Data structures & Algorithms

e.g. tree search

used in Watson for speed

parallel cluster (100+ nodes?)

fast queries (tree search)

e.g. in game theory, A*

chess

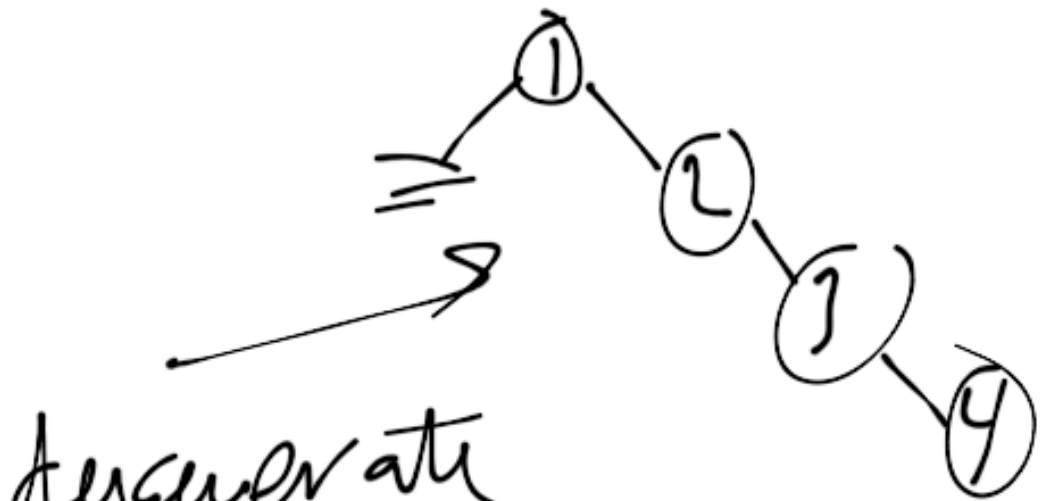
BSTs:

find_min:

return t->left == NULL ?

t :

find_min(t->left)



degenerate trees => unbalanced

degenerate characteristic of BSTs:

$\Rightarrow$ $O(n)$ search per item

balanced tree:

$O(\lg n)$ search per item

(for n searches, BST $\Rightarrow O(n^2)$)

BST search (for item x)

contains(x, t)
↑ node

if (t == null) return false;

operator< for x

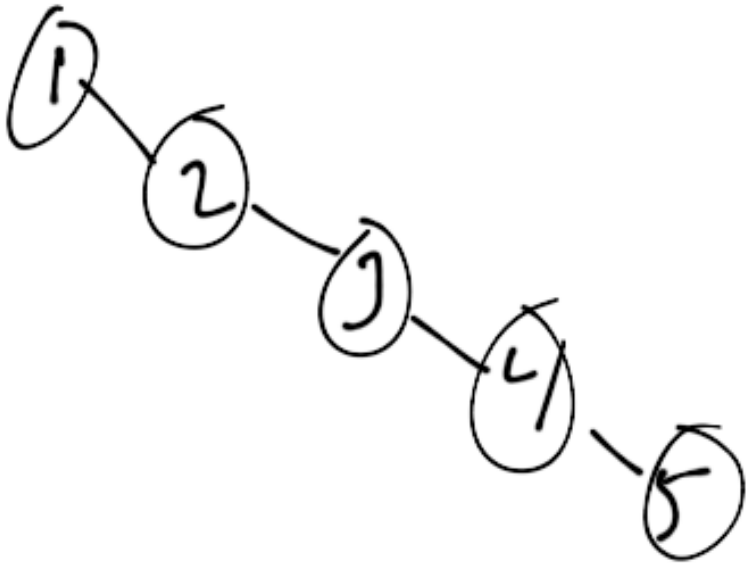
else if (x < t->data)
return contains(x, t->left)

else if (x > t->data)
return contains(x, t->right)

else return true;

```
insert(x, t)
        ↑ node
if(t == NULL)
        t = new node(x, NULL, NULL)
else if(x < t->data)
        insert(x, t->left)
else if(x > t->data)
        insert(x, t->right)
else ; // duplicate, no-op
```

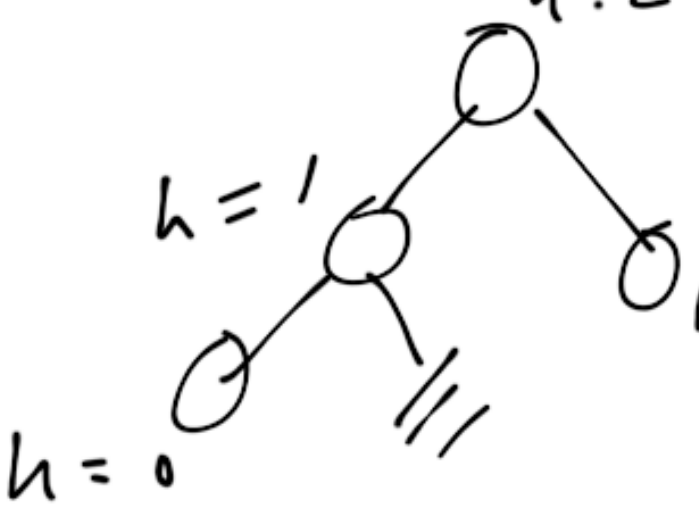
problem: tree degenerates
if input is sorted,
insert 1, 2, 3, 4, 5, ...



basically a list

AVL Trees: like BSTs but balanced:

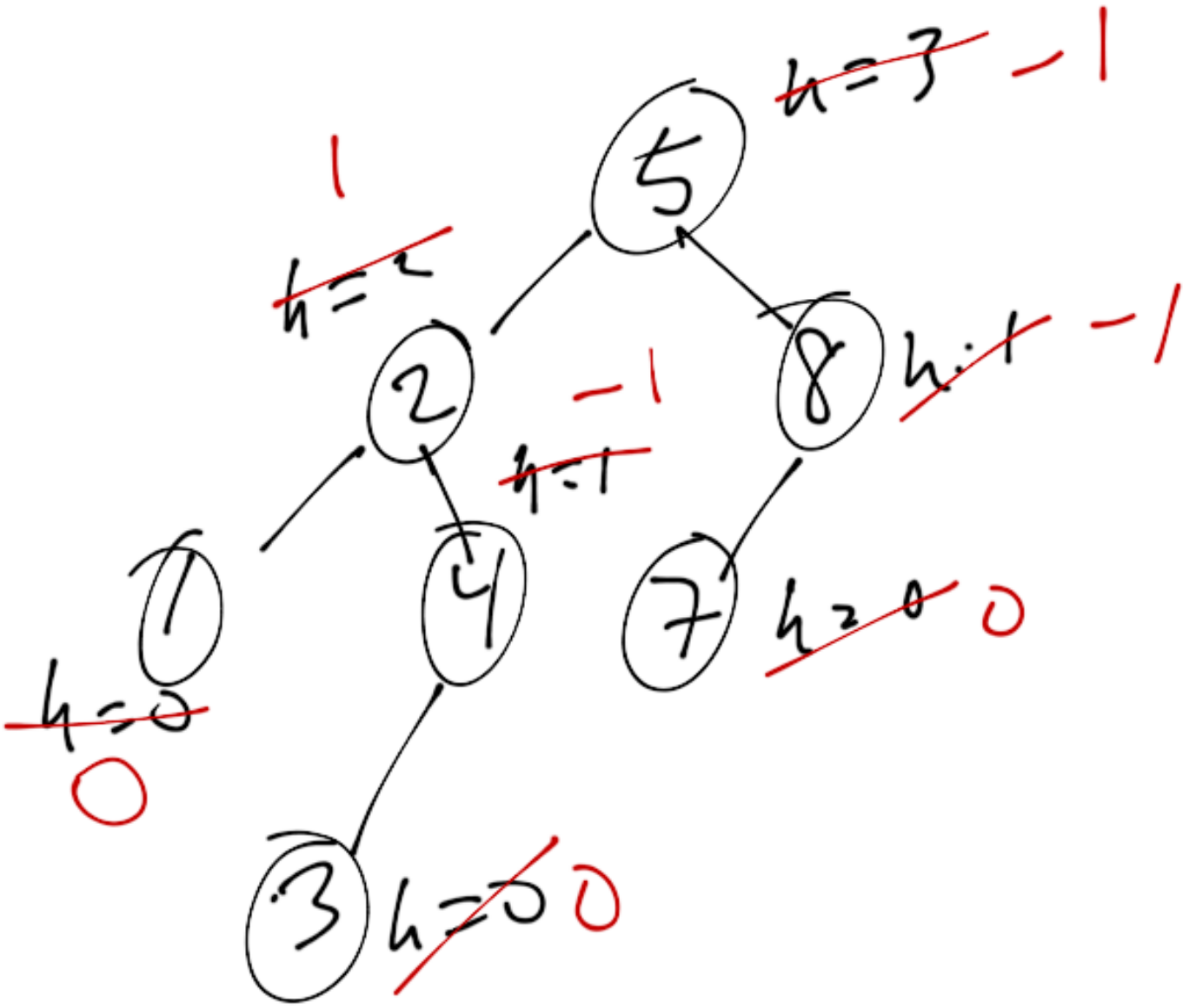
for every node, height of left & right subtrees differ by at most 1



h=2: of left subtree (h=1 of right subtree)

Is this an AVL tree? Yes!

- my preference is for storing balance info at nodes, not max heights like in textbook

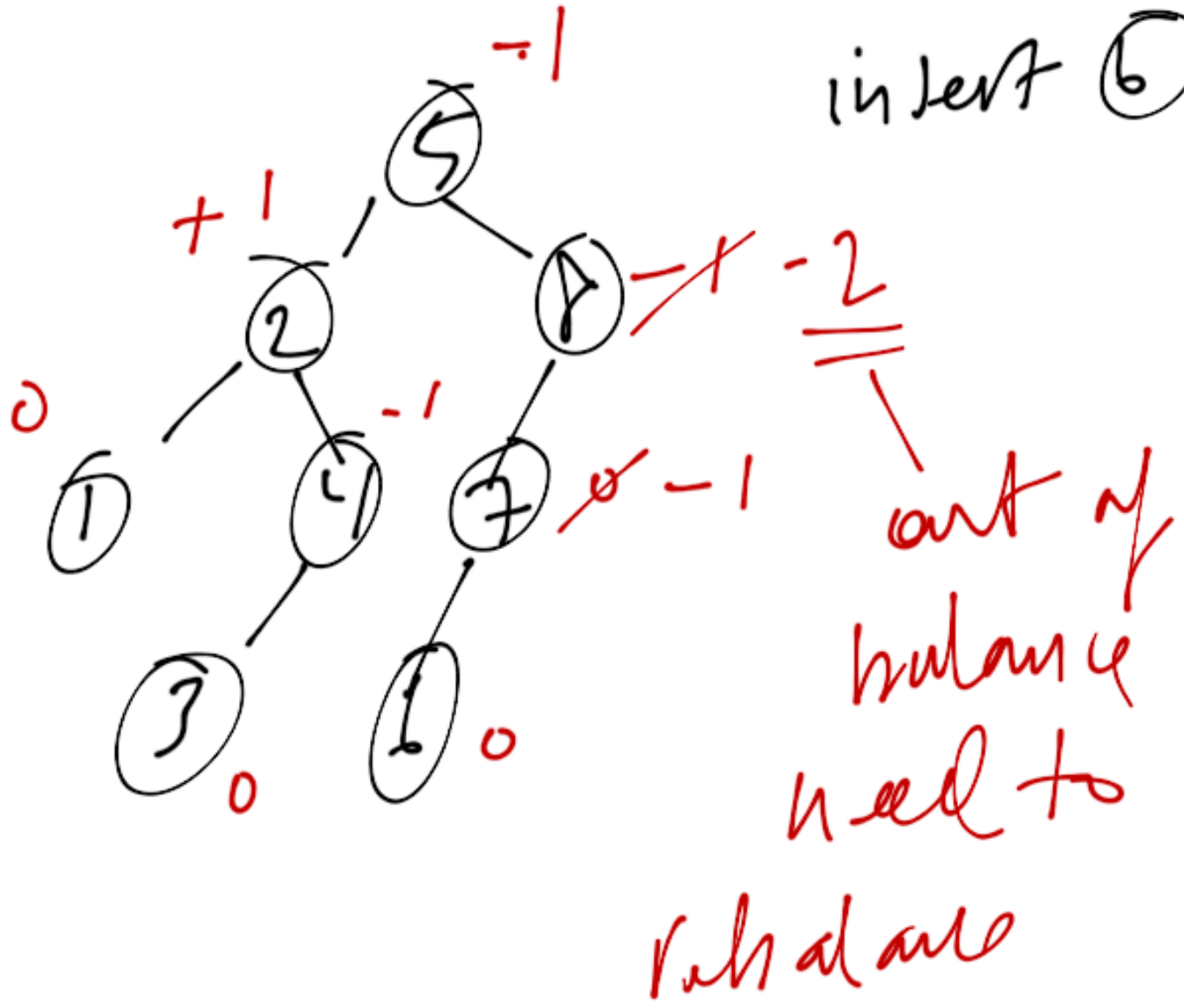


bal: diff. in subtree heights: $-1, 0, 1$

left right

- with bal property,
  nodes with $-1, 0, 1$
  are considered balanced

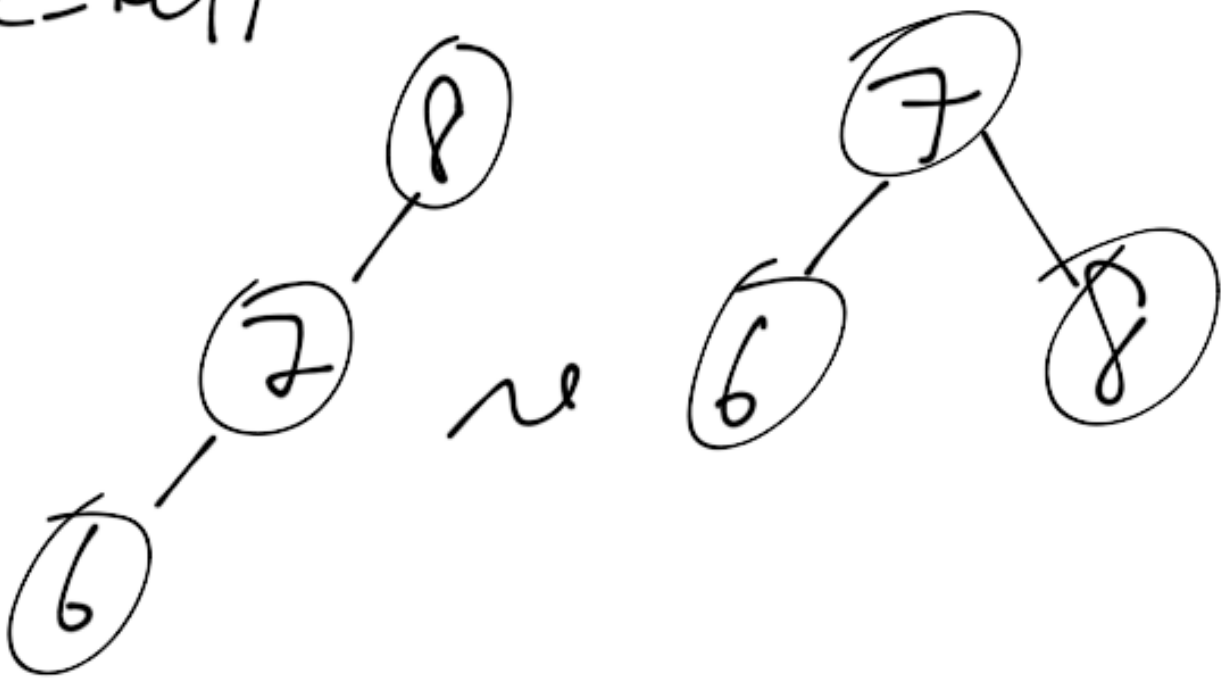
- insertion & deletion
  can disturb the balance
- when inserting, need to
  update balance of each
  node on the path back
  up to the root



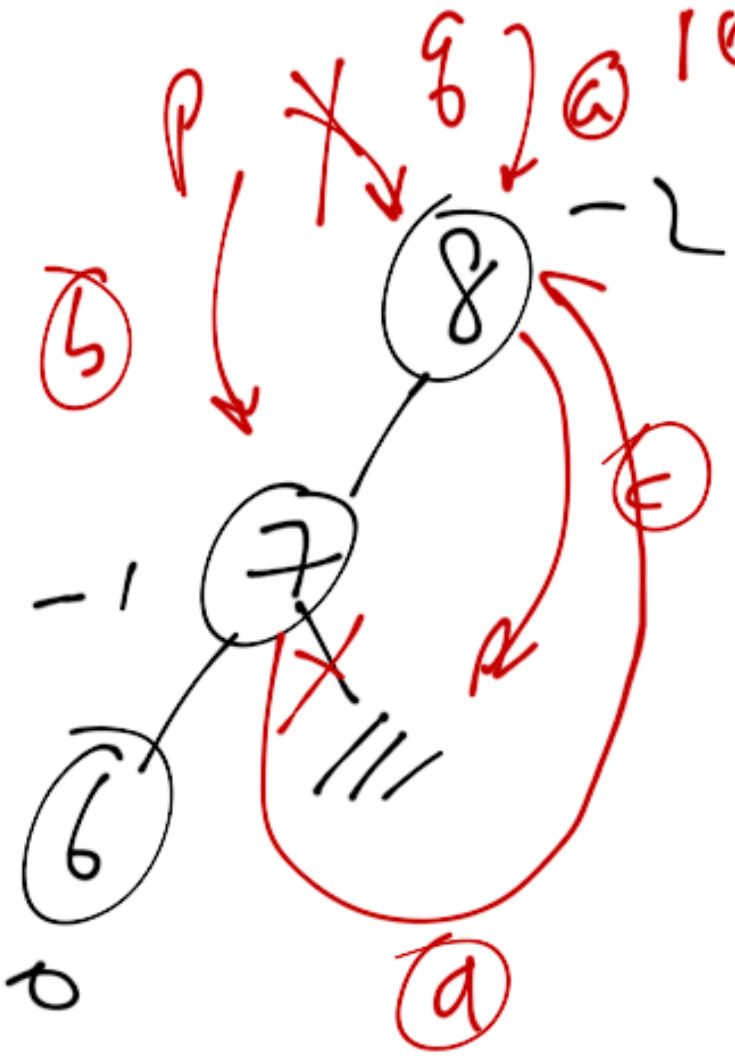
rotate_left (8)

= rotate tree to the right with left subtree

rotate_left



single rotation (to the right) with left subtree



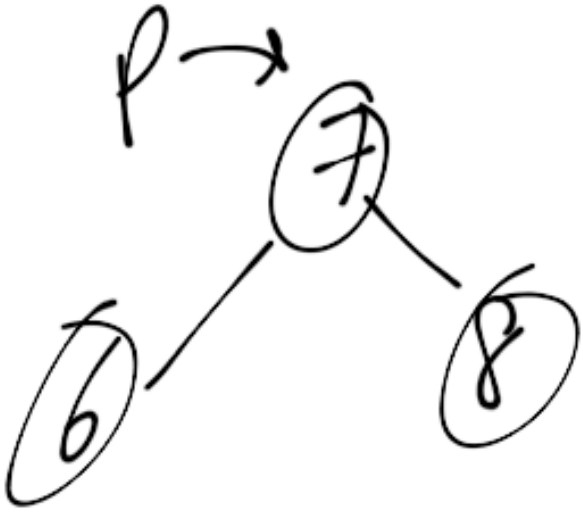
rotate_left(p)

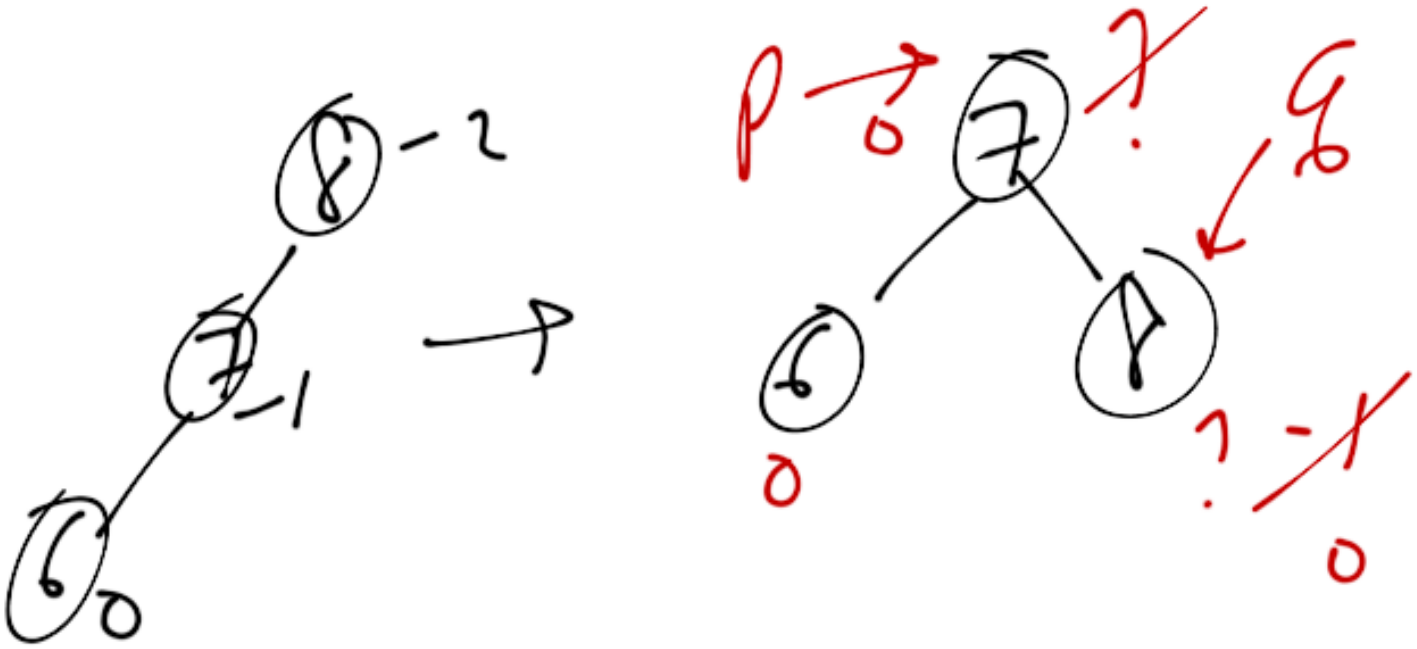
ⓐ $q = p$

ⓑ $p = p \rightarrow left$

ⓒ $q \rightarrow left = p \rightarrow right$

ⓓ $p \rightarrow right = q$





e) q->bal++

f) if (p->bal < 0)

q->bal -= p->bal;

g) p->bal++

h) if (q->bal > 0)

p->bal += q->bal

- 4 insertion cases !

case 1 & 4 :

insertion into :

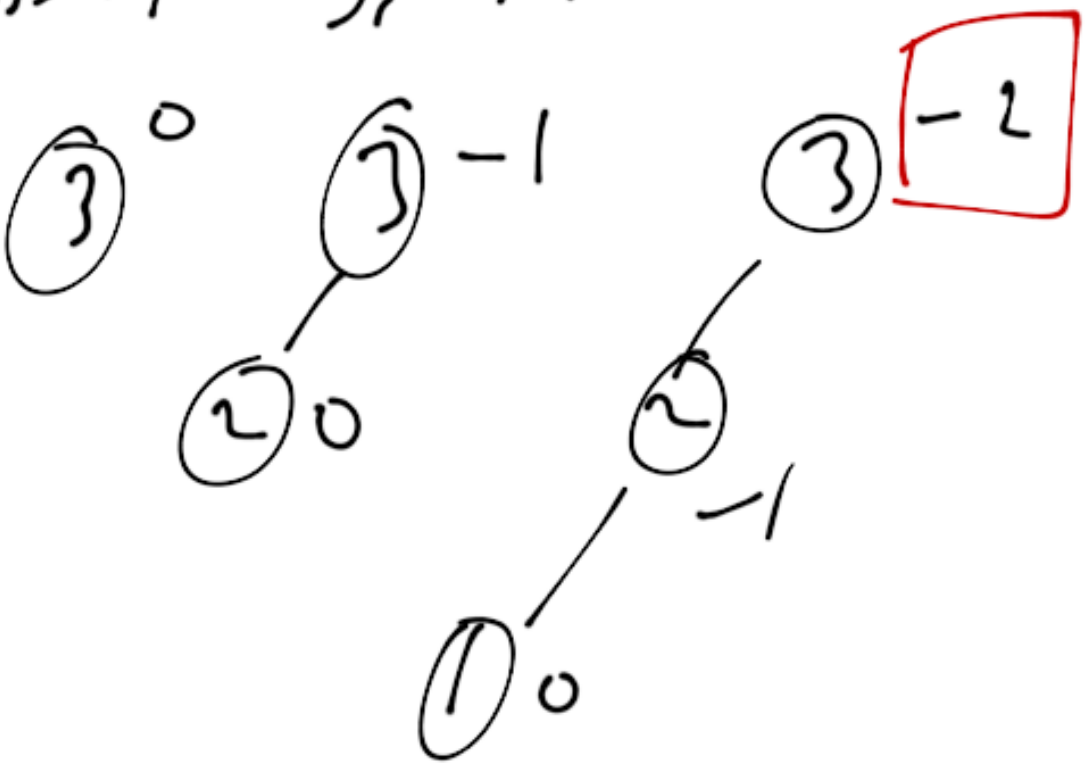
left subtree of
left child

right subtree of
right child

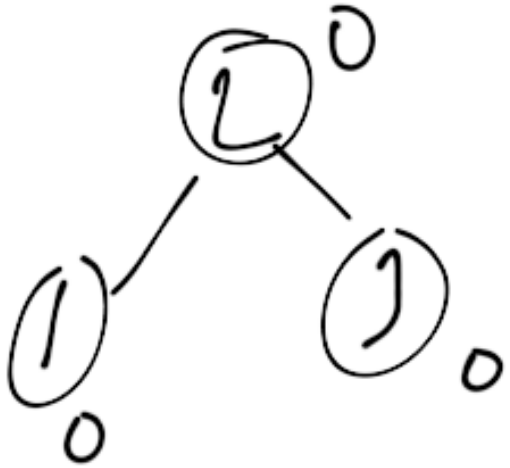
→ on the outside

→ single rotations

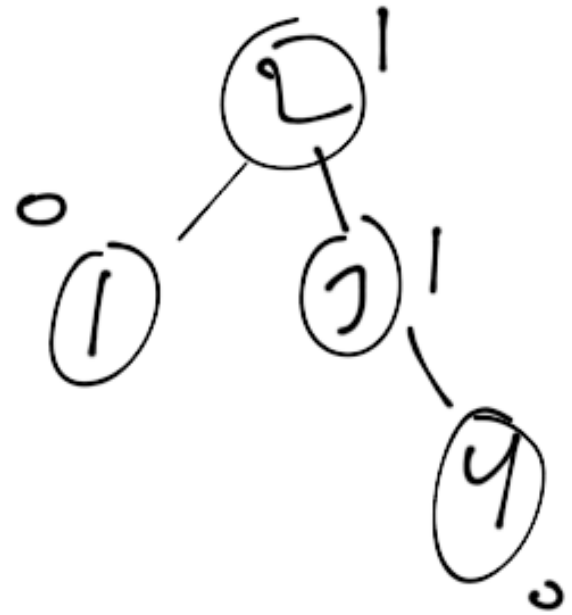
insert 3, 2, 1 :



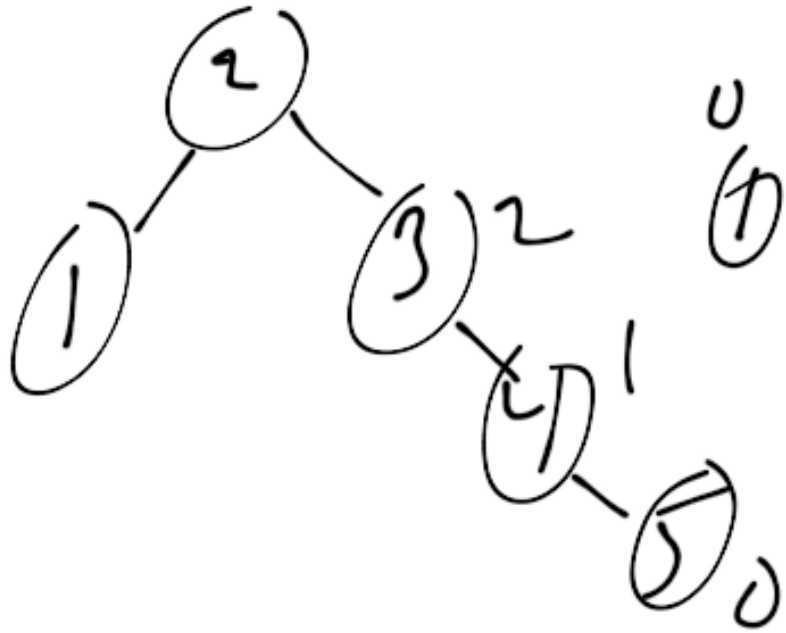
rotate_left :



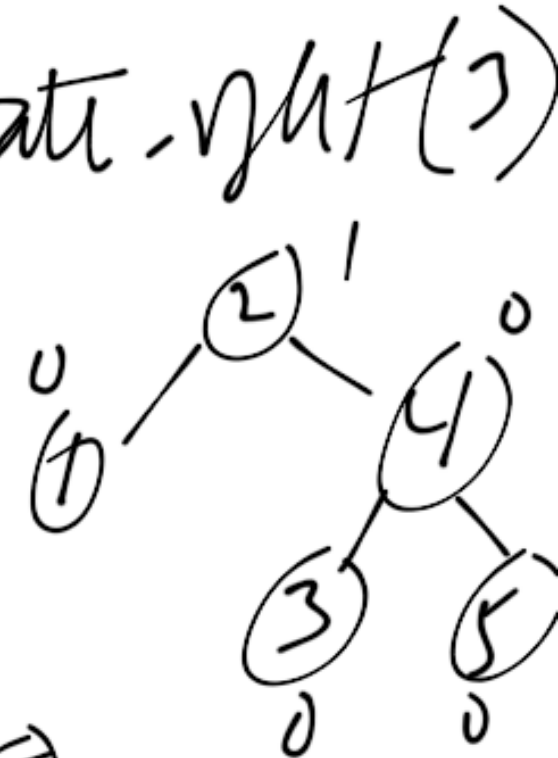
insert 4



insert 5

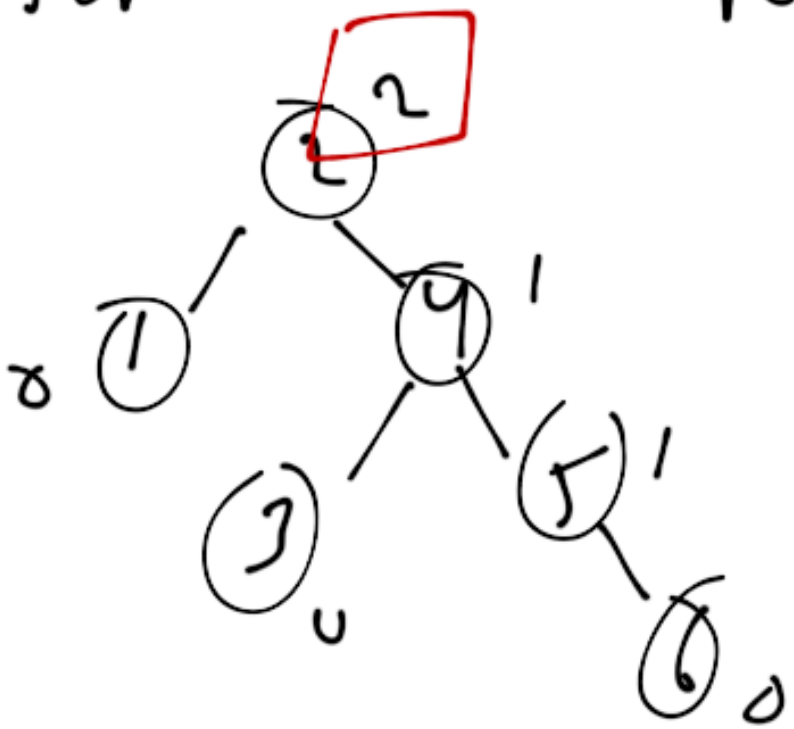


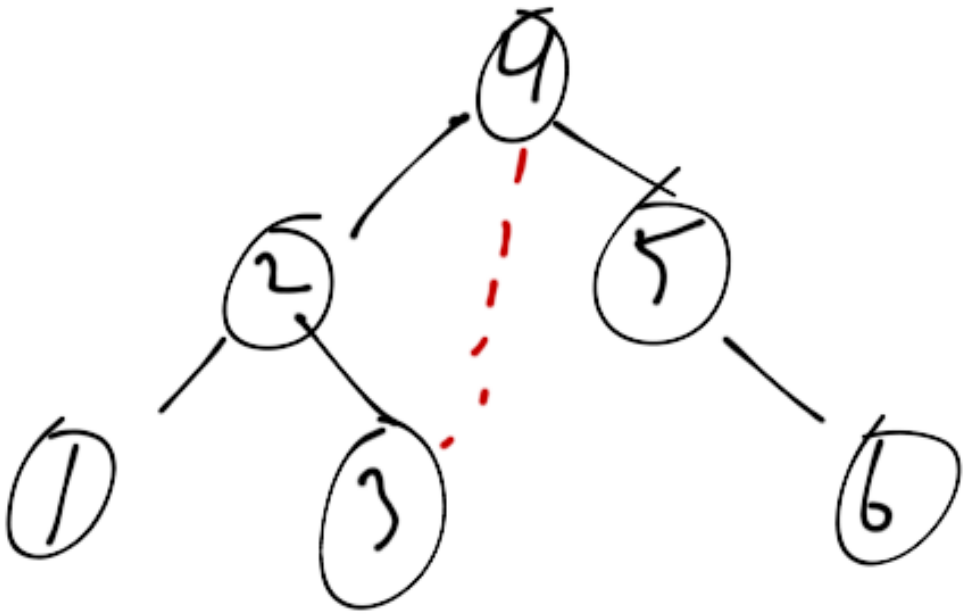
rotate right(3)



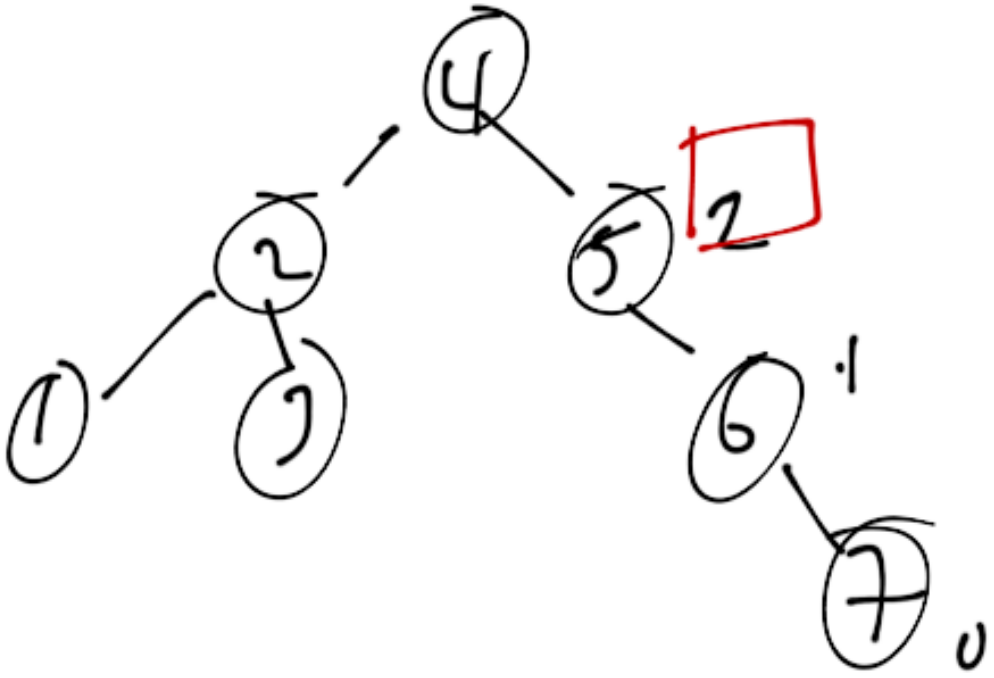
Insert 6 :-

Rotate - right (2)

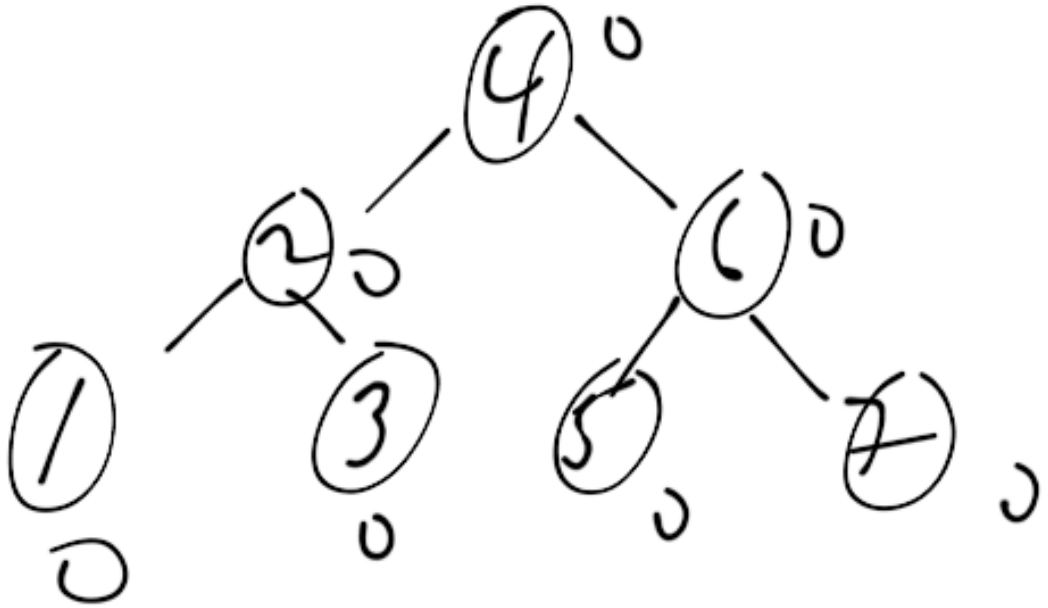




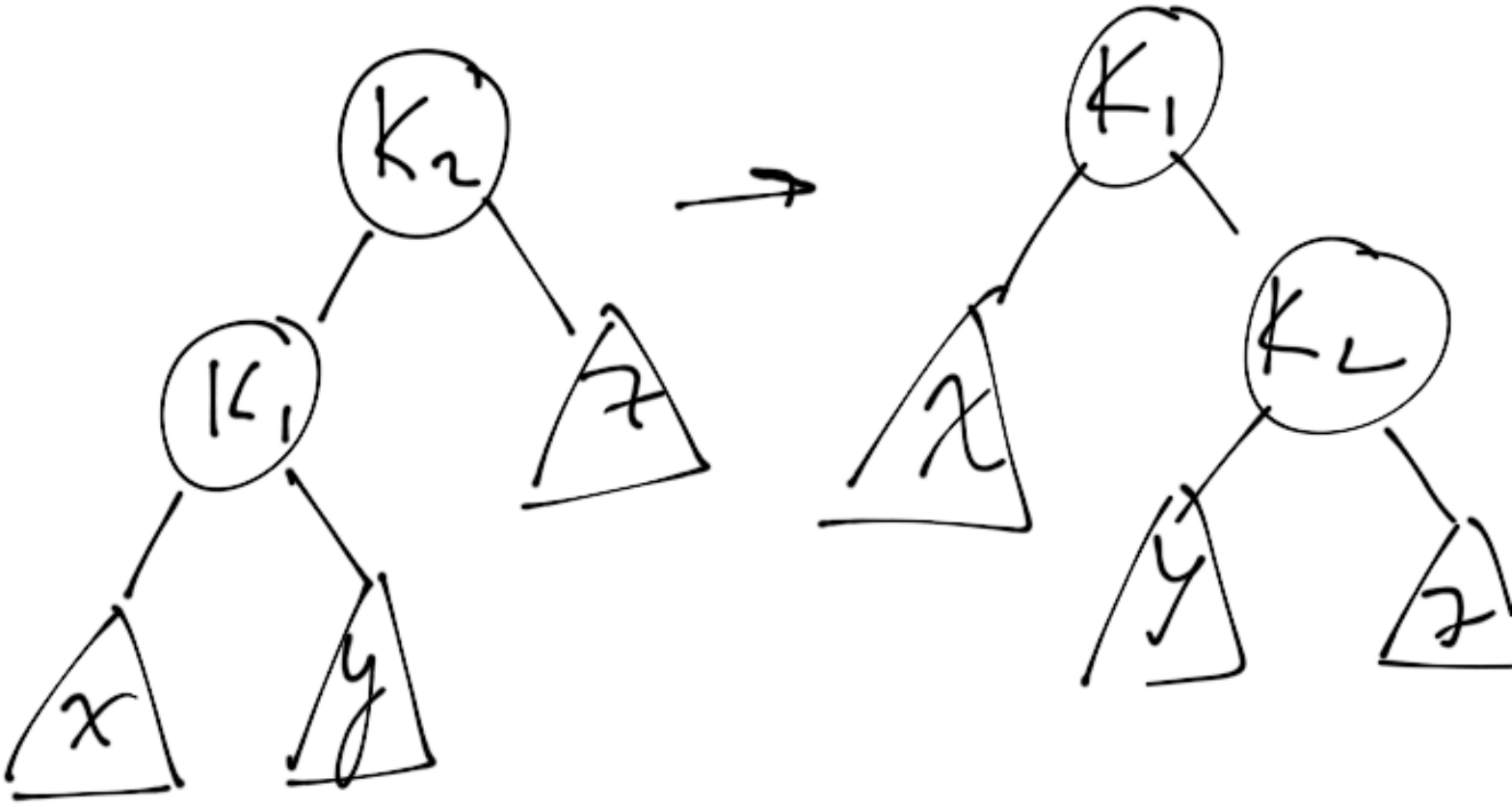
insert 7:



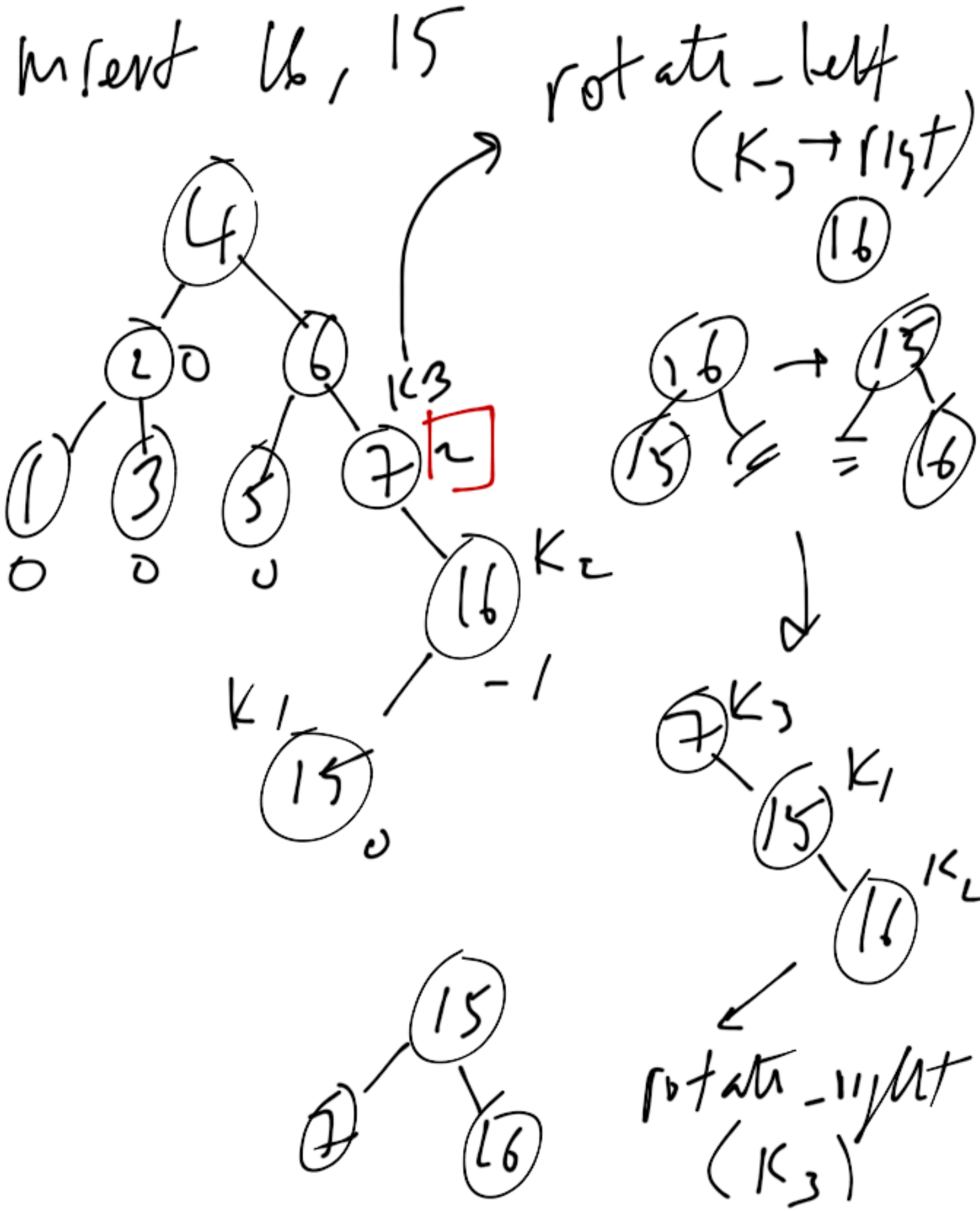
RotateRight(5)

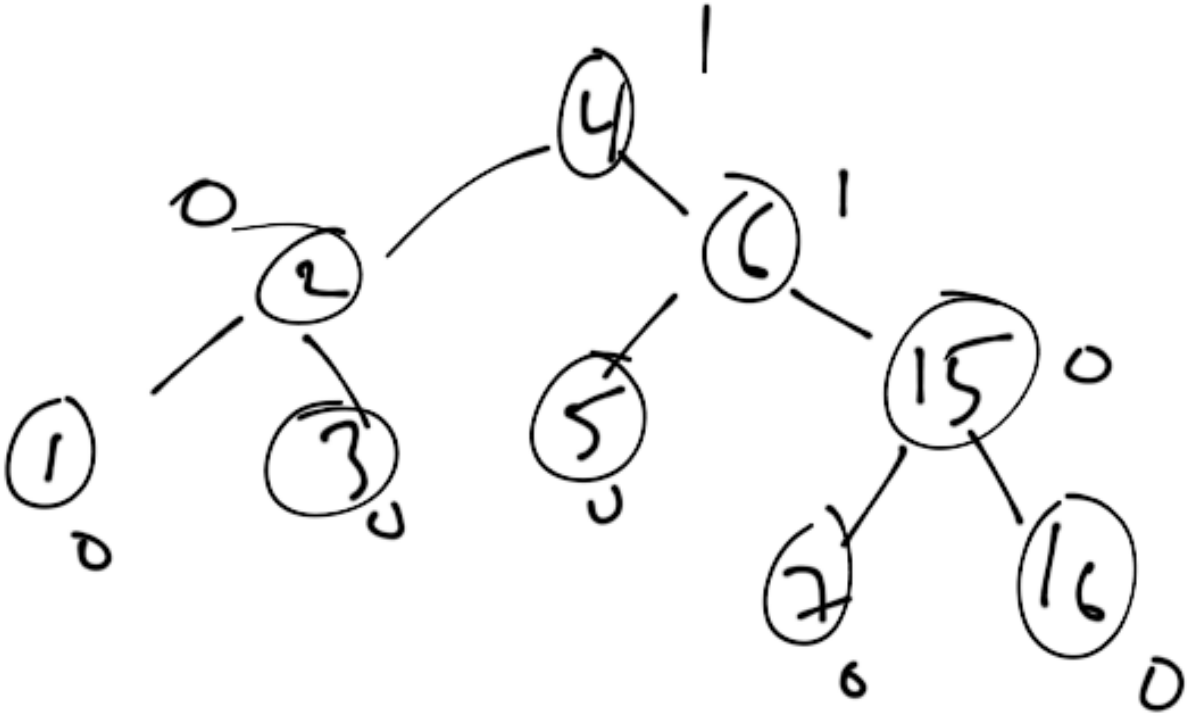


Single rotation



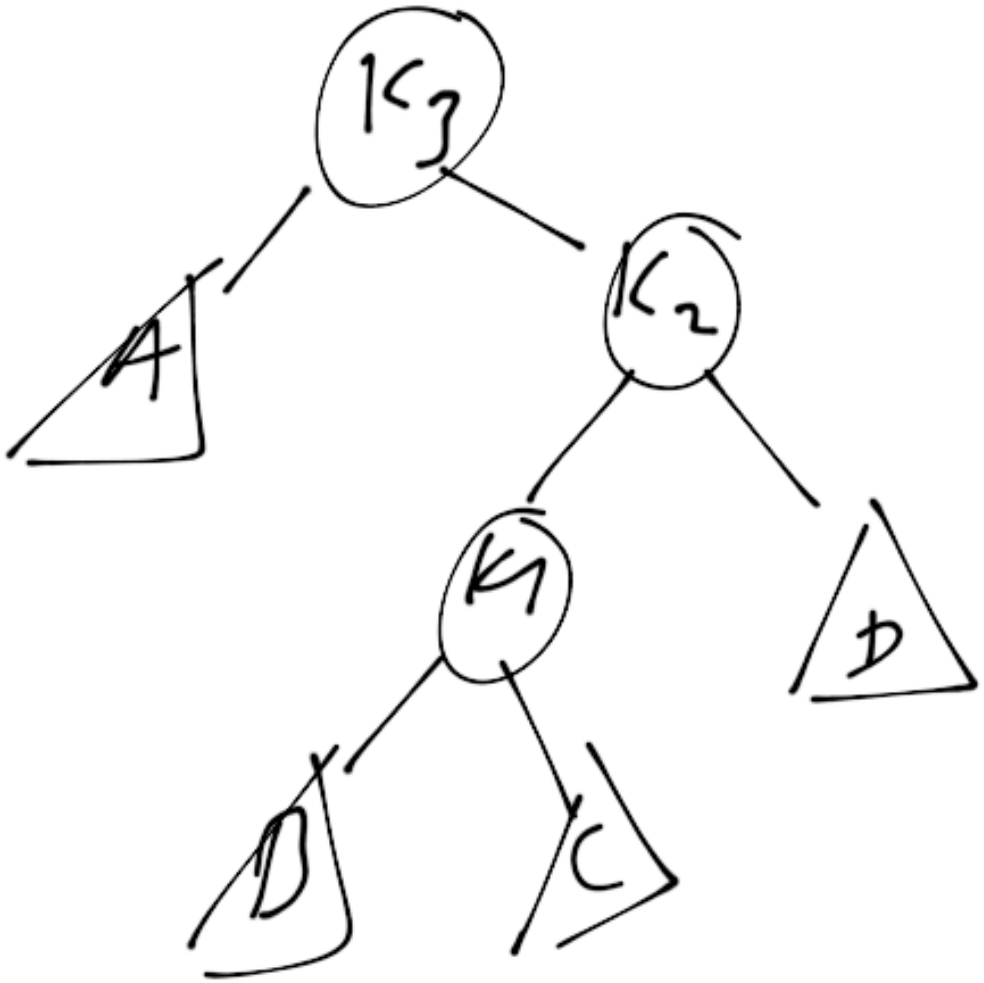
- insertion cases 2 & 3:
  insertion into internal
  nodes: double notation
  (two single notations
  in succession)


insert 16, 15
rotate_left
(K3 → right)
4
2
6
1
3
5
7
2
K3
0
0
0
16
K2
-1
K1
15
0
16
16
13
15
16
7
K3
15
K1
16
K2
rotate_right
(K3)
15
7
16

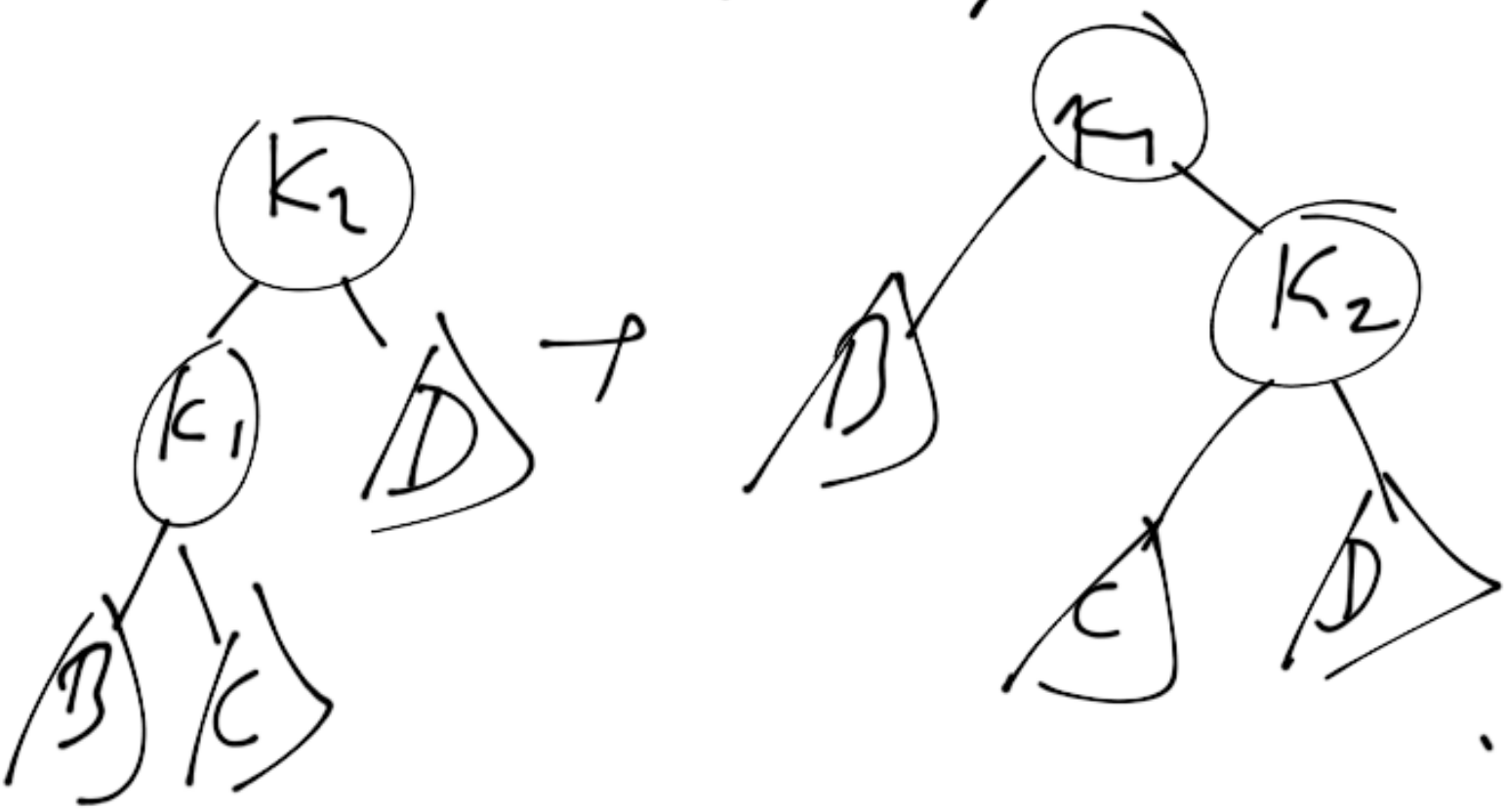

4
2
6
1
3
5
15
7
16

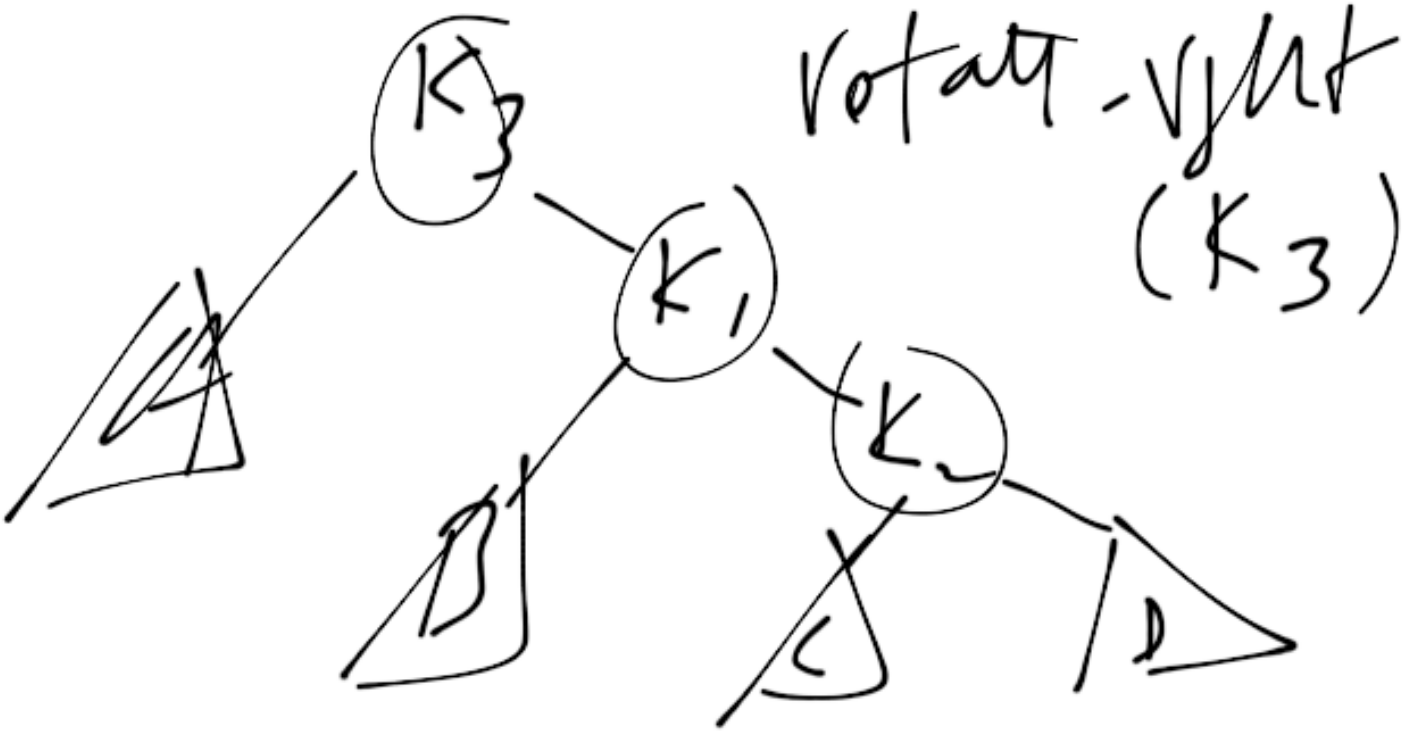
- in general, double-right rotation:

rotate_left(k_3 → right)



rotate-left(K_2)





rotate-right(K_3)

Rotate right (K_3)