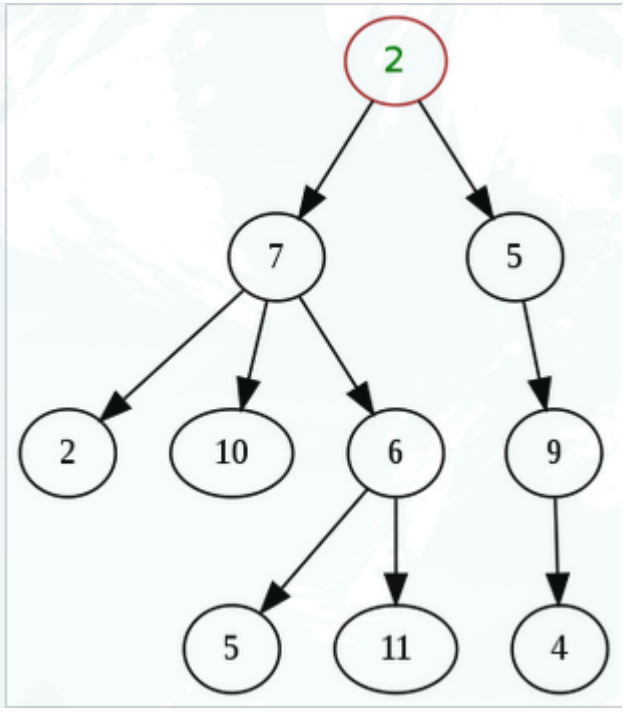
Tut 3

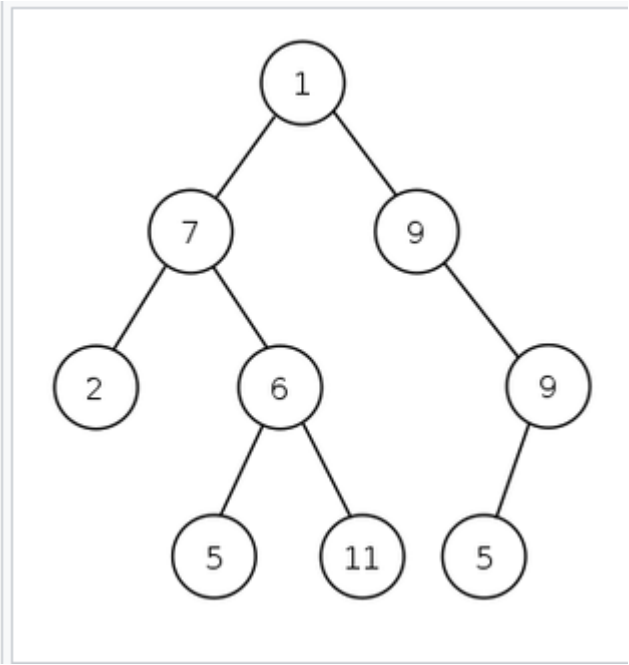
Trees

- It is an ADT used to represent hierarchical tree structures
- Each node can have many childeren (depending on the type of the tree)
- Nodes with no childeren are called leaf nodes
- Nodes with atleast one child are called branch nodes
- Each node has a single parent (except the root node, which has none)
- Each node is the root node for its own subtree
- Height /Depth of a node is the length of the **longest** path from the node to a leave
- Height/Depth of the root is the height of the tree
- Eg



Binary trees

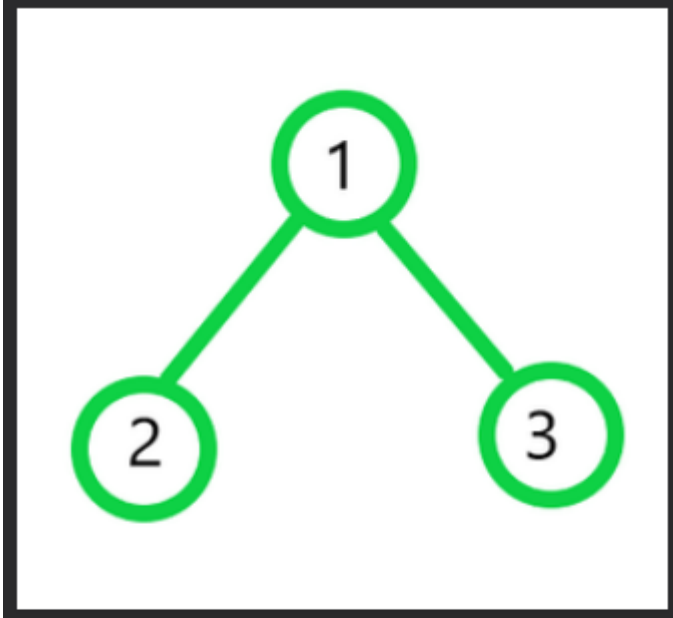
- Trees in which every node has a maximum of 2 children (left child and right child)
- Eg



Traversal

- Three types
  - Preorder
  - Inorder
  - Postorder
- Preorder
  1. Yourself
  2. Left subtree
  3. Right subtree
- Inorder
  1. Left subtree
  2. Yourself
  3. Right subtree
- Postorder
  1. Left subtree
  2. Right subtree
  3. Yourself

Input:



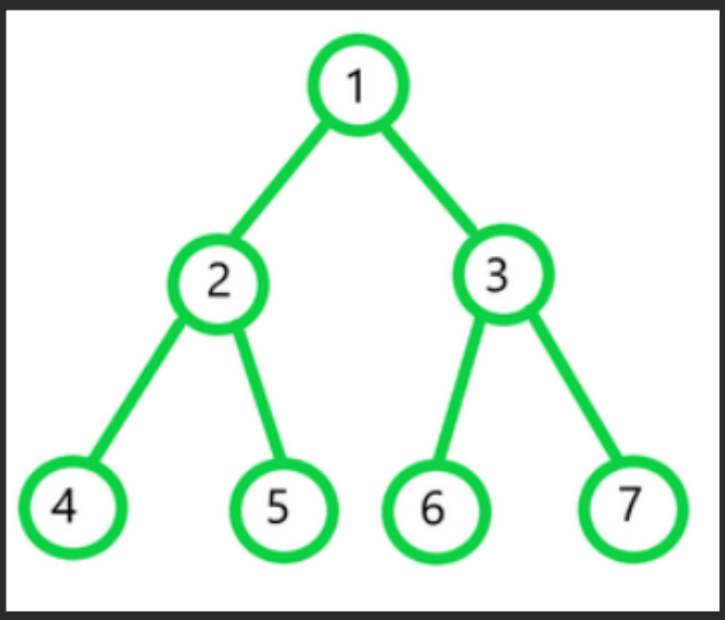
Output:

Preorder Traversal: 1 2 3

Inorder Traversal: 2 1 3

Postorder Traversal: 2 3 1

Input:



Output:

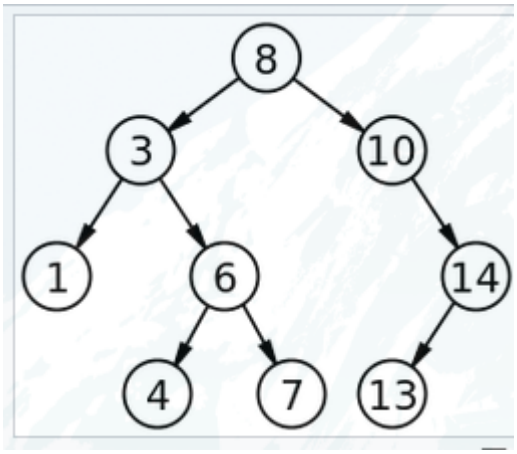
Preorder traversal: 1 2 4 5 3 6 7

Inorder traversal: 4 2 5 1 6 3 7

Post-order traversal: 4 5 2 6 7 3 1

Binary search trees (BST)

- Binary tree with the additional following conditions
  - value of left child < value of node
  - value of node $\leq$ value of right child
  - This is true for all nodes



- Operations in Binary seach tree:
  - Search(x) -> Search for a value x in the tree and return the node
  - Insert(node) -> Insert a node in the BST
  - Delete(node) -> deletes a node in the BST

Search(x)

```
if x == val:
    return node
else if x < val:
    search in left subtree
else:
    search in right subtree
```

Insert(node)

```
if curr_node is a leaf:
    if node_val < curr_node_val:
        insert node as left child
    else:
        insert node as right child
else
    if node_val < curr_node_val:
        go to left child
    else
        go to right child
```

Delete(node)

```
node = search(node_val)
if node is a leaf:
    delete node
else if node has only 1 child:
    Replace node with the child
else:
    Find the smallest element in the right subtree (also called inorder
successor)
    Replace the node with the inorder successor
```