NON-LINEAR DATA STRUCTURES

BY
GEORGE ASANTE

OUTLINE

- Tree
- Binary Search Tree
- Heap

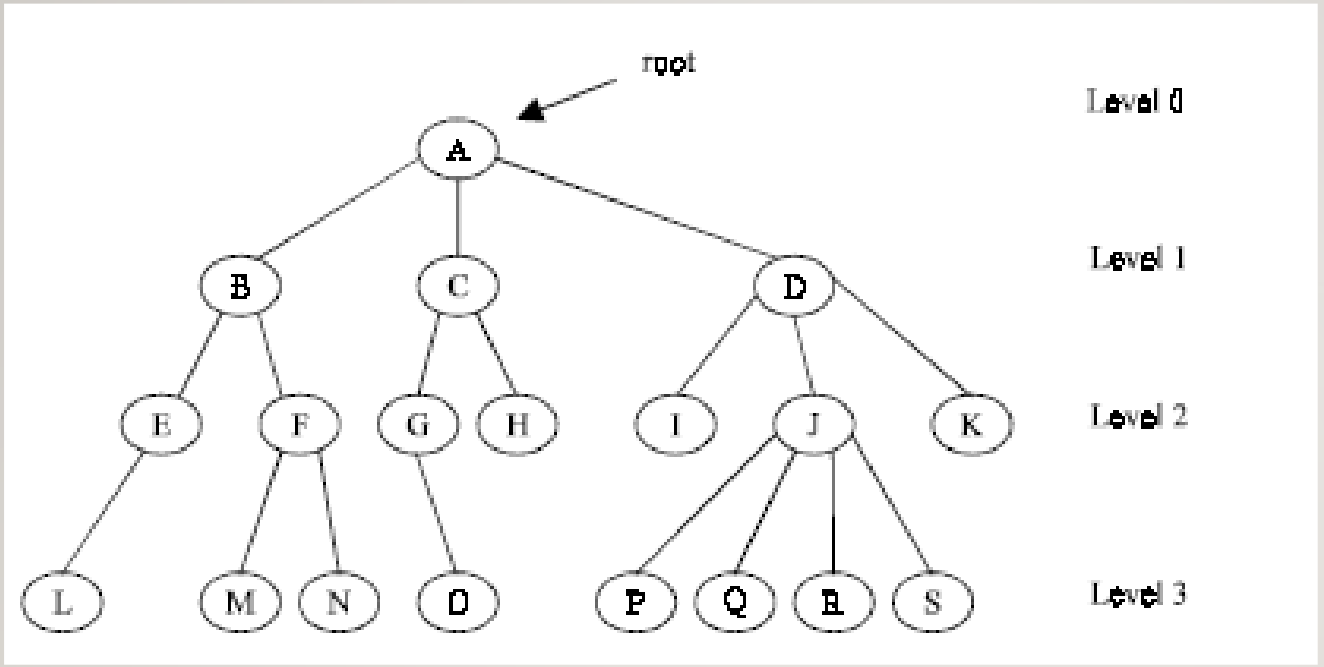
TREE

A Tree is a non-linear data structure that consist of a root node and potentially many levels of additional nodes that form a hierarchy.

A tree can be empty with no nodes called the **null** or **empty** tree

Properties of a tree

- One node is considered as a root
- Every node (excluding the root) is connected by a directed edge from exactly one other node (parent to child).


root
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
Level 0
Level 1
Level 2
Level 3

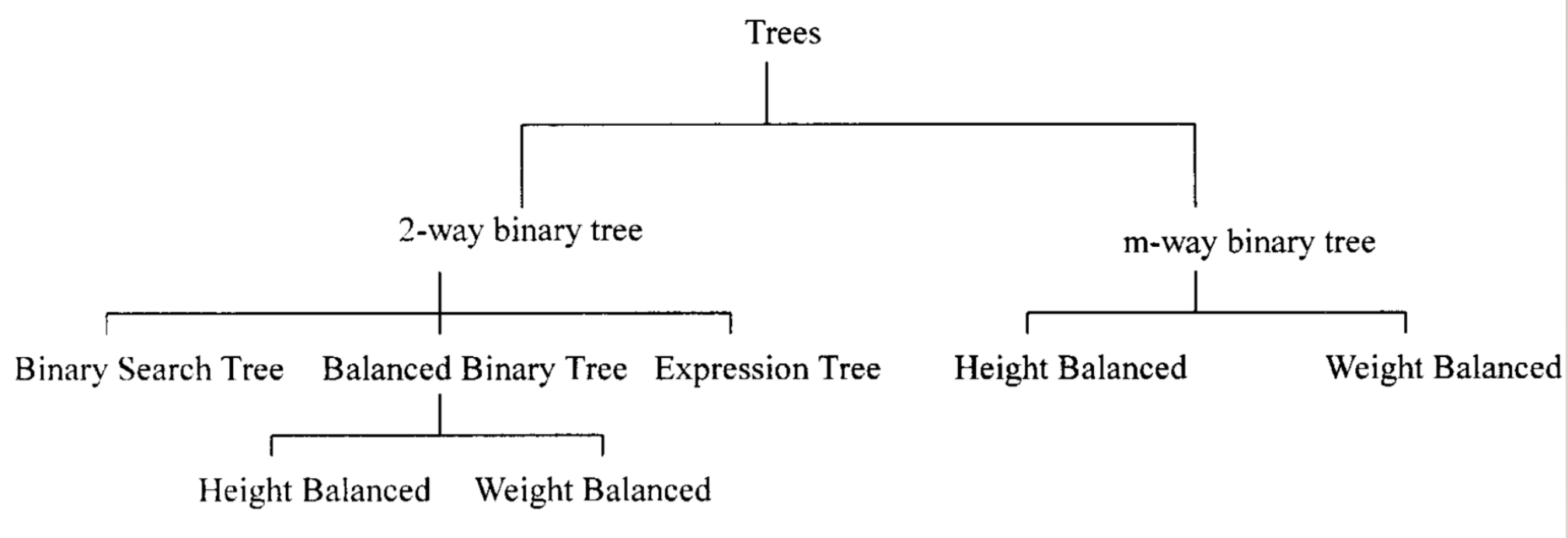
TERMINOLOGIES ASSOCIATED WITH TREES

- **Root**: It is the first node in the hierarchical arrangement of the data items;
  - the topmost node in the tree.
- **Parent**: a node that has a child.
- **Siblings**: nodes with the same parents.
- **Leave (terminal node):** any node without a child or node with degree zero.
- **Internal node**: all nodes apart from the leaves and roots are considered as internal nodes
- **Degree of a node**: the number of children or the number of subtrees of a node in a given tree.

TERMINOLOGIES ASSOCIATED WITH TREES

- **Degree** of a tree is the maximum degree of node in a given tree.
- **Edge**: the connection between two nodes.
- **Depth of a node**: The depth of a node M in the tree is the length of the path from the root of the tree to M.
  - All nodes of depth d are at level d in the tree. The root is the only node at level 0, and its depth is 0.
- **Height of a tree**: The height of a tree is one more than the depth of the deepest node in the tree.
- **Forest**: the connection of trees or set of n>0 disjoint tree. If a root is removed from a tree, it forms a forest

CLASSES OF TREES

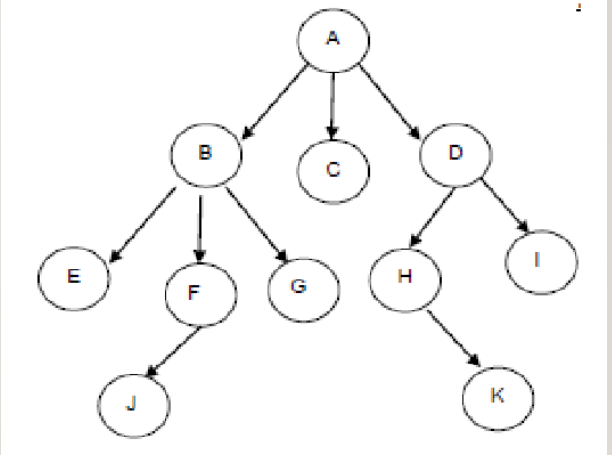


LIST REPRESENTATION OF A TREE

In representing a tree as a list, the information in the root node comes first followed by a list of the subtrees of that node. Parenthesis are used to denote children of a node. And comma is used to separate siblings.

The tree below can be represented as a list as :

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



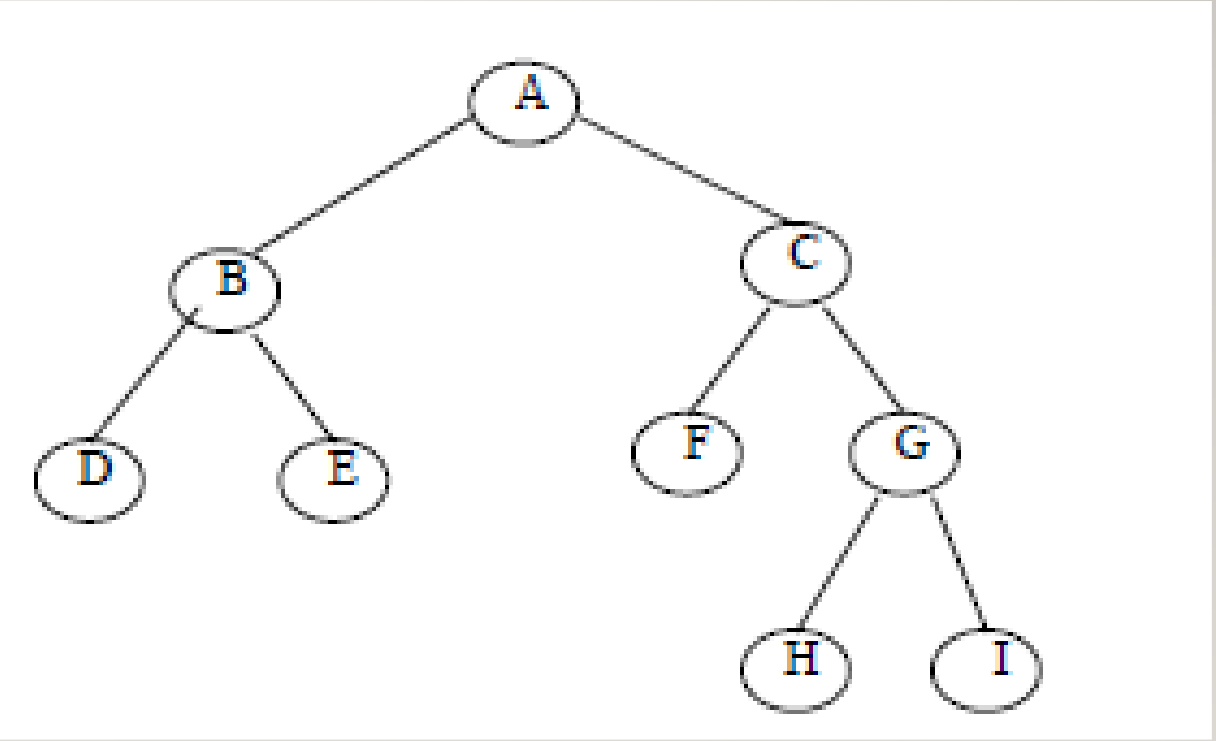
BINARY TREE

Binary tree is a special type of a tree where no node of the tree can have a degree more than two.

Typically, these children are described as “left child” and “right child”.

FOR A BINARY TREE:

- The maximum number of nodes at level i will be 2^i.
- The root is at level 0, so at that level the maximum number of nodes at that level is $2^0 = 1$.
- The maximum number of nodes in a binary tree is $2^k - 1$.
  - Where k is the height of the tree.

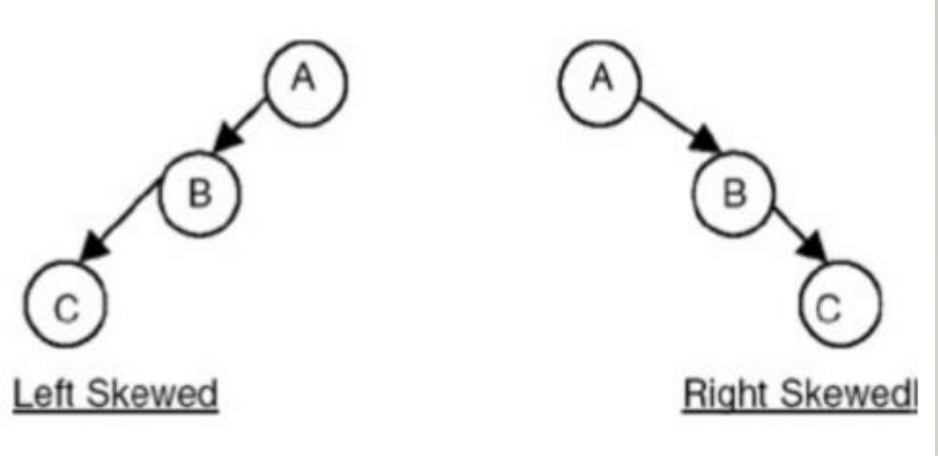


TYPES OF BINARY TREE

Skewed Binary Tree

If a binary tree has only left sub trees, then it is called left skewed binary tree.

If a binary tree has only right sub trees, then it is called right skewed binary tree.

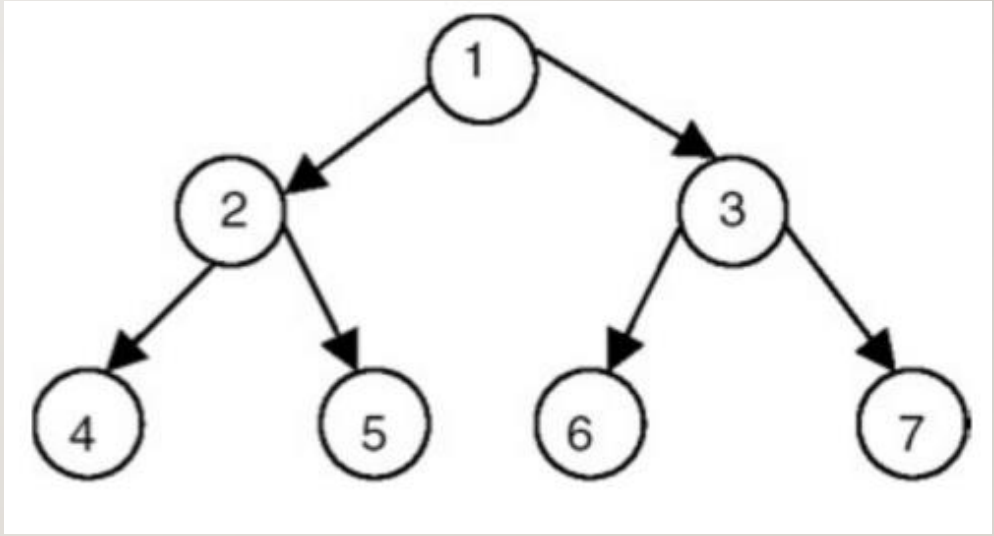


TYPES OF BINARY TREE

Full Binary Tree

A binary tree is a full binary tree if and only if all its levels are completely filled. That means

- Each non-leafy node has exactly two children
- All leaf nodes are at the same level



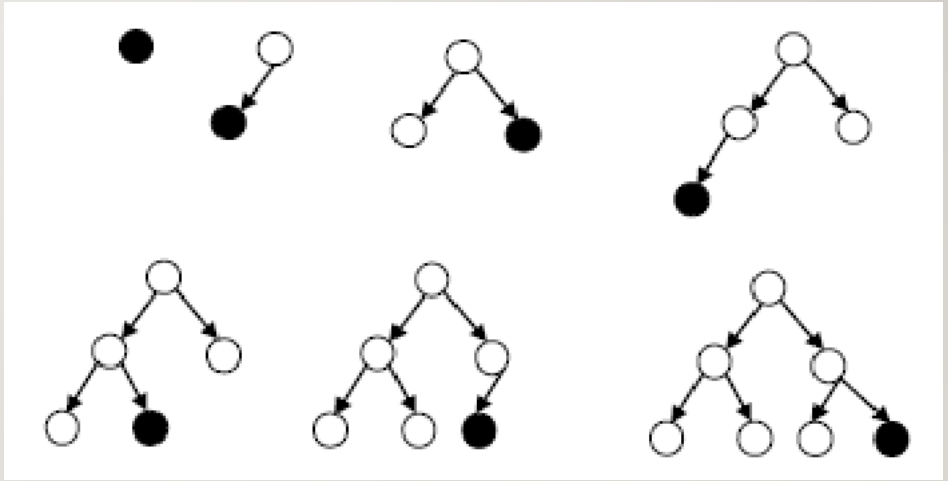
TYPES OF BINARY TREE

Complete Binary Tree

A complete binary tree has a restricted shape obtained by starting at the root and filling the tree by levels from left to right.

In the complete binary tree, all levels except possibly the bottom are completely filled.

The bottom level has its nodes filled in from left to right.



REPRESENTATION OF BINARY TREES

- Array representation
- Linked list representation

ARRAY REPRESENTATION OF BINARY TREE

- An array can be used to store the nodes of a binary tree.
- The nodes stored in an array of memory can be accessed sequentially.
- If a binary tree is a *complete binary tree*, it can be represented using an array which is capable of holding n elements where n is the number of nodes in a complete binary tree.

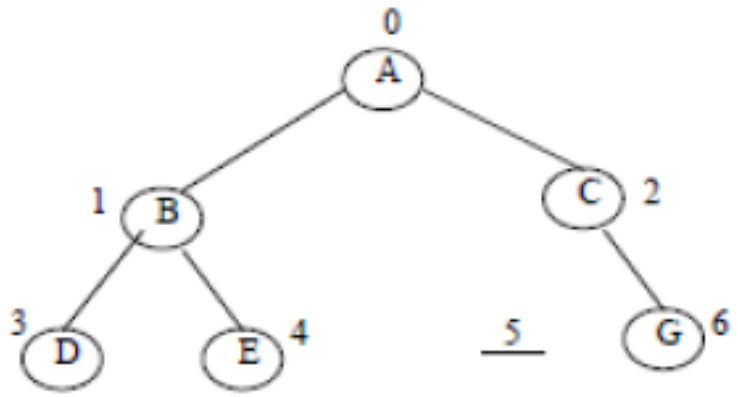
ARRAY REPRESENTATION OF BINARY TREE

Following is the rule to decide the location of any node of a tree in the array.

- The root node is at location 0
- For any node with index i, $0 < i \leq n$
  1. parent(i)=(i-1)/2 [note: Truncate decimal point eg. 2-1/2=0.5=0]
  2. Left child (i) = 2*i +1
  3. Right child (i) =2*i+2

- If the left child is at array index n, then its right brother is at (n + 1). Similarly, if the right child is at index n, then its left brother is at (n – 1).

Let consider the tree below:



The above tree can be represented using array as shown below"

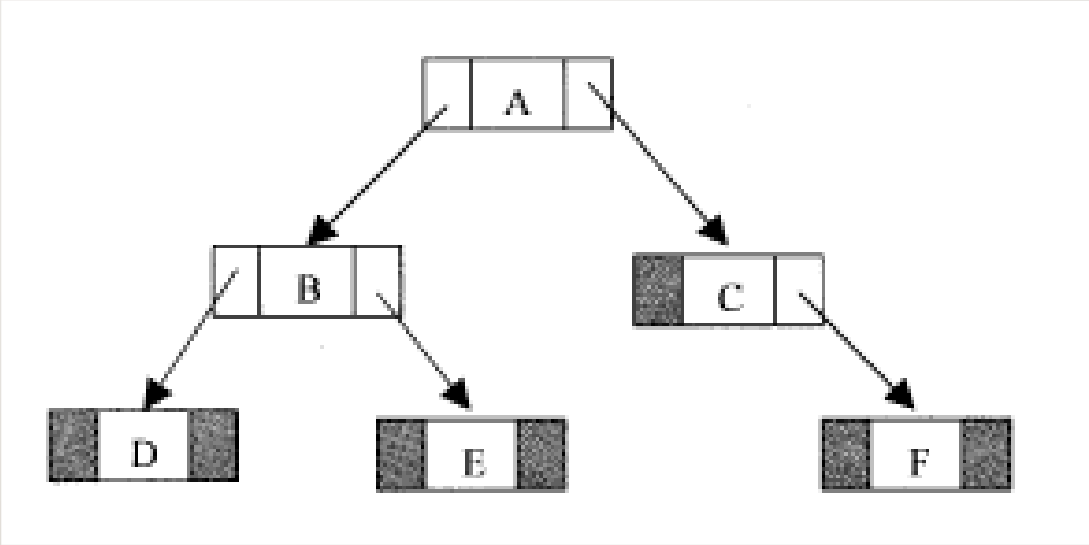
A[ ]	A	B	C	D	E		F
	[0]	[1]	[2]	[3]	[4]	[5]	[6]

LINKED LIST REPRESENTATION OF BINARY TREE

The most popular and practical way of representing a binary tree is using linked list. In linked list, every element is represented as node. A node consists of three fields such as:

a) Left Child pointer
b) Information of the Node (Info)
c) Right Child pointer

The left child pointer holds the address of the left child node of the parent, info holds the information of every node and right child pointer holds the address of the right child node of the parent.



OPERATIONS ON BINARY TREE

1. Create an empty Binary Tree
2. Traversing a Binary Tree
3. Inserting a New node
4. Deleting a Node
5. Searching for a Node
6. Copying the mirror image of a tree
7. Determine the total no: of Nodes
8. Determine the total no: leaf Nodes
9. Determine the total no: non-leaf Nodes
10. Find the smallest element in a Node
11. Finding the largest element
12. Find the Height of the tree
13. Finding the Father/Left Child/Right Child/Brother of an arbitrary node

TRAVERSING BINARY TREES

Tree traversal is one of the most common operations performed on tree data structures. It is a way in which each node in the tree is visited exactly once in a systematic manner. There are three standard ways of traversing a binary tree. They are:

1. Pre Order Traversal (Node-left-right)
2. In order Traversal (Left-node-right)
3. Post Order Traversal (Left-right-node)

TRAVERSING BINARY TREES

Pre order traversal

To traverse a non-empty binary tree in pre order following steps one to be processed.

1. Visit the root node
2. Traverse the left sub tree in preorder
3. Traverse the right sub tree in preorder

That is, in preorder traversal, the root node is visited (or processed) first, before traveling through left and right sub trees recursively.

In Order Traversal

The in order traversal of a non-empty binary tree is defined as follows:

1. Traverse the left sub tree in order
2. Visit the root node
3. Traverse the right sub tree in order

In order traversal, the left sub tree is traversed recursively, before visiting the root. After visiting the root, the right sub tree is traversed recursively, in order fashion.

TRAVERSING BINARY TREES

Post Order Traversal

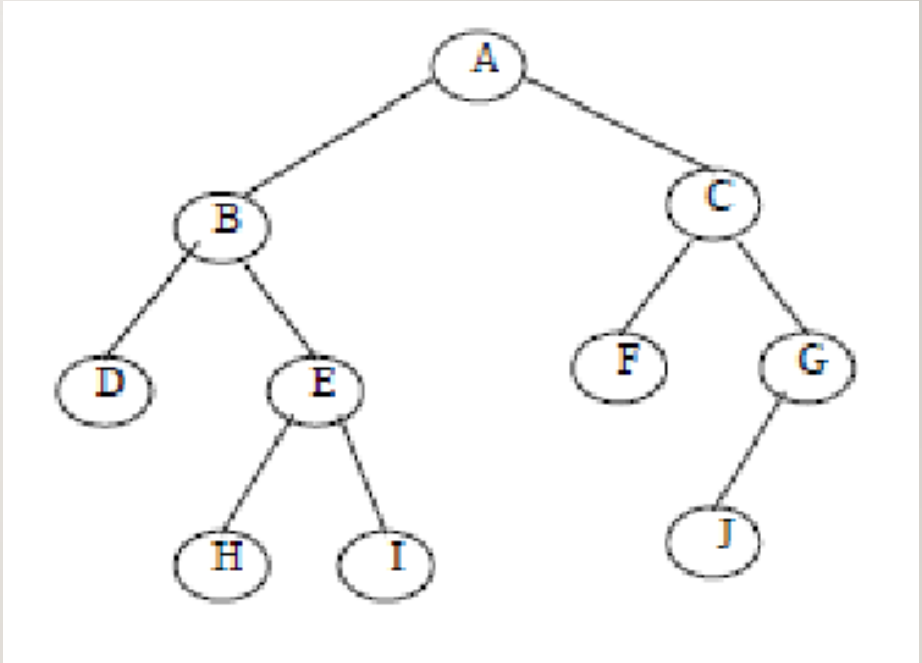
The post order traversal of a non-empty binary tree can be defined as :

1. Traverse the left sub tree in post order
2. Traverse the right sub tree in post order
3. Visit the root node

In Post Order traversal, the left and right sub tree(s) are recursively processed before visiting the root.

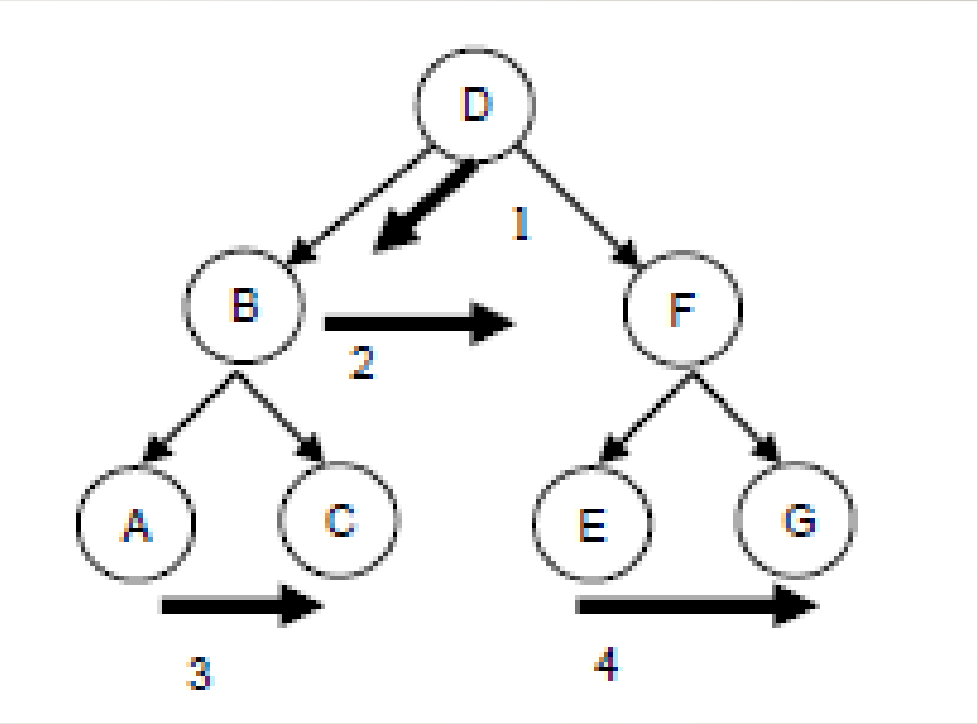
EXAMPLE

- The preorder traversal is A, B, D, E, H, I, C, F, G, J
- The in order traversal is D, B, H, E, I, A, F, C, J, G.
- The post order traversal is D, H, I, E, B, F, J, G, C, A



TRAVERSING BINARY TREES

- **Breadth First Traversal**
  Trees can also be traversed in ***level-order***, where we visit every node on a level before going to a lower level. This is also called ***breadth first traversal***.
  The breadth first traversal of the previous tree is D, B, F, A, C, E, G
- **Depth First Traversal**
  The preorder, in order and post order traversals are types of depth first traversal.



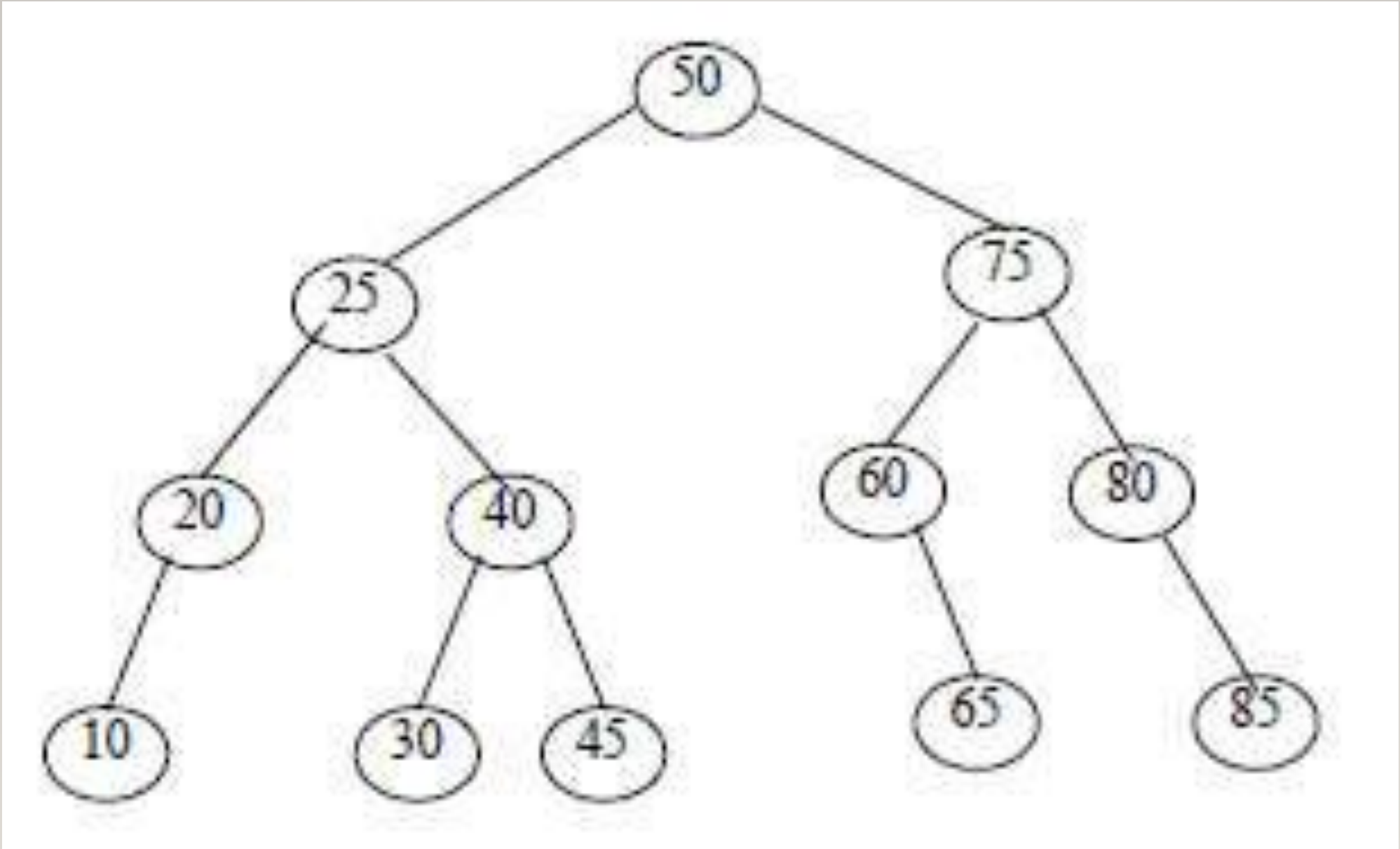
BINARY SEARCH TREE

A Binary Search Tree (BST) or **ordered binary tree** is a tree in which all the nodes have the following properties:

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- he right sub-tree of a node has a key greater than or equal to its parent node's key.
- Both the left and right subtrees must also be binary search trees.

Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as

left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)


50
25
75
20
40
60
80
10
30
45
65
85

OPERATIONS ON BINARY SEARCH TREE

- **Searching for an element**
  Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.
- **Inserting an element**
  Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data
- **Deleting or Removing an element from a Binary Search Tree**
  If we are trying to delete a leaf node just delete that node and the rest of the tree is left as it is earlier. If the node has one child, then the child will be linked to the parent of the deleted node. If the node has two children here, we have to either choose the largest value of the left subtree or smallest value of the right subtree to replace the deleted node.

HEAP

- A heap is a tree-based data structure in which all the nodes of the tree are in a specific order.
- The maximum number of children of a node in a heap depends on the type of heap.

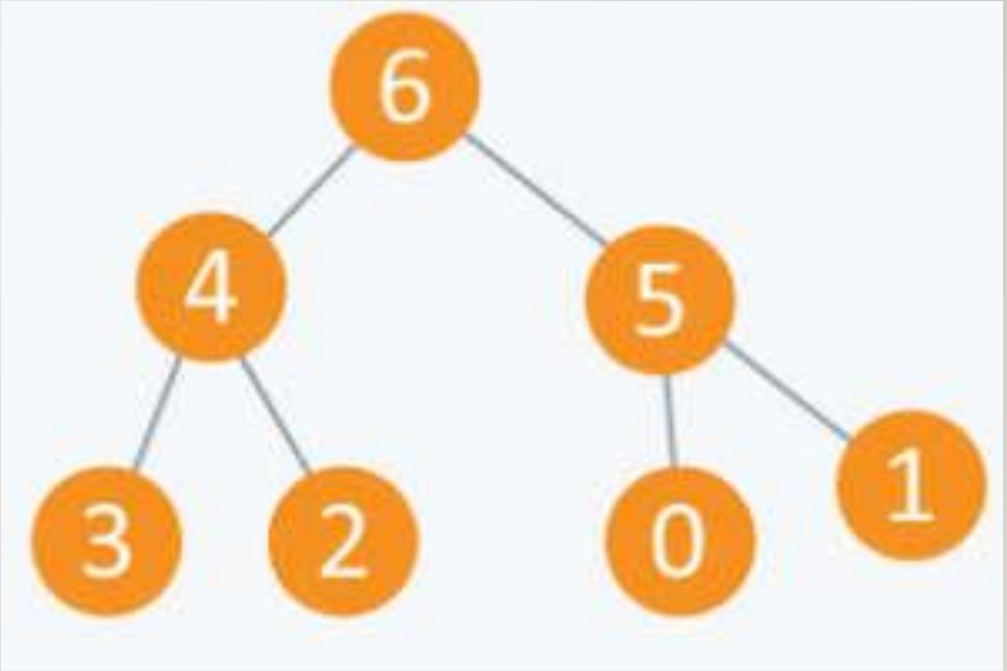
Types Of Heap

-Max Heap

-Min Heap

MAX HEAP

- In this type of heap, the value of parent node will always be greater than or equal to the value of child node across the tree and the node with highest value will be the root node of the tree. The above heap is an example of Max Heap.



MIN HEAP

- In this type of heap, the value of parent node will always be less than or equal to the value of child node across the tree and the node with lowest value will be the root node of tree