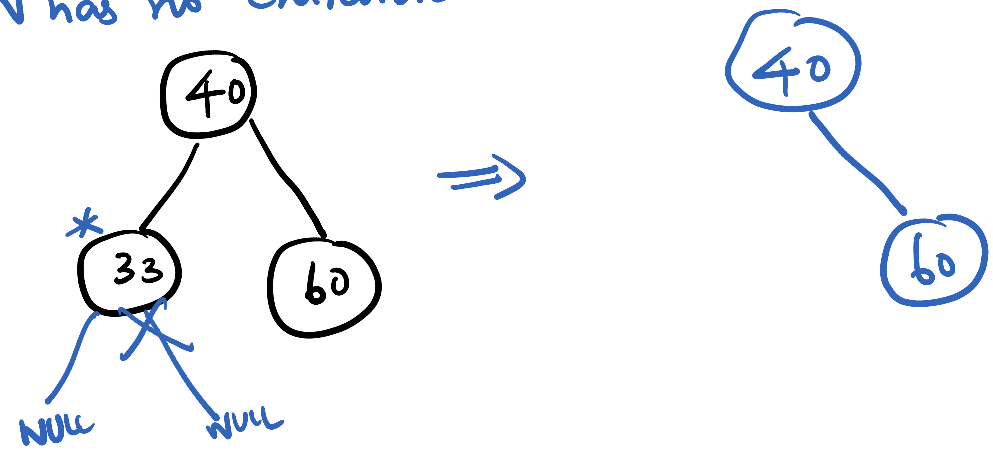
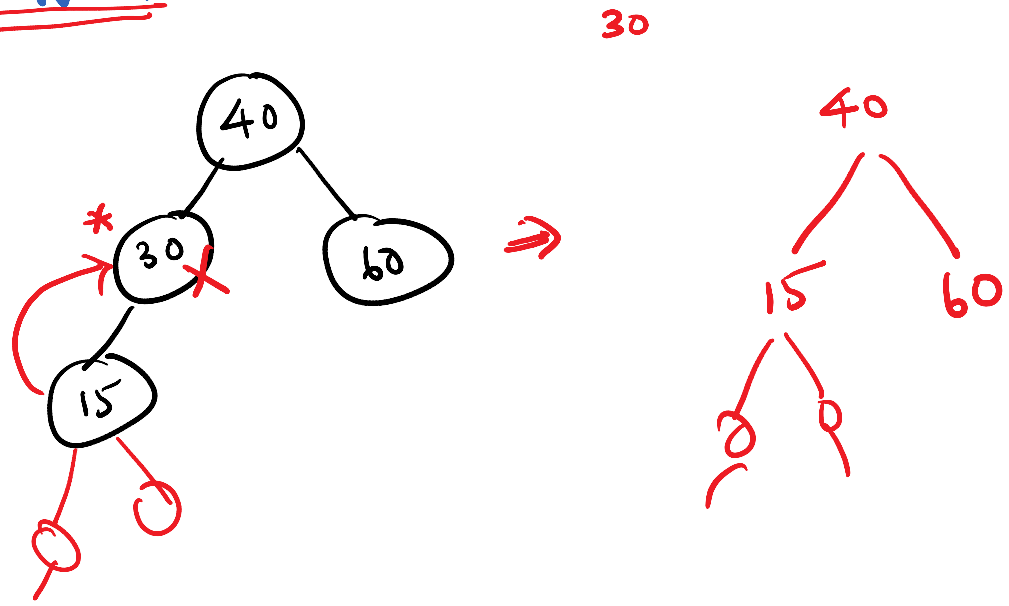
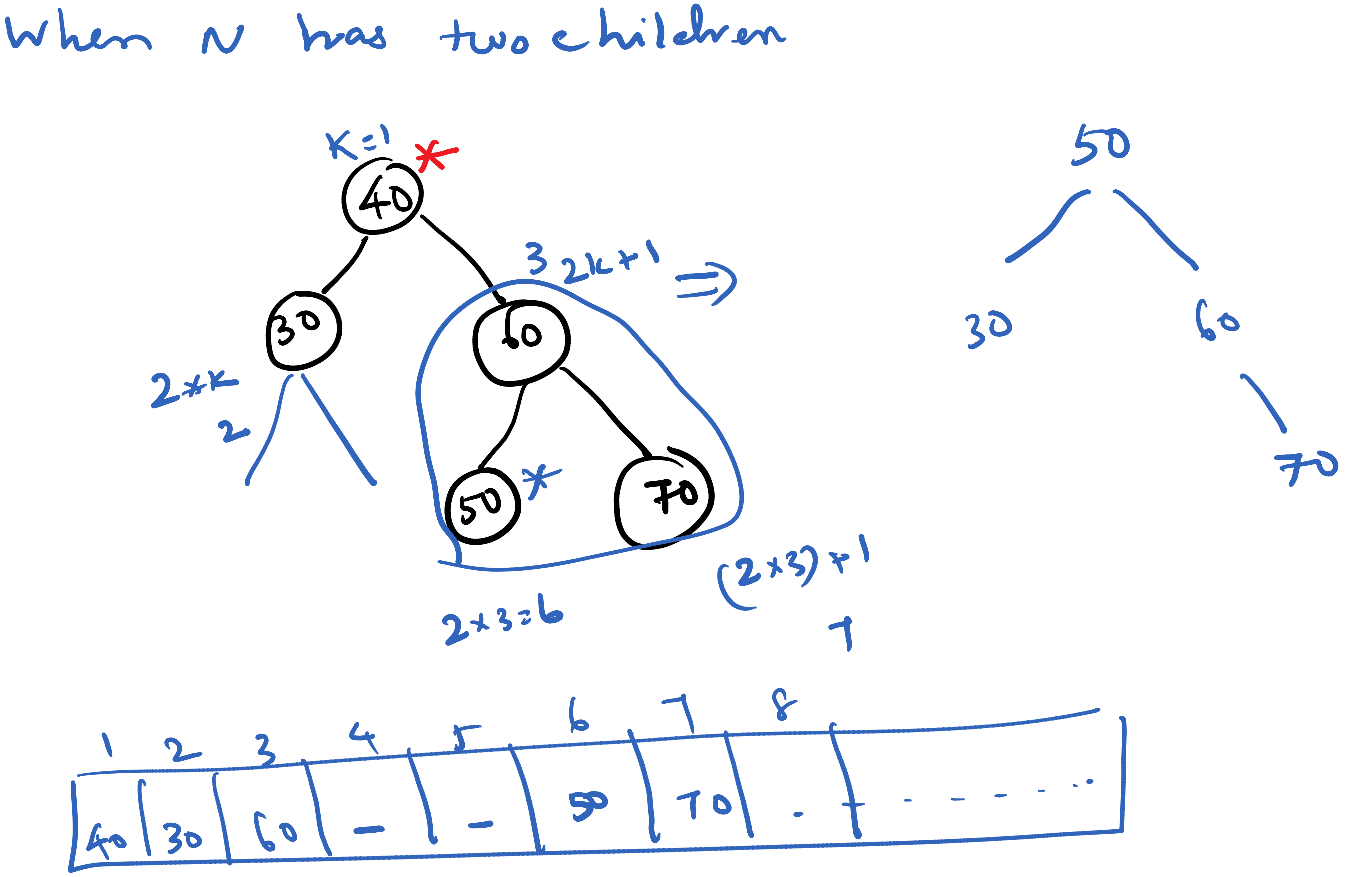
Binary Search Tree (Deleting node from BST)

When N has no children



When N has exactly one child





Algorithm to delete a node from binary search tree

1. Algorithm delete(K,r)
2. // K is a node of key to be deleted
3. {
4. if r<>null
5. if (K< r.key)
6. delete(K,r.left)
7. else if (K>r.key)
8. delete(K, r.right)
9. else if (r.right = null) and (r.left = null)
10. r = null
11. else if (r.left = null)
12. r= r.right
13. else if (r.right=null)
14. r= r.left
15. else

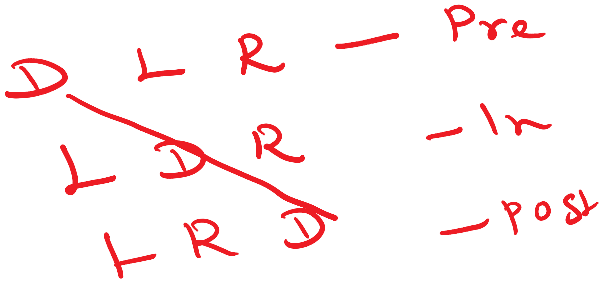
```
16.       r.key = deletemin (r.rightchild)
17.  }
```

Tree Traversal

Traversal is applicable only of binary tree. It involves examining every node in the given data

Techniques

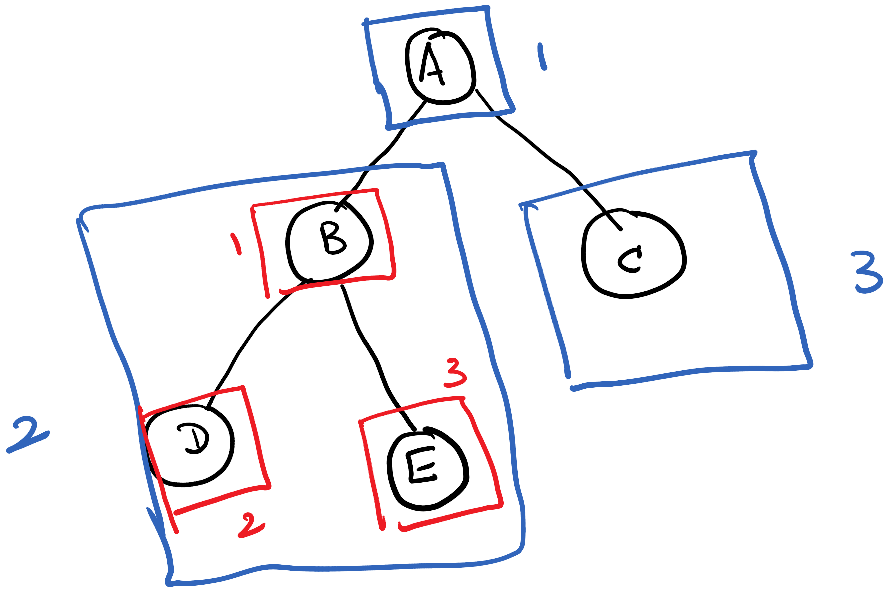
Pre-Order Traversal (DLR)
In-Order Traversal (LDR)
Post-Order Traversal (LRD)



D- Process the root node
L – Process Left subtree
R – Process Right subtree

PREORDER TRAVERSAL

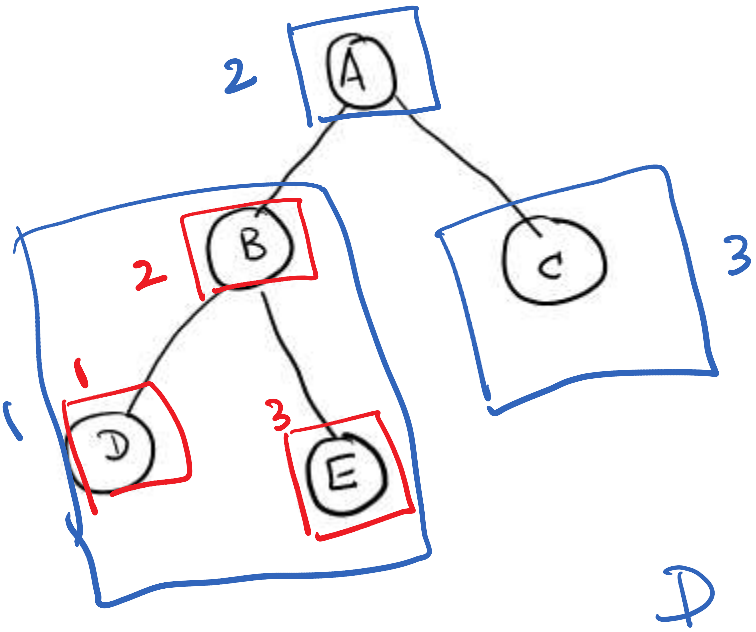
D L R



A B D E C

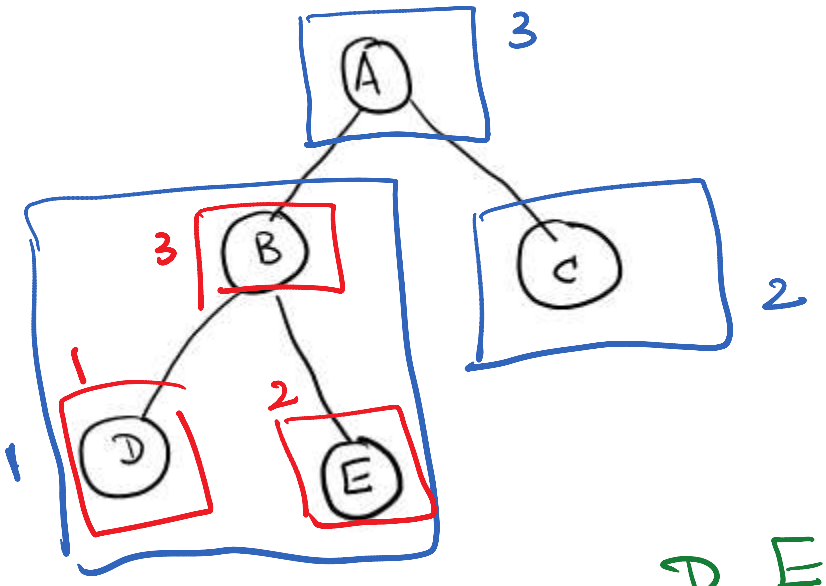
INORDER TRAVERSAL

L D R

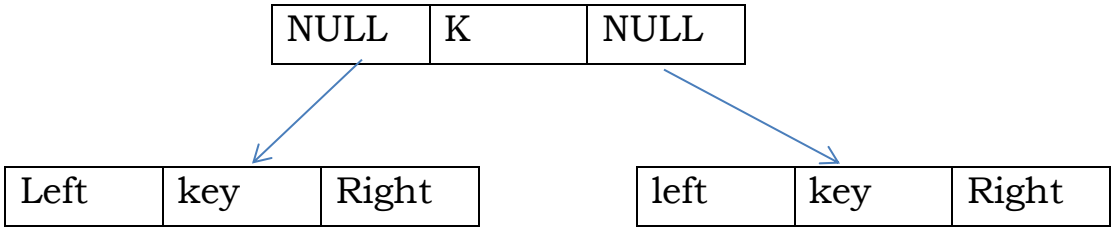


D B E A C

POSTORDER TRAVERSAL L R D



D E B C A



Algorithm to search node in BST

```
Algorithm Search(K,r)
{ // K – node to be searched, r is a pointer
if (r=Null)
     return(false)   // no node in the tree
else if ( K=r.key)
    return (true)  // node found in the tree
else if (K<r.key)
     Search(K, r.left)  // 'k' may in left subtree
else if (K>r.key)
    Search(K, r.right)  // 'k' may in right subtree
}
```

Algorithm to insert a node in a BST

```
Algorithm INSERT(K,r)
{
if (r=null)
   new(r)
   r.key = K
   r.left = null
   r.right=null
}
else if (K<r.key)
     INSERT(K, r.left)
else if (K>r.key)
    INSERT(K, r.right)
else if (K = r.key)
   Print (“Node already present”)
}
```