DATA STRUCTURES

UNIT-4

CLASS NOTES

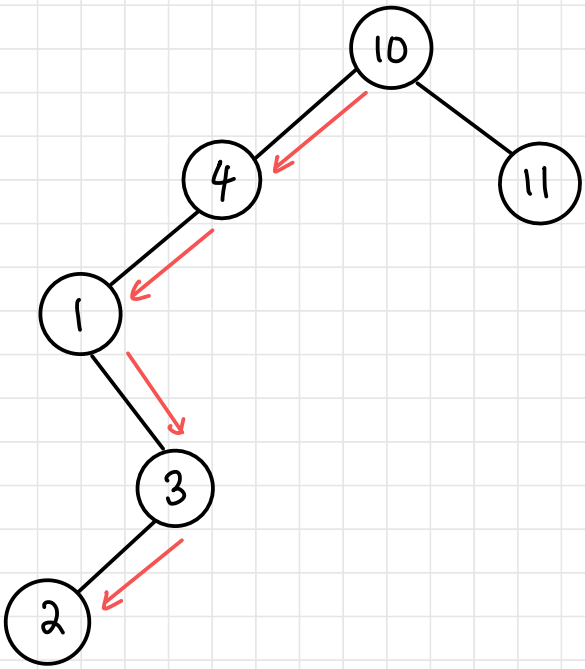
BALANCED TREES

feedback/corrections: vibha@pesu.pes.edu

Vibha Masti

BALANCED TREES

Binary Search Tree



to find element 2,
4 traversals needed

Worst Case: O(n)
Best Case: O(log(n))

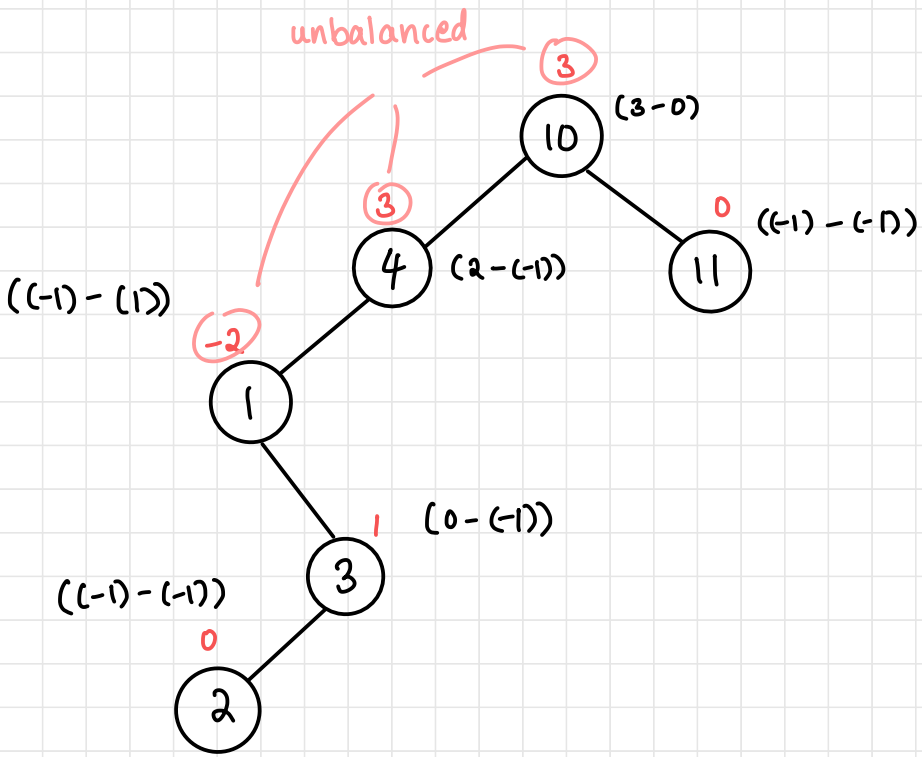
- The tree above is a BST but it is not a very efficient BST for searching
- If elements are added randomly, then searching a BST is more efficient than an array
- If elements are added in order, BSTs are a slower implementation of linear arrays / lists
- Height of BST depends on the order of insertion and can be as bad as n-1 for n elements

Balanced BST - AVL Trees

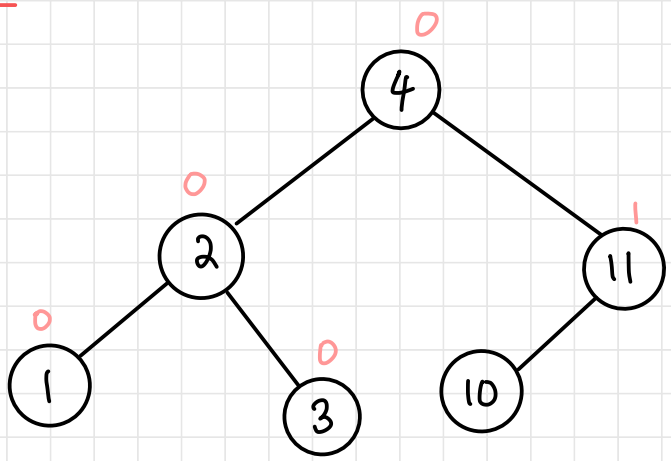
- Here, the height of the tree is always log(n) [empty → -1]
- AVL tree : GM Adelson-Velskii, EM Landis (two Russian mathematicians)
- Balance factor of a node is always -1, 0 or +1

> balance factor = height (left subtree) - height (right subtree)

Not an AVL Tree



AVL Tree



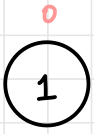
ROTATION OF TREES

- To convert a BST to an AVL Tree
- Left rotation, right rotation (single rotation), double rotations

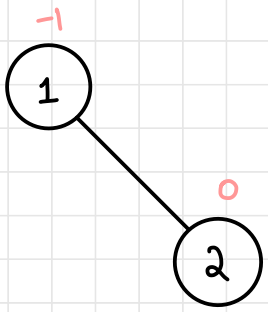
Left Rotation

- Constructing an AVL tree
- Performed when new key inserted into right subtree of right child of a node whose balance factor was -1 before insertion
- Tree is heavy on right side

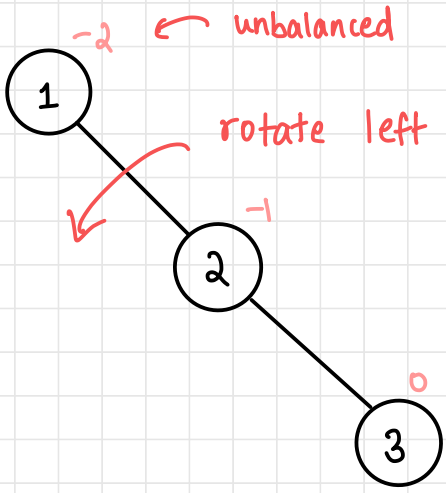
1) Insert 1



2) Insert 2



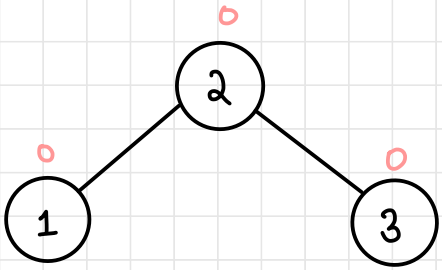
3) Insert 3



- Need to balance - perform left rotation
- New key inserted into right subtree of right child of a node whose balance factor was -1 before insertion
- Perform L(1) as balance factor of 1 changed from -1 to -2

4) Perform Left Rotation

L(1)

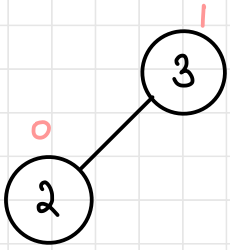


Right Rotation

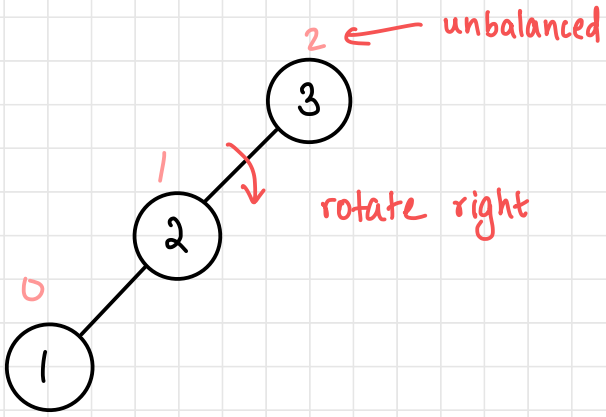
- Constructing an AVL tree
- Performed when new key inserted into left subtree of left child of a node whose balance factor was +1 before insertion
- Tree is heavy on left side

1) Insert 3

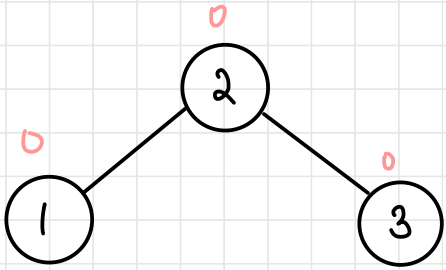
2) Insert 2



3) Insert 1



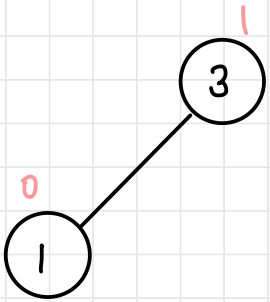
- Need to balance
- New key inserted into left subtree of left child of a node whose balance factor was +1 before insertion
- Perform R(3) as balance factor of 3 changed from 1 to 2



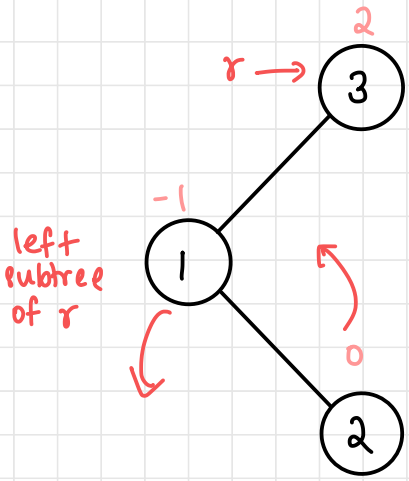
Left Right Rotation

- New node inserted into right subtree of left child of a node whose balance factor was +1 before insertion
- Left rotation on the left subtree of r (the one that becomes unbalanced is r)
- Then followed by right rotation of r

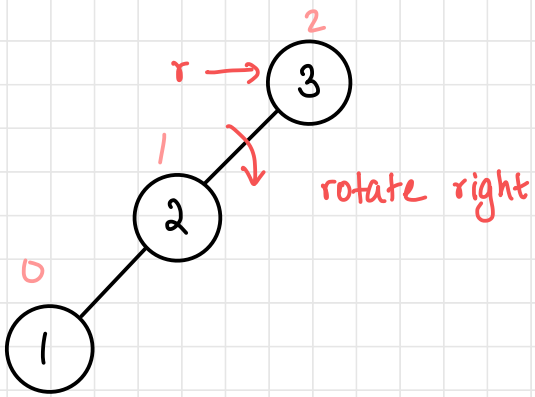
1) Tree before insertion



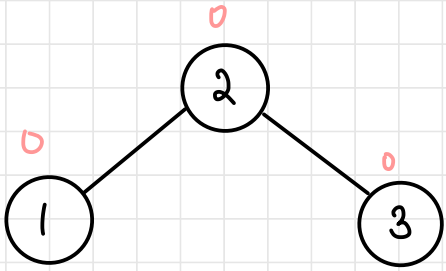
2) Insert 2



3) L(1)



4) R(3)

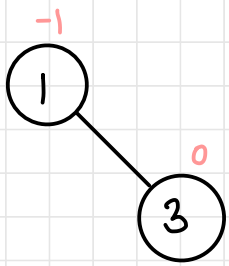


- We performed LR(3)

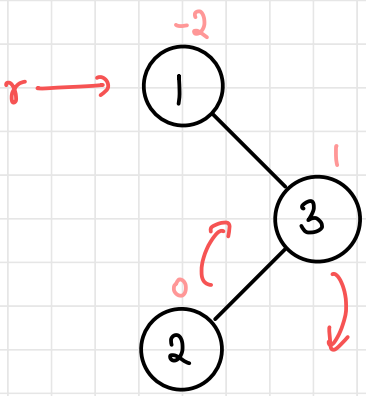
Right Left Rotation

- New node inserted into left subtree of right child of a node whose balance factor was -1 before insertion
- Right rotation on the right subtree of r (the one that becomes unbalanced is r)
- Then followed by left rotation of r

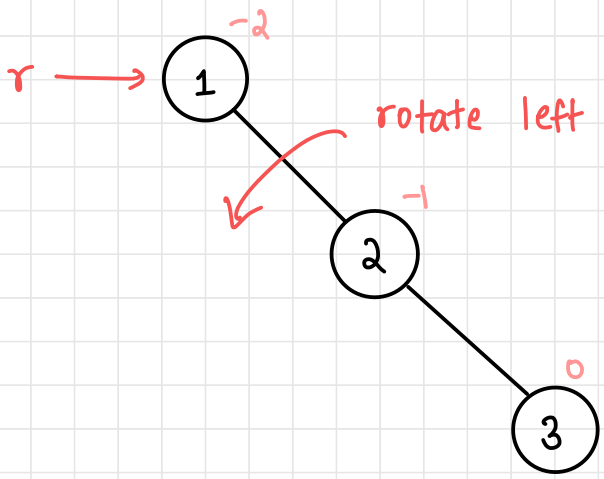
1) Tree before insertion



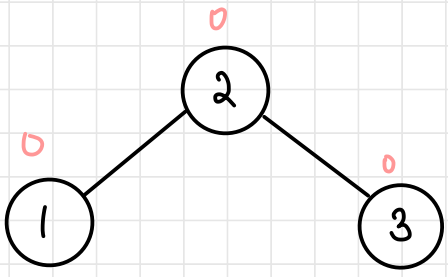
2) Insert 2



3) R(3)



4) L(1)



- We performed RL(1)

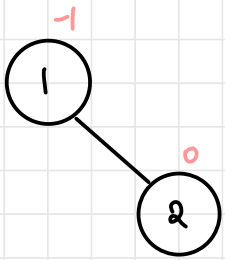
INSERTING A NODE INTO AN AVL TREE

Question 1

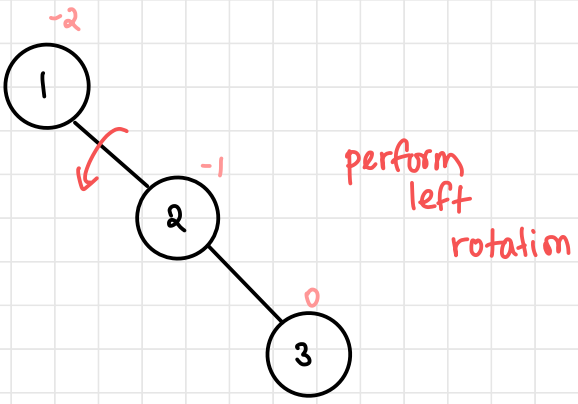
Convert list = 1, 2, 3, 4, 5, 6 to AVL

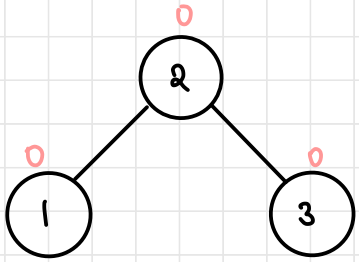
1) Insert 1

2) Insert 2

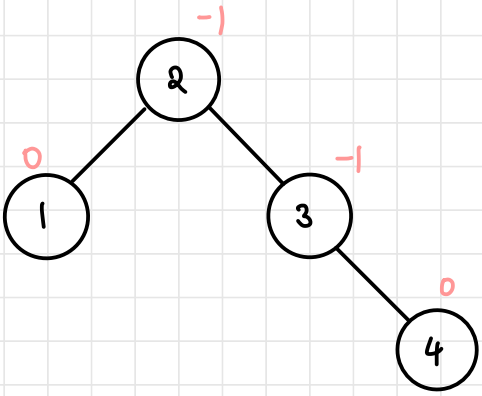


3) Insert 3

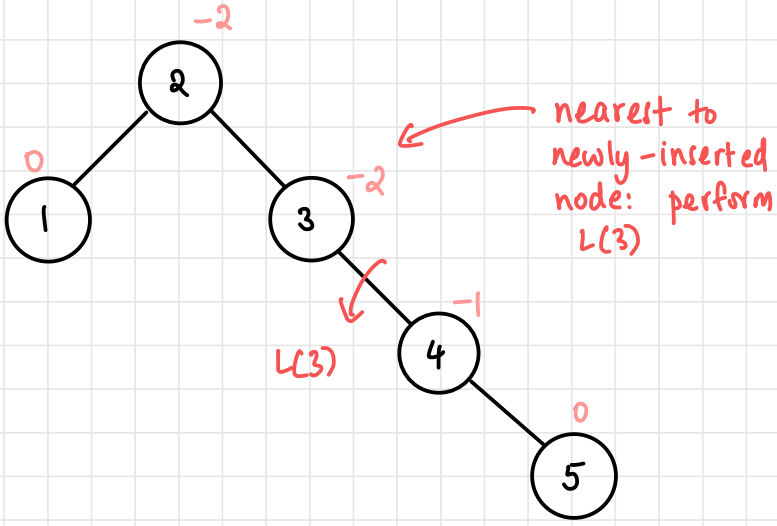


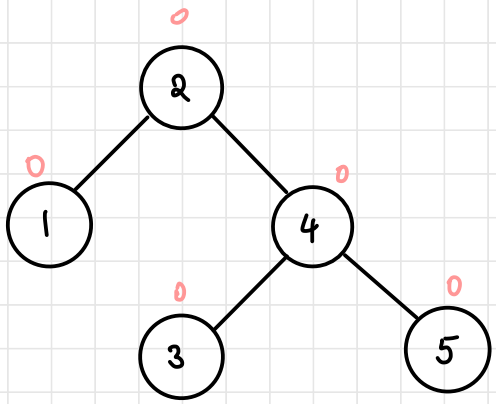


4) Insert 4

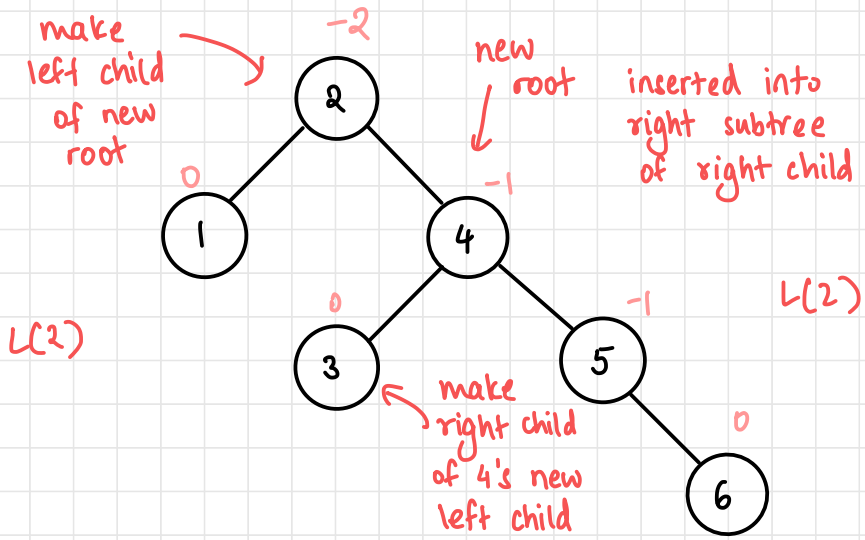


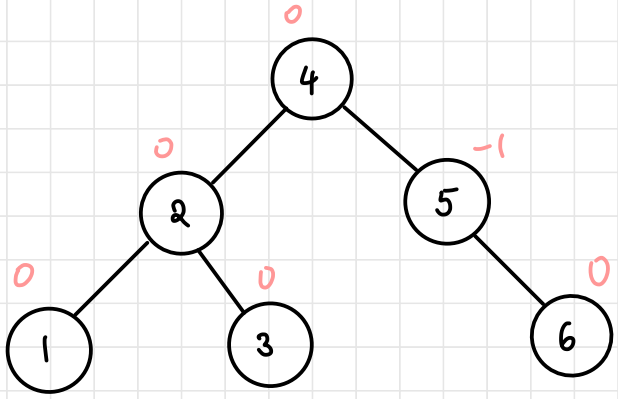
5) Insert 5





6) Insert 6

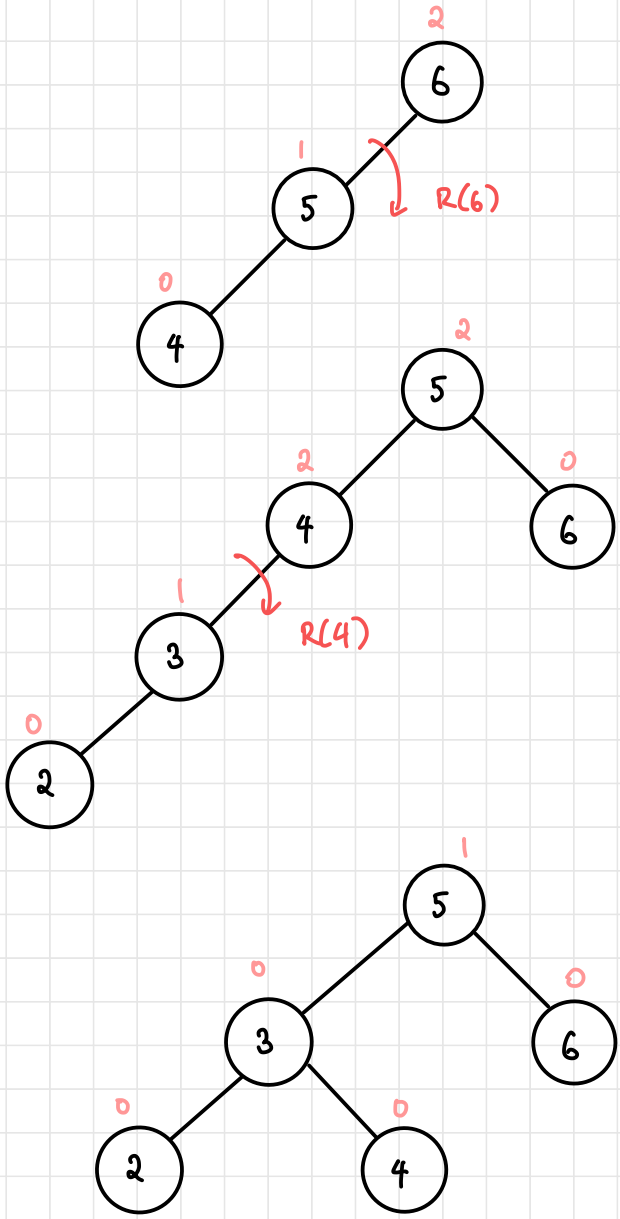


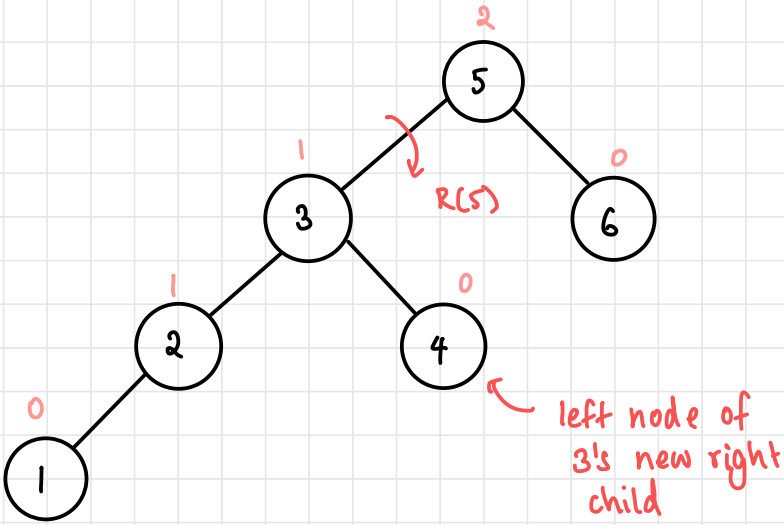


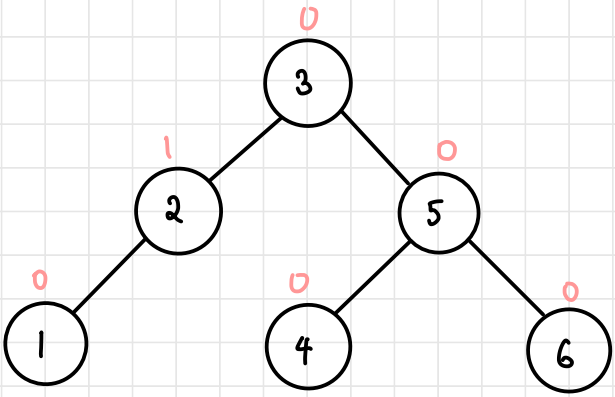
note: we only performed left rotation as elements were read in ascending order

Question 2

6, 5, 4, 3, 2, 1 — make AVL



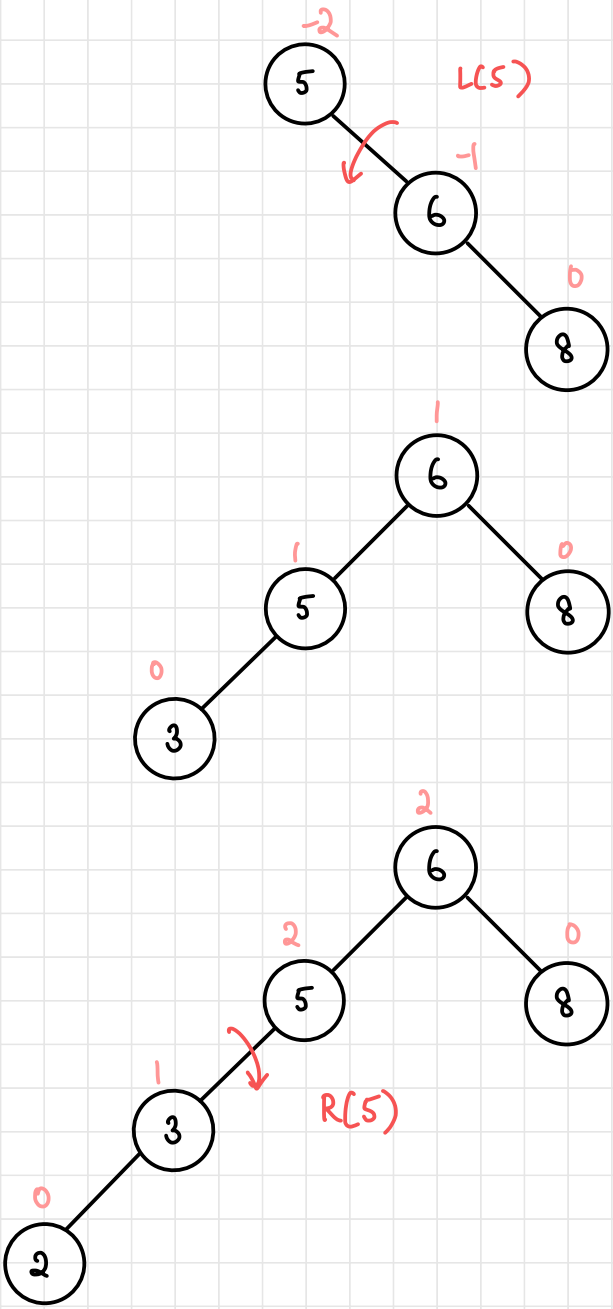


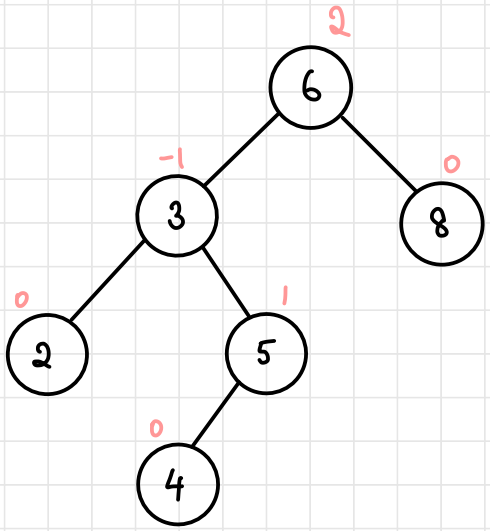


note: we only performed right rotation as elements were read in descending order

Question 3

5, 6, 8, 3, 2, 4, 7 — make AVL

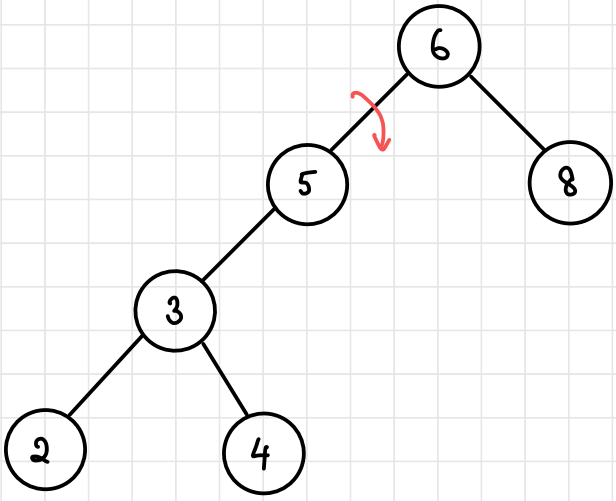


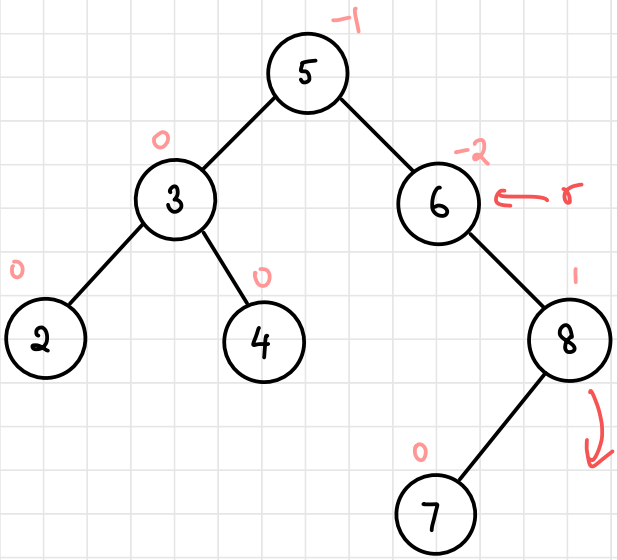


right subtree of left child

LR(3)

L(3) & R(6)

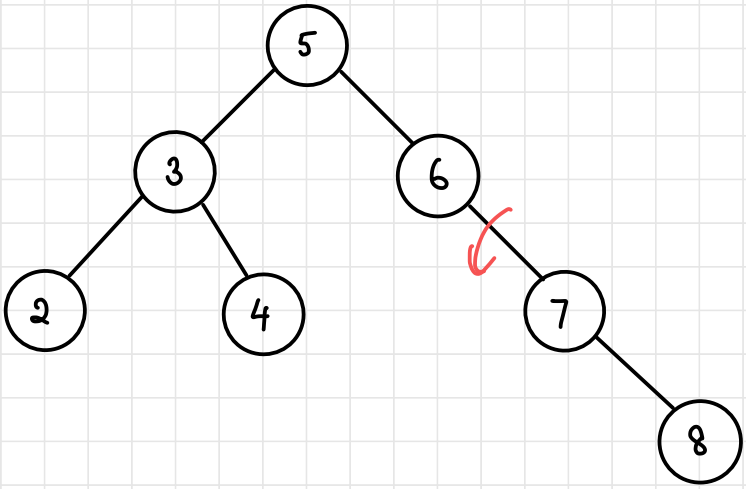


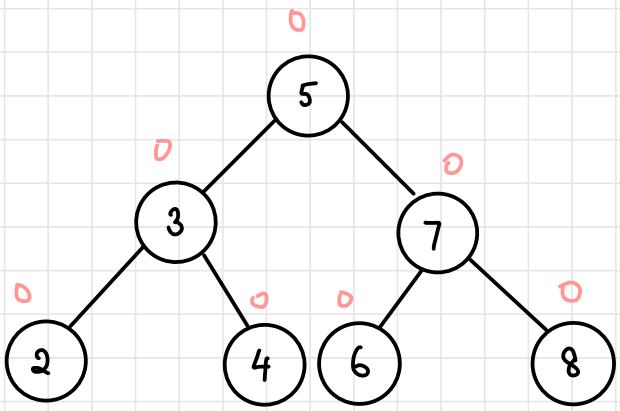


left subtree of right child

RL(6)

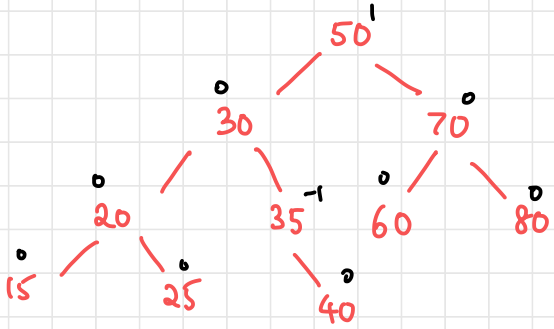
R(8) & L(6)





Question 4

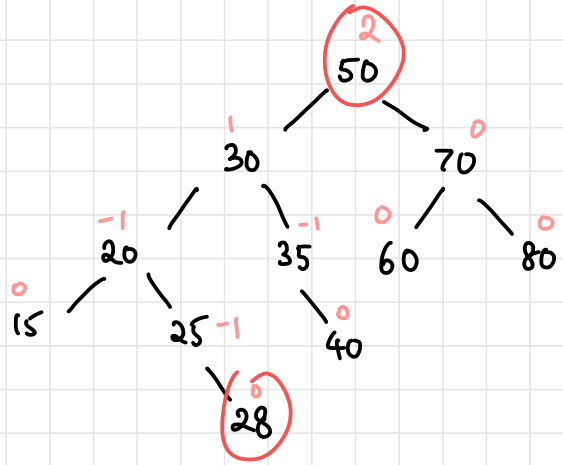
Is the following tree an AVL?



yes

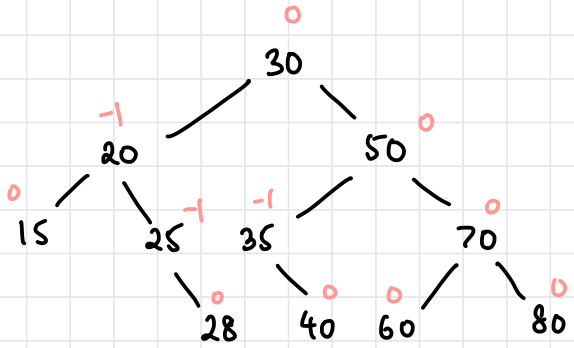
Question 5

Insert 28 into the above tree



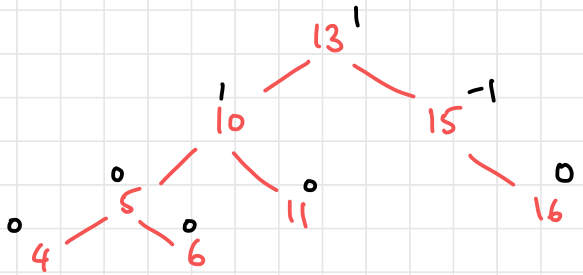
left subtree of left node

R(50)

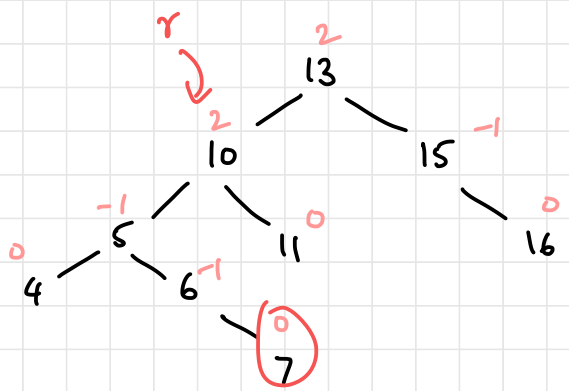


Question 6

Insert 7 into the AVL

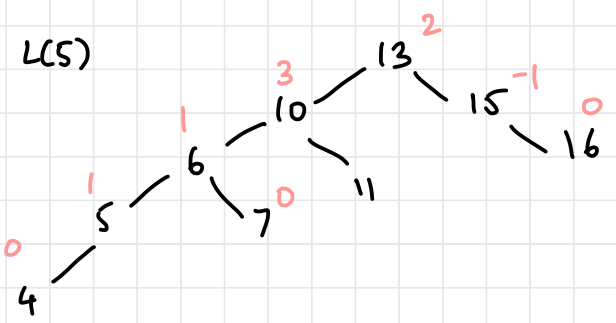


- it is an AVL
- insert 7

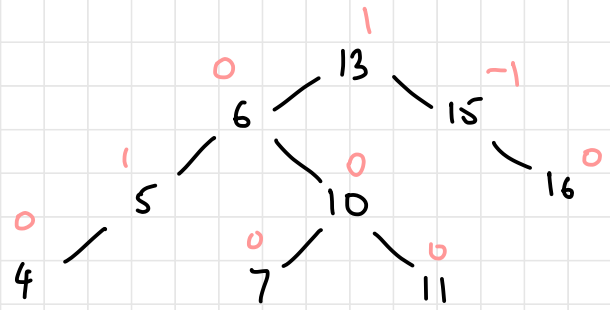


right subtree of left node

LR(10)



R(10)



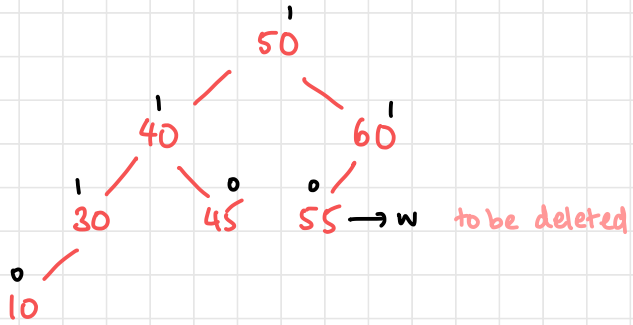
DELETING A NODE FROM AN AVL TREE

- Perform standard BST deletion of a node w
- Starting from w, travel up and find the first unbalanced node (z)
- Let y be the larger height child of z (or any for equal)
- Let x be the larger height child of y (any for equal)
- Rebalance the tree by performing appropriate rotations on subtree rooted at z
- Rotation depends on one of four possible cases wrt the alignment of x, y and z

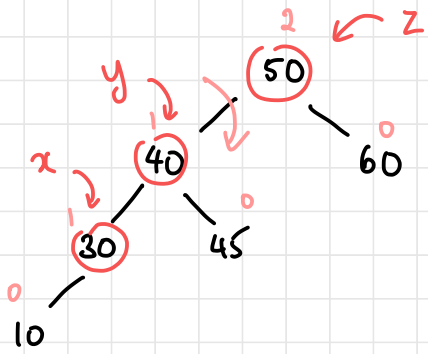
1. Left-left case: y left of z and x left of y : R(z)
2. Left-right case: y left of z and x right of y : LR(z)
3. Right-right case: y right of z and x right of y : L(z)
4. Right-left case: y right of z and x left of y : RL(z)

Question 7

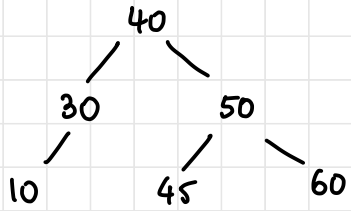
Delete 55 from the given AVL Tree



- Delete as you would from a regular BST
- Travel upwards and find first unbalanced node

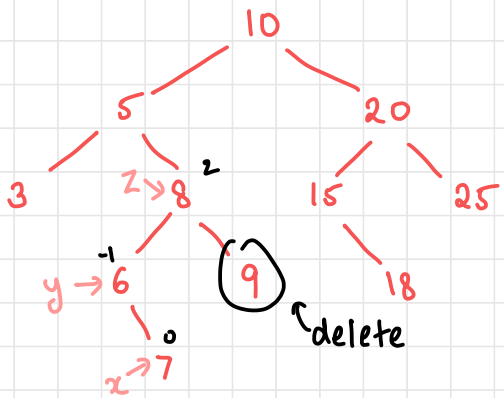


- y is left of z and x left of y: R(z)



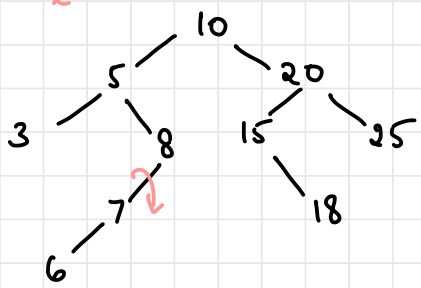
Question 8

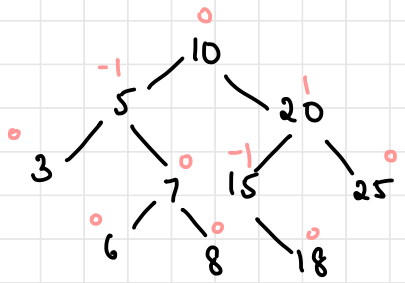
Delete 9 from the tree



y left of z
x right of y

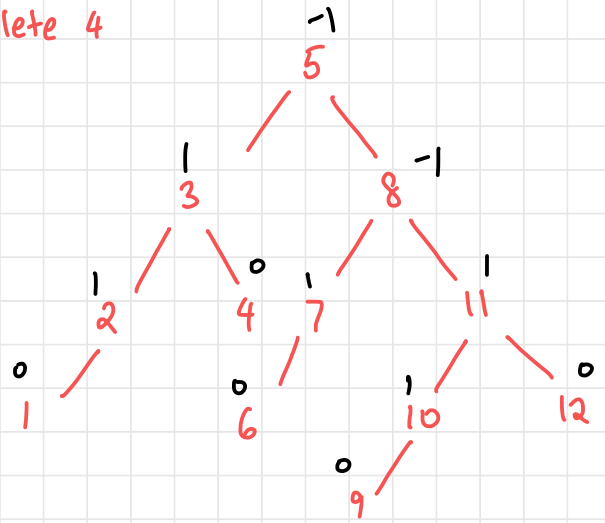
LR(8)

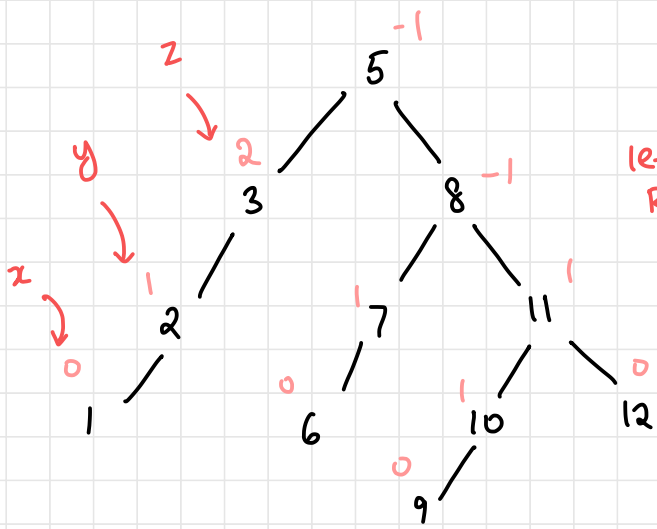




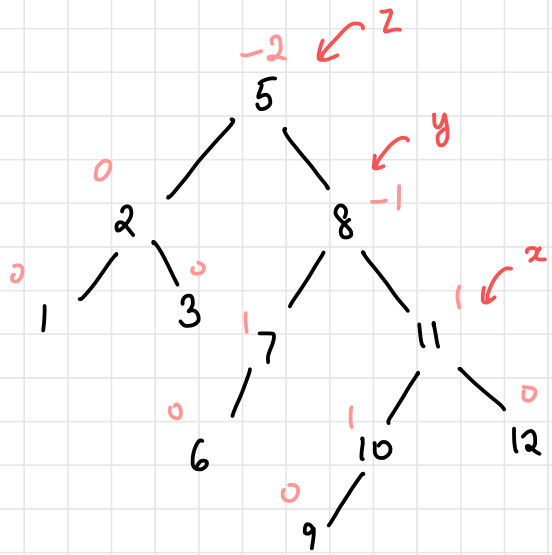
Question 9

Delete 4



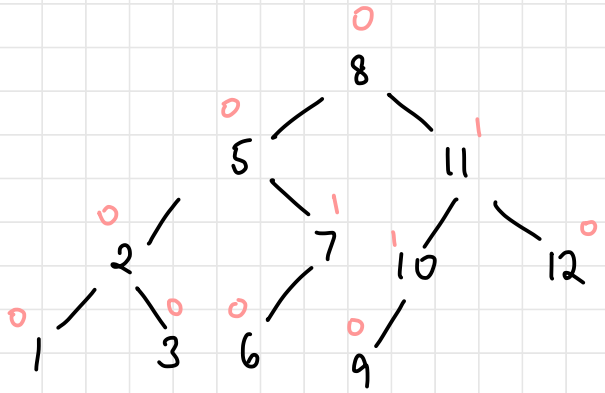


left-left:
R(3)



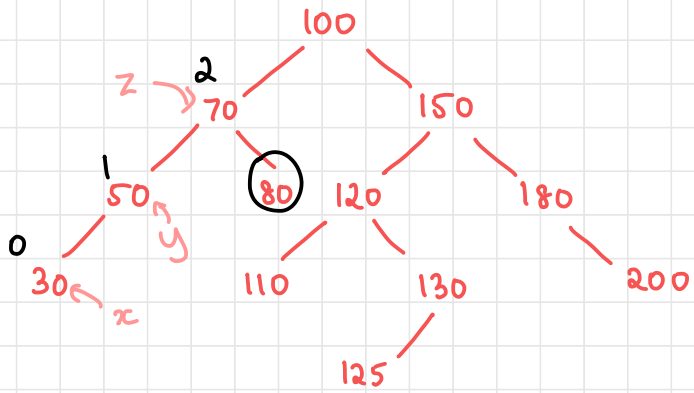
right-right

L(5)

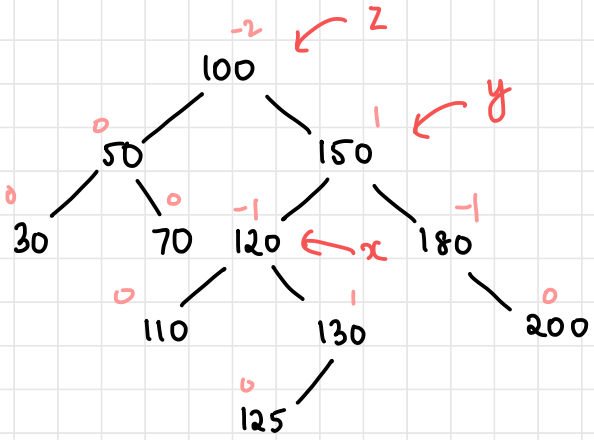


Question 10

Delete 80



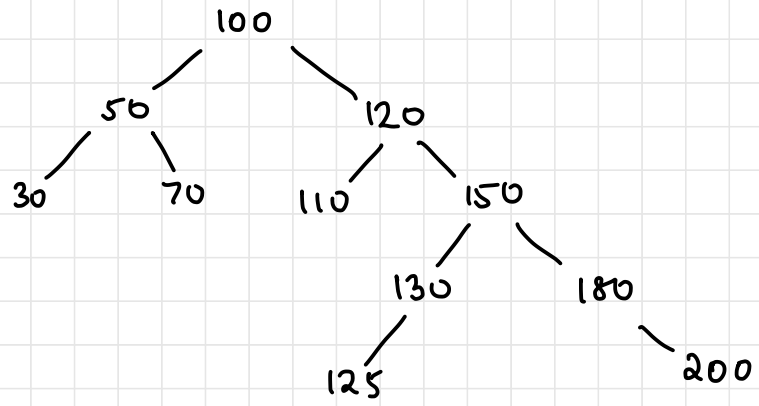
R(70) — left left case



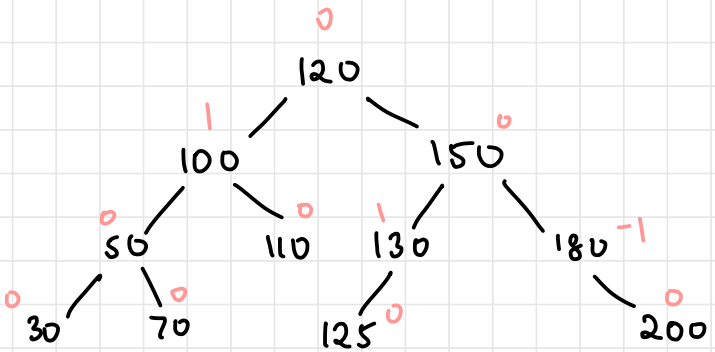
y right of z
x left of y

RL (100)

R(150)

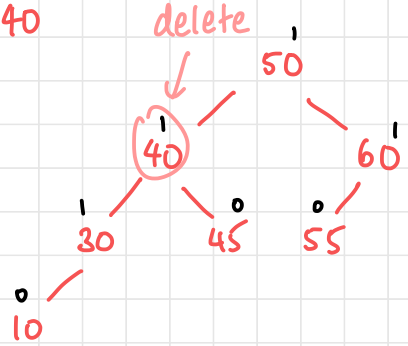


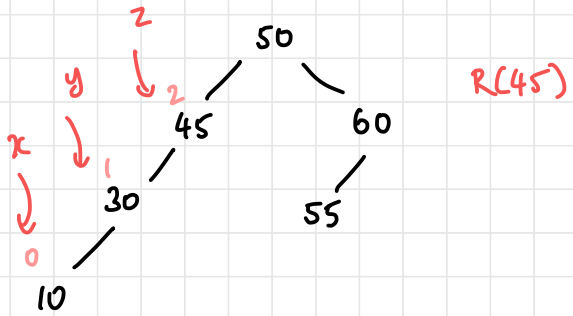
LC(100)

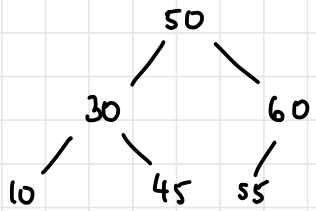


Question 11

Delete 40




z
y
x
2
1
0
50
45
60
30
55
10
R(45)


50
30
60
10
45
55