Splay Trees

→ Self-adjusted BST

Time Complexity in BST:

Worst Case $O(n)$ if tree is left skew or right skew

In all other Cases $O(\log n)$

Where as in Case of AVL trees time Complexity $O(\log n)$

Since these are self balancing trees.

In some practical situation, Can we do better than $O(\log n)$ time Complexity?

Splay tree is the solution.

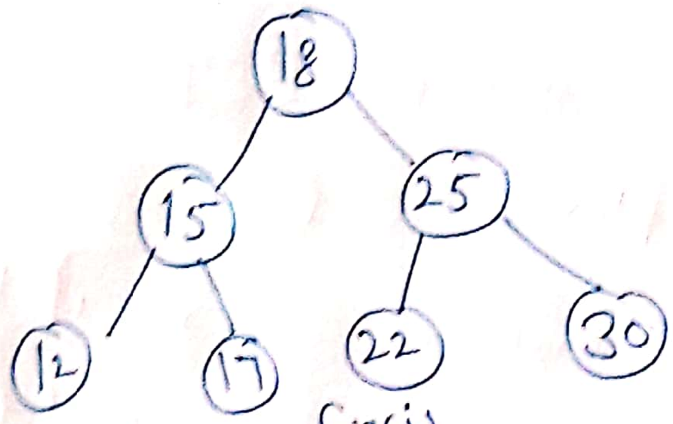
— Splaying property

All operations searching, Insertion and deletion ~~all~~ are similar to BST along with one more operation called Splaying is to be done.

— Not strictly balanced

eg:

Search 15

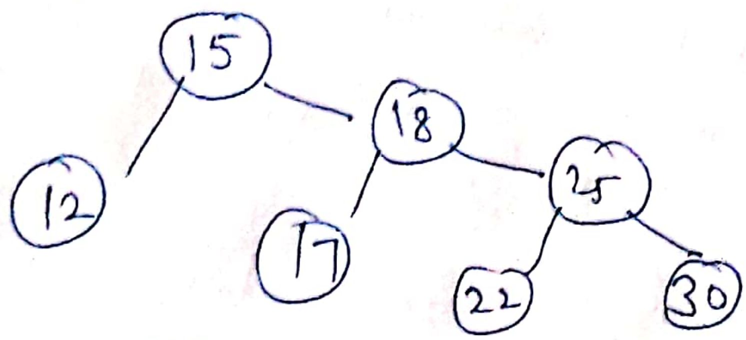


fig(i)

Similar to BST along with one more operation Called Splay that is make that element as root of the Tree. This process is known as splaying.

Splay Tree Self adjusted BST in which each operation on element, re arrange the tree so that that element will become the root of the tree. for re arranging the tree, perform some rotations.

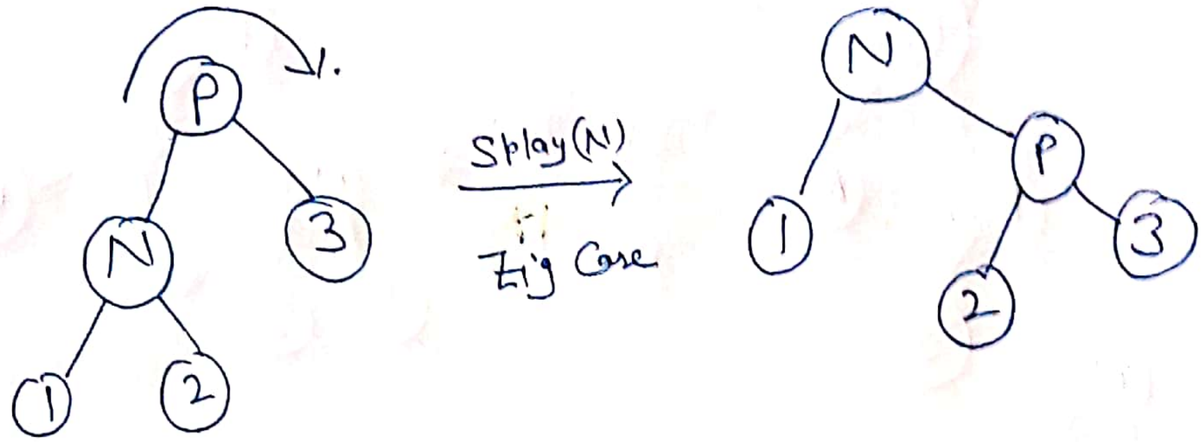
eg. In fig(i), To make 15 as root perform right rotation



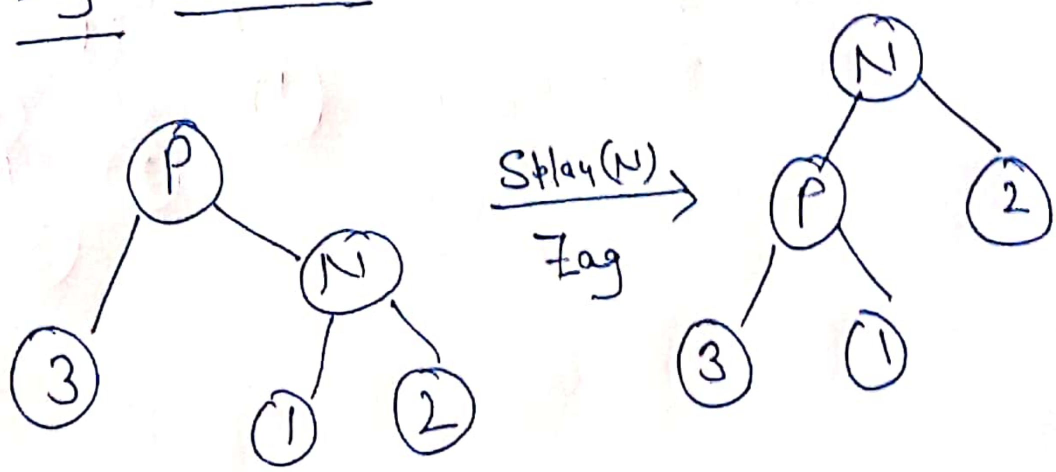
Splay Tree (Zig rotation)

1 Zig Rotation

(i) if the element on which splaying is to be performed is the left/right child of root node. That is, the node does not have any grand parent. eg.



(ii) Zag Rotation

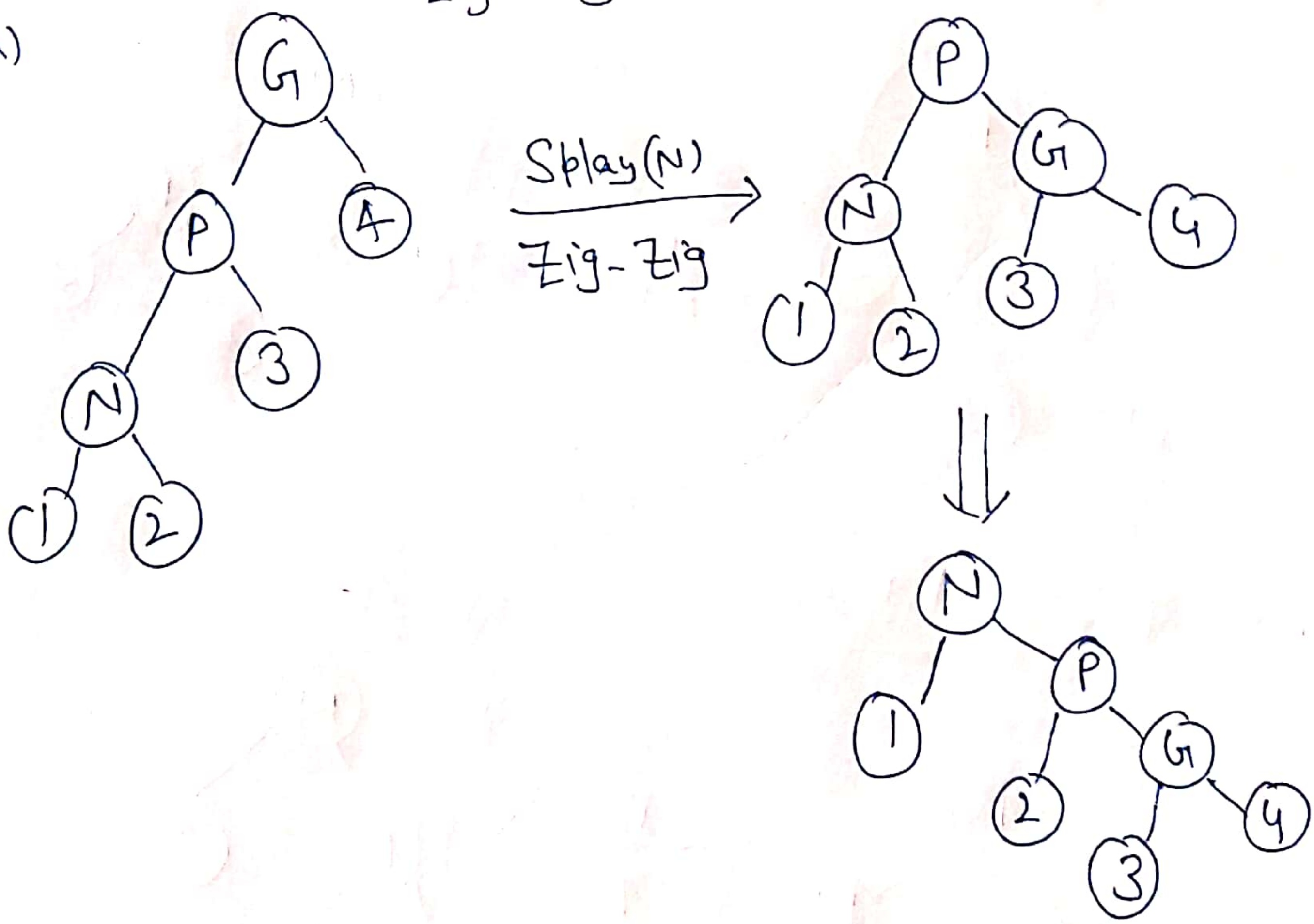


Case (ii)

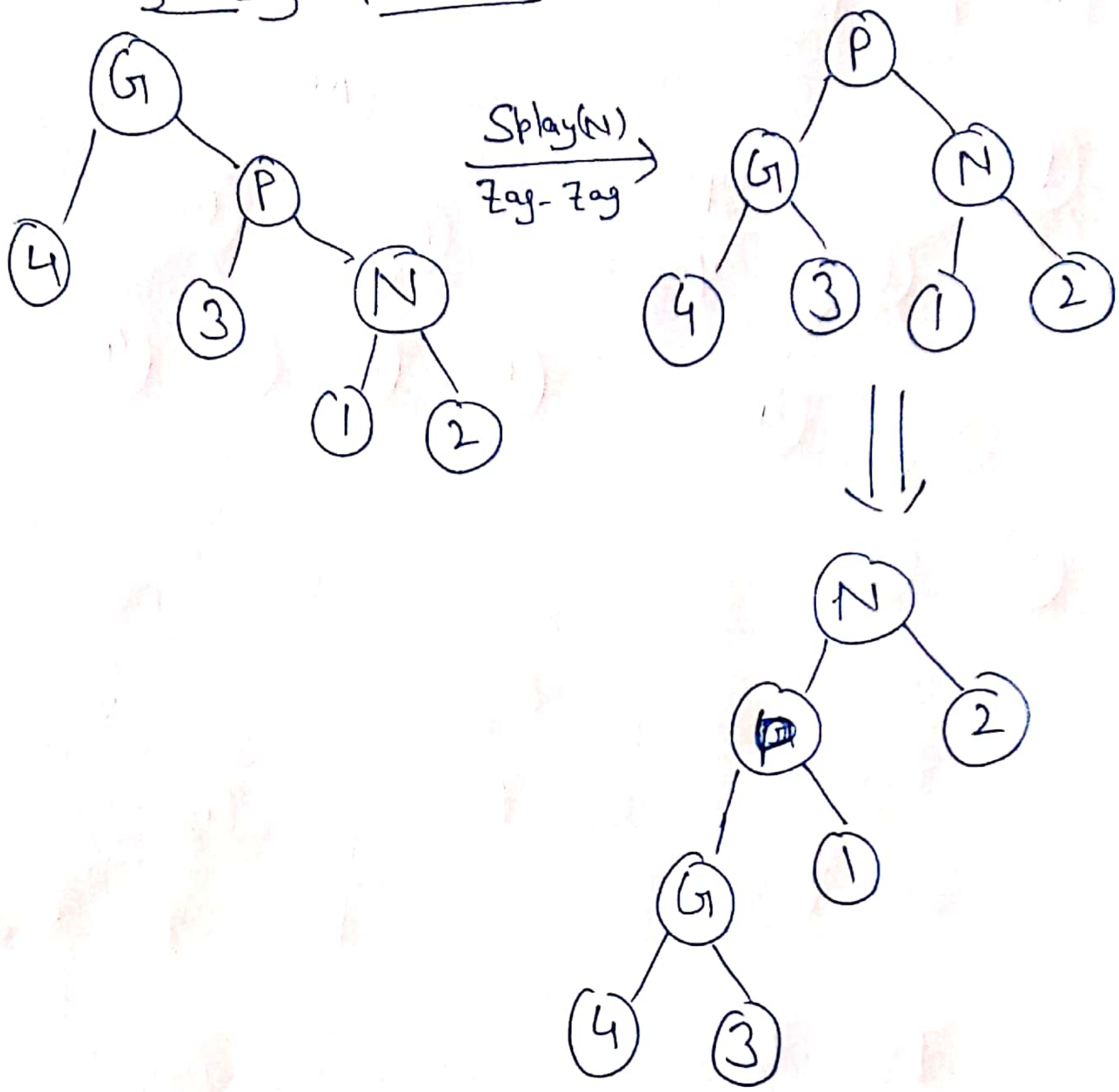
Suppose the node on which we want to perform splay operation has parent as well as grand parent. Then we have four cases.

Zig-Zig Rotation

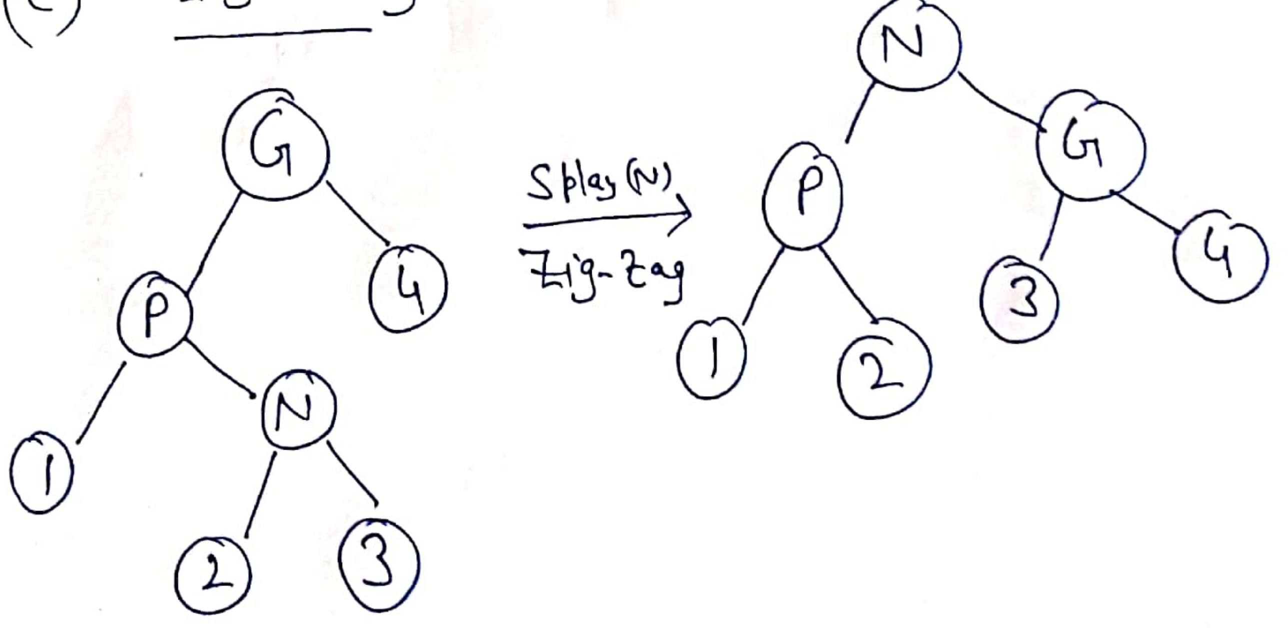
(a)



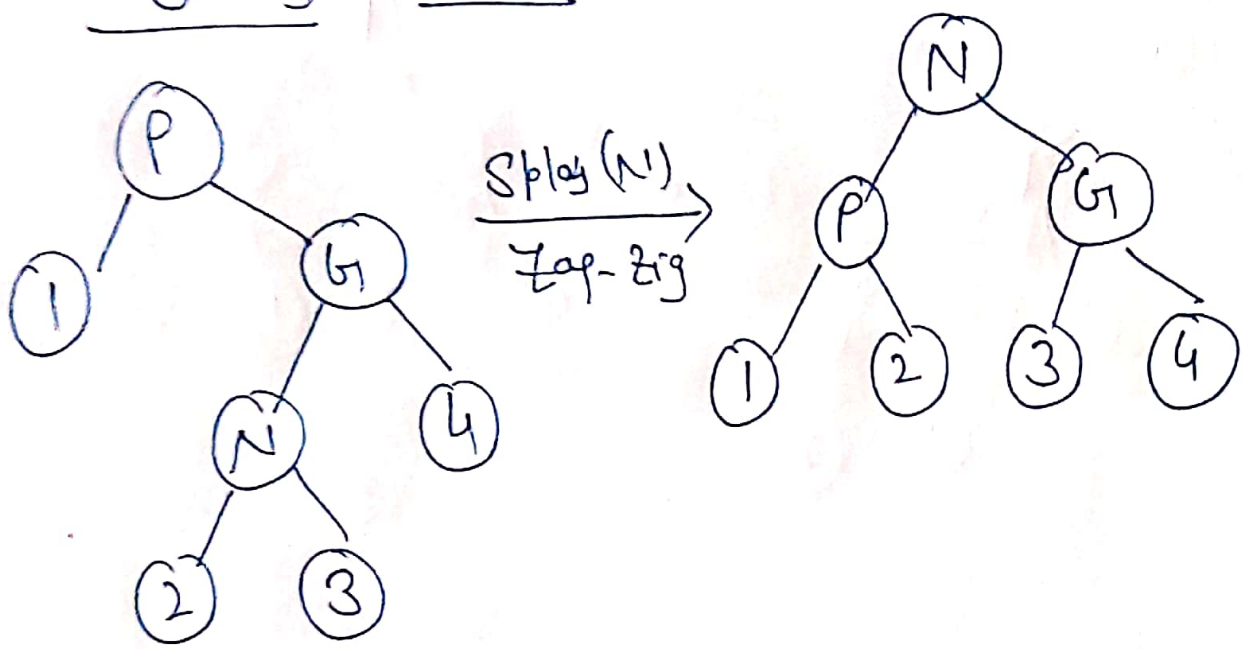
(b) Zag-Zag Rotation



(c) Zig-Zag Rotation



(d) Zag-Zig Rotation



So.

- Zig
  - Zig
  - Zag
- Zig-Zig
  - Zig-Zig
  - Zag-Zag
- Zig-Zag
  - Zig-Zag
  - Zag-Zig

if an element is most freaquently accessed element, then it will take O(1) time Complexity. which is the main advantage of Splay tree. That is most freaquently accessed element is near to the root.

Due to this advantage of Splay tree, in practical situation Splay tree is better than AVL tree.

→ Splay tree are used to implement Caches.

→ No extra info is stored.

→ Easy to implement.

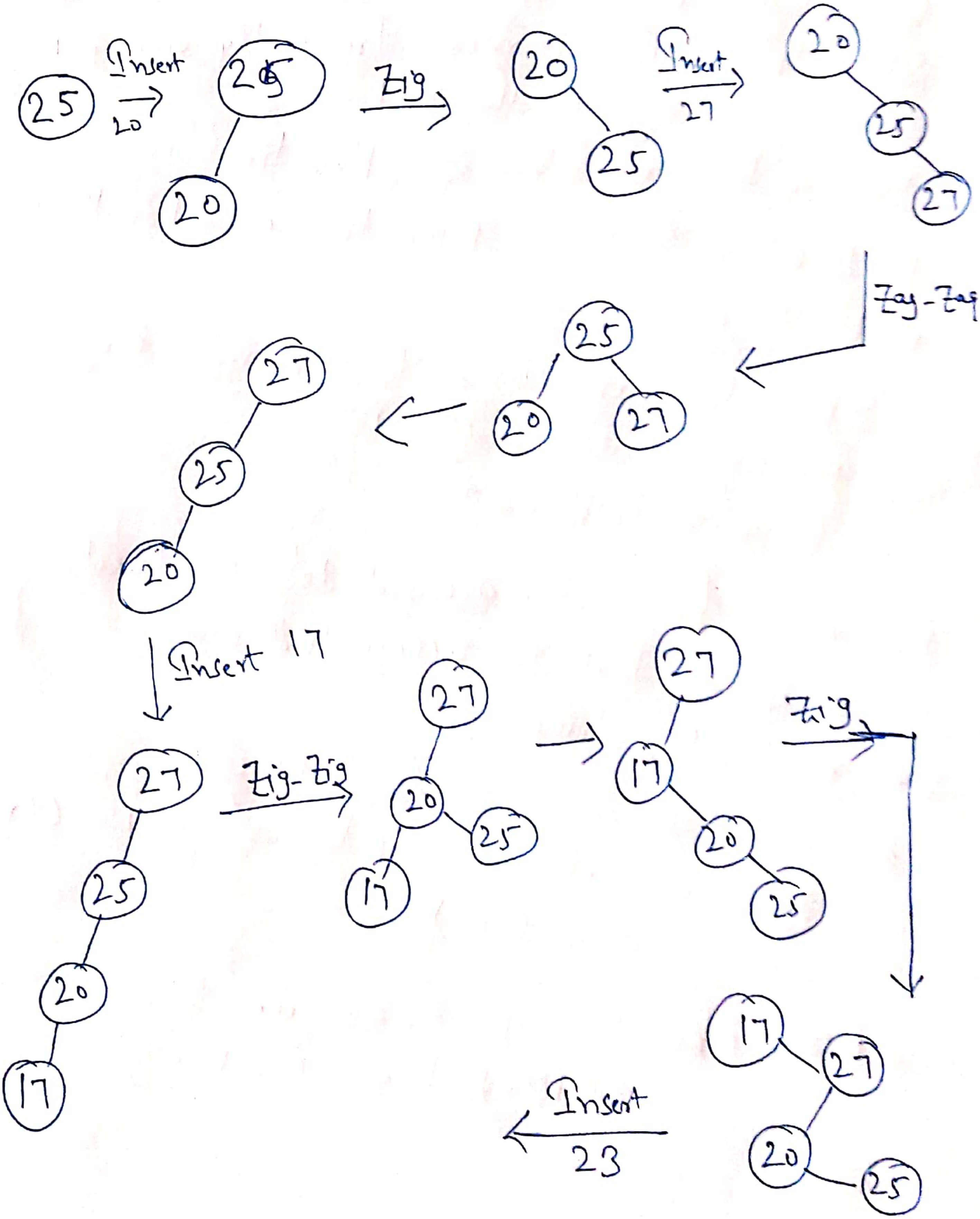
The most freaquently accessed data is stored in Caches. So that to access that data we need less time.

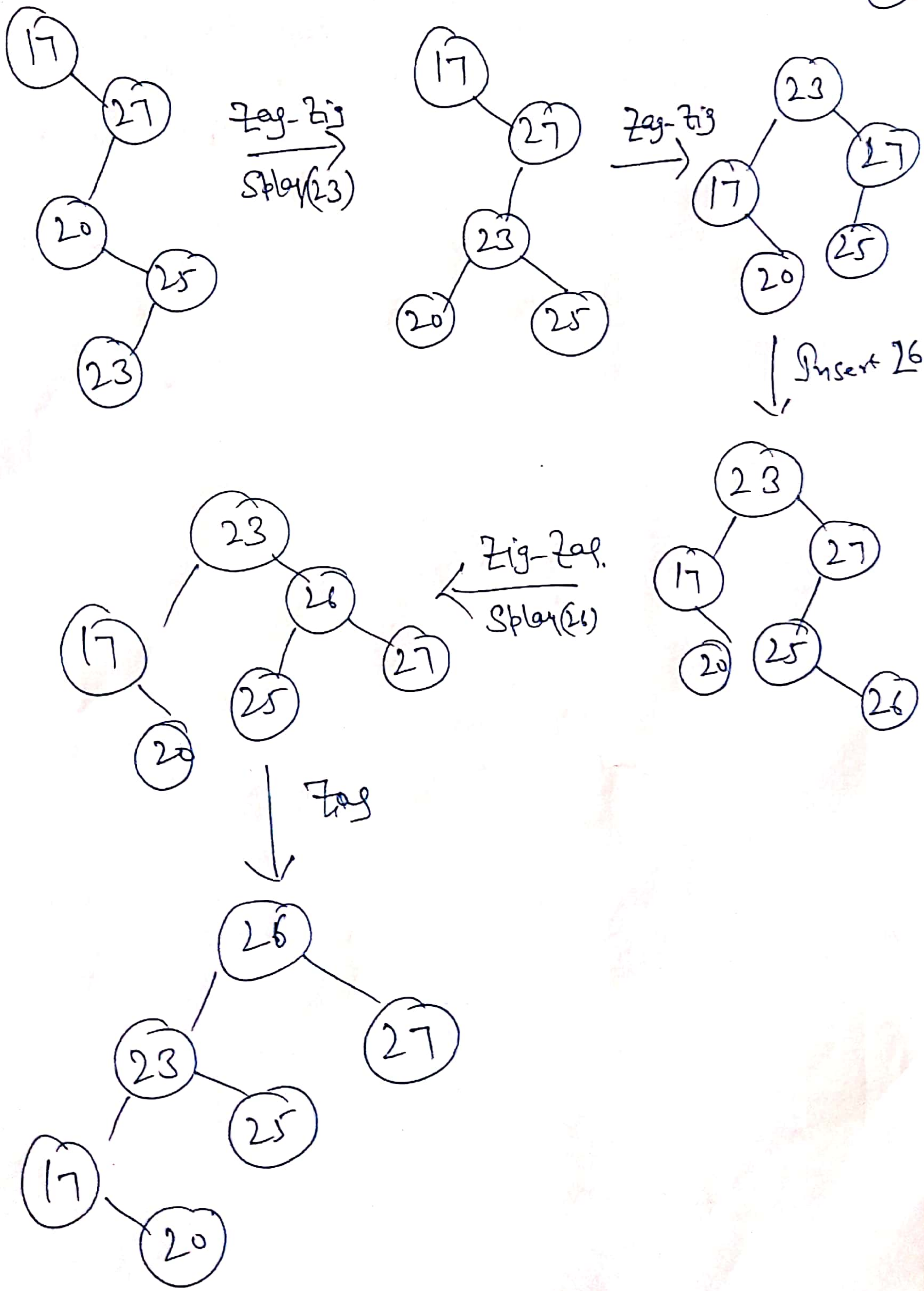
→ Since it is not strictly balanced so sometimes height may be linear i.e. O(n) (Very rare Case)

→ Max. operations in Splay tree take O(logn) time Complexity.

Construction of Splay Tree (Insertion operation)

25, 20, 27, 17, 23, 26

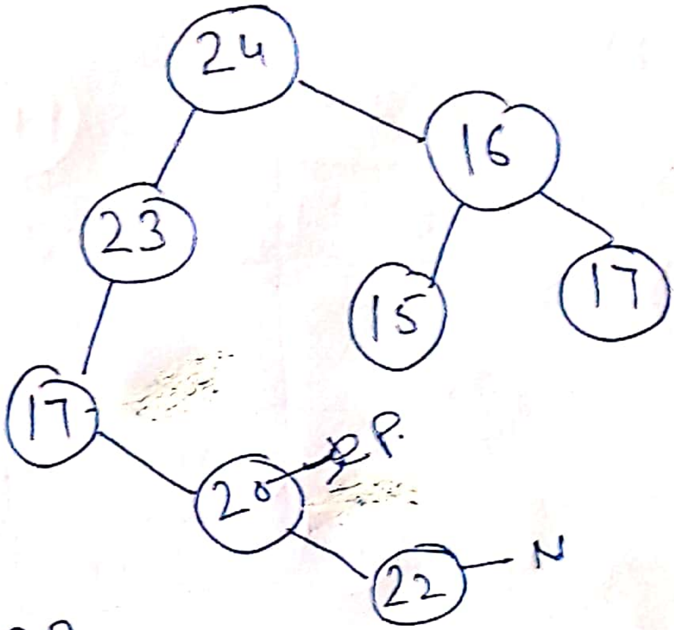



17
27
20
25
23
Zag-Zig
Splay(23)
17
27
23
20
25
Zag-Zig
23
17
27
20
25
Insert 26
23
17
27
20
25
26
Zig-Zag
Splay(26)
23
17
26
20
25
27
Zag
26
23
27
17
25
20

Deletion of an element from Splay Tree

Case 1

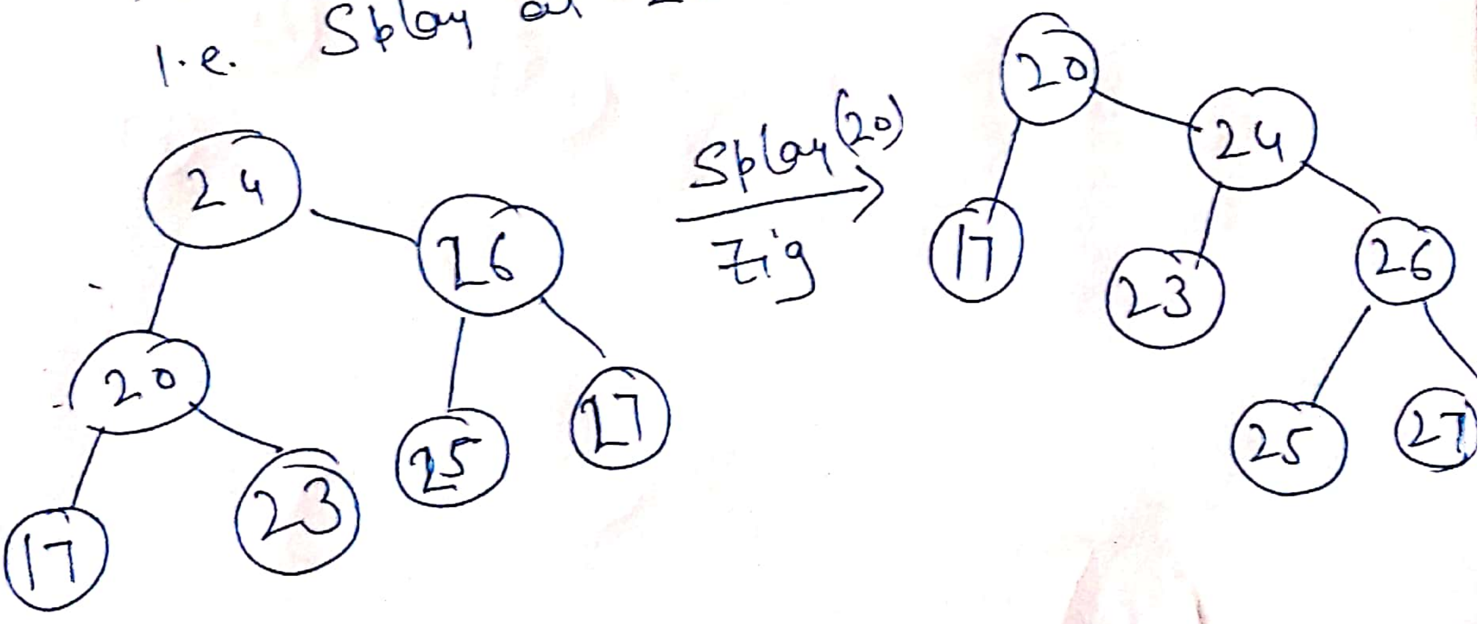
eg.



Delete 22

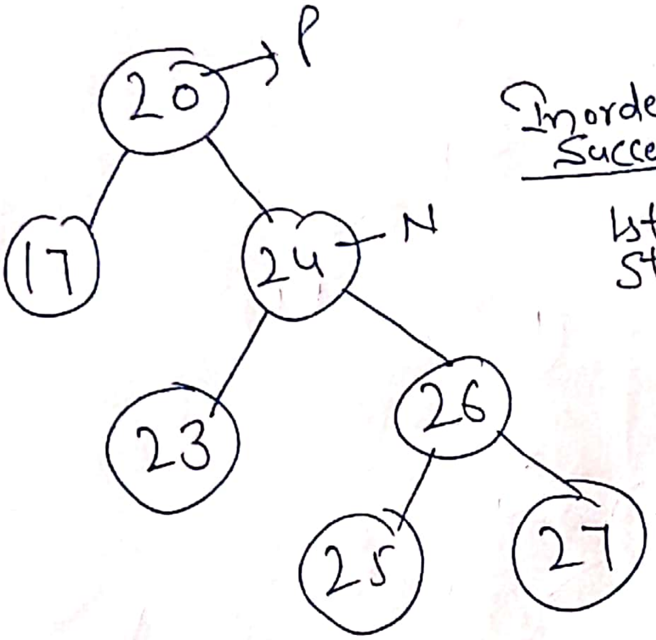
if the node which we want to delete has no child, then delete it and perform splaying operation at its parent node.

i.e. Splay at 20

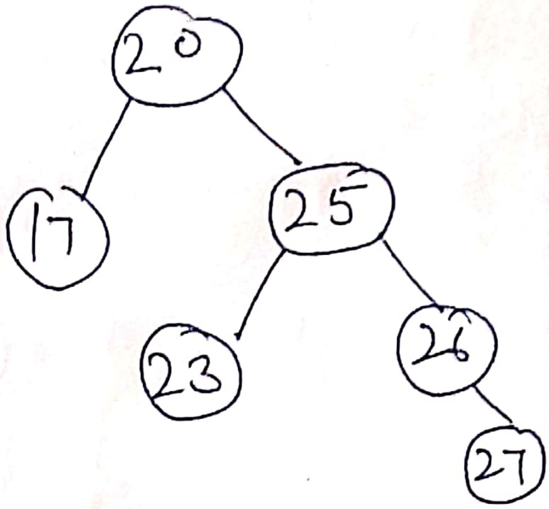


Case

Delete 24

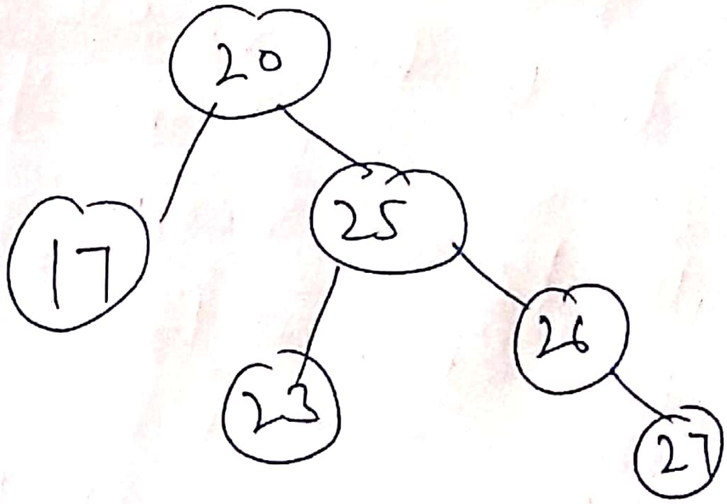


Inorder Successor
1st Step



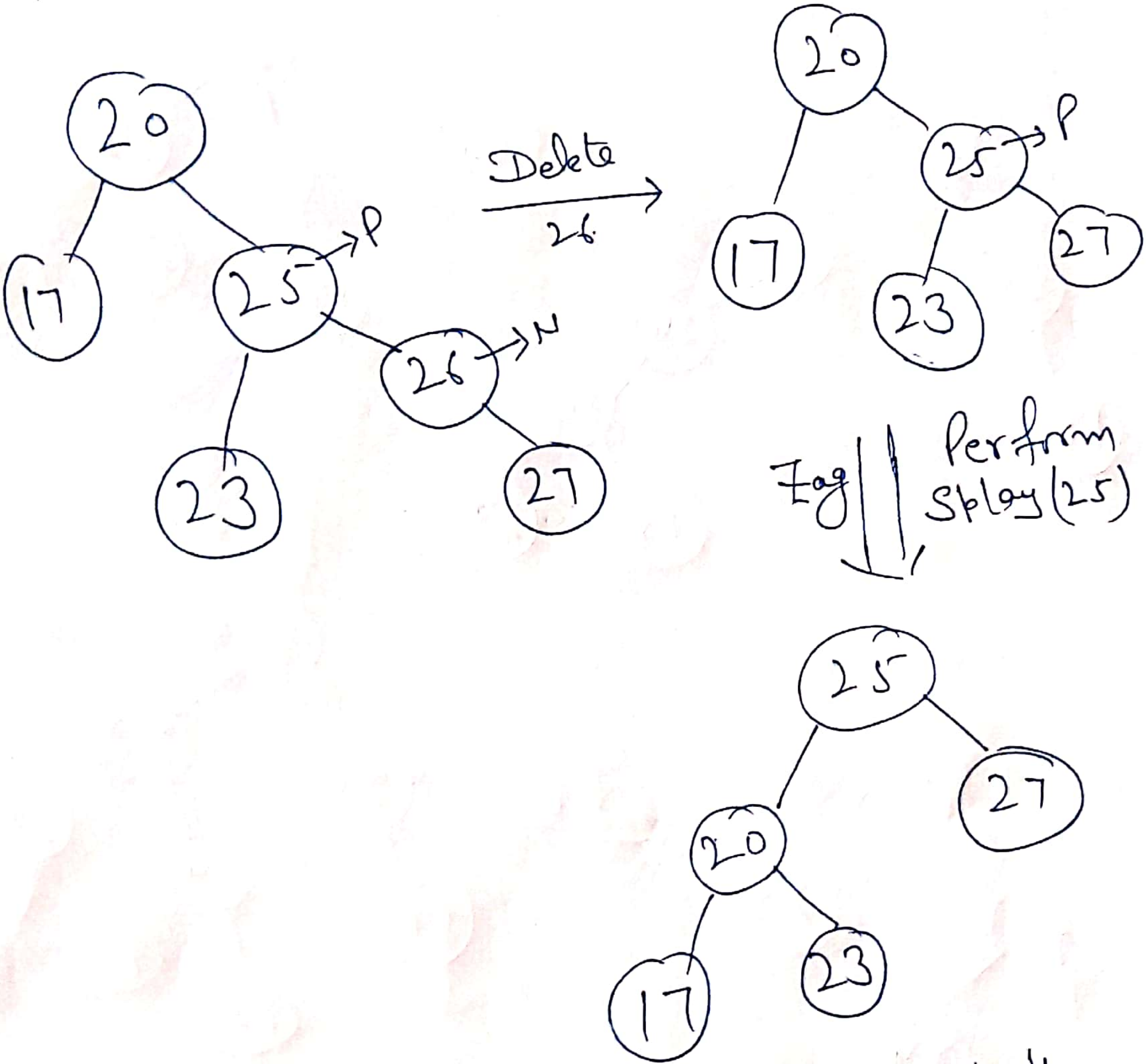
Perform Splaying operation on the parent of deleted node

No change

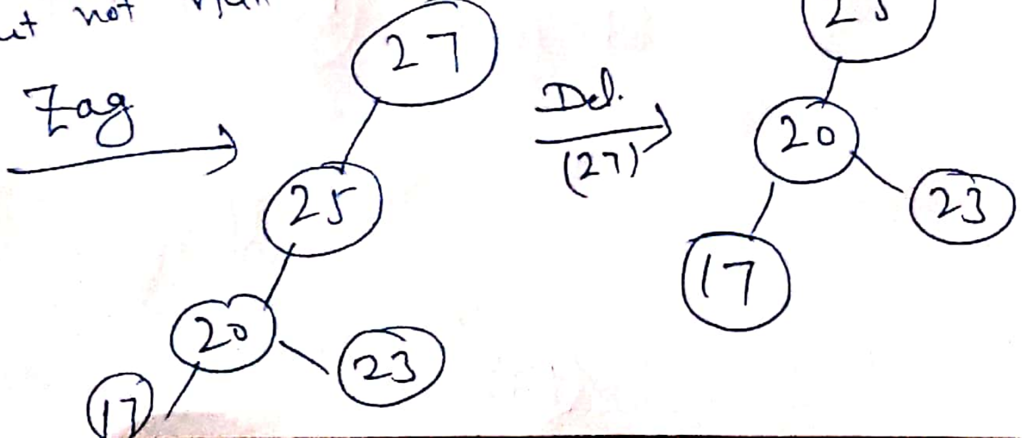


Scanned with CamScanner

Case (iii) Delete 26



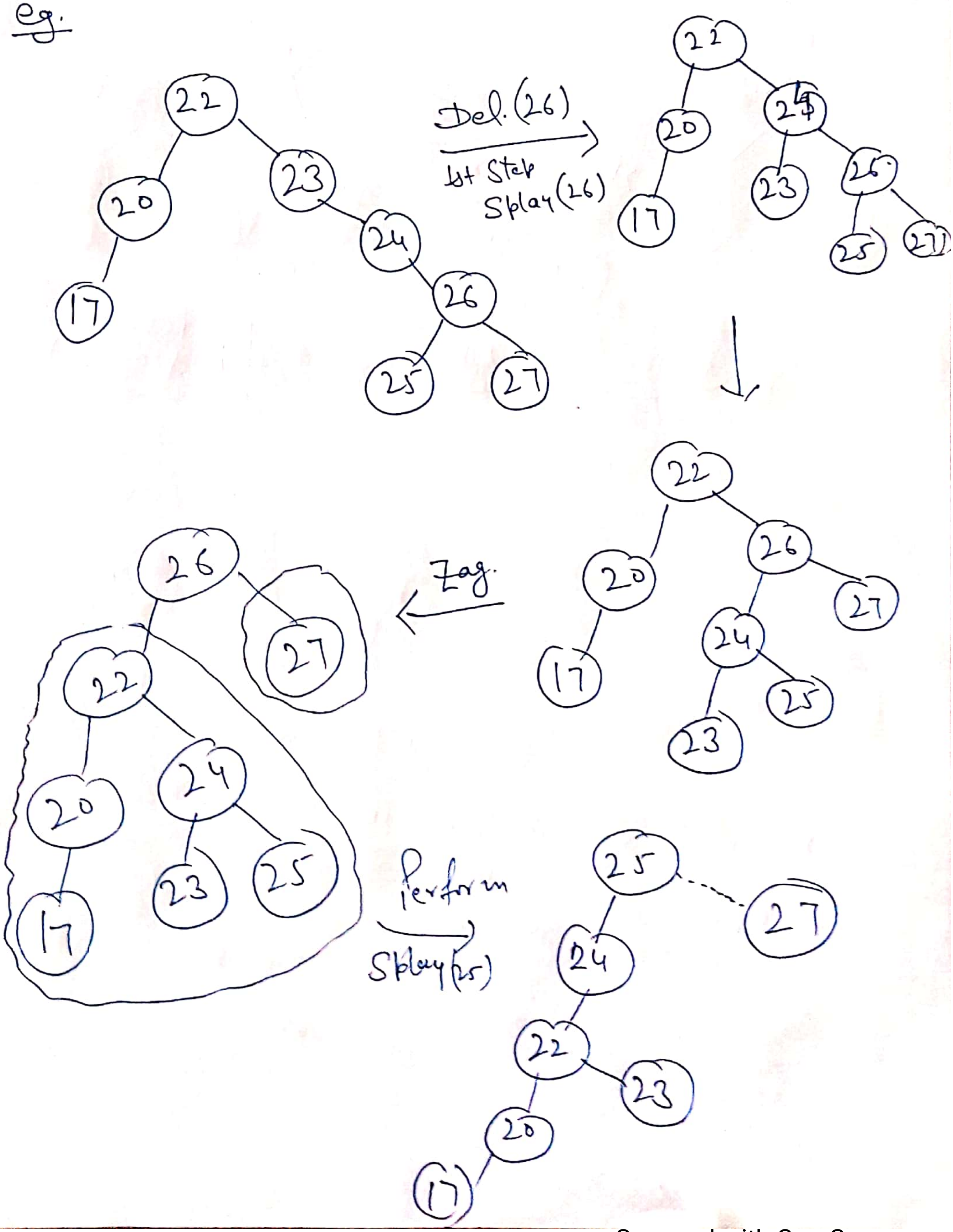
Case (iv) Delete 30, if data is not present then Perform Splay on the node which is last accessed but not null node.



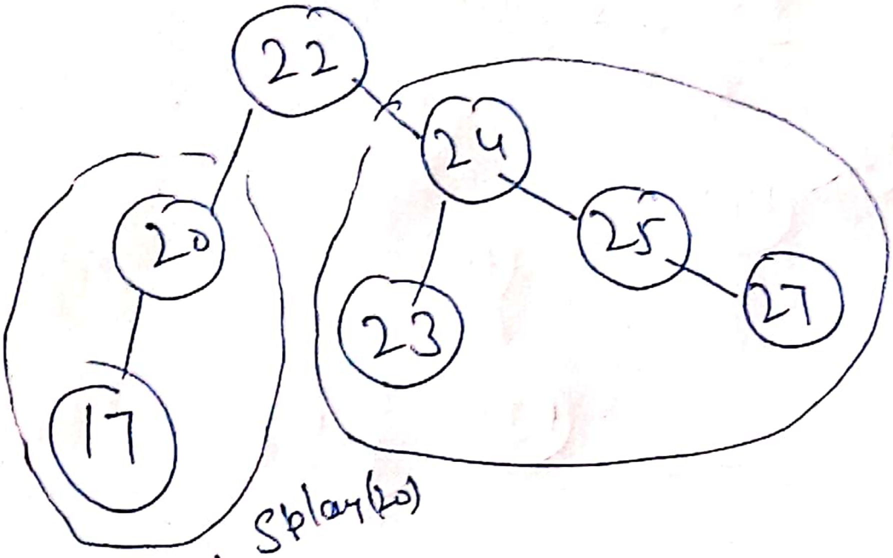
Scanned with CamScanner

Deletion (Top-Down Approach) in Splay Tree.

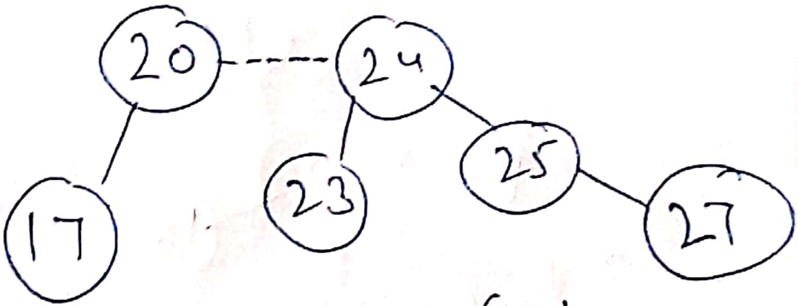
eg.



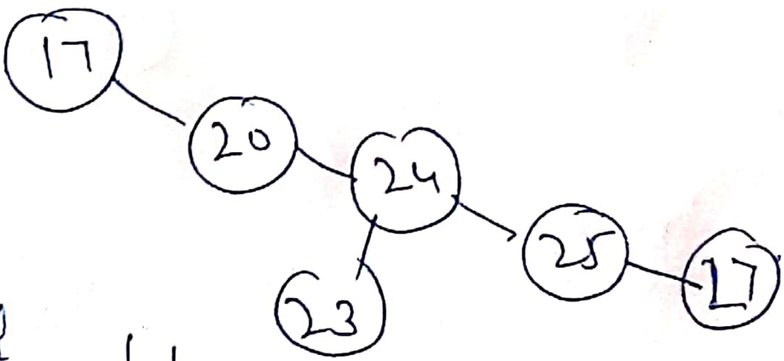
Delete 22



Splay(20)

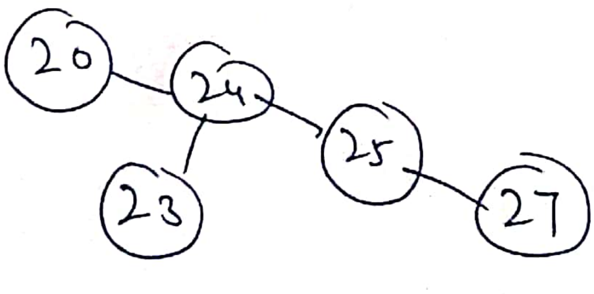


Del. (17)

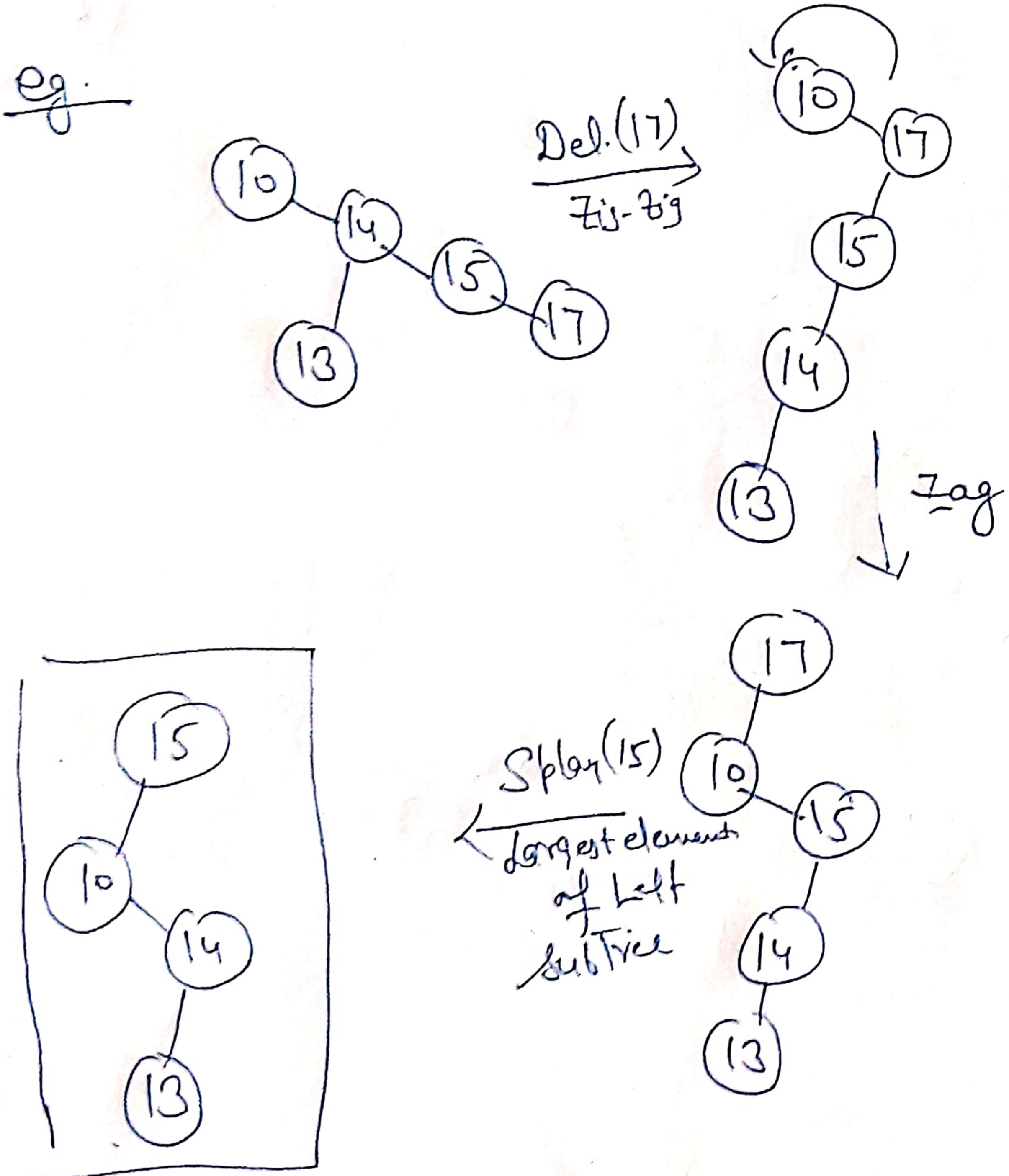


Root of right subtree become the root.

No left SubTree



eg.



final Tree: