# Lesson:


# TREES One Shot

# Pre Requisites:

- Recursion

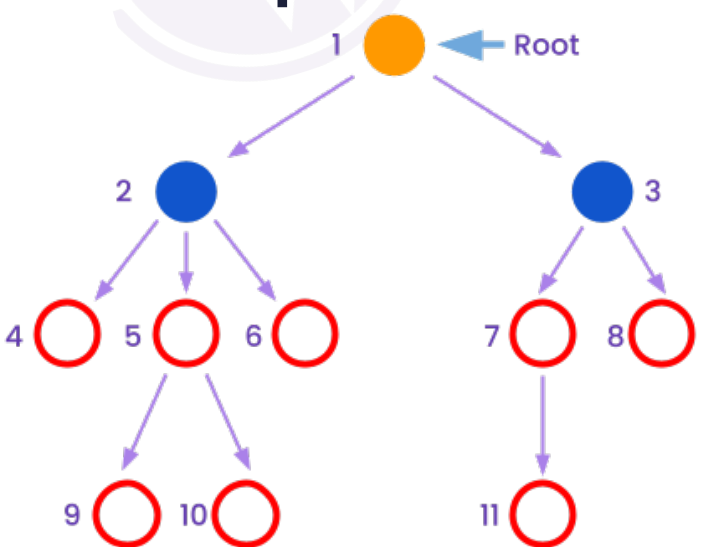
# What we will learn:

- What is a tree data structure
- Representation
- Terminology
- Important Properties of trees
- Types of trees
- Applications of tree data structure
- What is a Binary Tree?
- Implementation
- Traversals
- Problems
- Types of binary trees

# Topic: What is a Tree Data Structure?

- A tree is one of the data structures that represents hierarchical data.
- A tree data structure is a non-linear data structure because it does not store data in a sequential manner.
- A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.
- It is a hierarchical structure as elements in a Tree are arranged in multiple levels.
- **Real life example:** A family tree - Shows hierarchy in a family.

# TOPIC: Representation




- The topmost node is known as a root node.
- Each node contains some data and the link or reference to other nodes that can be called children of that node.
- Each node contains some data, and data can be of any type.
- The tree is represented by its root node, as all other nodes can be traversed if we know the root node.

# TOPIC: Terminology of a tree data structure

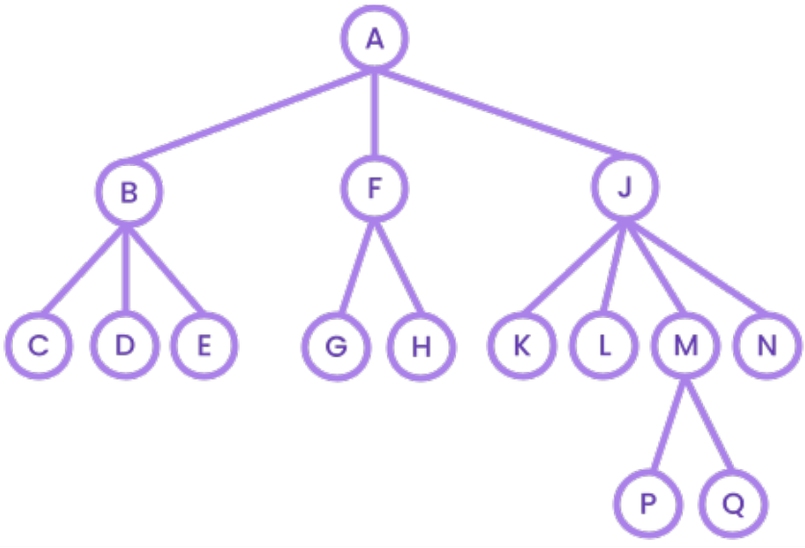
- Root: The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent. In the above structure, node numbered 1 is the root node of the tree.
- Parent-Child Relationship: If a node is directly linked to some other node, it would be called a parent-child relationship.
- Child node: If the node is a descendant of any node, then the node is known as a child node.
- Parent Node: If the node contains any sub-node, then that node is said to be the parent of that sub-node.
- Sibling Nodes: The nodes that have the same parent are known as siblings.
- Leaf Node: The node of the tree, which doesn't have any child node, is called a leaf node. There can be any number of leaf nodes present in a general tree. Leaf nodes that are also called external nodes.
- Internal nodes: A node that has at least one child node is known as an internal node.
- Ancestor node:- An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors. In the tree shown in the above image, nodes 1, 2, and 5 are the ancestors of node 10.
- Descendant: A descendant node of a node is any node in the path from that node to the leaf node (including the leaf node). In the above figure, 9 and 10 are the descendants of node 5.
- Level - Level of a node represents the generation of a node. Root node is at level 0, then its next child node is at level 1, its grandchild is at level 2 and so on.
- Path: A path in a tree is a sequence of distinct nodes in which successive nodes are connected by edges in the tree.
- Number of edges: If there are n nodes, then there would be n-1 edges. Each arrow in the structure represents the link or path. Each node, except the root node, will have atleast one incoming link known as an edge. There would be one link for the parent-child relationship.
- Height: The height of a tree (also known as depth) is the maximum distance between the root node of the tree and the furthest leaf node.
- Size: Size of a tree is the total number of nodes in the tree.

# TOPIC: Important properties of trees

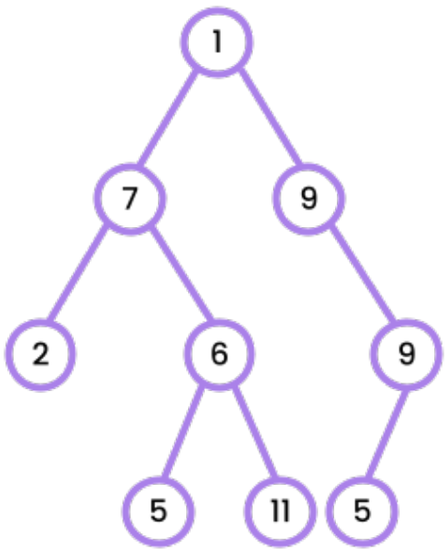
- Traversing in a tree is done by depth first search and breadth first search algorithm.
- Tree has no loop and no circuit
- Tree has no self-loop

# TOPIC: Types of Trees

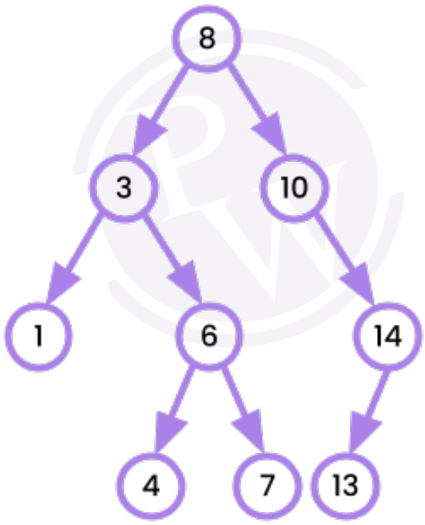
- **Generic tree:** The generic tree is one of the types of tree data structure. In the generic tree, a node can have either 0 or maximum n-1 number of child nodes where the first node is the root. There is no restriction imposed on the degree of the node (the number of nodes that a node can have as child nodes).



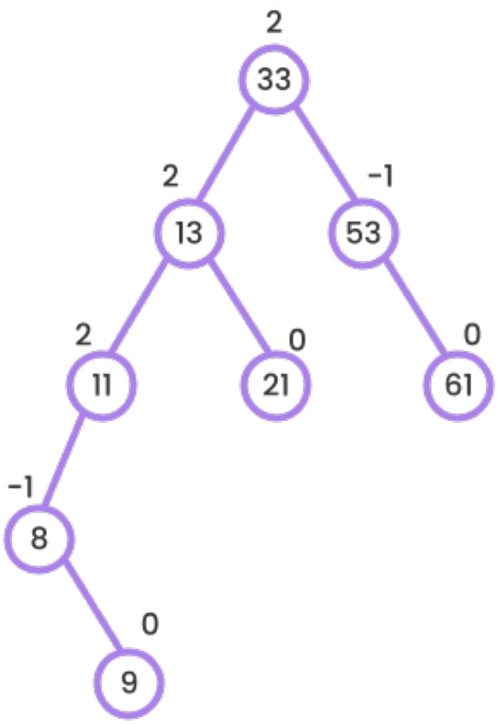
- **Binary tree:** Here, binary name itself suggests two numbers, i.e., 0 and 1. In a binary tree, each node in a tree can have utmost two child nodes. Here, utmost means whether the node has 0 child nodes, 1 child node or 2 child nodes.




- **Binary search Tree:** Binary search tree is a binary tree in which every node in the left subtree must contain a value less than the value of the root node, and the value of each node in the right subtree must be bigger than the value of the root node. This property is true for every node in the binary search tree and its left and right subtree.




- **AVL Trees:** It is a self-balancing binary search tree. Here, self-balancing means balancing the heights of the left subtree and right subtree.



Each node is associated with a balance factor which is calculated by subtracting the height of its right subtree from that of its left sub-tree.

For example, in the above AVL tree, the balance factor of the node with value 53 is because height of left subtree of that node = 0 and height of right subtree of that node = 1. Hence, balance factor = 0 - 1 = -1. Similarly, for node with value 13, height of its left subtree = 3, height of its right subtree = 1, hence its balance factor = 3 - 1 = 2.

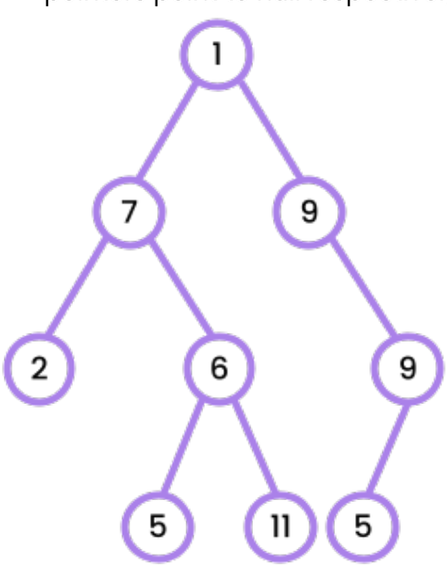
Tree is said to be balanced if the Balance Factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.

# TOPIC: Applications of Tree Data Structure

- Trees are used to model hierarchical structures, such as the file system in a computer or the organizational chart in a company.
- Trees can be used to sort data efficiently. For example, in a self-balancing binary search tree, the data is automatically sorted as it is inserted into the tree, making it easy to find the minimum, maximum, and other values in the tree.
- Trees are dynamic data structures, which means that they can grow and shrink as needed. This makes them well-suited for applications where the data changes frequently, such as in real-time systems.
- Trees provide efficient algorithms for inserting and deleting data, which is important in many applications where data needs to be added or removed frequently.
- Trees are relatively easy to implement, especially when compared to other data structures like graphs. This makes them a popular choice for many programming projects.

# TOPIC: What is a Binary Tree?

- A binary tree is a type of tree where the only restriction is that each node can have at most 2 children.
- A node can either have 0(leaf node), 1 or 2 child nodes.
- Instead of keeping an arraylist to store pointers to the children for each node, every node in a binary tree has 2 pointers, i.e. left and right.
- These two pointers denote the 2 children: left and right child of the node.
- In case, a node has no children, both the pointers point to null and in case a node has only 1 child, one of the pointers point to null respectively.




# TOPIC: Implementation

Creating node class

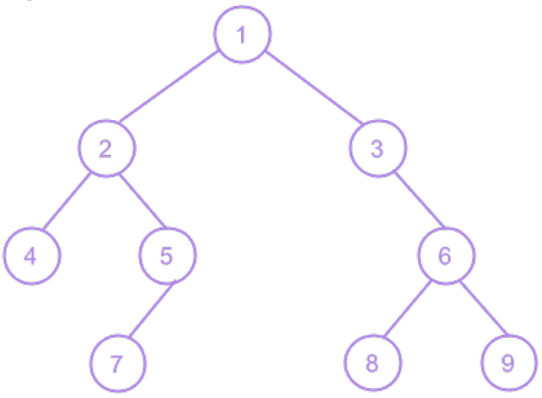
```
Class Node{
   int data;
   Node left;
   Node right;
}
```

# Important Questions

**1) Print elements on nth level**

Given a binary tree and an integer K, the task is to print all the integers at the Kth level in the tree from left to right.

**Input:**




K = 3

**Output:** 4 5 6

**Code:** https://pastebin.com/mmg3cWym

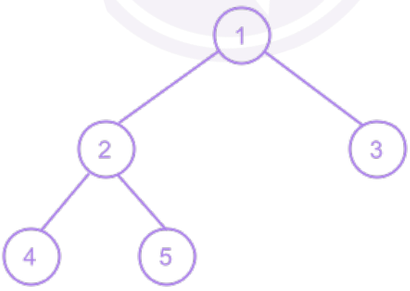
**Time complexity:** O(n) because it needs to visit each node once in order to print the elements at the desired level. The number of nodes in the binary tree determines the runtime of the algorithm.

**Space complexity:** O(h), where h is the height of the binary tree. This is because the space required for the recursive calls depends on the maximum depth of the recursion, which is equivalent to the height of the tree. In the worst case, when the tree is skewed, the space complexity can be O(n), but for a balanced tree, it is O(log(n)), where n is the number of nodes in the tree.

**2) Find sum of tree nodes**

Given a binary tree, the task is to find the sum of all tree nodes.

**Input:**




**Output:**

```
Sum of all nodes in the binary tree: 15
```

**Code:** https://pastebin.com/9k9hbVsR

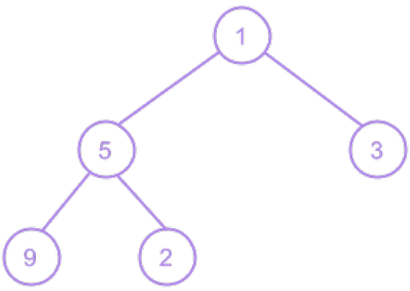
**Time complexity:** O(n), where n is the number of nodes in the tree. This is because we need to visit each node once to calculate the sum.

**Space complexity:** O(h), where h is the height of the tree. In the worst case, if the tree is skewed, the height of the tree can be equal to the number of nodes (h = n), resulting in O(n) space complexity.

**3) Find node with max value**

Given a binary tree with unique values. The task to find the node with the maximum value.

**Input:**



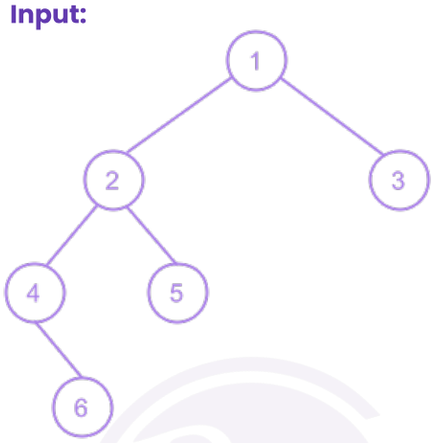
**Output:**

```
Node with the maximum value: 9
```

**Code:** https://pastebin.com/8m9tdkuV

**Time complexity:** O(n), where n is the number of nodes in the tree. This is because we need to visit each node once to compare its value with the maximum value.
**Space complexity:** O(h), where h is the height of the tree. In the worst case, if the tree is skewed and resembles a linked list, the height of the tree can be equal to the number of nodes (h = n), resulting in O(n) space complexity.

**4) Find height of Binary Tree**
Given a Binary tree, the task is to find the height of Binary tree.
**Input:**



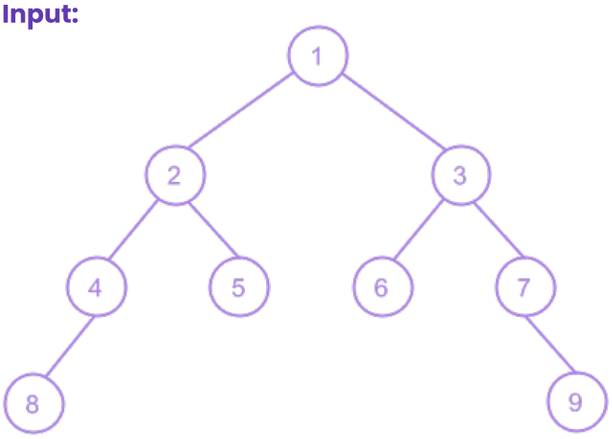

**Output:**

```
Height of the binary tree: 4
```

**Code:** https://pastebin.com/mFKnTC1P

**Time complexity: O(n),** where n is the number of nodes in the tree. This is because we visit each node exactly once during the recursive traversal.
**Space complexity: O(h),** where h is the height of the tree. In the worst case, if the tree is skewed and resembles a linked list, the height of the tree can be equal to the number of nodes (h = n), resulting in O(n) space complexity due to the recursive call stack.

**5) Find size of Binary Tree**
Given a Binary tree, find the size of this binary tree. The size of a binary tree is the total number of nodes present in the tree. It represents the count of all the nodes in the tree, including the root and all its descendants.
**Input:**




**Output:**

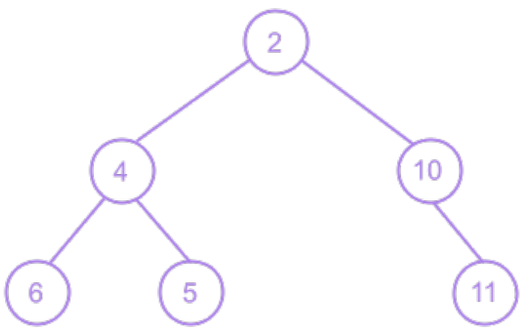
```
Size of the binary tree: 9
```

**Code:** https://pastebin.com/yrjkAPDv

**Time complexity:** O(n), where n is the number of nodes in the tree. This is because we visit each node exactly once during the recursive traversal to calculate the size.
**Space complexity:** O(h), where h is the height of the tree. In the worst case, if the tree is skewed and resembles a linked list, the height of the tree can be equal to the number of nodes (h = n), resulting in O(n) space complexity due to the recursive call stack.

# TOPIC: Traversals

Let's consider this binary tree to understand various ways of traversing through it:




- **DFS**

Depth First Search (DFS) traversals of binary trees is generally done in 3 ways.

- **Preorder**

The order for Preorder traversal is Node -> Left Node -> Right Node.
For the given tree, the preorder traversal would be: 2 4 6 5 10 11

**Code:**
https://pastebin.com/yLBZAJm5

**Output:**

```
"C:\Program Files\Java\jdk-14.0.1\
2 4 6 5 10 11
Process finished with exit code 0
```

**Time Complexity:** O(n) where n is the size of the array because we are visiting all the nodes of the binary tree exactly once.
**Space Complexity:** O(h) where h denotes the height of the tree. This is because the addresses are removed from the recursion stack when returning. This space is re-used when making a new call from a level closer to the root. So, the maximum number of memory addresses on the recursion stack at a given point of time can be equal to the height of the tree.

- Inorder

The order for Inorder traversal is Left Node -> Node -> Right Node.
For the given tree, the inorder traversal would be: 6 4 5 2 10 11

**Code:**
https://pastebin.com/0V0gfbYj

**Output:**

```
"C:\Program Files\Java\jdk-14.0.1\
6 4 5 2 10 11
Process finished with exit code 0
```

**Time Complexity:** O(n) where n is the size of the tree
**Space Complexity:** O(h) where h denotes the height of the binary tree.

- **Postorder**

The order for Postorder traversal is Left Node -> Right Node -> Node.
For the given tree, the postorder traversal would be: 6 5 4 11 10 2

**Code:**
https://pastebin.com/HGuvFxHD

**Output:**

```
"C:\Program Files\Java\jdk-14.0.1
6 5 4 11 10 2
Process finished with exit code 0
```

**Time Complexity:** O(n) where n is the size of the tree
**Space Complexity:** O(h) where h denotes the height of the binary tree.

- **BFS**

Breadth First Search (BFS) traversals in binary trees is also called Level order traversal as we traverse the nodes of the tree level wise . We use a Queue data structure to traverse in a BFS fashion.

**Code:**
https://pastebin.com/3TFcTytz

**Output:**

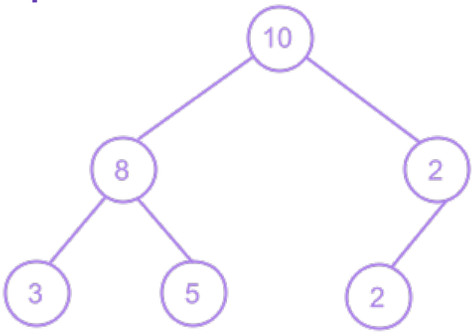
```
"C:\Program Files\Java\jdk-14.0.1\
2
4 10
6 5 11

Process finished with exit code 0
```

**Time Complexity =** O(n) where n is the size of the tree
**Space Complexity =** O(n) where n is the size of the tree because the maximum number of nodes in the queue at any given time in the worst case could be n-1.

# Topic: Problems

**Q1. Given the root of a binary tree, return all root-to-leaf paths in any order.** [Easy]

**Input:**



**Expected Output:**
10 8 3
10 8 5
10 2 2

**Explanation:**

- The approach for this problem is to use DFS until we hit a null node and print the path collected so far, renew through another path.
- We use a helper function here, which takes the root node, an arraylist to store the current path and an arraylist of arraylists that stores all the paths.
- In the helper function, we first check if the node is null(base case), which means we have encountered a dead end so we return.
- Add the data of current node to the arraylist.
- Now check, if this was a leaf node and if so, it has marked a completed path and here, we add this arraylist to the arraylist that is storing all our paths and we return.
- Else, recursively call this function, for the left and the right child of the root node.
- Also, created another function here where we declare the arraylists for cleaner code.
- In the main function, we call this function.

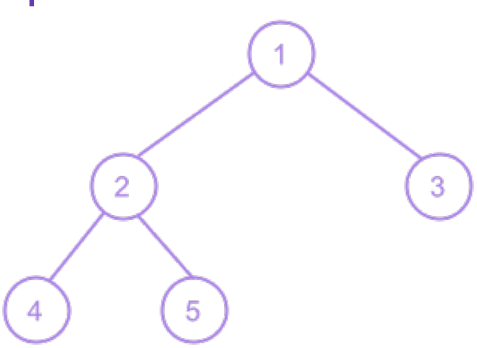
**Code:**
https://pastebin.com/qpmvL8wa

**Output:**

```
"C:\Program Files\Java\jdk-14.0.1\
10 8 3
10 8 5
10 2 2

Process finished with exit code 0
```

**Q2. Given a binary tree, print its diameter.**
**The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.The length of a path between two nodes is represented by the number of edges between them.** [Medium]

**Input:**




**Expected output:**
3

**Explanation:** 3 is the length of the path [4,2,1,3] or [5,2,1,3] which is the longest path between two end nodes 4 and 3 or 5 and 3.

**Approach:**

- Diameter of a tree can be of a path across the root or in a subtree.
- Consider the case when diameter is of a path through the root, in this case, we need the heights of left subtree, right subtree, and add one more for the root node and that would be the diameter.
- In the case when diameter lies in a subtree, we recursively call the diameter function for the left and the right subtree.
- Compare all three diameters and return the longest.

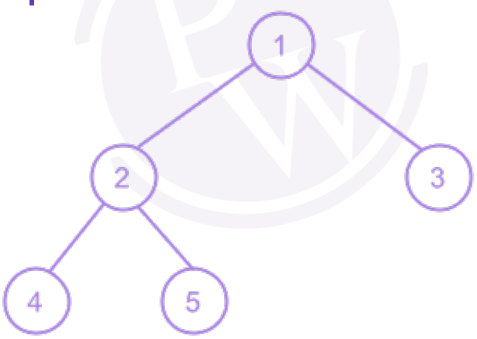
**Code:**
https://pastebin.com/47D2SSxT

**Output:**

```
"C:\Program Files\Java\jdk-14.0.1
3

Process finished with exit code 0
```

**Q3. Given a binary tree, print the lowest common ancestor of 2 given nodes. The lowest common ancestor is the lowest node in the tree that has both node1 and node2 as descendants.** [Medium]

**Input:**




n1 = 5
n2 = 3

**Expected Output:**
1
**Explanation:**
If we trace back on the paths of 5 and 3, we see a common ancestor 1. 1 is common because if we go from 5 to 1 we can go from 1 to 3.

**Approach:**

- Find a path from the root to n1 and store it in a vector or array.
- Find a path from the root to n2 and store it in another vector or array.
- To find the path from a node to root, create an empty arraylist.
- Add the root node data to the arraylist.
- Now check, if this node is equal to the node we are searching for or if there exists a path from the left child or

the right child.
If not, remove the root node from the arraylist and return.
Traverse both paths till the values in arrays are the same. Return the common element just before the mismatch.

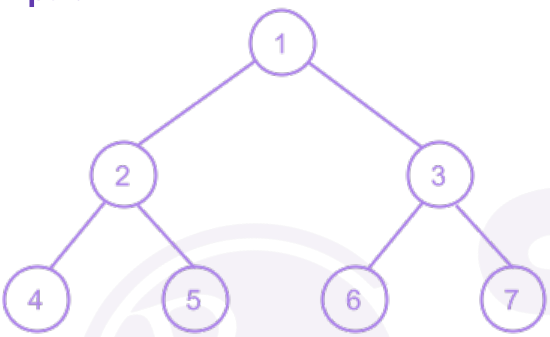
**Code:**
https://pastebin.com/mBnUYaqR

**Output:**

```
"C:\Program Files\Java\jdk-14.0.1\
1

Process finished with exit code 0
```

**Q4. Write a program to find the diagonal traversal of the binary tree.**

**Input:**




**Output:**

```
Diagonal Traversal: 1 3 7 2 5 6 4
```

**Solution Code:** https://pastebin.com/KUA8hKhw

**Explanation:**

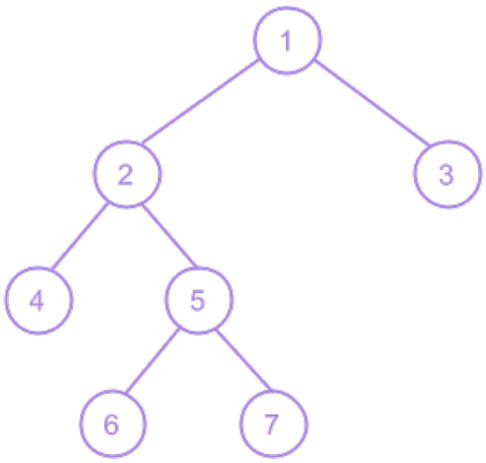
- Start by adding the root node of the binary tree to a queue.
- While the queue is not empty, remove the first node from the queue.
- Print the value of the current node.
- If the current node has a left child, add it to the end of the queue.
- Move to the right child of the current node.
- Repeat steps 3 to 5 until the queue is empty.
- Repeat steps 2 to 6 for each diagonal of the binary tree.

**Time complexity:** O(N), where N is the number of nodes in the binary tree. This is because we need to visit each node exactly once during the traversal.

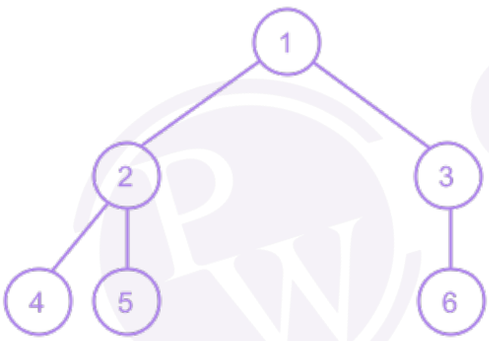
**Space complexity:** O(N), where N is the number of nodes in the binary tree. This is because in the worst case, the queue used for the traversal can contain all the nodes of the binary tree, which is N.

# Topic: Types of Binary Trees

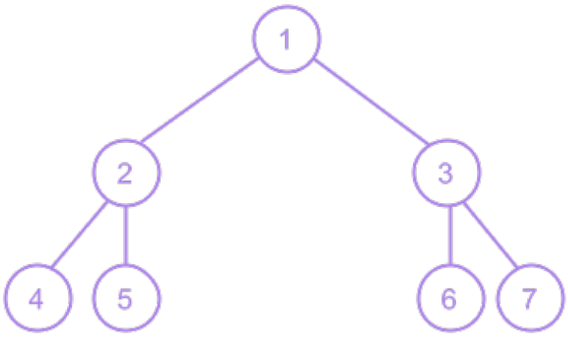
- **Full Binary Tree:** It is a special kind of a binary tree that has either zero children or two children. It means that all the nodes in that binary tree should either have two child nodes or the node is itself the leaf node or the external node.



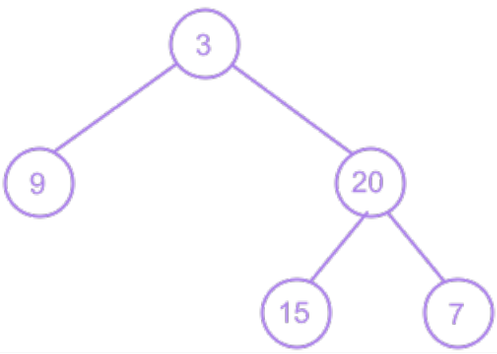

- **Complete Binary Tree:** A complete binary tree is another specific type of binary tree where all the tree levels are filled entirely with nodes, except the lowest level of the tree. Also, in the last or the lowest level of this binary tree, every node should possibly reside on the left side.



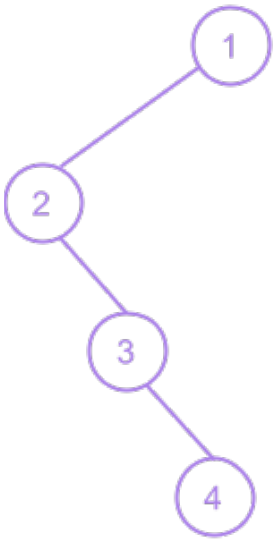

**Perfect Binary Tree:** A binary tree is said to be 'perfect' if all the internal nodes have strictly two children, and every external or leaf node is at the same level or same depth within a tree.




**Balanced Binary Tree:** A binary tree is said to be 'balanced' if the height of the left and the right subtrees of each node should vary by at most one.



- **Degenerate Binary Tree:** A binary tree is said to be a degenerate binary tree or pathological binary tree if every internal node has only a single child.




## Upcoming Class Teasers:

- Binary Search Tree