Data structures & Algorithms

→ file

→ search

e.g. used in IBM Watson for speed

parallel cluster (100+ nodes?)

fast queries

e.g. in game theory chess

(file search)

BSTs:

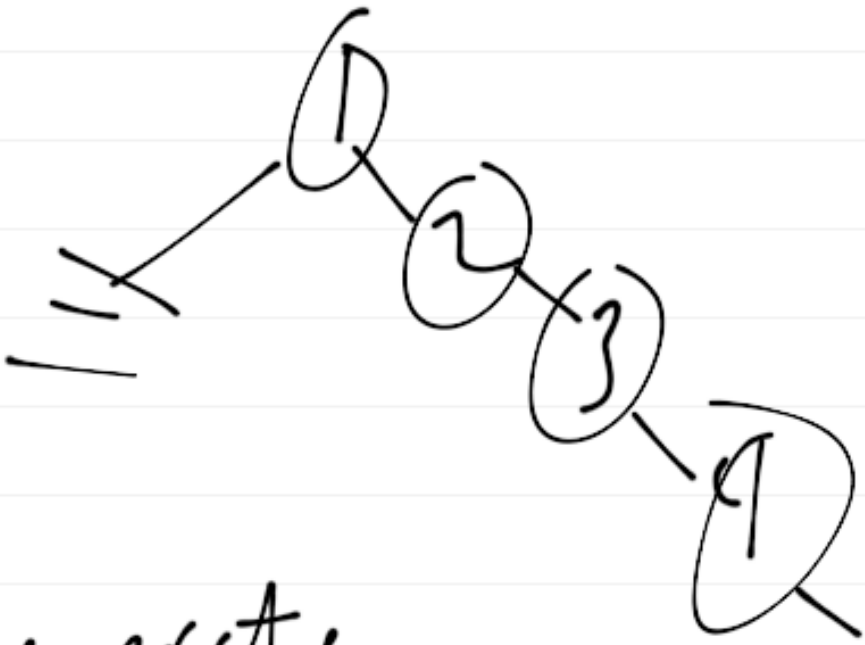
find_min: if tree empty ← default case

return t->left == NULL ?

t :

find_min(t->left);

(? : syntax)



degenerate

no $\Rightarrow$ unbalanced

- characteristic:

  $O(n)$ search per item

- balanced tree:

  $O(\lg n)$ search per item

(for n searches, $O(n^2)$
for unbalanced BST,
$O(n \lg n)$ for balanced
BST, AKA AVL tree)

BST search (for item x)

Contains (x, t)
↖ node

```
if (t == NULL) return false;
else if (x < t->data)
    return Contains(x, t->left)
else if (x > t->data)
    return Contains(x, t->right)
else return true;
```

in tree.h:

```
public:
    Contains(const T& x)
    { Contains(x, root); }

private:
    node_t* root;
    contains(const T& x,
             node_t*& t);
```

```
insert(x, t) :
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
```

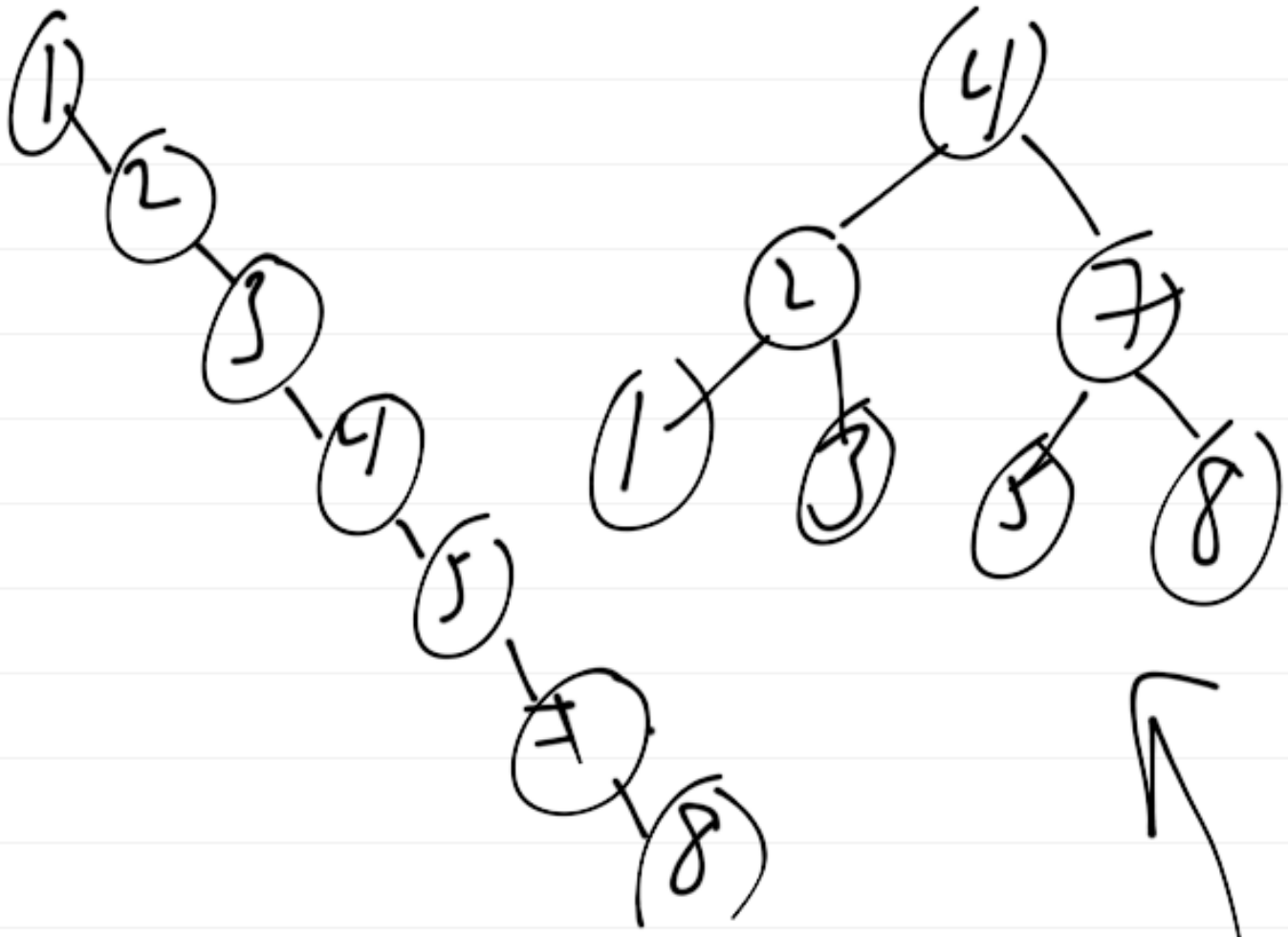
SPECIAL ANNOUNCEMENT:

MON 5:45-7:00pm

114 McADAMS

Dr. House's C to C++ transition lecture

BST insert: 1, 2, 3, 4, 5, 7, 8



BST degenerates to a list

Instead, want:

AVL tree insertion

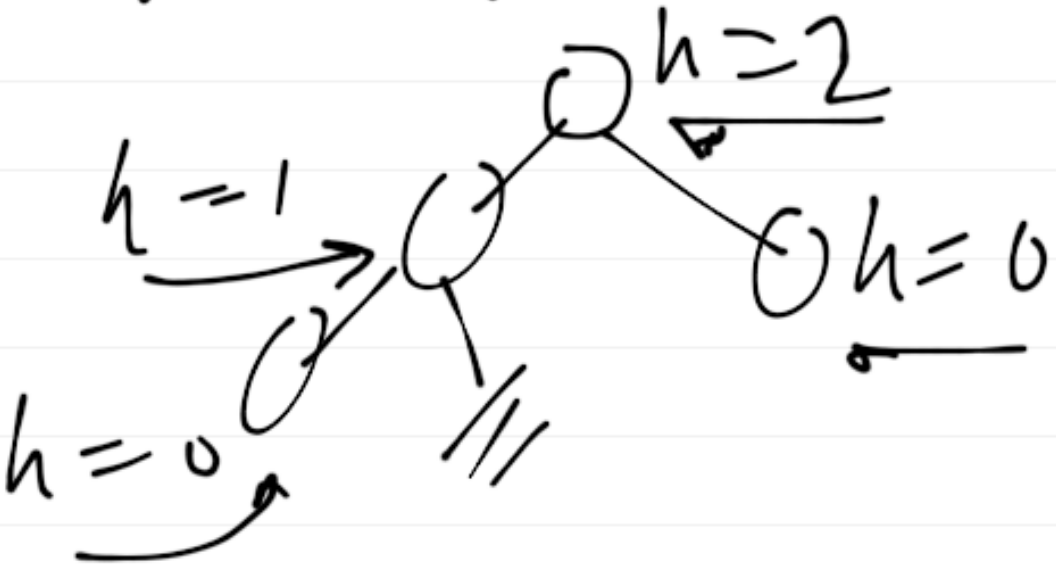
AVL Trees: like BSTs

but balanced:

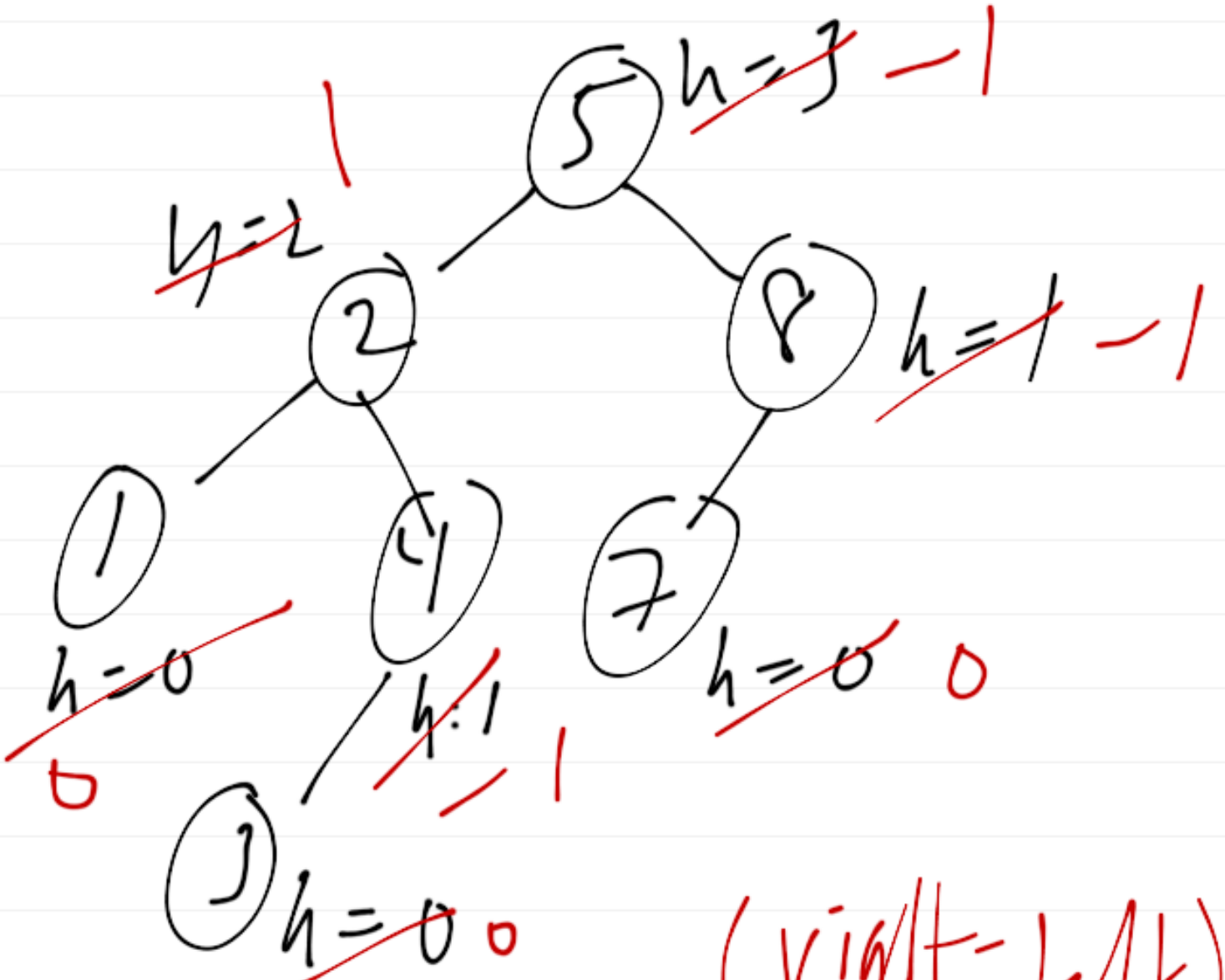
for every node, height

of left & right subtrees

differ by most 1



- My preference is to store balance info at nodes, not max heights like the text



(right - left)

bal: diff in subtree heights:

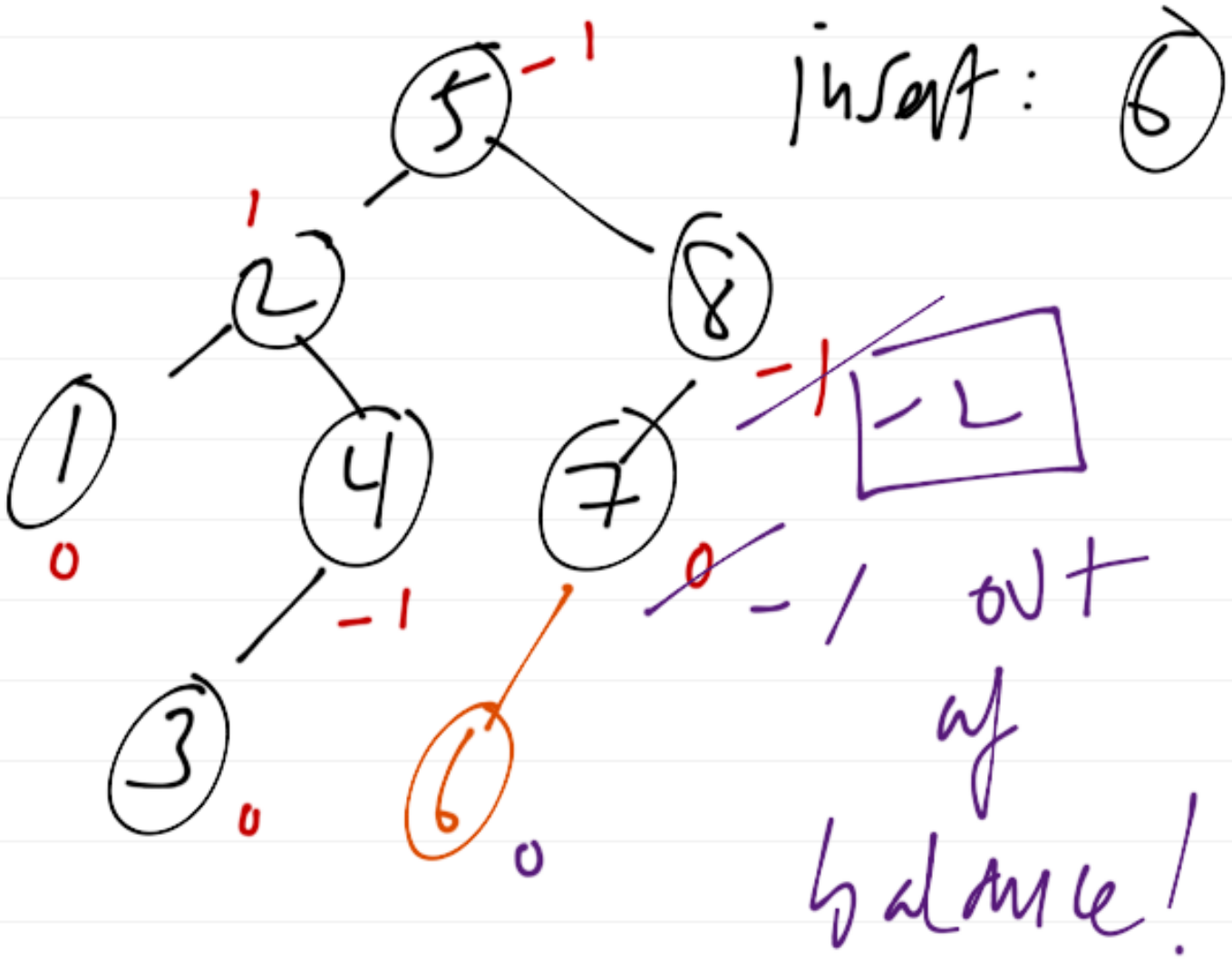
$-1, 0, 1$

left-heavy ↖ even ↑ ↗ right heavy

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

- insertion & deletion can disturb the balance

- when inserting, need to update the bal of each node on the path back up to the root

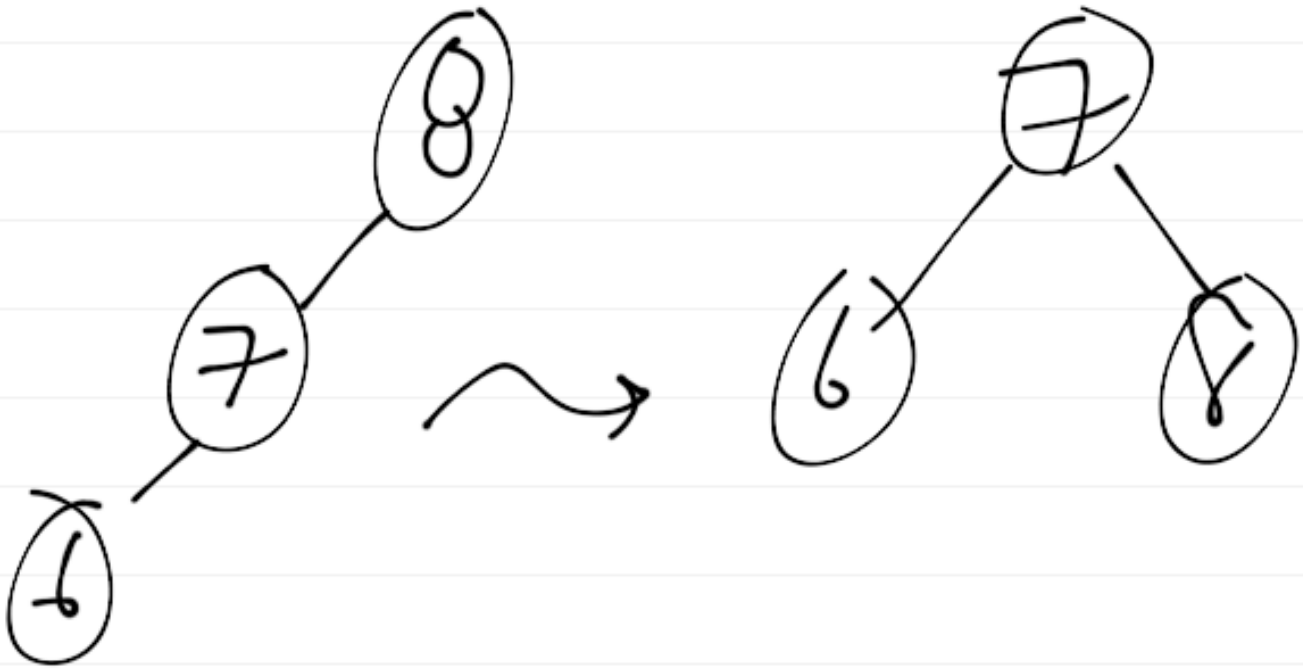


need to rebalance

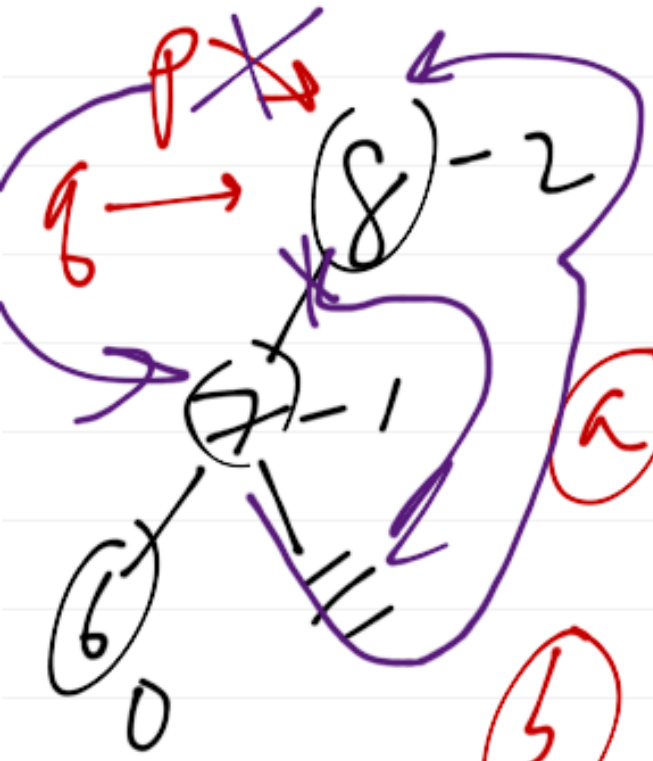
rotate_left (8)

rotate left subtree to the right

rotate_left(8)



single rotation (to the right) with left subtree



rotate_left(p)

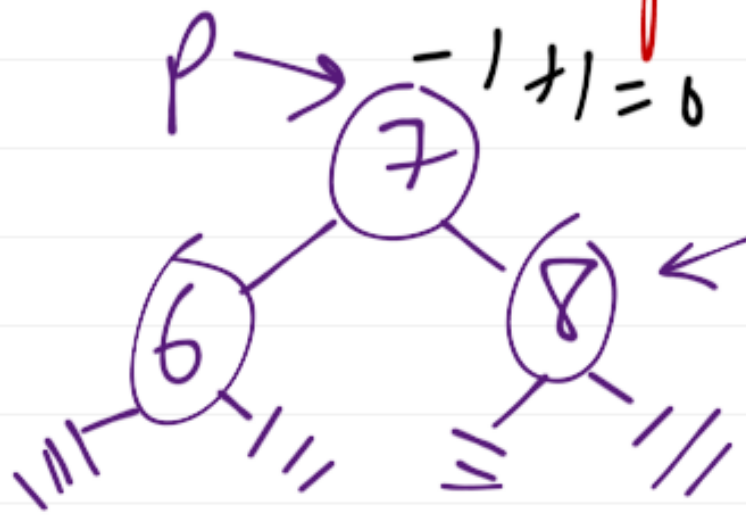
① $q = p$

② $p = p \rightarrow left$

③ $q \rightarrow left =$

$p \rightarrow right$

④ $p \rightarrow right = q$



q $<2-1+1$

$=0$

(e) q->bal++ // left sibling reward

(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:

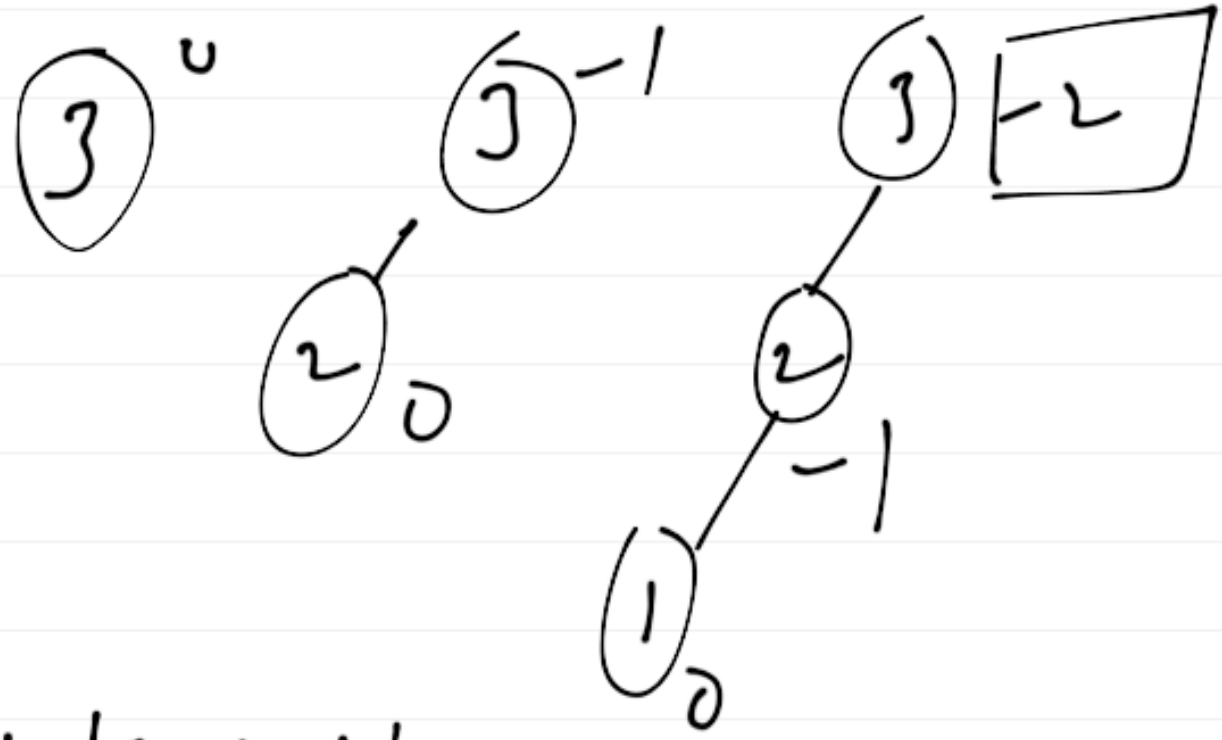
left subtree of left child

right subtree of right child

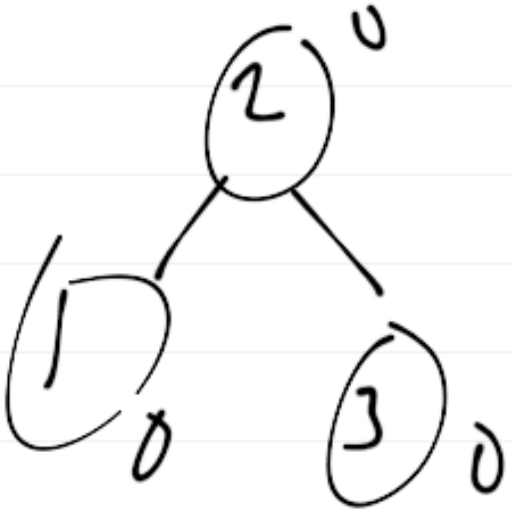
→ on the outside

→ single rotations

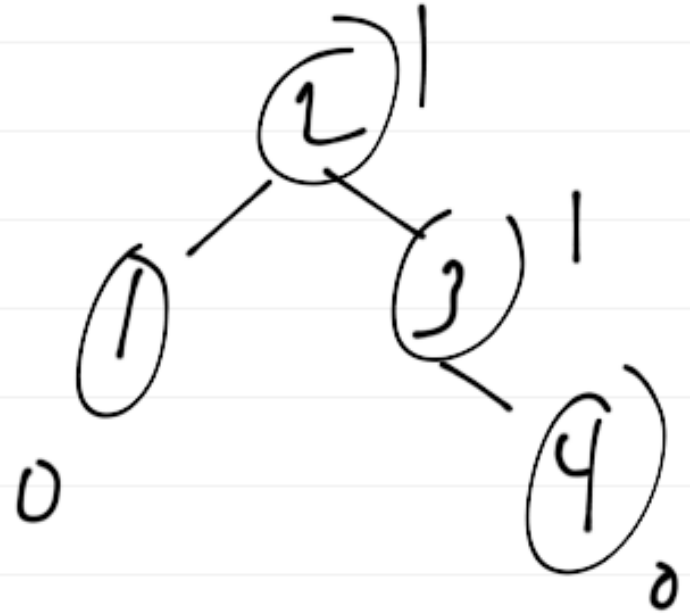
- insert 3, 2, 1:



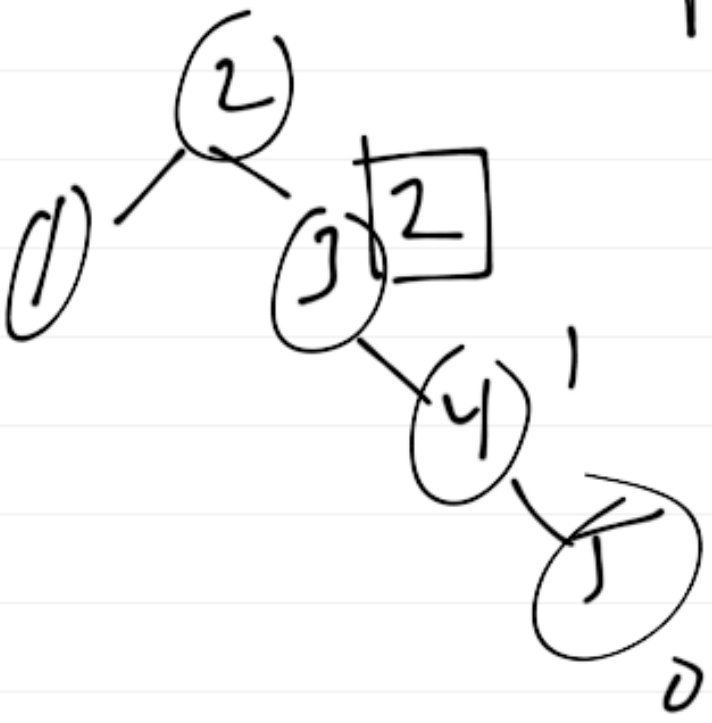
Rotate_left:



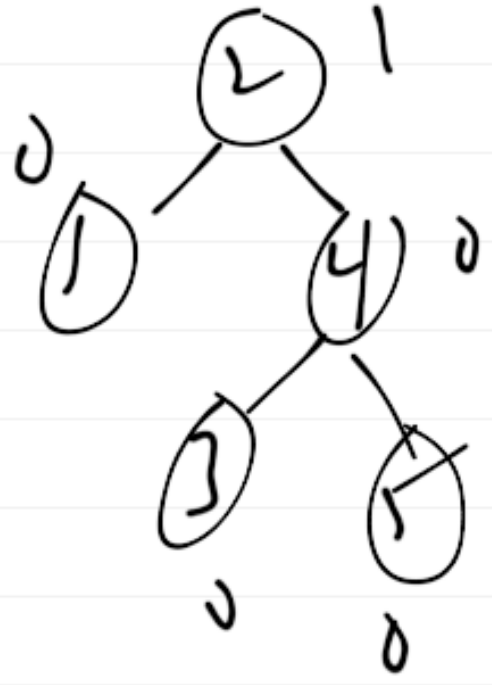
insert 4:



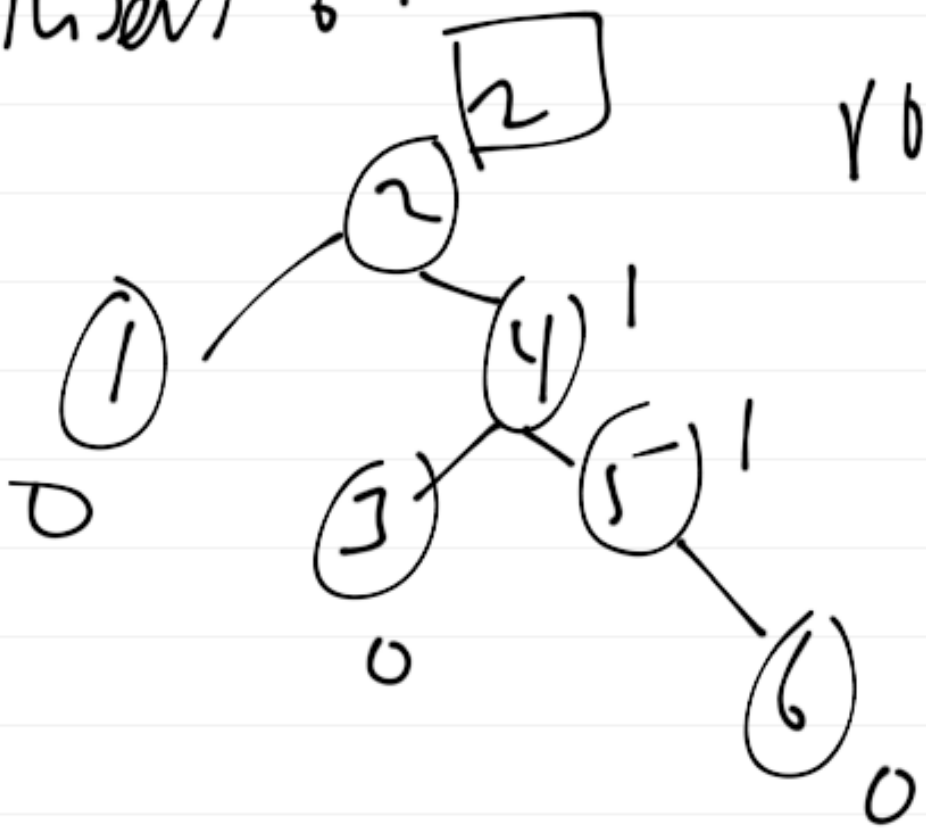
Insert 5:



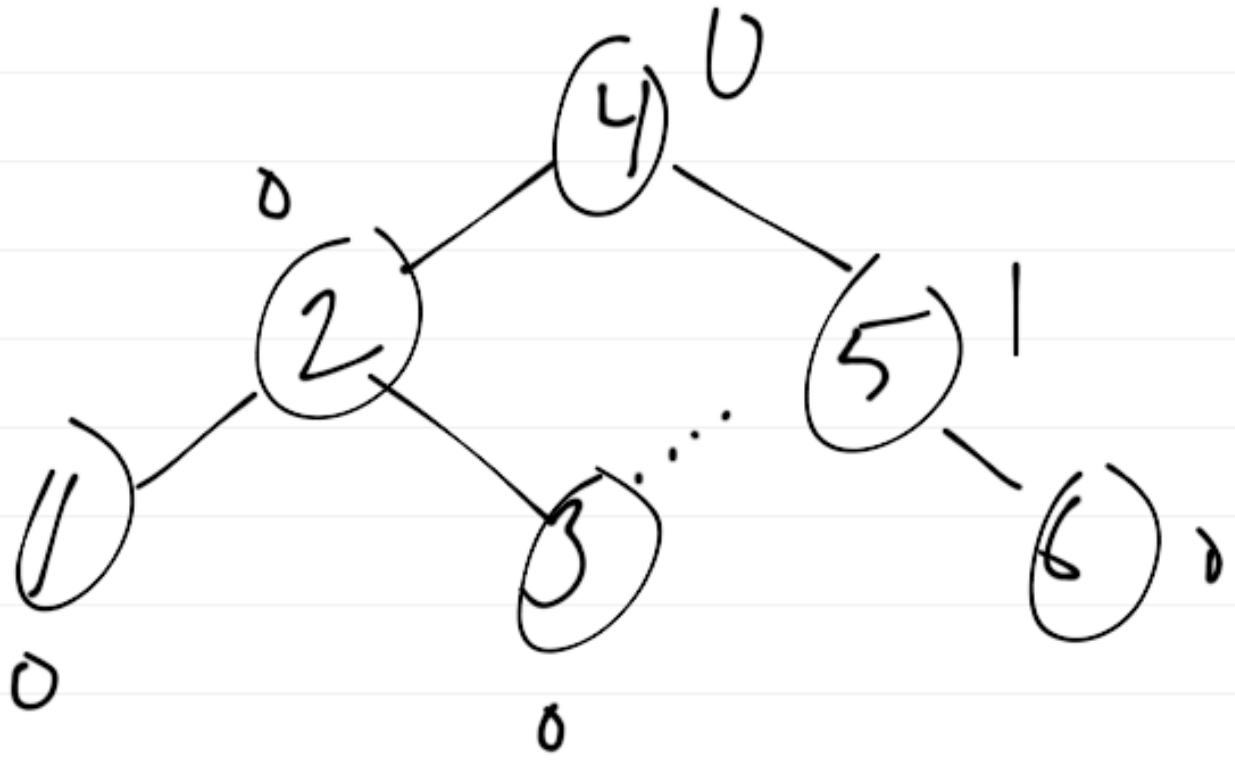
rotate_right(3)



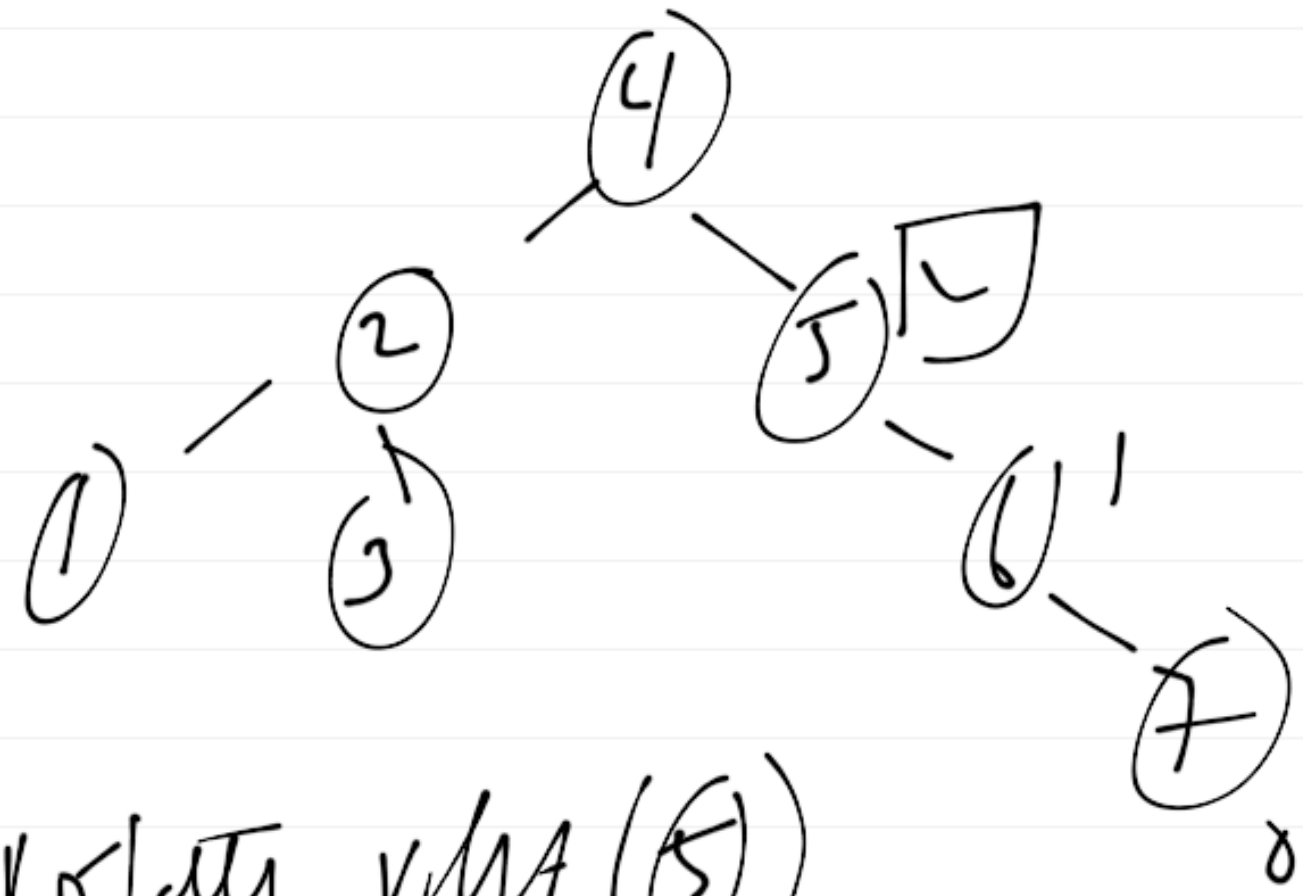
insert 6:



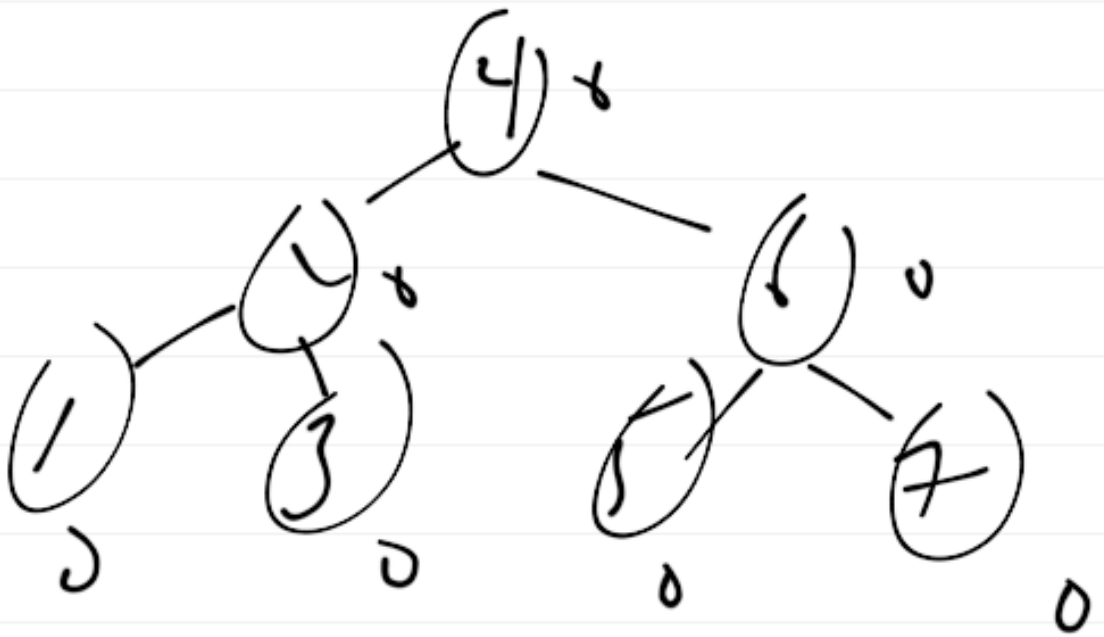
rotate_right(2)



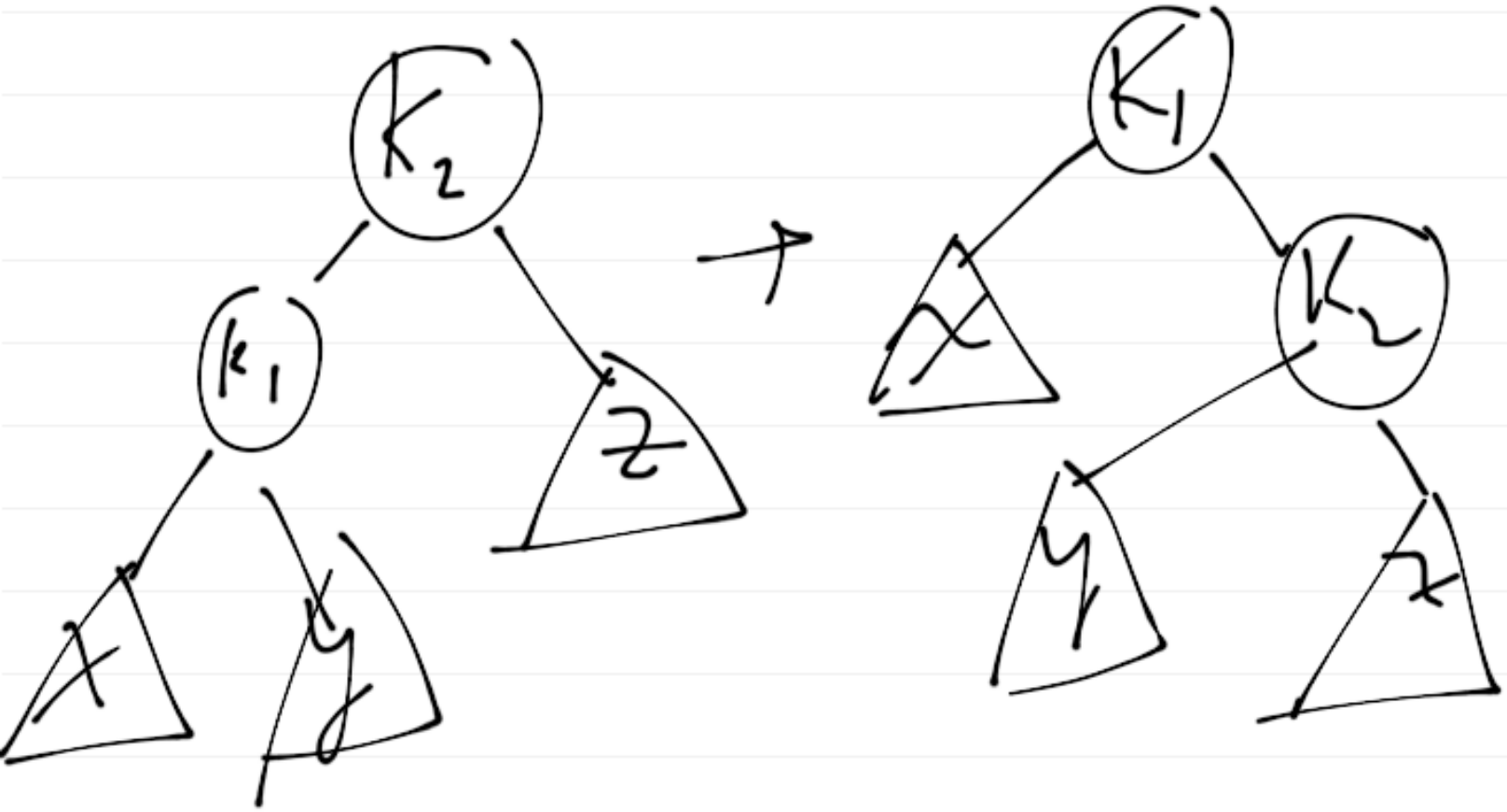
insert 7:



rotate right(5)



Single rotation (rotate_left)



- insertion cases 2 & 3:

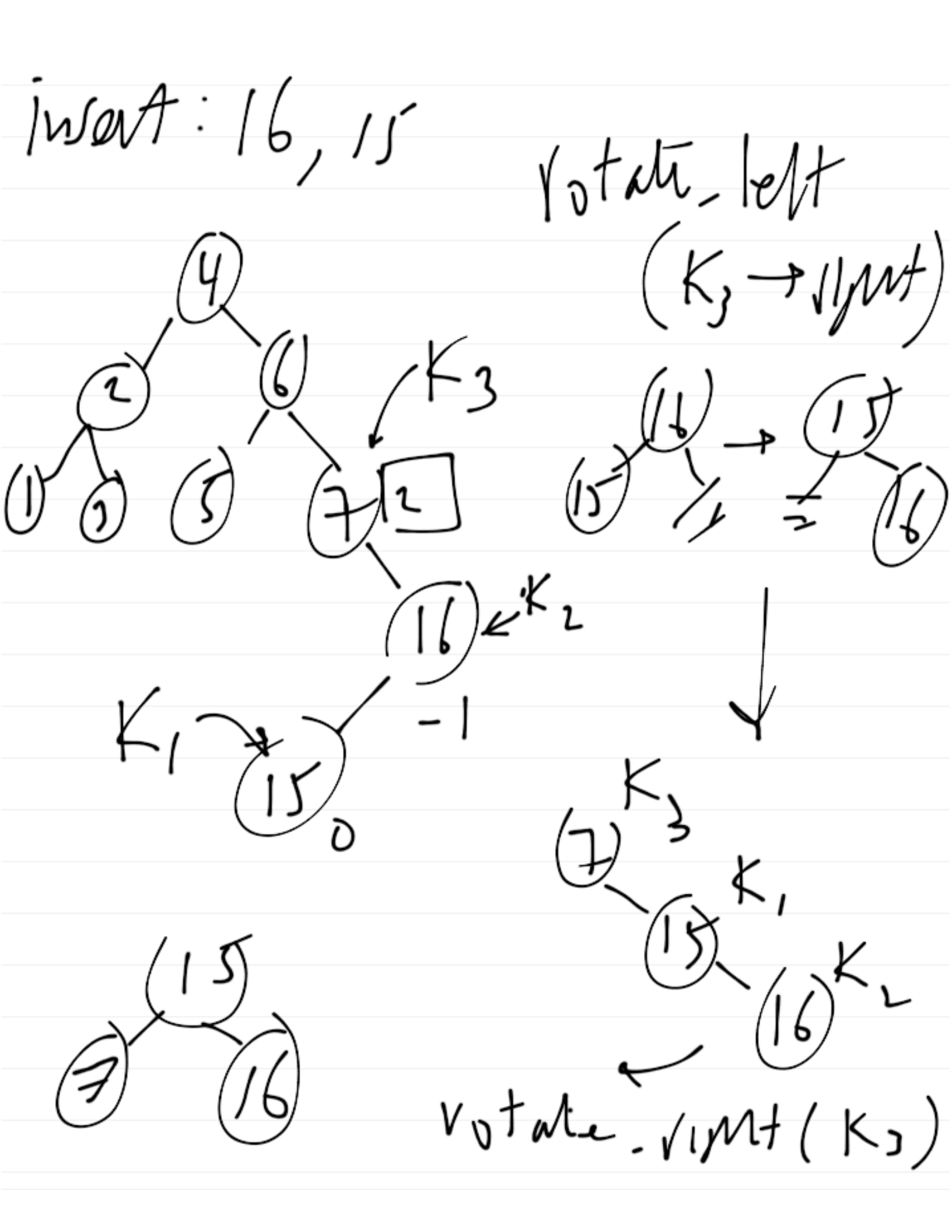
  insertion into interior

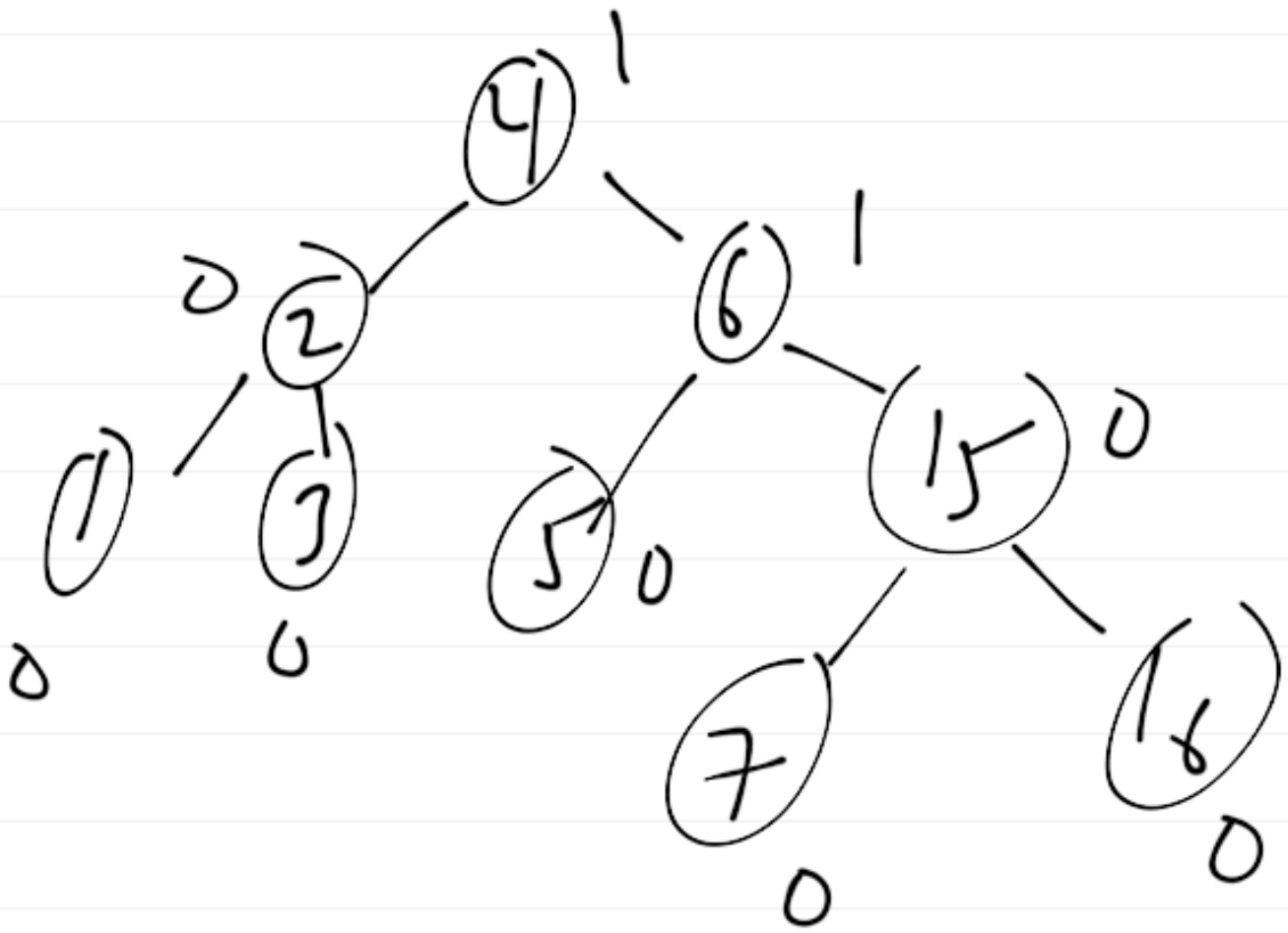
  nodes: double rotation

  (two single rotations

  in succession)

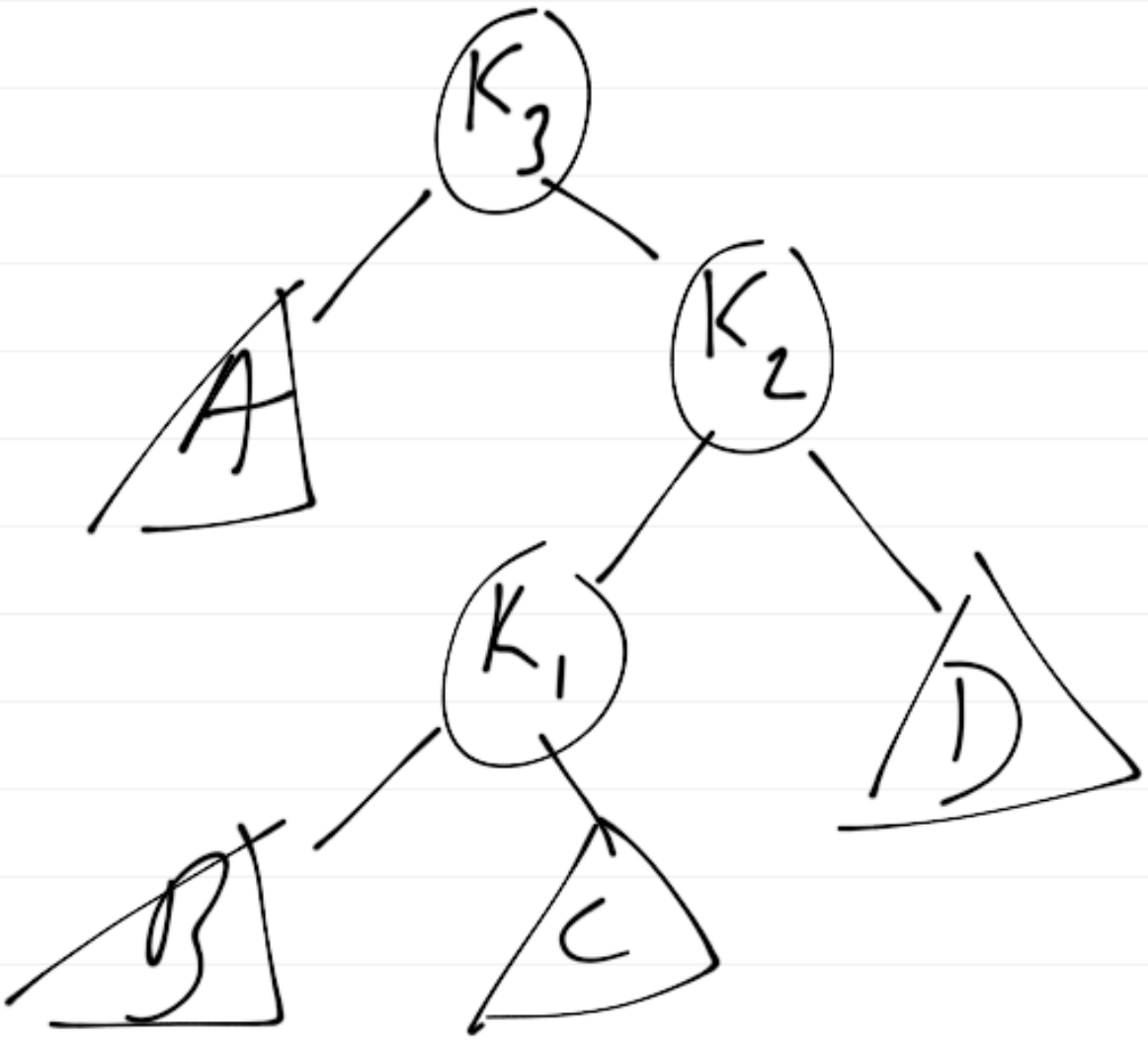
insert: 16, 15



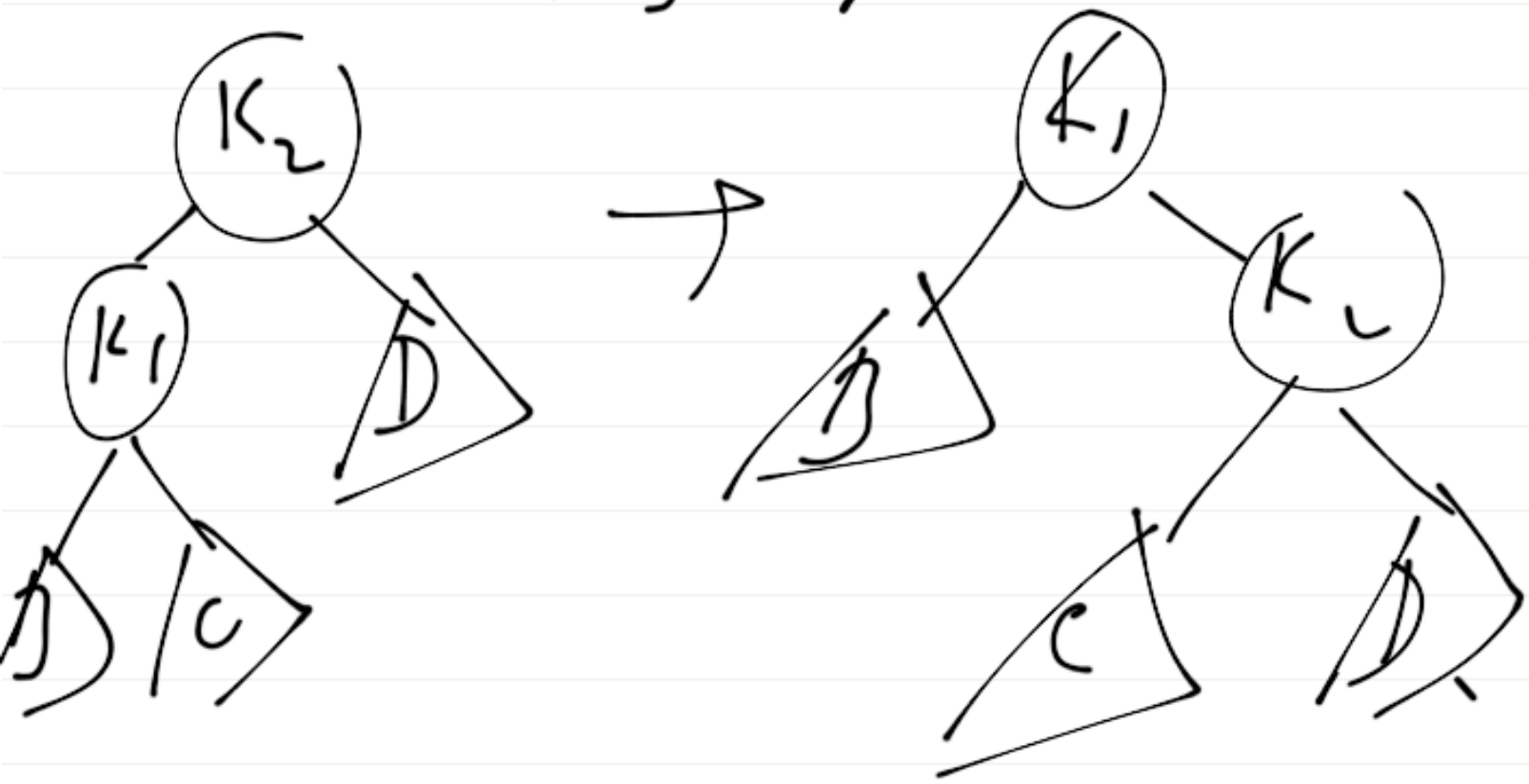

4
1
0
2
6
1
1
3
5
0
15
0
0
0
7
0
16
0

- In general, double_right rotation

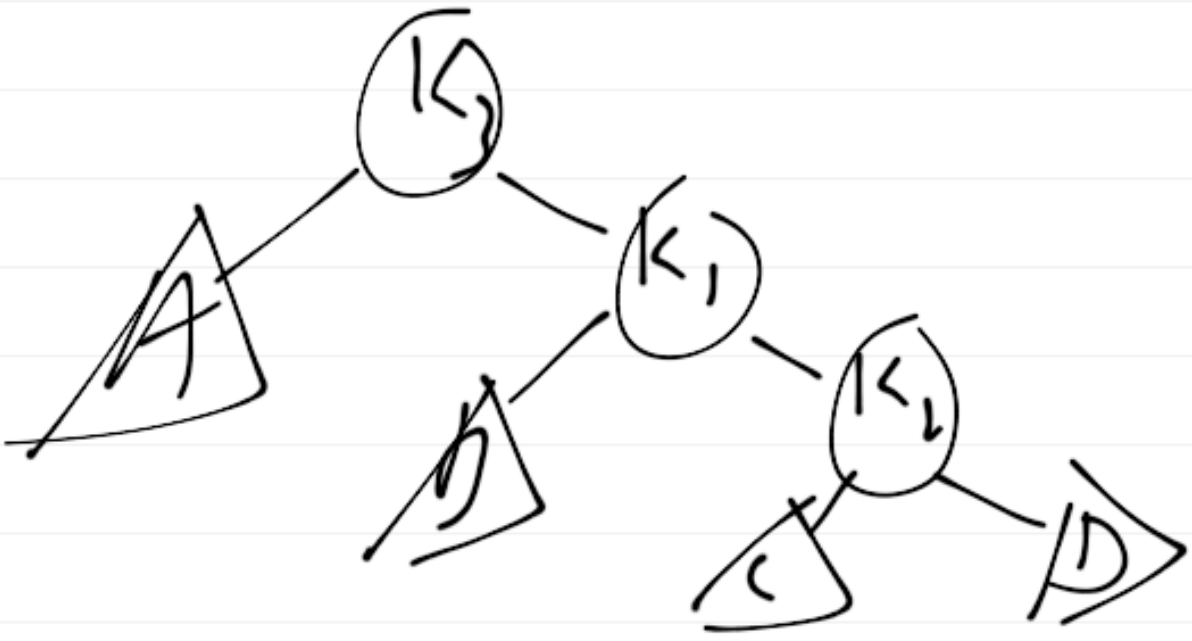
rotate_left($K_3 \rightarrow$ right)



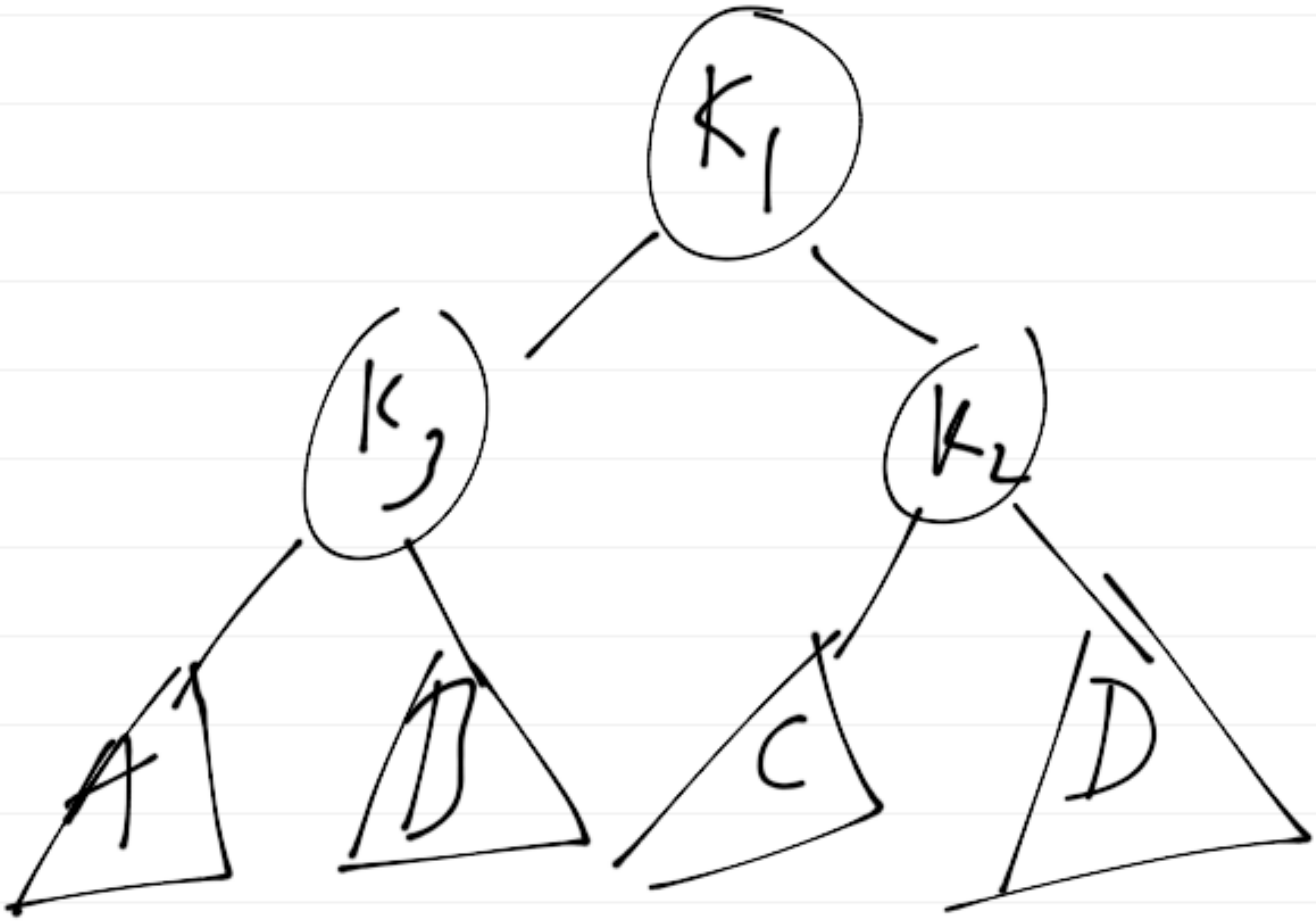
rotate_left(k_3 → 右旋)



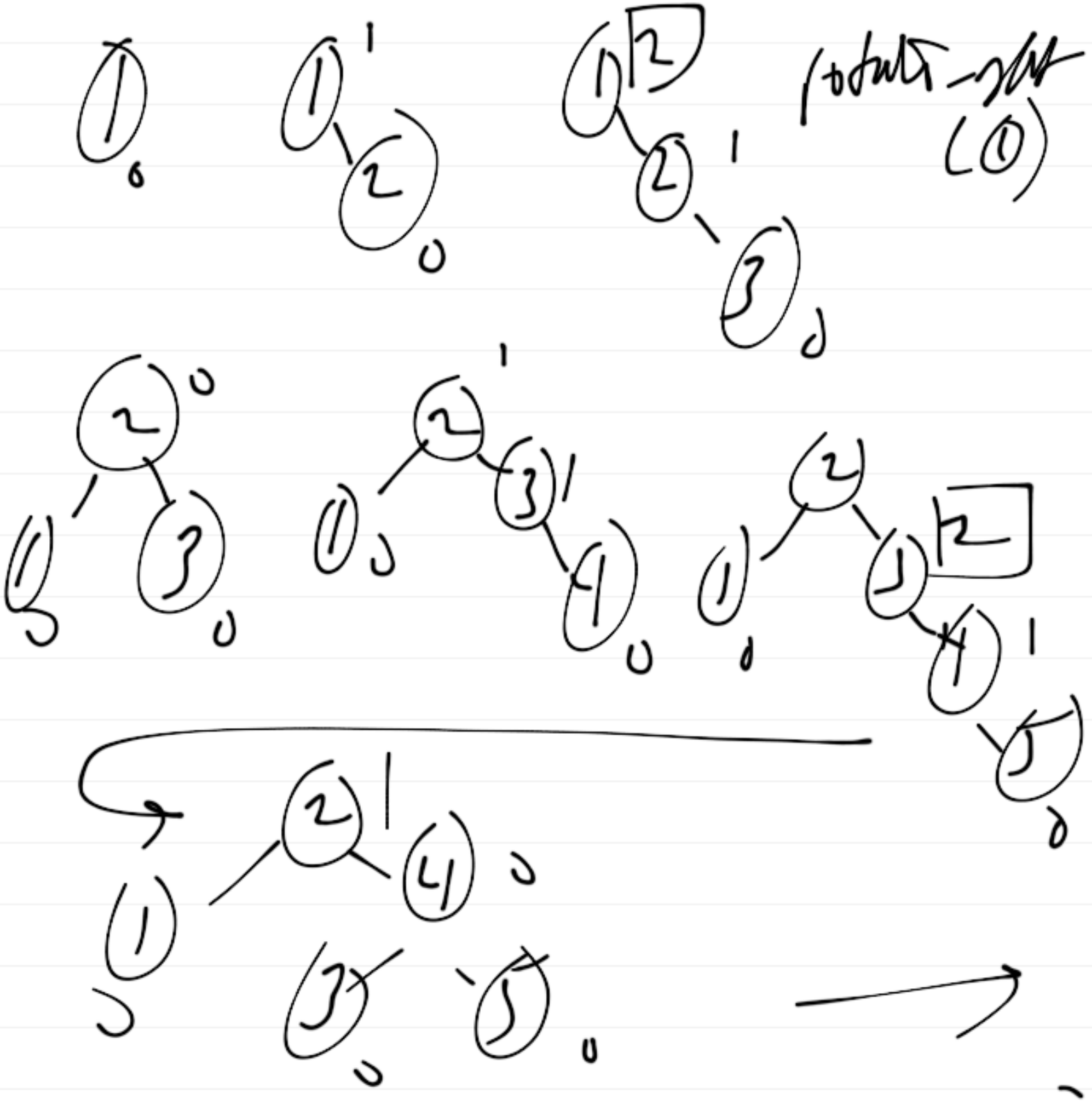
rotate_right(k_3)

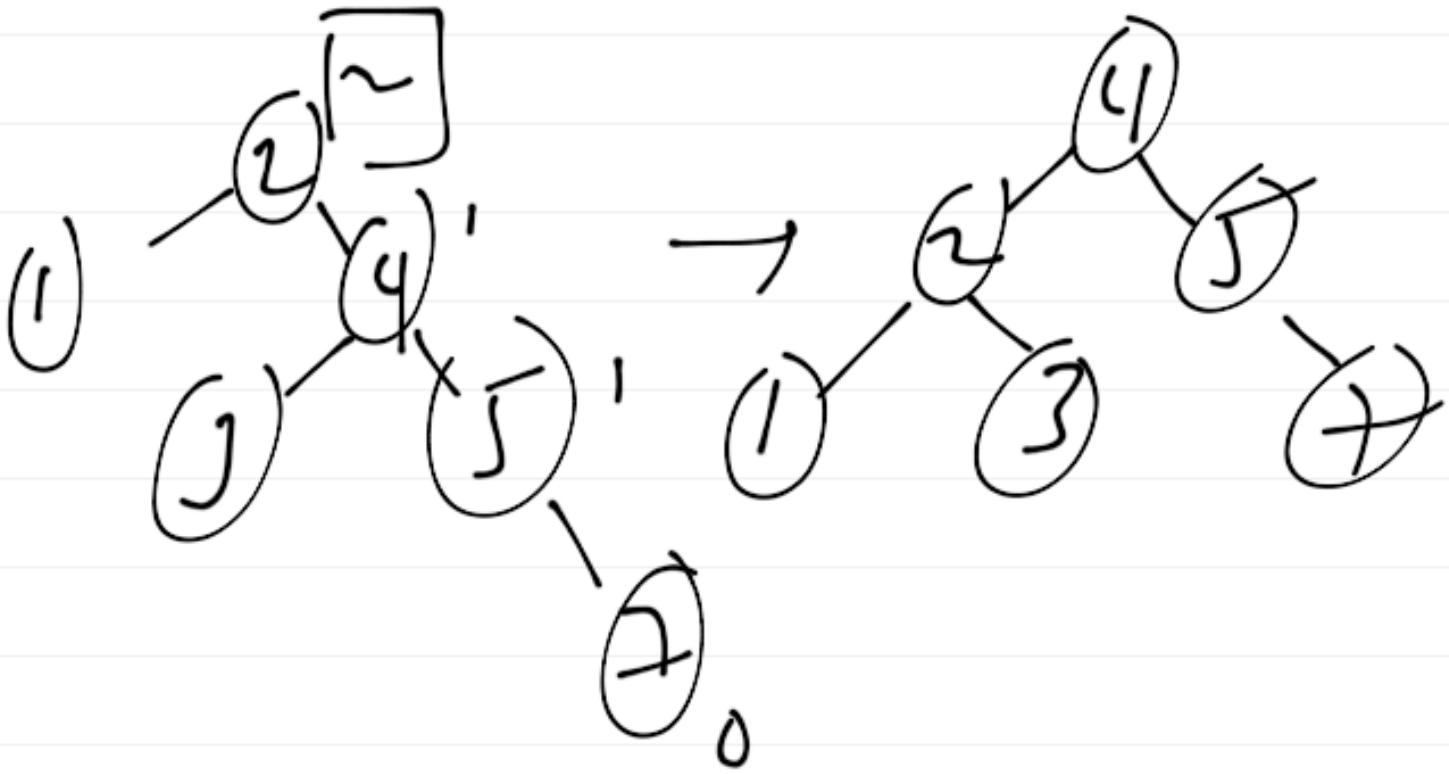


rotate_right(K_3)



Insert 1, 2, 3, 4, 5, 7, 8





insert 6: