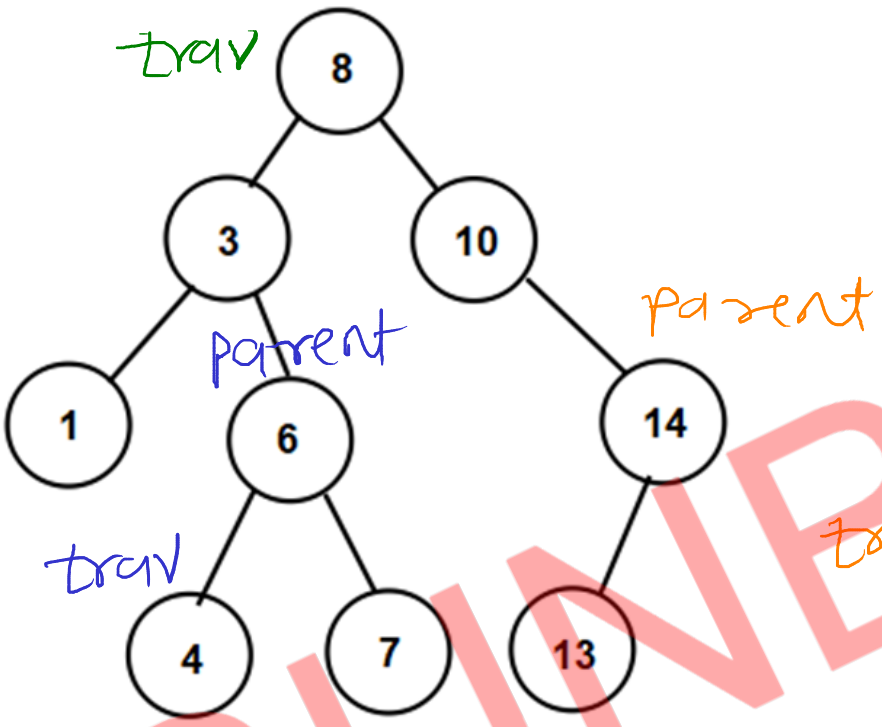
BST - Search with Parent



Key = 15

trav	parent
&8	null
&10	&8
&14	&10
null	&14

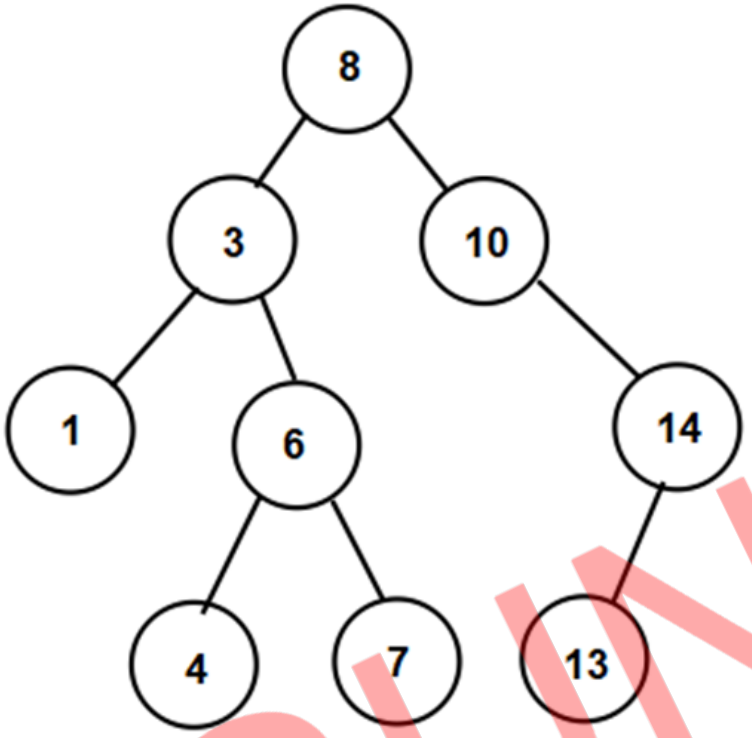
Key = 4

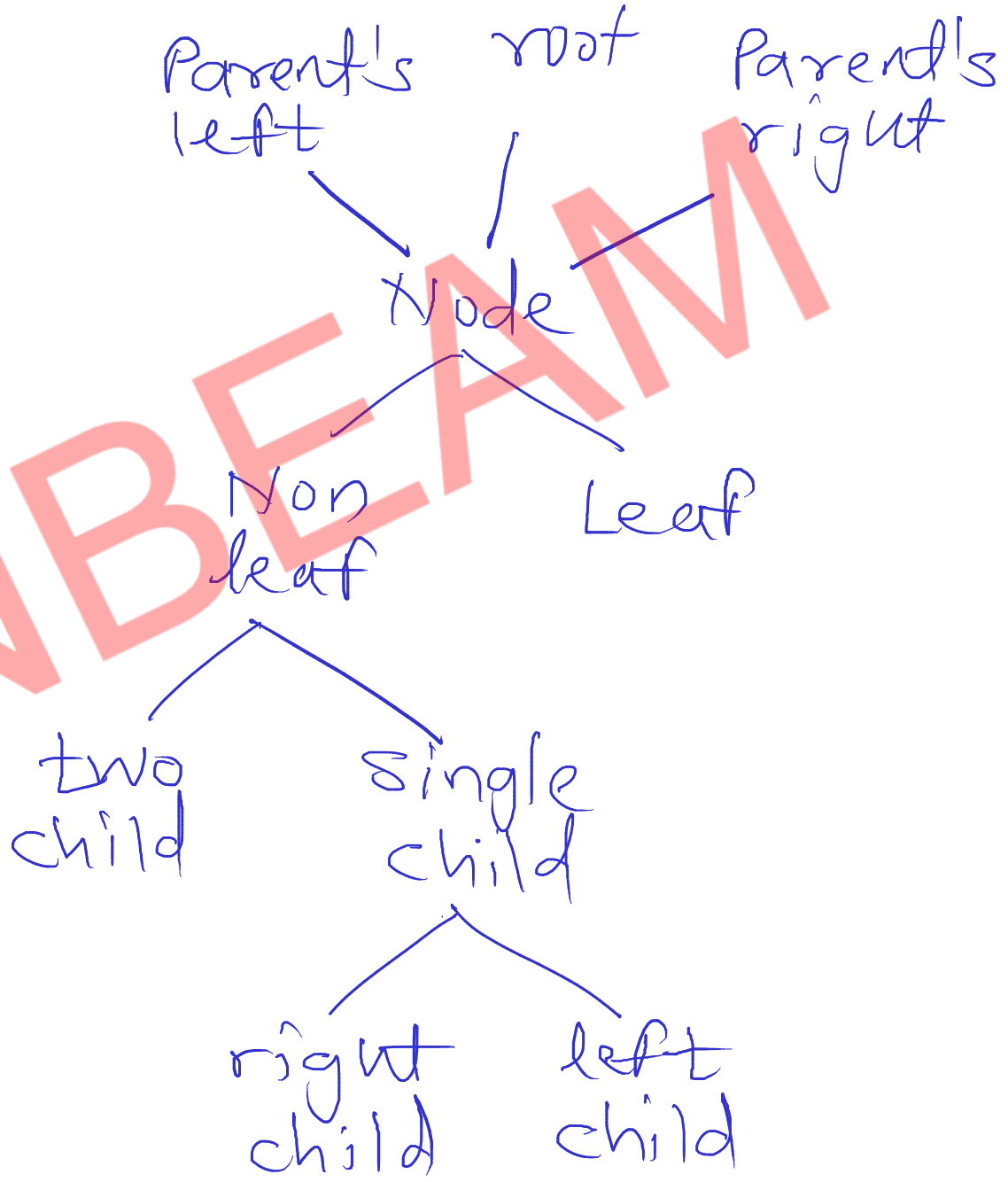
trav	parent
&8	null
&3	&8
&4	&3
	&6

Key = 8

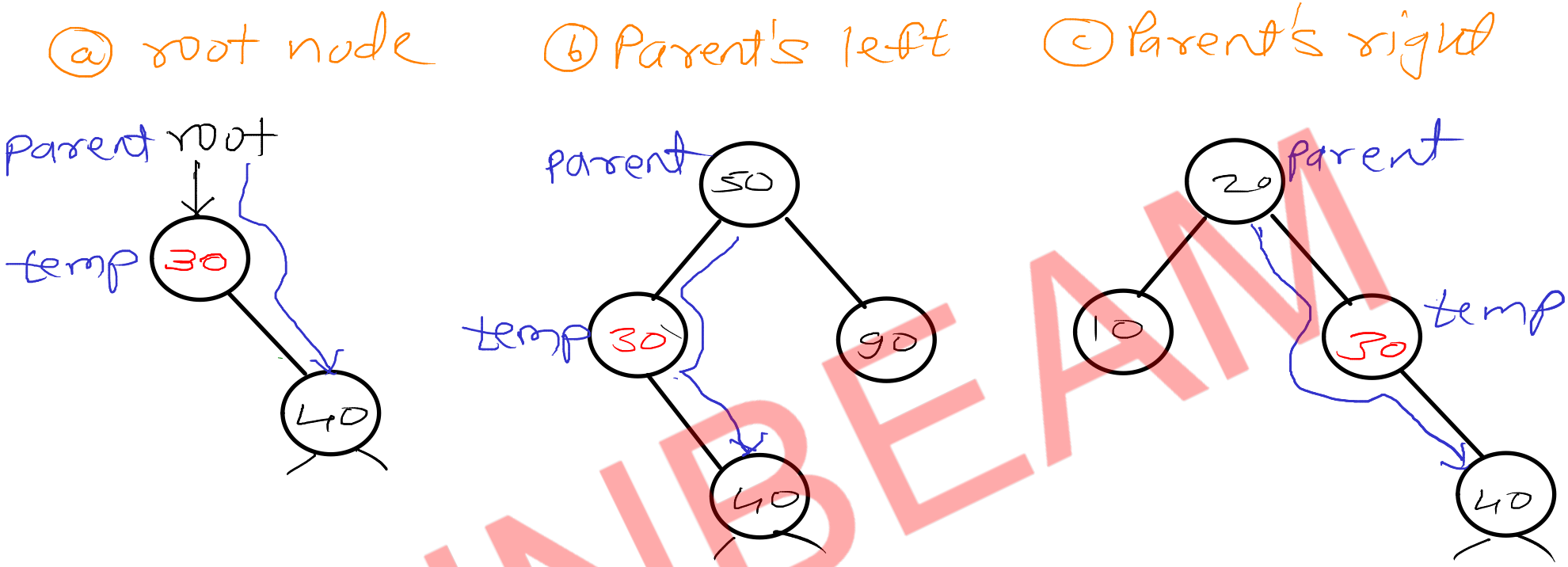
trav	parent
&8	null

BST - Delete Node



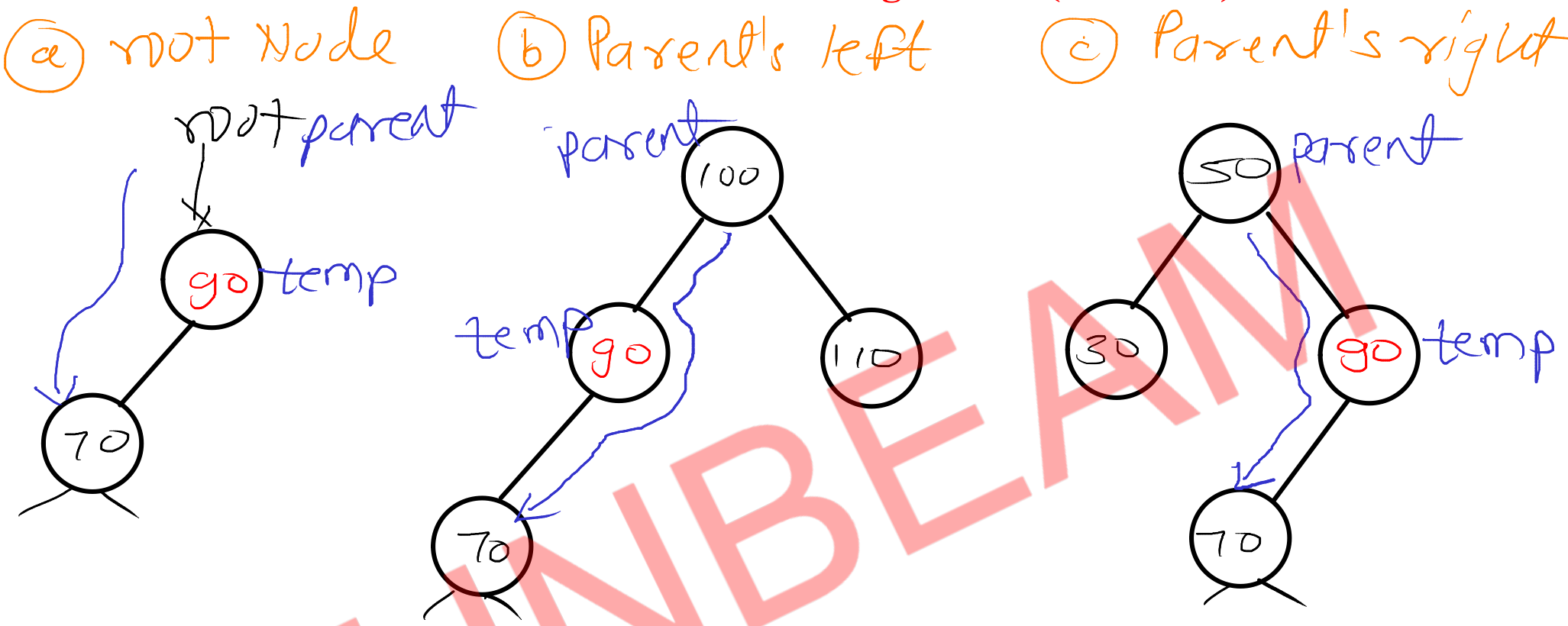


BST - Delete node which has single child (right child)



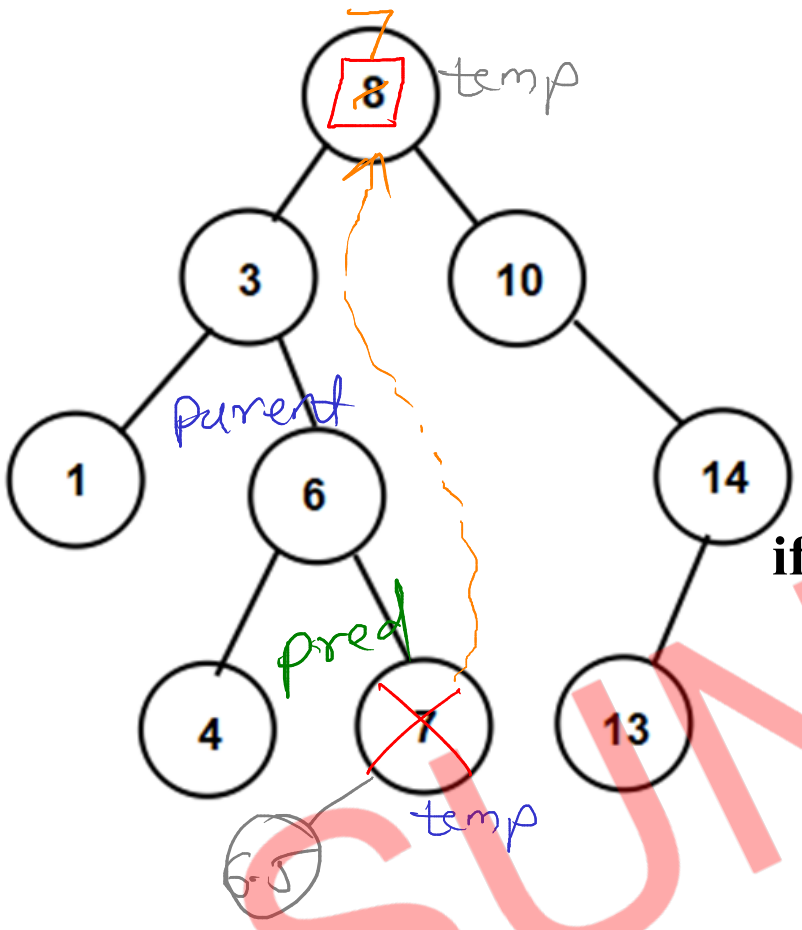

```
if(temp.left == null){
    if(temp == root)                    // root node
        root = temp.right;
    else if(temp == parent.left)        // Parent's left
        parent.left = temp.right;
    else if(temp == parent.right)       // Parent's right
        parent.right = temp.right;
}
```

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




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

BST - Delete node which has two childs



Inorder:

1 3 4 6 7 8 10 13 14

inorder predecessor

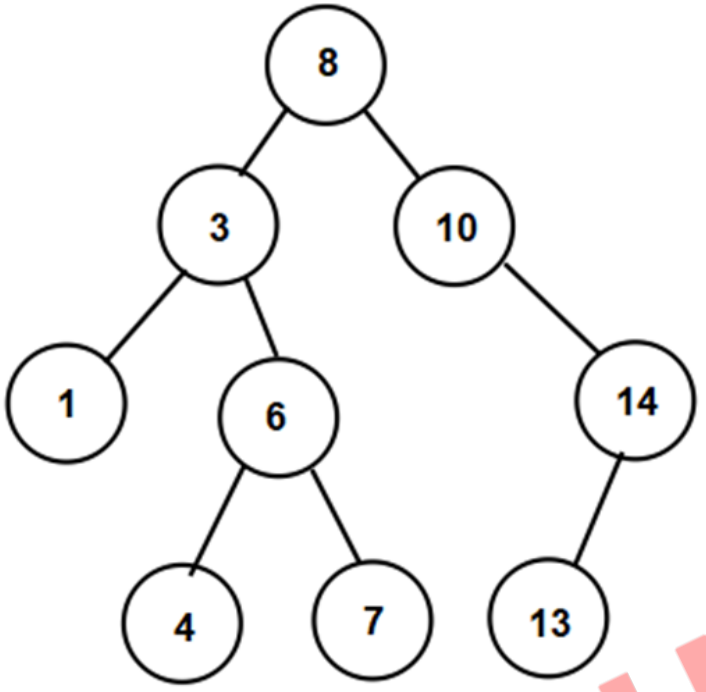
left extreme right

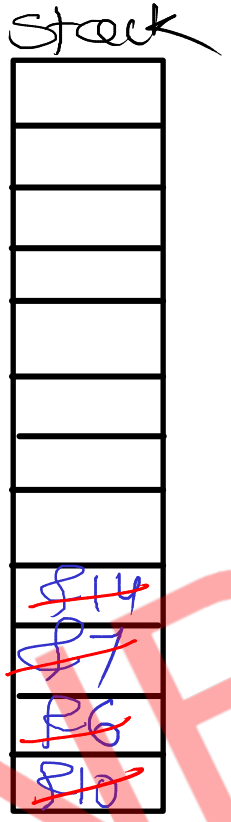
inorder successor

right extreme left

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

BST - Preorder



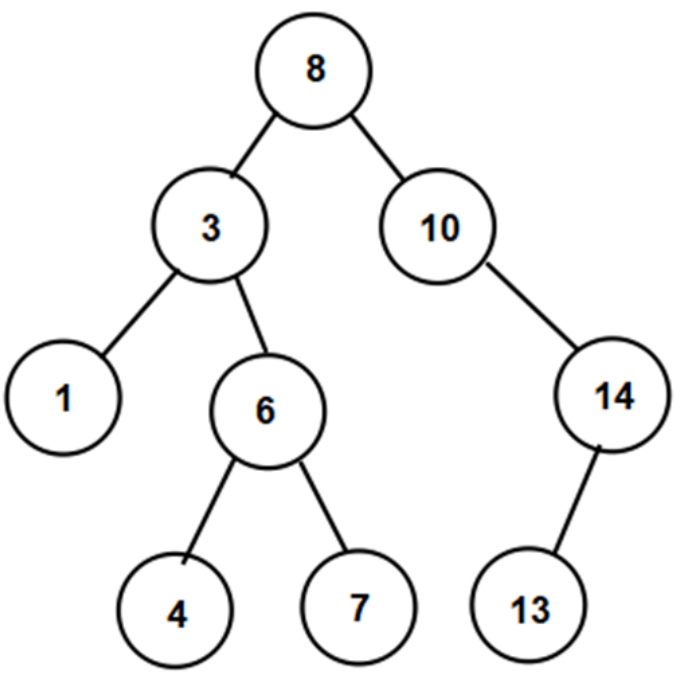


Traversal:
8, 3, 1, 6, 4, 7, 10, 14, 13

AS(n) = O(n)

//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

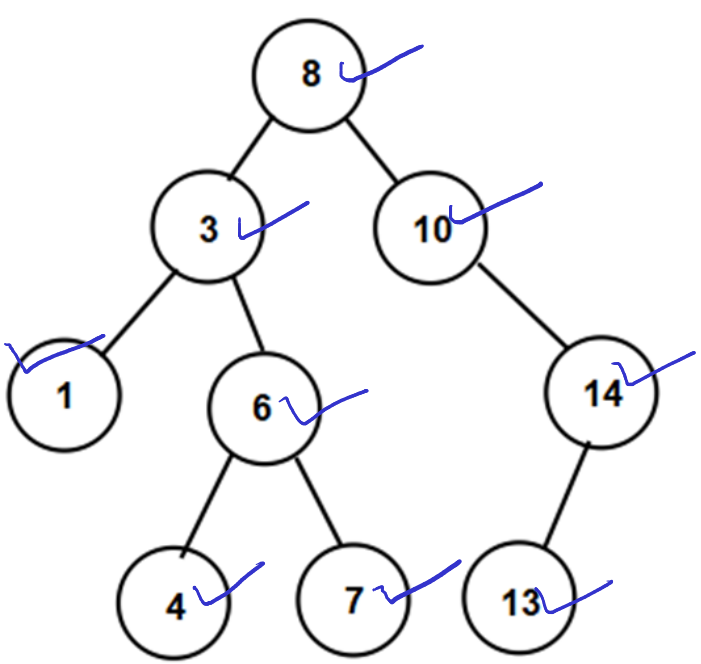
BST - Inorder



Traversal:
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

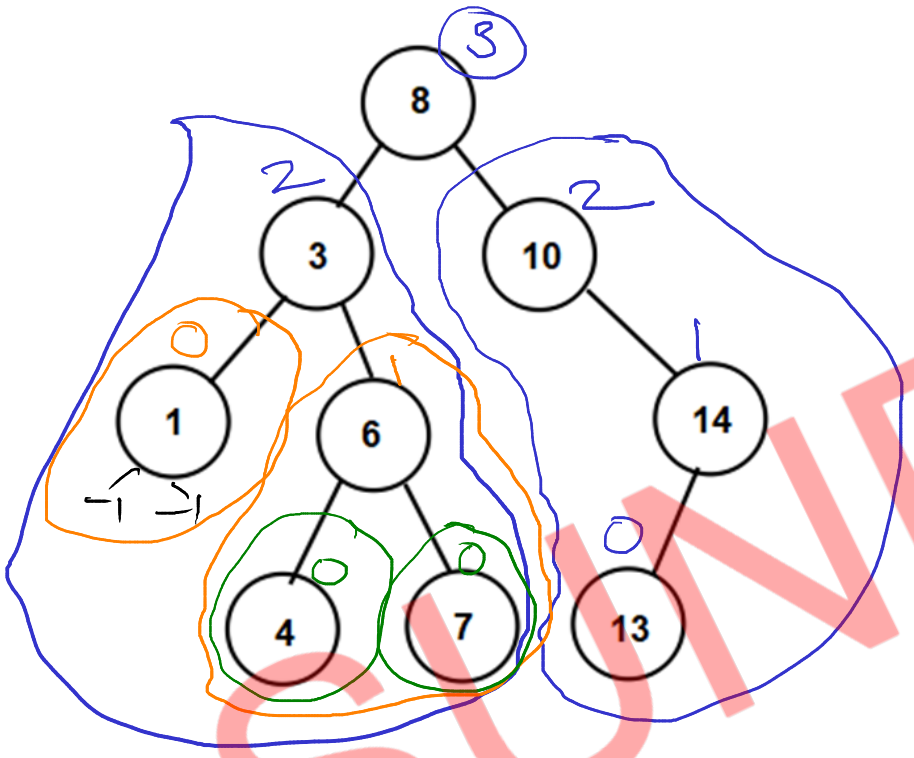


1, 4, 7, 6, 3, 13, 14, 10, 8

```
// 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 - Height

Height(Node) = MAX(Height(left sub tree), Height(right sub tree)) + 1




```
//1. if sub tree is absent(empty) then
       return -1;
//2. find height of left subtree
//3. find height of right subtree
//4. find max of both heights
//5. return max height + 1
```

root → empty tree / null tree → height = -1