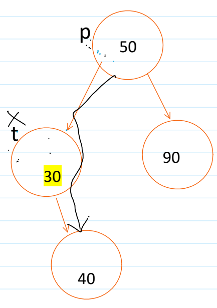
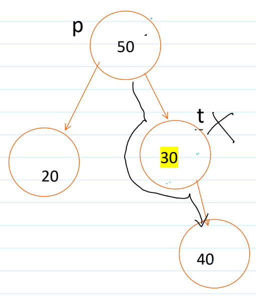
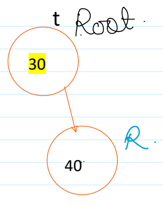
Node to be deleted has a left child as NULL   Trav->left == NULL

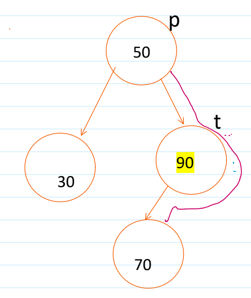


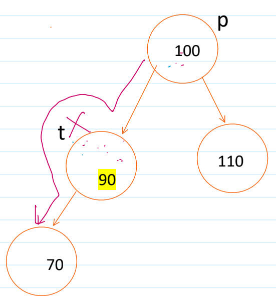


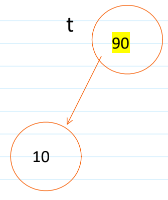


```
If(temp == root)
    Root = temp->right;
Else if(Temp == parent->left)
    Parent->left = temp->right
Else
    Parent->right = temp->right;
```

Node to be deleted has a Right child as NULL   Temp->right == NULL

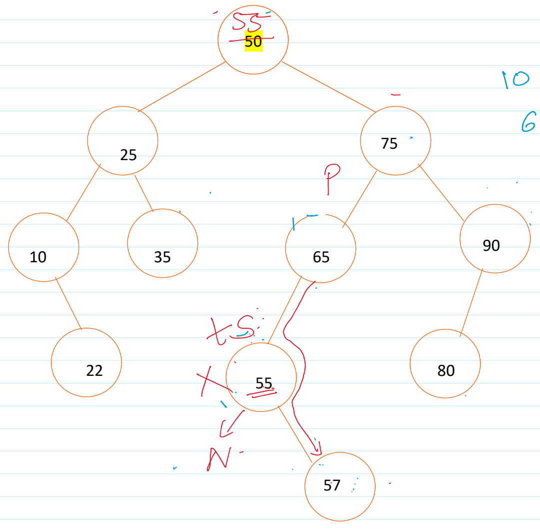






```
If(temp == root)
    Root = temp->left;
Else if(temp == parent->left)
    Parent->left = temp->left;
Else
    Parent->right = temp->left;
```

node to be deleted has both, left and right child   Temp->left != NULL && temp->right != NULL



10 22 25 35 $\boxed{50}$ 55 57
65 75 80 90

```
succ = trav->Right.
while(succ->left != NULL)
{
    parent = succ
    succ = succ->left;
}
trav->data = succ->data;
```

To delete the element that has the left and the right child, we have to find the element that can replace the element to be deleted.

To get that element, we can arrange the nodes in in-order traversal and find the in-order successor node or in-order predecessor node of the node to be deleted.

To find in-order predecessor, traverse 1 step left and go to extreme right.(till nodes right is not null)

To find in-order successor, traverse 1 step right and go to extreme left(till nodes left is not null)

To delete a node we need to find the successor or predecessor with its parent.

To find successor: with parent

```
Parent = temp
Succ = temp->right;

While(succ->left != NULL)
{
    Parent = succ;
    Succ = succ->left;
}
Temp->data = succ->data;
Temp = succ;
```