Inorder algorithm:- (LNR)

```
algo inorder (root)
    if (root ≠ null && root → leftlink ≠ NULL)
    {   inorder (root → leftlink
    }
    if (root ≠ NULL)
    {   process (root → info)
    }
    if (root ≠ null && root → rightlink ≠ NULL)
    {
        Inorder (root → rightlink)
    }
```

Preorder: (NLR)

```
algo- preorder (root)
if (root ≠ NULL)
{
  process (root → info)
}
if (root ≠ NULL & root → leftlink ≠ NULL)
{
  preorder (root → leftlink)
}
if (root ≠ NULL & root → rightlink ≠ NULL)
{
  preorder (root → rightlink)
}
```

post order: (LRN)

```
algo postorder (root)
    if (root ≠ null & root->left link ≠ NULL)
    {
        postorder (root → left link)
    }
    if (root ≠ NULL & root → right link ≠ NULL)
    {
        Postorder - traversal (root → right link)
    }
    if root ≠ NULL
    {
        process (root → info)
    }
```

Searching in Binary Search Tree.

```
algo_bst search (root, item, par, loc)
{
    if (root == null)
    {
        loc = null, par = null
        return
    }
    if (root -> info == item)
    {
        loc = root, par = null
    } return;
    if (root -> info > item) ptr = root -> left
    else ptr = root -> right
    temp = root
    while (Ptr ≠ null)
    {
        if (Ptr -> info == item)
        { loc = Ptr, Par = temp
          return
        }
```

NOTES

APPOINTMENTS

```
        temp = ptr
08.00   if (ptr → info ≠ item) ptr = ptr → left
        else ptr = ptr → right
09.00   }
10.00   loc = null , par = temp
        return
11.00   }
```

12.00

Insertion in BST

13.00

```
14.00   algo_ bst insert (root, newinfo)
15.00   {
        Call bst_search (root, newinfo, par, loc)
16.00   if (loc ≠ null) return
17.00   newnode = getnode ( )
        newnode → info = newinfo
18.00   newnode → llink = null
        newnode → rlink = null
```

SUNDAY 23

```
if (par == null)
 {
   root = newnode
   return
 }
if (par->info < newinfo)
 {
   par -> left = newnode
 }
else
 {
   par -> right = newnode
 }
}
```

Deleting a node from a BST:

```
algo del-bst (root, item)
{
  Call find (root, item, par, loc)
  if (loc == null) return
  if (loc -> right ≠ Null and loc -> left ≠ NULL)
  {
    Call del.two.bst (root, loc, par)
  }
  else
  {
    Call del.one-bst (root, loc, par)
  }
  freenode(loc)
  return.
```

Delete a node with one or zero child'.

```
algo_delOnebst(root, loc, par)
{ if(loc→left == null & loc→right == null)
  { child = = null
  }
  else if (loc→left ≠ null
  { child = loc→left
  }
  else
  { child = loc→right
  }
  if (par ≠ null)
  { if(loc = = par→left) par→left = child
    else par→right = child.
  }
  else
  { root = child
  }
  return
}
```

Delete a node from BST with 2 child'.

```
algo del_twobst (root, loc, par)
{
  ptr = loc -> right
  save = loc
  while (ptr -> left != null)
  {
    save = ptr
    ptr = ptr -> left
  }
  suc = ptr, parsuc = save

Call del_onebst (root, suc, parsuc)
if (par != NULL)
  if (loc == par -> left)
    par -> left = suc
  else
    par -> right = suc
else
    root = suc
suc -> left = loc -> left
suc -> right = loc -> right
return.
```