**Lab Tasks**

**Q1.** Create a BT (Binary Tree) of Nodes, with each Node having a data element, left pointer and right pointer. Then add the following functionalities to the BinaryTree class:

* Insert(val) -> Creates a node and inserts it at the first available position (prioritise left)
* Delete(val) -> Deletes the node which matches this value
* Search(val) -> returns the node if found, otherwise returns a null pointer.
* getHeight -> returns the height of the tree. Assume a tree of 1 node has height = 1.
* getSize -> returns the number of nodes in the tree.

**Q2.** Create a BST (Binary Search Tree) of Nodes, with each Node having a data element, left pointer and right pointer. Then add the following functionalities to the BinaryTree class:

* Insert(val) -> Creates a node and inserts it at the **appropriate position (smaller value on left, larger value on right)**
* Delete(val) -> Deletes the node which matches this value. **Optimise your search, it should be quicker since it’s a BST.**
* Search(val) -> returns the node if found, otherwise returns a null pointer. **Optimise your search, it should be quicker since it’s a BST.**
* getHeight -> returns the height of the tree. Assume a tree of 1 node has height = 1.
* getSize -> returns the number of nodes in the tree.

**Q3.** Perform the Inorder, Preorder, Postorder and Level-order traversals on a BST:

* Inorder: **(left, root, right)**
* Preorder: **(root, left, right)**
* Postorder: **(left, right, root)**
* Levelorder: **(1st level, 2nd level … until level = height)**

**Q4.** Write the following functions to be run on a BT, to check whether the tree is:

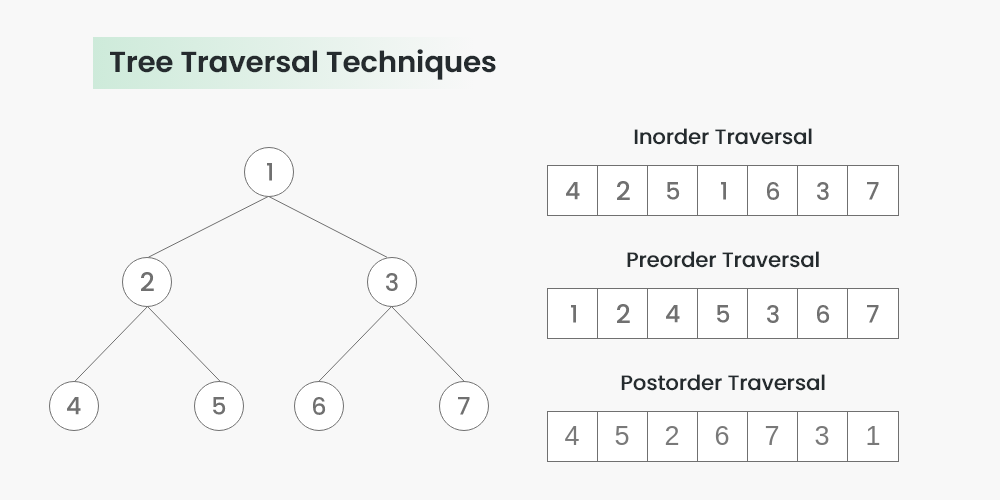
* A Binary Search Tree (left smaller than root, right greater than root, true for each root)
* A Full Binary Tree (every node has either 0 or 2 children)
* A Complete Binary Tree (all levels filled, last level could be unfilled, but left-leaning)
* A Perfect Binary Tree (every internal node has 2 children and all leaf nodes are at the same level)
* A Degenerate Binary Tree (every node has exactly 1 child - basically makes a linked list)

**Q5.** Write a function which takes 2 tree roots as argument, and returns whether both these trees are identical or not.

**Q6.** Write a function which returns the **Lowest Common Ancestor (LCA)** of 2 nodes in a BST. Lowest Common Ancestor can be described as the common parent furthest down from the root. In case the 2 node values are not found, the function should return a null pointer. Otherwise, return the node pointer of the LCA when found.

**Q7.** Write a function which takes a mathematical equation as an infix string in parameters, and models the equation using a Binary Tree. The operator should be the root and the operands should be left and right child. Evaluate the result of the mathematical expression by traversing the tree and evaluating the expression at each node recursively.

**Q8.** Given a pre-order traversal (integer array) of a binary tree, construct the corresponding binary tree. For example:



Pre-order traversal = 1,2,4,5,3,6,7. You need to construct the above binary tree, given the array of integers which represent the pre-order traversal.