Dynamic Graph

- Splay tree
- Link-cut tree.

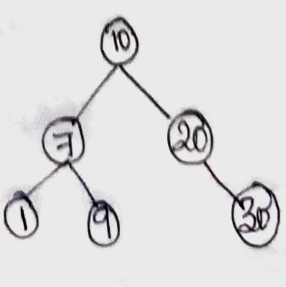
Splay tree ⇓

It is actually same as BST; only thing is that the search, insert & delete should be followed by a 'splay' operation.

In 'splay' operation, which ever node we are trying to perform the operation on (search, dlt,...) becomes the next root / near to the root.

- Splay
  - →Zig
    - Left Right
    - Only 1 step rotation is required.
  - →Zig Zig
    - Left Right
    - 2 steps rotation required
  - →Zig Zag.
    - Mixed.
    - 2 steps but on diff. sides.

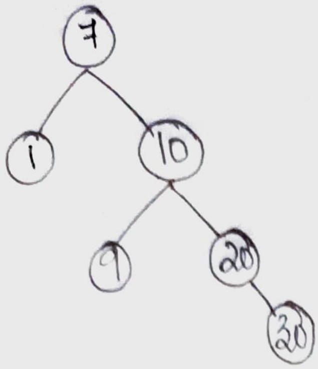
Q.



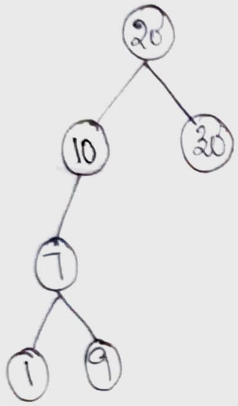
w.r.t. the above original graph:-

- Search 7

"Zig Right"



- Search 20



"Zig left"

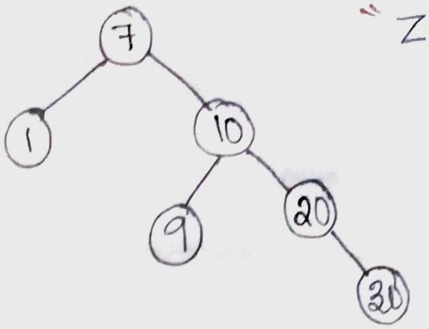
- Search 1.

w.r.t. the original graph, '1' is ~~one~~ on the same side of parent & grandparent; if on "same side" then "same type rotation" (i.e. ~~z~~ zig zig)

- If parent & grandparent on the same side of target node then rotation should start from the grandparent.

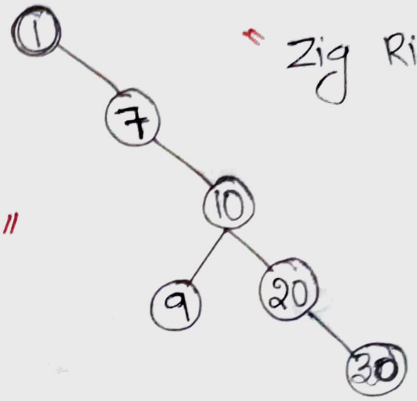
Step 1:-

"Zig Right 1"



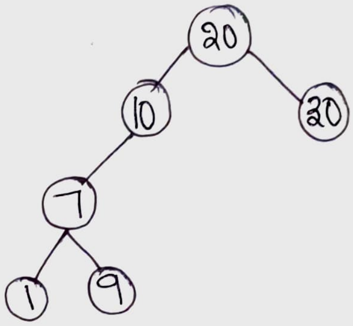
Step 2:-

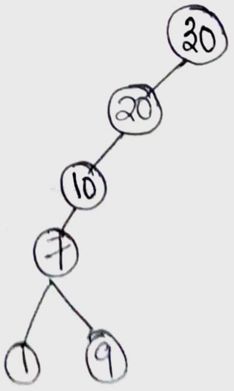
"Zig Right 2"



"Zig Zig Right"

- Search 30.



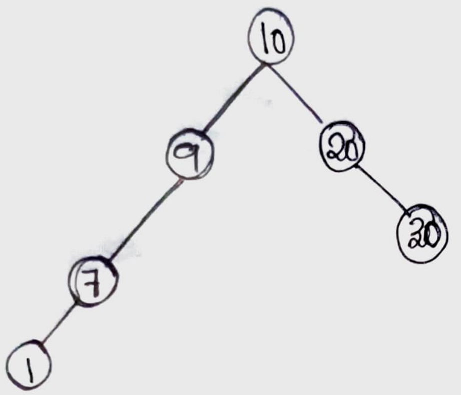


"Zig Zig Left"

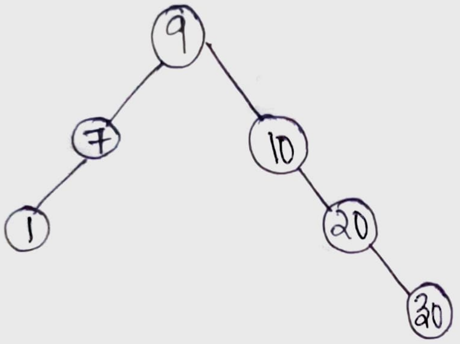
- Search 9.

⊛ parent & grandparent on diff sides;
∴ zig zag rotation.

➔ Since on diff. sides, 1st rotation is on parent.



"Zig Zag"

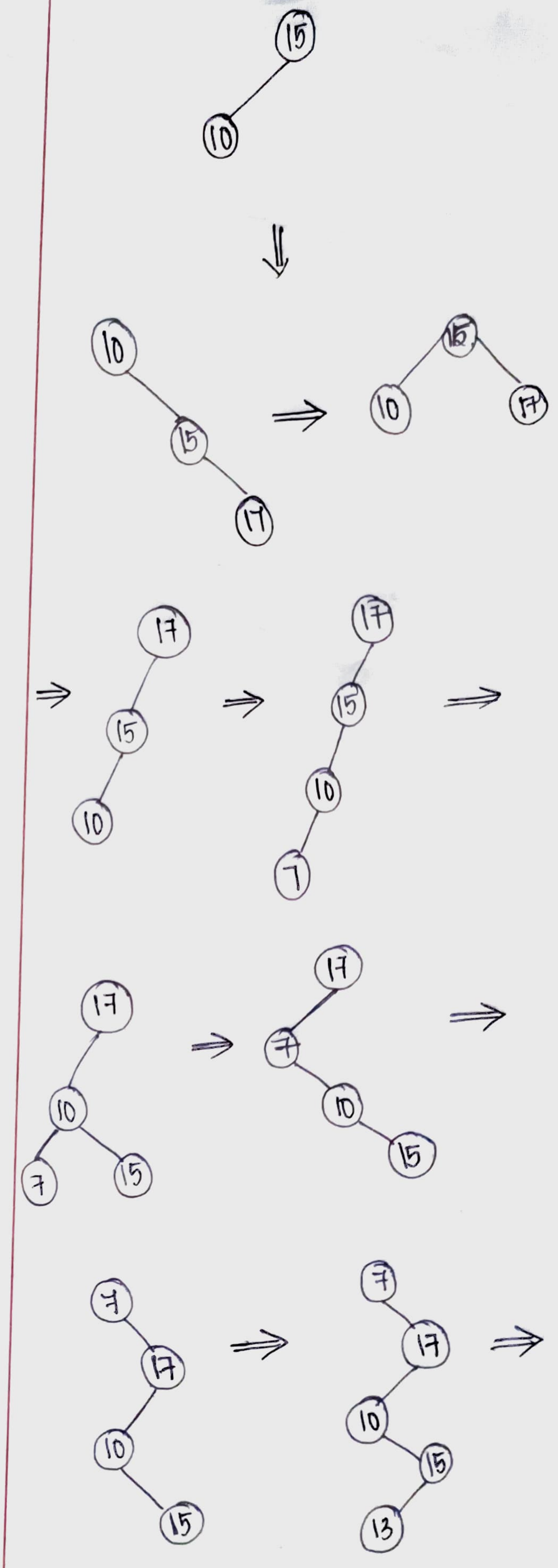


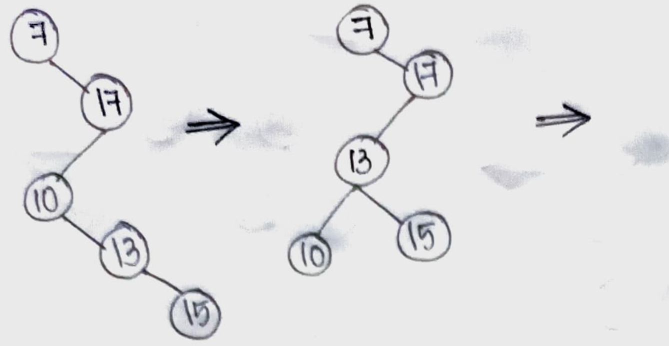
* Time complexity of splay tree → O(n)

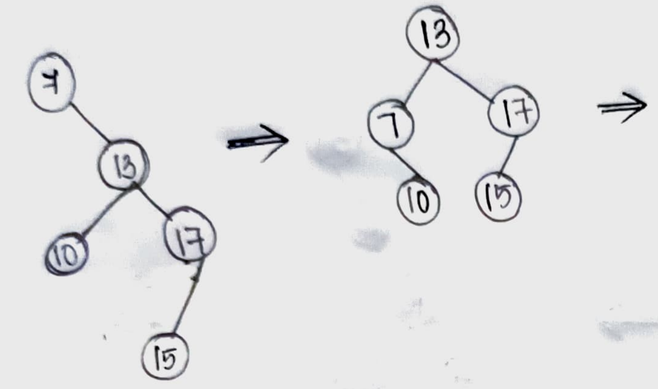
* Application of splay tree → Cache memory

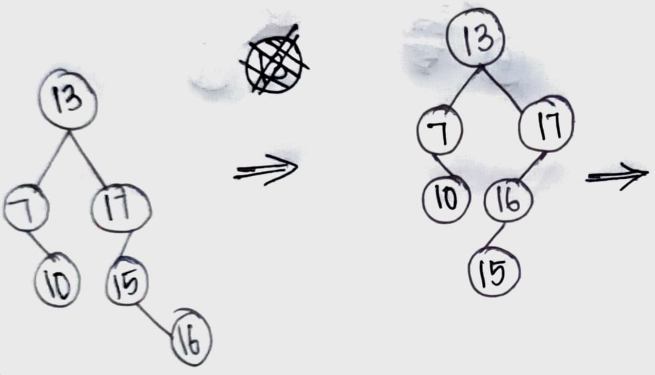
* Insertion in splay tree.

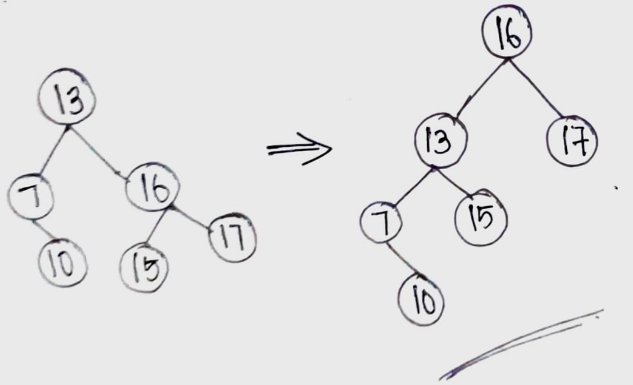
Q: 15, 10, 17, 7, 13, 16 ↓
Insert these elements into splay tree.







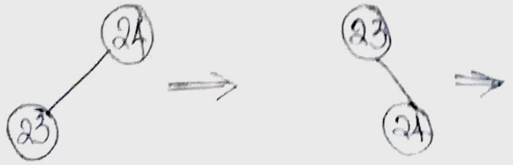


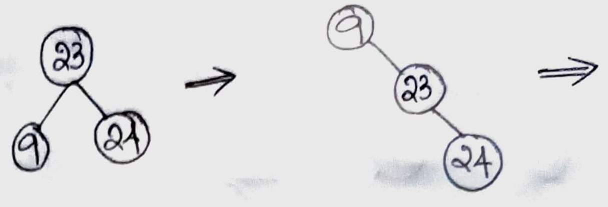


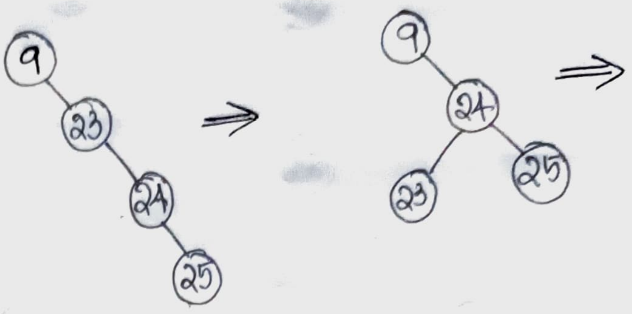
Q.

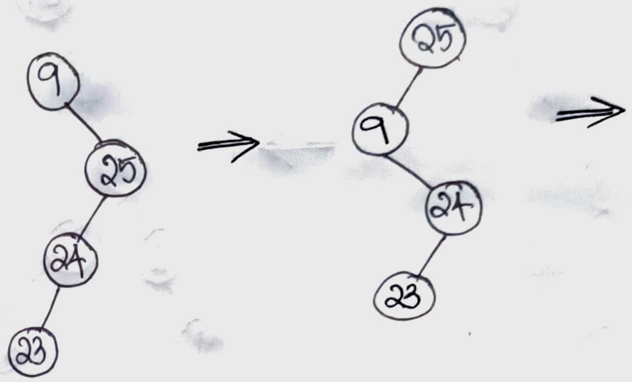
Insert :-

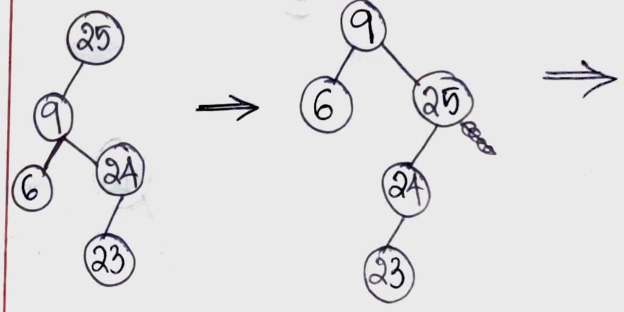
24, 23, 9, 25, 6, 12, 37, 10.

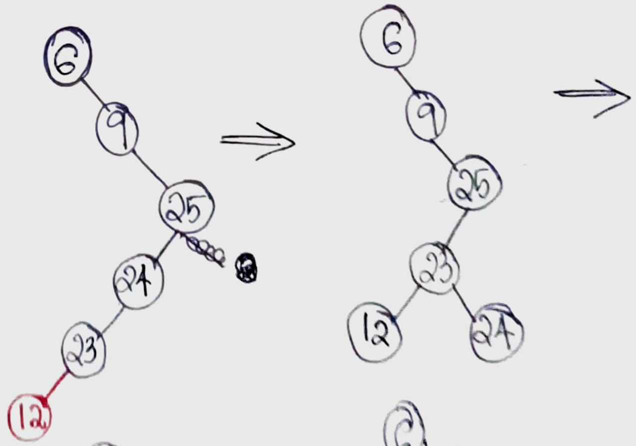


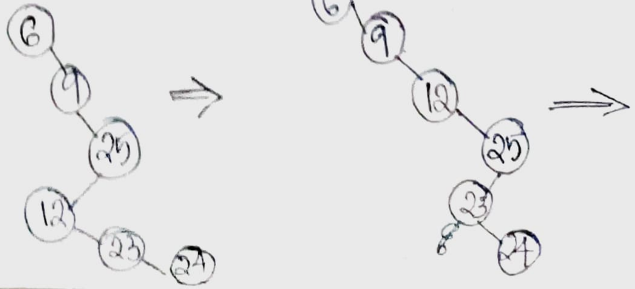


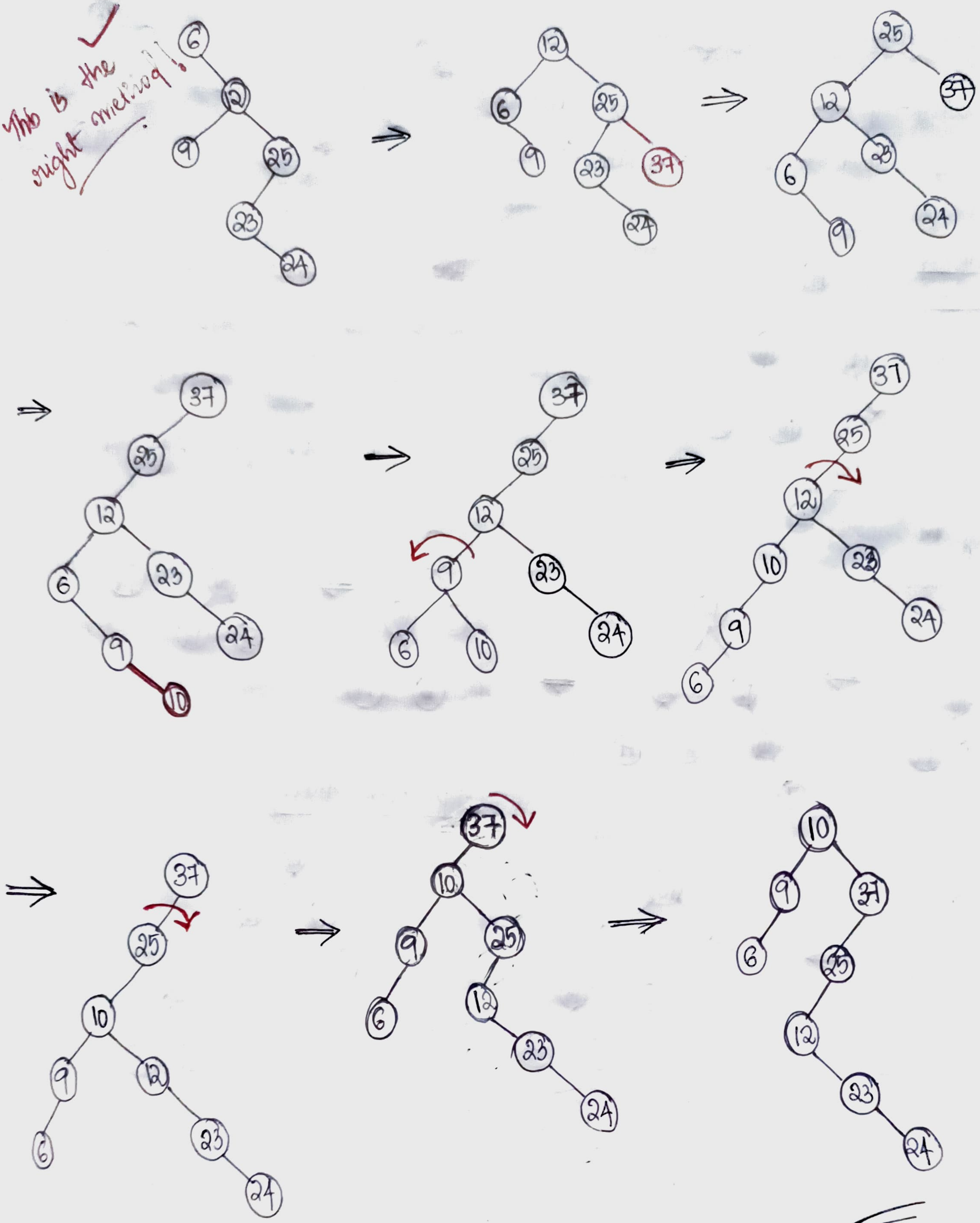












This is the right method!!

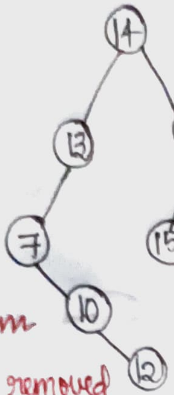
once we perform a 2 step rotation, after completing that ~~if~~ ~~as~~ if we have to continue the rotation then we can only perform ~~I~~ ~~an~~ one step rotation.

* Deletion operation in splay tree.

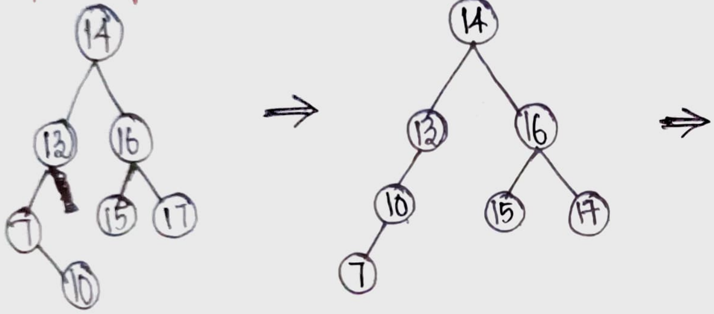
- bottom-up → delete + splay. (BST rule)
- top-down → splay + delete. (no rules)

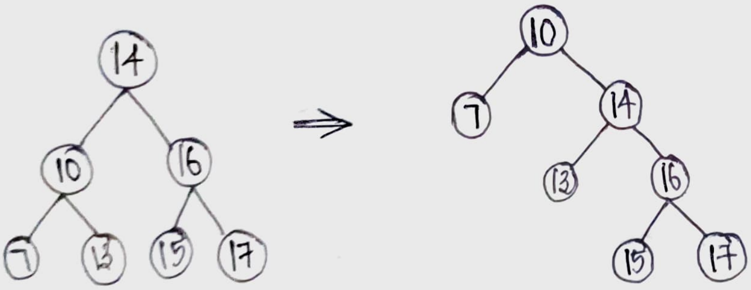
→ Bottom-up deletion.

Here follow BST rule, remove the node & perform splaying on removed node's parent.



Remove 12.

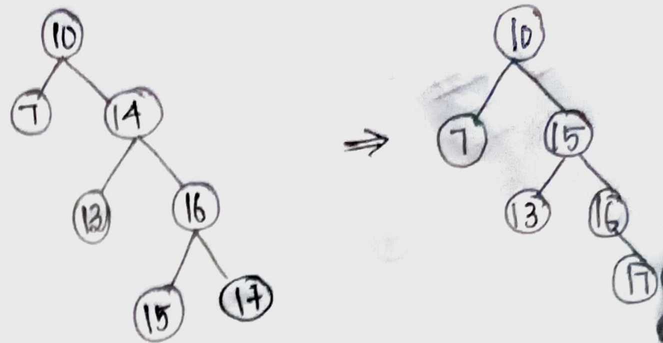




Remove 14 from this tree.

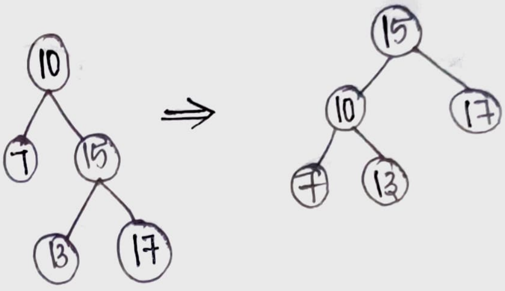
* Inorder successor → lowest in Right subtree.

* Inorder predecessor → Highest in Left subtree.

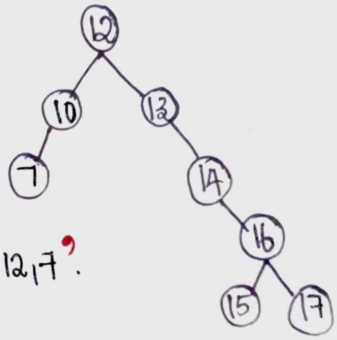


Here the parent of 14 is 10. 10 is already in root position so no splaying needed.

From the above final tree delete 16.



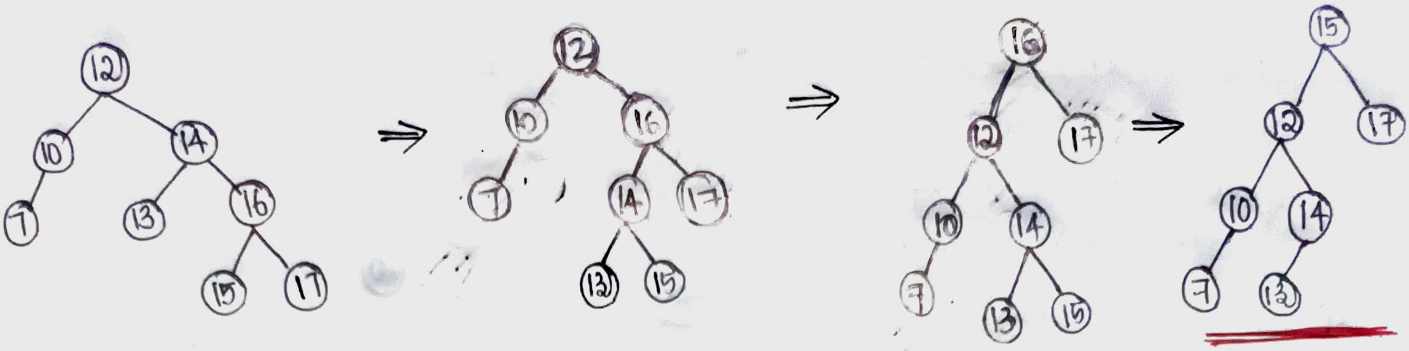
→ Top-down deletion.

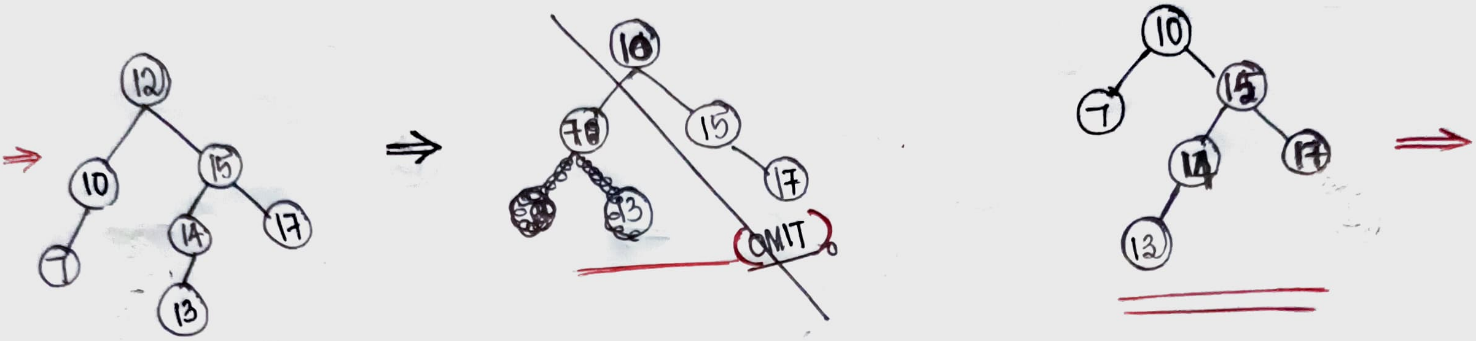


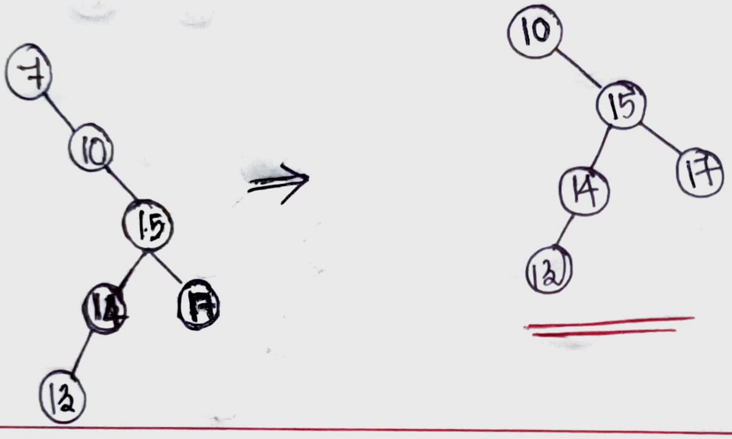
'Delete 16,12,17'.

Here perform the splaying on the to be deleted node & then do the deletion & then do the "join()" operation.

Join operation → find the highest element in the left subtree & then make it the root.





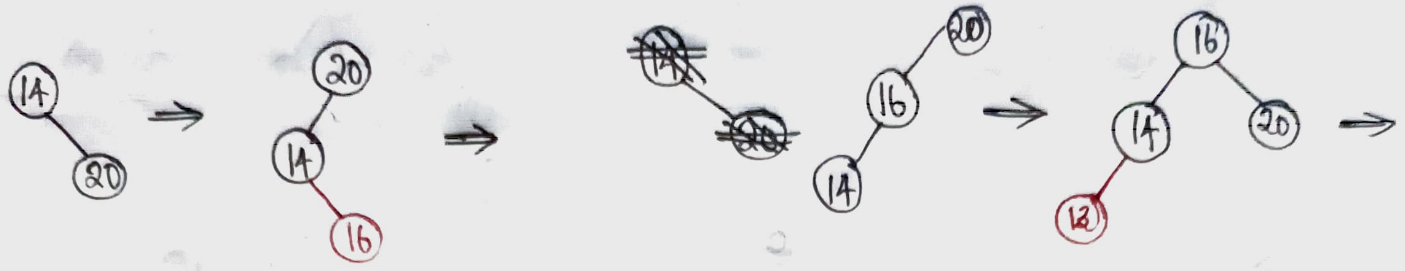


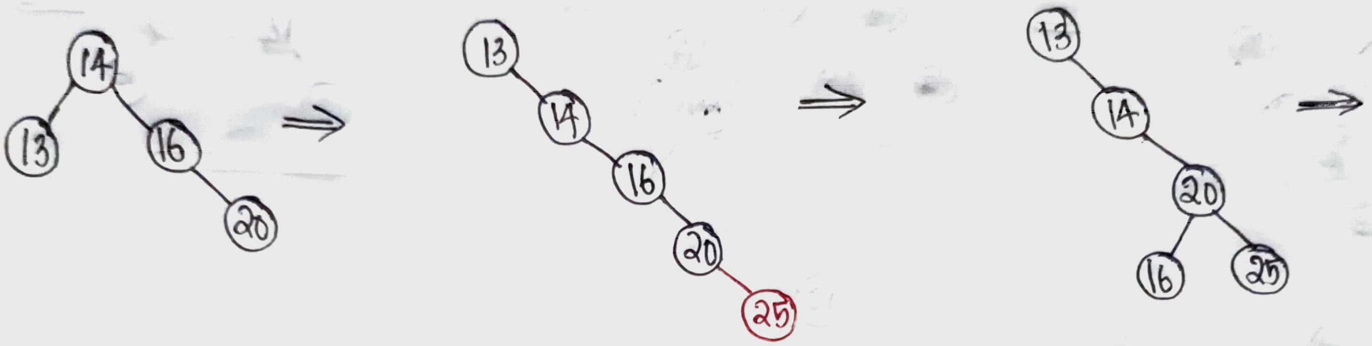
Suppose we are asked to find an element, in the tree, that is not there. for e.g. in the bottm. graph above we are asked to find 20. Then we have to look at the root node & look into whether the no: to be searchef will be on the right or left subtree. In this case since the root is 10, it will be on right subtree, so we will take the last element in the right subtree & splay it. Similarly, if it was on left subtree, we will find the last element there & splay it.

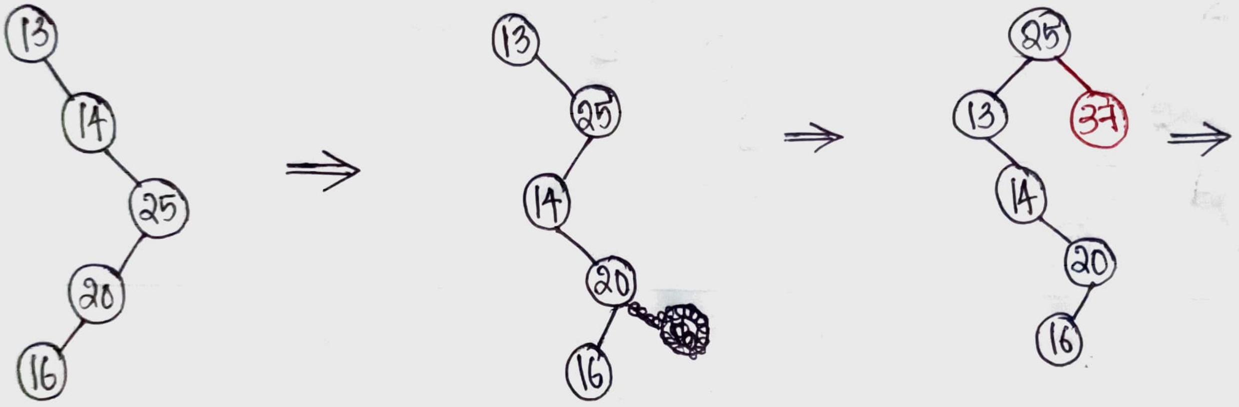
Q. Insert 14, 20, 16, 13, 25, 37.

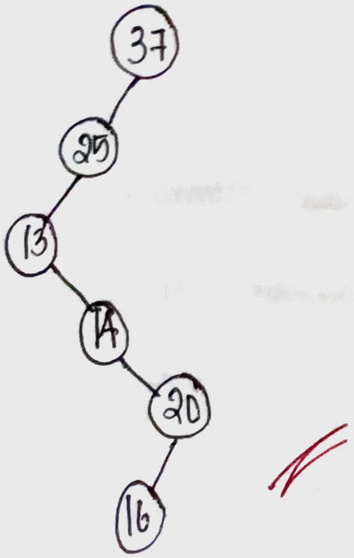
Search 20, 30.

Remove 13, 16, 37.

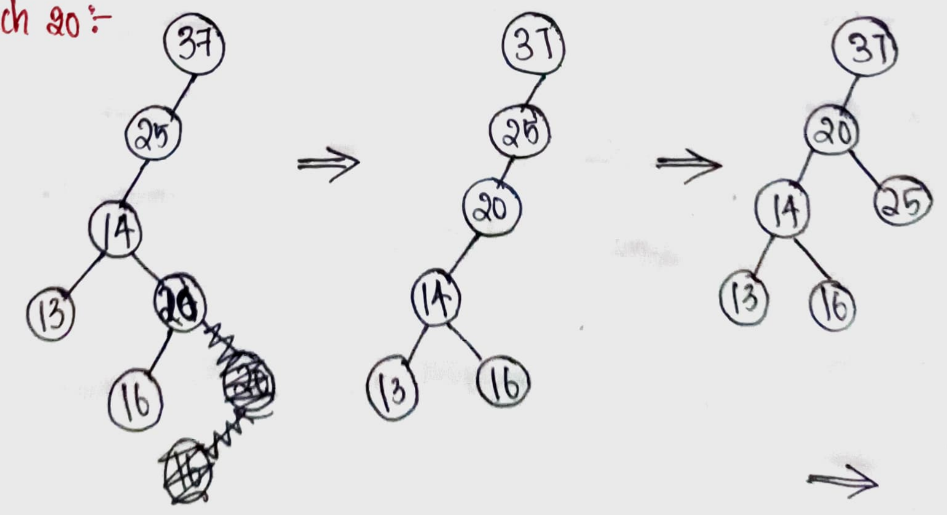


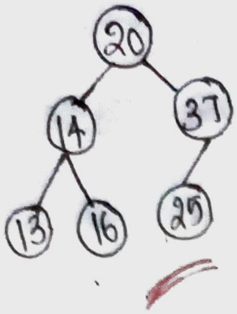




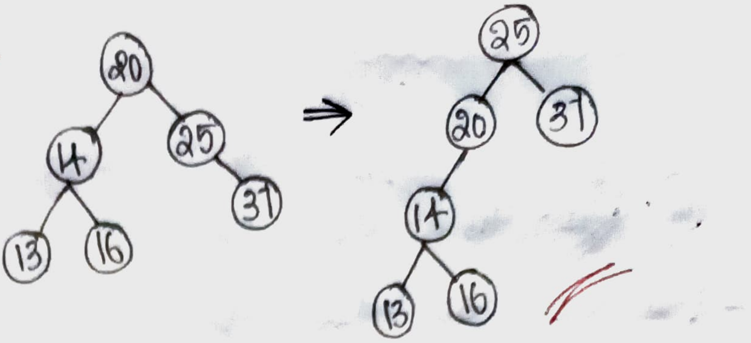


Search 20 :-



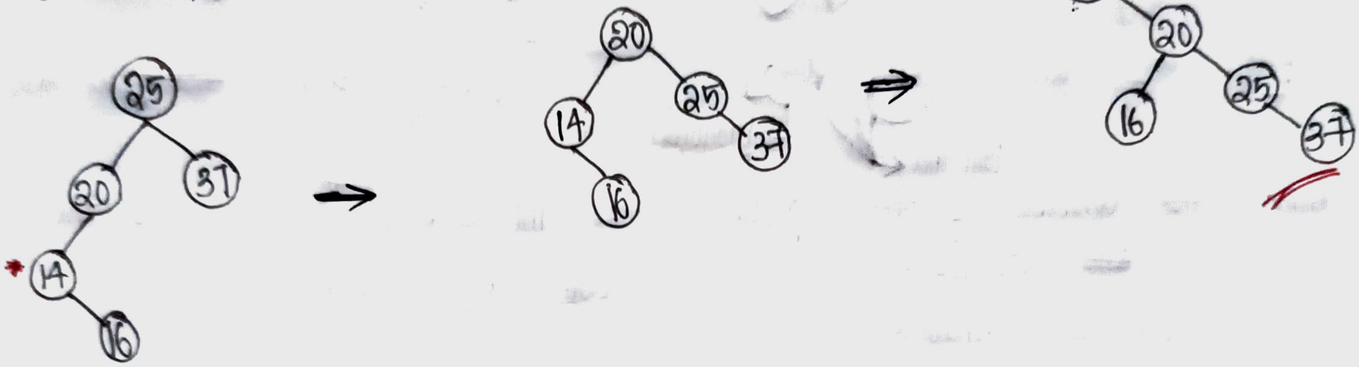


Search 30:-

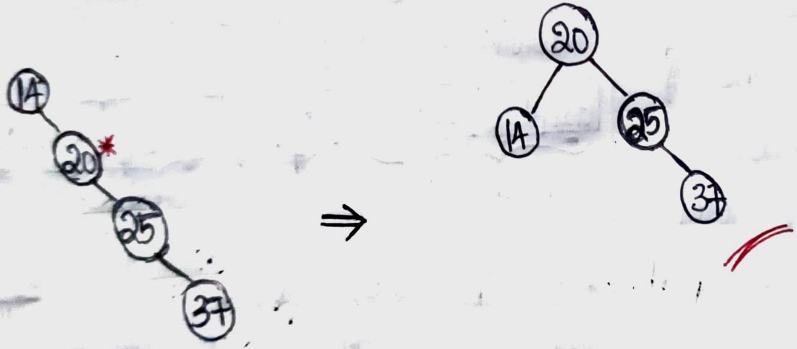


Remove 13:-

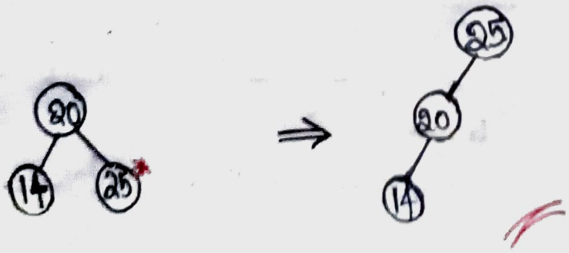
(Bottom-up)



Remove 16:-



Remove 37:-


20
14
25*
25
20
14