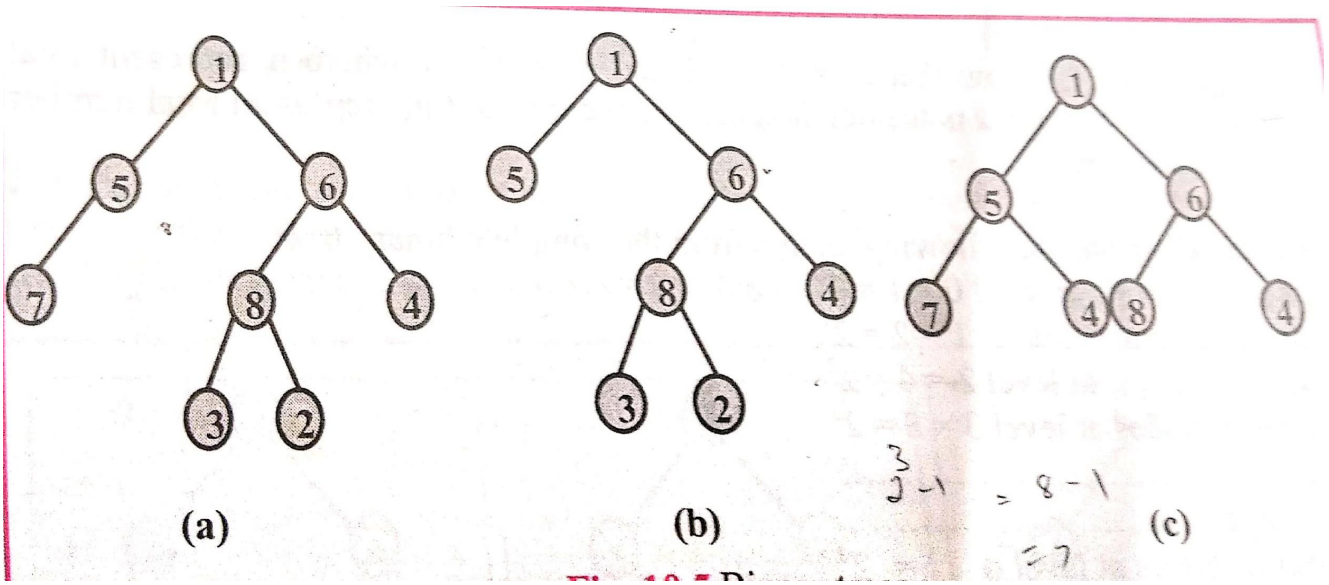


In: 7 5 1 3 8 2 6 4 **5 1 3 8 2 6 4** **7 5 4 1 8 6 4**

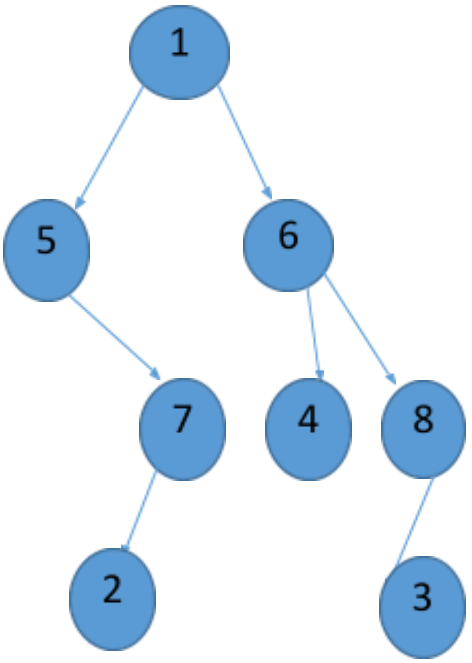
Pre: 1 5 7 6 8 3 2 4 **1 5 6 8 3 2 4** **1 5 7 4 6 8 4**

Pos: 7 5 3 2 8 4 6 1 **5 3 2 8 4 6 1** **7 4 5 8 4 6 1**

Tree Construction

```
struct node
        {
          int info;
          struct node *llink, *rlink;
        };
     typedef  struct node *NODE
  void main()
{
    NODE Root = NULL;
    //Design menu
}
```

```
NODE Insert(NODE R)
{
    NODE   NN,  CN,  PN; //  New node, child node, parent node
     char dir[10];  // to read the direction
      // create new node  with malloc
     // Read and assign info
     NN->llink = NN->rlink = NULL;
   if(R==NULL)
     return NN;
   // Read direction
    CN = R; PN = NULL; // 1
    For(i=0;i<strlen(dir); i++) // LLR
      {
          if (CN==NULL) break;
           PN = CN  // 1, 5
         If(dir[i] == 'L')   CN = CN->llink;  //  5 , NULL
             Else     CN = CN->rlink;//                    //   Inorder:  5 2 7  1 4  6 3 8
    }                  // Preorder: 1  5  7  2  6  4   8  3
    if(CN   !=NULL      ||    i != strlen(dir))
  {  // Insertion not possible  to the given direction free NN;  return R }
     If(dir[i-1] =='L')
```



```
        PN->llink = NN;
  Else
        PN->rlink = NN;
Return R;     }
```

Tree Traversals:

Depth First Traversals

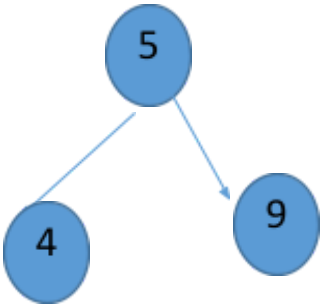
- **Preorder**
- **Inorder**
- **Post order**

Breadth First Traversals(Level Order traversal)

Depth First Traversals

- **Inorder**
- **Preorder**
- **Post order**

<u>Inorder Traversal: (Left Root Right) 4 5 9</u>

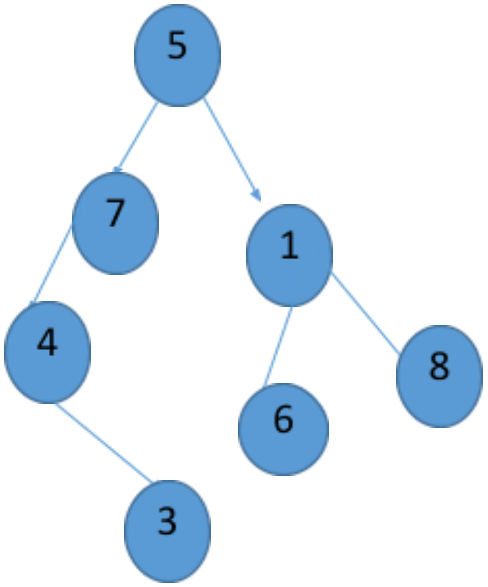


Traverse Left sub tree in Inorder

Visit the Root

Traverse Right sub tree in Inorder

```
void  Inorder( NODE Root)
{       if (Root = NULL)
        Return;
Inoder(Root->llink);
Printf("%d  ", Root->info);
Inoder(Root->rlink);
}
```



// 4 3 7 5 6 1 8

```
Inorder(5)
{
   Inoder(7);
  Printf(Root->info) // 5
  Inorder(1);
}
```

Inorder(7)

```
{
  Inoder(4);
  Printf(Root->info)//  7
   Inorder(null)
}
```

Inoder(4)

{

  Inoder(Null)

  Printf(Root->info) // 4

  Inoder(3)

}

Inoder(null)

{

} Return;

Inoder(3)

{

** Inoder(null)**

** Printf(Root->info) // 3**

Inorder(NULL)

}

Inoder(null)

{

** Return; }**

Traversing ST rooted at 1

Inorder(1)

 {

   Inorder(6);

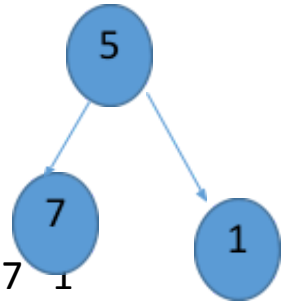
```
    Pf // 1
Inorder(8)
}
Inorder(6)
{
    Inorder(null)
    printF(root->info)// 6
  Inorder(null)
}
Inorder(8)
{
    Inorder(null)
    printF(root->info)// 8
  Inorder(null)
}
```

Preoredr Traversal:(Root Left Right)

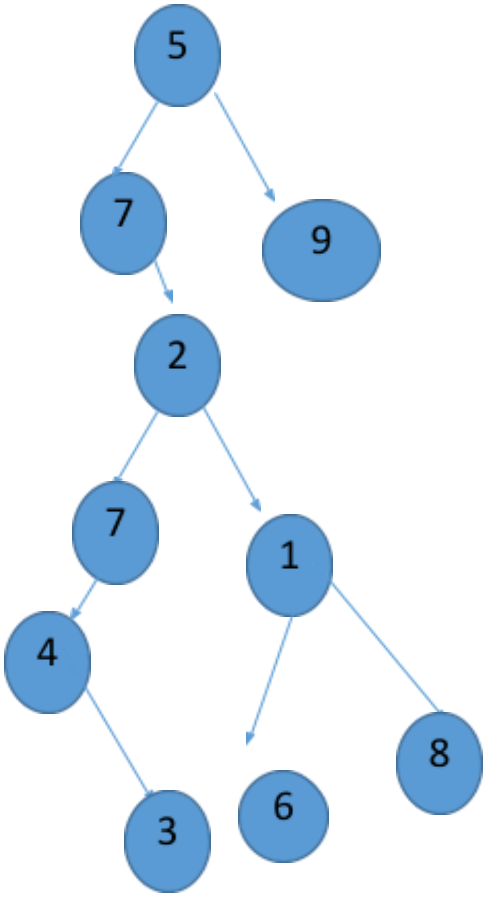
Visit the Root

Traverse Left sub tree in Preorder

Traverse Right sub tree in Preorder



5 7 1




```
void Preorder( NODE Root)
{    if(Root==NULL)
        Return;
Printf("%d  ", Root->info);
Preoder(Root->llink);
Preoder(Root->rlink);
}
        //  5 7  2  7  4  3 1  6  8   9
```

Postoredr Traversal:(Left Right Root) // 7 1 5

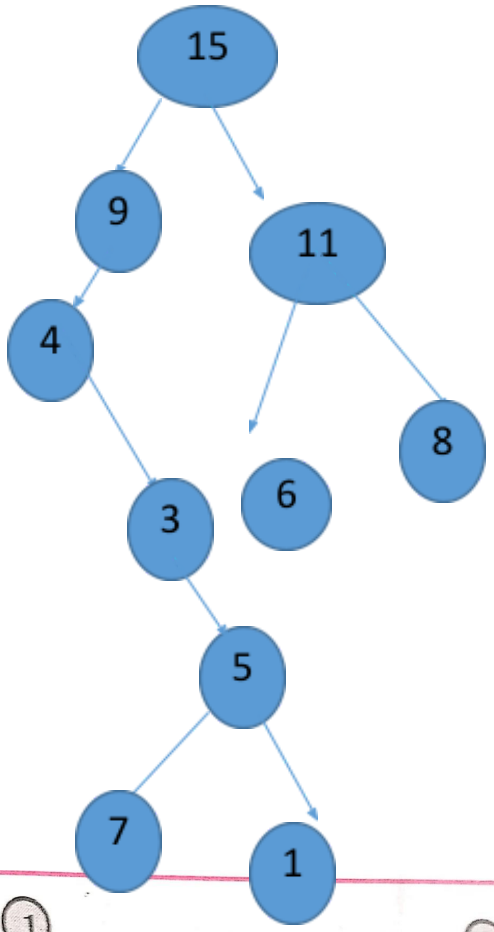
Traverse Left sub tree in Postorder

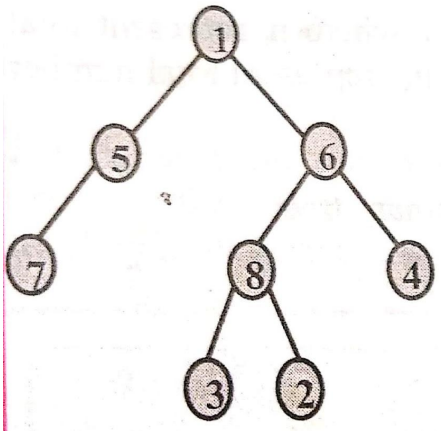
Traverse Right sub tree in postorder

Visit the Root

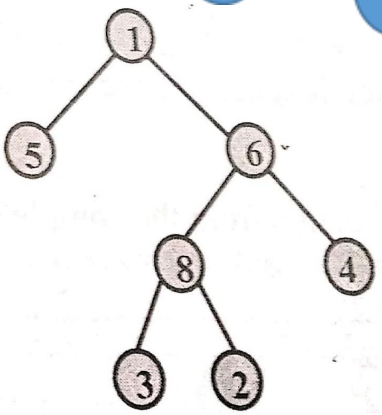
```
void Postorder( NODE Root)
{   if(Root==NULL)
        Return;

Postoder(Root->llink);
Postoder(Root->rlink);
Printf("%d  ", Root->info);}
}   // 7  1 5  3 4  9  6 8 11  15
```

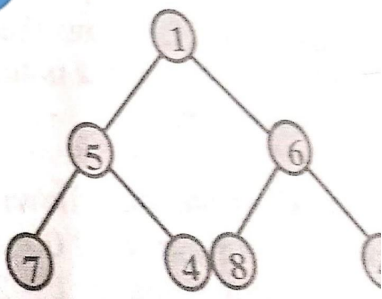




(a)



(b)

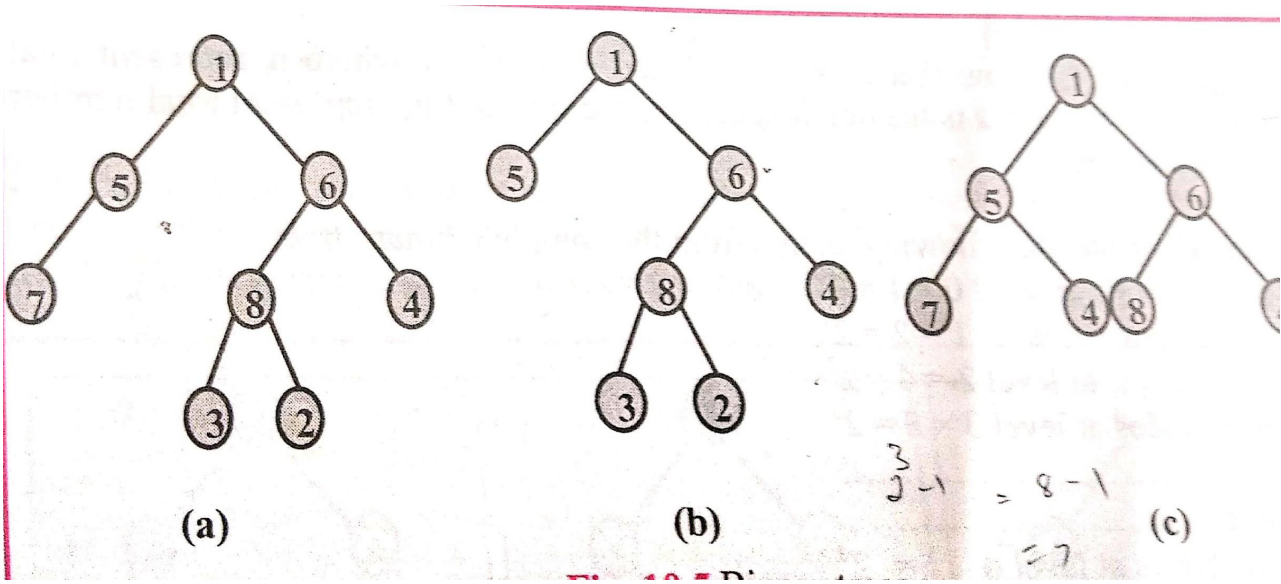


(c)

	6
	5
	4
	3
	2
	1
	0

7 5 1 3 8 2 6 4

	6
	5
	4
	3
	2
	1
	0

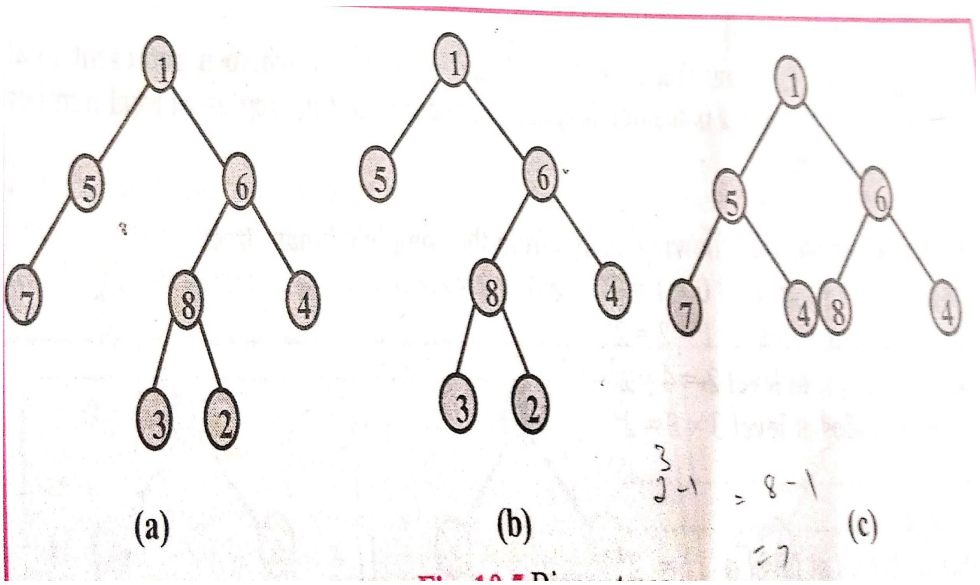


C	6
	5
	4
	3
3	2

2	1
4	0

Level Order Traversal

	7
	6
	5
	4
	3
	2
	1
1	0



1 5 6 7 8 4 3 1 5 6 8 4 3 2 1 5 6 7 4 8 4

Q.f=0 Q.r= -1

Insert(Q, root)

While(!Isempty(Q))

Insert(Q, left(Q.front))

Insert(Q, right(q>front))

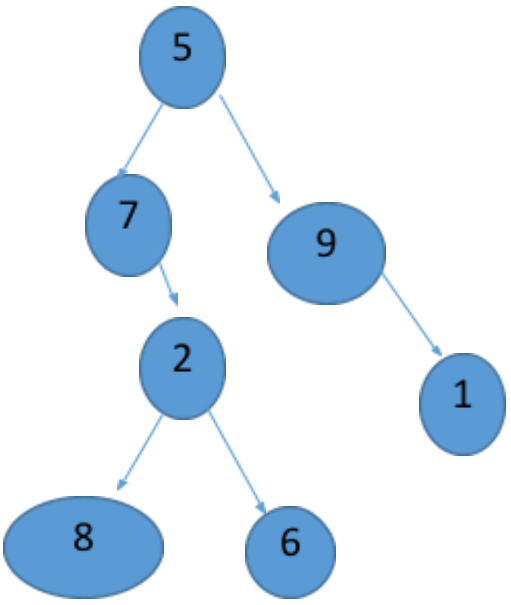
Dqueu(Q)

Search Operation:

NODE Search(NODE Root, int key) // Preorder

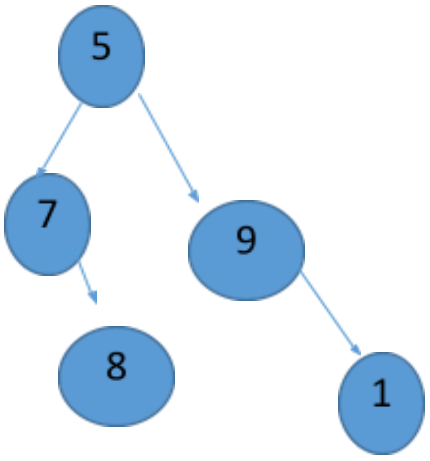
{ NODE NS;

```
If(Root==NULL) return Root;
If(R->info == key)
  NS = R;
If (NS ==NULL)
  NS = Search( Root->llink, key)
If(NS==NULL)
  NS =Search(Root->rlink, key);
  Return NS
}
```



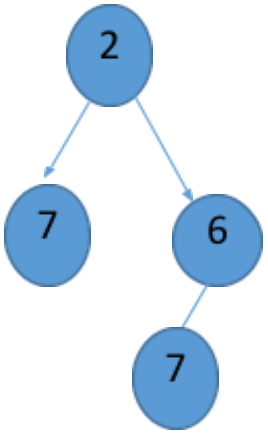
Creating Copy

```
NODE Copy(NODE Root)
{    NODE NN
  // Base case  if Root == NULL return NULL
  // Create NN
   NN->info = Root->info;
 NN->llink = copy(Root->llink)
  NN->rlink = copy(Root->rlink)
  Return NN;
  // general case
}
```



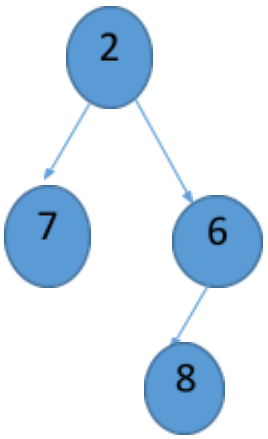

```
copy((5)
{
        NN->info = 5
        NN->llink = copy((7) // 5->llink = 7
        NN->rlink = copy((9)) // 5->rlink = 9
         Retun 5
        }
        Copy((7))
        {
            NN->info=7
            NN->llink = copy(null) // 7->llink = null
```

```
    NN->rlink = copy((8))  // 7_rlink = 8
    Return 7
}
Copy((8))
{
  NN->info = 8
   NN->llink = copy(null) // 8_llink = null
  NN->rlink = copy(null)// 8->rlink = null
  Return 8
}
```

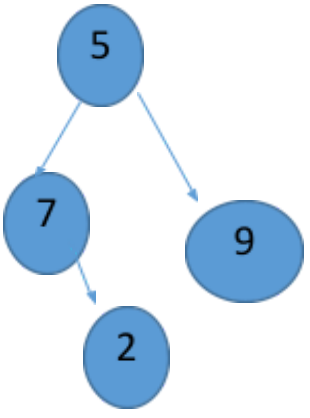


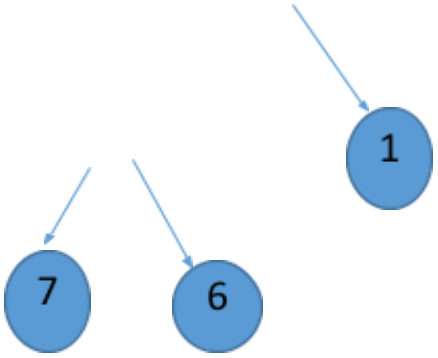
To find Height:

```
 Int Height(NODE Root)
{
  If (Root== NULL ) return -1;
Return 1 +  Max( Height(Root->left,) Height( Root->right))
}
```




```
 Height((2)
{
    return 1 + max(  HT(7), HT((6))
    //    1  +  max( 0, 1) = 2
    }
    HT((7))
    {
      Return 1 + max(HT(NULL), HT(NULL)) //  1 + max( -1, -1) =  0
    }
    HT((6))
    {
      Return 1 + max(HT(8), HT(NULL)) //  1 + max( 0, -1) =  1
    }
    HT((8))
    {
      Return 1 + max(HT(NULL), HT(NULL)) //  1 + max( -1, -1) =  0
    }
```





Delete Operation:

Case 1: If node to be deleted is a leaf.

Delete by setting its parent's left or right link based on whether it is left or right child to NULL.

Case 2 : If node to be deleted is a non leaf having one child(left or right)

Delete by setting its parent's left or right link based on whether it is left or right child to left link if left chid exists otherwise to right link.

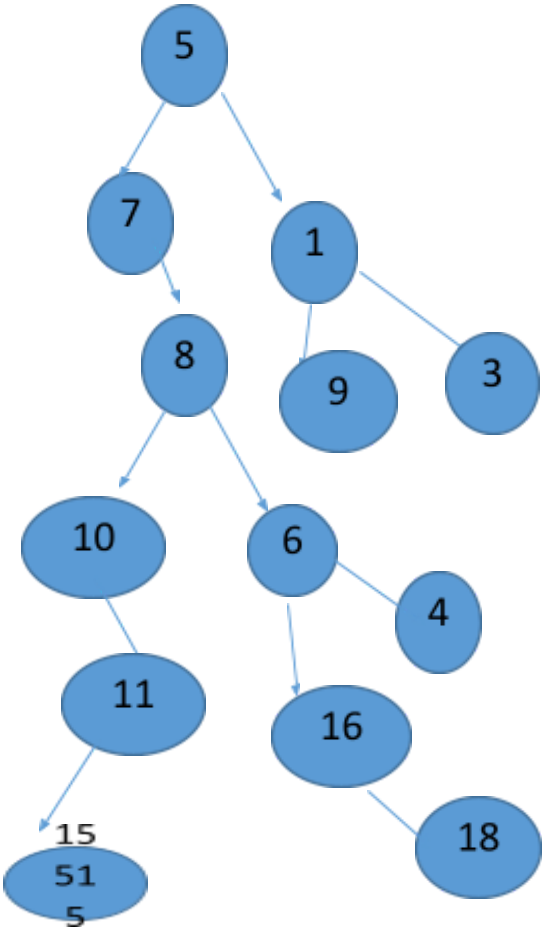
Case 3 : If node to be deleted is a non leaf having both the children

Replace the node to be deleted with its inorder successor and delete this inorder successor node.

Inorder successor of the node to be deleted is the **leftmost node in its right subtree .**

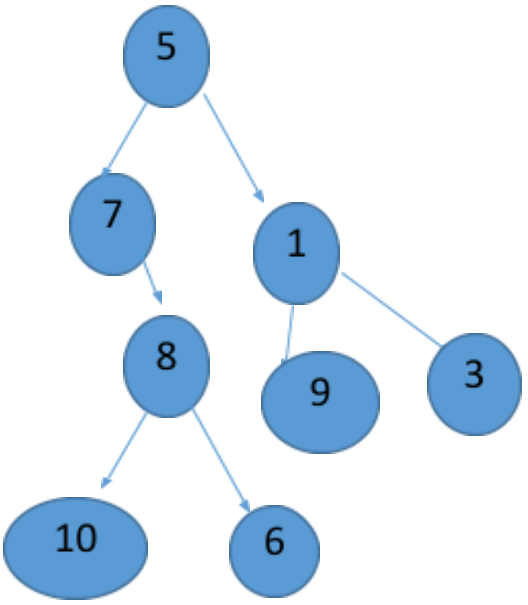
Inorder successor may be a leaf or non leaf. If it is a non leaf, it will have only one child i.e right child .

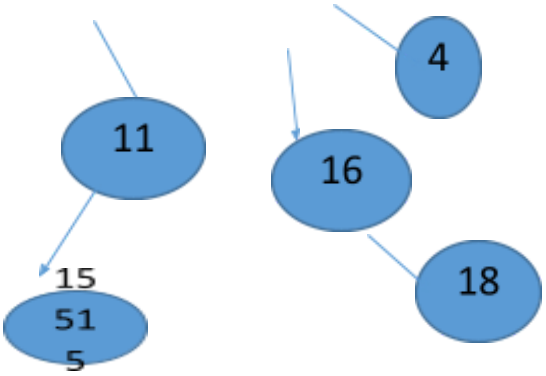
Deleting this is same as case 2.



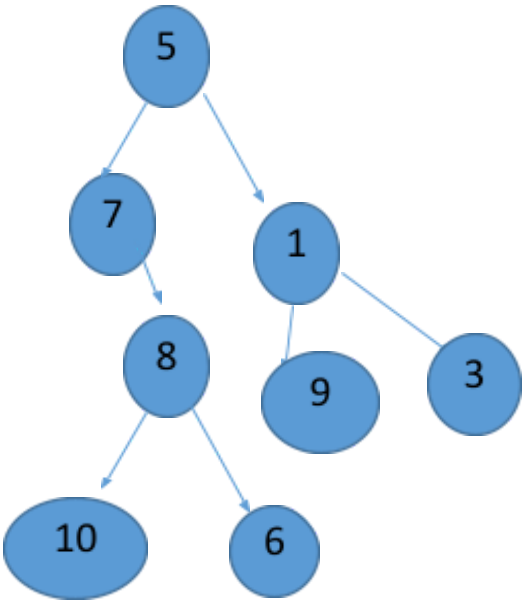
Inorder: 7 10 15 11 8 16 18 6 4 5 9 1 3

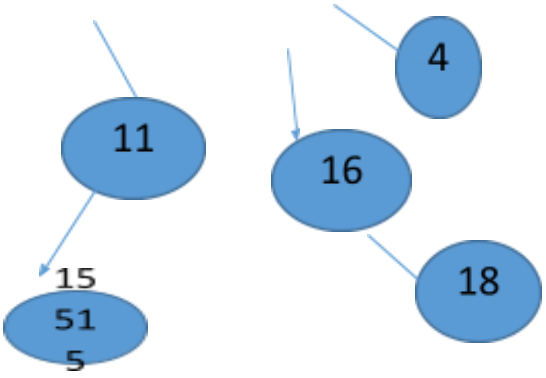
Delete: 15, 18, 9, 3, 4(Leaf nodes)

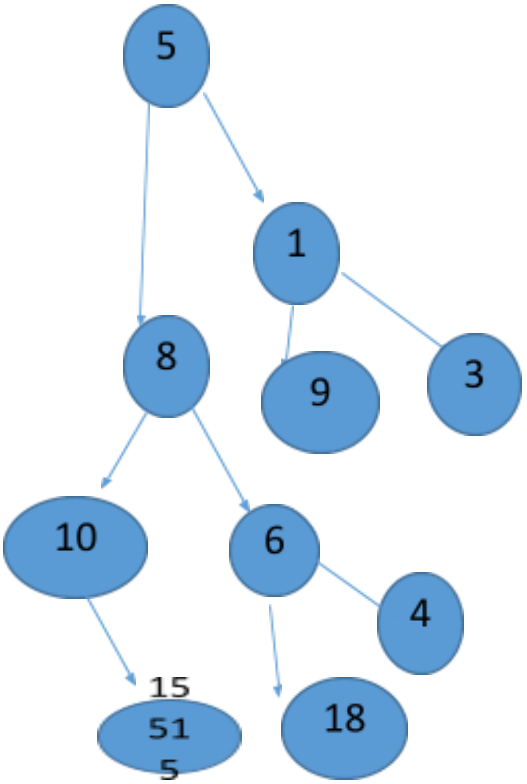




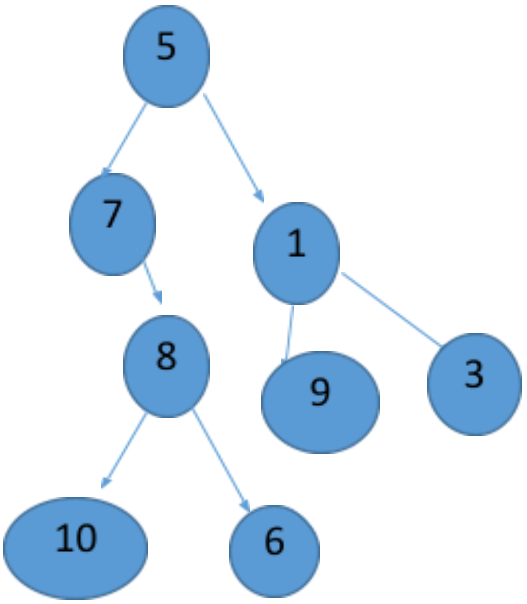
Deleting : 11, 16, 7

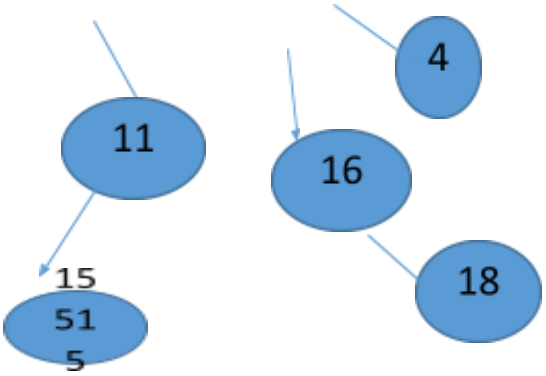


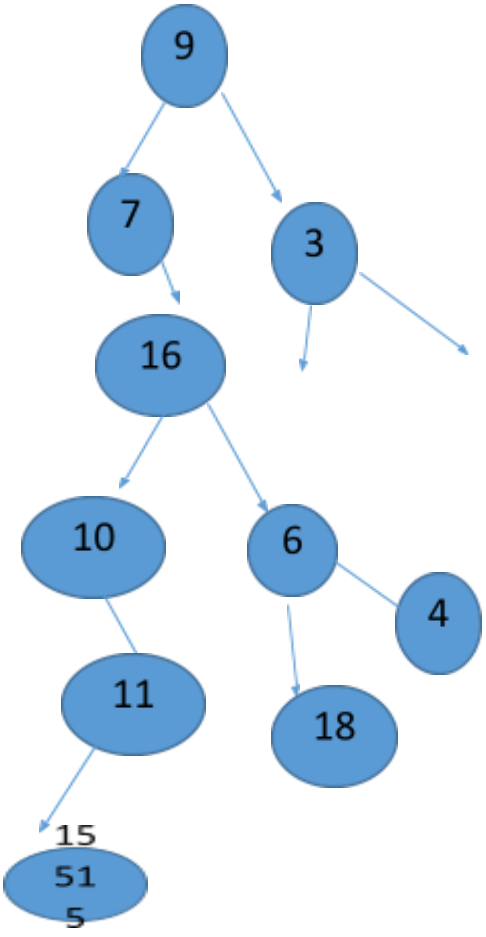




Deleting: 8 , 1, 5




4
11
16
18
15
51
5


9
7
3
16
10
6
4
11
18
15
51
5

Delete 15:

```
Delete(R, 15)
{
  search(R, 15)
    ND, PN
    If(ND->llink==NULL && ND->rlink==NULL)  // case 1
      {
         If(ND == PN->rlink)   PN->rlink=NULL
        Else PN->llink=NULL;
   }
```

```
//Case 2
If(ND->rlink!=NULL && ND->llink==NULL) || ND->rlink==NULL &&
ND->llink!=NULL)
{
      If (ND->rlink==NULL)
        {
         If (ND== PN->llink)
             PN->llink =  ND->llink ;
        Else PN->rlink = ND->llink ;
        Free(ND);
    }
  Else
           {
          If (ND== PN->llink)
             PN->llink =  ND->rlink;
        Else PN->rlink = ND->rlink ;
        Free(ND);
    }
} //  case 2

    Else
//Case3   if non leaf having both left and right children

{
```

```
    IS = ND->rlink;  // Inorder Successor
   PN = ND; // Parent of IS
    While(IS->llink !=NULL)
     {
        PN = IS;
         IS = IS->llink;
   }
ND->info = IS->info;
// Delete Is by checking wther it is leaf or non leaf
//If leaf case1
//If non leaf( only right child) case2
}




}
```