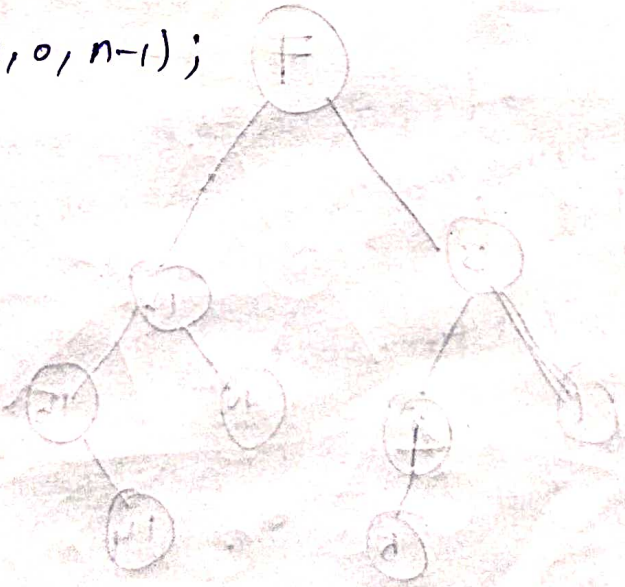
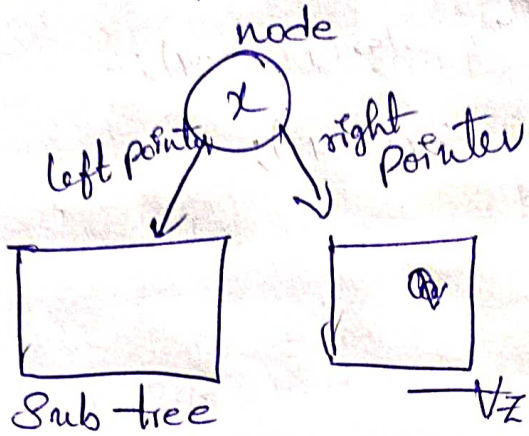
```
int n = 5;
void print (int a[], int n)
{
int i,j;
for (i=0; i<n,i++)
{
printf ("%d\t", a[i]);
}
printf ("\n")
}
void swap (int *a, int *b)
{
int temp;
temp = *a;
*a = *b;
*b = temp;
}
Void permutation ( int *a, int start, int end)
{
if (start==end)
{
print (a, end+1);
return;
}
int i;
for (i=start ; i<=end ; i++)
```

```
{
swap ((a+i) , (a+start));
permutation (a, start+1, end);
swap ((a+i) , (a+start));
}
}
int main()
{
int n=5;
int a[5]={ , , , , , }
Permutation (a, 0, n-1);
return 0;
}
```



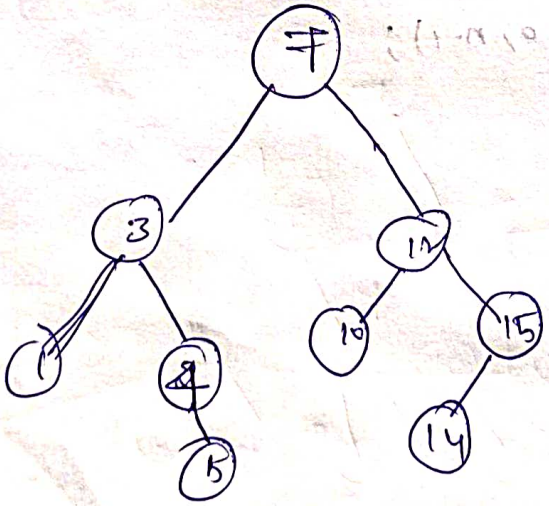
Tree

Binary Search tree



→ Vz in left Sub tree of node n
⇒ z → data < n → data

→ z → data > n → data



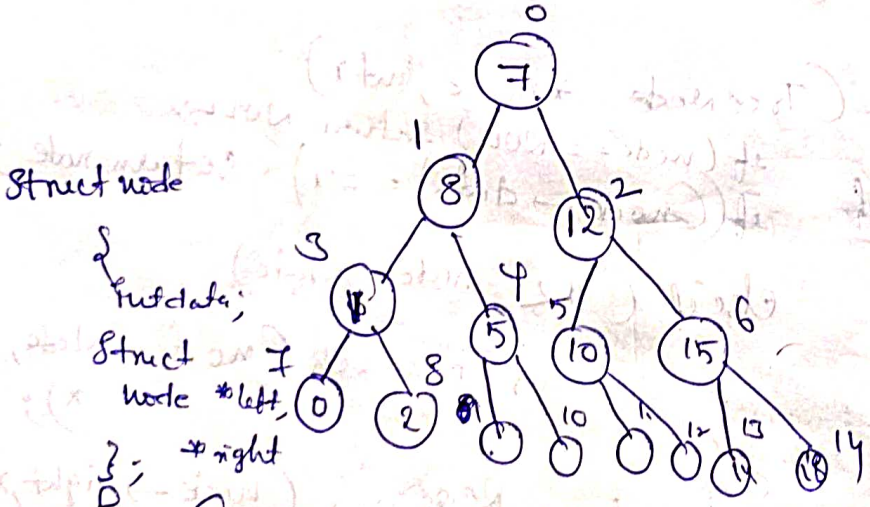

```
~~tree~~ Struct treeNode {
        int data;
        Struct treeNode *left, *right;
    };
Treenode* Search (treeNode * root, int x)
{
    struct treenode *CurrentNode = root;
```

```
while (Current node != NULL)
{ if (CurrentNode->data == x)
  {
    return Current node;
  }
  else if (Current Node->data >= x)
  {
    Current Node = Current Node->left;
  }
  else
  {
    Current Node = Current Node->right;
  }
}
return NULL;

Research (TreeNode *node, int x)
{ if (node == NULL) return NULL;
  if (node->data) == x) return node;
  else if (x < node->data)
    return Research (node->left,
                     x);
  else
    return Research (node->right, x);
}
```

```
void display (TreeNode * node)
    if (node == NULL)
    { printf ("NULL");
      return NULL;
    }
    display (node -> left);
    printf ("%d", node -> data);
    display (node -> right);
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
7	8	12	1	5	10	15	0	2	4	6	-1	-1	

9, 13, 14,




```
Struct node
{
  int data;
  Struct node *left, *right;
};
```

```
for (Pos = 0; Pos < 7; Pos++)
    leftpos = 2 * pos + 1;
    node -> data = arr [Pos];
    node -> left = (struct node*) mall.
    node -> left -> data = arr [leftpos];
    node -> left -> left = NULL;
    node -> left -> right = NULL;
```

```
node → right = (str  ) malloc
node → right → left = node → right → right = NULL;
Parr [leftpos] = node → left ;
Parr [ rightpos] = node → right ;
```

```
typedef struct TN {
        int data;
        Struct TN* left, * right;
} * TNP;

Void main()
  {
int a[15] =

TNP Pa[15];
TNP root = (TNP) malloc (sizeof(struct ..))
root → data = a[0];
root → left = root → right = NULL
for (int i=0; i<7; i++) Pa[0] = root;
  {
    TNP node = Pa[i];
    int left pos = i*2+1;
    node → left = (TNP malloc (s- ..);
    node → left → data = a[left pos];
    node → left → left = node → left →
                         right = NULL;
    int right pos = i*2+1
```

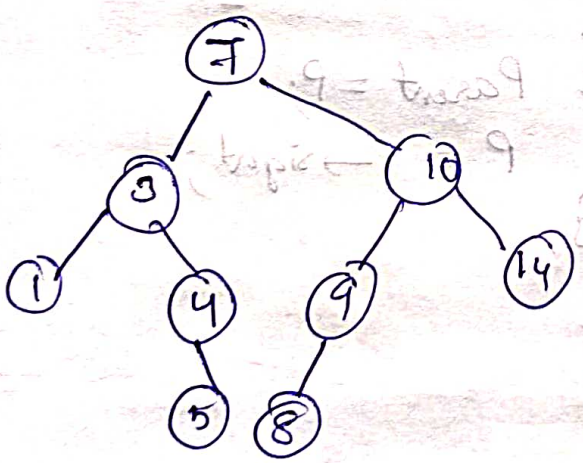
```
node -> right = (TNP) malloc(
node -> right -> data = a[rightpos];
node -> right -> left = node -> right -> right
pa[leftpos] = node -> left; = NULL;
pa[rightpos] = node -> right;
}
void display(TNP node)
{
    if (node == NULL) return;
    display(node -> left);
    Print("%d -> ", node -> data);
    display(node -> right);

insertion(TN root, int x)
{
TNP P = root;
while(1) {
    if ((x < P -> data) && (P -> left == NULL))
        P -> left = Newnode;
        break;
    else
    { if ((x > P -> data) && (P -> right == NULL))
        { P -> right = newnode;
          break;
        }
    }
}
```

TNP newnode = (TNP) malloc(sizeof(struct

newnode -> data = x.

newnode -> left = newnode -> right = NULL;

```
if (x < p→data)
    p = p→left;
else if (x > p→data)
    p = p→right;
else
{ printf ("%d already exist\n", x);
  break;
}
```

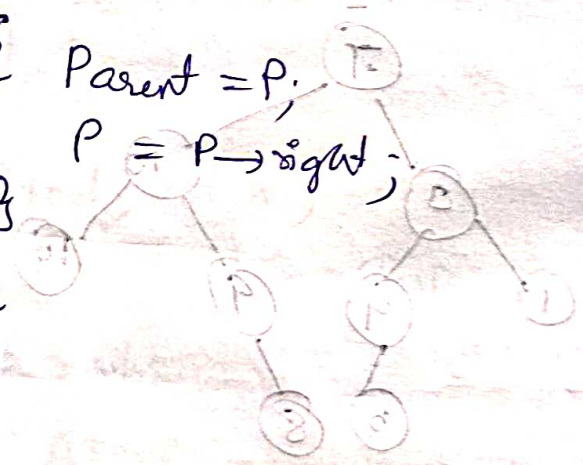


right = NULL;

delete_Smallest_right_side-tree_element

```
delete_SRSC()
    p→left = dN→left
    p→right = dN→right
```

```
delete (TNP root, int x)
{
TNP Parent = root;
TNP P = root;
while (P->data != x && (P != NULL)
{ Parent=P;
if (x < P->data)
{
P = P->left;
Parent=P;
}
else if (x > P->data)
{ Parent = P;
P = P->right;
}
else
{
}
}
if ((P != NULL) && (!P->left) &&
                    (!P->right))
{ if (Parent->left == P)
{
parent->left = NULL;
delete(P);}
else
{
Parent->right = NULL; delete(P);}
```



```
else if (P != NULL) && ((!P->left)||(!P->right))
{
  if (Parent->left == P)
    {
      if (P->left)
        { Parent->left = P->left)}
      else
        { parent->~~right~~left = P->right;
        }
    }
  else if
    {
      if (P->left)
        { Parent->right = P->left;
        }
      else { Parent->right = P->right;
           }
    }
  free(P);
}
  else if (P!=NULL)
  TNP PP = deleter_SRS∈(P->right);
  PP->Left = P->left
             P->right;
```

```
if (parent -> left == p)
    {
      parent -> left = pp;
    }
else
    {
      parent -> right = pp;
    }

delete_BST (TNP root)
    {
      TNP p = root; parent = NULL;
      while (p->left)
        {
          parent = p; p = p->left;
        }
      if ((p->right) && parent != NULL)
        {
          parent -> left = p -> right
        }
      else
        { if (parent != NULL)
            {
              parent -> left = NULL;
            }
        return p;
    }
```

```
inorder(TNP root)
{ if (root != NULL)
  {
    inorder (root → left);
    Printf ("%d", root → data);
    inorder (root → right);
  }
  return;
}
```

$x_1, x_2, \ldots\ldots x_N$

$x_1 \rightarrow x_N \rightarrow x_2 \rightarrow x_{N-1} \rightarrow x_{N-2}$

Pre-order

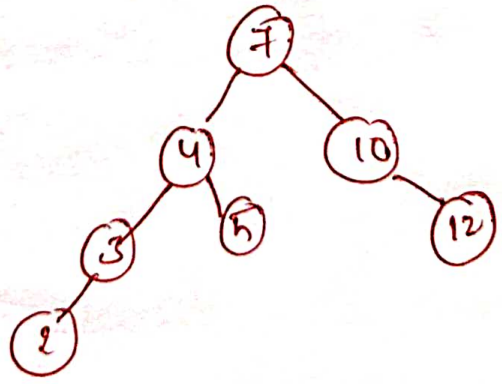
7 3 2 5 4 12 10

In-order

2, 3, 4, 5, 7, 10, 12

Post-order

2, 4, 5, 3, 10, 12, 7



⇒ 2 3 4 5 7 10 12

⇒ 2 3 5 4 12 10 7

```
struct node* deleteNode (struct node* root, int key){
if (root == NULL)
        return root;
if (key < root->key)
    root -> left = deleteNode (root -> left, key);
else if (key > root -> key)
    root -> right = deleteNode (root -> right, key);
else {
    if (root -> left == NULL){
        struct node* temp = root -> right;
        free (root);
        return temp;
    }
    else if (root -> right == NULL){
        struct node* temp = root -> left;
        free (root);
        return temp;
    }
struct node* temp = minValueNode (root -> right);
root -> key = temp -> key;
root -> right = deleteNode (root -> right, temp -> key);
}
return root;
}
```

```
struct node* minValueNode (struct node* node){
    struct node* current = node;
    while (current && current -> left != NULL)
        current = current -> left;
    return current;
}
```