LINEAR DATA STRUCTURES

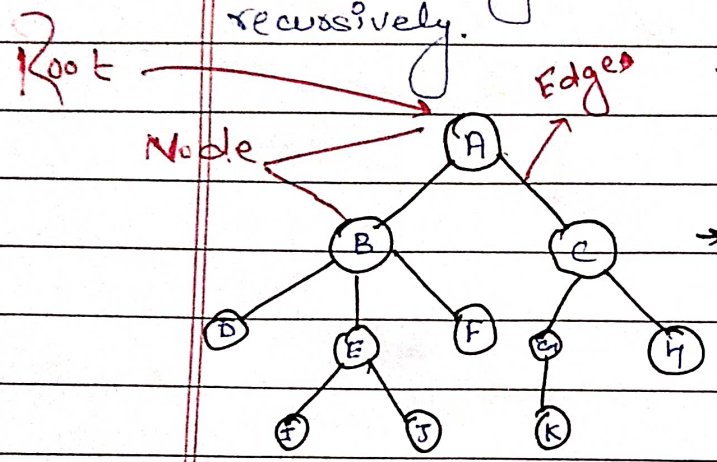
Date
Page

Tree

TREE - TERMIONOLOGY

- Tree is a non-linear data structure which organizes data in hierarchial structure and this is a recursive definition.

- Tree data structure is a collection of data (Node) which is organized in hierarchial structure recursively.



Tree with 11 Nodes and 10 Edges

→ In any tree with 'N' nodes there will be maximum of 'N-1' edges.

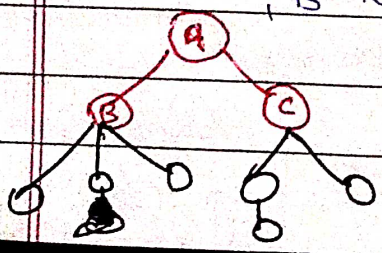
→ In a tree every individual element is known as Node.

Terminology

- Root → First Node of a tree data structure.

- Edge → In any tree, Edge is connecting link between two nodes.

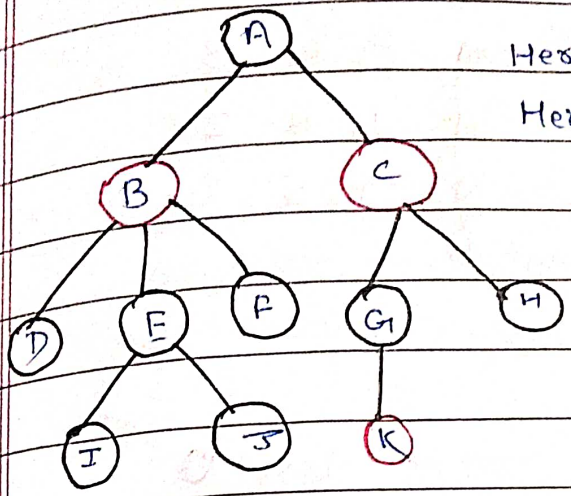
- Parent → The node which is predecessor of any node, is known as Parent.



Here, A, B C are Parent Nodes.

classmate
Date
Page

- **Child** → The node which is descendant of any node is known as CHILD Node



Here B & C are Children of A.
Here G & H are Children of C
⋮
Here K is Child of G

- **Siblings** → Nodes which belong to same Parent, are known as Siblings.

Here B & C are Siblings
Here D E & F are Siblings
Here G & H are Siblings
Here I & J are Siblings

- **Leaf** → The node which does not have a child is known as Leaf Node or External Node or Terminal Node.

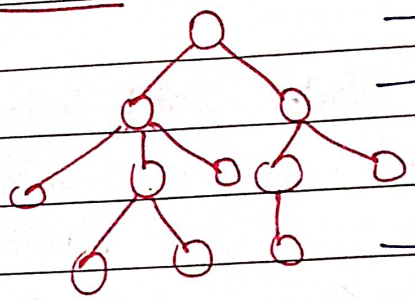
Here D, I, J, F, K & H are leaf Nodes.

- **Internal Nodes** : Node which has atleast one child are known as Internal Nodes or Non-Terminal Nodes
  - The root node is also known as an Internal Node.

- **Degree :** The total number of children of a Node is known as Degree.

In figure, Degree of B is 3
Degree of A is 2
Degree of F is 0
⋮

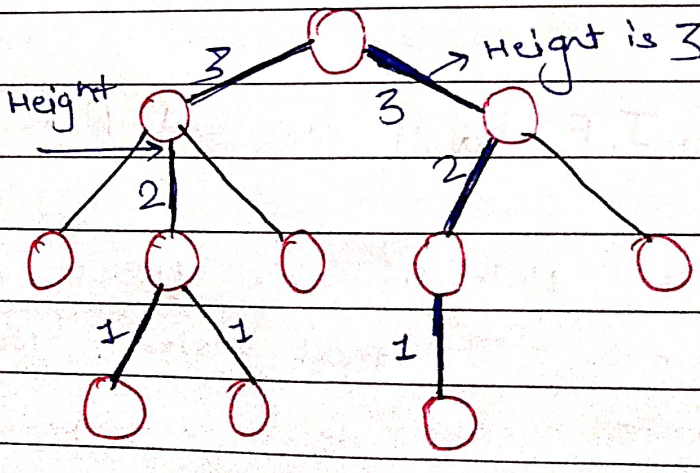
- **Level ;**



→ Level 0
→ Level 1
→ Level 2
→ Level 3.

- **Height :** Total number of edges from leaf node to a particular node in the longest path is known as Height of the Tree..

In a tree, height of root node is said to be the height of the tree.



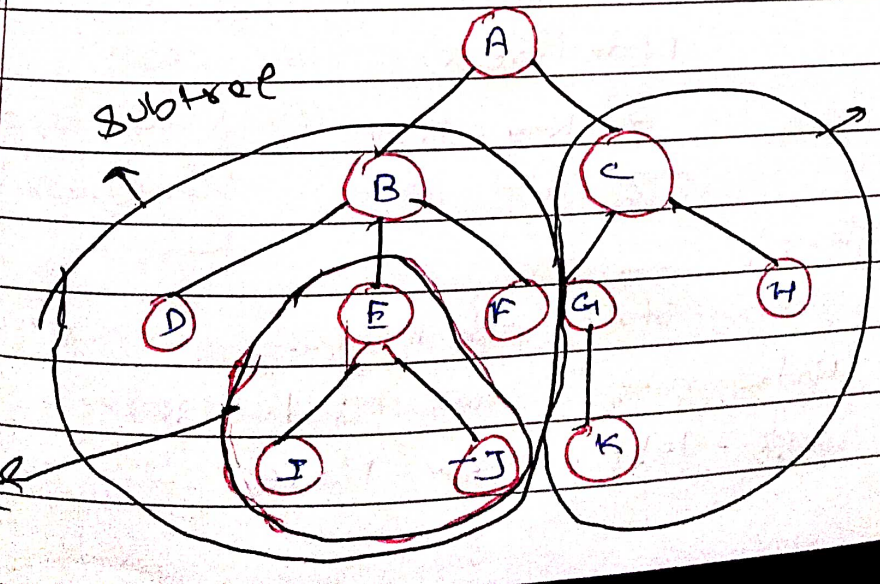
Here Height of tree is 3

- **Depth:** The total number edges from root node to a particular node is known as Depth of that Node.
  - Depth of root node is 0.
- **Path:** The sequence of Nodes and Edges from one node to another node is known as Path between that two Nodes.
  - Length of a Path is total numbers of nodes in that path.

Here, Path between A and J is
A - B - E - J.

Here, Path between C and K is
C - G - K.

- **Sub Tree:** Each child from a node forms a subtree recursively. Every child node will form a subtree on its parent Node.

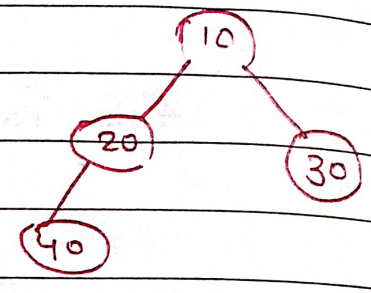


BINARY TREE

- Binary tree is the mostly used tree data structure.

C++

```
Struct Node {
 int Key;
Node *left;
Node *right;
Node(int k) // Constructor
{ Key = k;
 left = right = NULL;
}
};
int main() {
Node *root = new Node(10);
root -> left = new Node(20);
root -> right = new Node(30);
root -> left -> left = new Node(40);
}.
```



Java

```
class Node {
    int key;
    Node left;
    Node right;
    Node (int k)
    { key = k;
    }
class Test {
    public static void main ...
{ Node root = new Node (10);
    root . left = new Node (20);
} :
```

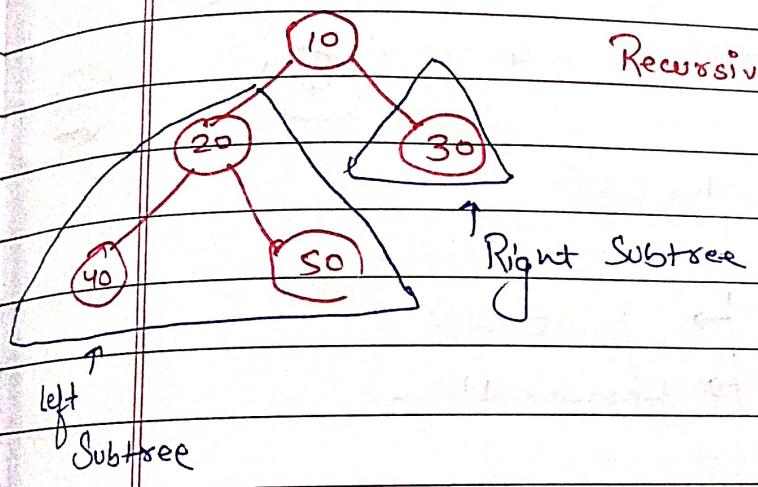
// Left and Right are by default initialized as null.

Tree Traversal

(Breadth First (OR Level Order)

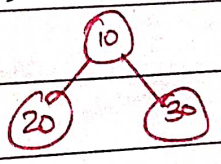
Depth First
- In Order
- Pre Order
- Post Order

• Depth-first Traversal



Recursive
- Traverse Root
- Traverse Left Subtree
- Traverse Right Subtree

- In Order (Left Root Right)
- Pre Order (Root Left Right)
- Post Order (Left Right Root)



Left → Left Subtree

Right → Right Subtree

Inorder: 20 10 30

Preorder: 10 20 30

Postorder: 20 30 10

• Implementation of Inorder Traversal

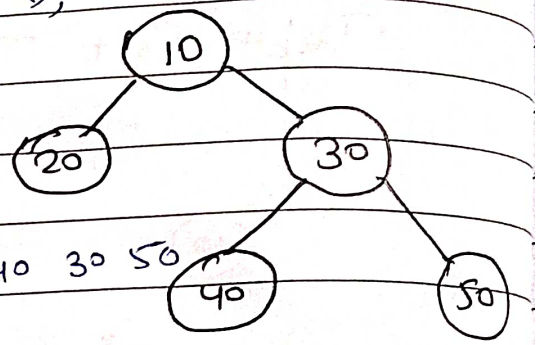
C++

```
void inorder (Node *root)
{
    if (root != NULL)
    {
        inorder (root -> left);
        cout << (root -> key) << " ";
        inorder (root -> right);
    }
}
```

Left
Root
Right

Address is passed in C++
& Reference is passed in Java

eg:



Inorder: 20 10 40 30 50

Inorder (10)
→ Inorder (20)
→ Inorder (NULL)
→ print (20)
→ Inorder (NULL)
→ print (10)
→ Inorder (30)
→ Inorder (40)
→ Inorder (NULL)
→ print (40)
→ Inorder (NULL)
→ print (30)
→ Inorder (50)
→ Inorder (NULL)
→ print (50)
→ Inorder (NULL)

Time Complexity: O(n)

Auxiliary Space: O(h)

N nodes will be visited each time doing some constant amount of work.

• Implementation of PreOrder Traversal

C++
```
void preorder (Node *root) {
    if (root != NULL)
    {
        cout << (root -> key) << " ";
        preorder (root -> left);
        preorder (root -> right);
    }
}
```

Time Complexity: O(n)
Auxiliary Space: O(h)

• Implementation of Post Order Traversal

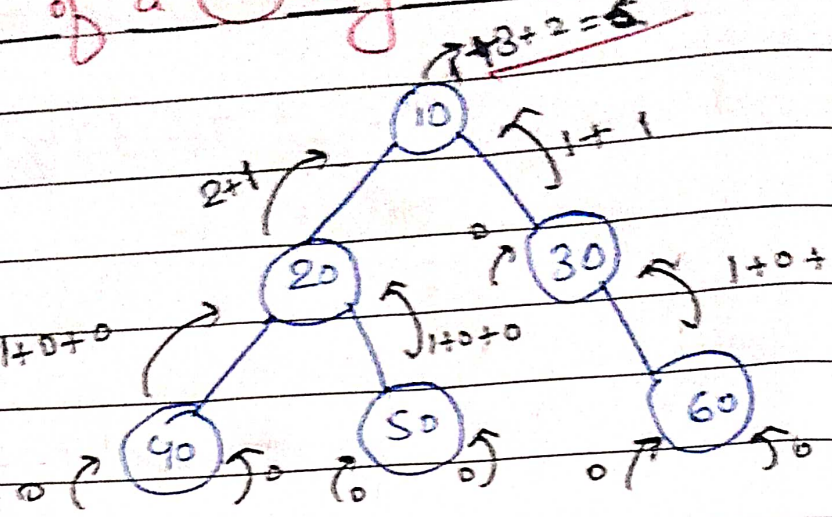
C++
```
void postorder (Node *root)
{
    if (root != NULL)
    {
        postorder (root -> left);
        postorder (root -> right);
        cout << (root -> key) << " ";
    }
}
```

→ Exact time

Time Complexity: O(n) or Θ(n)
Auxiliary space: O(h) or Θ(h)

h = height of the tree.

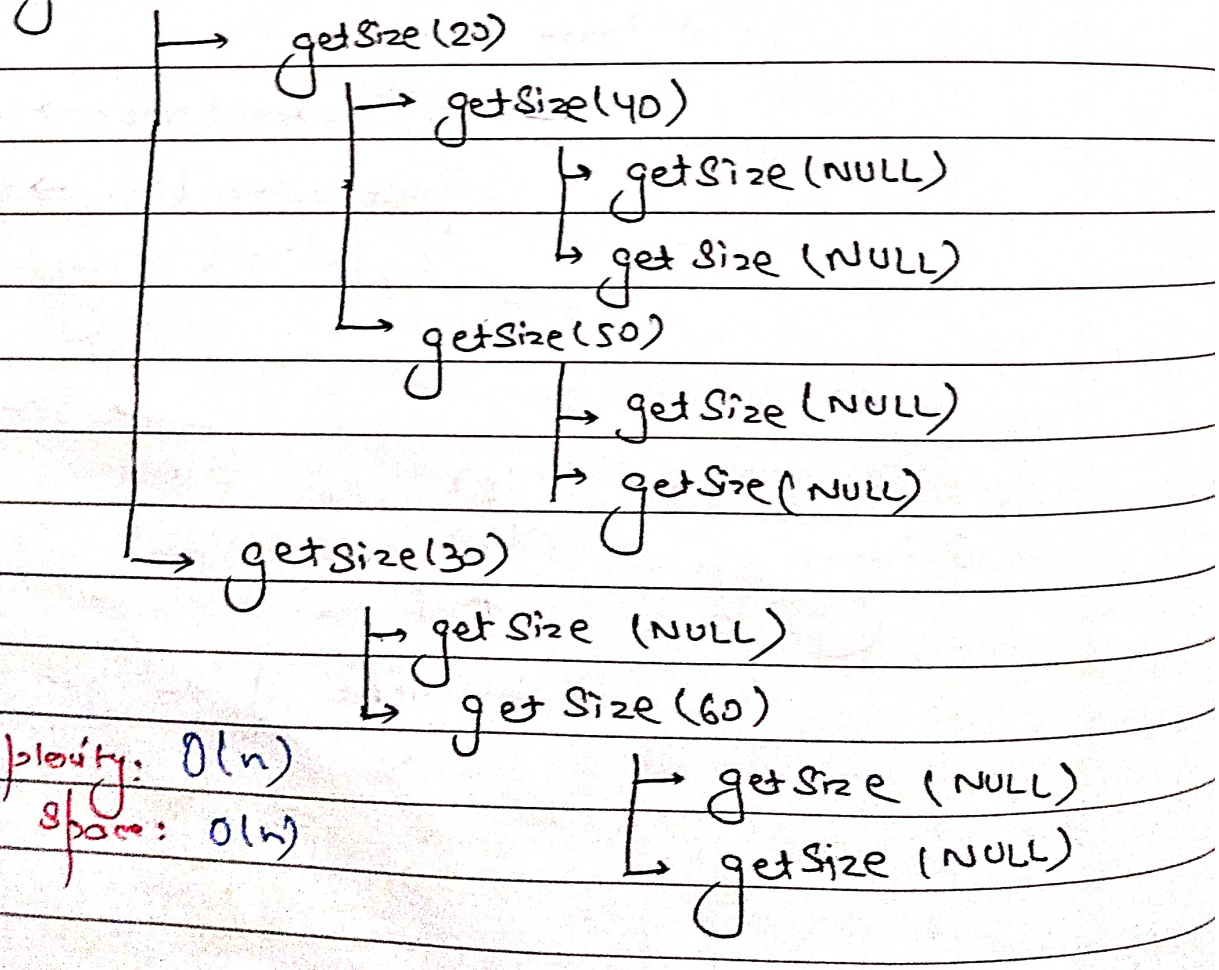
• Size of a Binary Tree



C++

```
int getSize (Node *root)
{
    if (root == NULL)
        return 0;
    else
        return 1 + getSize(root->left) +
               getSize(root->right);
}
```

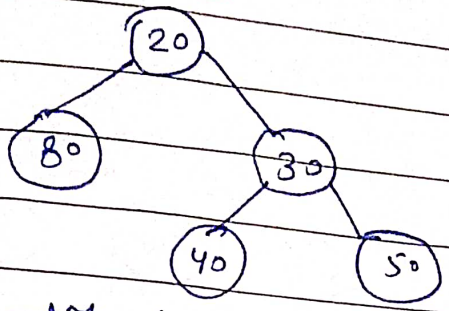
• getSize(10)



Time Complexity: O(n)

Auxiliary Space: O(n)

• Maximum in Binary Tree



C++

```
int getMax (Node *root)
{   if (root == NULL)
        return INT_MIN;
    else
        return max(root->key, max(getMax(root->left),
                                  getMax(root->right)));
```

(-∞)

Java

```
int getMax (Node root)
{   if (root == null)
        return Integer.MIN.VALUE;
    else
        return Math.max(root.key, Math.max(getMax(root.left),
                                           getMax(root.right)));
}
```

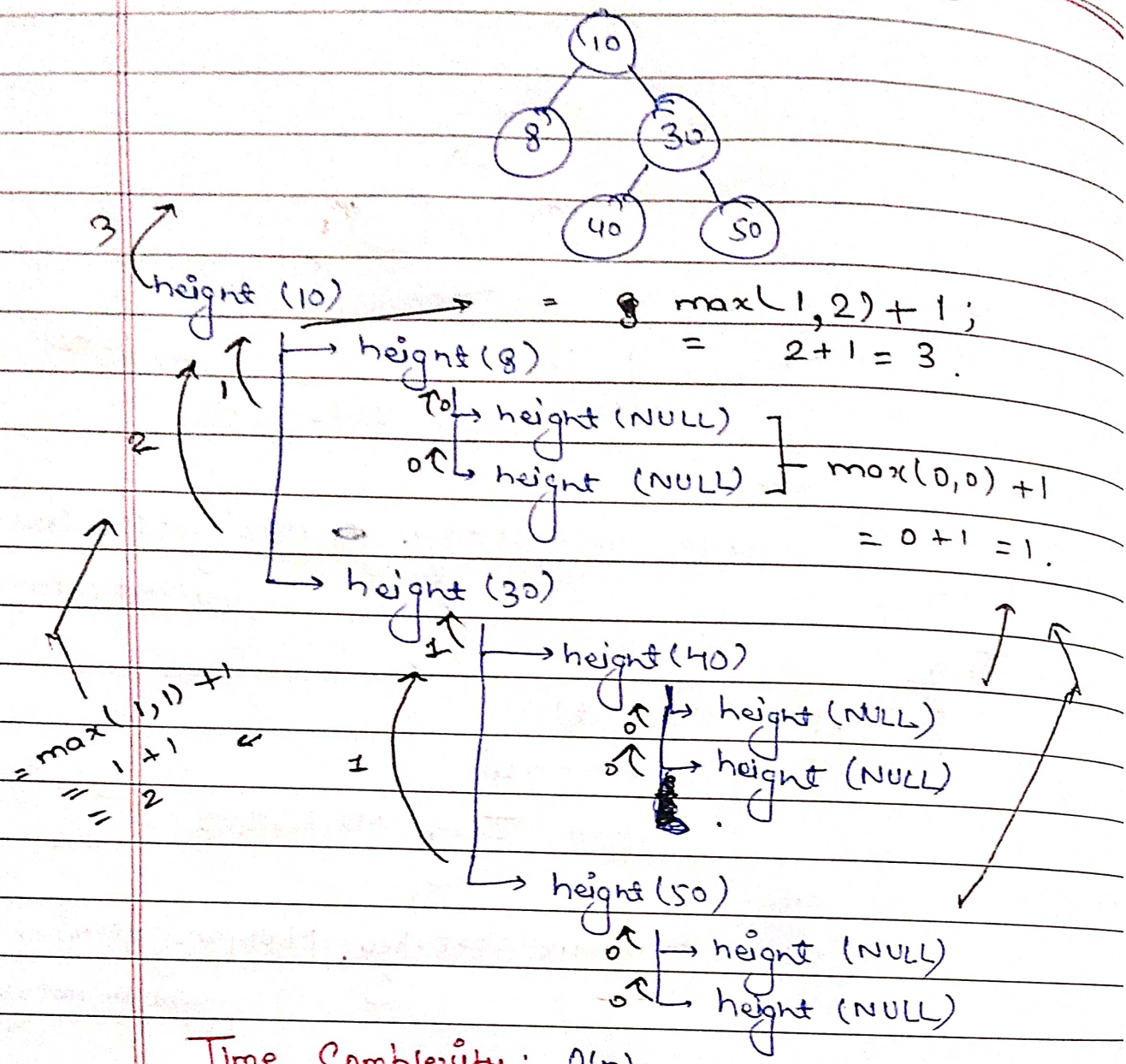
(-∞)

• Height of a Binary Tree

C++

```
int height (Node *root)
{   if (root == NULL)
        return 0;
    else
        return max(height(root->left), height(root->right))+1;
```

Recursion Tree for Height of a Binary Tree.



Time Complexity : O(n)

Auxiliary Space : O(h)

- ~~Some~~ Applications of Tree :

- To represent hierarchial data
- Binary Search Trees
- Binary Heap
- B and B+ Trees in DBMS
- Spanning and shortest path in Computer Networks