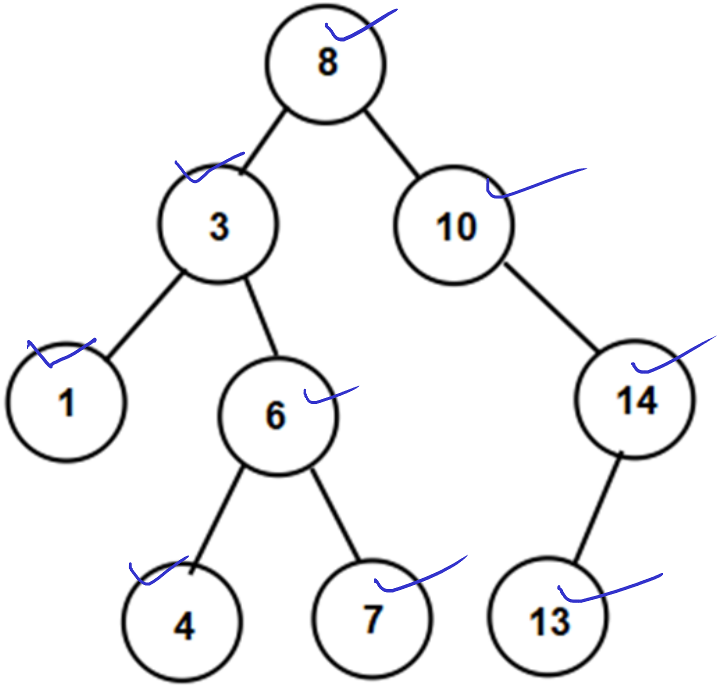
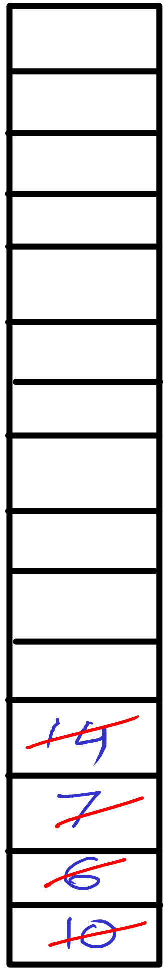
BST - Preorder

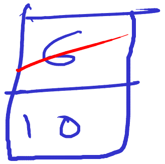




Preorder: 8, 3, 1, 6, 4, 7, 10, 14, 13

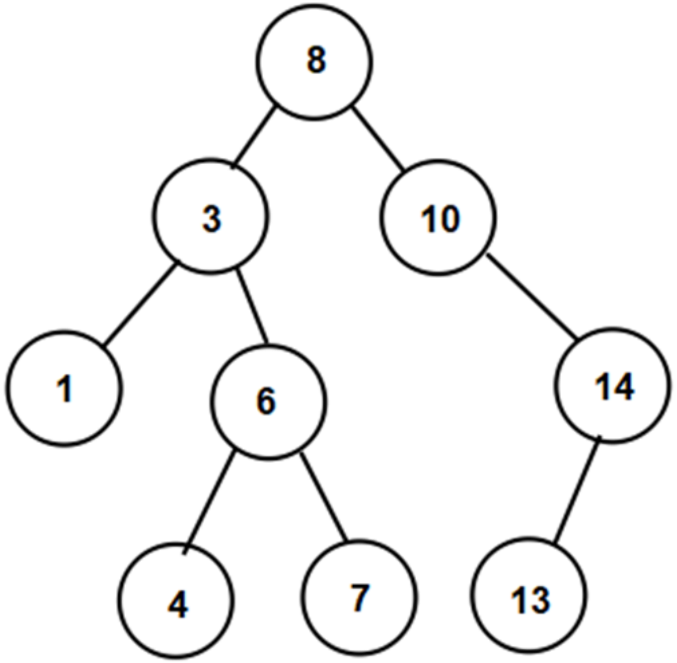
trav = ~~8~~, ~~3~~, 1, null, 6

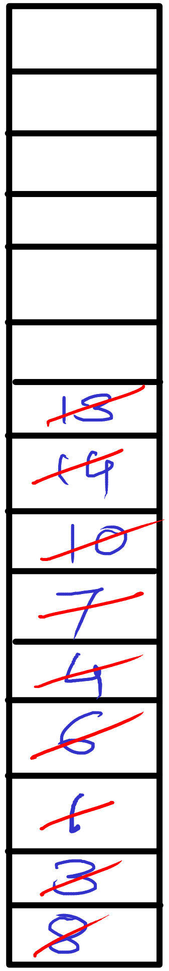
8, 3, 1



```
//1. start traversing from root
        //2. visit trav
        //3. if trav has right, push trav->right on stack
        //4. go to left of trav
//5. repeat 2-4 until trav is null
//6. pop node from stack into trav
//7. repeat 2-6, until trav is null or stack is empty
```

BST - Inorder

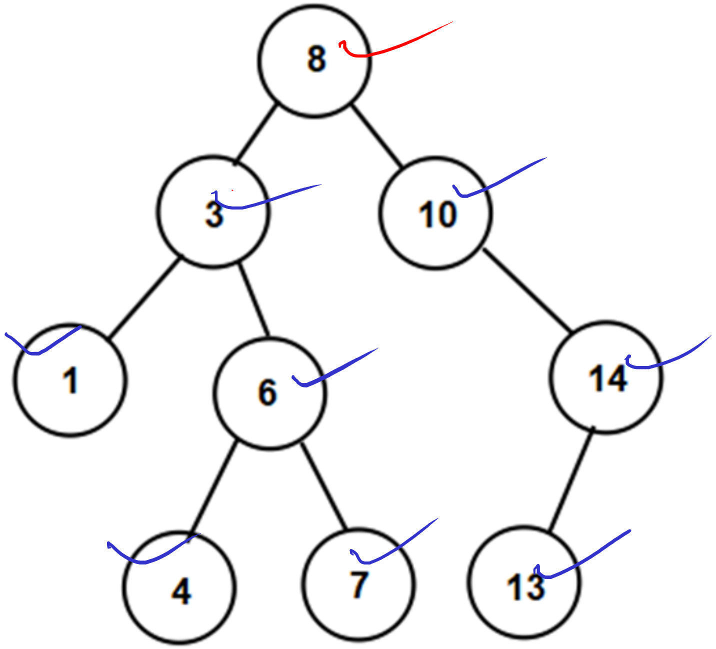


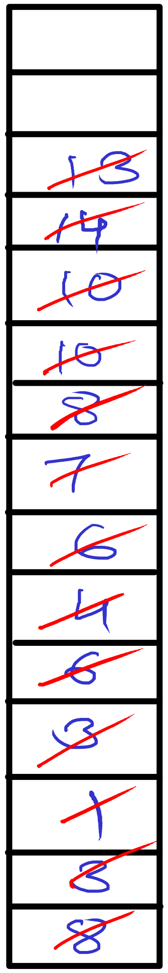


Inorder : 1, 3, 4, 6, 7, 8, 10, 13, 14

```
//1. start traversing from root
        //2. push trav on stack
        //3. go to left of trav
//4. repeat 2-3 until trav is null←
//5. pop node from stack into trav
//6. visit trav
//7. go to the right
//8. repeat 2-7, until trav is null or stack is empty
```

BST - Postorder

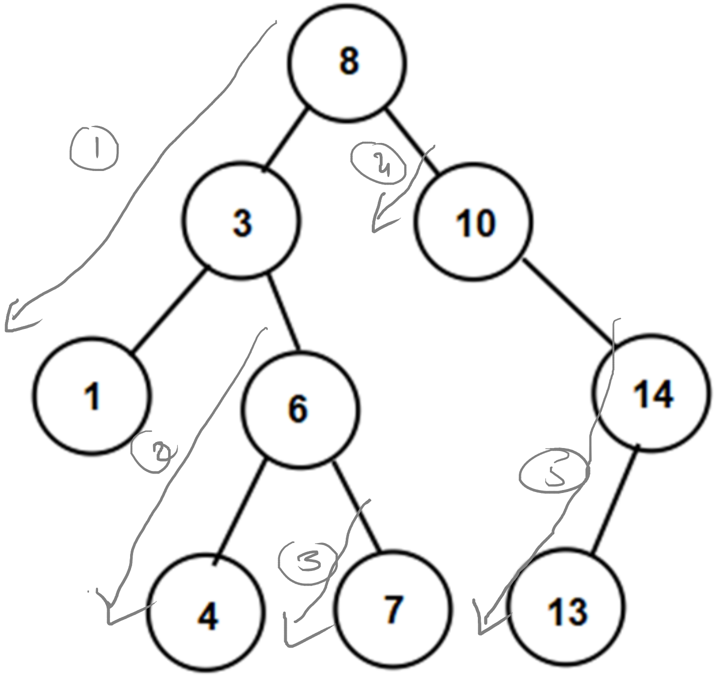


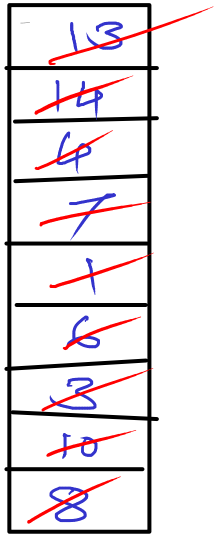


Postorder: 1,4,7,6,3,13,14,10,8

```
// start trav from root
// while trav is not null or stack is not empty
    // until null is reached
        // push trav on stack
        // go to trav's left
    // if stack is not empty
        // pop node from stack into trav
        // if trav's right is not present or visited
            // visit trav & mark it as visited
            // make trav null (so that next node
            will be popped from stack)
        // otherwise
            // push node on stack
            // go to its right
```

BST - DFS

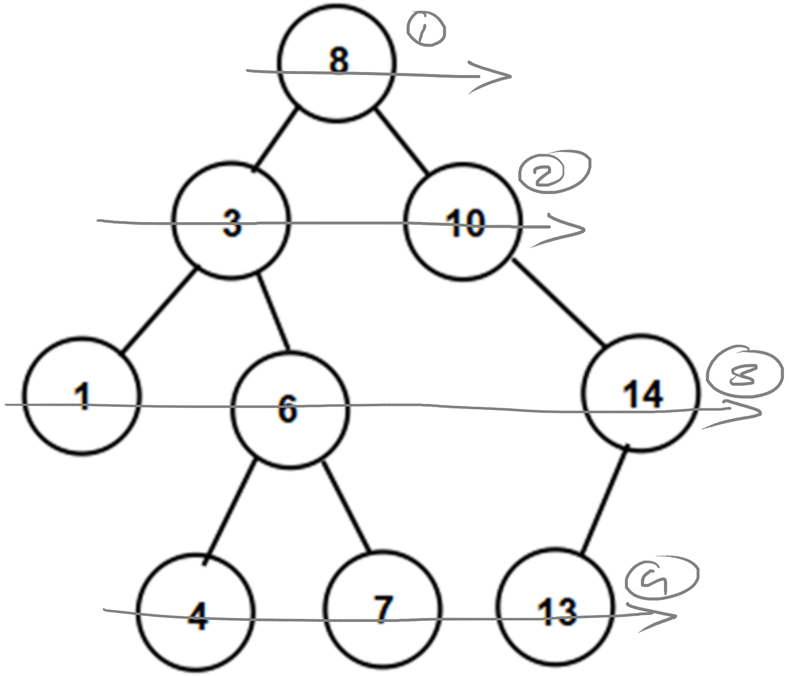


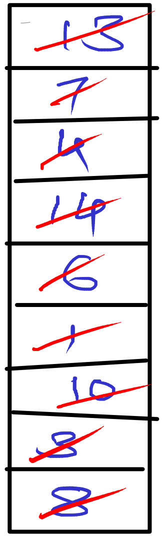


DFS:8,3,1,6,4 7,10,14,13

```
//1. push root on stack
//2. pop node from stack
//3. visit poped node
//4. if popped node has right
      // push it on stack
//5. if poped node has left
      // push it on stack
//6. repeat step 2 to 5 till stack iss not empty
```

BST - BFS

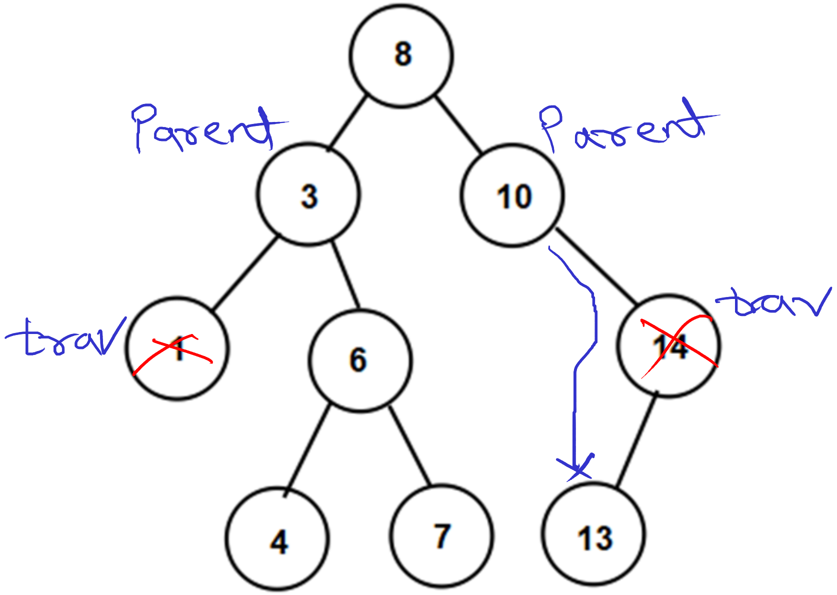




BFS=8,3,10,1,6,14,4,7,13

```
//1. push root on queue
//2. pop node from queue
//3. visit poped node
//5. if poped node has left
        // push it on queue
//4. if popped node has right
        // push it on queue
//6. repeat step 2 to 5 till stack iss not empty
```

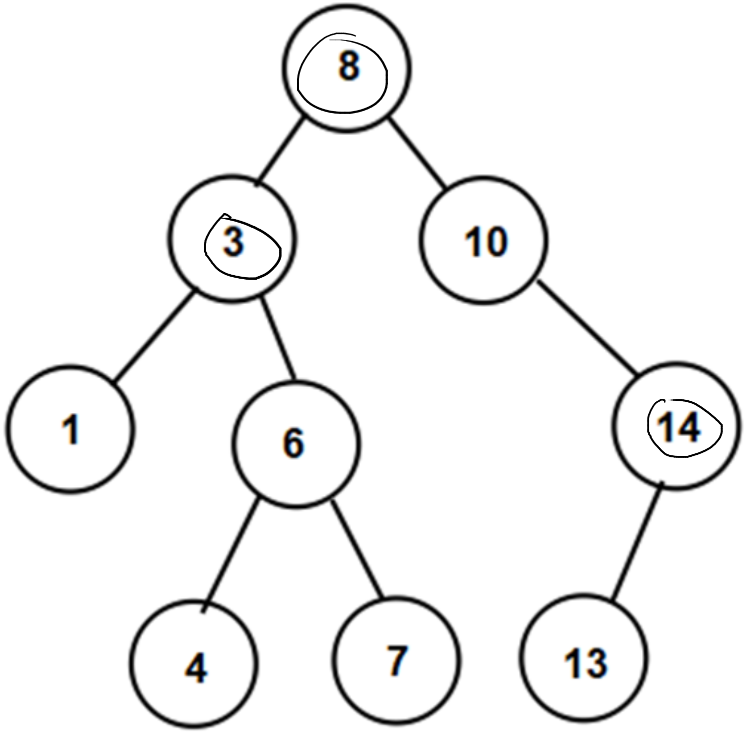
Search with Parent

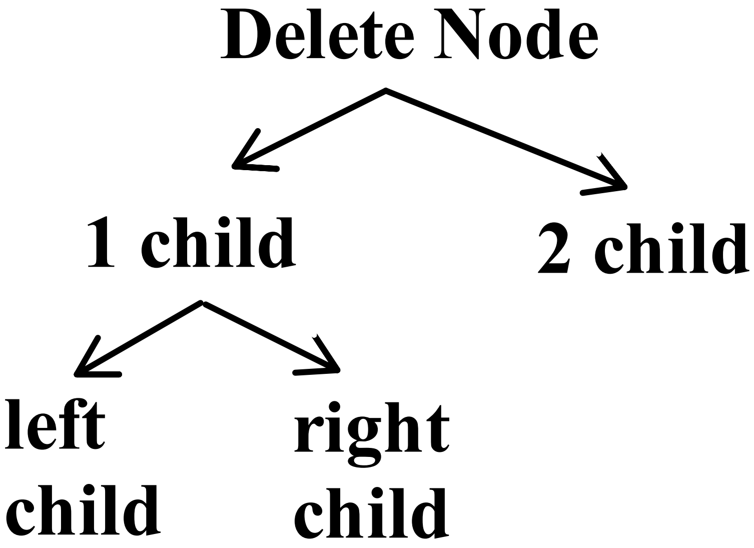


```
Node [] searchWithParent(int key) {
    Node parent = null;
    Node trav = root;
    while(trav != null) {
        if( key == trav.data)
            break;
        parent = trav;
        else if(key < trav.data)
            trav = trav.left;

        else
            trav = trav.right;
    }
    if(trav == null) //not found;
        parent = null;
    return new Node[] {trav,
                       parent};
}
```

BST - Delete Node





Delete Node

1 child

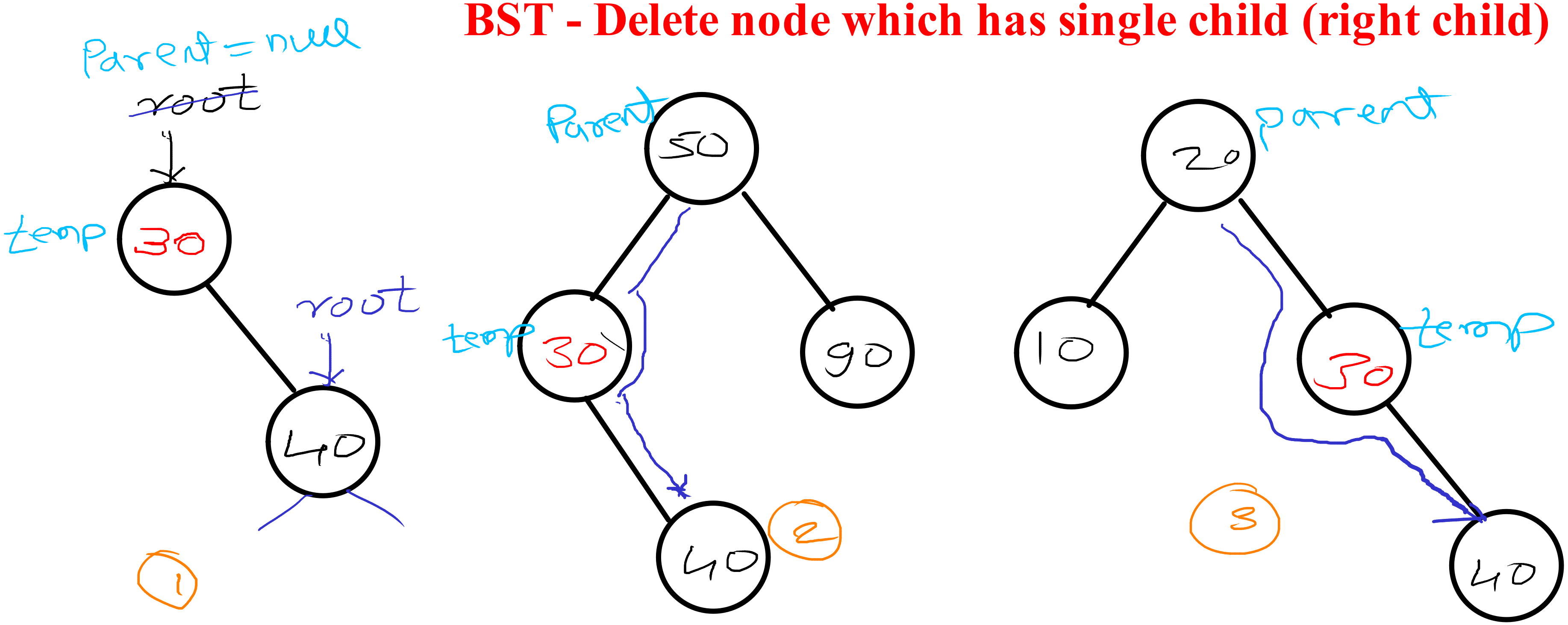
2 child

left child

right child

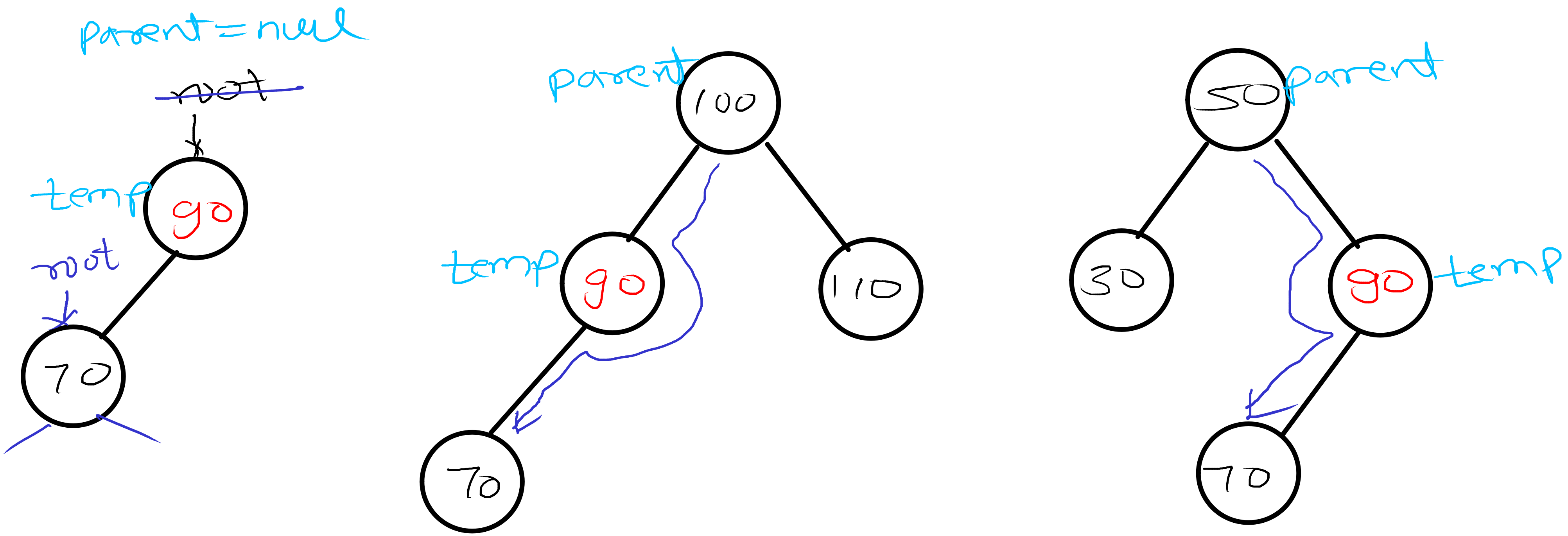
root
parent's left
parent's right

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



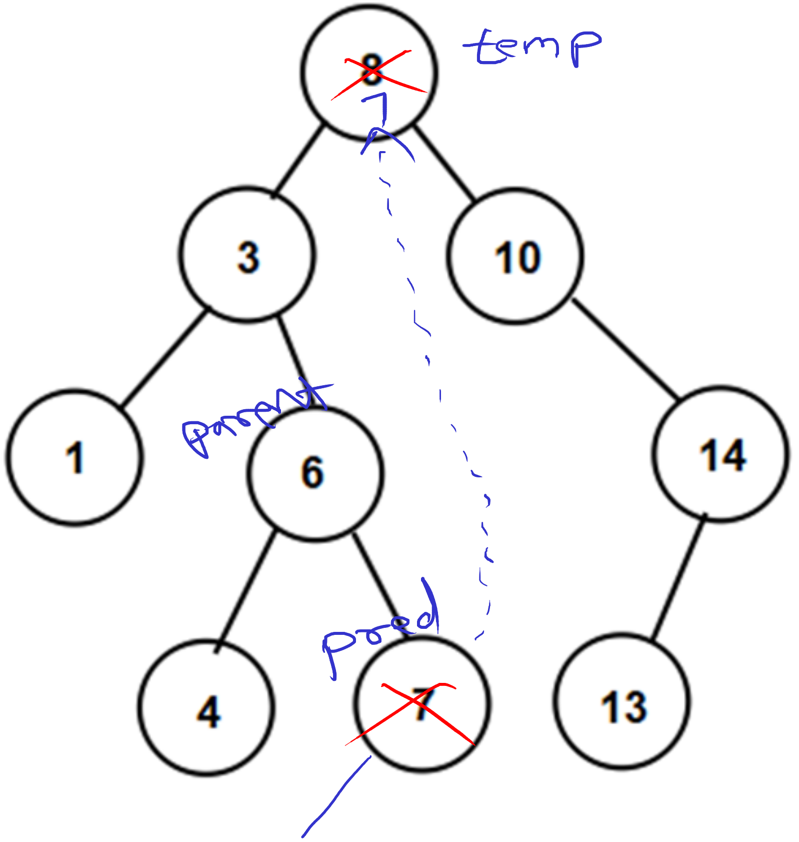
```
if(temp.left == null){
        if(temp == root)
1               root = temp.right;
        else if(temp == parent.left)
2               parent.left = temp .right;
        else
3               parent.right = temp.right;
}
```

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



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

BST - Delete node which has two childs



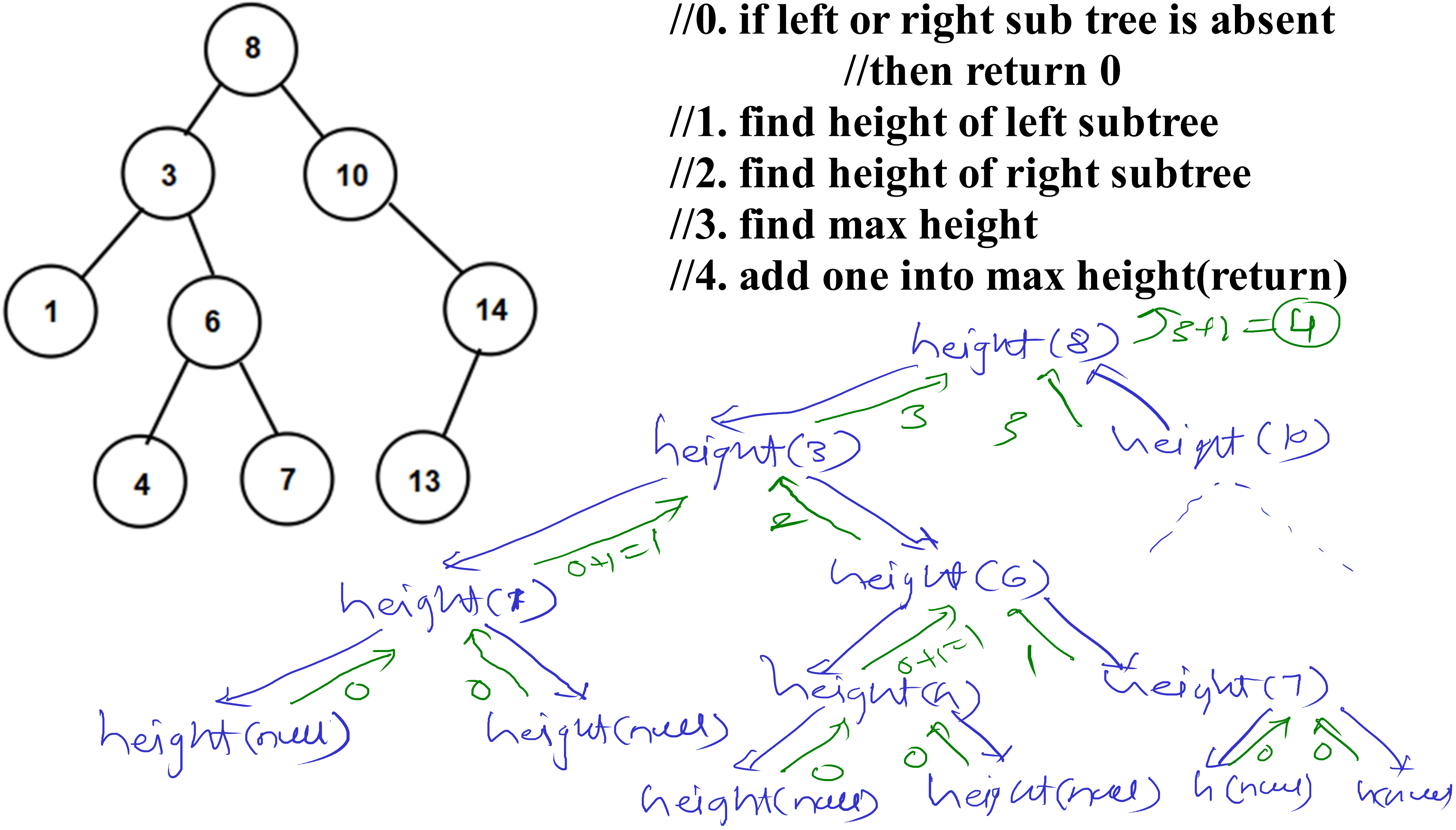
```
if(temp.left != null && temp.right != null){
        Node pred = temp.left;
        parent = temp;
        while(pred.right != null){
                parent = pred;
                pred = pred.right;
        }
        temp.data = pred.data;
        temp = pred;
}
```

Inorder : 1 3 4 6 7 8 10 13 14

predecessor → left extreme right

successor → right extreme left

BST - height



//0. if left or right sub tree is absent

//then return 0

//1. find height of left subtree

//2. find height of right subtree

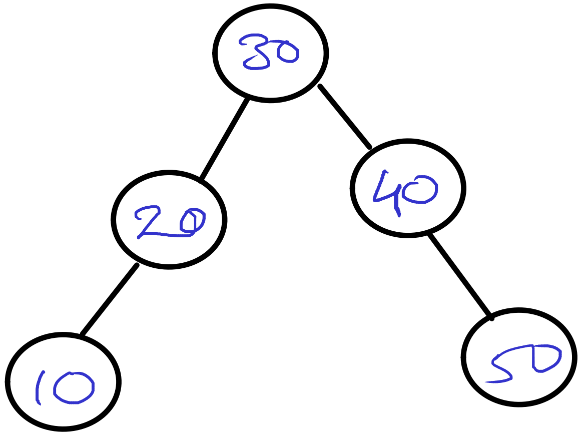
//3. find max height

//4. add one into max height(return)

Height(BST) = max(Height(left sub tree), Height(right sub tree)) + 1

Skewed BST

Keys : 30, 40, 20, 50, 10

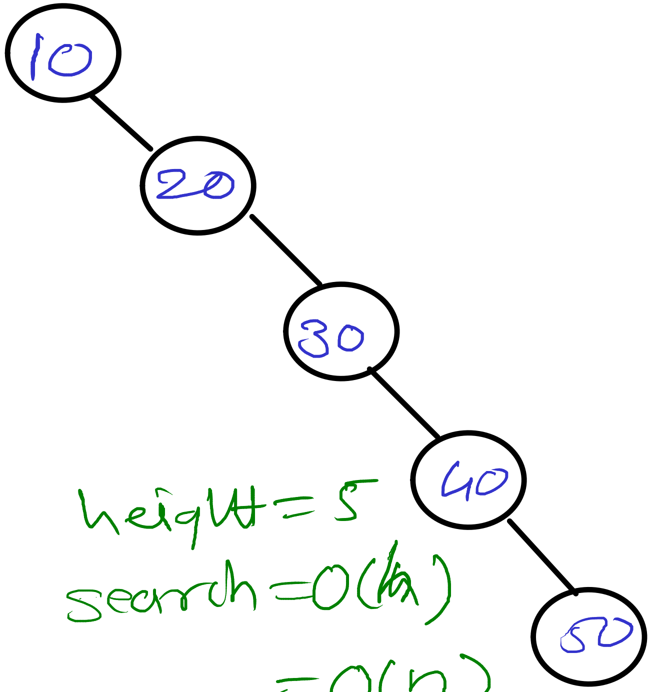


height = 3

search = O(h)

= O(log n)

Keys : 10, 20, 30, 40, 50

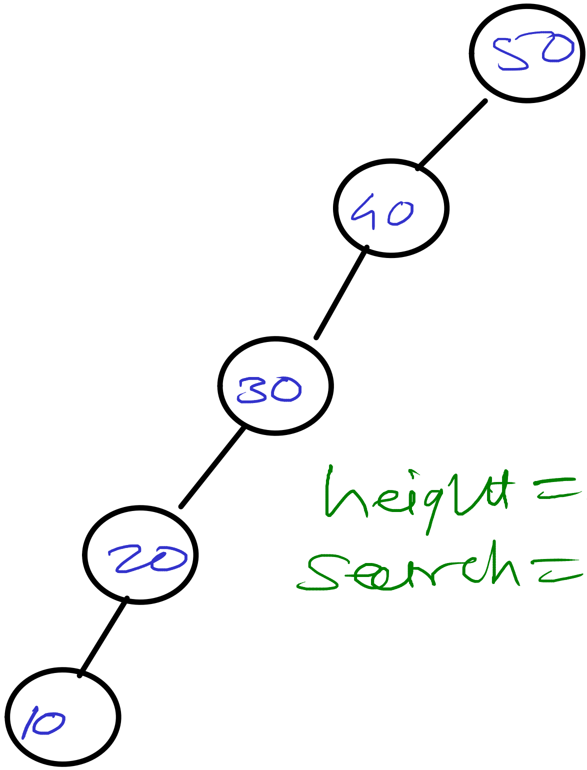


height = 5

search = O(h)

= O(n)

Key : 50, 40, 30, 20, 10

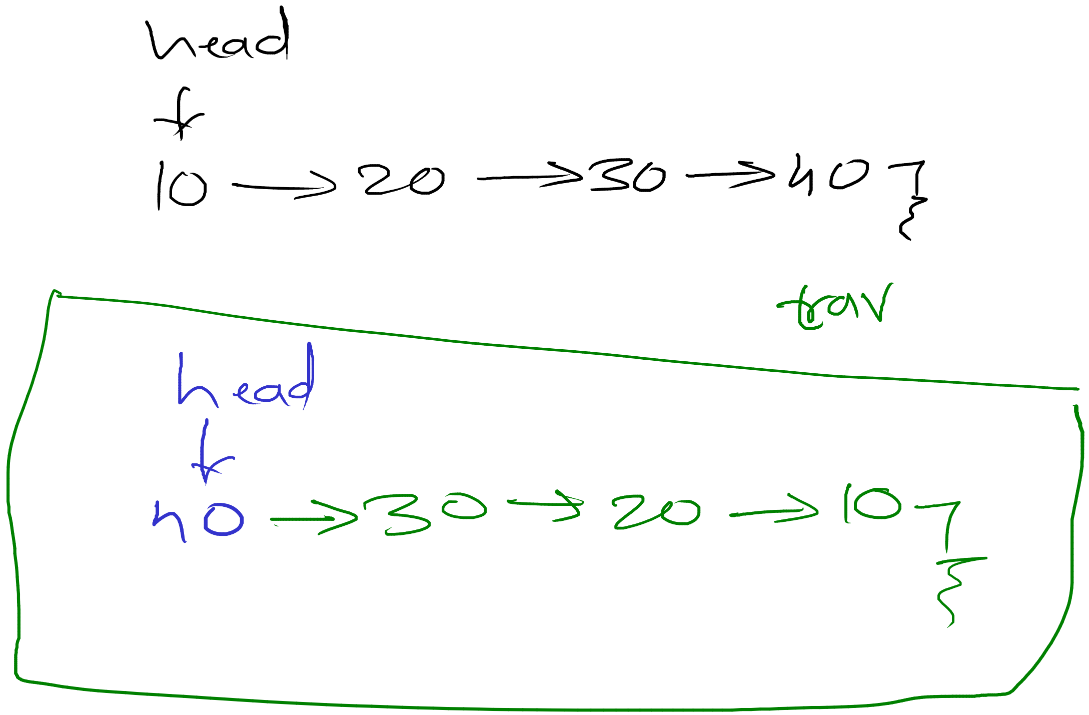


height = 5

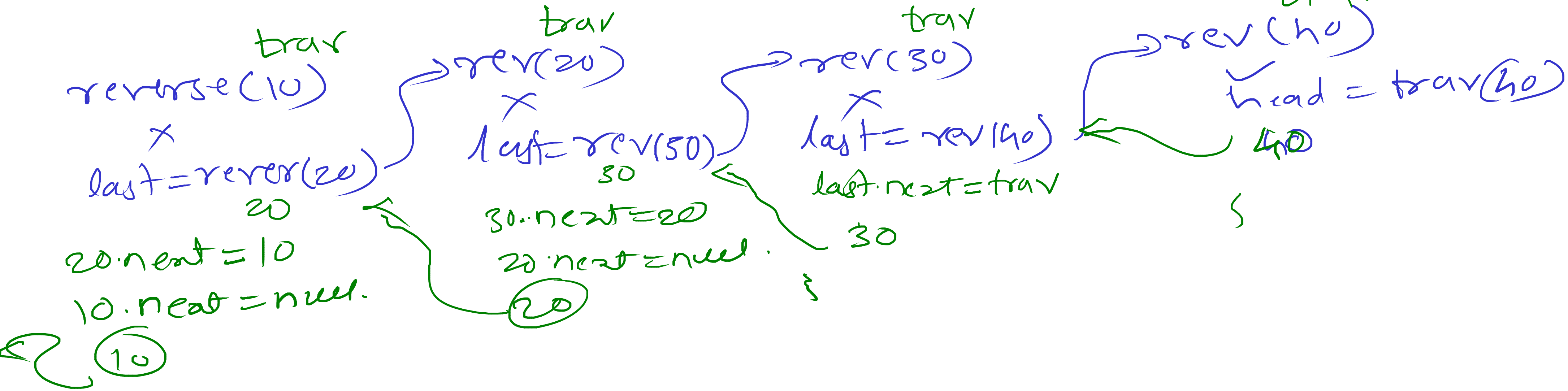
search = O(h)

O(n)

- if BST is growing only in one direction, such tree is called as skewed BST
- if BST is growing in right direction only, such tree is called as Right skewed tree
- if BST is growing in left direction only, such tree is called as left skewed tree



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


trav
reverse(10)
last=rever(20)
20
20.next=10
10.next=null.
10
trav
rev(20)
last=rev(50)
30
30..next=20
20.next=null.
20
trav
rev(30)
last=rev(40)
last.next=trav
30
trav
rev(40)
head=trav(40)
40