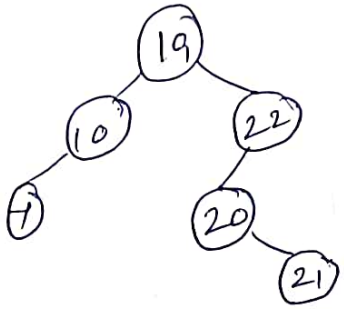
# UNIT - II

→ Binary search Tree:

→ Left subtree – smaller value than root.

→ Right subtree – larger value than root




* Difference b/w BT & BST

Basic operations:

Insertion, deletion, find, findmin, findmax, makeempty.

```
struct TreeNode
{
  int element;
  searchtree left;
  searchtree right;
};
```

Makeempty:

→ mainly for initialization

→ initialize the first element as one node tree

```
searchtree Makeempty ( searchTree T)
{
  if (T != NULL)
  {
    Makeempty (T → left);
    Makeempty (T → Right);
    free (T);
  }
  return NULL;
}
```

Insert:

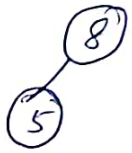
→ Insert the 'x' element into the tree.
→ check with the root node tree
→ if x < Root → insert into leftside T→left == NULL
x > Root - " " rightside T→Right == NULL

Eg: 8, 5, 10, 15, 20, 18 Construct BST

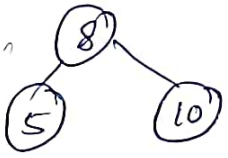
① Insert '8'



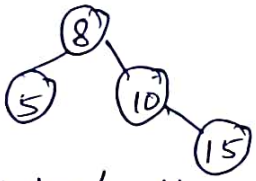

② Insert 5
* Compare with 8,
* 5 < 8 insert left




③ Insert '10'
* Compare with root '8'
* 10 > 8 insert into Right




④ Insert '15'
* Compare with root '8', 15 > 8, so traverse right subtree, again Compare '10' 15 > 10, insert 15 in right subtree

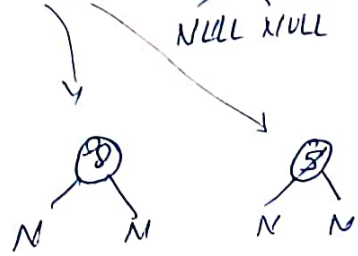



⑤ Insert '20' < '18'

5.
8

Routine:

```
Searchtree Insert ( int x, searchtree T)
{
  if (T==NULL)
  {
    T= malloc ( sizeof ( struct Treenode));
    if (T!=NULL)
    {
      T→ element = x;
      T→ left = NULL;
      T→ Right = NULL;
    }
  }
```



```
else if (x < T -> element)
    T -> left = Insert (x, T -> left);
else if (x > T -> element)
    T -> right = insert (x, T -> right);
return T;
}
```

Eg: 3, 1, 4, 6, 9, 2, 5, 7

Find operation:

→ check Root is NULL, if so return NULL.

→ otherwise check the value 'x' with root node.

if T -> data = x return T.

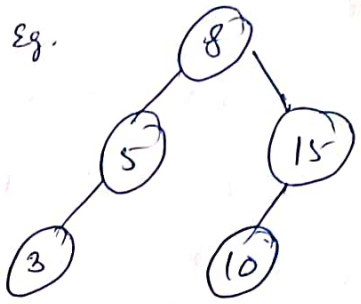
if x < T -> data traverse the left of T recursively

if x > T -> data " " Right " " "

Routine:

```
int find (int x, searchtree T)
{
    if (T == NULL)
        return NULL;
    if (x < T -> element)
        return find (x, T -> left);
    else if (x > T -> element)
        return find (x, T -> right);
    else
        return T;
}
```

find the value 10 in the following eg.




① find (10, 8) initialize pointing Root '8'

if (T == NULL) X

if (10 < 8) X

elseif (10 > 8) ✓

find (10, T → right):

find (10, 8 → right)

return find (10, 15)

② find (10, 15) (Recursive call)

if (T == NULL) X

if (10 < 15) ✓

return find (10, T → left)

find (10, 10).

③ find (10, 10) recursive call.

x T

if (T == NULL) X

if (10 < 10) X

if (10 > 10) X

else

return T

10 ✓

findMin:

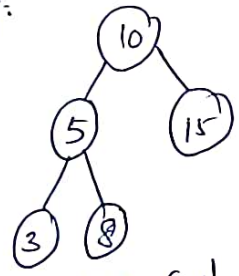
→ return position of smallest element in the tree.

→ start from root, go left,

Recursive alg for findMin:

```
int findMin (searchTree T)
{
 if(T == NULL)
   return NULL;
 else if (T → left == NULL)
  return T;
 else
   return findmin (T → left);
}
```

Eg:




findmin (T),
findmin(10)

findmin (10)

① T==NULL ✗ false

T → left == NULL
5 == NULL false

return findmin (T → left)
findmin (5)

② findmin (5)

T == NULL ✗

T → left == NULL
3 == NULL ✗

return findmin (3).

③ findmin (3)

T == NULL ✗

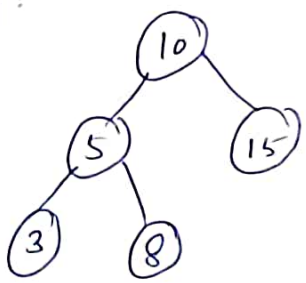
T → left ≠ NULL ✓

return 3.

Non-Recursive alg for findMin.

```
int findmin (searchtree T)
{
  if (T != NULL)
    while (T -> left != NULL)
      T = T -> left;
    return T;
}
```

Eg:




① (10 != NULL) ✓

while (T->left 10 != NULL)

T = T -> left

(T0) = 5

T = 5

② 5 != NULL

3 != NULL

T = T -> left

T = 3

③ 3 != NULL

while (T -> left != NULL)

3 -> left != NULL ×

return T

return 3

findmax;

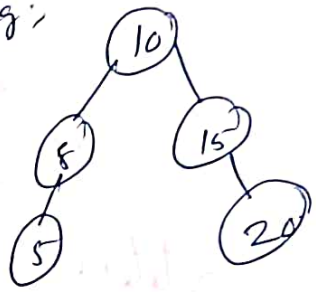
→ return position of largest element in the tree.

→ start from root, traveral right.

Recursive alg for findmax:

```
int *findmax (searchtree T)
{ if (T==NULL)
    return NULL;
  elseif (T->right == NULL)
    return T;
  else
    return findmax (T->right);
}
```

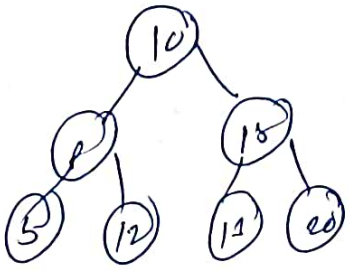
Eg:




Non Recursive alg:

```
int findmax(searchtree T)
{
  if (T != NULL)
    while (T->right != NULL)
      T = T->right;
    return T;
}
```

Eg:



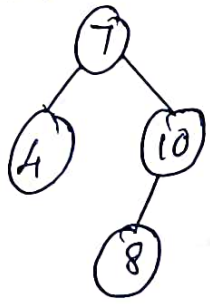
# Deletion in BST:

→ It is a Complex operation in BST

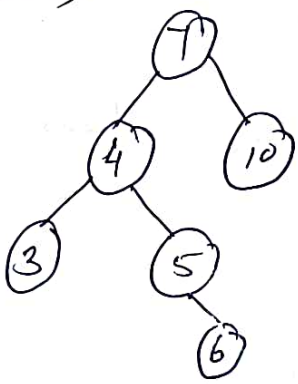
→ for deleting an element Consider 3 possibilities

* Node to be deleted is a leaf node. (ie) No Children
* Node with one child.
* Node with two child.

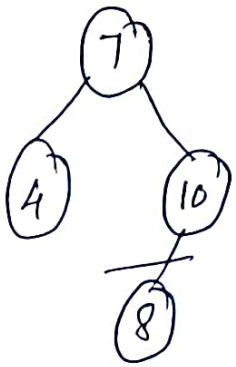
delete(8)



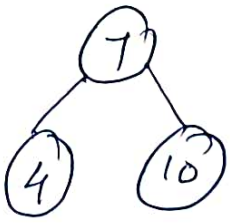

delete(5)




Case 1: Node with no children

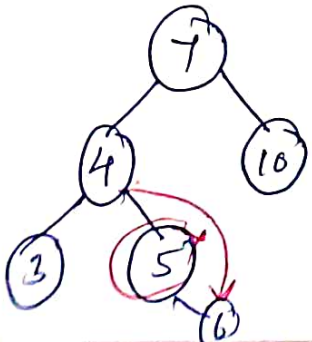



After deletion

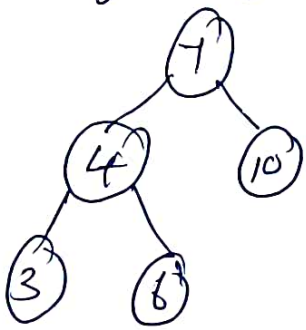



Case 2: Node with one child

→ It can be deleted by adjusting pointer.

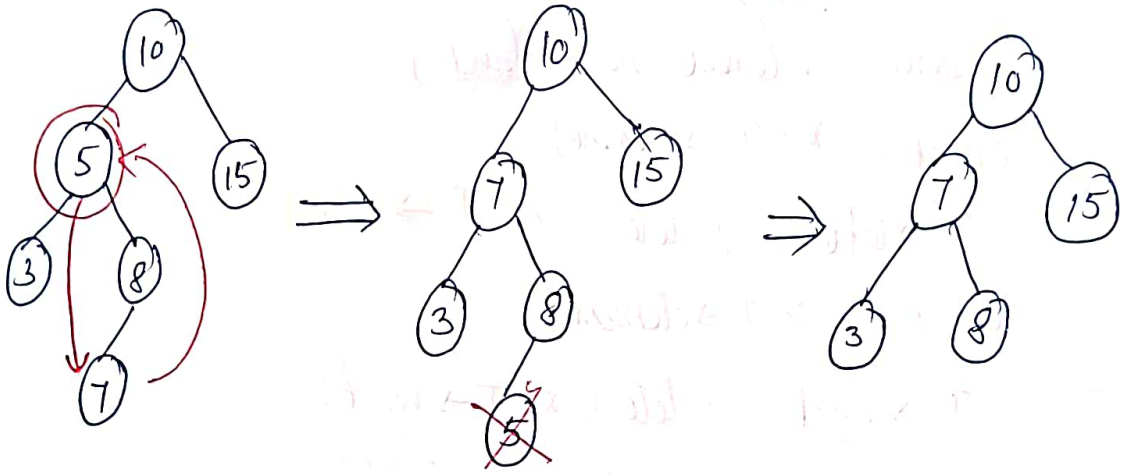



After deletion

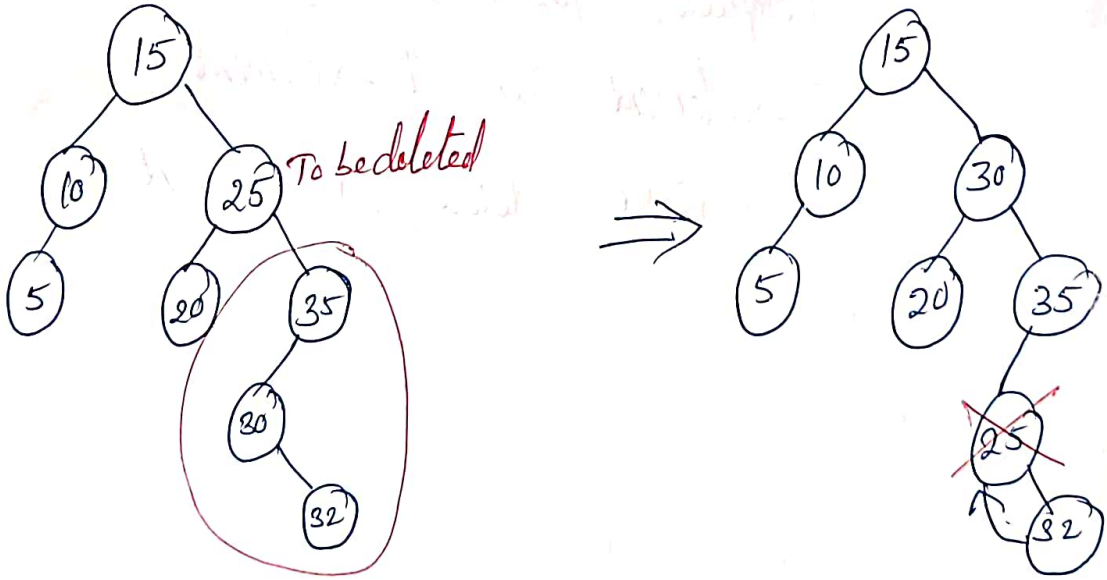


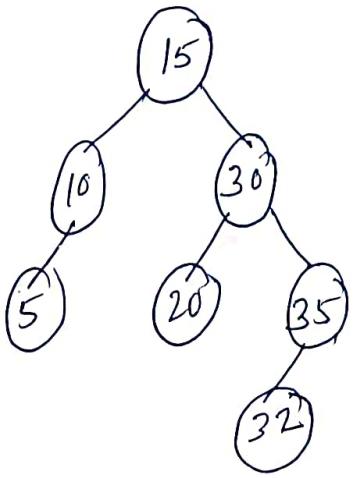
3: Node with two children (Delete 5)

* 1st replace the data of the node to be deleted with its smallest node of the right subtree, and recursively delete the node.




Delete 25:

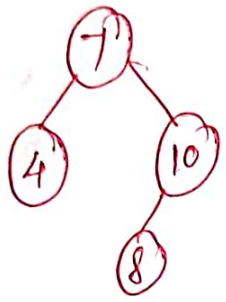



Swap minimum value in Right subtree

Routine to delete:

```
SearchTree Delete ( int x, SearchTree T)
{
  int tempcell;
  if (T==NULL)
    Error("Element not found");
  elseif ( x<T->element)
   T->left = Delete (x, T->left);
  elseif( x> T->element)
   T->right = Delete (x, T->right);
  elseif(T->left && T->right)
   {
     Tempcell = findmin (T->right)
     T->element = Tempcell ->element;
     T-> right = delete (T->element, T->right);
   }
  else
   {
     Tempcell =T;
     if(T->left ==NULL)
      T=T->right;
     elseif (T->right==NULL)
     T=T->left;
     free(Tempcell);
   }
  return T;
}
```




delete (8)

$x = 8 \quad T = 7$ delete (8, 7)

① $x < T$ $8 < 7$ false

$8 > 7$ true

$T \rightarrow right = delete(x, T \rightarrow right)$

$= delete(8, 7 \rightarrow right)$

$= delete(8, 10)$

② delete ($\overset{x}{8}, \overset{T}{10}$) $8 < 10$ True

$T \rightarrow left = delete(x, T \rightarrow left)$

$= delete(8, 8)$

③ delete (8, 8) $8 < 8$ ✗

$8 > 8$ ✗

else

Tempcell = T (8)

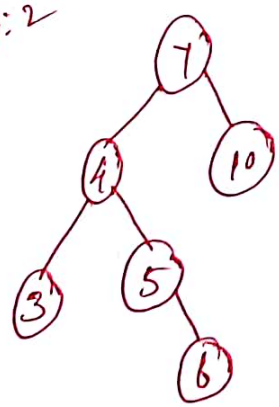
$T = T \rightarrow right$

$T \rightarrow right == NULL$

free (temp cell)

return T

Eg: 2 delete (5)




step 1:

delete (5, T)

delete (5, 7)

$5 < 7$ ✓

$T \rightarrow left = delete(5, T \rightarrow left)$

$= delete(5, 4)$

step 2:

delete (5, 4)

$5 < 4$ ✗

5>4 True

T→right = delete ( 5, 4→right

= delete (5, 5)

Step 3: delete (5, 5)

5<5 X

5>5 X

T→left && T→right (false) bec'z it has only right child

else

Tempcell = 5

If(T→left == NULL) True

T = T→right

T = 6

free (Tempcell)

free (5)

return 6

when call delete function, call step2, Hence T=4

T→right = 6,

4→right = 6