

Dummy Pdf

PRESENTATION ON

Tree

in

Data Structure

DATA STRUCTURE

Store and organize data in computer.

□ Linear Data Structure

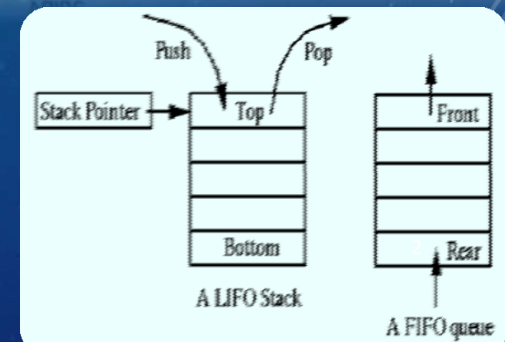
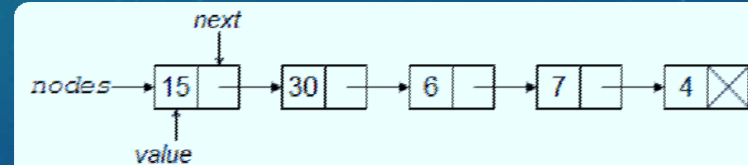
Arrays

Linked List

Stacks

Queues

element	5	7	2	12
Address	1000	1002	1004	n

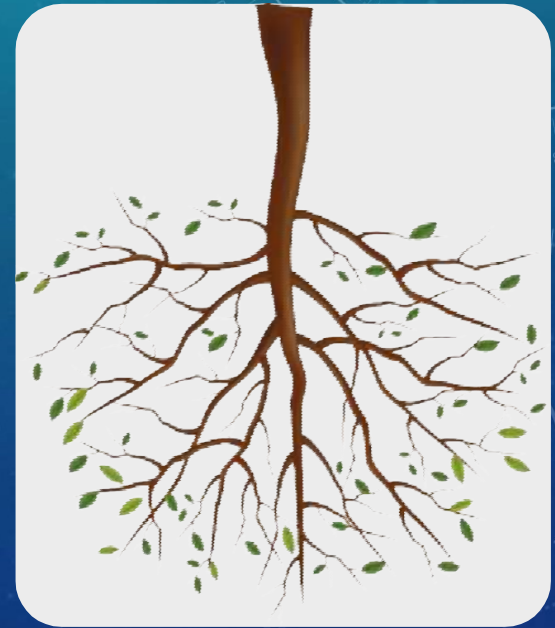


LOGIC OF TREE

- Used to represent hierarchica data.

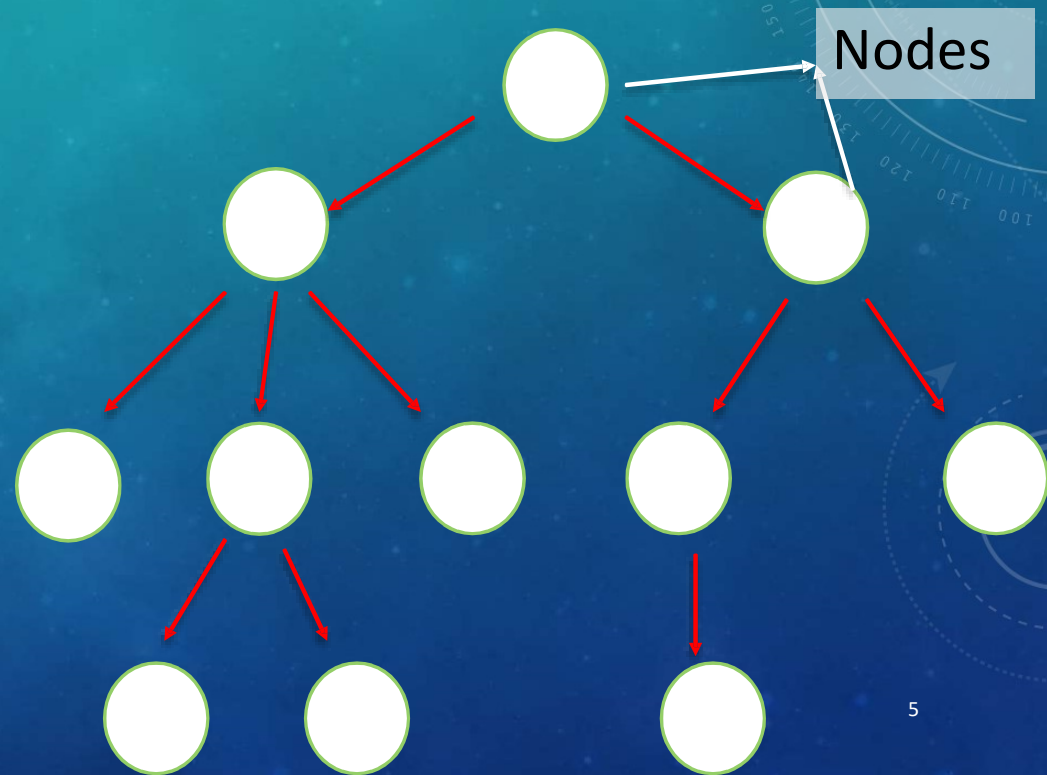


❑ Used to represent hierarchical data.



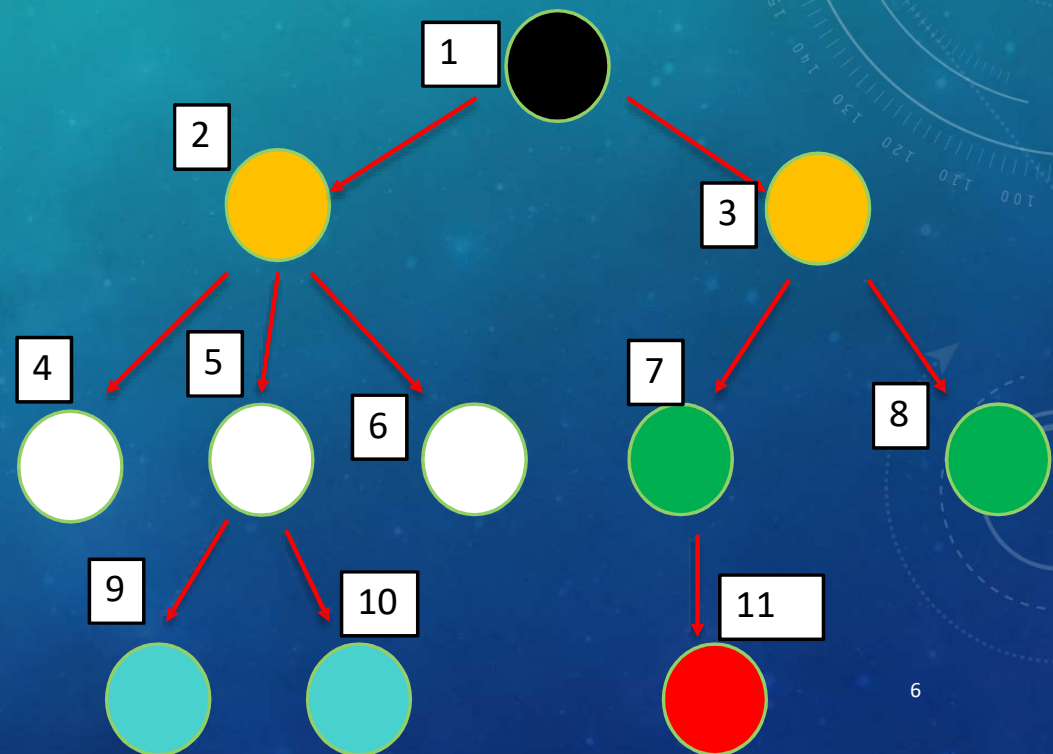
TREE

- A Collection of entities called Nodes.
- Tree is a **Non-Linear Data Structure**.
- It's a **hierarchica Structure**.



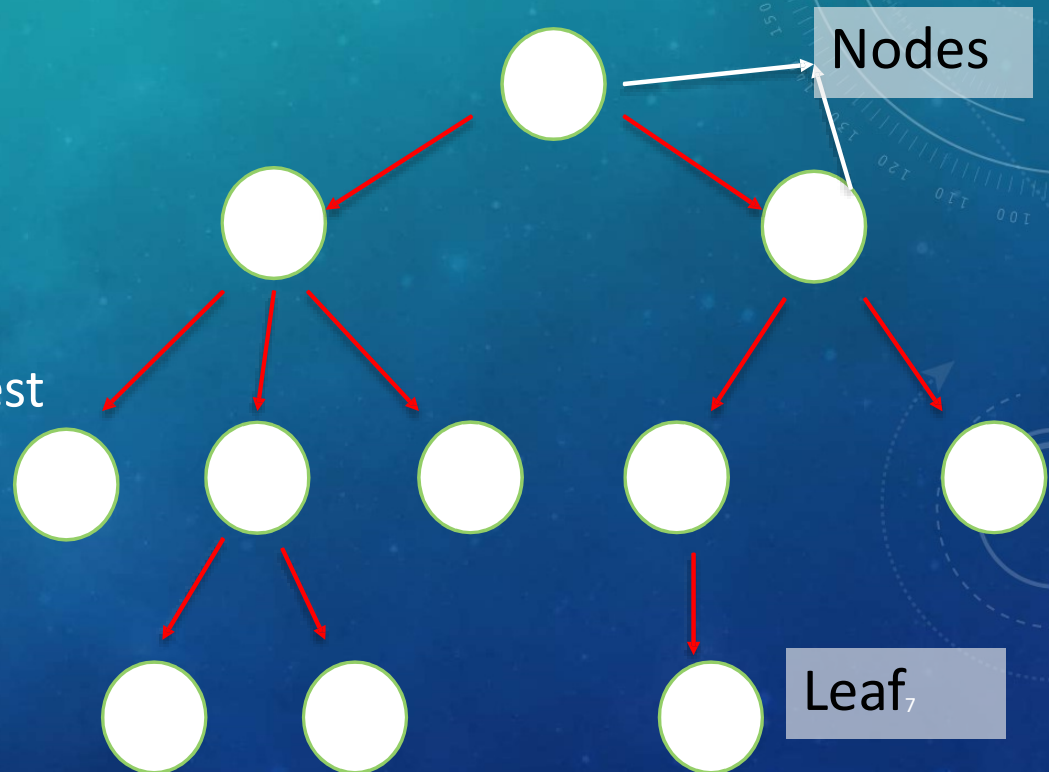
RELATION OF TREE

- **Root**-The top most Node.
- **Children**
- **Parents**
- **Siblings**- Have same parents.
- **Leaf**- Has no Child.



EDGES, DEPTH, HEIGHT

- ❑ **Edges:** If a tree have N nodes
It have N-1 edges.
- ❑ **Depth of x:** Length of path from
Root to x.
- ❑ **Hight of x:** No. Of edges in longest
Path from x to a leaf

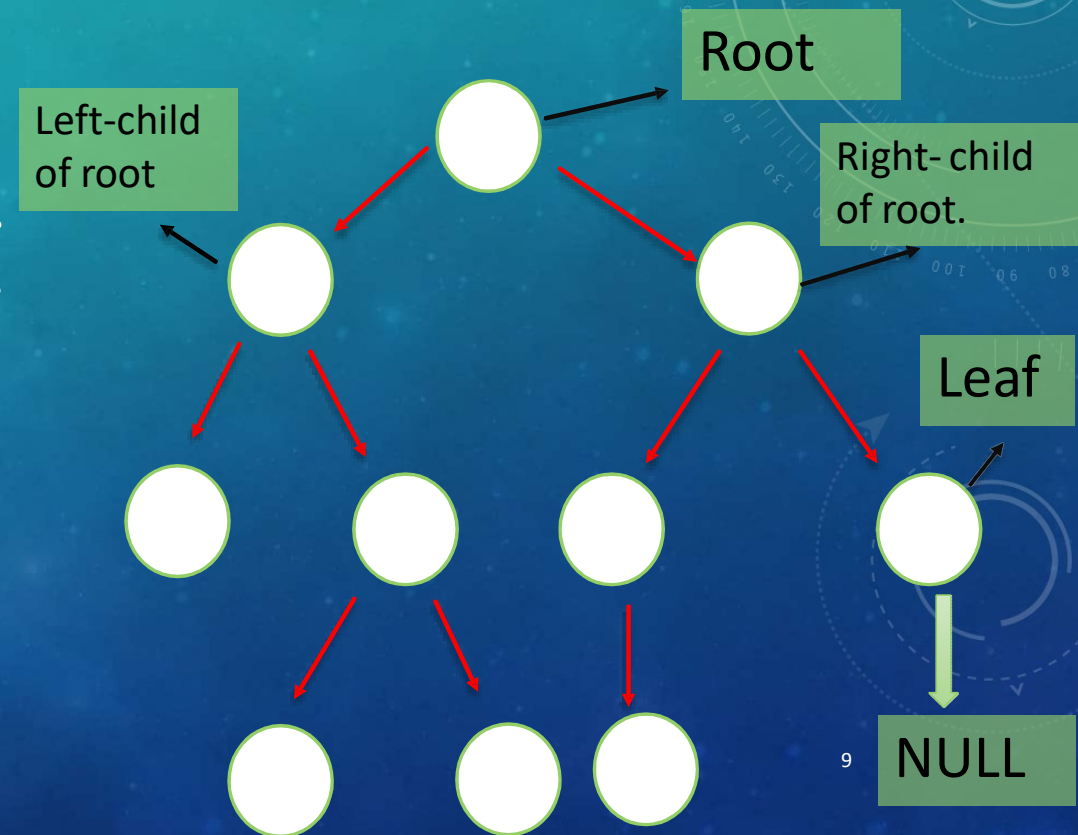


SOME APPLICATION OF TREE IN COMPUTER SCIENCE

1. Storing naturally hierarchical data- **File system**.
2. Organize data for quick search, insertion, deletion- **Binary search tree**.
3. **Dictionary**
4. Network Routing Algorithm.

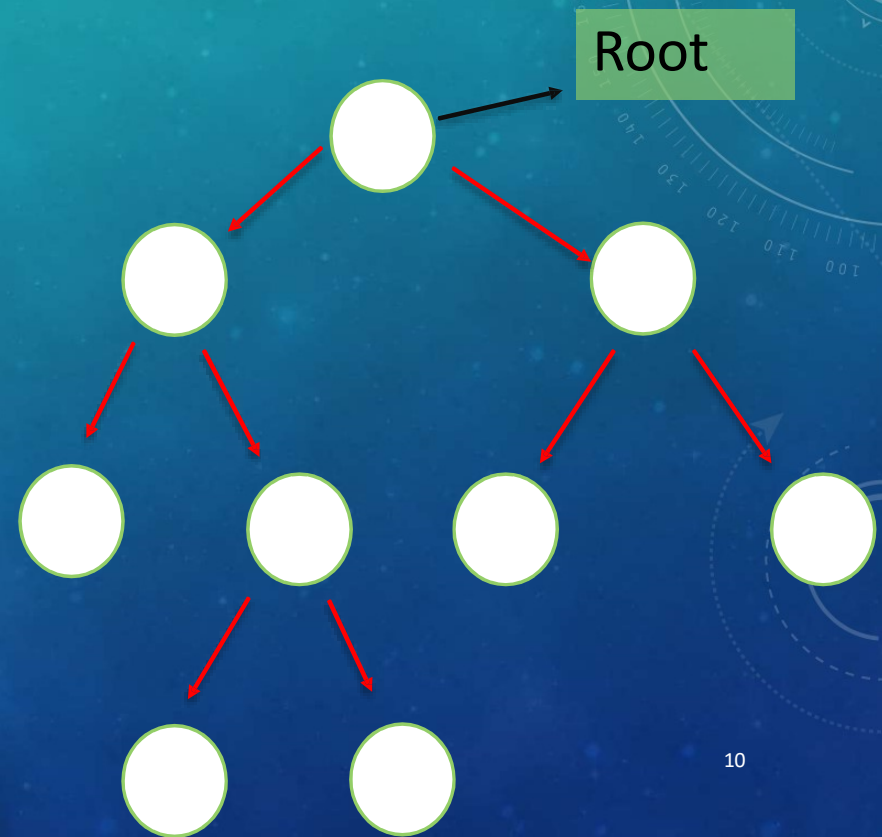
BINARY TREE

- Each node can have at most 2 children.
- A node have only left and right child or
- Only left child or
- Only right child.
- A leaf node has no left or right child.
- A leaf node has only NULL.



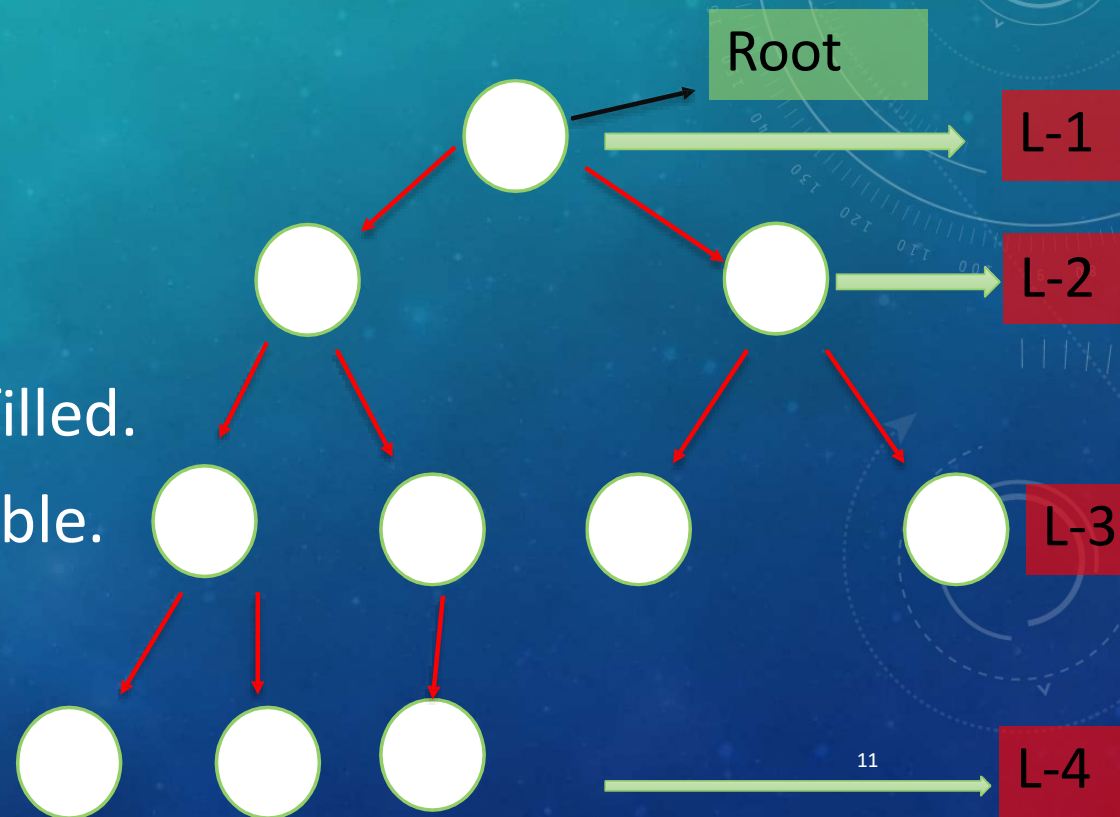
STRICT/PROPER BINARY TREE

Each node can have either
2 or 0 child

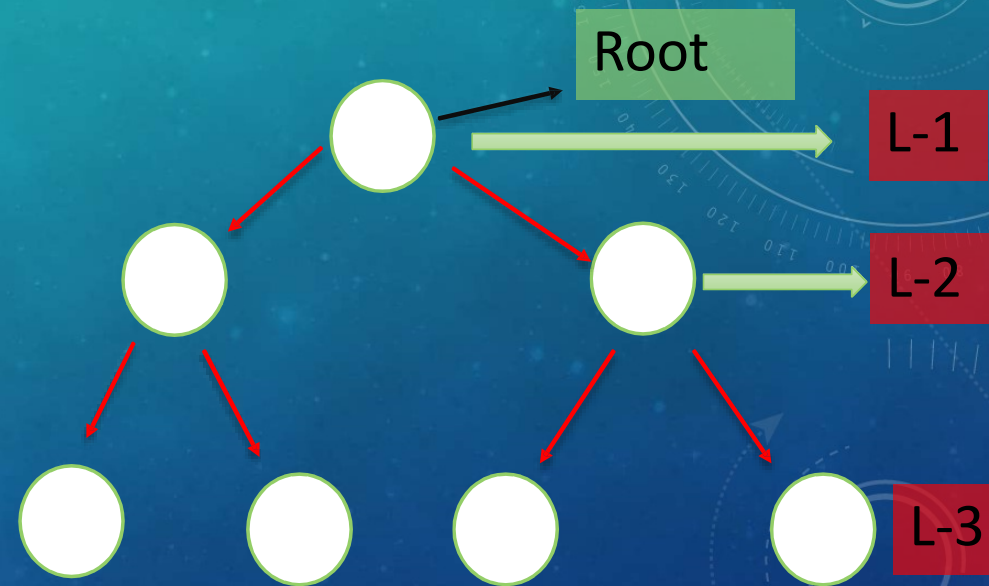


COMPLETE BINARY TREE

- The last level is completely filled.
- All nodes are as left as possible.



PARFECT BINARY TREE



All the levels are completely filled.

WE CAN IMPLEMENT BINARY TREE USING

A) Dynamically created nodes.

struct node

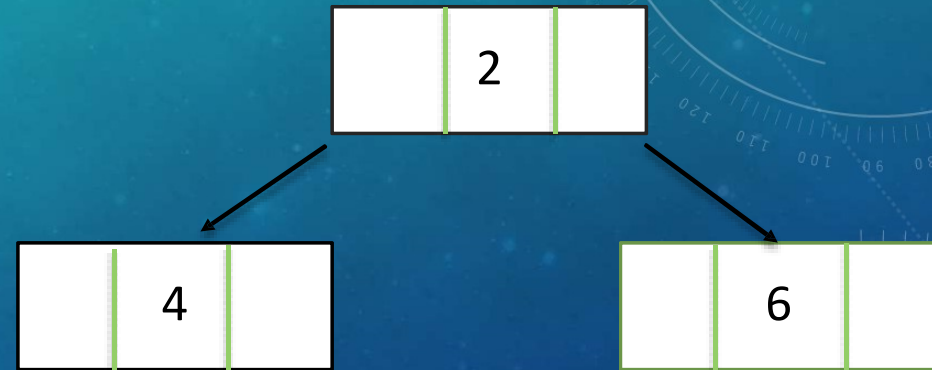
{

int data;

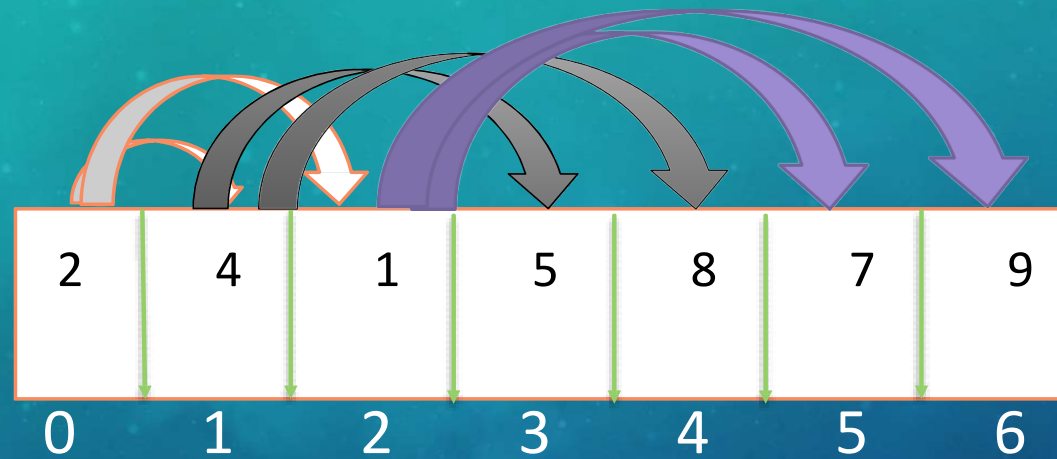
struct node* left;

struct node* right

}



OR



B) Arrays: It only works “complete Binary tree”.

For node at index i ;

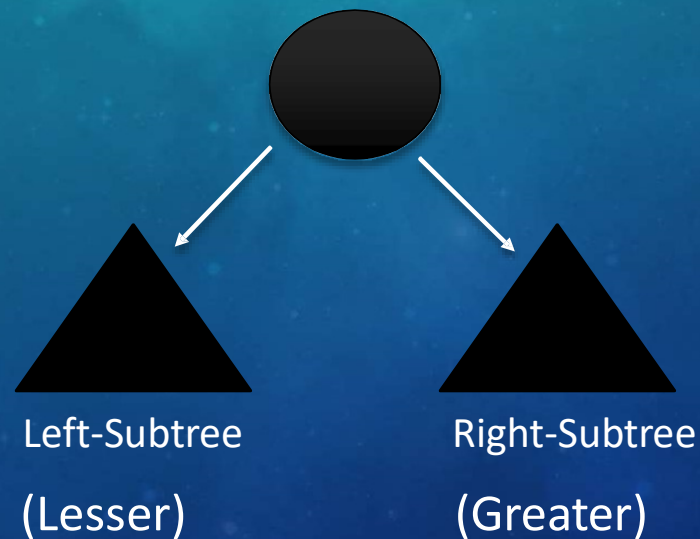
Left-child-index= $2i+1$

Right-child-index= $2i+2$

IMPLEMENT OF BINARY SEARCH TREE

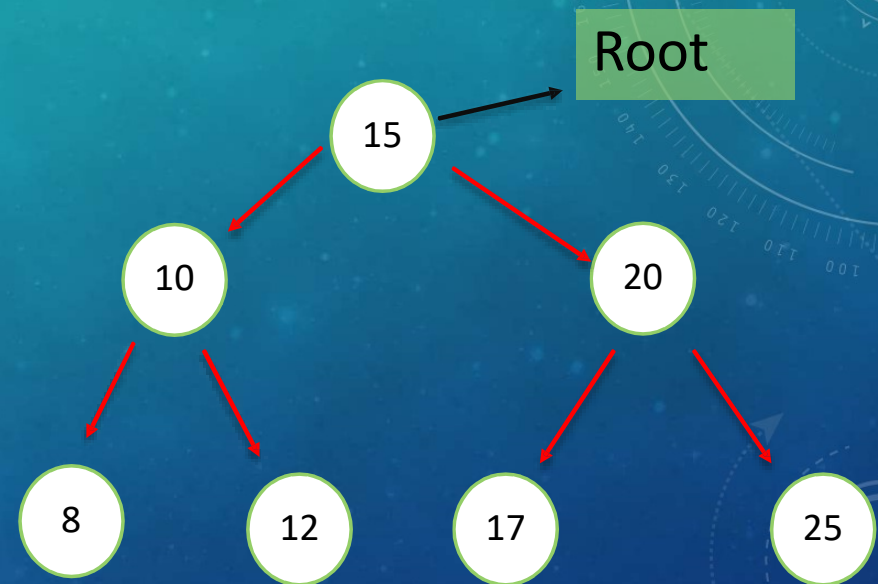
Value of all the nodes in left subtree is Lesser or Equal.

Value of all the nodes in right subtree is greater.



EXAMPLE

- $15 > 10$ - Left
- $15 < 20$ - Right
- $10 > 8$ - Left
- $10 < 12$ - Right
- $20 > 17$ - Left
- $20 < 25$ - Right



BINARY TREE TRAVERSAL

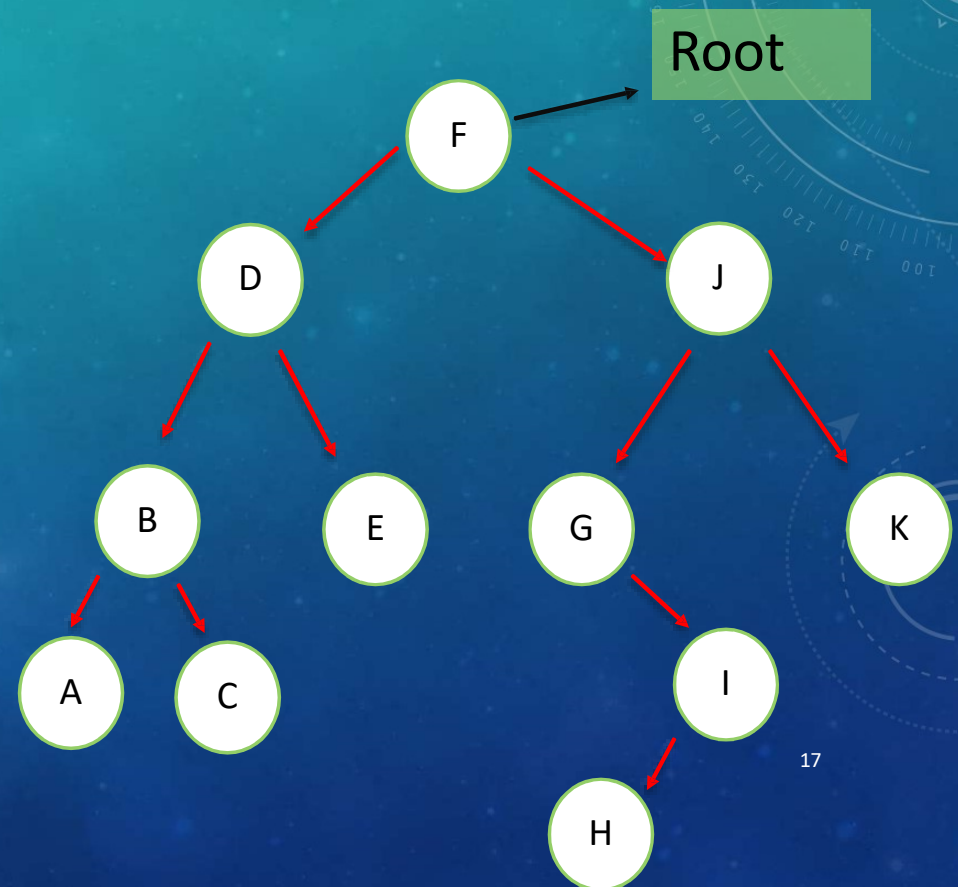
Tree traversal

Breadth-first or
Level-order

F,D,J,B,E,G,K,A,C,I,H

Depth-first

Preorder, Inorder &
Postorder



PREORDER(SLR)

Start(Data)

Left Right

<root><left><right>

F,D,B,A,C,E,J,G,I,H,K

```
Void Postorder(struct bstnode* root)
```

```
{
```

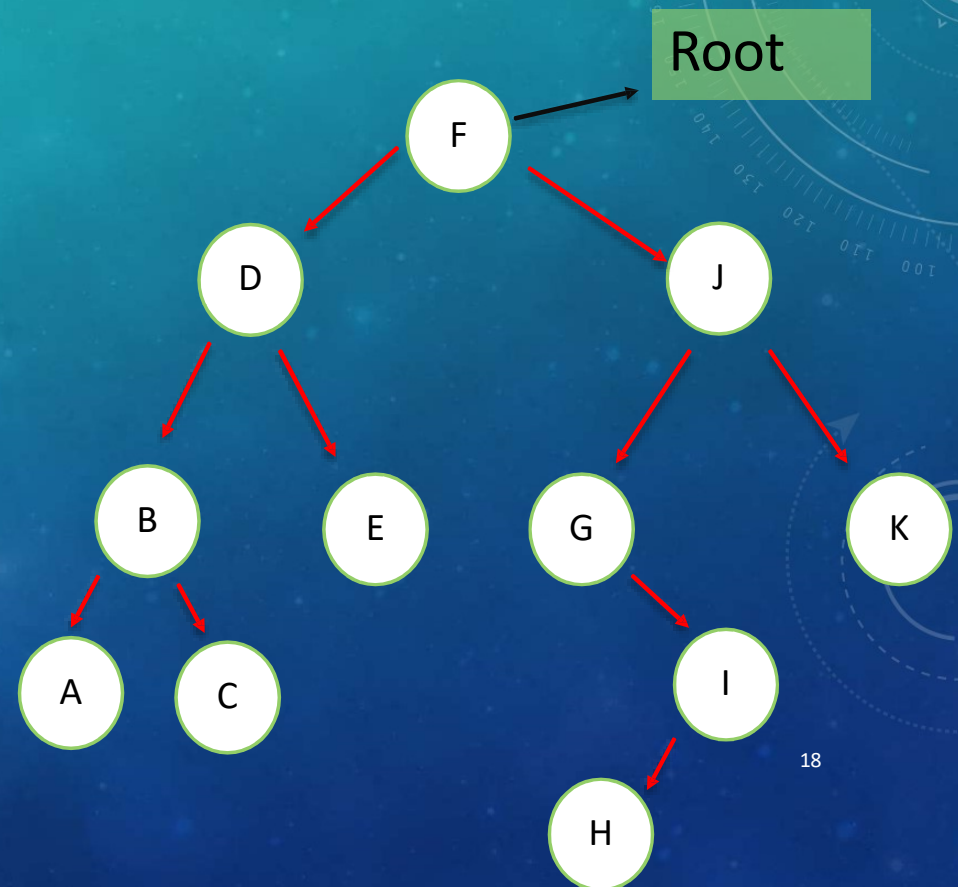
```
    if(root==NULL)
```

```
        Postorder(root->right);
```

```
        Postordrt(root->right);
```

```
        printf("%c",root->data);
```

```
}
```

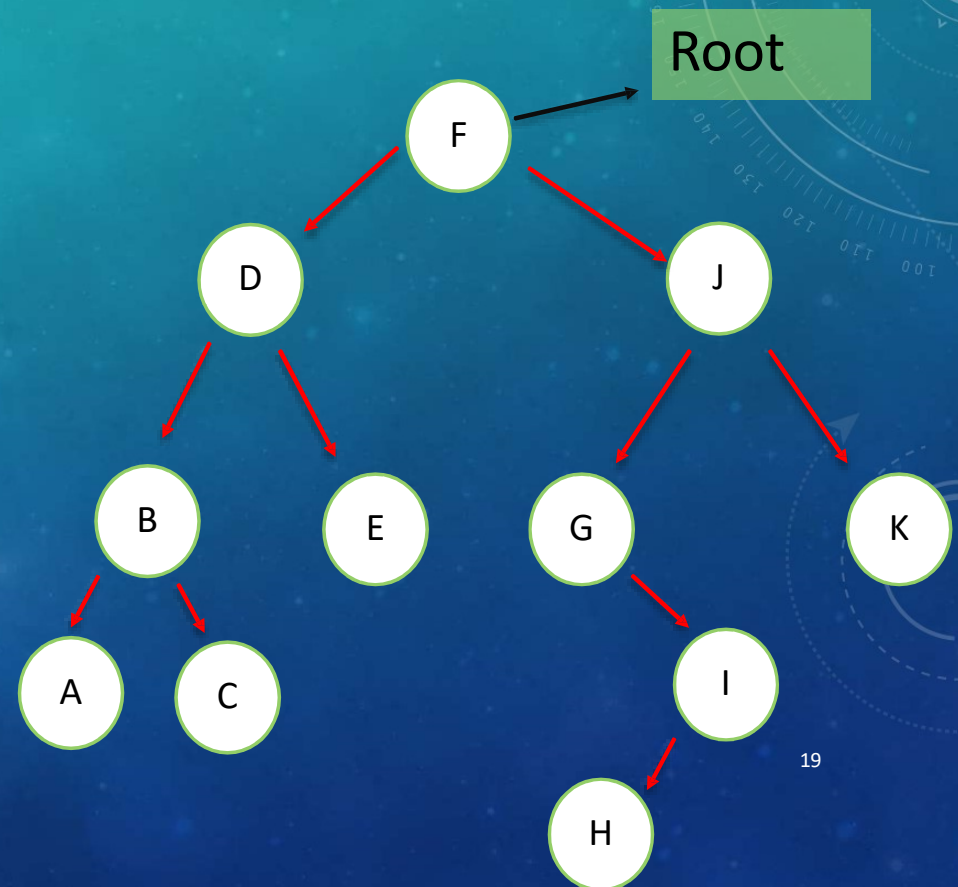


INORDER(LSR)

Left S(Data)Right
↓ ↓ ↓
<left><root><right>

A,B,C,D,E,F,G,H,I,J,K

```
Void Inorder(struct bstnode* root)
{
    if(root==NULL) return;
    Inorder(root->left);
    printf("%c",root->data);
    Inorder(root->right);
}
```



C Examples

// Tree traversal in C

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct node {
    int item;
    struct node* left;
    struct node* right;
};
```

// Create a new Node

```
struct node*
createNode(value) {
    struct node* newNode =
    malloc(sizeof(struct node));
    newNode->item = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
```

// Insert on the left of the node

```
struct node* insertLeft(struct node* root, int value)
{
    root->left = createNode(value);
    return root->left;
}
```

// Insert on the right of the node

```
struct node* insertRight(struct node* root, int value)
{
    root->right = createNode(value);
    return root->right;
}
```

// Preorder traversal

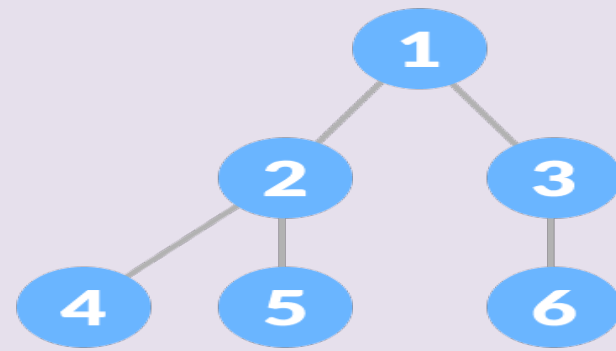
```
void preorderTraversal(struct node* root) {  
    if (root == NULL) return;  
    printf("%d ->", root->item);  
    preorderTraversal(root->left);  
    preorderTraversal(root->right);}124536
```

// Postorder traversal

```
void postorderTraversal(struct node* root) {  
    if (root == NULL) return;  
    postorderTraversal(root->left);  
    postorderTraversal(root->right);  
    printf("%d ->", root->item);}452631
```

// Inorder traversal

```
void inorderTraversal(struct node* root) {  
    if (root == NULL) return;  
    inorderTraversal(root->left);  
    printf("%d ->", root->item);  
    inorderTraversal(root->right); } 4 2 5 1 3 6
```



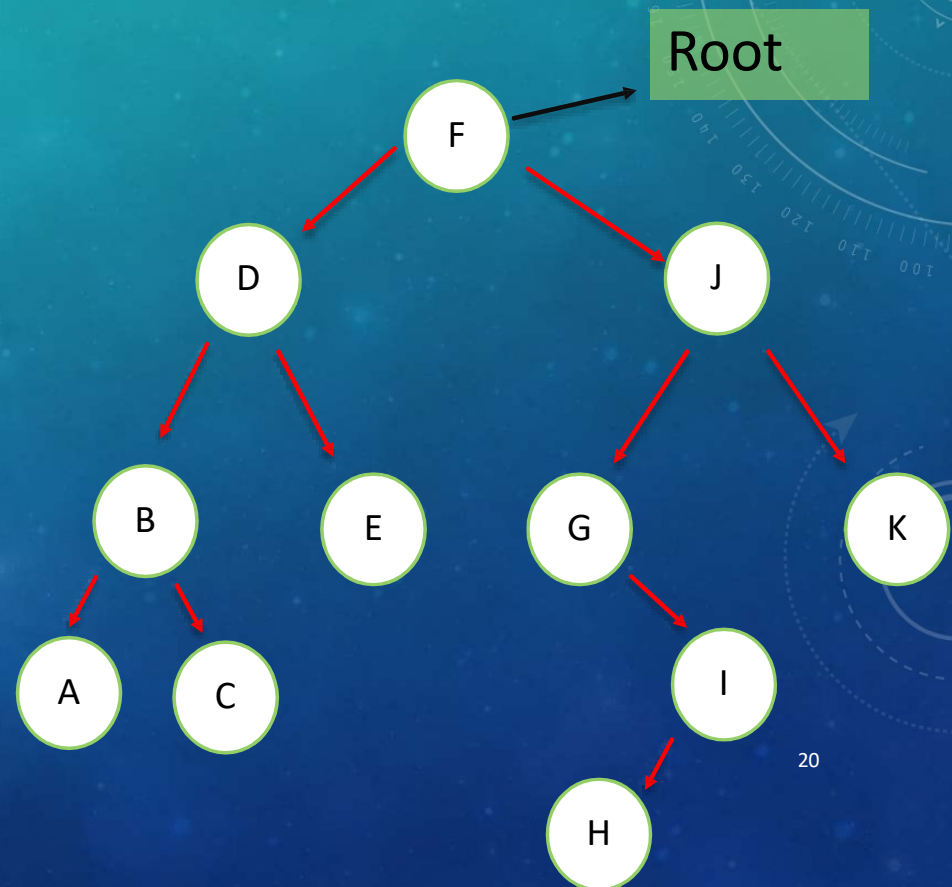
```
int main() {  
    struct node* root = createNode(1);  
    insertLeft(root, 2);  
    insertRight(root, 3);  
    insertLeft(root->left, 4);  
    printf("Inorder traversal \n");  
    inorderTraversal(root);  
    printf("\nPreorder traversal");  
    preorderTraversal(root);  
    printf("\nPostorder traversal\n");  
    postorderTraversal(root);}
```

POSTORDER(LRS)

Left Right (Data)Start
↓ ↓ ↓
<left><right><root>

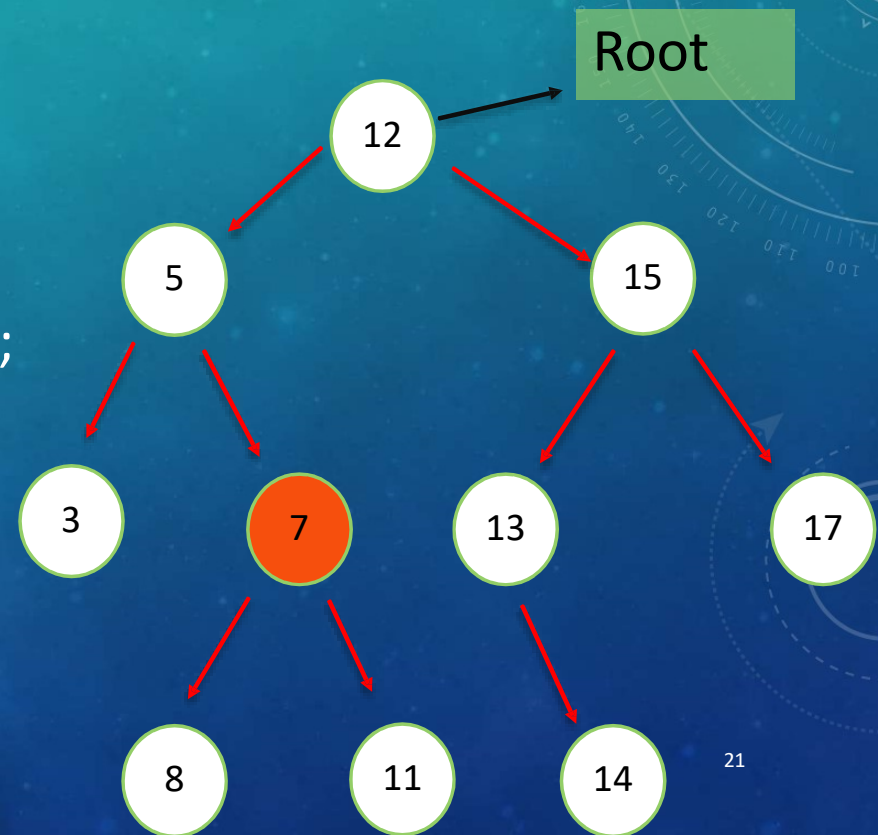
A,C,B,E,D,H,I,G,K,J,F

```
Void Postorder(struct bstnode* root)
{
    if(root==NULL)
        Postorder(root->right);
    Postordrt(root->right);
    printf("%c",root->data);
}
```



SEARCH AN ELEMENT IN BST

```
bool Search( bstnode* root, data type)
{
    if (root==NULL) return false;
    else if(root->data == data) return true;
    else if(root->data <= data)
        return Search(root->left, data);
    else
        return Search(root->right, data);
}
```



RUNNING TIME OF OPERATION

Operation	Array	Link List	Binary Search Tree (average case)
Search(x)-Search for an element x.	$O(\log n)$	$O(n)$	$O(\log n)$
Insertion(x)-Insert an element x.	$O(n)$	$O(1)$	$O(\log n)$
Remove(x)-Remove an element x.	$O(n)$	$O(n)$	$O(\log n)$