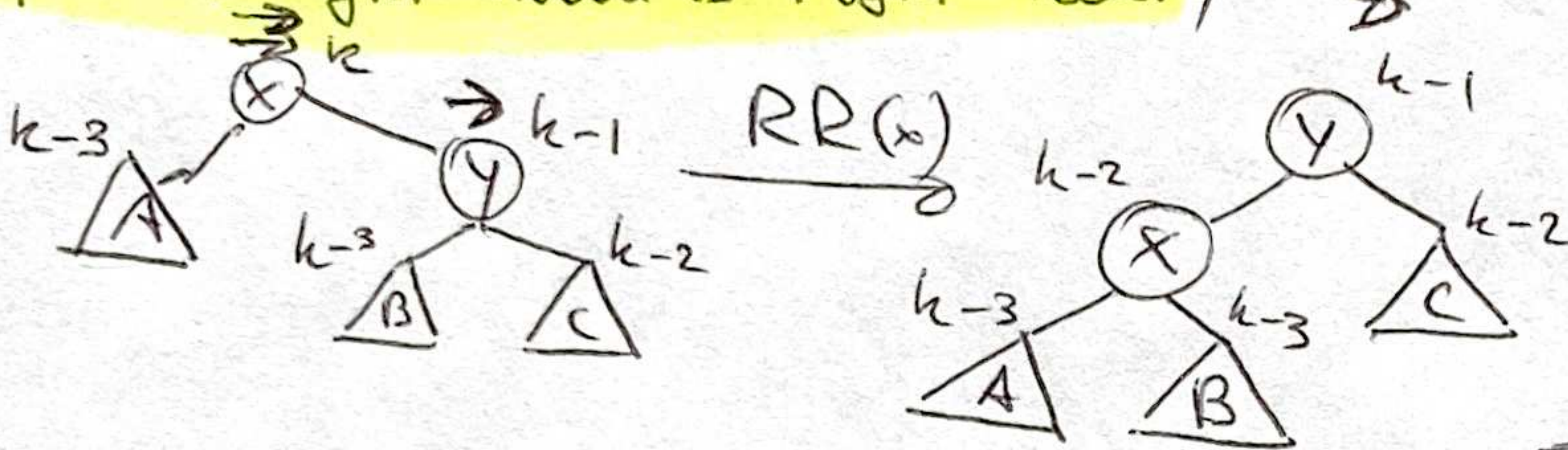


in-order traversal: $A x B y C = A x B y C = A x B y C$



- suppose x is lowest node violating AVL property from inserted node (leaf)
- assume right(x) higher (if left(x) higher, symmetric) see above example
- if x's right child is right-heavy →

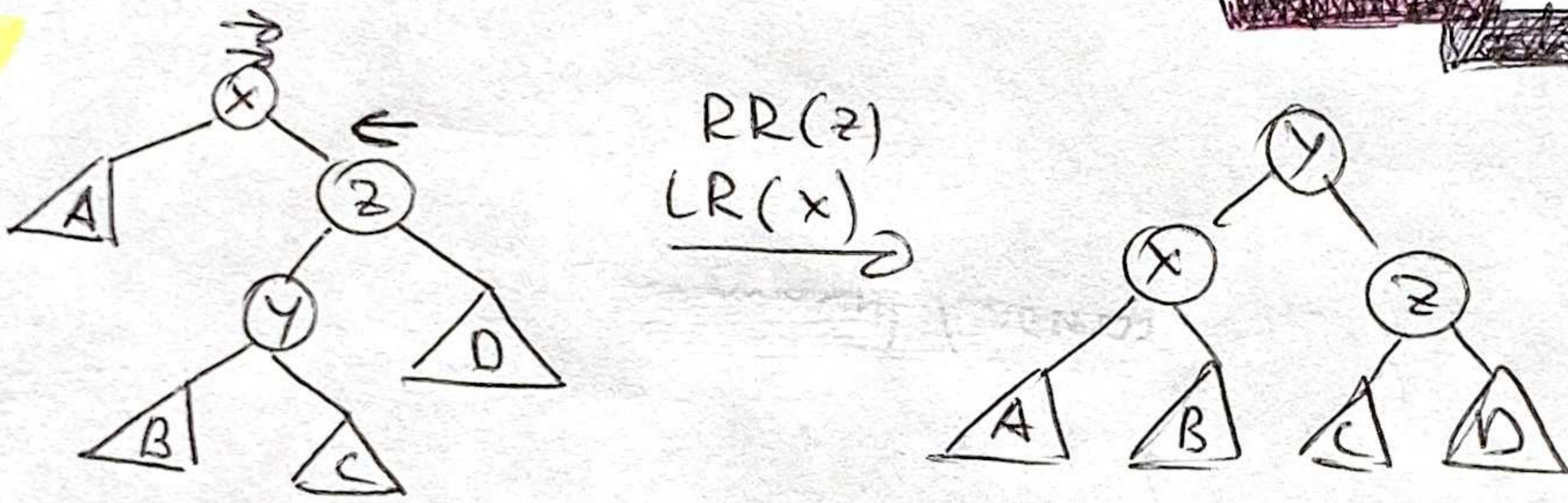
my comment: 1 or 2 rotations for insertion needed



may need to go up and fix y's parents

[redacted]

- else:



AVL Sort

- Insert n items — $\Theta(nh) \overset{\text{AVL prop.}}{=} \Theta(n \lg n)$
- in-order traversal — $\Theta(n)$

in contrast to heaps, also get successor/predecessor

Abstract Data Type

Data Structure

- insert/delete } priority queue: heap, AVL
- min/max
- successor/pred.

} balanced BST