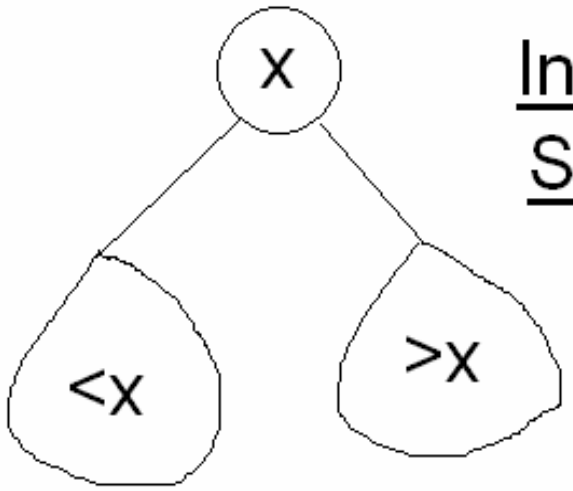
Binary Search Trees



Inorder Trav:
Sorted seq

```
int search_bst(int x, NODEPTR tree)
{       NODEPTR p;
        int found;
        found=0; p=tree;
        while ((found !=1)&&(p != NULL)){
            if (p->info==x)  found=1;
            else if (p->info<x) p=p->right;
                    else                p=p->left;
        }/* end while */
        if (found==1) return p;
        else    printf ("Key not present \n");
}
```

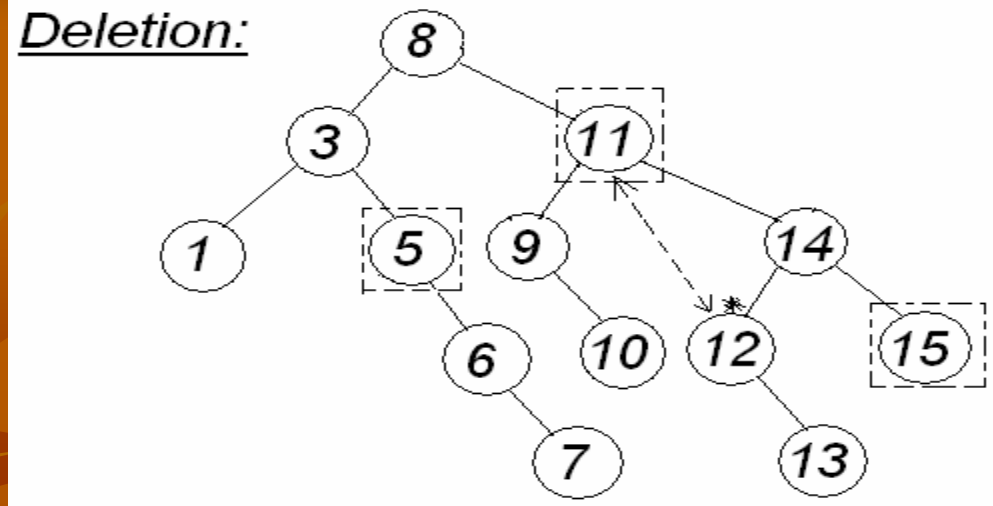
```
NODEPTR maketree (int x)
{       NODEPTR p;
        p=getnode();
        p->info=x;
        p->left=NULL;
        p->right=NULL;
        return p;
}/* end maketree */
```

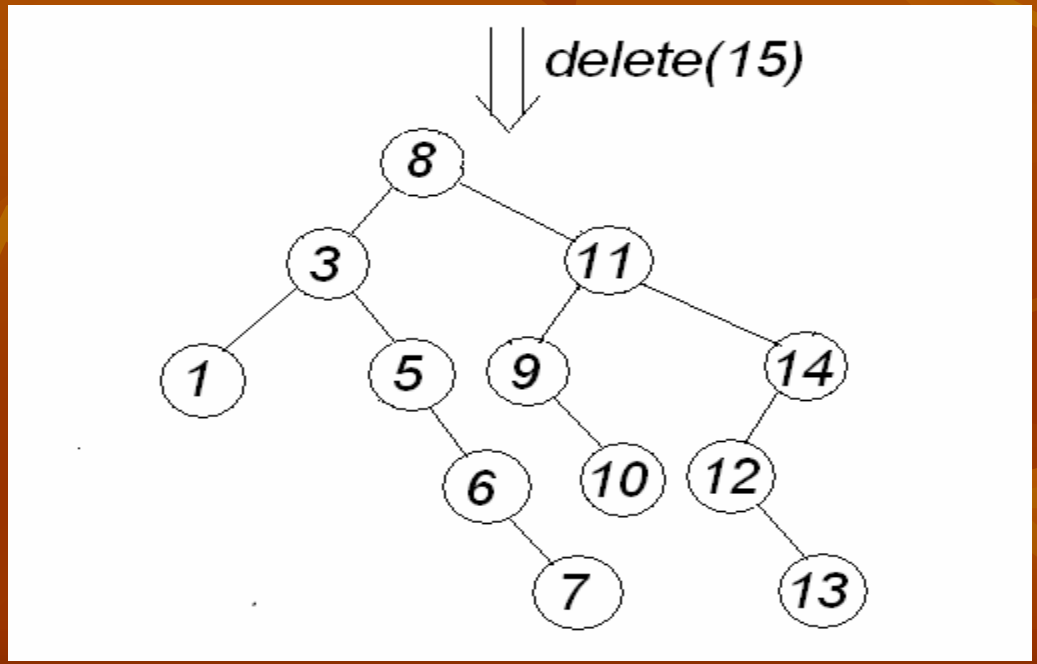
```
void setleft (NODEPTR p, int x)
{
        if (p==NULL)
                printf (“void insertion \n”);
        else if (p->left != NULL)
                printf(“invalid insertion \n”);
              else  p->left = maketree (x);
}/* end setleft */
```

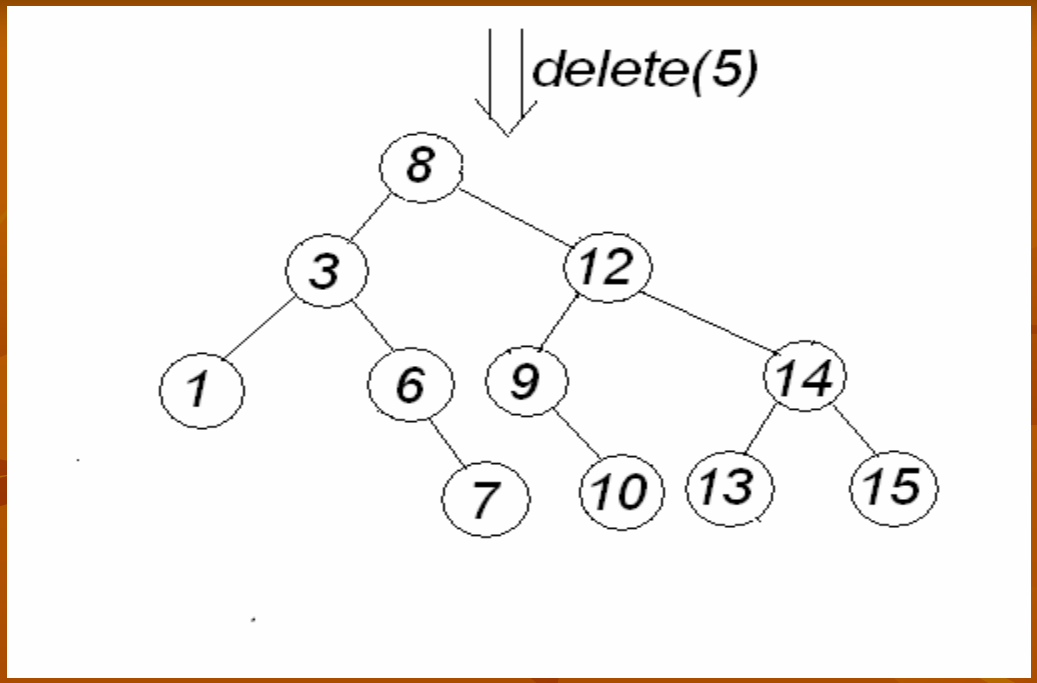
```
void setright (NODEPTR p, int x)
{
        if (p==NULL)
                printf (“void insertion \n”);
        else if (p->right != NULL)
                printf (“invalid insertion \n”);
              else  p->right = maketree (x);
}/* end setright */
```

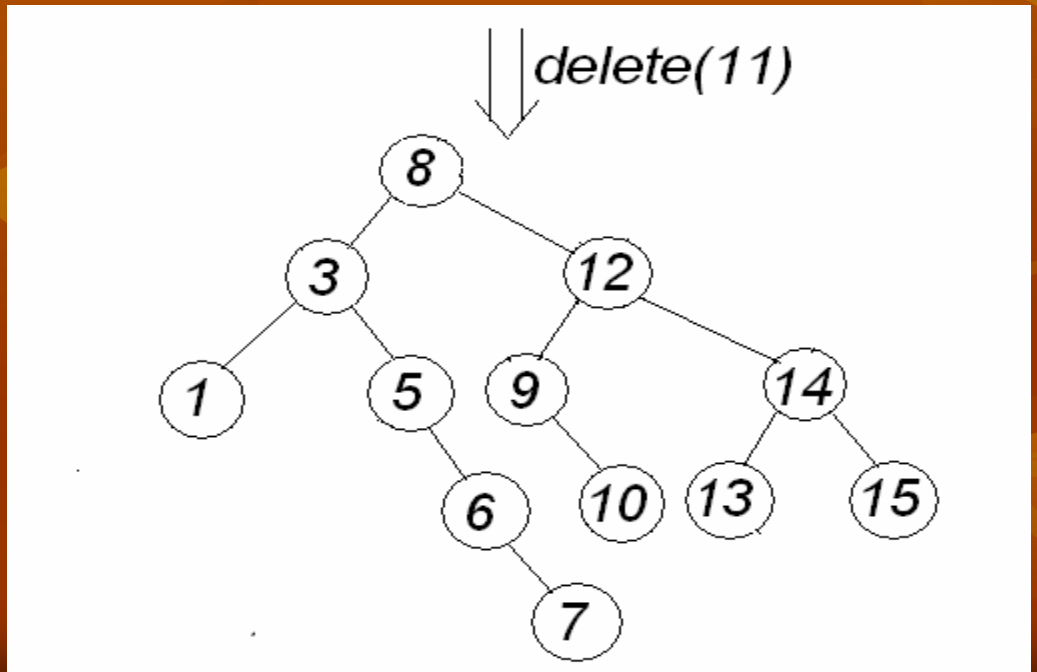
```
NODEPTR insert ( NODEPTR tree,  int number)
{       NODEPTR p, q;
        if (tree==NULL){
                tree=maketree(number);
                return tree;
        }else { p=q=tree;
                while ((number!=p->info) && (q!=NULL)){
                   p=q;
                   if (number<p->info) q=p->left;
                   else  q=p->right;
                }/* end while */
                if (number==p->info)
                     printf ("%d is duplicate \n", number);
                else if (number<p->info) setleft (p, number);
                     else                setright (p,number);
                return tree;
               }
}
```

Deletion:






delete(5)
8
3
12
1
6
9
14
7
10
13
15


delete(11)
8
3
12
1
5
9
14
6
10
13
15
7

```
NODEPTR Delete(int key,  NODEPTR T)
{
    NODEPTR  temp;
    if (T == NULL)
      printf (“ERROR: Element not found.\n”);
    else
      if (key < T ->info) /*Go Left */
            T->left = Delete(key, T->left);
      else
      if (key > T->info) /* Go Right */
            T->right = Delete(key, T->right);
      else /* Found element to be deleted */
```

```
    if (T-> left && T-> right) /* Two Children */
     {
      /*Replace with smallest in right subtree */
      temp = FindMin(T-> right);
      T-> info = temp -> info;
      T -> right = Delete( T -> info, T -> right);
     }
      else /* One or Zero Children */
      {
        temp = T;
        if ( T -> left ==  NULL)  /*Also handles 0 children */
                T = T -> right;
        else if (T -> right == NULL)
                 T = T -> left;
        free ( temp );
       }
      return T;
} /* End Delete */
```