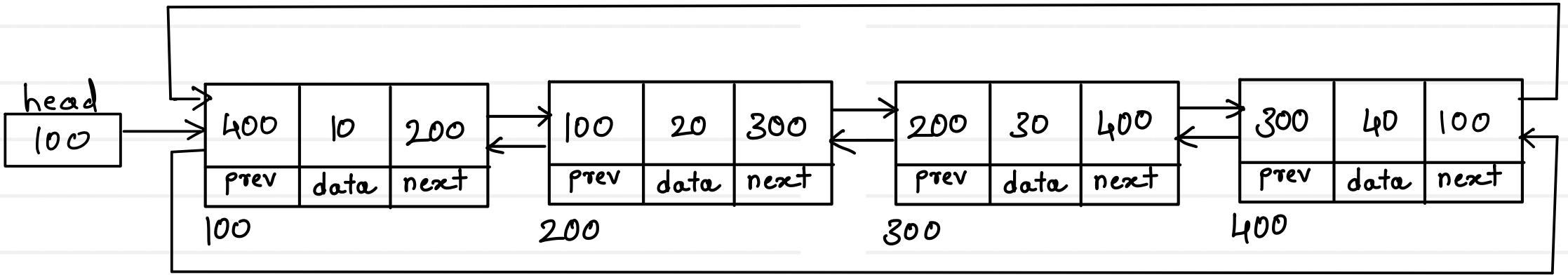
Sunbeam Institute of Information Technology
Pune and Karad

Module – Data Structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Doubly Circular Linked List - Display

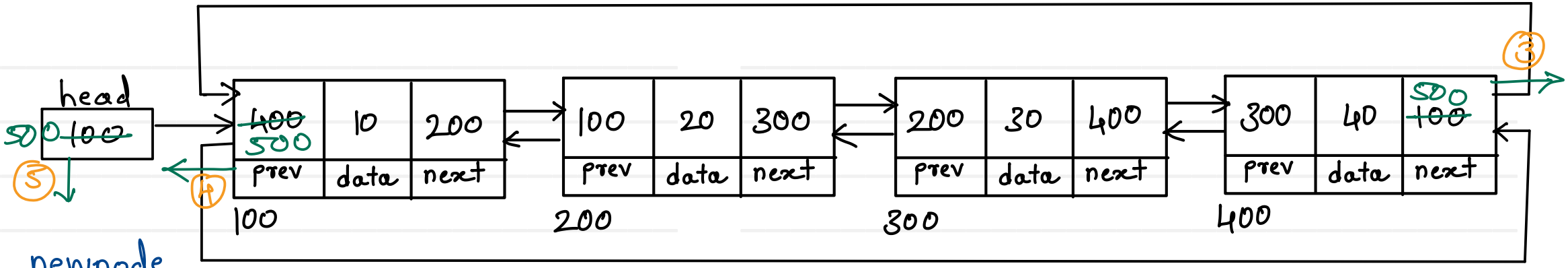


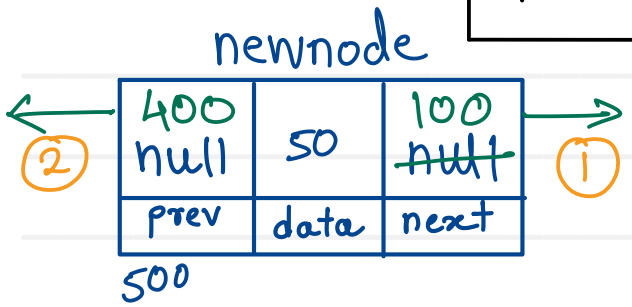
1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeat step 2 & 3 till last node

1. Create trav & start at last node
2. print current node
3. go on prev node
4. repeat step 2 & 3 till first node

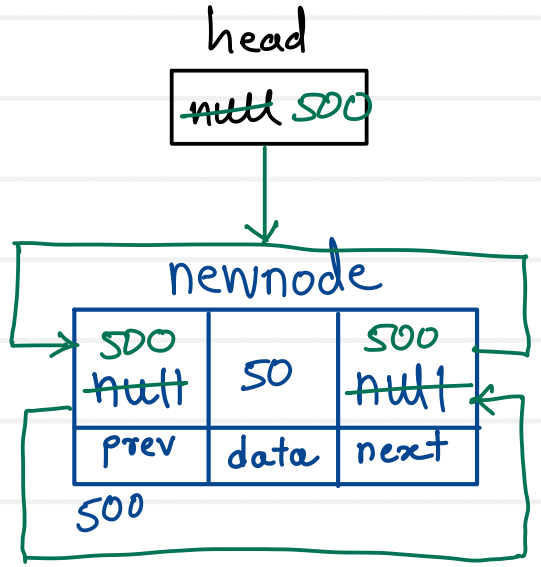
$$T(n) = O(n)$$

Doubly Circular Linked List - Add first

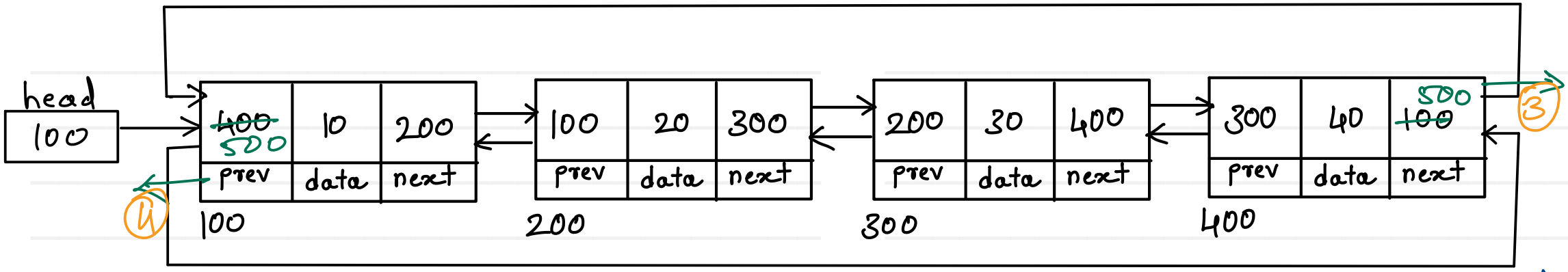




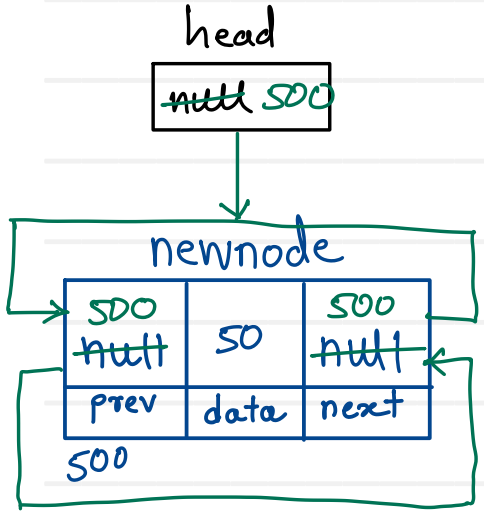
a. Create node
b. if list is empty
   1. add newnode into head
   2. make list circular
c. if list is not empty
   1. add first node into next of new node
   2. add last node into prev of new node
   3. add newnode into next of last node
   4. add newnode into prev of first node
   5. move head on newnode

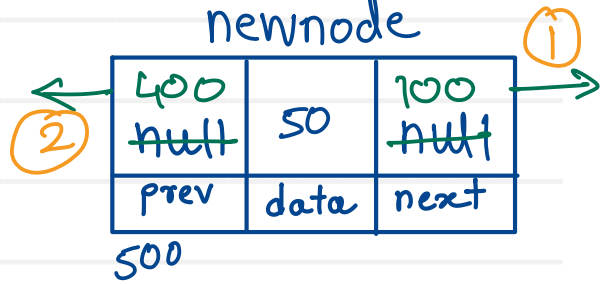


Doubly Circular Linked List - Add last



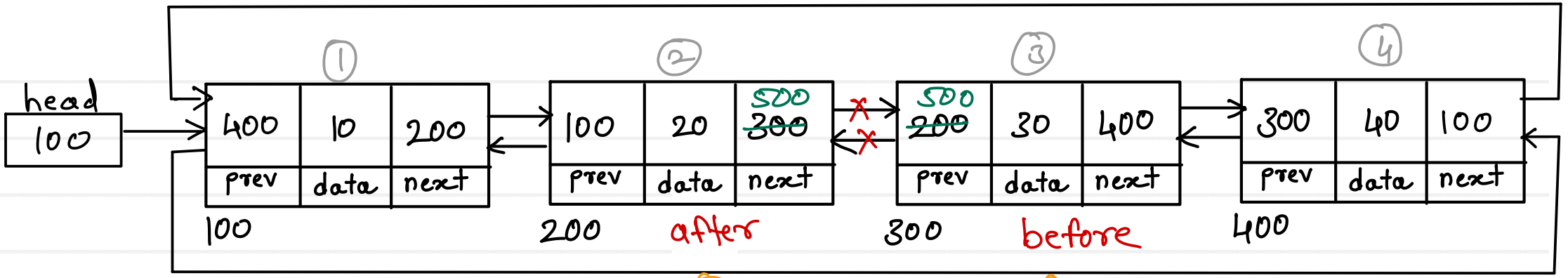
a. Create node
b. if list is empty
    1. add newnode into head
    2. make list circular
c. if list is not empty
    1. add first node into next of newnode
    2. add last node into prev of newnode
    3. add newnode into next of last node
    4. add newnode into prev of first node

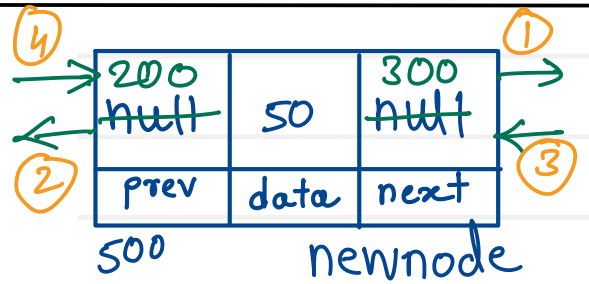




T(n) = O(1)

Doubly Circular Linked List - Add position

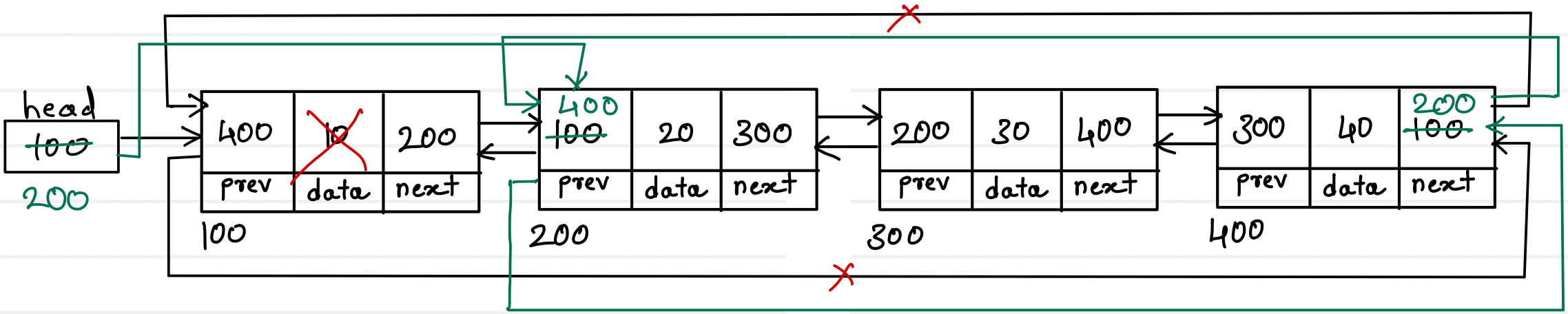





```
newnode.next = before;
newnode.prev = after;
before.prev = newnode;
after.next = newnode;
```

$T(n) = O(n)$

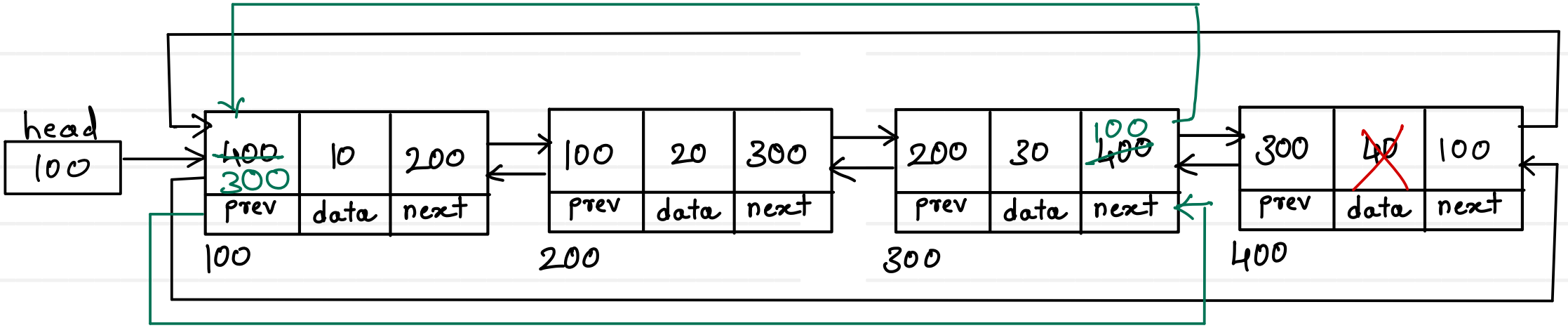
Doubly Circular Linked List - Delete first



1. if list is empty, return
2. if list has single node
   head = null.
3. if list has multiple nodes
   a. add second node into next of last node
   b. add last node into prev of second node
   c. move head on second node.

$T(n) = O(1)$

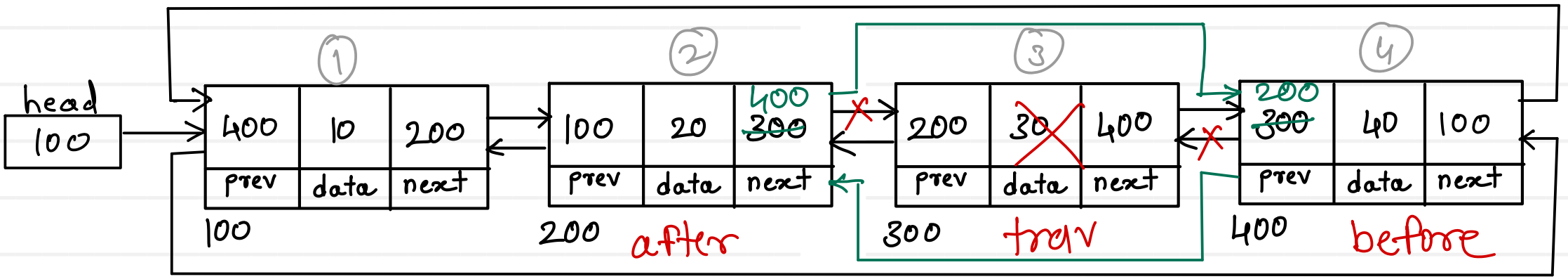
Doubly Circular Linked List - Delete last



1. if list is empty, return
2. if list has single node
   head = null;
3. if list has multiple nodes
   a. add first node into next of second last node
   b. add second last node into prev of first node

$T(n) = O(1)$

Doubly Circular Linked List - Delete position

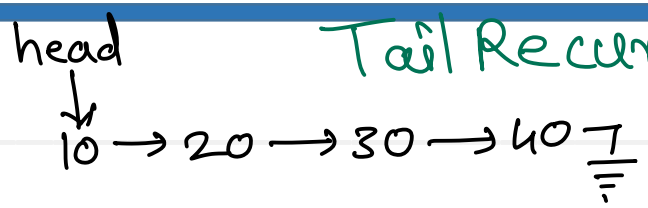



```
after = trav.prev;
before = trav.next
after.next = before
before.prev = after
```

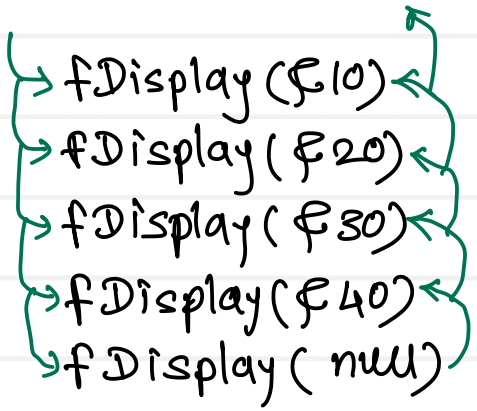
$T(n) = O(n)$

Singly linear linked list - Display reverse

Tail Recursion

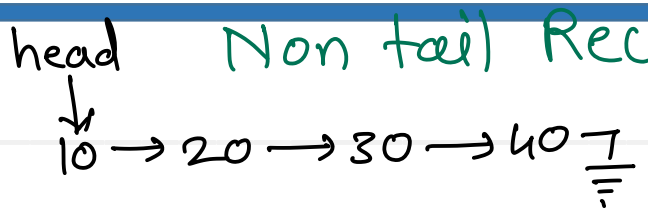



```
void fDisplay( Node trav ) {
    if( trav == null)
        return;
    sysout(trav.data);
    fDisplay( trav. next);
}
```

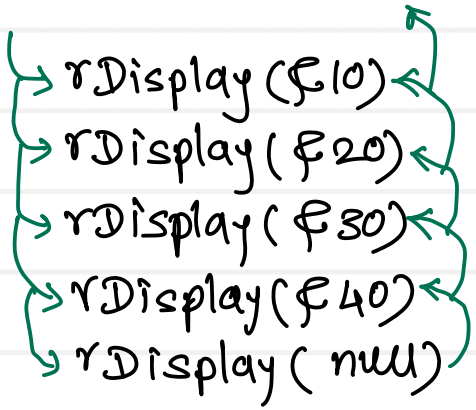


10 20 30 40

Non tail Recursion

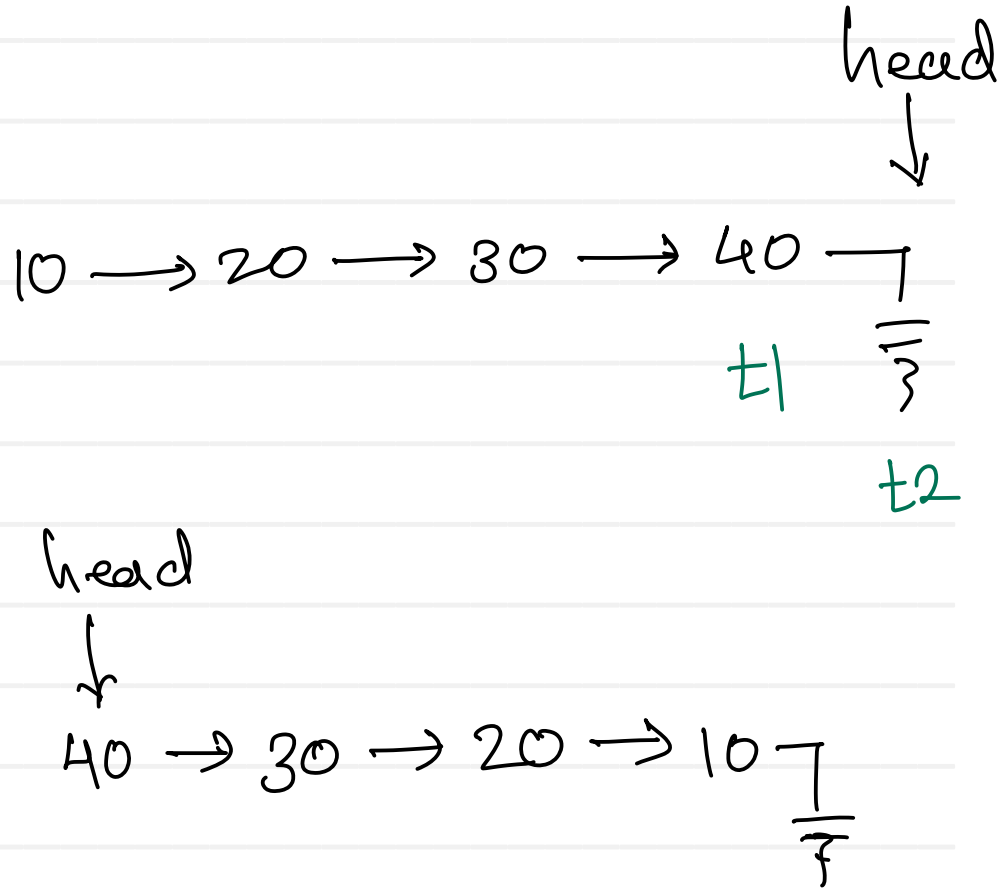



```
void rDisplay( Node trav ) {
    if( trav == null)
        return;
    rDisplay( trav. next);
    sysout(trav.data);
}
```



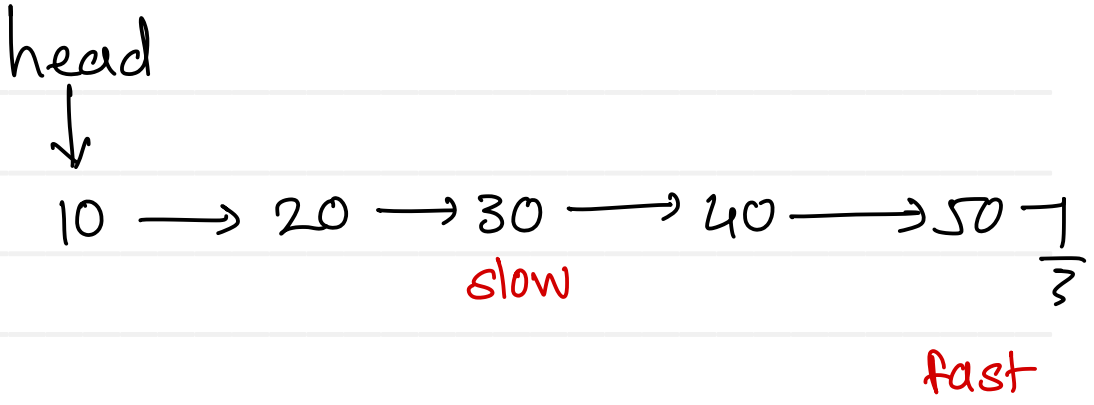
40 30 20 10

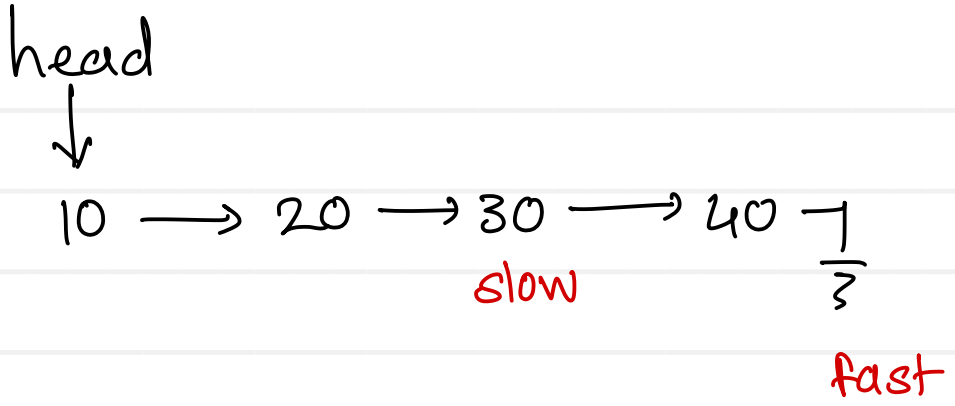
Singly linear linked list - Reverse




```
Node t1 = null;
Node t2 = head;
while (t2 != null) {
    head = t2.next;
    t2.next = t1;
    t1 = t2;
    t2 = head;
}
head = t1;
```

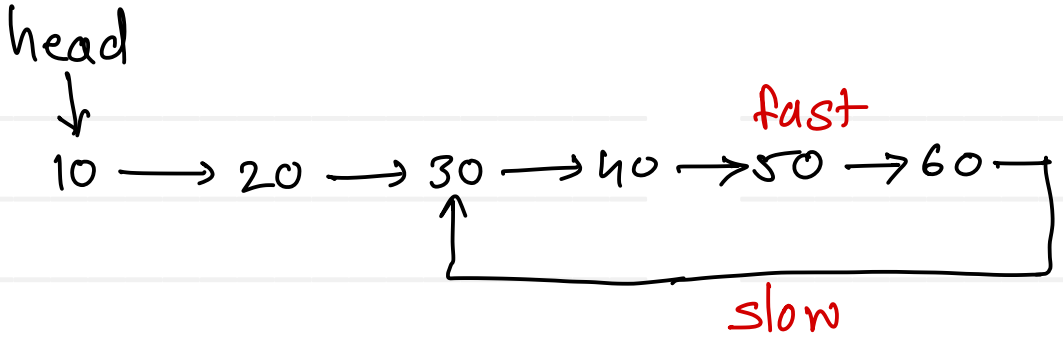
Singly linear linked list - Find mid






```
void findMid( ) {
    Node fast = head, slow = head;
    while( fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    sysout( slow.data);
}
```

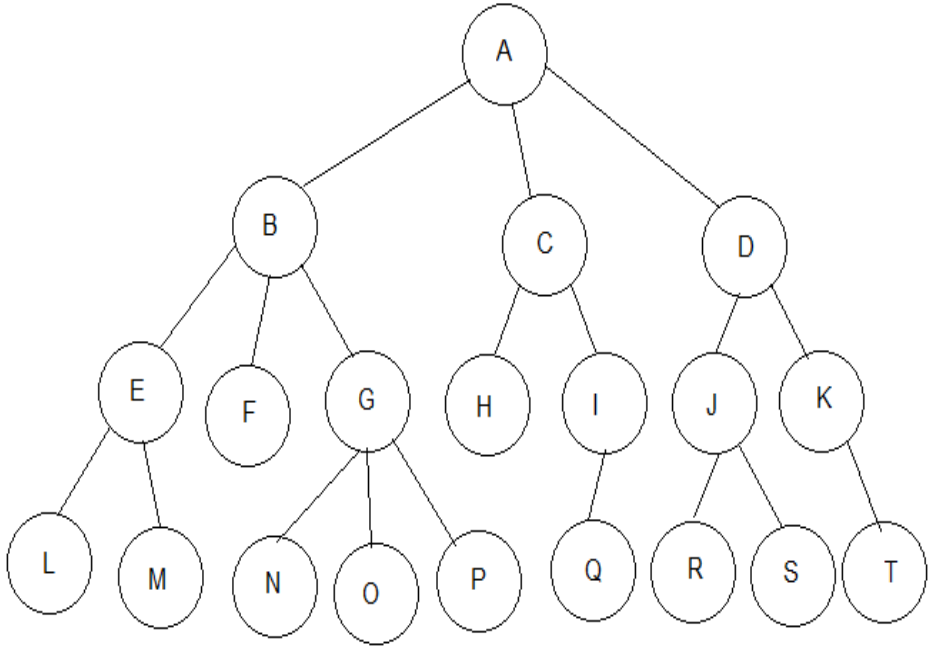
Singly linear linked list - Detect loop




```
boolean hasLoop( ) {
    Node fast = head, slow = head;
    while( fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
        if( fast == slow)
            return true;
    }
    return false;
}
```

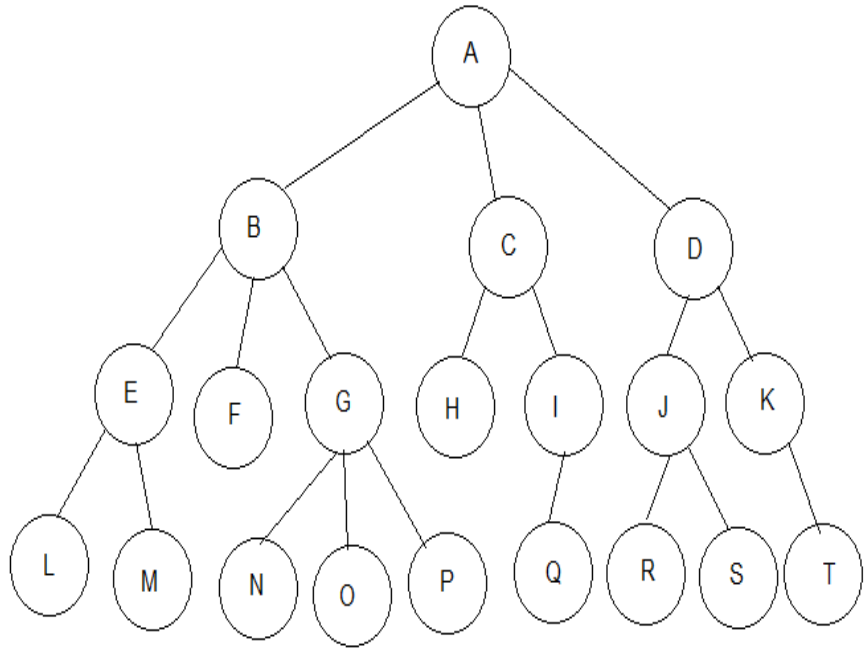
Tree - Terminologies

- **Tree** is a **non linear** data structure which is a finite set of nodes with one specially designated node is called as "**root**" and remaining nodes are partitioned into m disjoint subsets where each of subset is a tree..
- **Root** is a **starting point** of the tree.
- All nodes are connected in **Hierarchical manner (multiple levels)**.
- **Parent node:-** having other child nodes connected
- **Child node:-** immediate descendant of a node
- **Leaf node:-**
  - Terminal node of the tree.
  - Leaf node does not have child nodes.
- **Ancestors:-** all nodes in the path from root to that node.
- **Descendants:-** all nodes accessible from the given node
- **Siblings:-** child nodes of the same parent



Tree - Terminologies

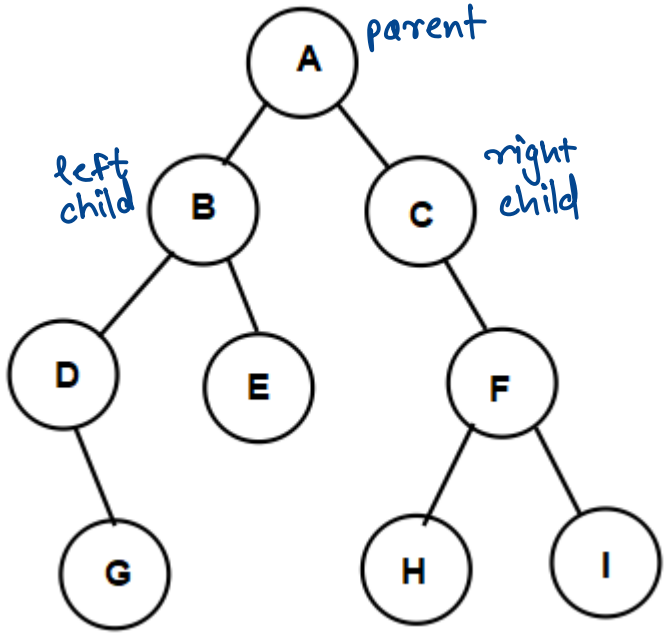
- **Degree of a node :-** number of child nodes for any given node.
- **Degree of a tree :-** Maximum degree of any node in tree.
- **Level of a node :-** indicates position of the node in tree hierarchy
  - Level of child = Level of parent + 1
  - Level of root = 0
- **Height of node :-** number of links from node to longest leaf.
- **Depth of node :-** number of links from root to that node
- **Height of a tree :-** Maximum height of a node
- **Depth of a tree :-** Maximum depth of a node



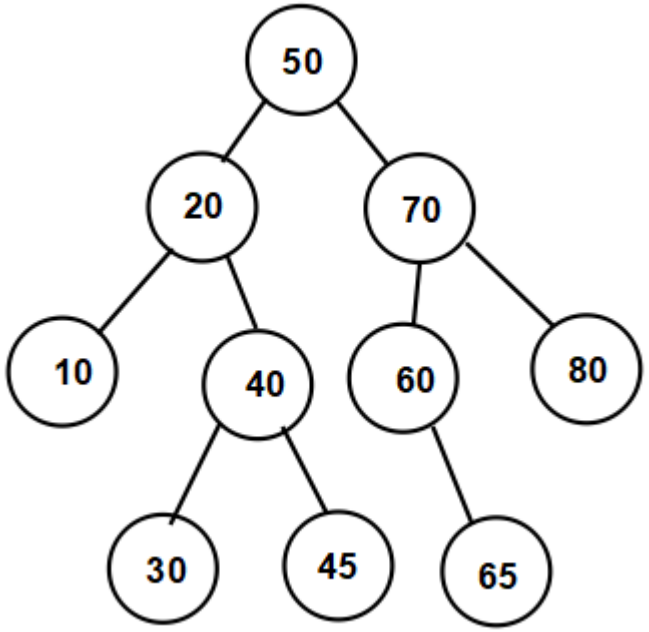
- Tree with zero nodes (ie empty tree) is called as "**Null tree**". Height of Null tree is -1.
  - Tree can grow up to any level and any node can have any number of Childs.
  - That's why operations on tree becomes un efficient.
  - Restrictions can be applied on it to achieve efficiency and hence there are different types of trees.

Tree - Terminologies

- **Binary Tree**
  - Tree in which each node has maximum two child nodes
  - Binary tree has degree 2. Hence it is also called as 2- tree



- **Binary Search Tree**
  - Binary tree in which left child node is always smaller and right child node is always greater or equal to the parent node.
  - Searching is faster
  - Time complexity : O(h) h – height of tree



Binary Search Tree - Implementation

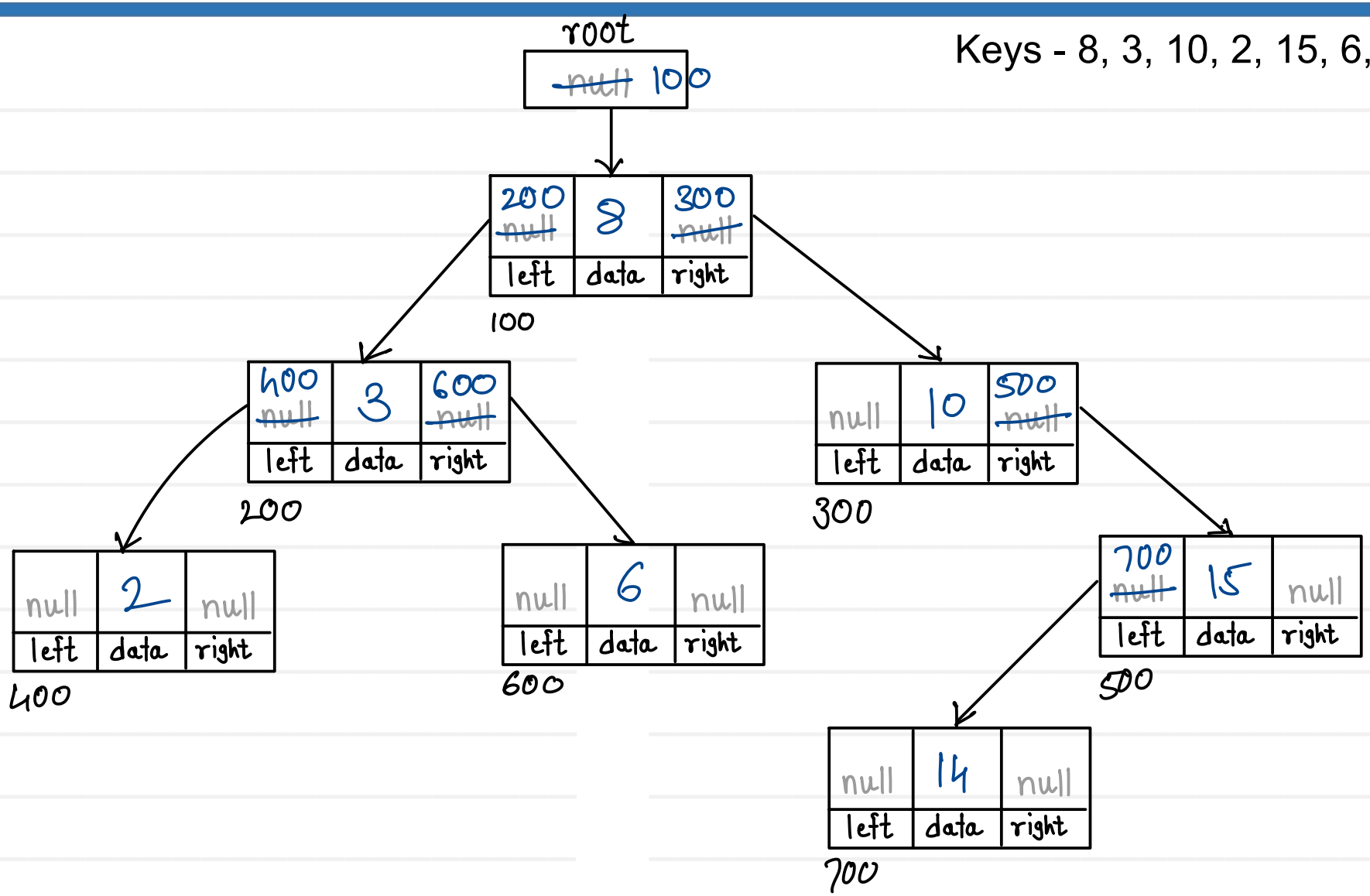
Node :
- data →
- left → referance
- right → referance

```
class Node {
    int data;
    Node left;
    Node right;
}
```

```
class BST {
    static class Node {
        int data;
        Node left;
        Node righ;
    }
    Node root;
    public BST( ) {...}
    public add( ) {...}
    public delete( ) {...}
    public search( ) {...}
    public traverse( ) {...}
}
```

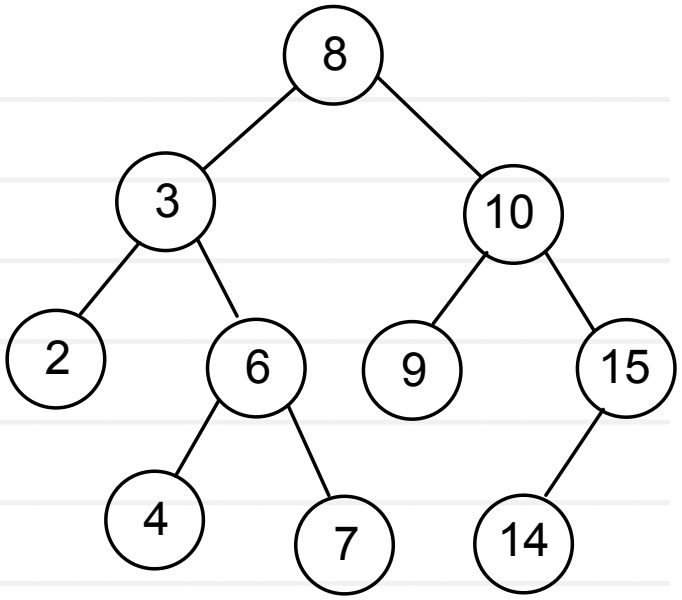
Binary Search Tree - Add Node

Keys - 8, 3, 10, 2, 15, 6, 14, 4, 7, 9

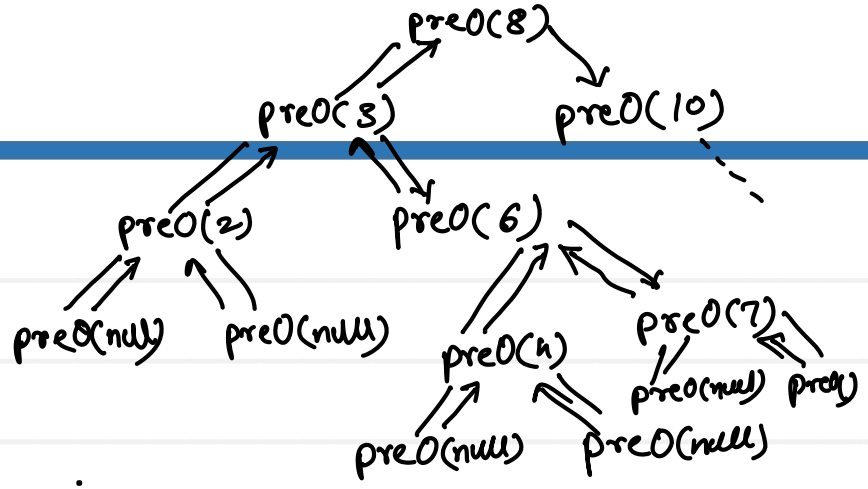


Binary Search Tree – Add Node

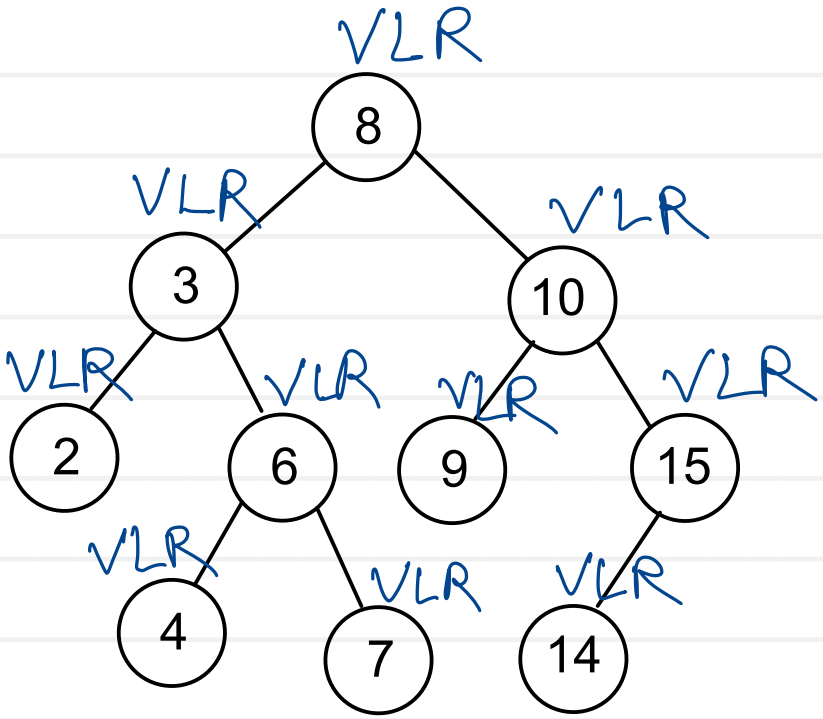
```
//1. create node for given value
//2. if BSTree is empty
      // add newnode into root itself
//3. if BSTree is not empty
      //3.1 create trav reference and start at root node
      //3.2 if value is less than current node data (trav.data)
            //3.2.1 if left of current node is empty
                  // add newnode into left of current node
            //3.2.2 if left of current node is not empty
                  // go into left of current node
      //3.3 if value is greater or equal than current node data (trav.data)
            //3.3.1 if right of current node is empty
                  // add newnode into right of current node
            //3.3.2 if right of current node is not empty
                  // go into right of current node
      //3.4 repeat step 3.2 and 3.3 till node is not getting added into BSTree
```



Tree Traversal Techniques



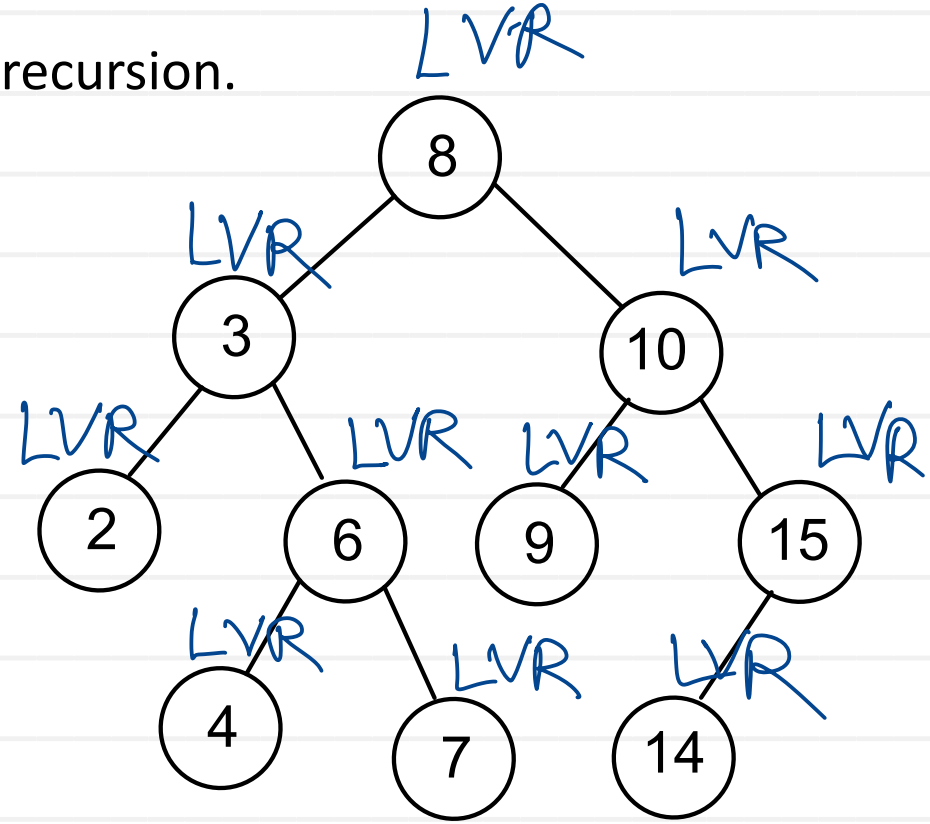
- **Pre-Order:-** V L R
- **In-order:-** L V R
- **Post-Order:-** L R V
- The traversal algorithms can be implemented easily using recursion.
- Non-recursive algorithms for implementing traversal needs stack to store node pointers.



- **Pre-Order :-** 8, 3, 2, 6, 4, 7, 10, 9, 15, 14

Tree Traversal Techniques

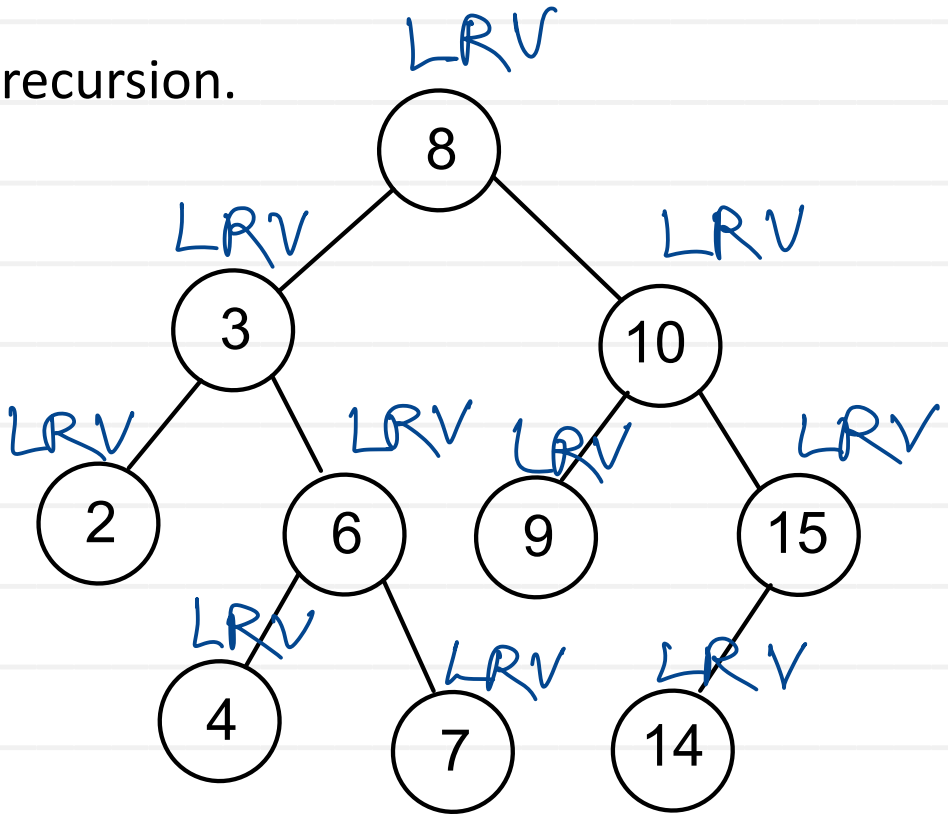
- **Pre-Order:-** V L R
- **In-order:-** L V R
- **Post-Order:-** L R V
- The traversal algorithms can be implemented easily using recursion.
- Non-recursive algorithms for implementing traversal needs stack to store node pointers.



- **In-Order :-** 2, 3, 4, 6, 7, 8, 9, 10, 14, 15

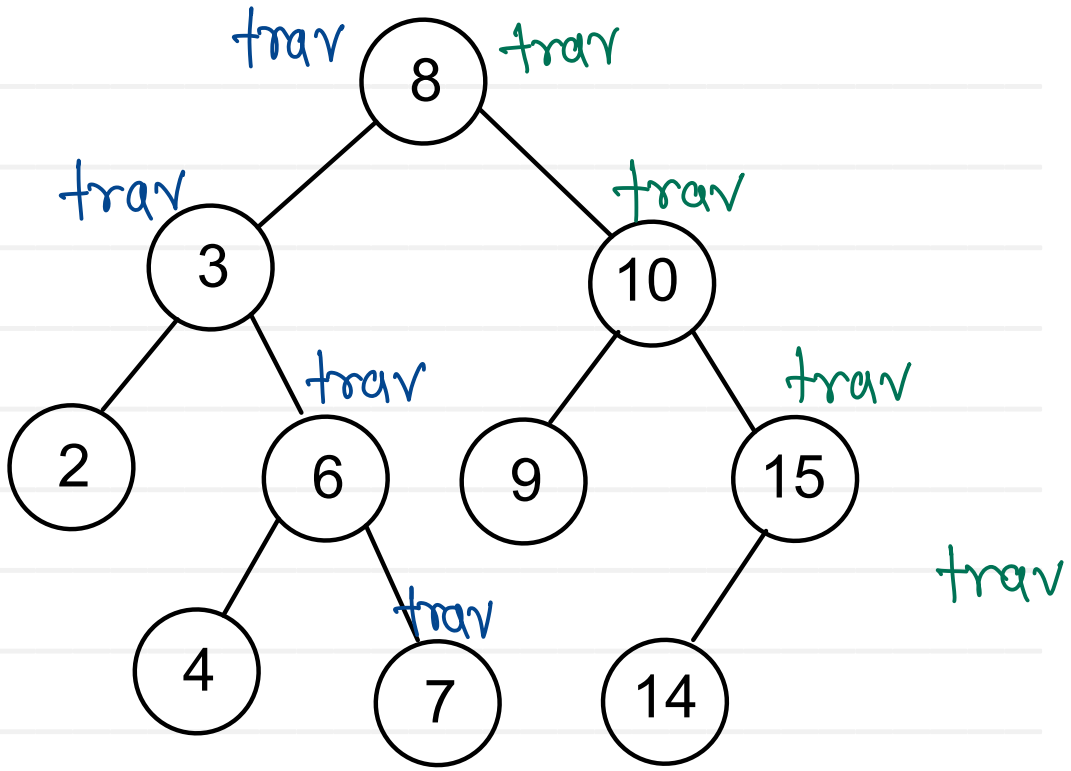
Tree Traversal Techniques

- **Pre-Order:-** V L R
- **In-order:-** L V R
- **Post-Order:-** L R V
- The traversal algorithms can be implemented easily using recursion.
- Non-recursive algorithms for implementing traversal needs stack to store node pointers.



- **Post-Order :-** 2, 4, 7, 6, 3, 9, 14, 15, 10, 8

Binary Search Tree - Binary Search



1. Start from root
2. If key is equal to current node data return current node
3. If key is less than current node data search key into left sub tree of current node
4. If key is greater than current node data search key into right sub tree of current node
5. Repeat step 2 to 4 till leaf node

key = 7 – key is found
key = 16 – key is not found

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com