GOODLUCK | Page No.
Date / /

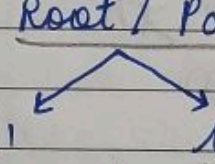
Assignment 1.

Name : Shruti Deepak Dhumne
Roll no : 227 Batch : E1
PRN: 0220200161
Subject : ADS

Q1. How Binary tree are stored ?

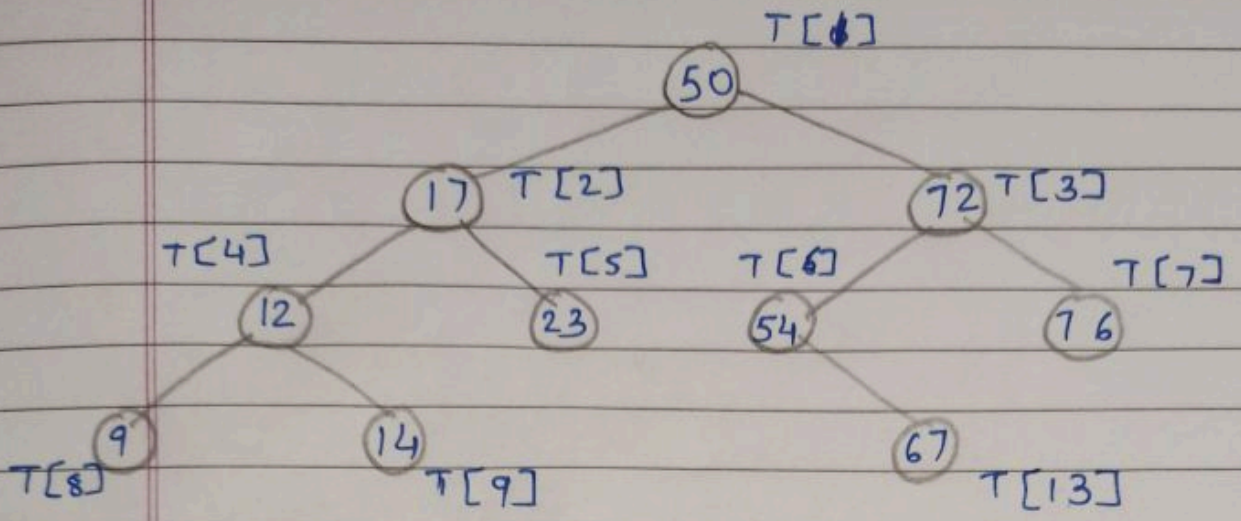
→

- A binary tree is a non-linear data structure in which data elements are stored in form of nodes and every node has at most two nodes.



- The very first starting node is called 'root' node.
- There are two ways to represent Binary tree in memory
  1 Sequential Rerepresentation
  2 Linked Rerepresentation

① Sequential Rerepresentation : This representation uses a linear array. (T size)
- Root node is stored in T[1]
- If the root occupies T[K] then its left child is stored in T[2*K] and its right child is stored into T[2*K+1].

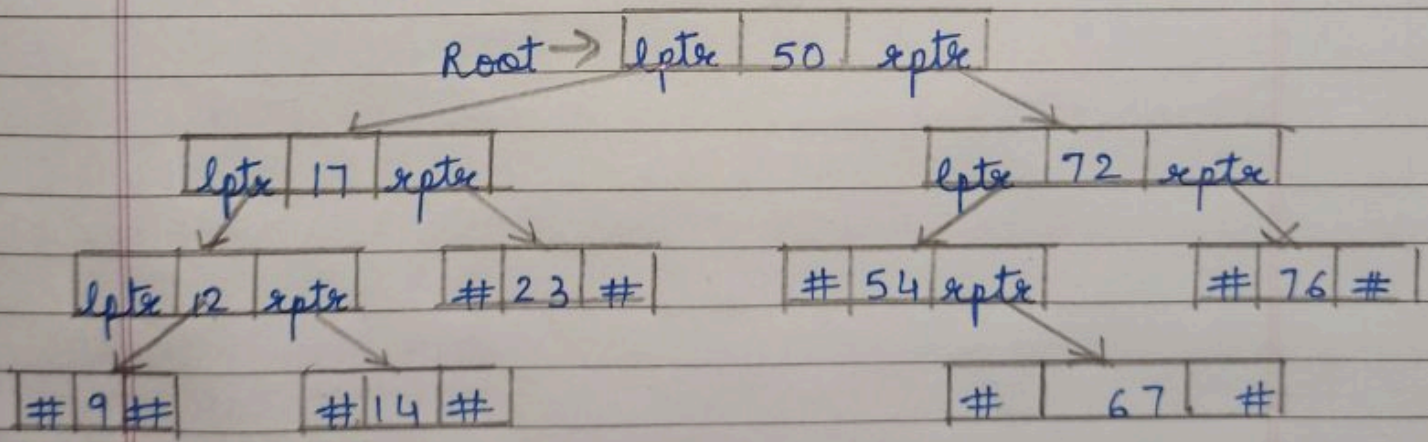


Array Representation →

50	17	72	12	23	54	76	9	14	-	-	-	67	-	-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(2) Linked Representation:

In linked representation we store each element in a node. Each node has two pointer fields and one data field.

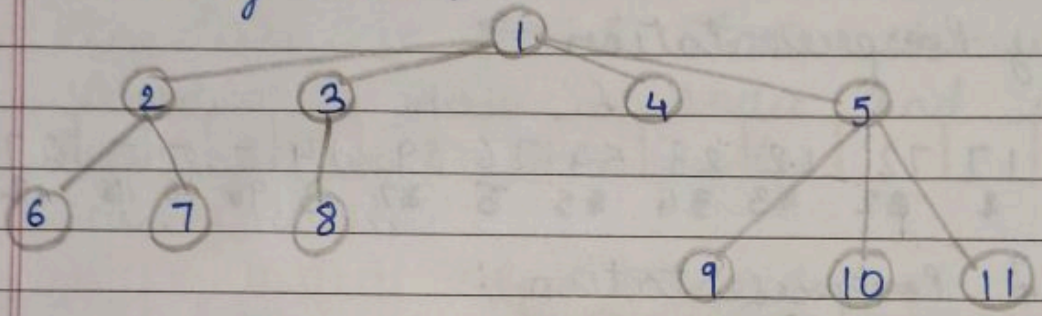


- If the left pointer (lptr) points to the left child and right pointer (rptr) points to the right child of the node.
- If, no link exists then it is set as 'NULL'.

eg:-

```
class Node {
        int data;
        Node * lptr, * rptr;
        Node () {
            lptr = rptr = NULL;
};      }
```

Q2 How m-ary tree is converted into Binary tree?

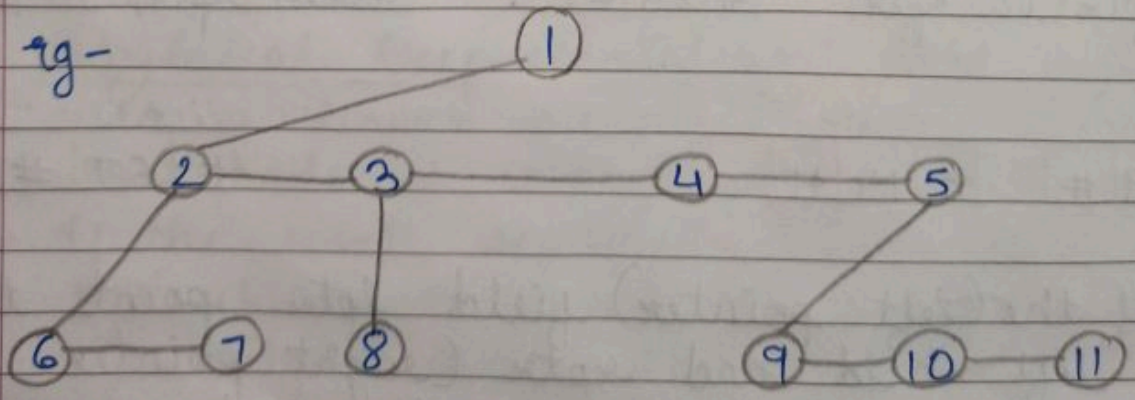


→ Step 1→ (algorithm)

(i) We delete all the ~~pren~~ branches originating in every node except the left most branch.

(ii) We draw edges from a node to the node on the right, if any, which is situated at same level.

eg-

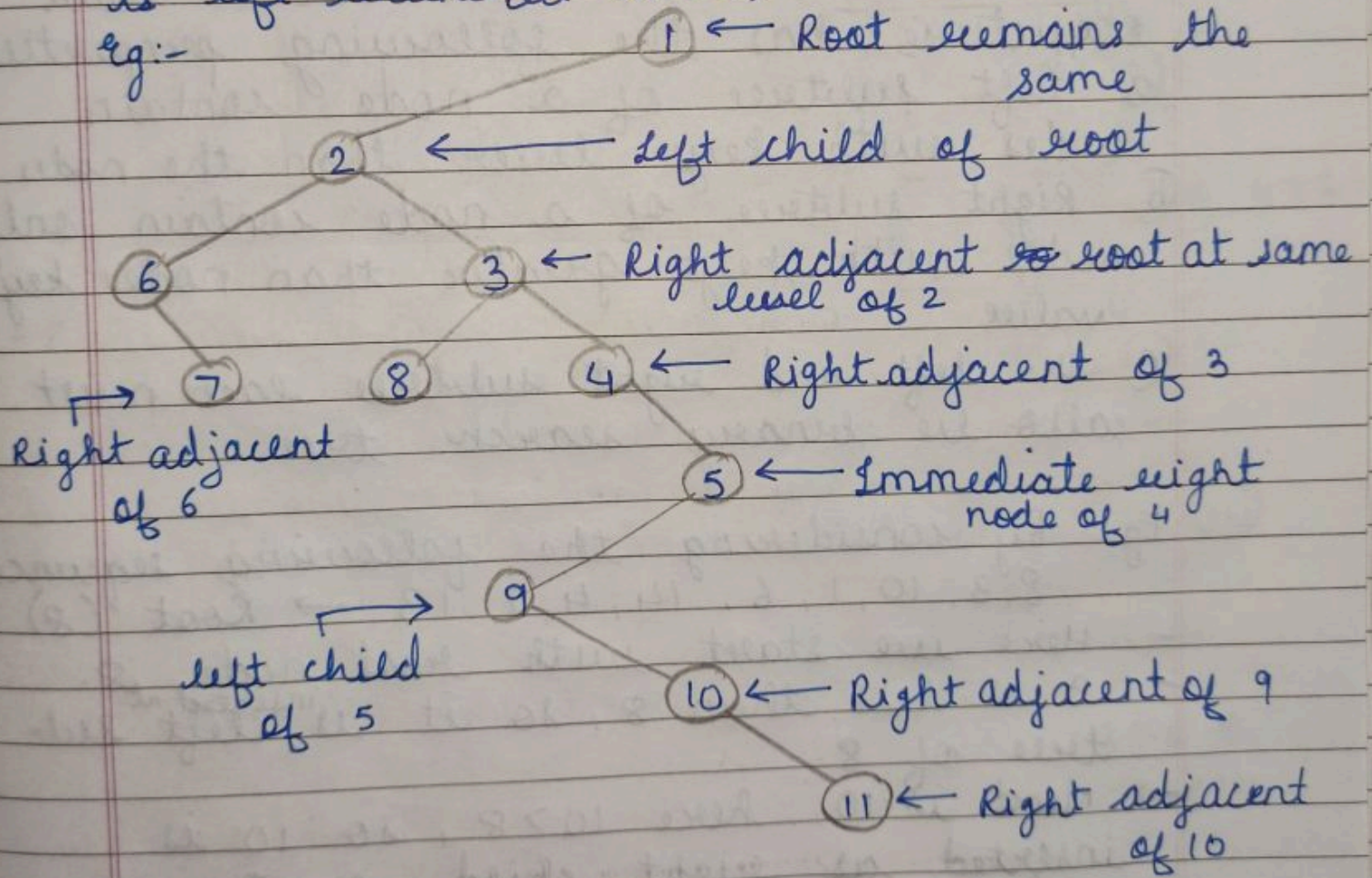


Step 2 →

(i) Once this is done then for any particular node we choose its left and right child as given below →

(ii) If left child is the node which is immediately below the given node and the right child is the node to immediate right of the given node on the same horizontal line. (Thus result is left Rbranched tree)

eg:-



Q3. What is binary tree? ----

→ **Binary search Tree:**

Binary search tree is a special kind of binary tree having maximum two children for every node, such that the left child of every node is always less than node and the value of right is always greater than the node value.

Explaination → This node based data structure has the following properties.

(a) Left subtree of a node contain nodes with keys lesser than the nodes key value.

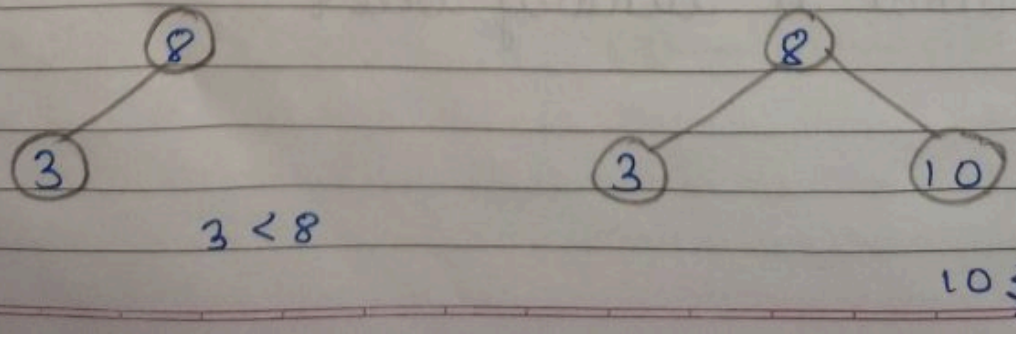
(b) Right subtree of a node contain only nodes with keys greater than nodes key value

(c) The left and right subtree each must also be binary search tree.

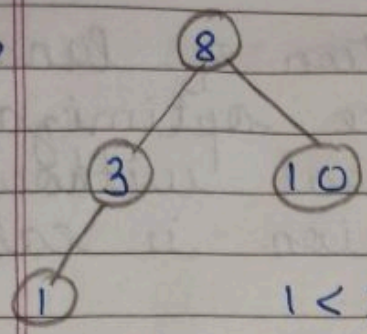
Eg: By considering the following sequence:
8, 3, 10, 1, 6, 14, 4, 7, 13 → Root (8)

- Here we start with root node 8.
- 3 is less than 8, so it is inserted at left sub tree of 8.
- Next is 10; here 10 > 8, so 10 is inserted as right child creating right subtree of 8.

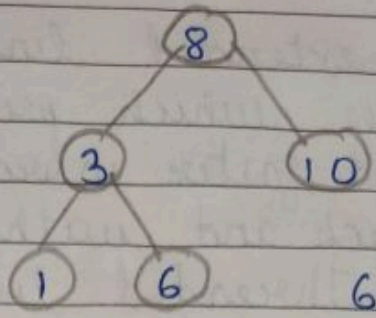
⇒



⇒

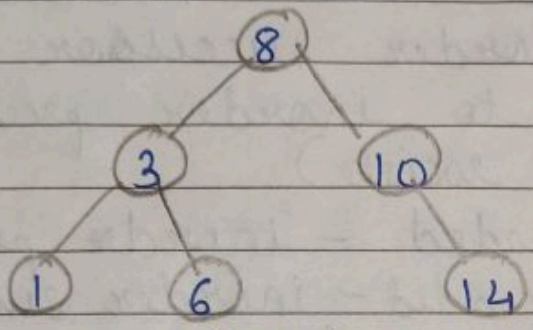


1 < 3 < 8

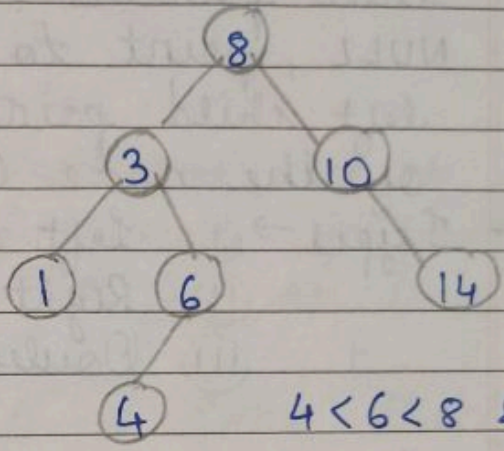


6 > 3 & 6 < 8

⇒

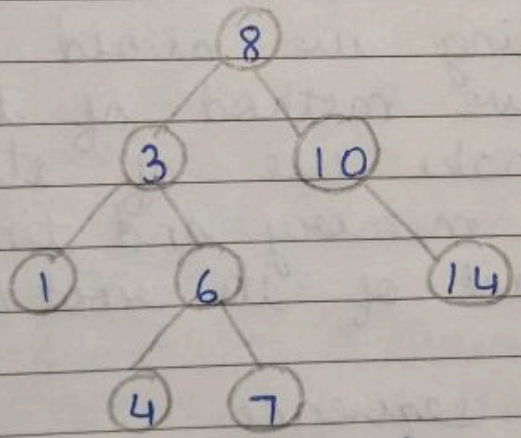


14 > 10 > 8



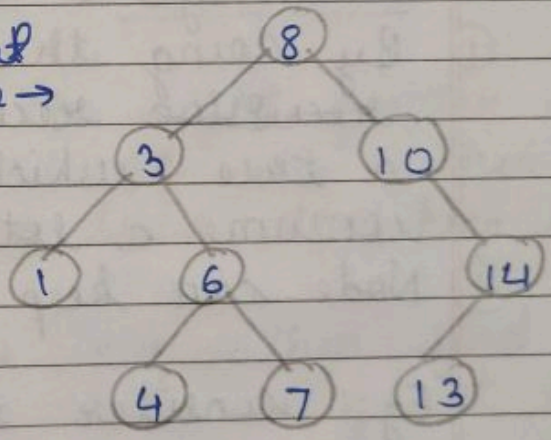
4 < 6 < 8 & 4 > 3

⇒



7 < 8 & 7 > 6

Final tree →



13 > 8 & 13 < 14

Thus, binary search tree has been created.

Q4. What is ~~Threaded~~ binary tree? ---

→ - An extended linking variation of binary tree which provides space optimization and faster inorder traversal without stack and without recursion is called as Threaded binary tree.

- It is made by making all the right child pointers, that would normally be NULL, point to inorder successor & left child pointers to inorder predecessor of the node (if exists)
- Types → (I) Left threaded - inorder predecessor
  (II) Right threaded - inorder successor
  (III) Double threaded - both.

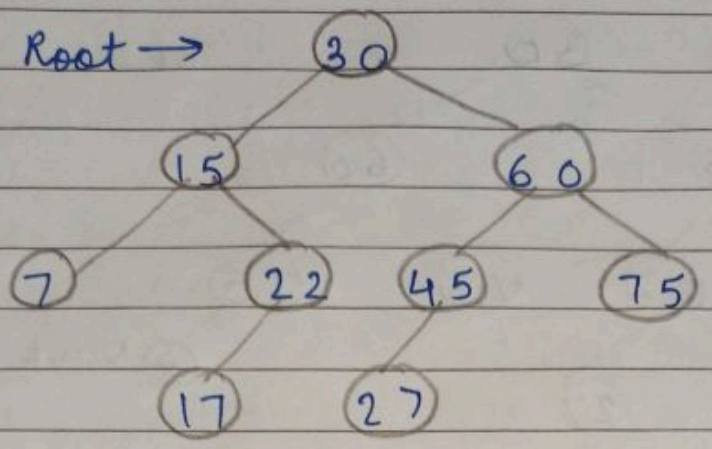
Advantages:

(I) By doing threading we avoid the ~~recursion~~ recursive method of traversing a tree which make use of stack and consume a lot of memory and time.

(II) Node can keep track of its root / parent.

Eg: Consider the sequence
30, 15, 60, 7, 22, 45, 75, 17, 27

(I) Construct the binary search tree first for the given sequence (starting from root as 30)



Initial Binary tree (Un-threaded)
→ int data
→ Node *left, *right

Algorithm:

(I) Starting from the root node check if the left pointer field of node is empty. If empty, then set its thread as true else make it false.

(II) Now if the L-thread is true, then find the inorder predecessor and assign it to the left of the node.

(III) Similarly, check if the right pointer field of node is NULL, ~~if~~ NULL, then set ~~#~~ its R-thread as true else keep it false.

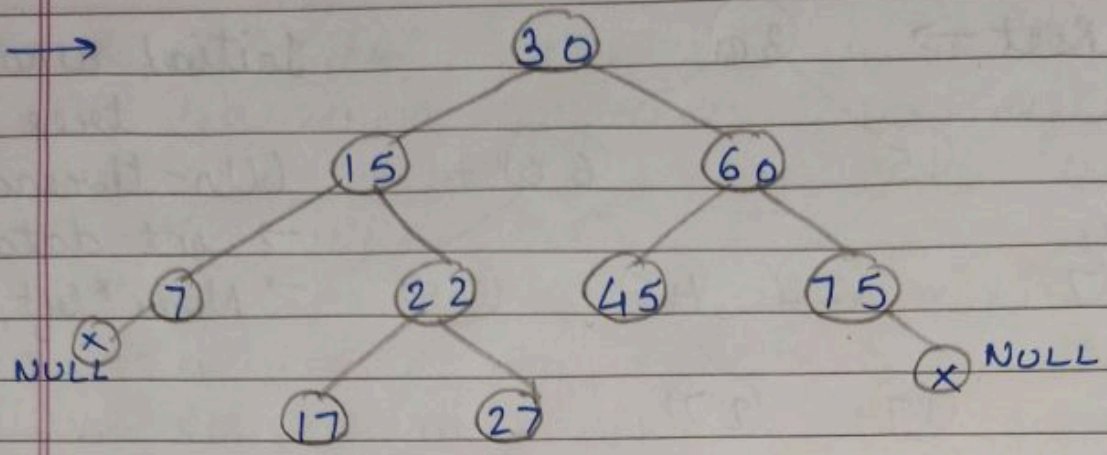
(IV) Now, if R-thread is true, then find the inorder successor and assign it to the right pointer field of the node

(V) If no inorder predecessor or successor is found the assign the field as NULL or connect it to dummy node.

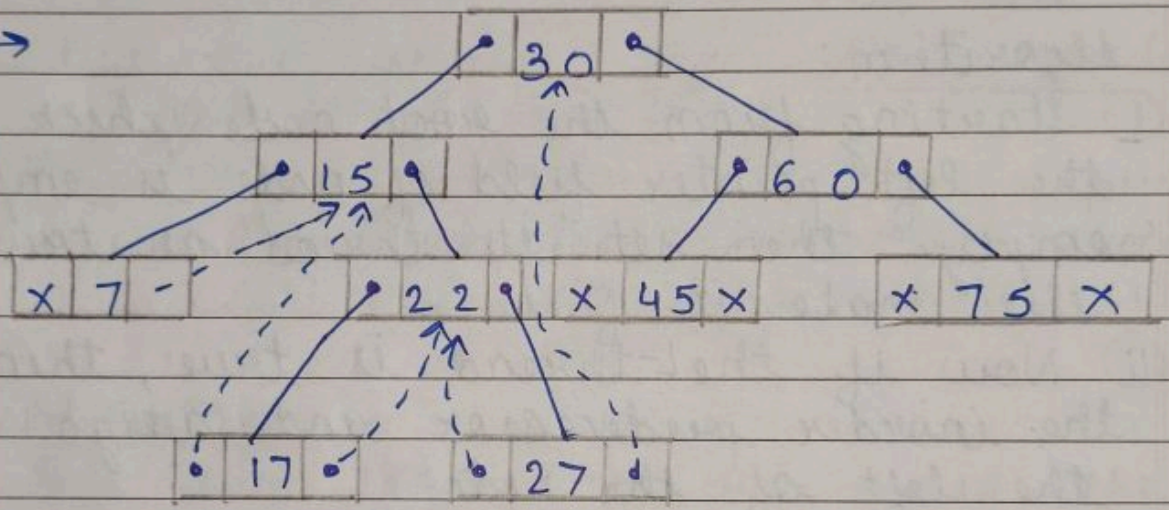
Initial Threaded Binary Tree

```
class Node {
        int data;
        Node * left, * right;
        bool L_thread ; R_thread ;
};
```

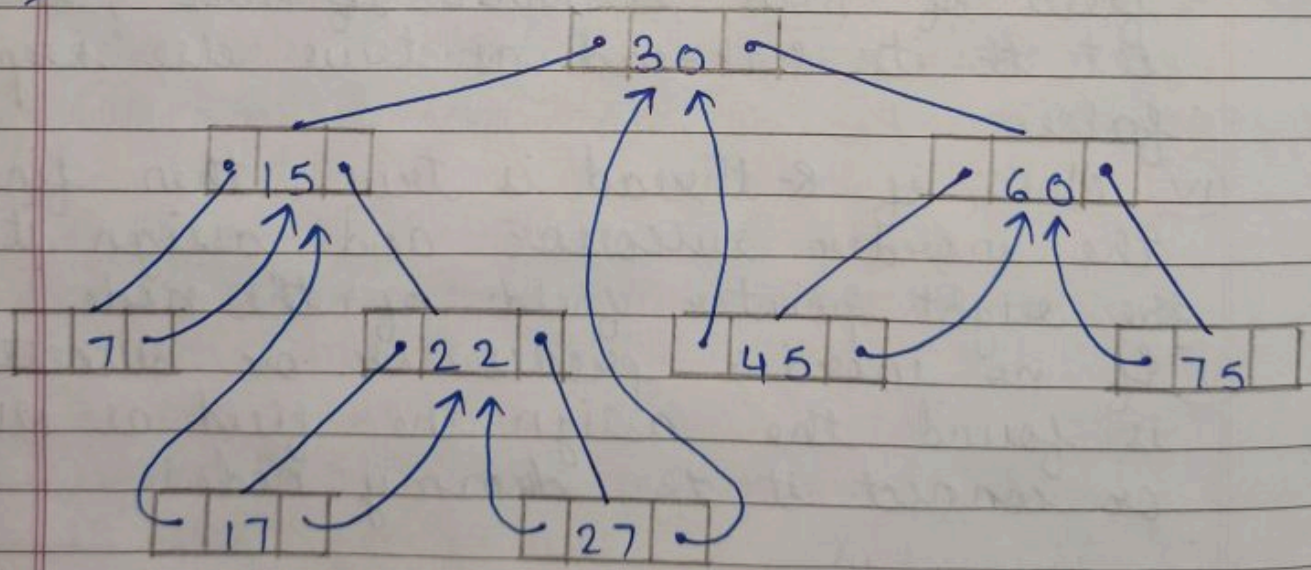
→



→



→



This is the final Threaded Binary tree

Q5. Algorithm for non-recursive preorder & postorder traversal of BST.

1 Algorithm for non-recursive preorder traversal → (BST)

To convert an inherently recursive procedure to interactive, we need an explicit stack.

① Create an empty stack and push root node to stack
② Do step 3 to 5, while stack is not empty
③ Pop an item from the stack and print it.
④ Push right child of ~~poped~~ popped item to stack
i.e. push (node → right)
⑤ Push left child of a popped item to stack
i.e. push (node → left)
⑥ Stop

(Right child is pushed first before left so that the left subtree is processed first)

2 Algorithm for non-recursive postorder traversal (BST) (~~One~~ Using one stack only)

① Create an empty stack.
② Do step 3 & 4 while root is not NULL
③ Push root's right child and the root to stack.
④ Set root as root's left child.
⑤ Pop an item from stack and set it as root.
⑥ If the popped item has a right child and the right child is at top

of stack, push the root back and set root as root's right child.
(7) else print root's data and set root as NULL.
(8) Repeat set step 2 and 5 while stack is not empty
(9) Return and stop.

Q6. Non-Recursive inorder & preorder traversal in TBT →

1 Algorithm for non-recursive inorder traversal (TBT) → (without stack)

(1) Start at the leftmost node and print it.
(2) Follow thread of right and print it.
(3) Follow link to right, go to leftmost node and print it
(4) Follow thread to right and print it.
(5) Repeat step 2 to 4 while thread to right is not NULL.

2 Algorithm for non-recursive preorder traversal (TBT) -

(1) Start at root node (i.e. current = root)
(2) Repeat step until current is not NULL →
    (1) Print current's data
    (2) If left child of node exists then, current = current → left
    (3) else if right child of current node exists then,

- current = current → right

(iv) else

- Until right thread exists for current node,
  - Traverse the right thread & update current.
- If current == last node i.e. if right thread does not exists (NULL) the stop.
- else current = current → right

(3) stop

Thus, above given are the non-recursive inorder and preorder traversal without stack (Iterative traversal).