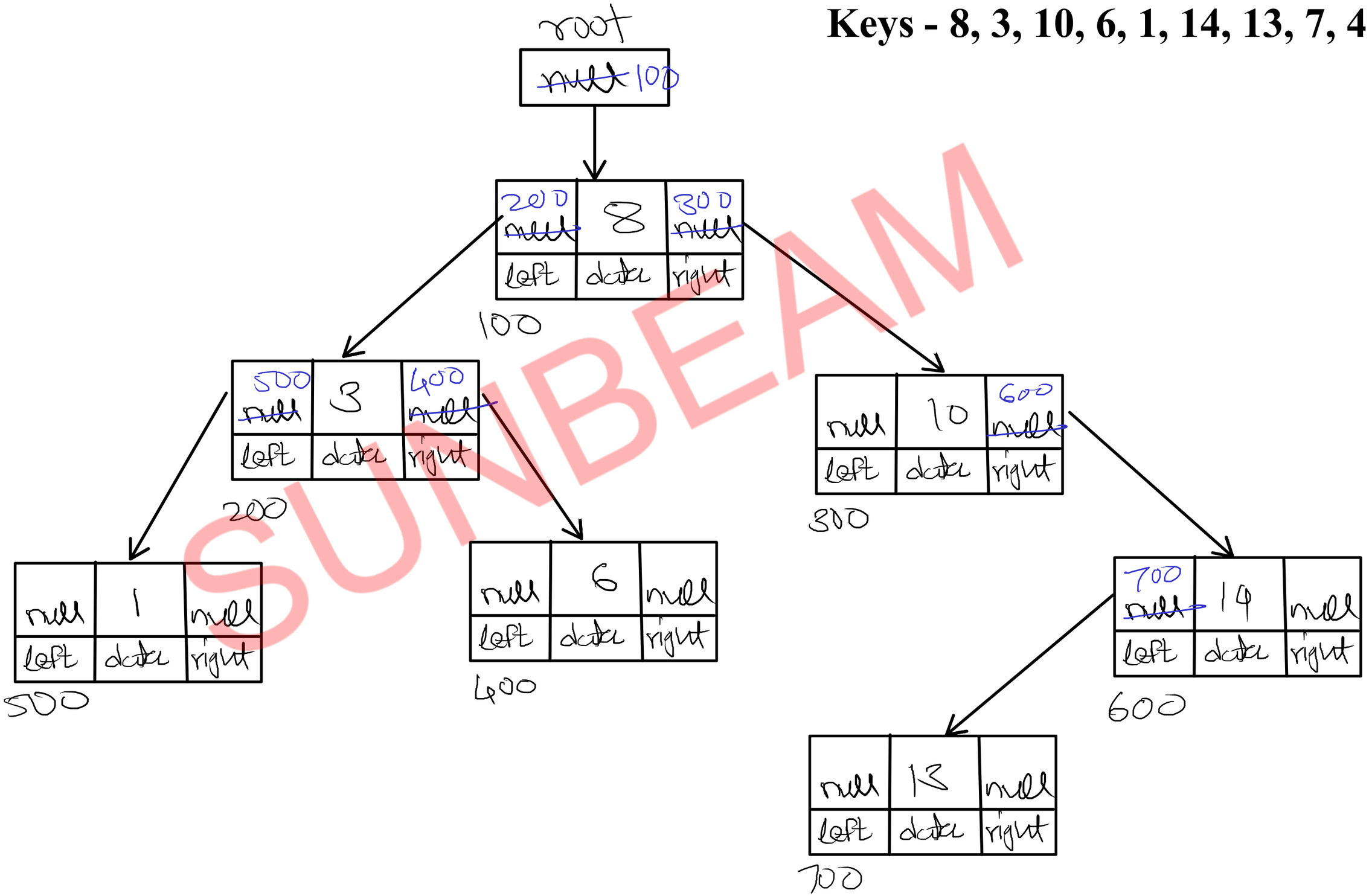
BST - Add Node

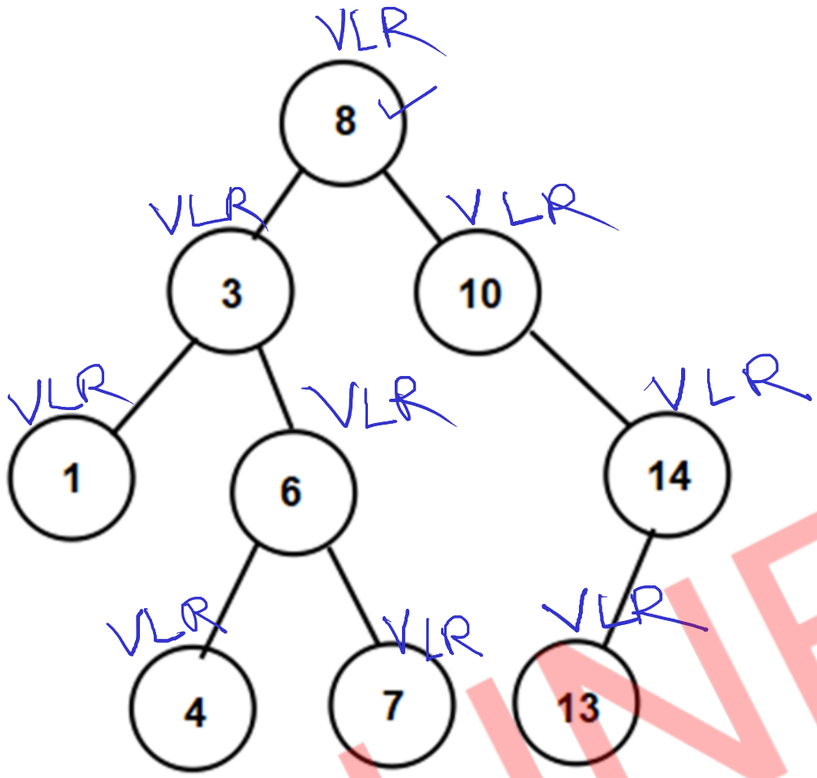
```
//1. create a newnode with given data
//2. if tree is empty
        // add newnode into root itself
//3. if tree is not empty
        //3.1 create trav pointer and start at root
        //3.2 if value is less than current node data
                //3.2.1 if left of current node is empty
                        // add newnode into left of current node
                //3.2.2 if left of current node is not empty
                        // go to left of current node
//3.3 if value is greater than current node 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 to right of current node
//3.4 repeat spep 3.2 and 3.3 till node is not added into BST
```

BST - Add Node

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

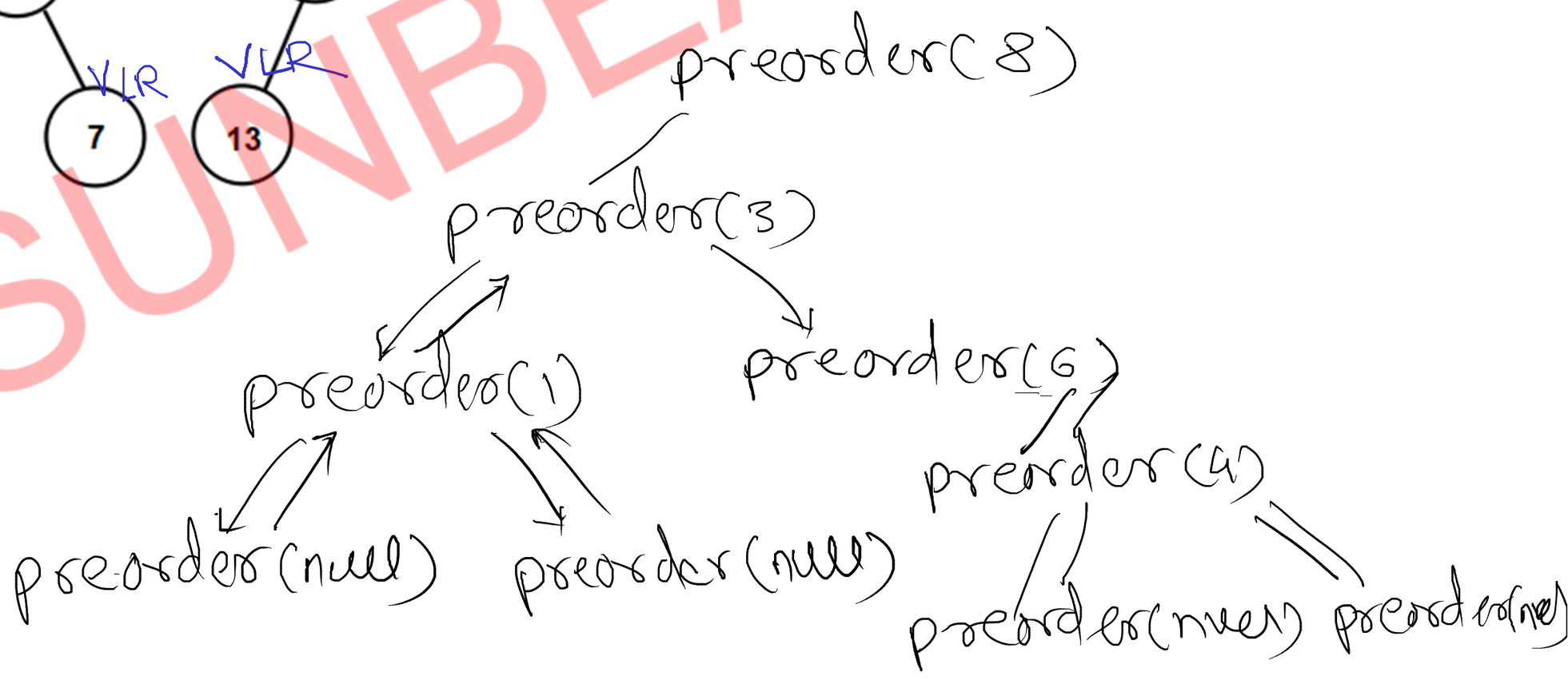


BST - Preorder Traversal

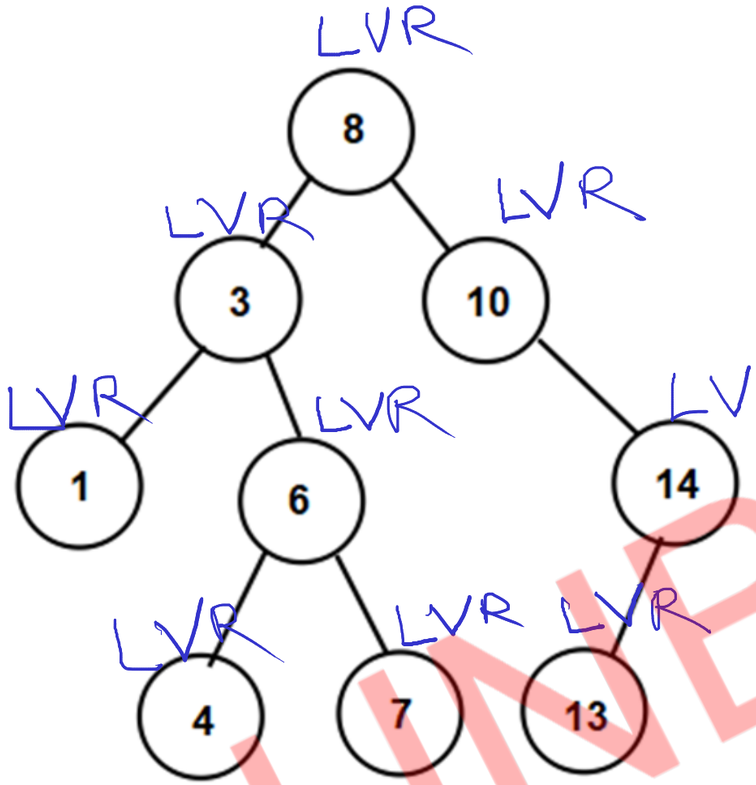


VLR

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



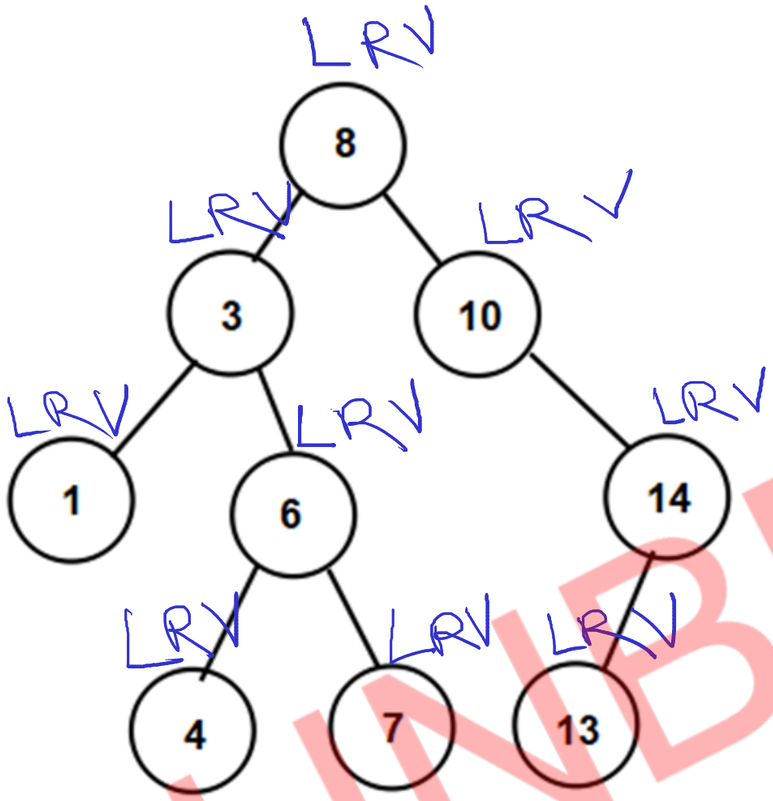
BST - Inorder Traversal



LVR

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

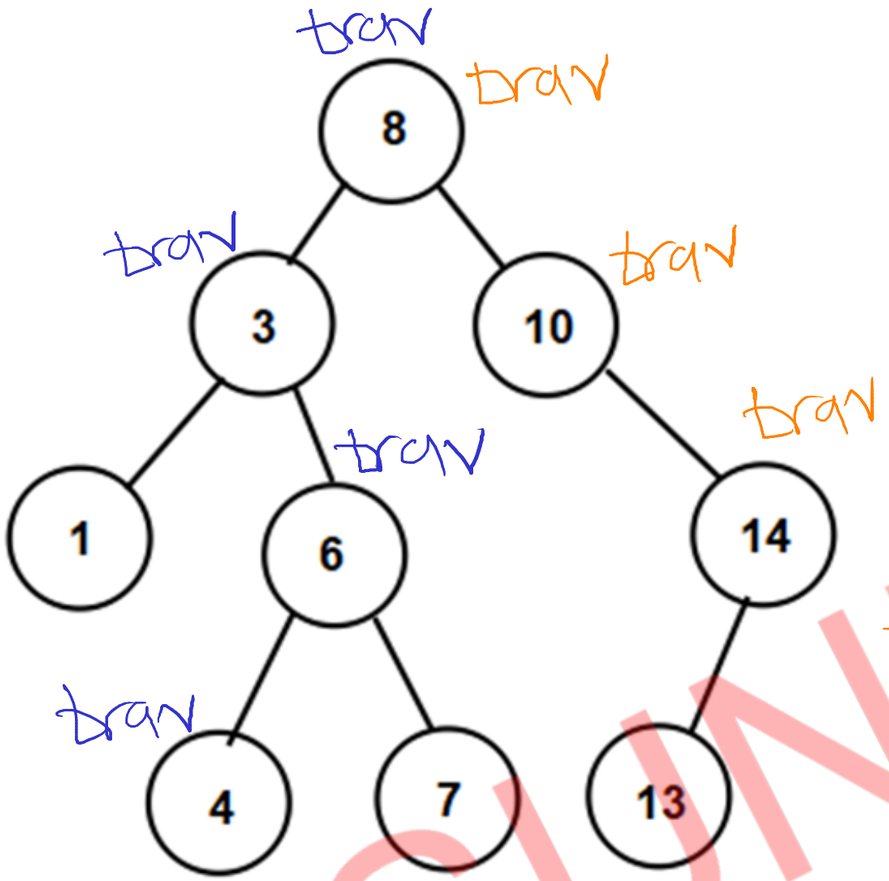
BST - Postorder Traversal



LRV

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

BST - Binary Search

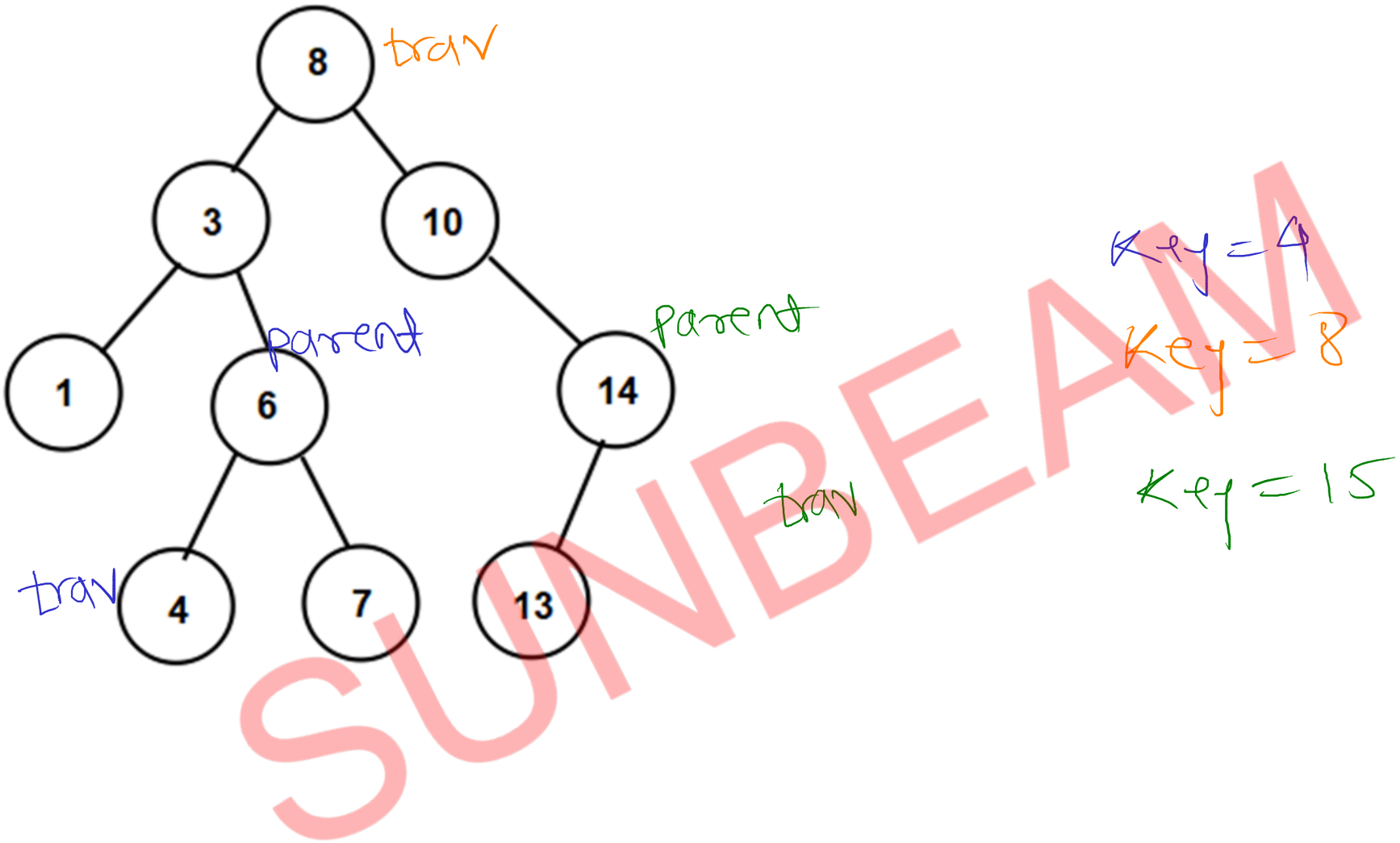



```
//1. start from root
//2. if key is equal to current data
      //return current node
//3. if key is less than current data
      // search key into left of current node
//4. if key is greater than current data
      // search key into right of current node
//5. repeat step 2 to 4 till leaf nodes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

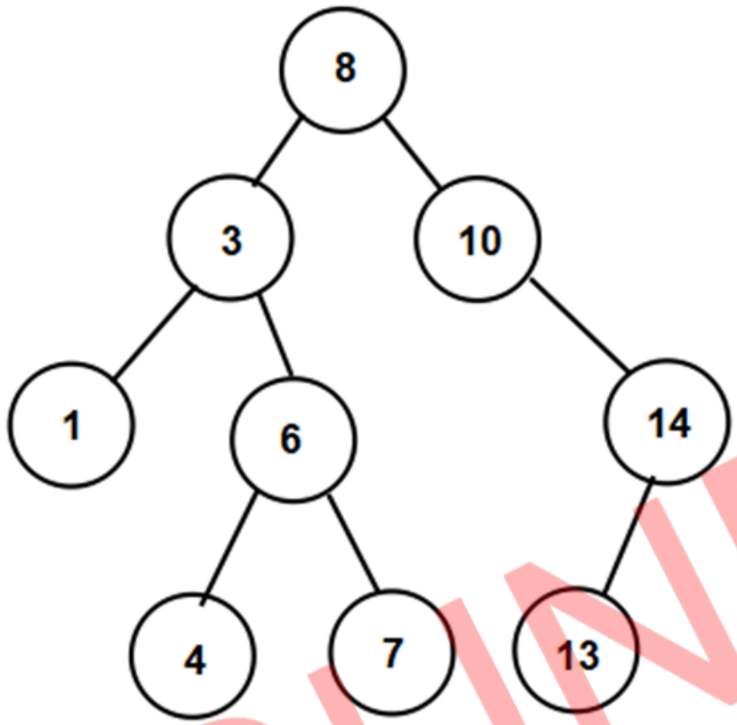
Key=4 → found

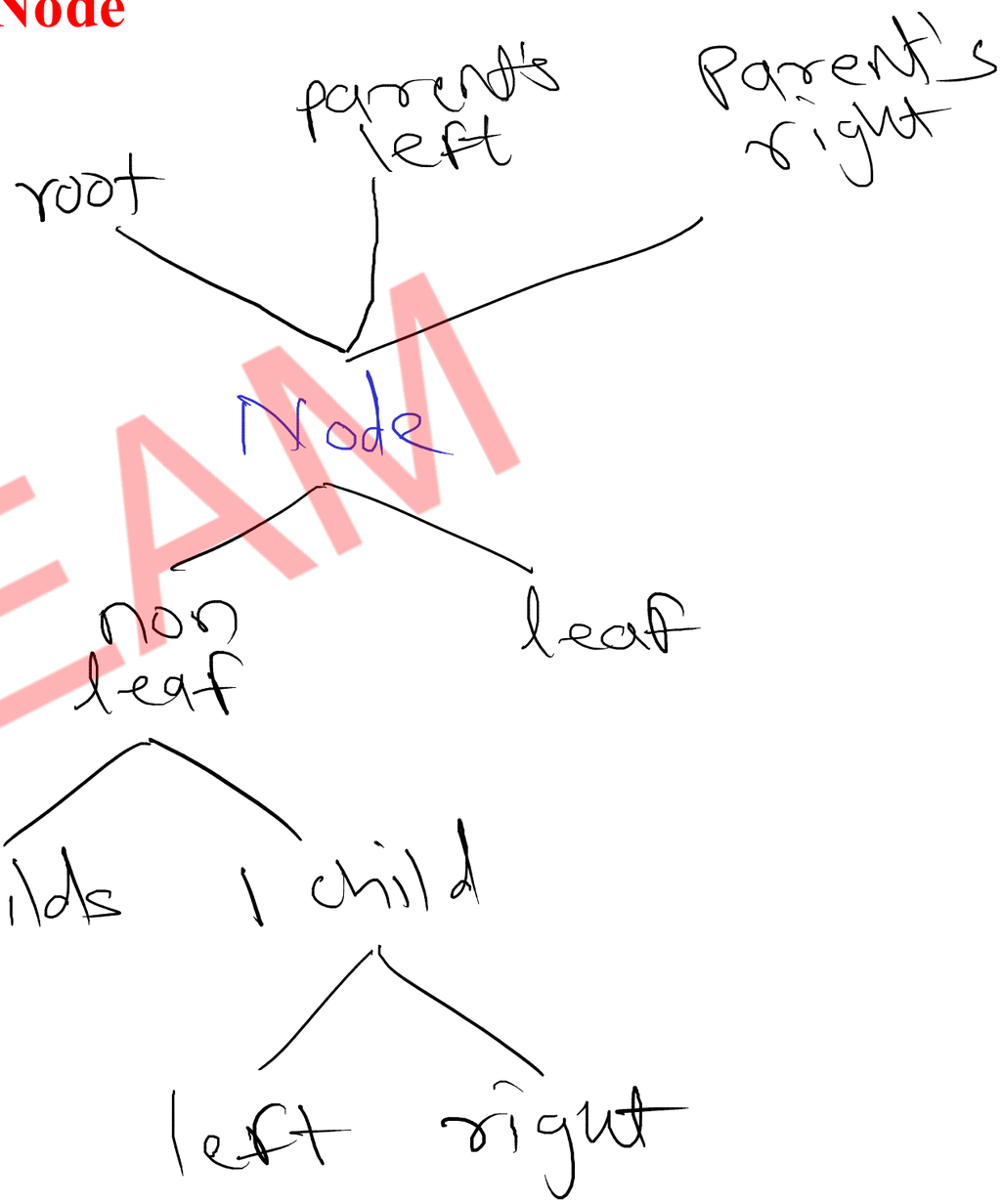
Key=15

BST - Binary Search with Parent

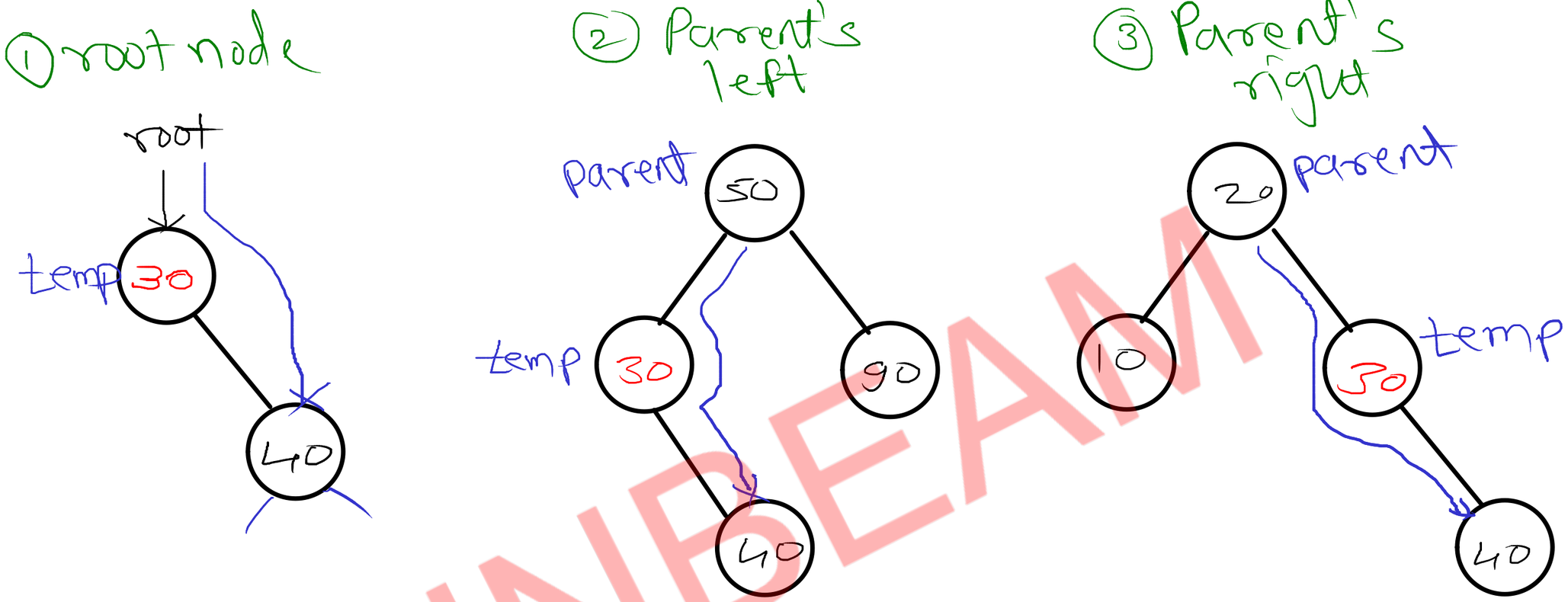


BST - Delete Node



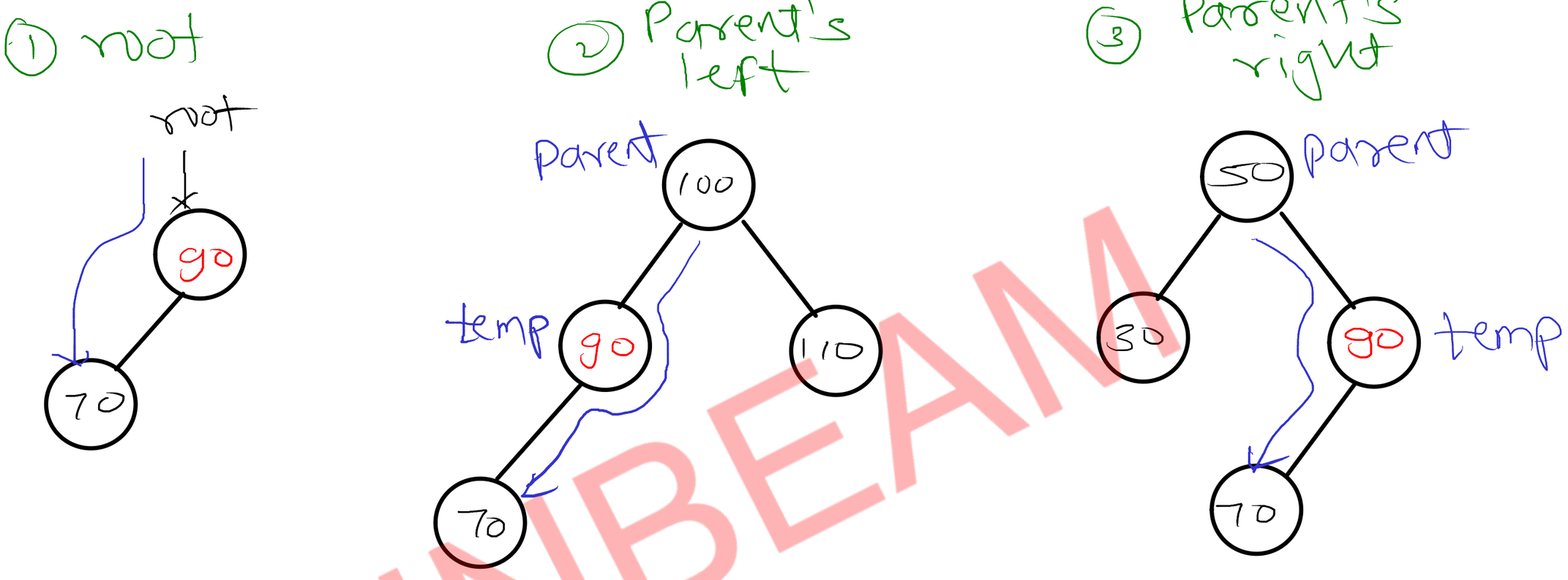


BST - Delete node which has single child (right child)



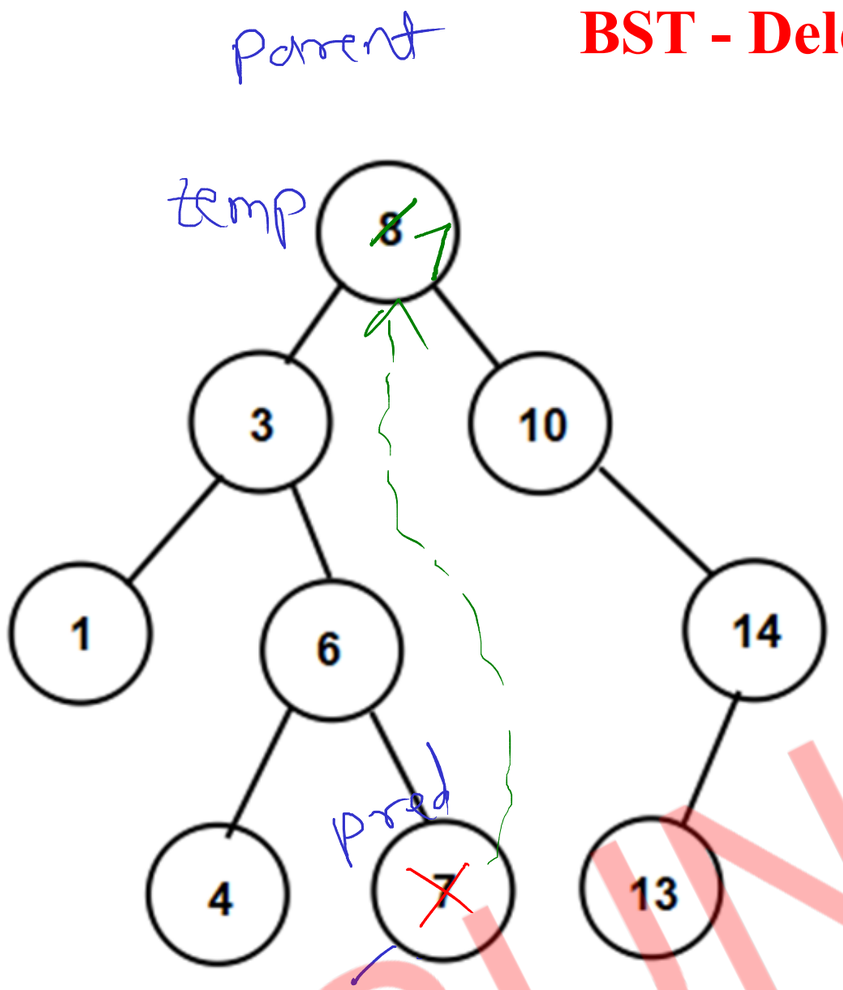

```
if(temp.left == null){
    if(temp == root)                    // root node
①       root = temp.right;
    else if(temp == parent.left)        // parent's left
②       parent.left = temp.right;
    else if(temp == parent.right)       // parent's right
③       parent.right = temp.right;

}
```

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




```
if(temp.right == null){
    if(temp == root)                        // root
        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




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

Inorder traversal - 1 3 4 6 **7** **8** **10** 13 14