Page No.
Date / /

05/04/24 MON.

UNIT: NO: 4: (FOUR):-

~~OPTIMAL BINARY~~ SEARCH TREE's

* SYMBOL TABLE REPRESENTATION:-

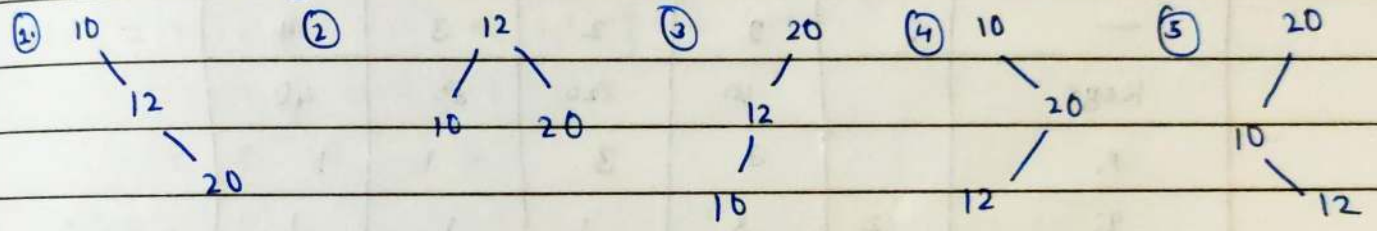
1. statically — Optimal Binary search Tree, Hashing.
2. Dynamically — AVL Tree.

Optimal Binary Search Tree. (OBS)

→ Eg: I/P

Keys [] = {10, 12, 20}, Freq[] = {34, 8, 50}

Following possible BST's:-



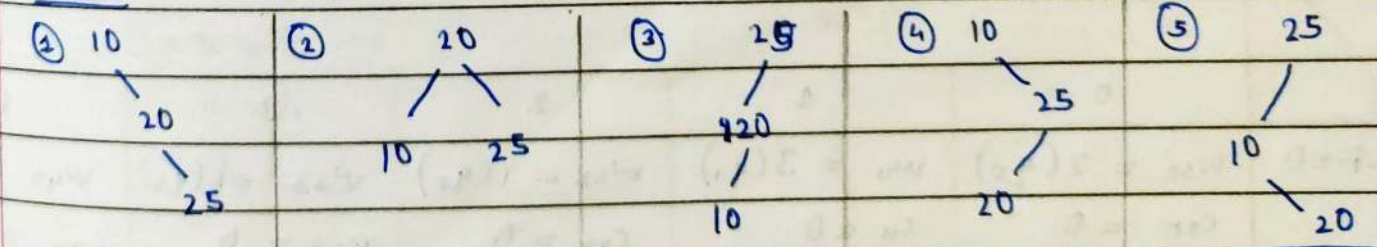
Among all the BST's, the cost of 5th BST is minimum

cost of 5th BST = (1×50) + (2×34) + (3×8)

= 50 + 68 + 24 = 142 (Minimum)

Q. Keys = {10, 20, 25}; Freq[] = {0.4, 0.5, 0.1}

Ans: Soln:-



Cost: ⇒ (1×0.4) + (2×0.5) + (3×0.1) = 0.4 + 1.0 + 0.3 = 1.7 ↓ (Minimum cost)

cost = (0.5×1) + (0.4×2) + (0.1×2) = ~~0.5~~ + 0.8 + 0.2 = ~~2.0~~ 1.5

cost = (0.1×1) + (0.5×2) + (0.4×3) = 0.1 + 1.0 + 1.2 = 2.3

cost = (0.4×1) + (2×0.1) + (3×0.5) = 0.4 + 0.2 + 1.5 = 2.1

cost = (0.1×1) + (0.4×2) + (0.5×3) = 0.1 + 0.8 + 1.5 = 2.4

Page No.
Date / /

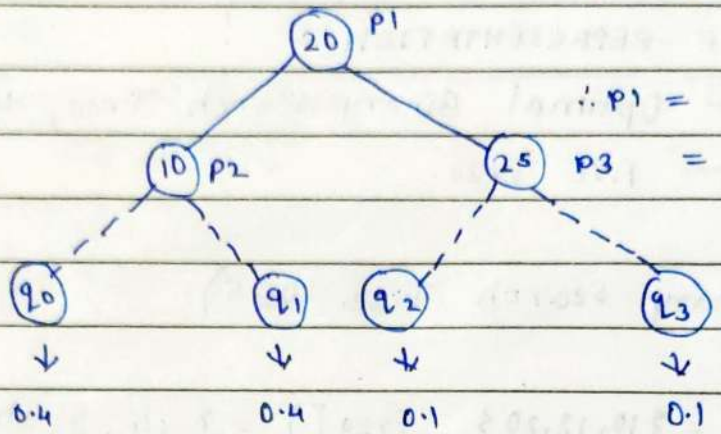
* Dynamic Programming:-

→ $P1 = (q_0 \times 2) + (q_1 \times 2) + (q_3 \times 2) + (q_2 \times 2)$

$\therefore P1 = 2 \times (0.4 + 0.4 + 0.1 + 0.1)$

$= 2 \times 1.0 = 2.0 = 2$

P1 = 2



—	0	1	2	3	4
Keys		10	20	30	40
P_i		3	3	1	1
q_i	2	3	1	1	1

* Weight Formula:-

$w(i,j) = w(i,j-1) + P_j + q_i$

* Cost formula:-

$c(i,j) = \min_{i \le k \le j} \{ c(i,k-1) + c(k,j) \} + w(i,j)$

	0	1	2	3	4
$j-i=0$	$w_{00} = 2(q_0)$	$w_{11} = 3(q_1)$	$w_{22} = 1(q_2)$	$w_{33} = 1(q_3)$	$w_{44} = 1(q_4)$
	$c_{00} = 0$	$c_{11} = 0$	$c_{22} = 0$	$c_{33} = 0$	$c_{44} = 0$
	$r_{00} = 0$	$r_{11} = 0$	$r_{22} = 0$	$r_{33} = 0$	$r_{44} = 0$
$j-i=1$	$w_{01} = 2+3+3 = 8$	$w_{12} = 3+3+1 = 7$	$w_{23} = 1+1+1 = 3$	$w_{34} = 1+1+1 = 3$	
	$c_{01} = 8$	$c_{12} = 7$	$c_{23} = 3$	$c_{34} = 3$	
	$r_{01} = 1$	$r_{12} = 2$	$r_{23} = 3$	$r_{34} = 4$	
$j-i=2$	$w_{02} = 8+3+1 = 12$	$w_{13} = 7+1+1 = 9$	$w_{24} = 3+1+1 = 5$		
	$c_{02} = 19$	$c_{13} = 12$	$c_{24} = 8$		
	$r_{02} = 1$	$r_{13} = 2$	$r_{24} =$		

Page No.
Date / /

	0	1	2	3	4
j-i=3	$W_{03} = 12+1+1 = 14$	$W_{13} = 9+1+1 = 11$			
	$C_{03} = 22$	$C_{13} = 19$			
	$r_{03} = 2$	$r_{13} = 2$			
j-i=4	$W_{04} = 14+1+1 = 16$				
	$C_{04} = 32$				
	$r_{04} = 2$				

(1) 1) 1 possibility $\therefore k = 1 \quad (0 < k \leq 1)$

$C_{01} = \min\{c(0,0) + c(1,1)\} + w(0,1)$

$= \min\{0 + 0\} + 8$

$= 0 + 8 = 8$

2) $1 < k \leq 2$

$k = 2$

$C_{12} = \min\{c(1,1) + c(2,2)\} + w(1,2)$

$= 7$

3) $2 < k \leq 3$

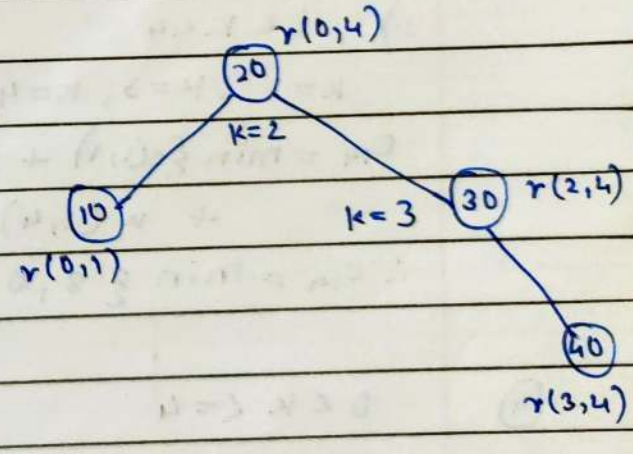
$k = 3$

$C_{23} = 3$

4) $3 < k \leq 4$

$k = 4$

$C_{34} = 3$



(2) 1) $0 < k \leq 2$

$k = 1 \; ; \; k = 2$

$C_{02} = \min \begin{Bmatrix} c(0,0) + c(1,2) \\ c(0,1) + c(2,2) \end{Bmatrix} + w(0,2)$

$= \min\{7, 8\} + 12$

$= 7 + 12 = \underline{19}$

Page No.
Date / /

2.) $1 < k \leq 3$

$k = 2 \; ; \; k = 3$

$c_{13} = \min \{ c(1,1) + c(2,3), \; c(1,2) + c(3,3) \} + w(1,3)$

$= \min \{ 3, 7 \} + 9$

$= 3 + 9 = 12$

3.) $2 < k \leq 4$

$k = 3, \; k = 4$

$\therefore c_{24} = \min \{ c(2,2) + c(3,4), \; c(2,3) + c(4,4) \} + w(2,4)$

$\therefore c_{24} = \min \{ 3, 3 \} + 5$

$\therefore c_{24} = 3 + 5 = 8$

(3) 1.) $0 < k < = 3$

$k = 1, \; k = 2, \; k = 3$

$c_{03} = \min \{ c(0,0) + c(1,3), \; c(0,1) + c(2,3), \; c(0,2) + c(3,3) \}$

$+ w(0,3)$

$c_{03} = \min \{ 12, 11, 19 \} + 14 = 11 + 14 = 25$

2.) $1 < k < 4$

$k = 2, \; k = 3, \; k = 4$

$c_{14} = \min \{ c(1,1) + c(2,4), \; c(1,2) + c(3,4), \; c(1,3) + c(4,4) \}$

$+ w(1,4)$

$\therefore c_{14} = \min \{ 8, 10, 12 \} + 11 = 8 + 11 = 19$

(4) $0 < k < = 4$

$k = 1, \; k = 2, \; k = 3, \; k = 4$

$\therefore c_{04} = \min \{ c(0,0) + c(1,4), \; c(0,1) + c(2,4), \; c(0,2) + c(3,4),$

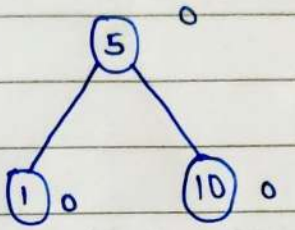
$c(0,3) + c(4,4) \} + w(0,4)$

$\therefore c_{04} = \min \{ 19, 16, 22, 22 \} + 16$

$c_{04} = 16 + 16 = 32$

AVL TREE:-

- It is a Binary Search Tree. (BST)
- Height of the tree (left and Right subtree) do not differ by more than 1 (one).
- **Balance Factor = Height of left subtree − Height of Right subtree.**
- AVL Tree has a balance factor of 1, -1 and 0.
  = = =
- For Eg:- 1.)



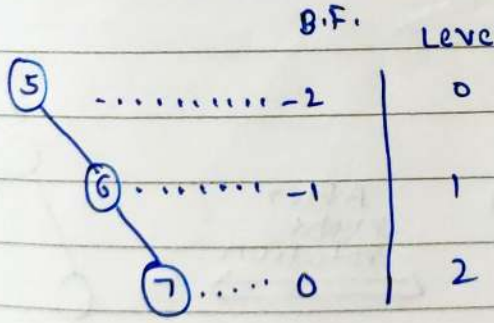
Balance Factor of node (1) (B.F.) = Height of left subtree − Height of Right subtree

= 0 − 0 = 0.

∴ (B.F.) of node (10) = 0 − 0 = 0

B.F. of node (5) = 1 − 1 = 0.

2.)



B.F. of node 6 = 0 − 1 = −1

B.F of node 5 = 0 − 2 = −2 .... (Not a B.F.) unacceptable.

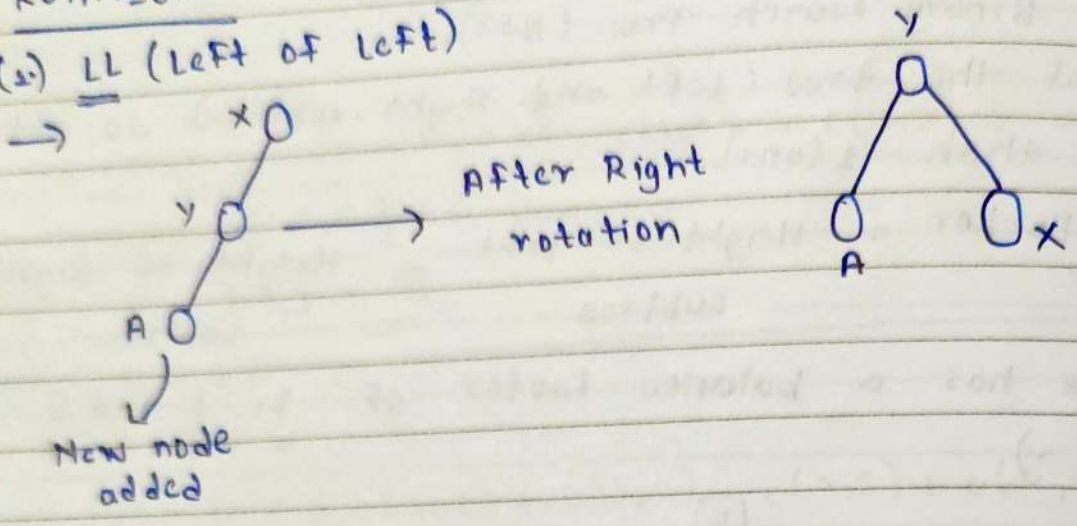
B.F. of node 7 = 0 − 0 = 0.

Thus, Given tree is not a AVL Tree.

ROTATION:-

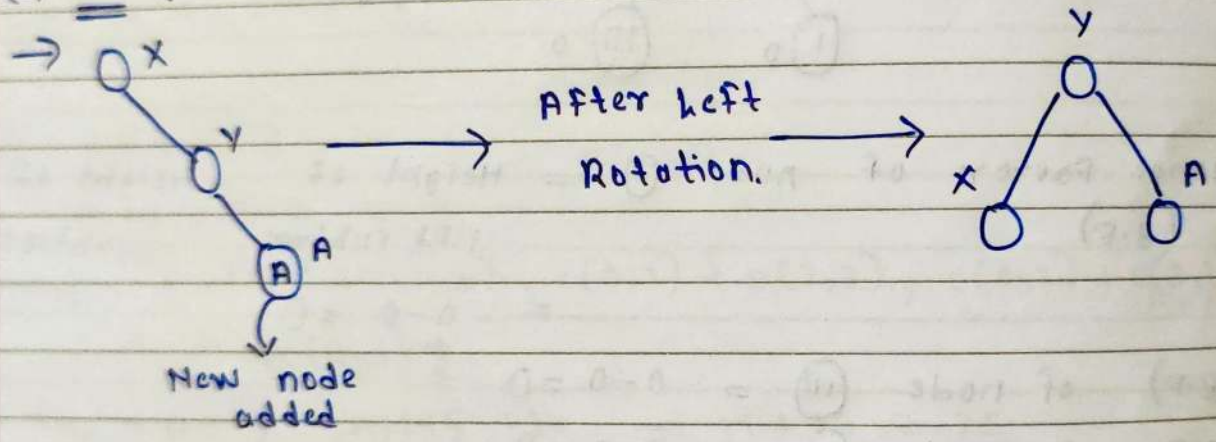
(1.) LL (Left of Left)

→



(2.) RR (Right of Right)

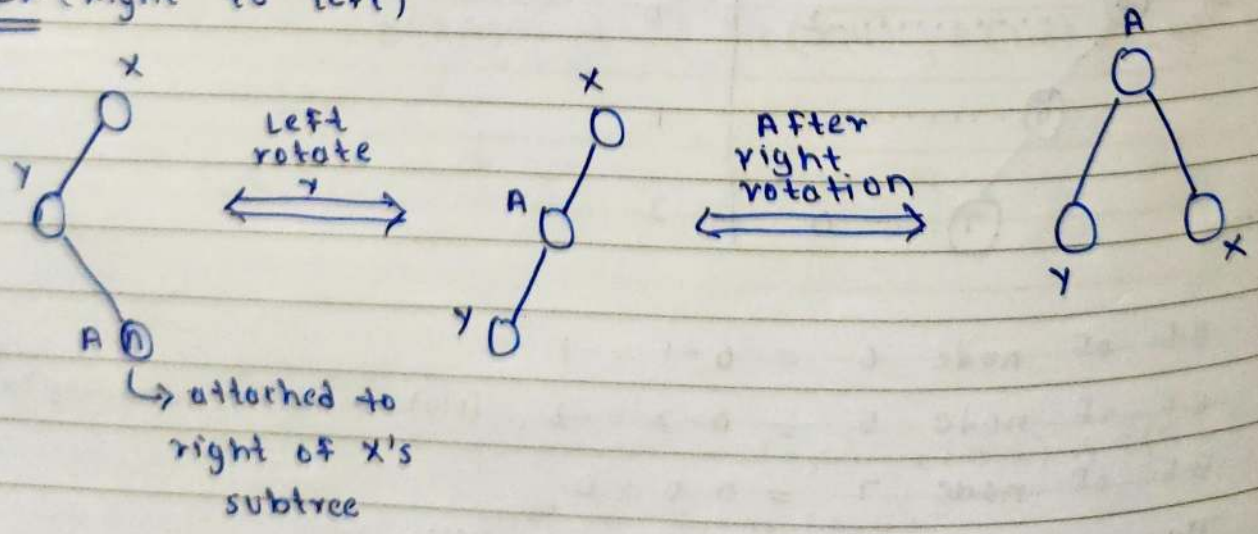
→

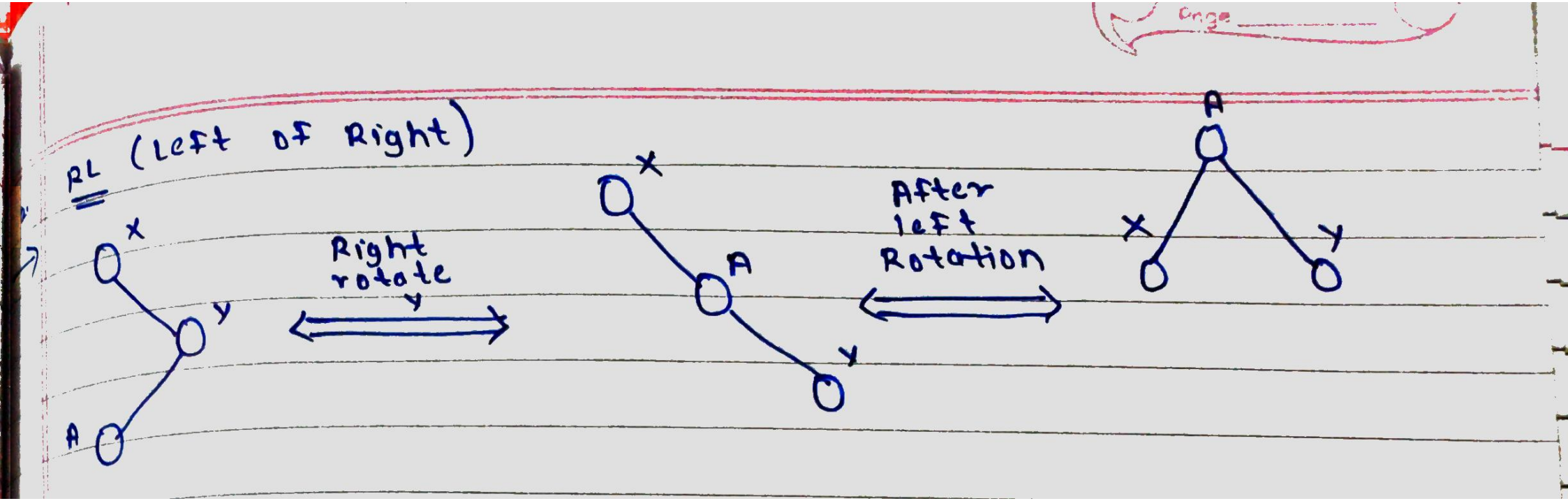


DOUBLE ROTATION:-

1. LR (Right to left)

→




RL (Left of Right)
x
y
A
Right rotate y
x
A
y
After left Rotation
A
x
y

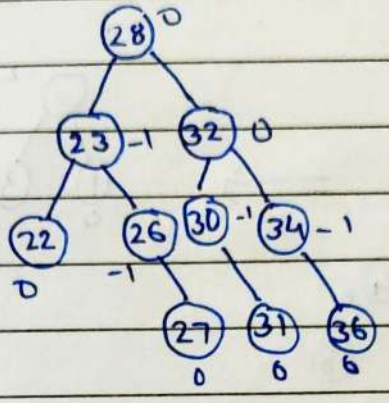
Page No.
Date / /

A.W

Q) Insert Following keys into the AVL Tree.
1, 2, 3, 4, 8, 7, 6, 5, 11, 10, 12.

→

Deletion in the AVL Tree:-

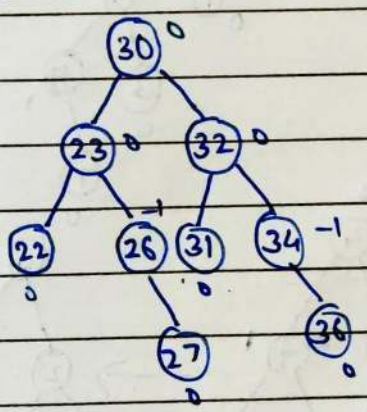


InOrder : 22, 23, 26, 27, 28, 30, 31, 32, 34, 36

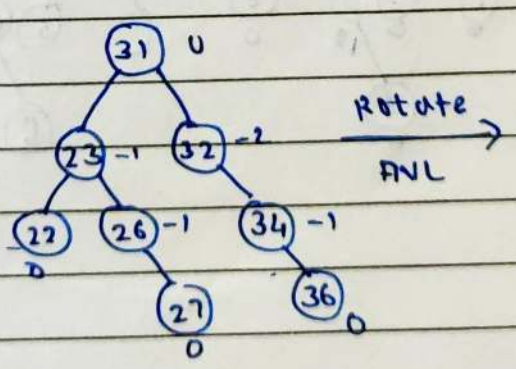
→

Now, Delete '28'

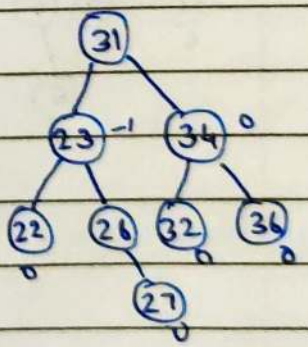
'28' is a node with two child, replace it with its inorder successor.



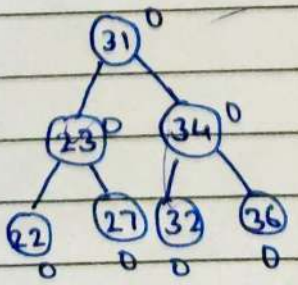
Delete '34'.-



Rotate AVL

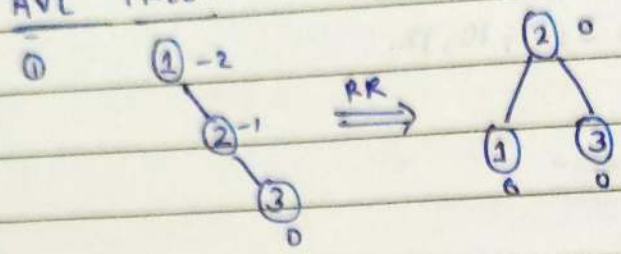


Delete 26:

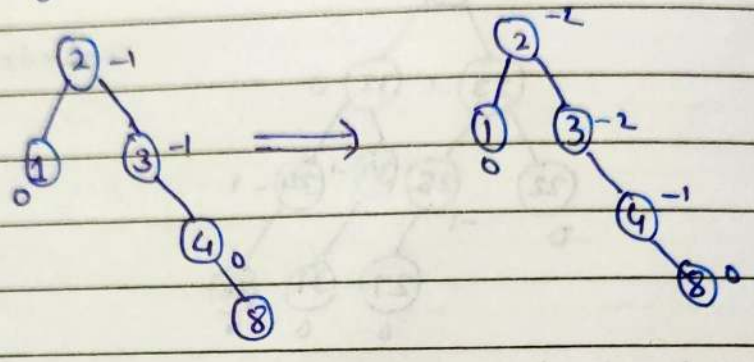


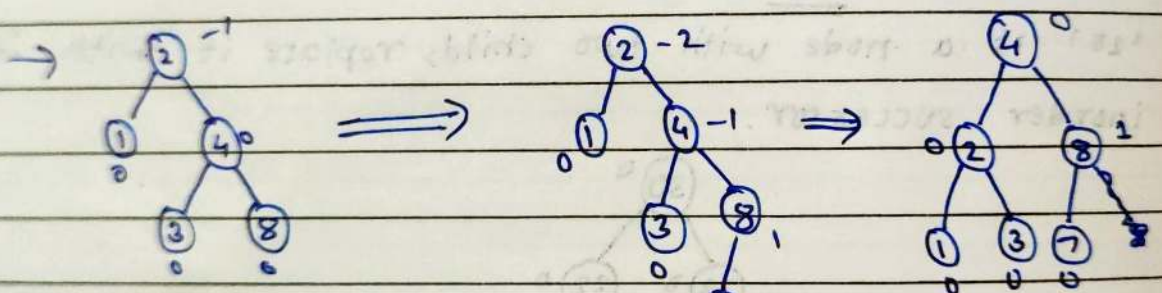
Page No.
Date / /

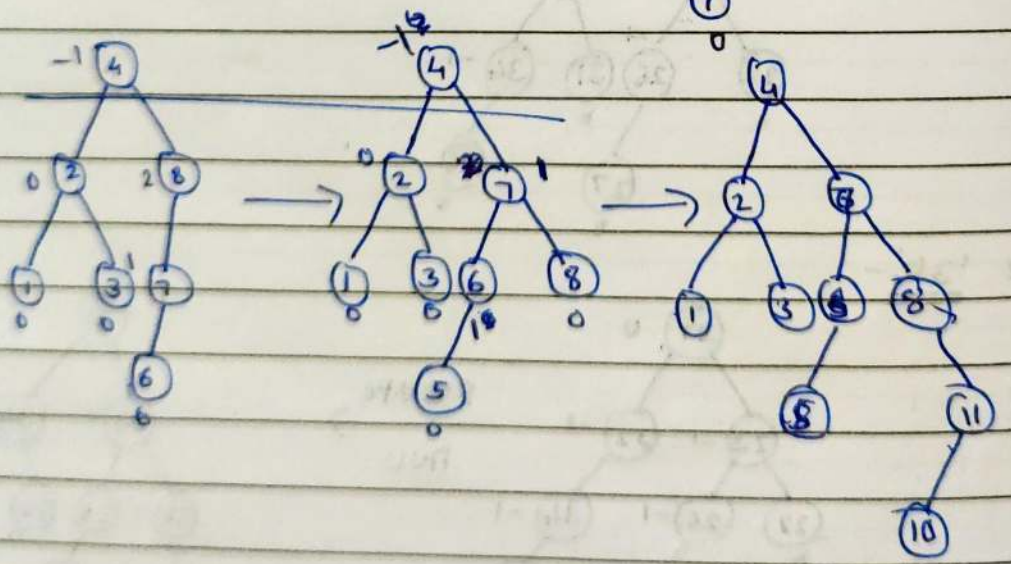
AVL TREE:- 1, 2, 3, 4, 8, 7, 6, 5, 11, 10, 12.



Insert 4 :





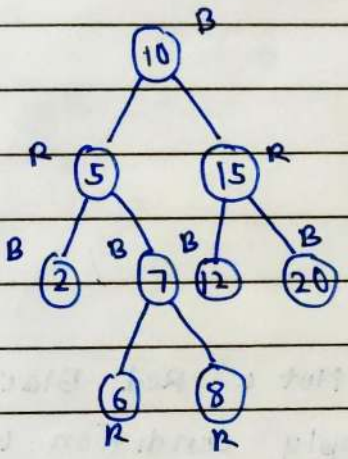


Page No.
Date / /

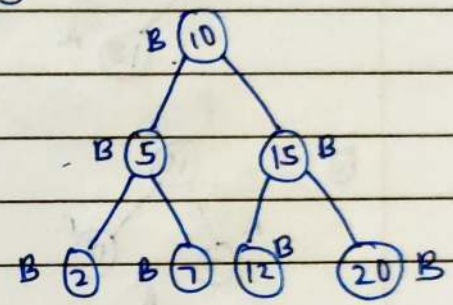
RED BLACK TREE:- (Properties)

1) It is a self balancing Tree (but not strict)
2) Roughly height balance Tree.
3) Every node is either Black or Red.
4) Root is always Black.
5) If node is red, then children are black. (No Adjacent red nodes) (May be Black)
6) Every path from a node from any of its descendant ~~has~~ till leaf node has same number of black nodes.

→ Eg ①:-



Eg ②:-



- The above tree is a Red-Black Tree.

Creation of Red Black Tree:-

→ Algorithm:-

STEP I: If tree is empty, create new node with black colour

STEP II:- If tree is not empty, create new node as leaf node with red colour.

STEP III:- If parent of new node is black, then exit.

STEP IV:- If parent of new node is Red, then check the colour of parent sibling of new node

Condition: ① If the colour of parent sibling is black or null, do suitable rotation. and recolour. (parent node)

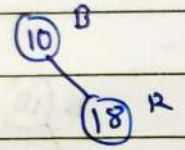
Page No.
Date / /

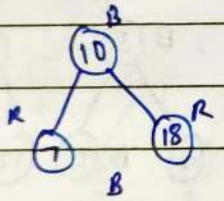
② If the colour of the parent sibling is Red, recolour both parent and sibling and also check if parent's parent of new node is not new node, then recolour and recheck all the condition.

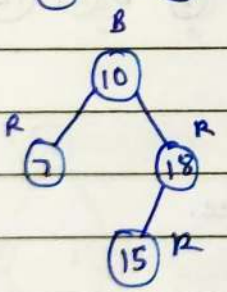
Eg: Create a Red Black tree:-

→ 10 18 7 15 16 30 25

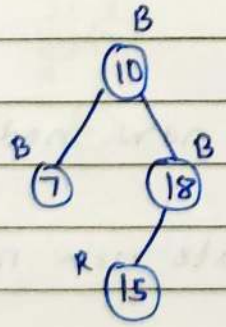
Ans Soln:-

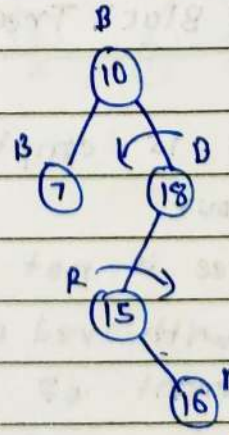
①


②


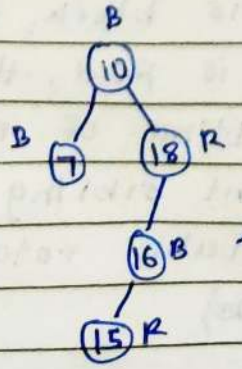
③


→ Not a Red Black tree
Apply condition (2), ∴ recolour (7) and (18)

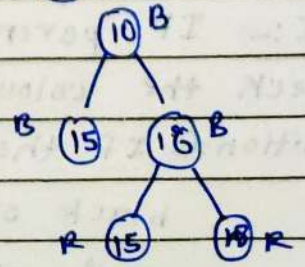
④


⑤


→ Not a red-black tree
∴ Apply condition (1), rotation and recolour ~~15~~ parent and parent's parent.

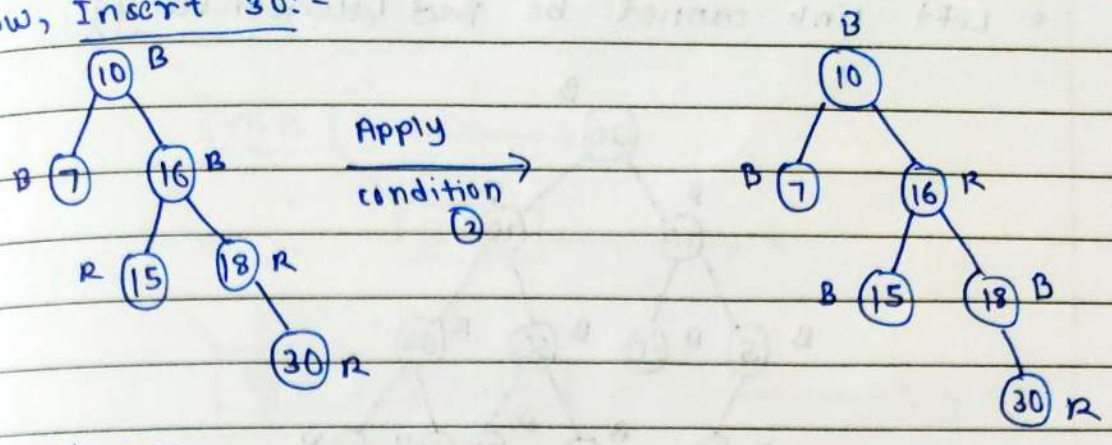
⑥


recolour and rotate. →



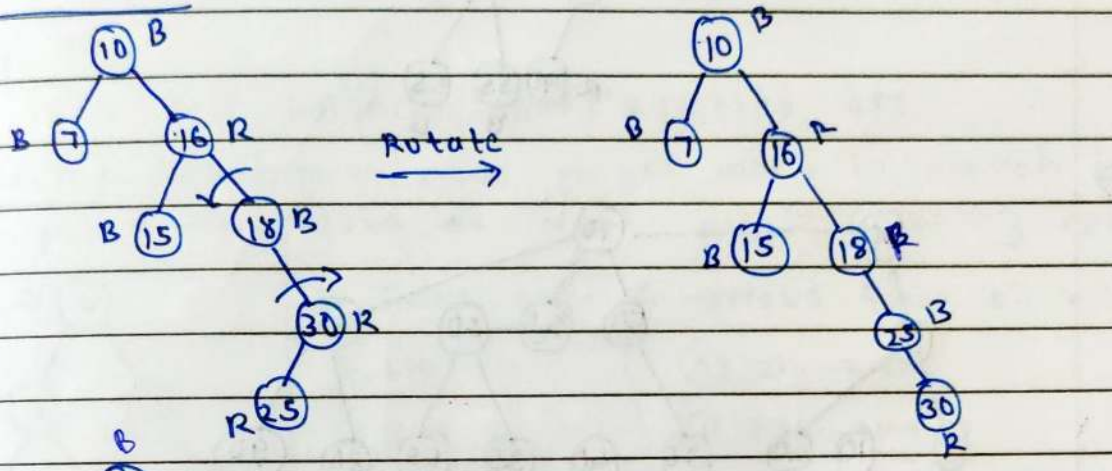
Page No.
Date / /

⑦ Now, Insert 30:-

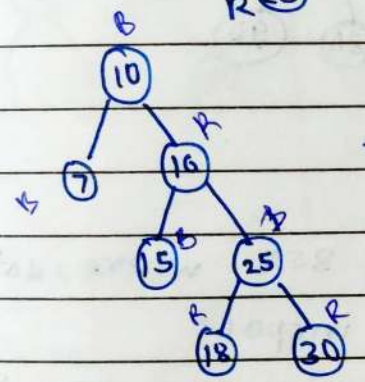


⑧ Insert 25:-

→



⑨



→ This is a Red Black Tree.

AA Tree:-

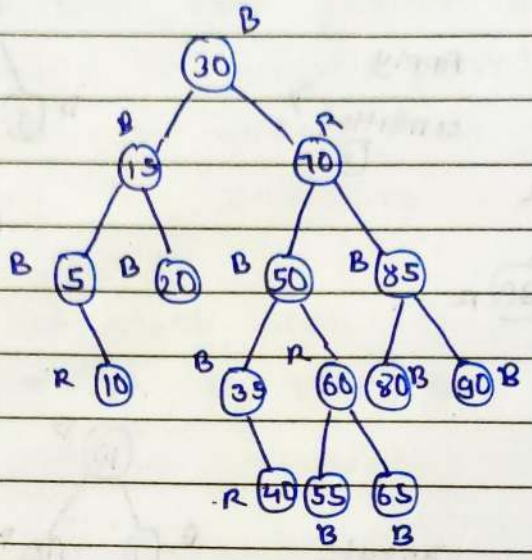
→ ◦ AA Trees are the variations of Red Black Tree.
- By default BST.
- Root should be Black.
- Level of Node is used for Balancing information all the leaf nodes are at same level.

RULE:-

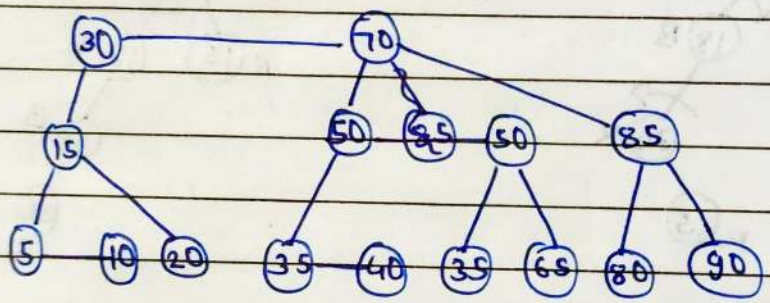
◦ Every ~~Red~~ Node can be Red or Black and Red child is horizontally linked to Black.
◦ No Adjacent Red Nodes.
◦ same Number of Black nodes in each path.

Page No.
Date / /

- Left link cannot be Red (Horizontally)



→



K- Dimensional Trees:-

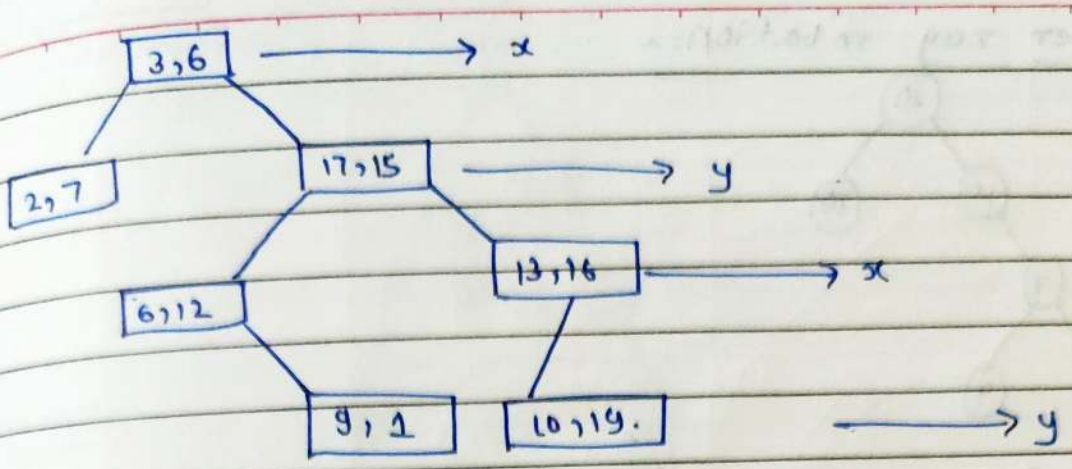
→ • K-D Tree is an extention of BST where data in each node is a K-D. point inspace

Pro's
- In one node, we can store more than one information.
- Efficient searching due to height balancing
- Easy to implement.

- Create a 2D Tree

→ (3,6), (17,15), (13,16), (6,12), (9,1), (2,7), (10,19)

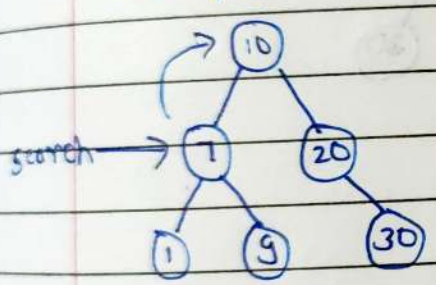
Page No.
Date / /



Splay Tree:-

→ It is a self balancing BST / Adjusted BST.

- search - whatever node ~~is~~ you want to search it will be print the data to root position using splaying



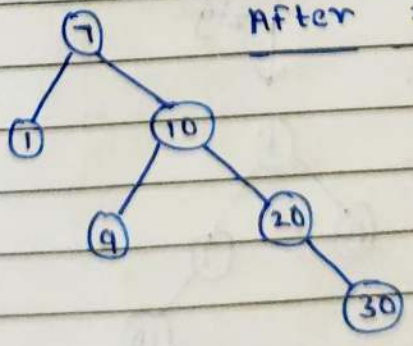
□ There are 4 various types of rotation

① zig	⑤ zig-zig
② zag	⑥ zag-zag.
③ zig-zag	
④ zag-zig	

1st rotation is called the zig rotation
↳ (right rotation)

case 1 : zig rotation

search (9)



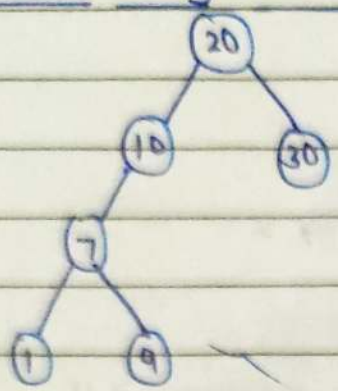
After zig rotation.

case 2 : zag rotation

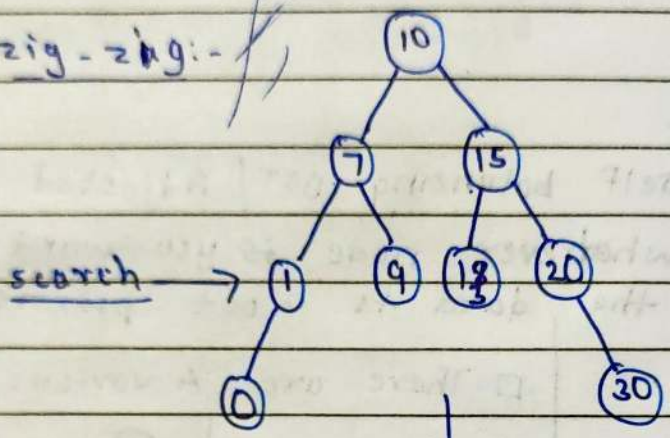
→ ~~rit~~ (left rotation)

Page No.
Date / /

After zag rotation:-

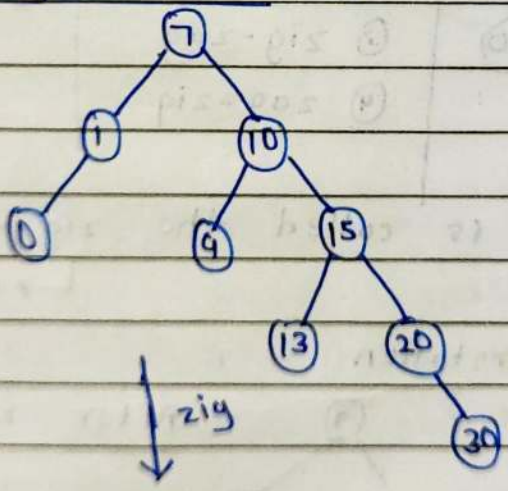


case 3: zig-zag:-

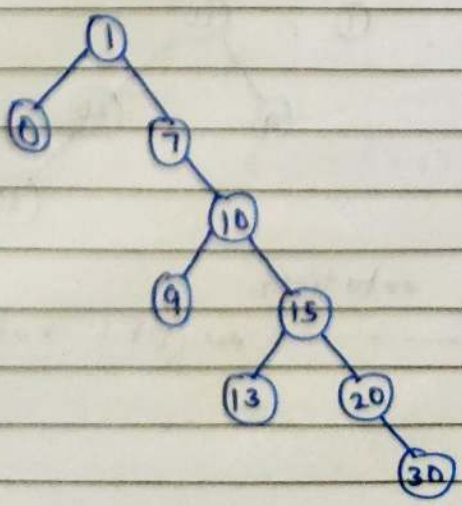


zig

• After zig-zag rotation.

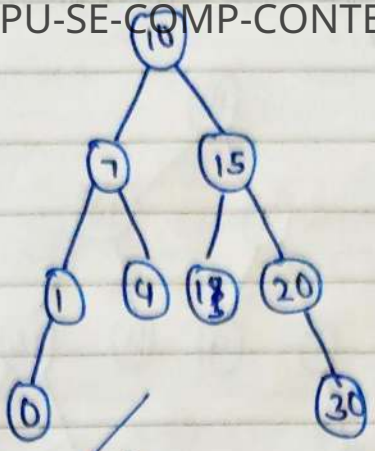


zig



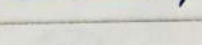
Case 4:- zag - zag rotation

→

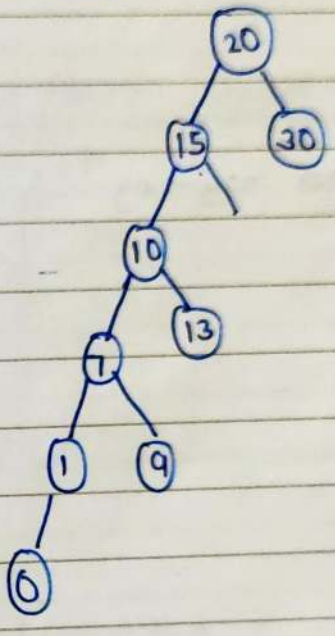


Search 20:

After zag

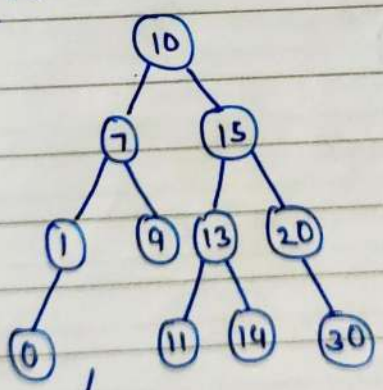


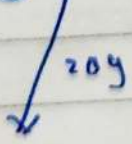
After zag-zag :-



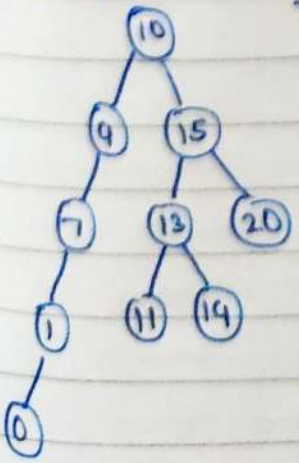
Case 5: zag-zig:-rotation

→

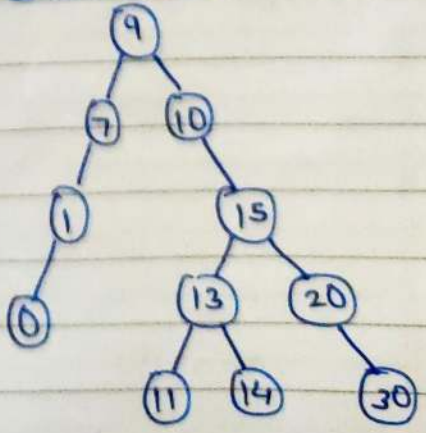




After zag:-



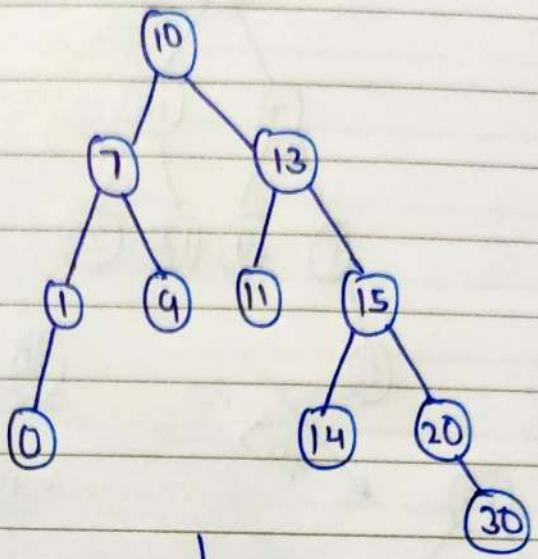
After zig-zag :-



CASE:- 6. zig-zag rotation

search (13)

After zig:-



zag

After zig-zag