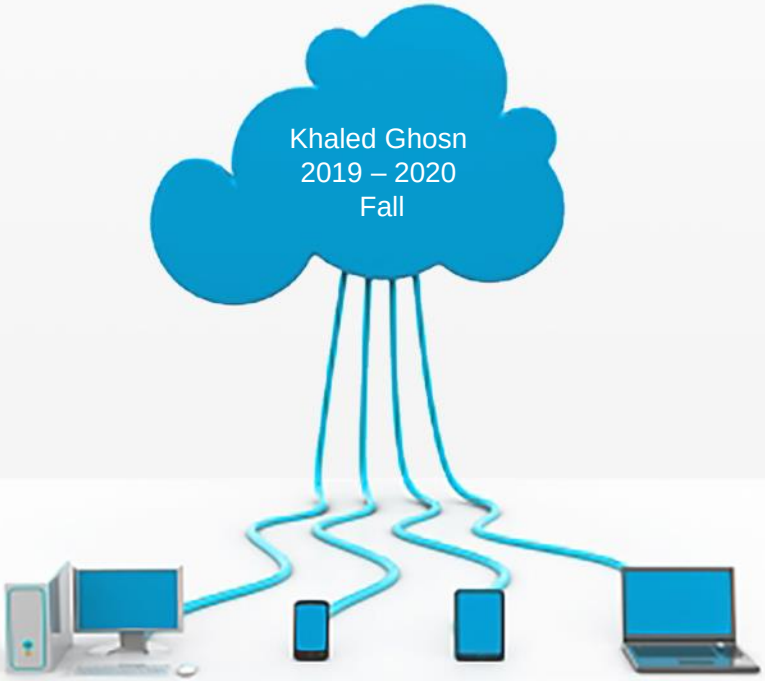
ARTS, SCIENCES & TECHNOLOGY
UNIVERSITY IN LEBANON
AUL

Binary Trees

Binary Trees & Traversal

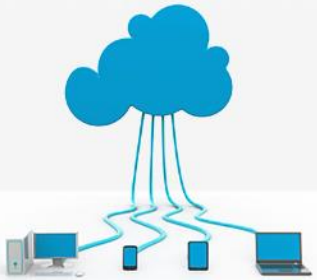
Khaled Ghosn
2019 – 2020
Fall



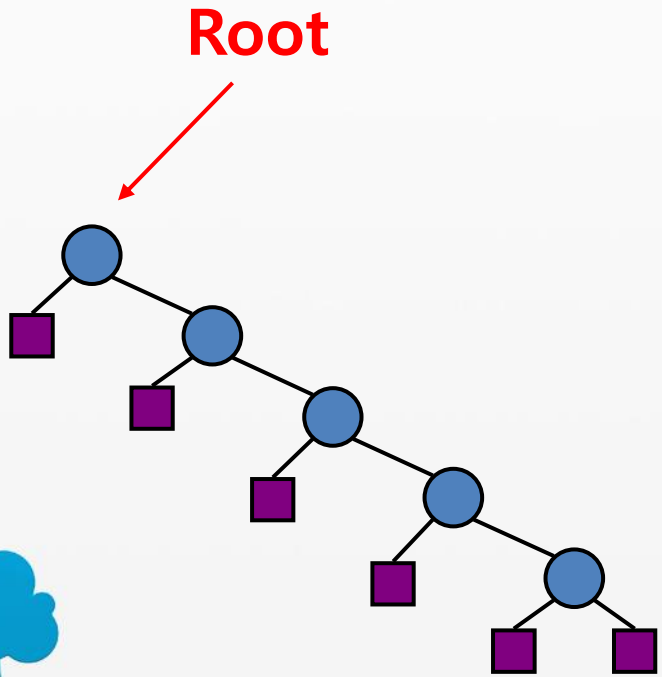
Binary Tree

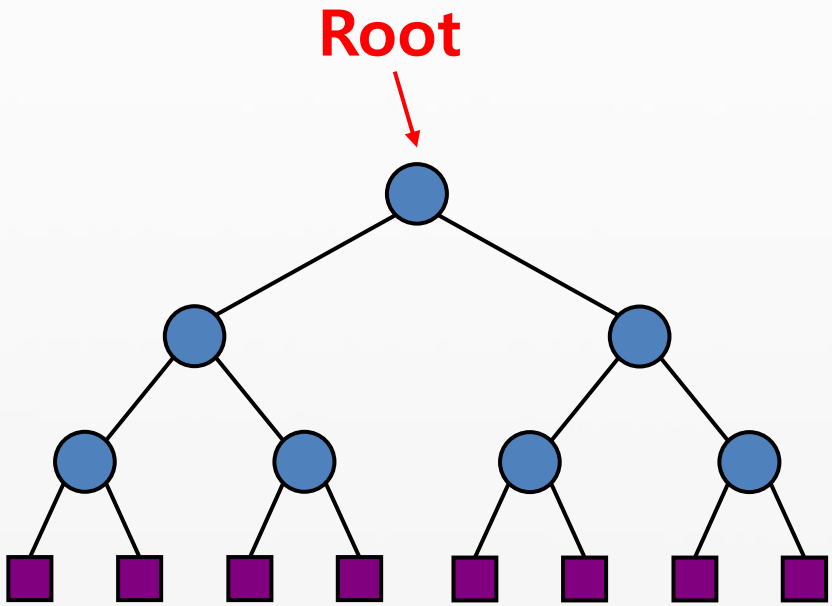
Definition

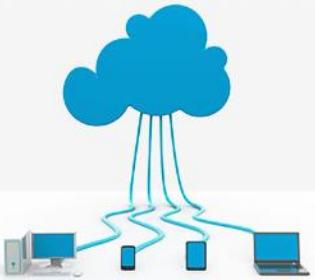
- ✓ The simplest form of a tree in which every node has at most two children (can be less)
- ✓ A binary tree is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element
  - The first child is referred to as the left child, while the second child is referred to as the right child
  - The left and right pointers recursively point to smaller "sub-trees" on either side
  - A left child precedes a right child in the order of children of a node
- ✓ The subtree rooted at a left or right child of an internal node is called a ***left subtree*** or ***right subtree***, respectively, of *the node*.



Binary Tree



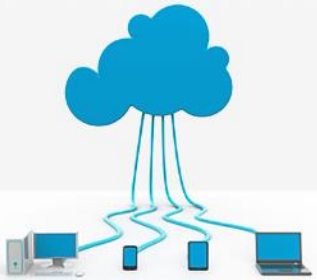




Binary Tree

Definition

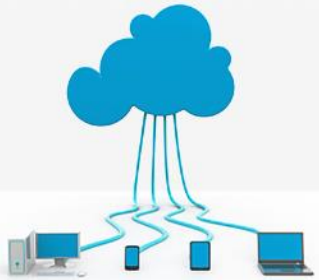
- ✓ The "root" pointer points to the topmost (highest) node in the tree
  - A null pointer represents a binary tree with no elements – the empty tree
- ✓ A binary tree is ***proper*** if each node has either zero or two children, also called ***full binary tree***
  - Thus, in a proper binary tree, every internal node has exactly two children
- ✓ A binary tree that is not proper is ***improper***.



Binary Tree

Binary Node operations

Element	
Left	Right



BinaryNode

- *Anytype* **Element**
- *BinaryNode* **Left**
- *BinaryNode* **Right**
- ~~*boolean* **Leaf**~~

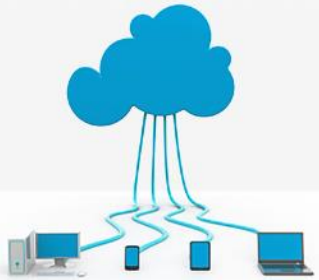
+ **BinaryNode** (*Anytype* newValue)

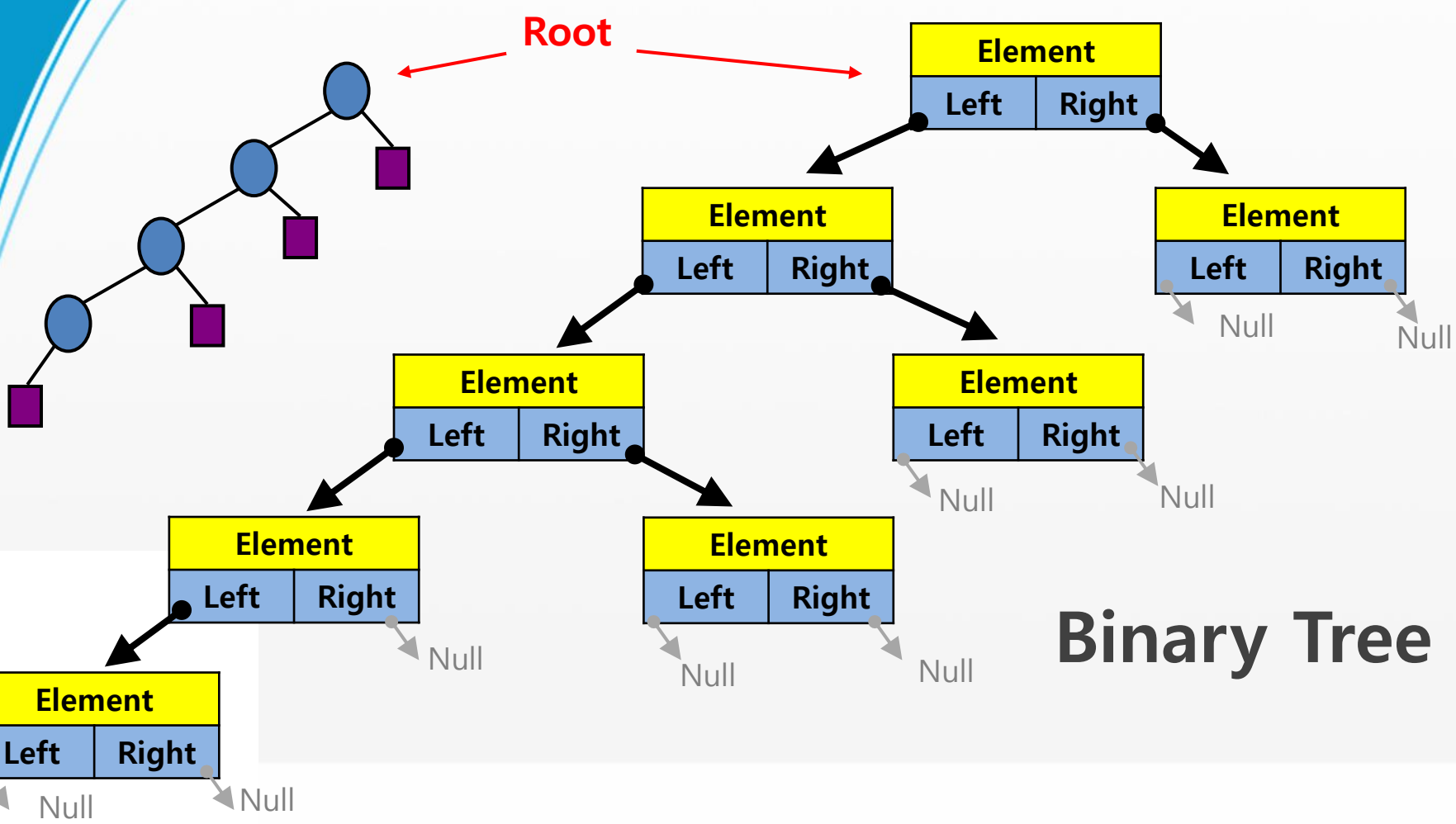
+ *Anytype* **getElement** ( )
+ *void* **setElement** (*Anytype* newValue)
+ *BinaryNode* **getLeft** ( )
+ *void* **setLeft** (*Anytype* newValue)
+ *BinaryNode* **getRight** ( )
+ *void* **setRight** (*Anytype* newValue)
+ *boolean* **isLeaf** ( )
+ *void* **PrintPreorder** ( )
+ *void* **PrintInorder** ( )
+ *void* **PrintPostorder** ( )
+ *int* **Size** ( )
+ *int* **Height** ( )

Binary Tree

Binary Tree operations

BinaryTree
- *BinaryNode* **Root**
+ **BinaryTree** (*Anytype* newValue) + **BinaryNode** (*BinaryNode* Root) + *Anytype* **getRoot** ( ) + *void* **setRoot** (*Anytype* newValue) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *void* **PrintPreorder** ( ) + *void* **PrintInorder** ( ) + *void* **PrintPostorder** ( ) + *int* **Size** ( ) + *int* **Height** ( )

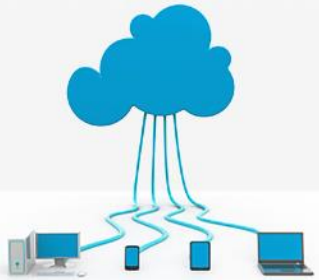



Root
Element
Left
Right
Element
Left
Right
Element
Left
Right
Null
Null
Element
Left
Right
Element
Left
Right
Null
Null
Element
Left
Right
Null
Element
Left
Right
Null
Element
Left
Right
Null
Null
Binary Tree

Binary Tree Traversal

Definition

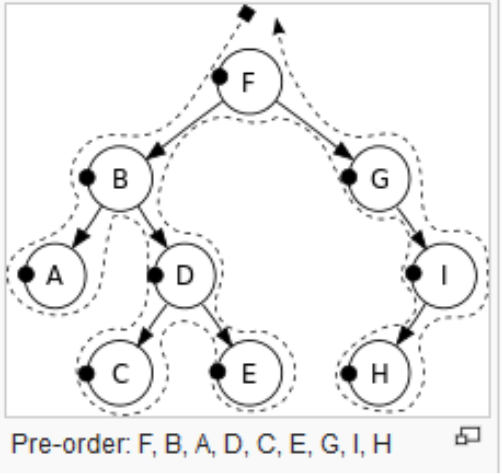
- ✓ Traversing: means to visit all the nodes in a specified order
- ✓ In a traversal, each element of the tree is **visited** exactly once
- ✓ During the visit of an element, all actions (make a copy, display, evaluate the operator, etc.) with respect to this element can be taken
- ✓ Complexity: O (n)
- ✓ Four simple ways to traverse a Binary Tree:
  - **Pre-order**
  - **Post-order**
  - **In-order** (Depth-First)
  - **Level-order** (Breadth-First)



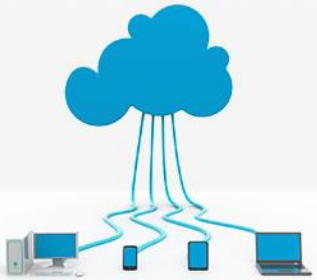
Binary Tree Traversal

Pre-Order

- ✓ The root of the sub-tree is processed first before going into the left then right sub-tree
  - the node is processed and then its children are processed recursively
- ✓ NLR: **N**ode, **L**eft, **R**ight
- ✓ Steps:
  1. Visit the root
  2. Traverse left sub-tree
  3. Traverse right sub-tree

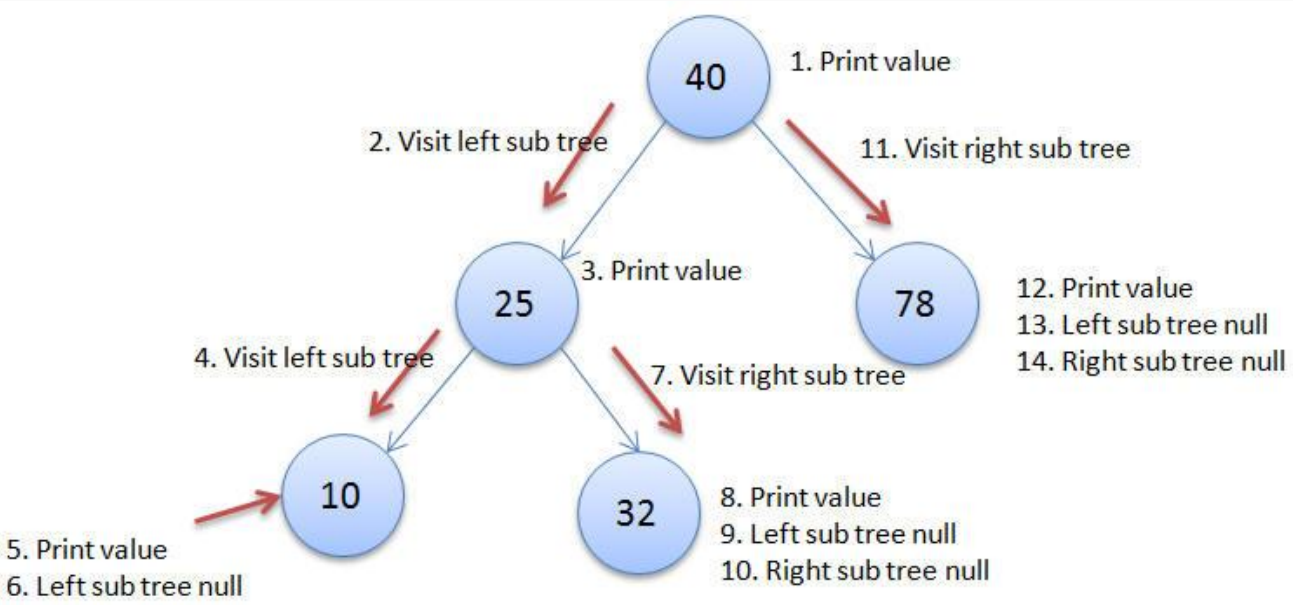


Pre-order: F, B, A, D, C, E, G, I, H

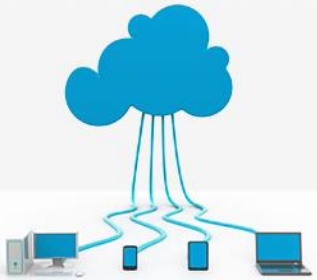


Binary Tree Traversal

Pre-Order



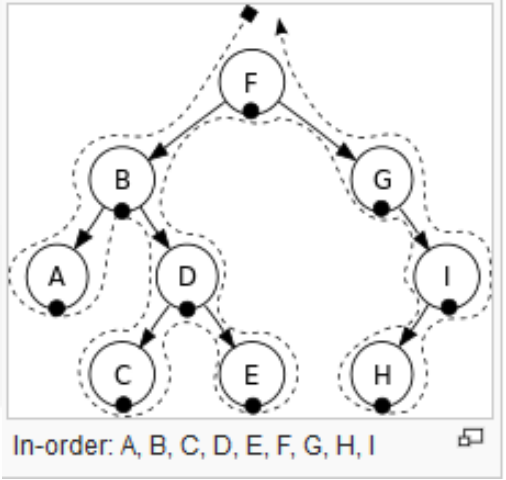
The above PREORDER traversal gives: **40, 25, 10, 32, 78**

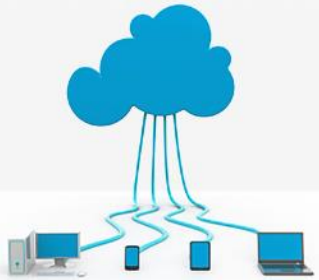


Binary Tree Traversal

In-Order

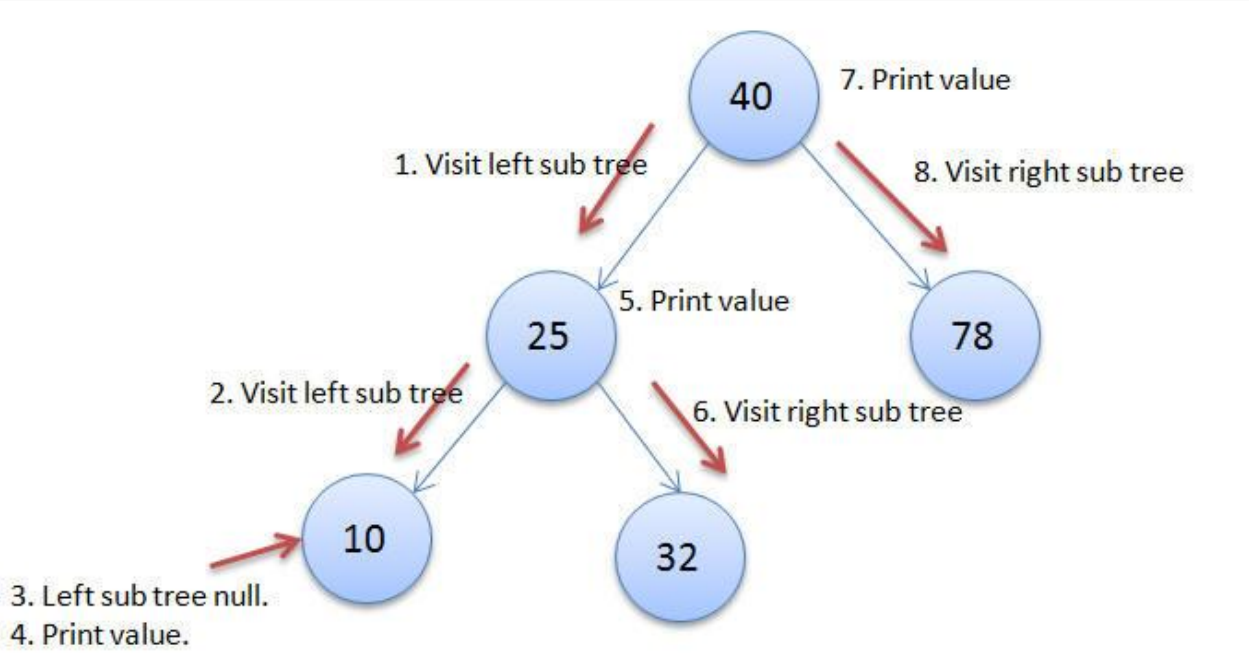
- ✓ The root is visited in-between left and right sub-tree traversal
  - after the complete processing of the left sub-tree the root is processed followed by the processing of the complete right sub-tree
  - the current node is processed between recursive calls
- ✓ LNR: **L**eft, **N**ode, **R**ight
- ✓ Steps:
  1. Traverse left sub-tree
  2. Visit the root
  3. Traverse right sub-tree



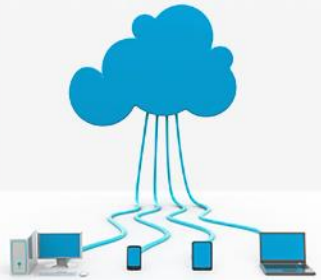


Binary Tree Traversal

In-Order



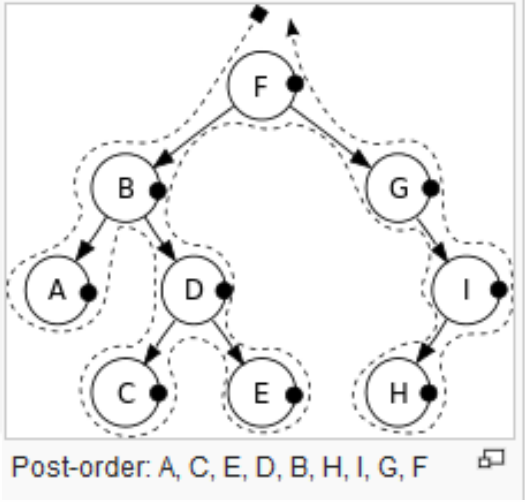
The above INORDER traversal gives: **10, 25, 32, 40, 78**

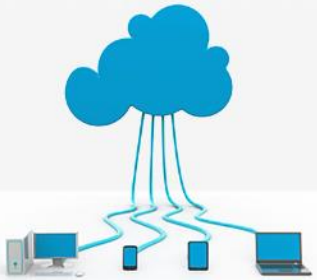


Binary Tree Traversal

Post-Order

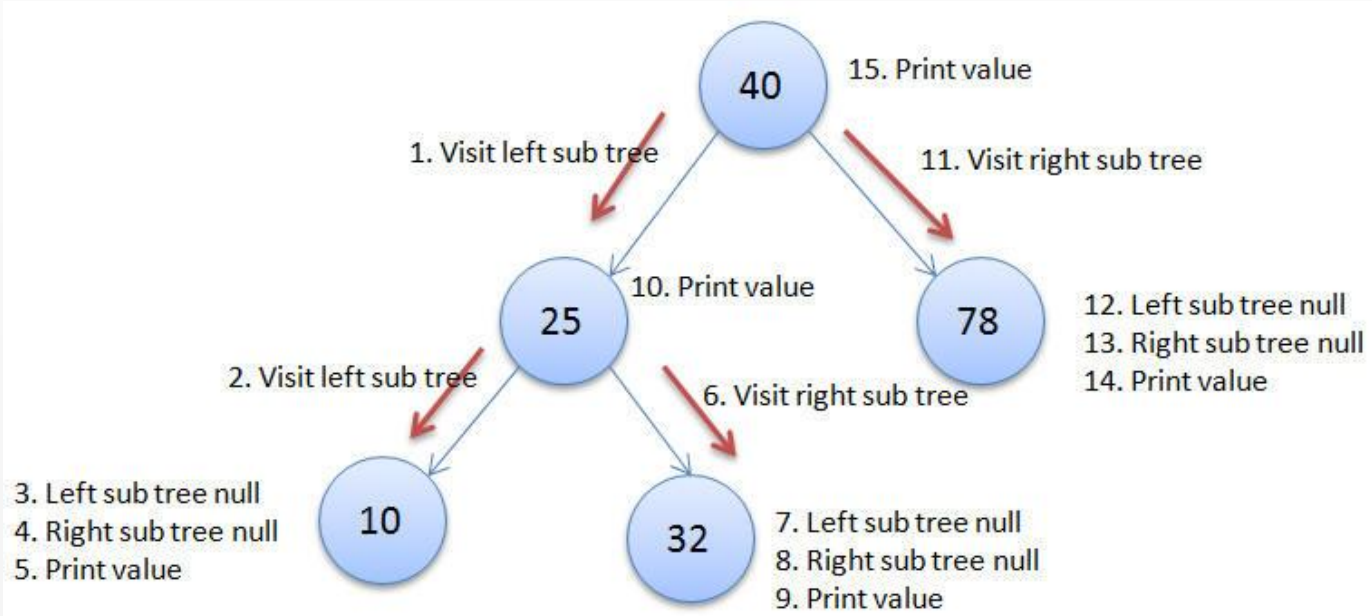
- ✓ In Preorder, the root is visited after (post) the sub-trees traversals
  - the root is processed only after the complete processing of the left and right sub-tree
  - the node is processed after both children are processed recursively
- ✓ LRN: **L**eft, **R**ight, **N**ode
- ✓ Steps:
  1. Traverse left sub-tree
  2. Traverse right sub-tree
  3. Visit the root



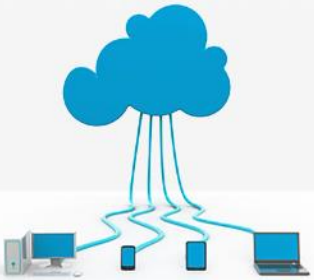


Binary Tree Traversal

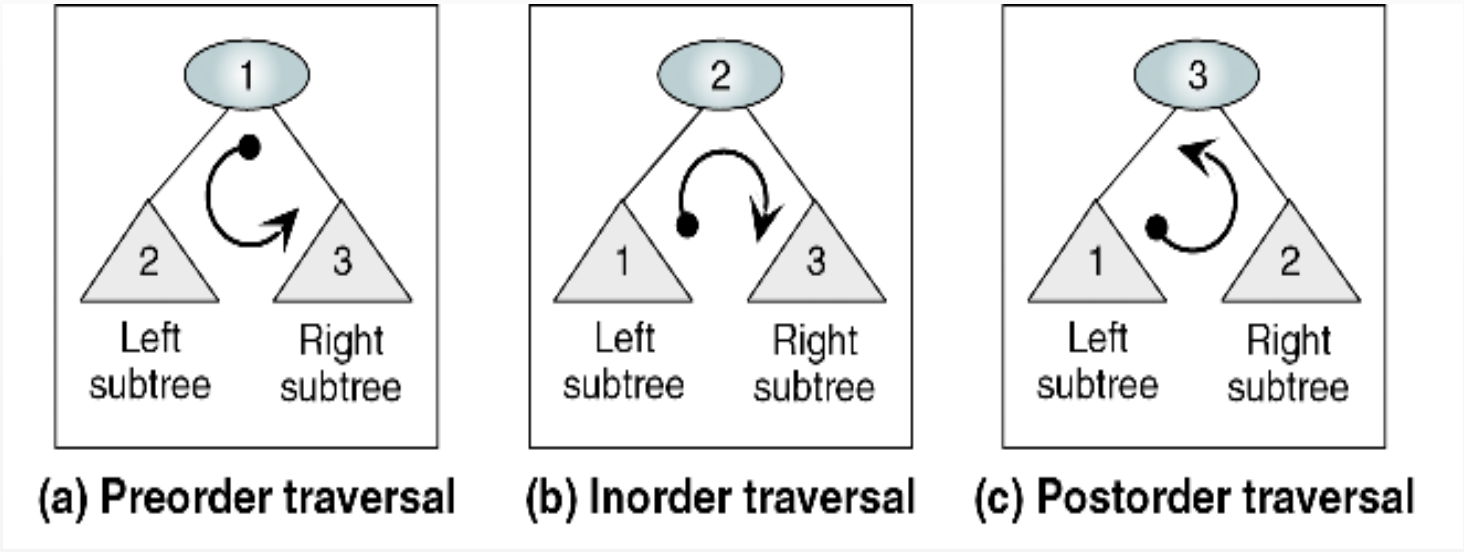
Post-Order



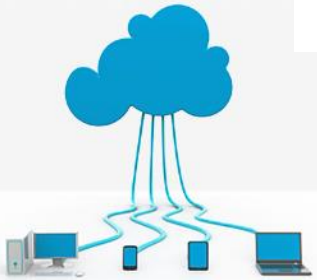
The above POSTORDER traversal gives: **10, 32, 25, 78, 40**



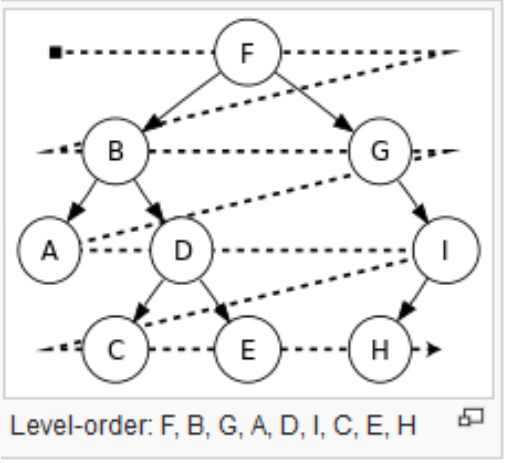
Binary Tree Traversal



(a) Preorder traversal (b) Inorder traversal (c) Postorder traversal

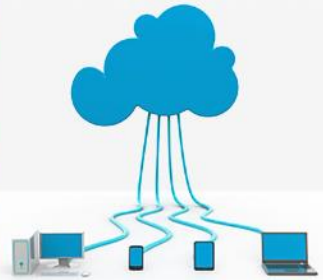


Tree Traversal



Level-Order

- ✓ Also known as: Breadth-First
- ✓ Visit all nodes at a given depth
  - Visits all nodes at depth k before proceeding onto depth k + 1
- ✓ The tree is processed by levels. So first all nodes on level i are processed from left to right before the first node of level i+1 is visited
- ✓ Can be implemented using a queue
- ✓ Run time is $O(n)$
- ✓ Memory is potentially expensive: maximum nodes at a given depth



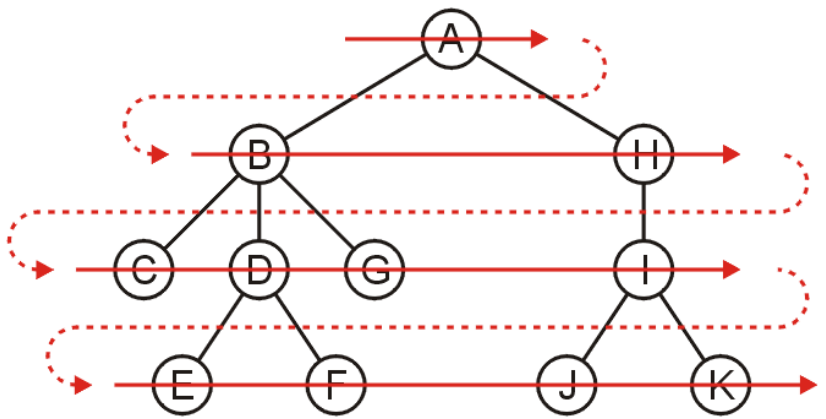
Tree Traversal

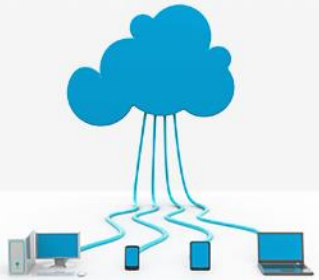
Level-Order

✓ Can be implemented using a **Queue**

- Create a queue and push the root node onto the queue
- While the queue is not empty:
  - Push all of its children of the front node onto the queue
  - Pop the front node

E.g. A B H C D G I E F J K





Tree Traversal

Exercise 1

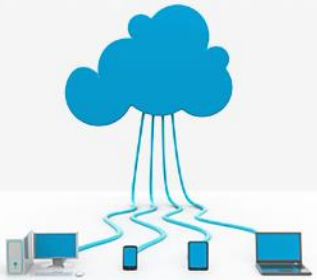
In the class BinaryTree, implement in java the method:

public void **PrintLevelOrder ( )**

Hint: it can be implemented in 2 ≠ ways:

1. The function **PrintLevelOrder ( )** implemented in the BinaryTree class calls the **Root.PrintLevelOrder ( )** ; i.e. you have to implement another function the TreeNode class

2. The function **PrintLevelOrder ( )** implemented in the BinaryTree class calls another private recursive function (with same name , in same class) and sends the root as a parameter



Tree Traversal

Exercise 2

In the BinaryTree class, implement in java, by using **STACK**, a method to traverse (and print) all the elements of a binary tree.

Note: You can choose any order of traversing