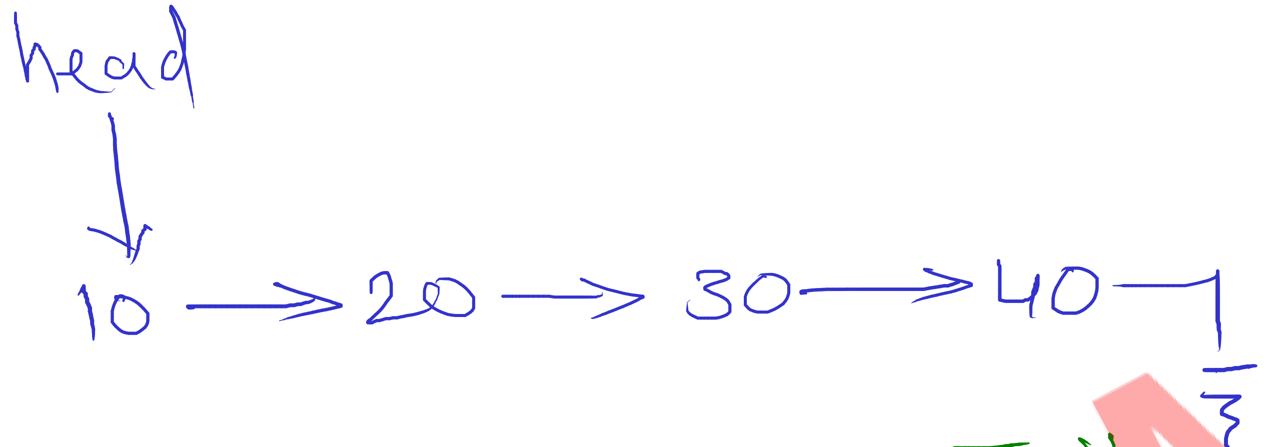
Singly Linear Linked List - Reverse Display



Tail Recursion

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

```
fDisplay( 10)
fDisplay (20)
fDisplay (30)
fDisplay(40)
fDisplay(null)
```

10 20 30 40

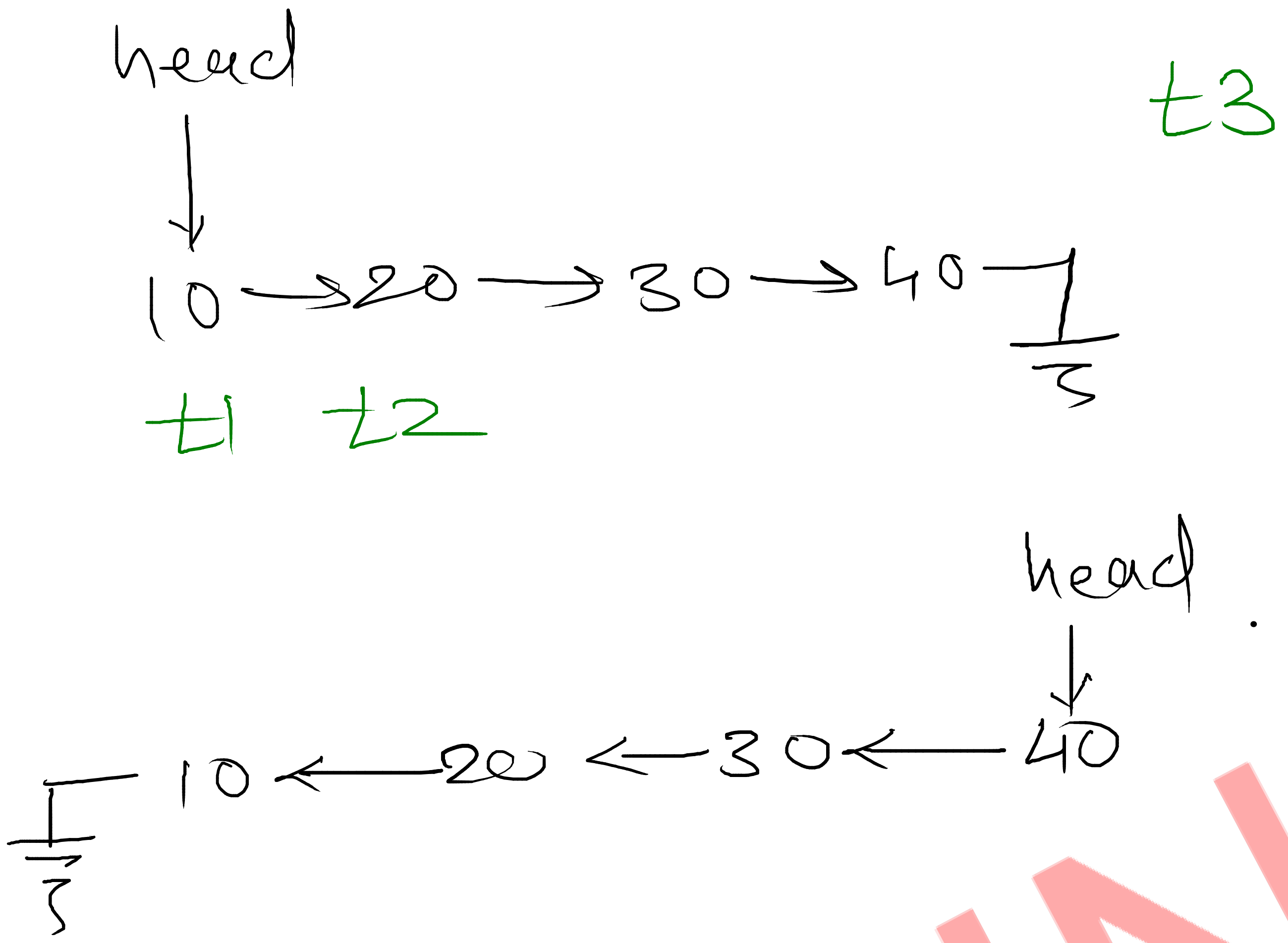
Non Tail recursion

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

```
rDisplay( 10)
rDisplay (20)
rDisplay(30)
rDisplay(40)
rDisplay(null)
```

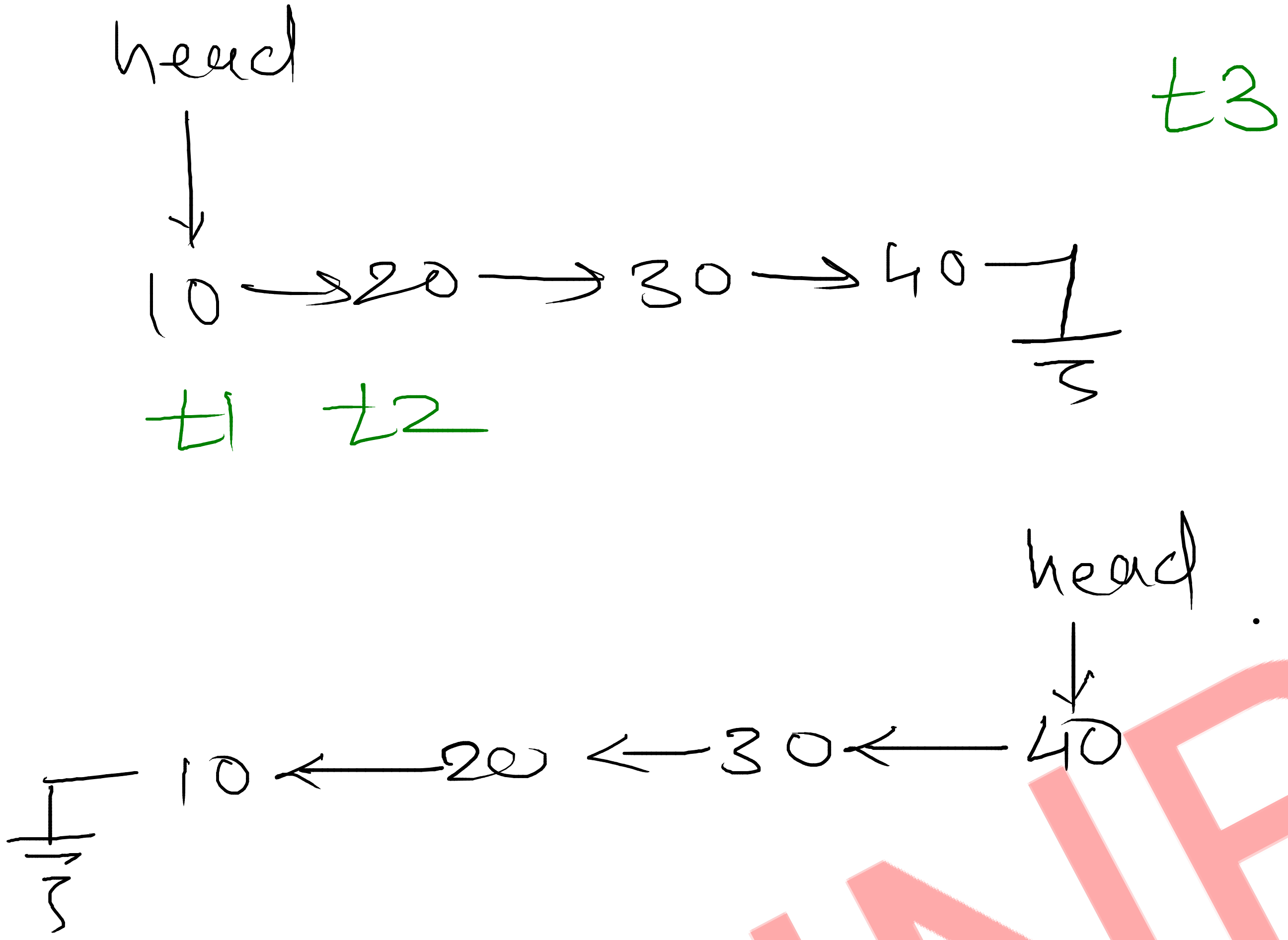
40 30 20 10

Singly Linear Linked List - Reverse List



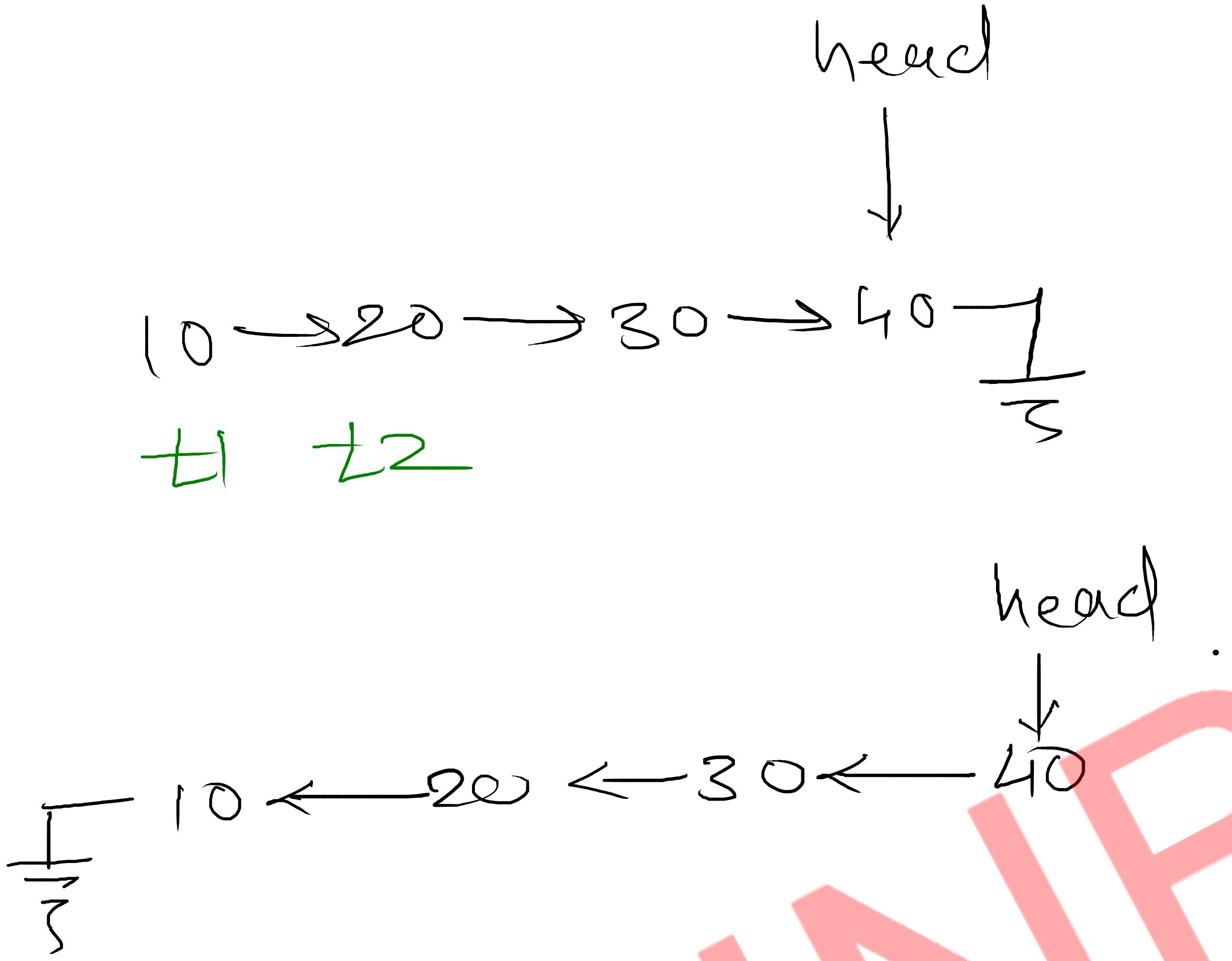
```
Node t1 = head;
Node t2 = head.next;
while(t2 != null) {
    t3 = t2.next;
    t2.next = t1;
    t1 = t2;
    t2 = t3;
}
head.next = null;
head = t1;
```

Singly Linear Linked List - Reverse List



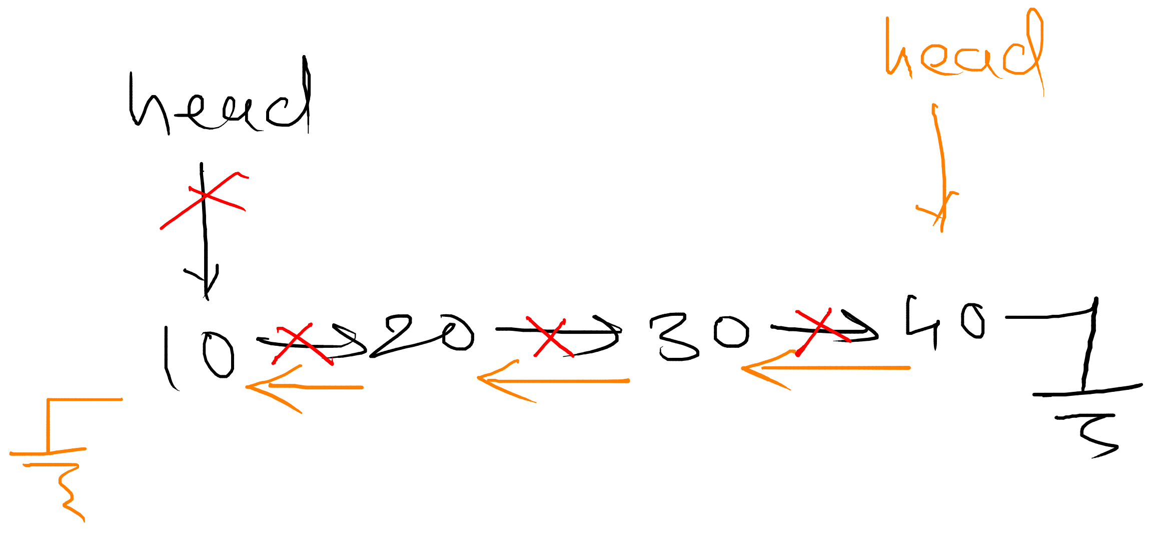
```
Node t1 = null;
Node t2 = head;
while(t2 != null) {
    t3 = t2.next;
    t2.next = t1;
    t1 = t2;
    t2 = t3;
}
head = t1;
```

Singly Linear Linked List - Reverse List



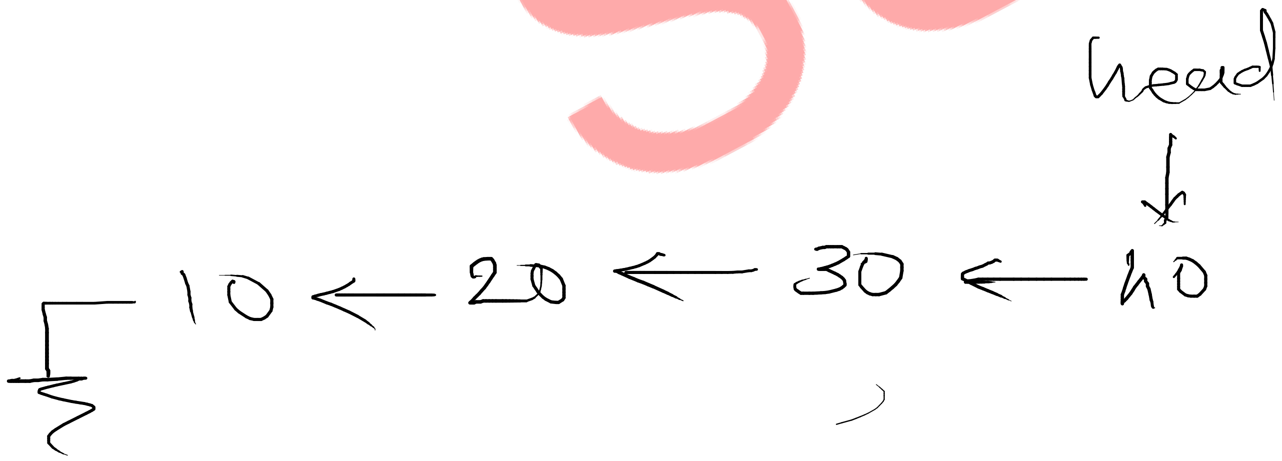
```
Node t1 = null;
Node t2 = head;
while(t2 != null) {
    head = t2.next;
    t2.next = t1;
    t1 = t2;
    t2 = head;
}
head = t1;
```

Singly Linear Linked List - Reverse List



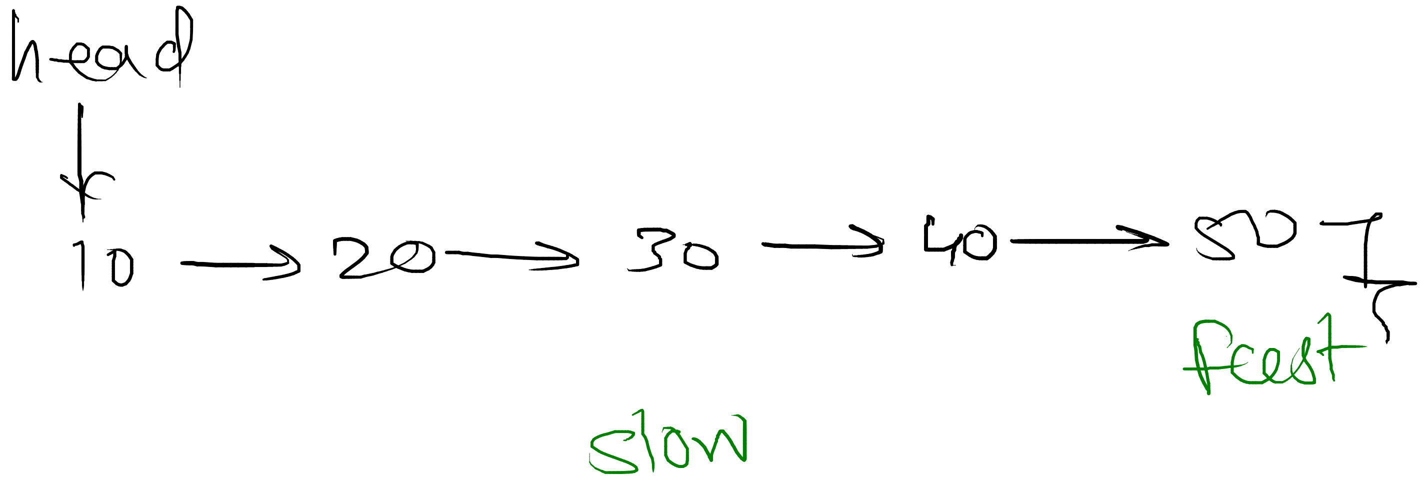
```
Node recReverse(Node trav) {
    if (trav.next == null){
        head = trav;
        return trav;
    }
    last = recReverse(trav.next);
    last.next = trav;
    trav.next = null;
    return trav;
}
```

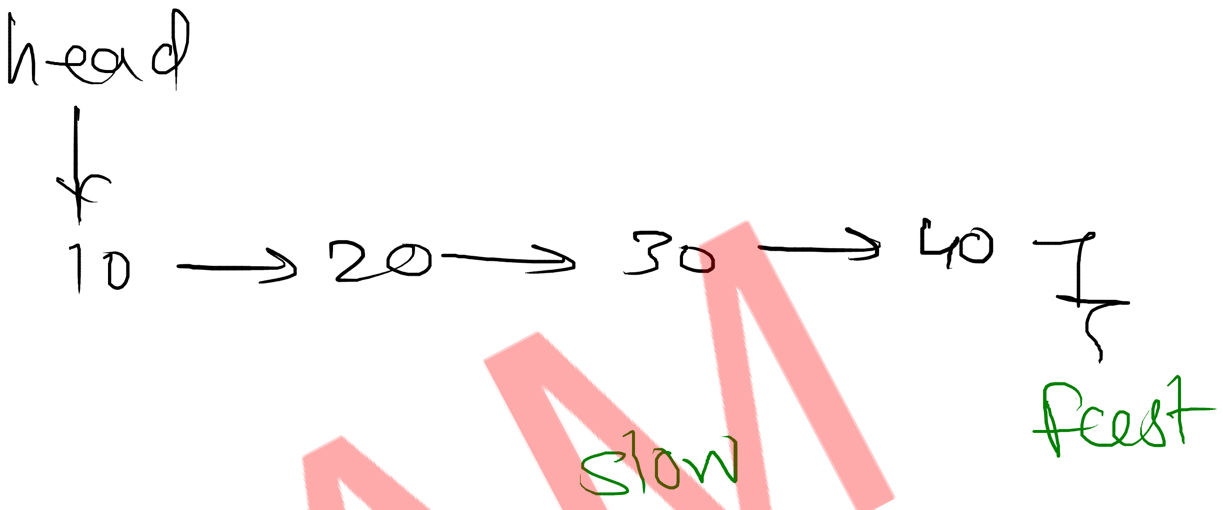
	trav	last
recReverse(10)	&10	&20
recReverse(20)	&20	&30
recReverse(30)	&30	&40
recReverse(40)	&40	



trav	last
&10	&20
&20	&30
&30	&40
&40	

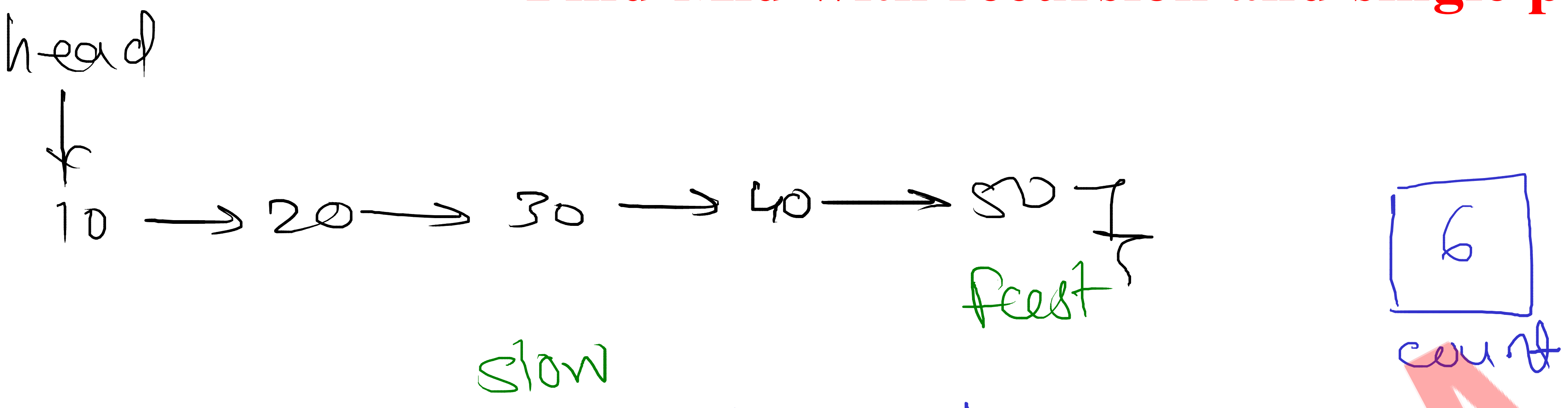
Singly Linear Linked List - Find mid





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

Find Mid with recursion and single pointer



```
findMid (Node trav, int index){
    if(trav == null {
        count = index
        return;
    }
    findMid (trav.next, index+1);
    if(index == count/2)
        sysout(trav.data);
}

findMid(head, 1)
```

{10, 1

{20, 2

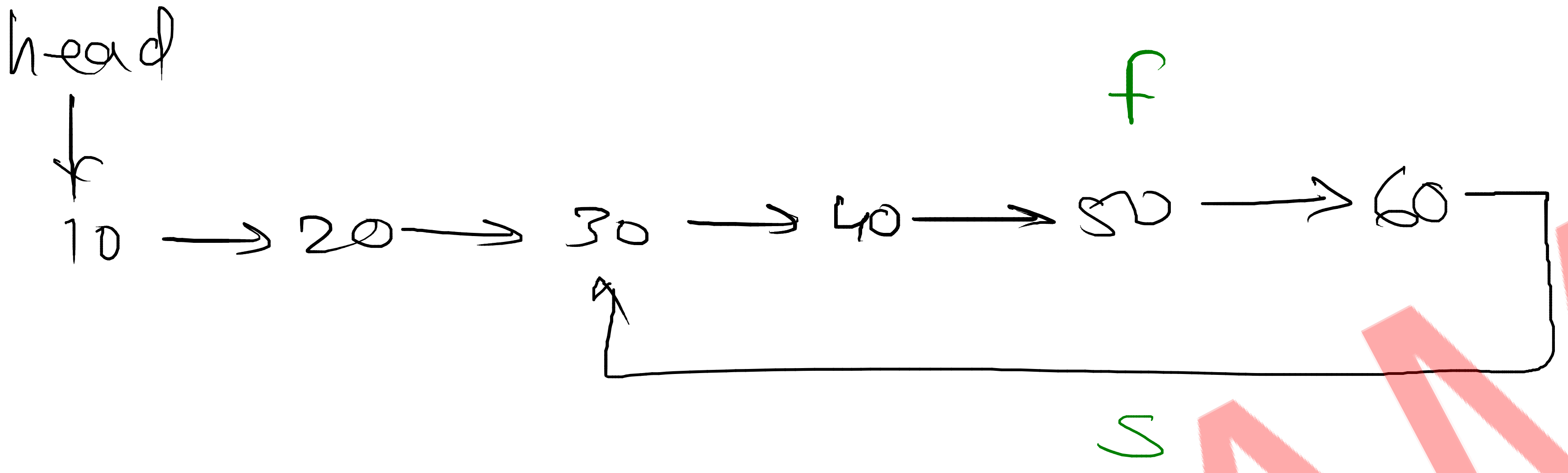
{30, 3

{40, 4

{50, 5

null, 6

Detect loop inside linked list



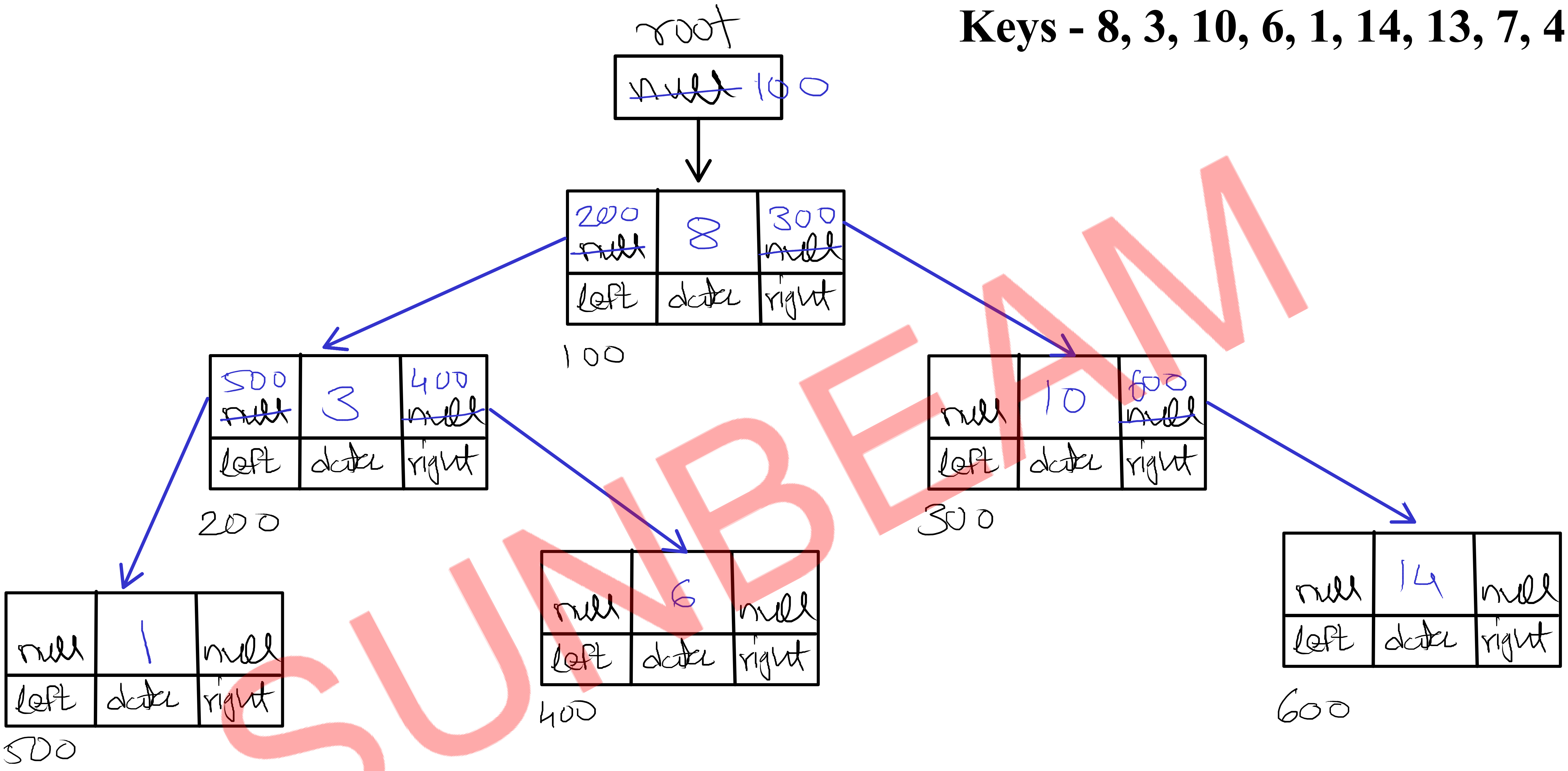
```
Node slow =head, fast=head;
while( fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
        if(fast == slow)
            return true;
}
return false;
```

BST - Add Node

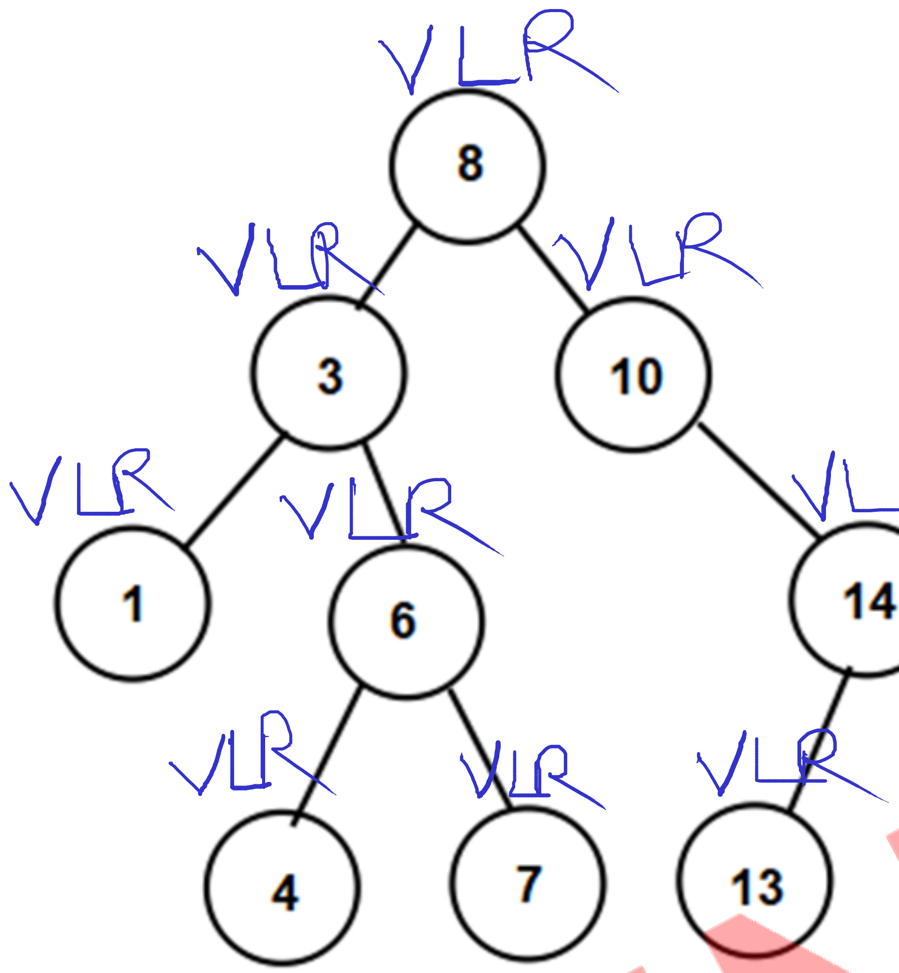
```
//1. create node for given value
//2. if BSTree is empty
    // add newnode into root itself
//3. if BSTree is not empty
    //3.1 create trav reference and start at root node
    //3.2 if value is less than current node data (trav.data)
        //3.2.1 if left of current node is empty
            // 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
```

BST - Add Node

Keys - 8, 3, 10, 6, 1, 14, 13, 7, 4



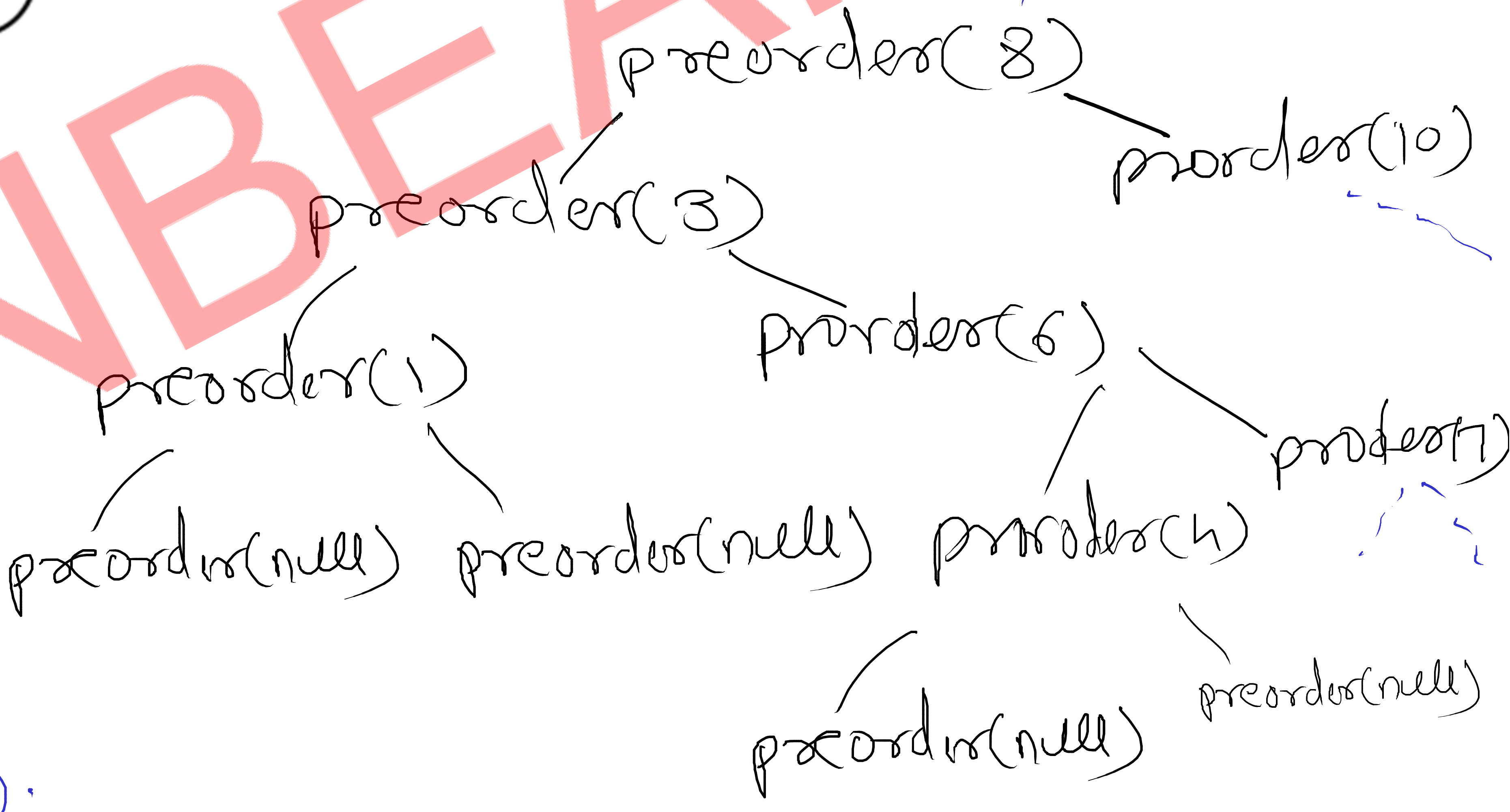
BST - Preorder Traversal



VLR

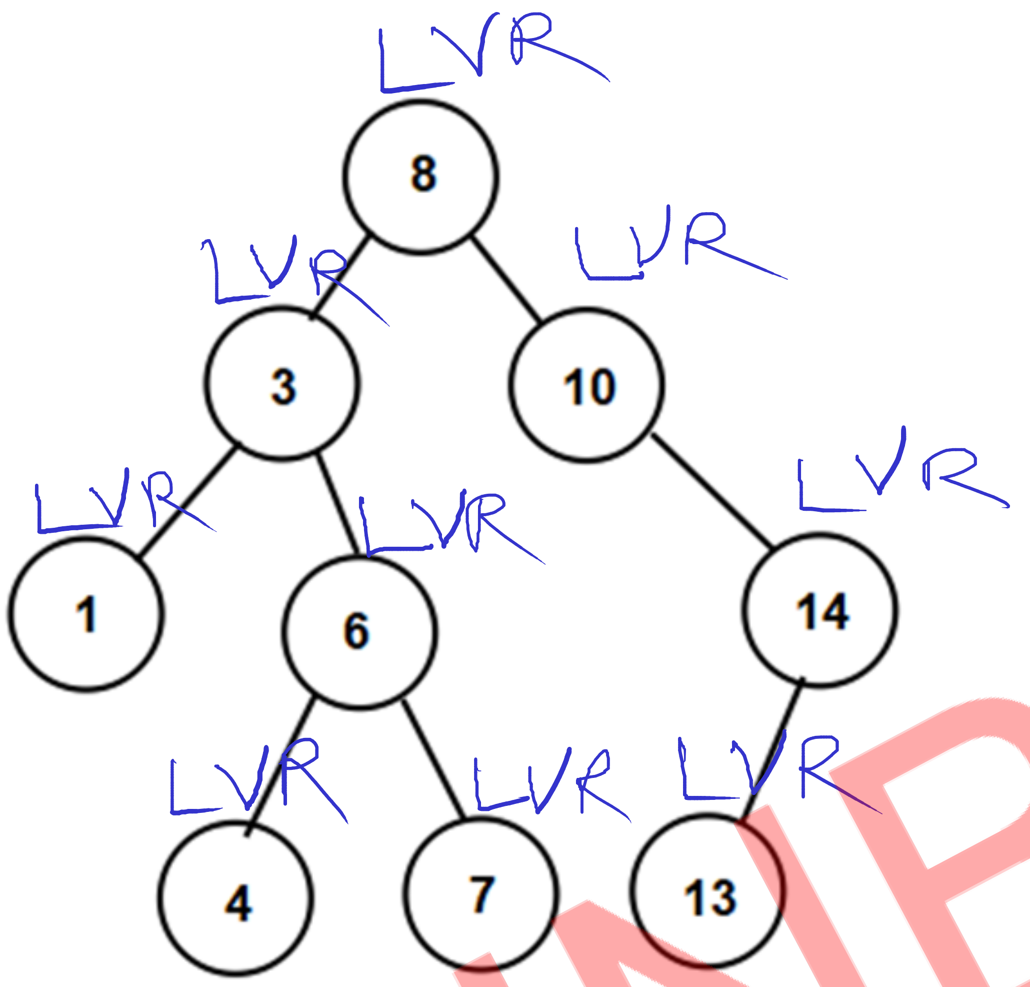
Traversal:

8, 3, 1, 6, 4, 7, 10, 14, 13



```
preorder(trav){
    if(trav==null)
        return;
    sysout(trav.data);
    preorder(trav.left);
    preorder(trav.right);
}
```

BST - Inorder Traversal

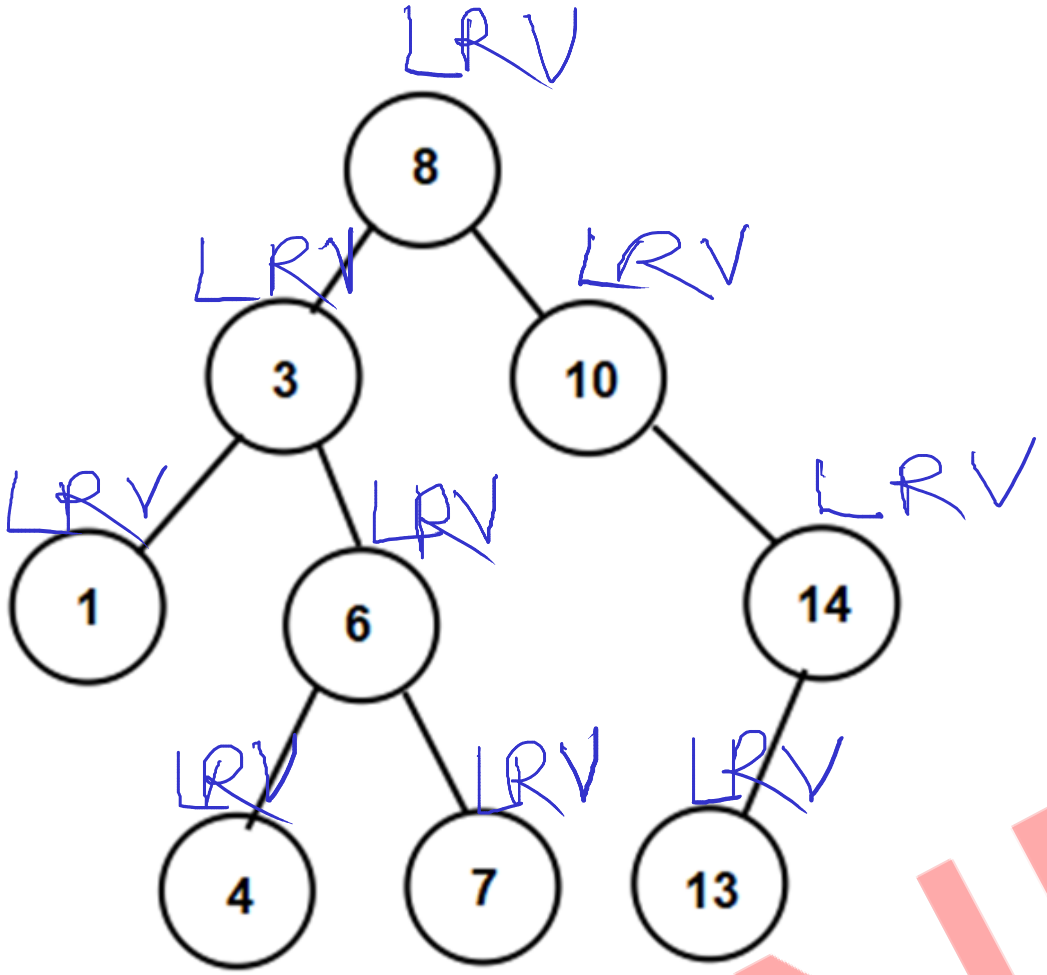


LVR

Traversal :

1, 3, 4, 6, 7, 8, 10, 13, 14

BST - Postorder Traversal

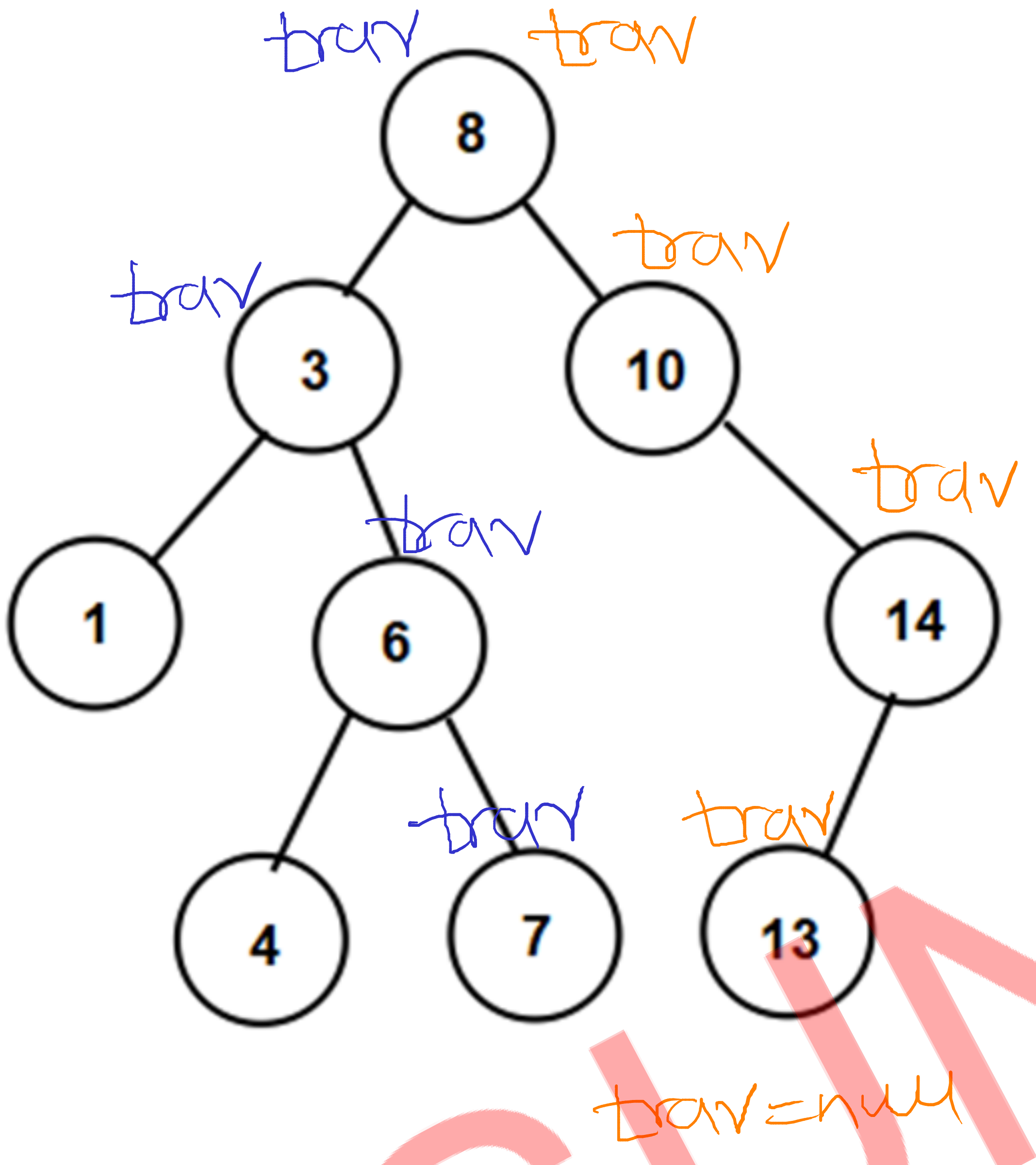


LRV

Traversal

1,4,7,6,3,13,14,10,8

BST - Binary Search



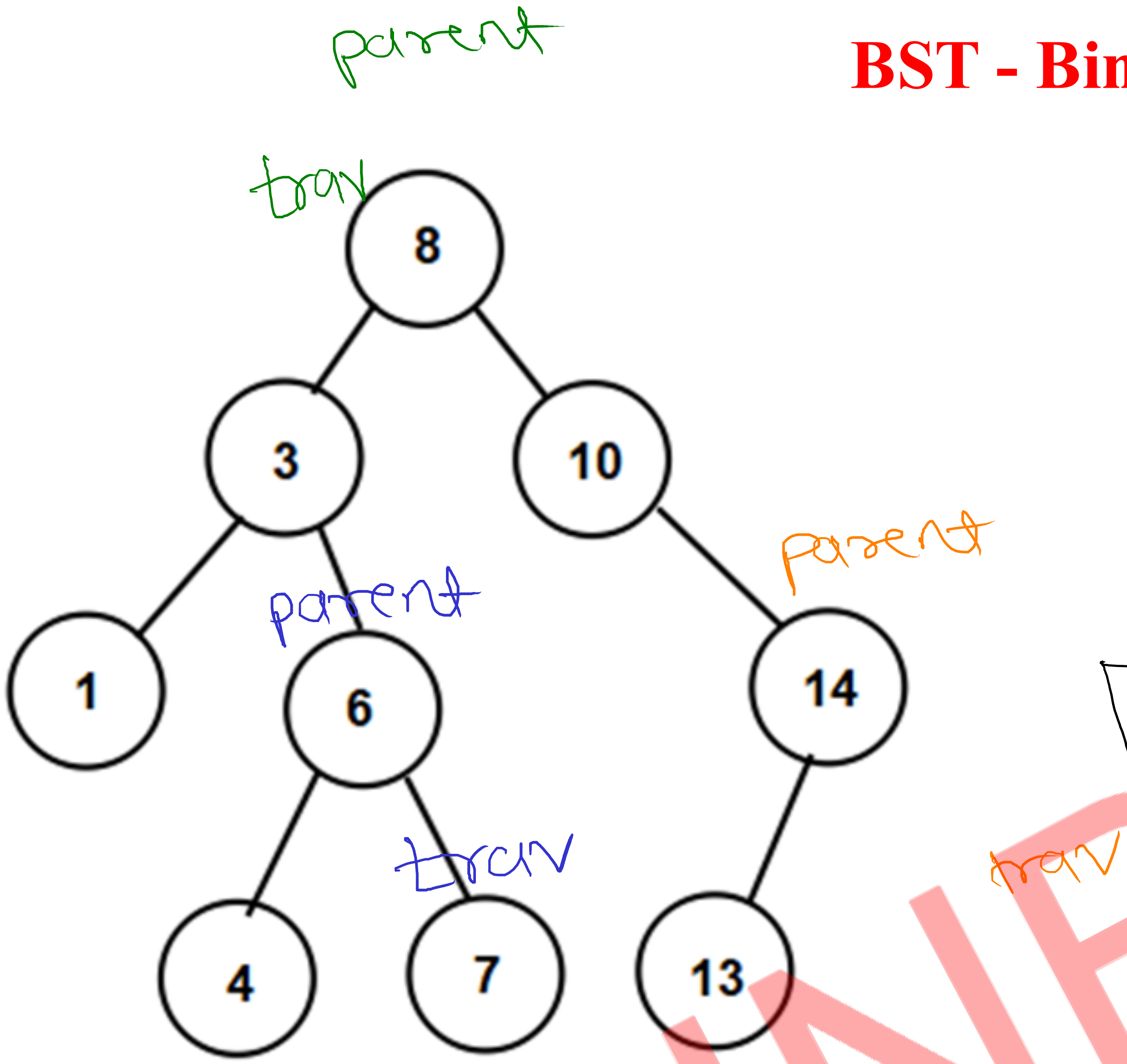
//1. start from root
//2. if key is equal to current data
    //return current node
//3. if key is less than current data
    // search key into left of current node
//4. if key is greater than current data
    // search key into right of current node
//5. repeat step 2 to 4 till leaf nodes

key=7 — key is found

key=12 — key is not found

$T(n) = O(h)$

BST - Binary Search with Parent



Key=7

trav	Parent
&8	null
&3	&8
&6	&3
&7	&6

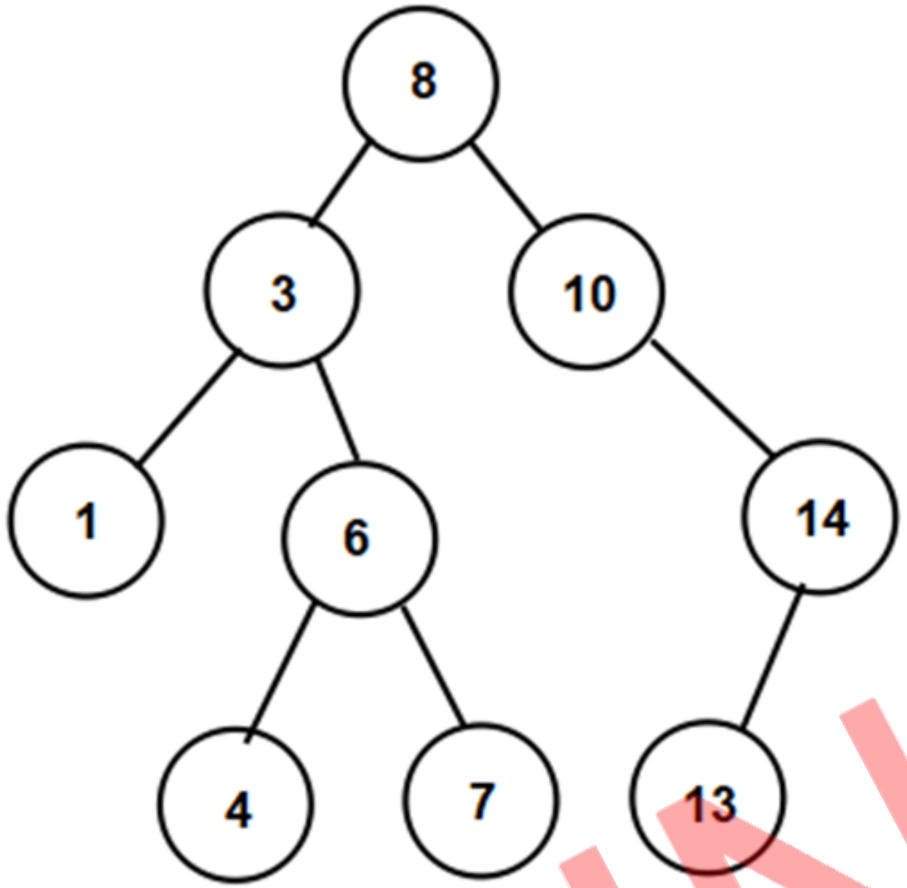
Key=15

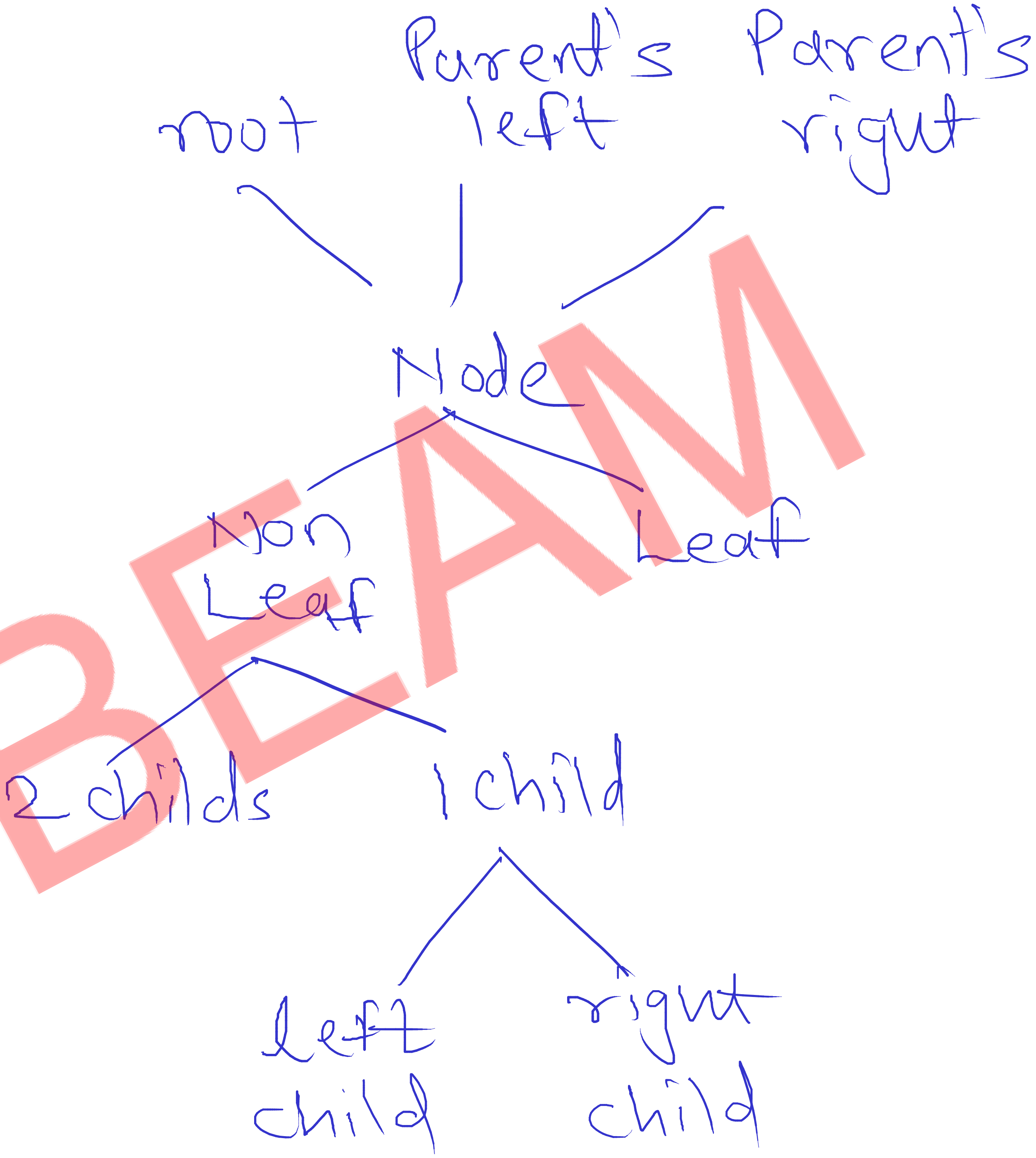
trav	Parent
&8	null
&10	&8
&14	&10
null	&14

Key=8

trav	Parent
&8	null

BST - Delete Node




root

Parent's left

Parent's right

Node

Non Leaf

Leaf

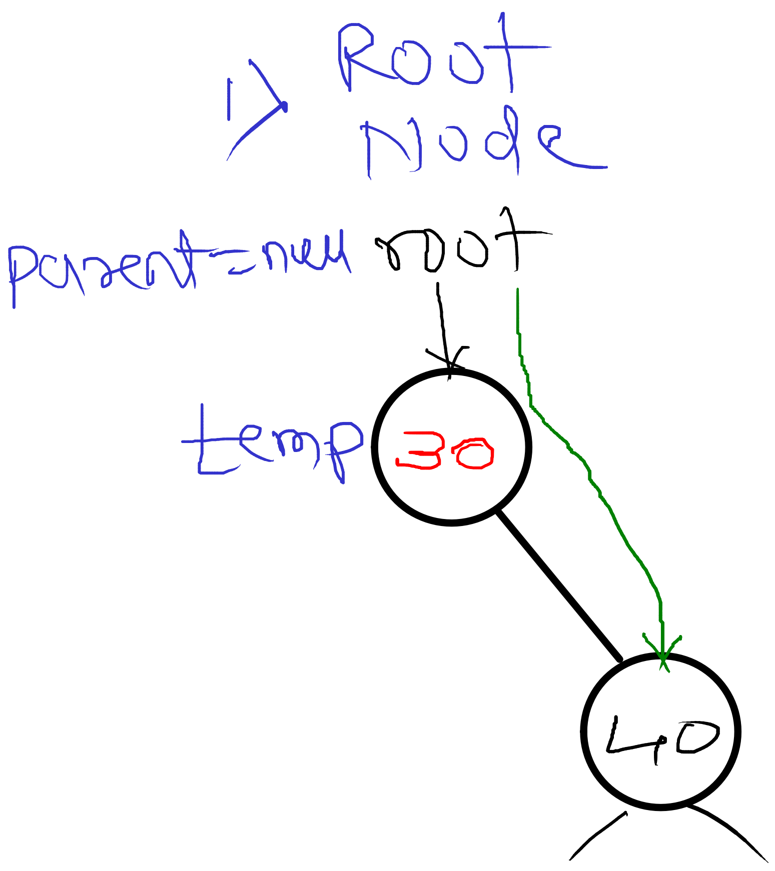
2 childs

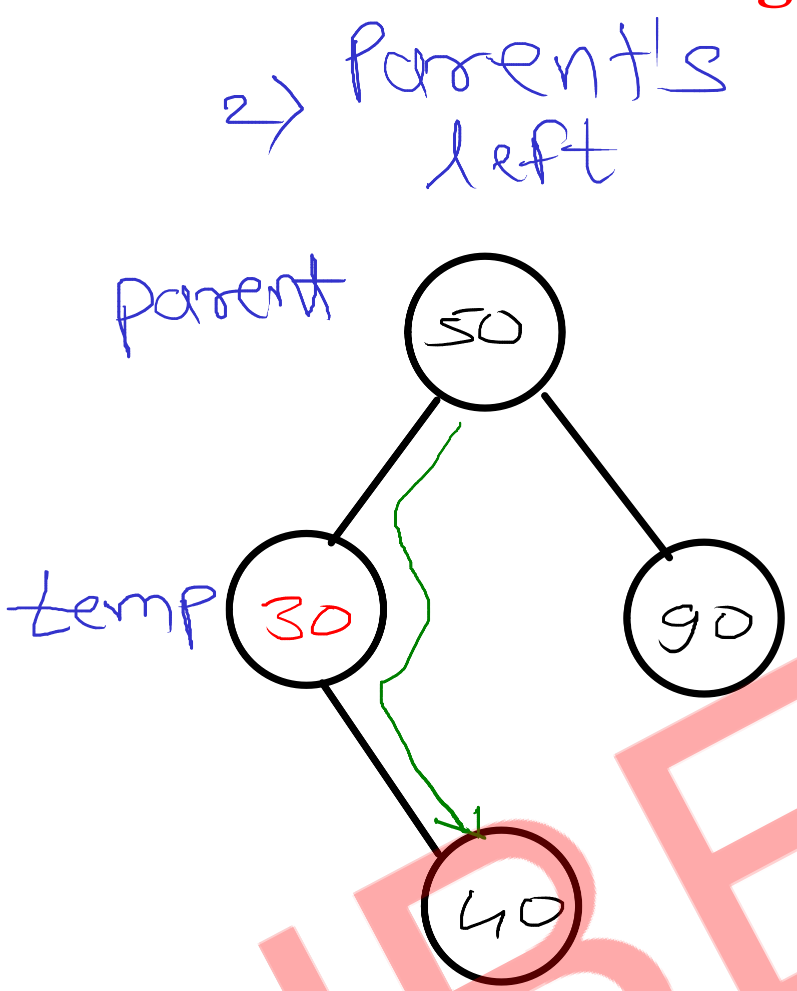
1 child

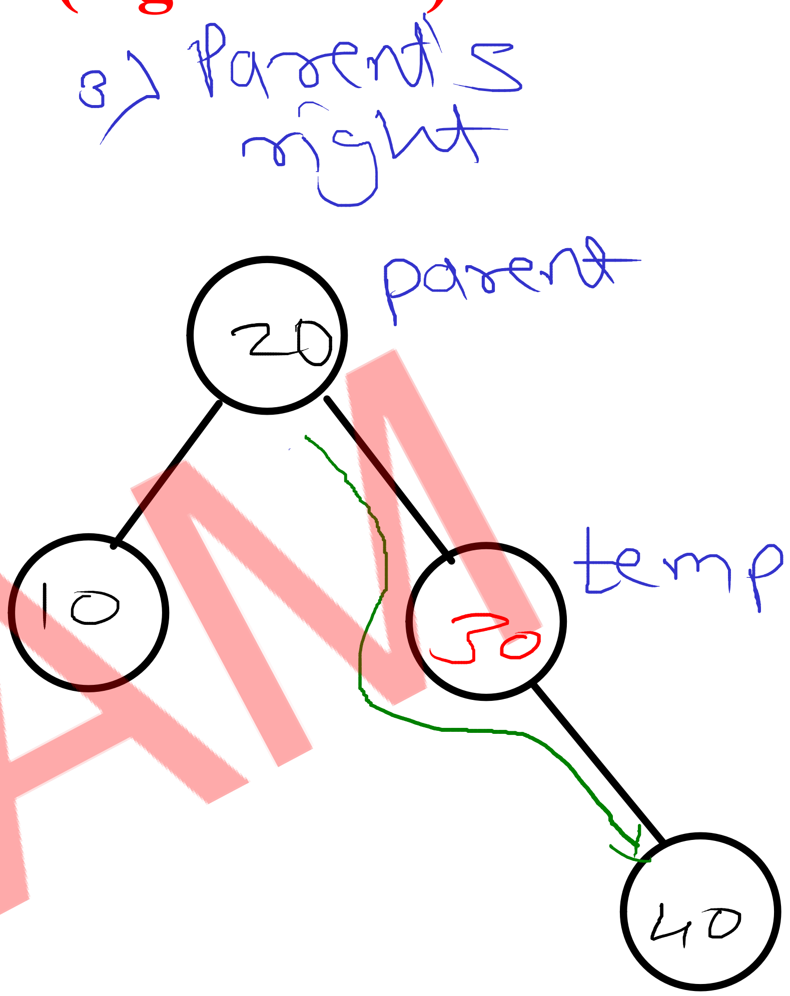
left child

right child

BST - Delete node which has single child (right child)







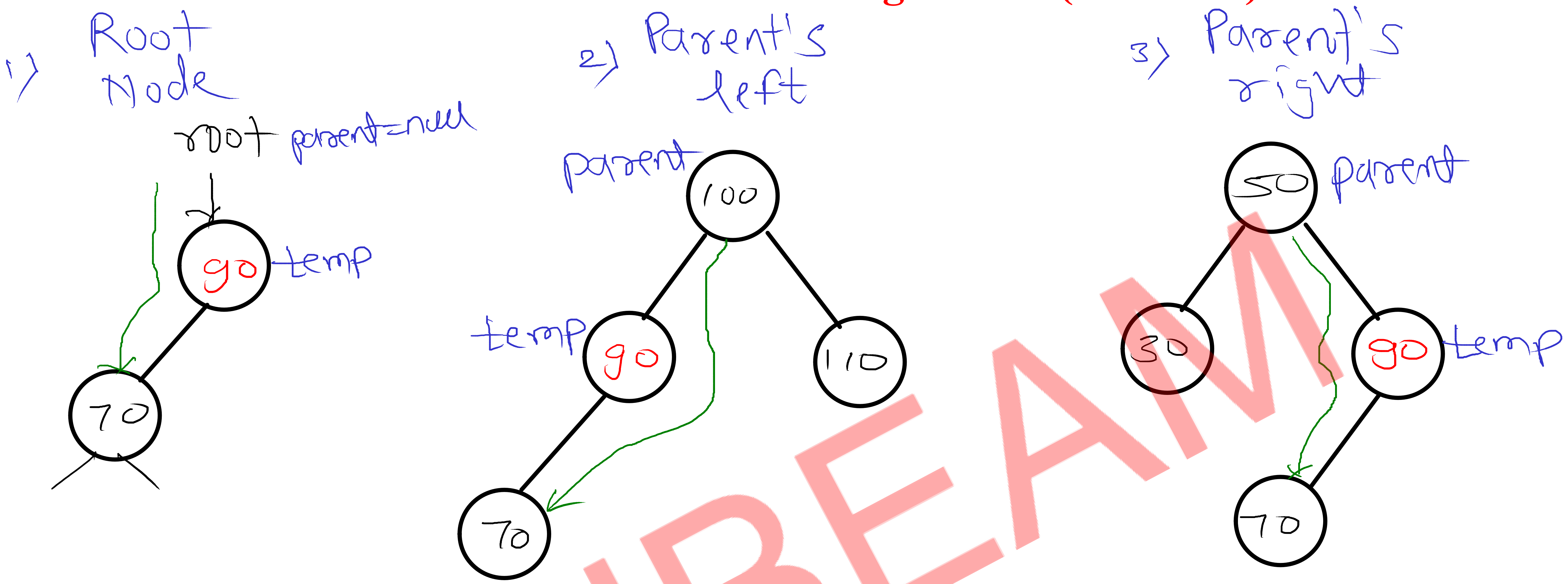
root → ①

parent's left → ②

parent's right → ③

```
if(temp.left == null){
        if(temp == root)
                root = temp.right;
        else if(temp == parent.left)
                parent.left = temp.right;
        else if(temp == parent.right)
                parent.right = temp.right;
}
```

BST - Delete node which has single child (left child)



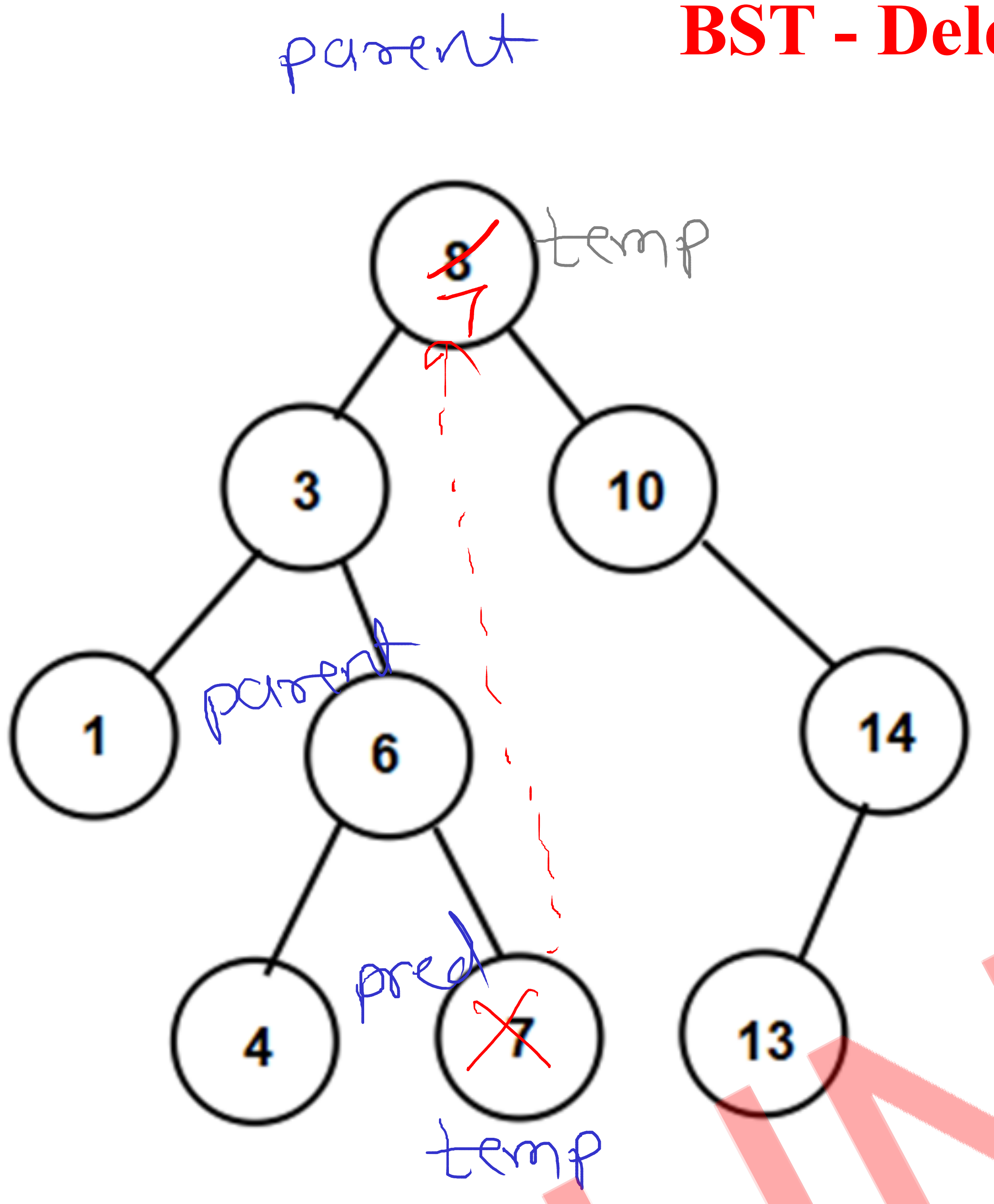
root → ①

parent's left → ②

parent's right → ③

```
if(temp.right == null){
        if(temp == root)
                root = temp.left;
        else if(temp == parent.left)
                parent.left = temp.left;
        else if(temp == parent.right)
                parent.right = temp.left;
}
```

BST - Delete node which has two childs



```
if(temp.left != null && temp.right != null){
        //1. find predecessor of temp
        Node pred = temp.left;
        parent = temp;
        while(pred.right != null){
                parent = pred;
                pred = pred.right;
        }
        //2. replace value of temp by predecessor
        temp.value = pred.value;
        //3. delete predecessor
        temp = pred;
}
```

Inorder : 1 3 4 6 7 8 10 13 14

left extreme right → inorder predecessor

inorder successor ← right extreme left