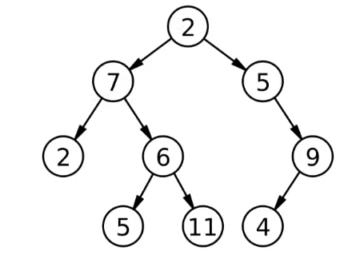
**Binary Tree Traversal Algorithms**

Binary tree is a data structure where each node has at most 2 children: *left child* and *right child*



Height: starting from lowest leaf node = 4

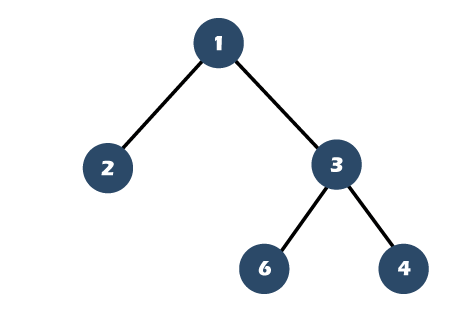
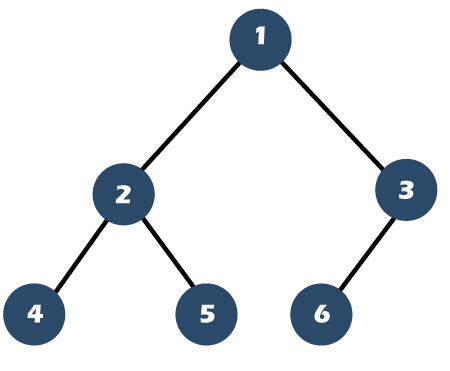
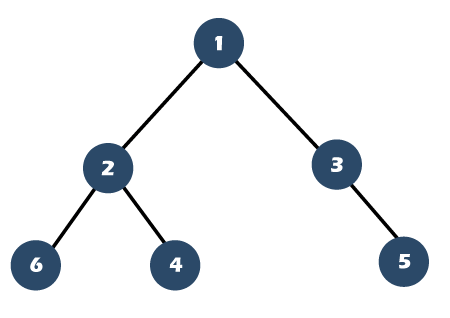
Depth: starting from root node = 4

Root Node

Leaves

Full binary tree: A full binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children. A node cannot have 1 child, it should be either 0 or 2.

Complete binary tree: A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled. Nodes should always be filled up from left side.



**🗵** **Full tree**

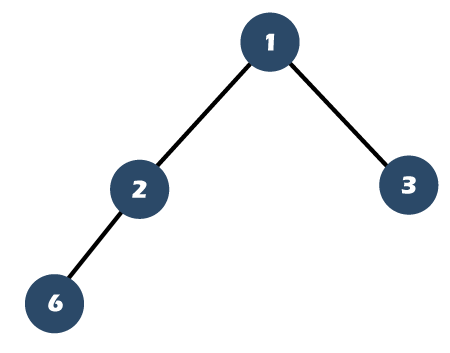
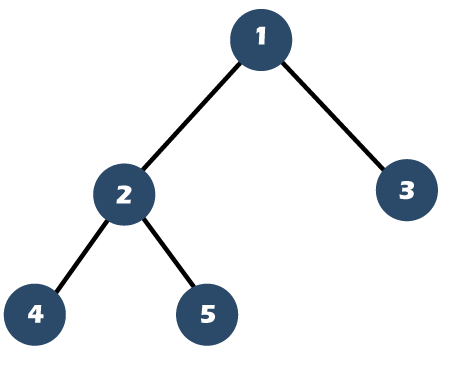
**☑ Complete Tree**

**☑** **Full tree**

**🗵 Complete Tree**

**🗵** **Full tree**

**🗵** **Complete Tree**



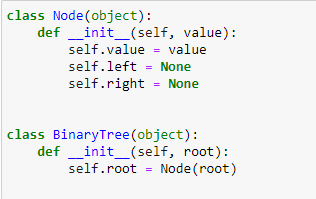
**☑** **Full tree**

**☑ Complete Tree**

**🗵** **Full tree**

**☑ Complete Tree**

**Python Binary Tree Data Structure:**



**1. Pre-order Traversal**

**Pre-order: 2-7-3-6-5-9-8-1-4**

* Start at root node
* Parse from left to right, go back to parent node after leaf is encountered
* Add or display nodes as you visit them for the first time
* Do not add nodes that you ‘revisit’

**2. In-order Traversal**

**In-order: 3-7-5-6-9-2-8-4-1**

* Start at root node
* Parse from left to right, go back to parent node after leaf is encountered
* First find all left sub children, add / display the left nodes
* Add parent only when all left sub children are done
* Move on to right sub children of current node

**3. Post-order Traversal**

**Post-order: 3-5-9-6-7-4-1-8-2**

* Start at root node, parse from left to right, go back to parent node after leaf is encountered
* First find all leaves of left subtree, add / display the leaf nodes
* A parent node can only be added / displayed when all sub-nodes (left + right) are done
* Move on to right sub children of current node

**Non-Binary Tree Traversal Algorithms**

**1. Breadth First Search (BFS)**

* Maintain two lists: queue and visited
* Root node is added to the queue at start
* Pop node from the queue (FIFO basis)
* Push node to visited
* Push node’s children to queue (Left 🡪 Right)

**BFS: [A, B, C, D, E, F, N, G, H, I, J, K]**

**2. Depth First Search (DFS)**

DFS is diving deep into the graph and backtracking when it hits bottom. DFS is like solving a MAZE. We'll continue to walk through the path of the maze until we reach a dead end.

21

14

1

11

10

14

90

2

20

19

15

13

4

3

18

12

5

8

17

7

6

16

**DFS: [A, B, E, F, I, J, C, N, D, G, K, H]**

Note: One cannot help but wonder how the exploration and backtracking looks too similar the pre-order traversal. However, pre-order is a traversal, it visits every node in a binary tree, which is easiest graph structure.

Depth First Search is a search algorithm however, goes around an arbitrary graph (graph can be cyclic unlike binary tree, the traversal may not even start from identifiable root node) looking for a certain node this alone is a large enough difference to call them difference names.

1

3

2

4

5

**DFS: [S, A, R, C, L]**

* Take current node
* Append it to visited list
* Iterate thought nodes children
* If child not in visited