UNIVERSITY OF
LEICESTER

CO1107
Data Structure

UNIVERSITY OF
LEICESTER

Tree Data Structure Part 1

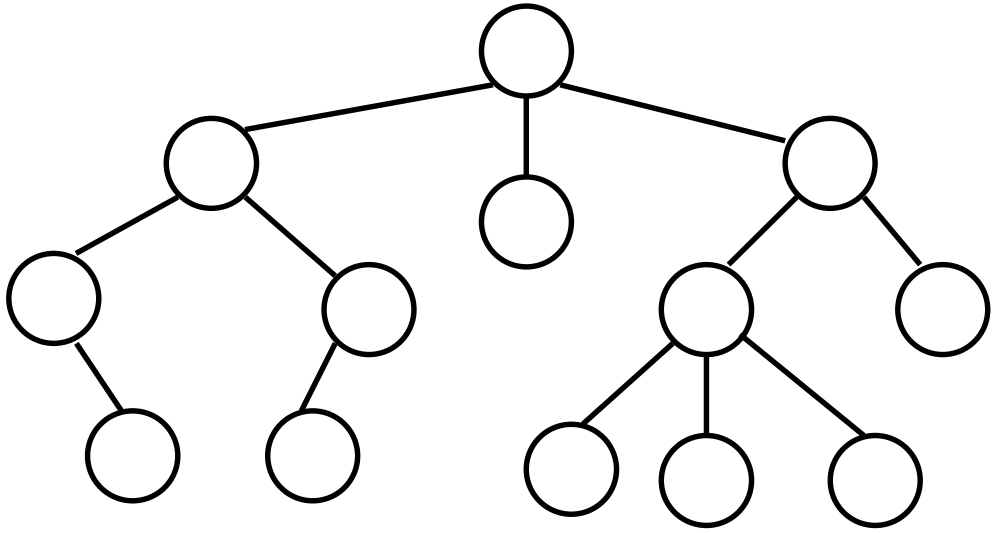
https://www.thespruce.com/twelve-species-of-ash-trees-3269661

Tree

- Very useful
- Used to model many things like:
  - Family trees
  - Structure chart of an organization
  - Structure of the chapters in a Book/Thesis
  - Object oriented class hierarchies

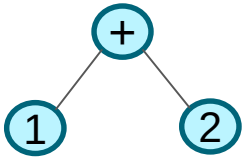
Trees

- Trees are graphs which are:
  - Simple,
  - Connected, and
  - Have no cycles.

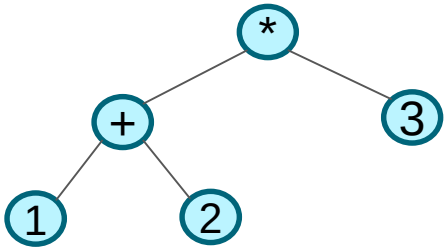


Arithmetic expressions

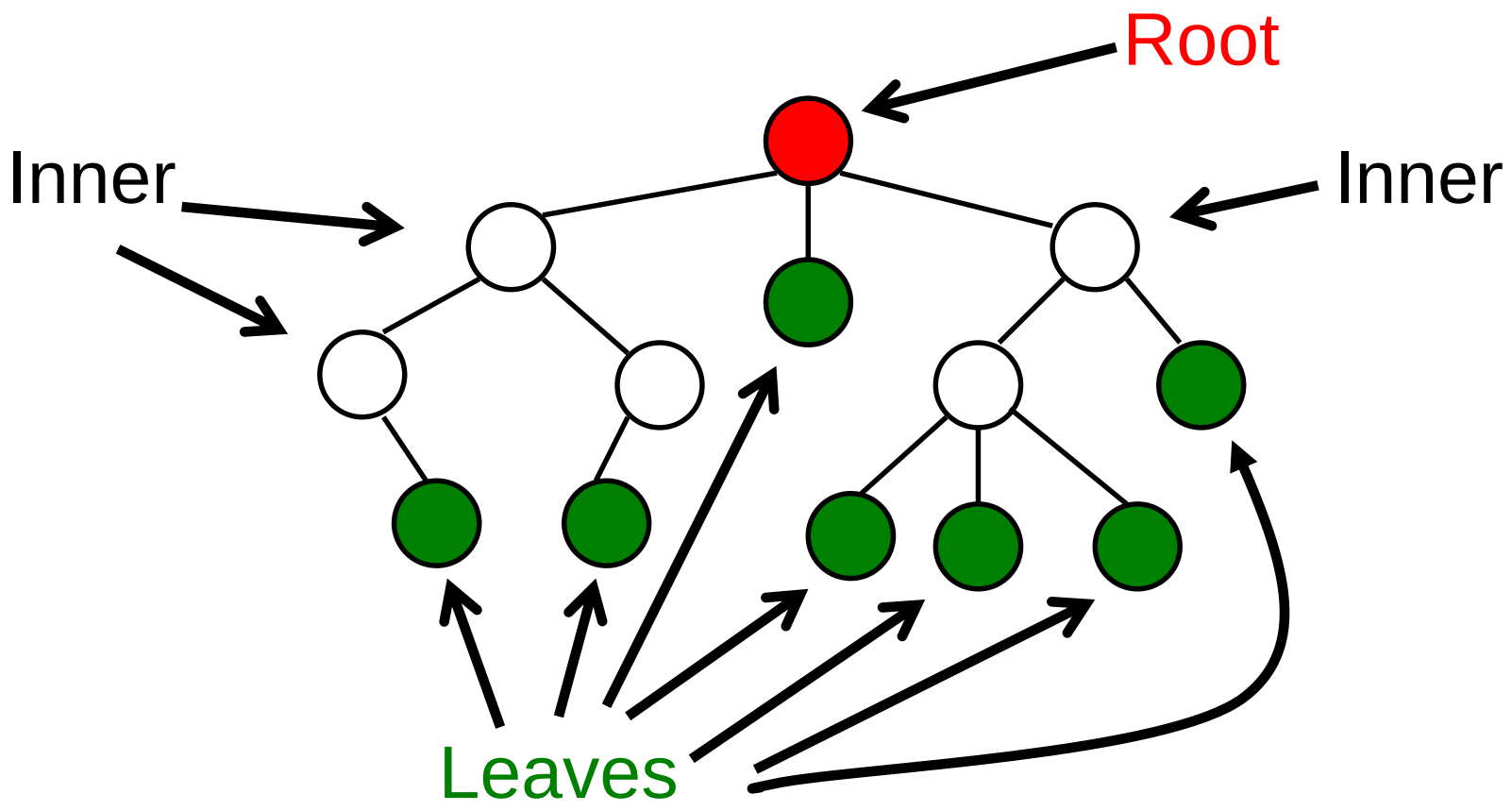
- 1+2



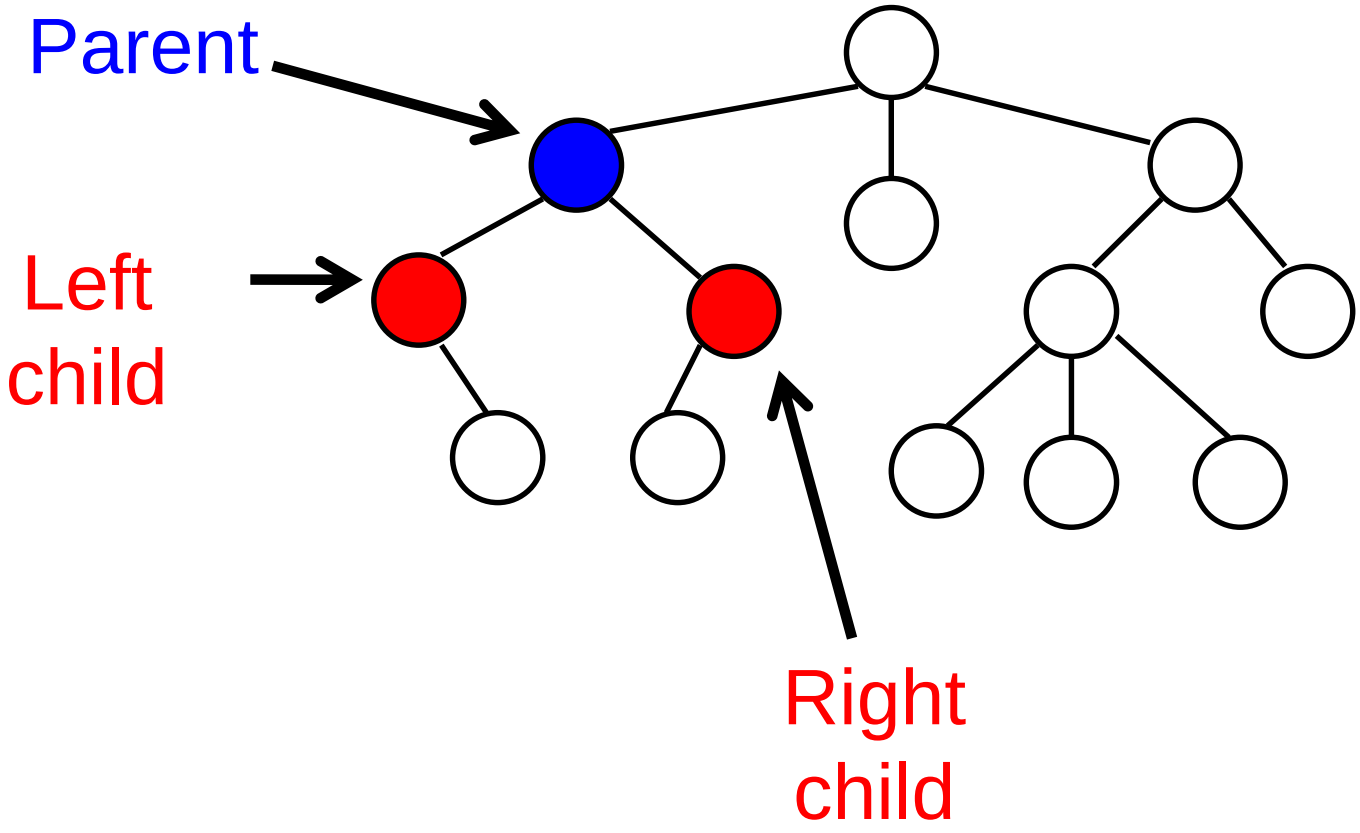
- (1+2)*3



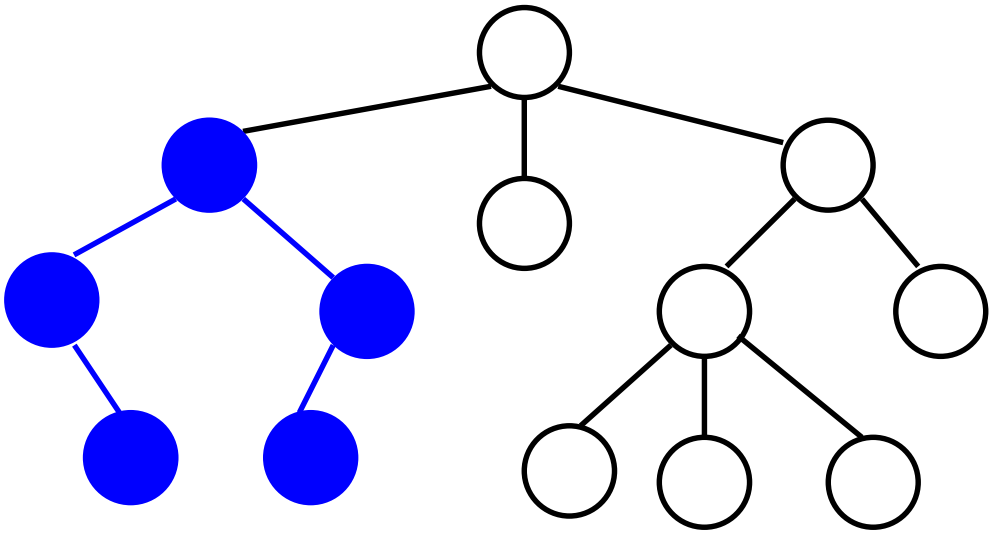
Tree Terminology



Tree Terminology

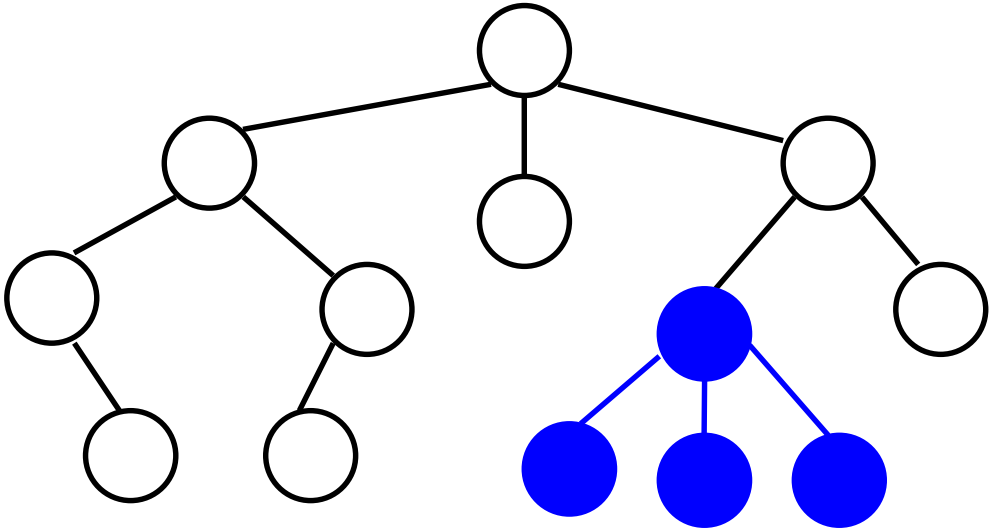


Tree Terminology



Subtree

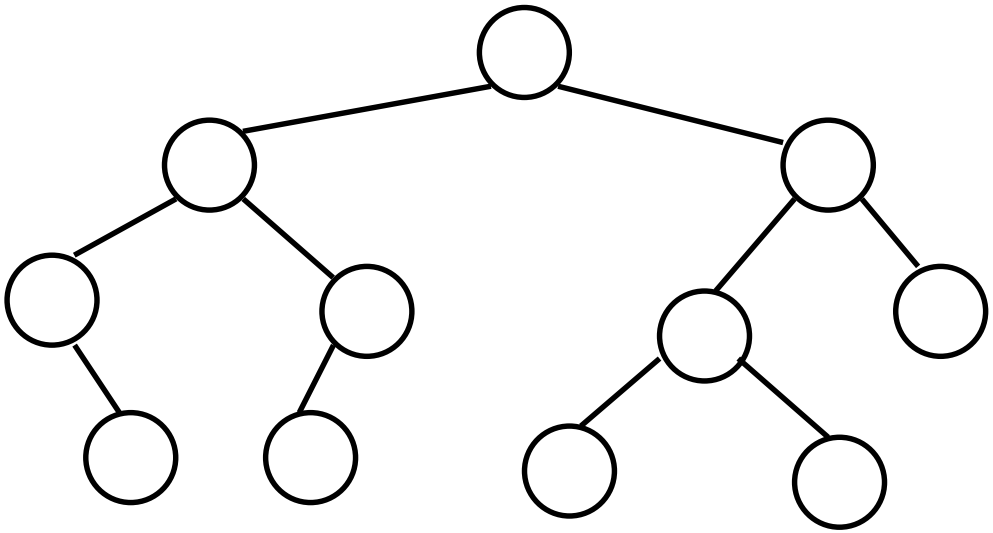
Tree Terminology



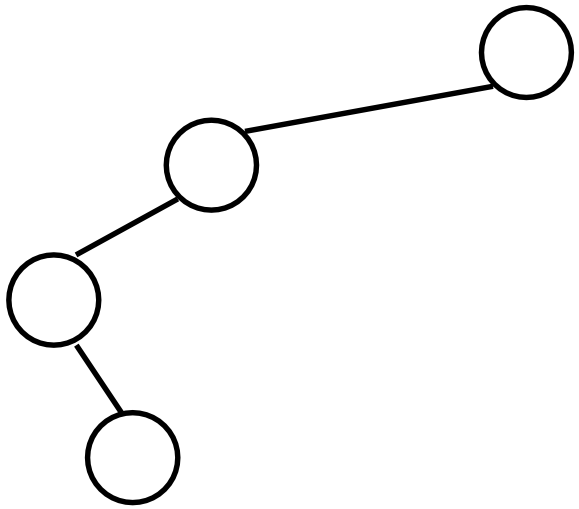
Subtree

What is Binary Tree?

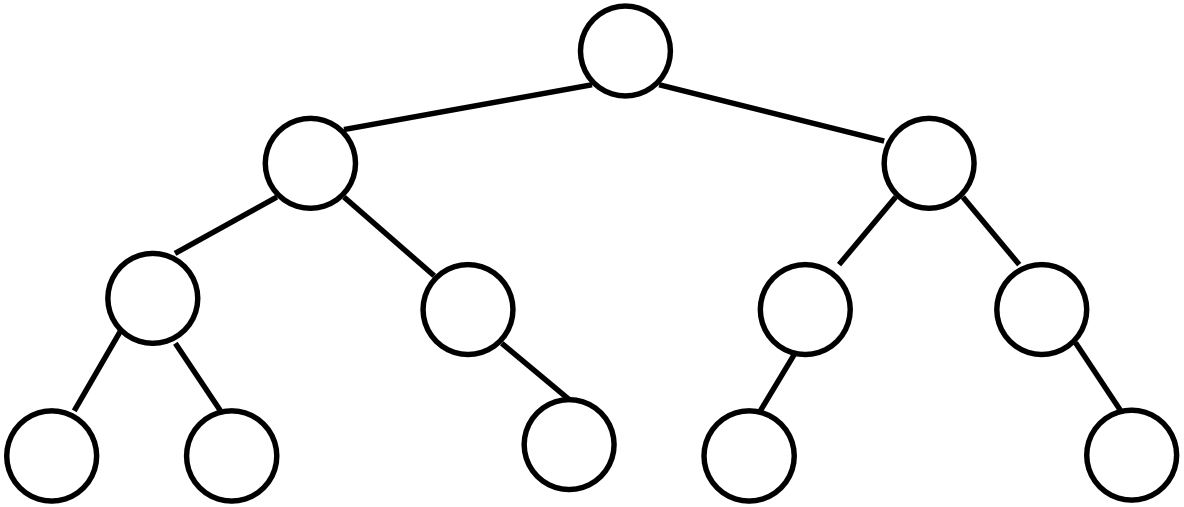
- Every node has at most two children.
- Every subtree is a Binary Tree
- Note: Empty tree is also a binary tree



Unbalanced Binary Tree



Balanced Binary Tree

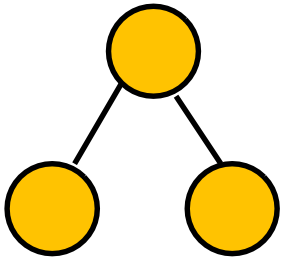


For every node

|height(left subtree) – height(right subtree)| ≤ 1

Perfect Binary Trees

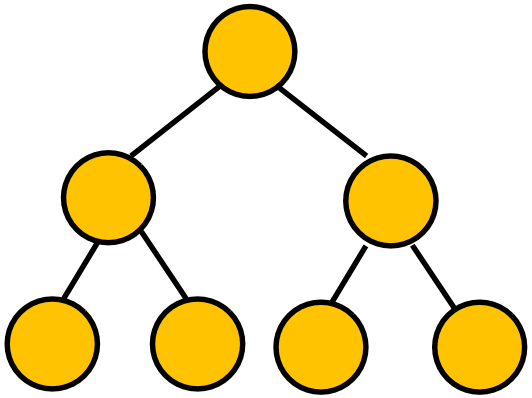
Size =1 Height = 0



Size =3 Height = 1

Each parent has two children

All leaves at same level



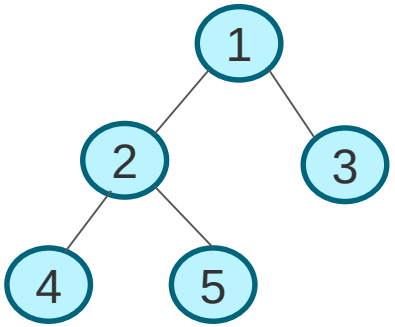
Size =7 Height = 2

Binary Tree Implementation in Python

- Live Demo

Binary Tree Implementation in Python

```
class TreeNode():
    def __init__(self,item):
        self.item=item
        self.left=None
        self.right=None


class BinaryTree():
    def __init__(self,root):
        self.root=TreeNode(root)

ex=TreeNode(1)
bt=BinaryTree(ex)
bt.root.left=TreeNode(2)
bt.root.right=TreeNode(3)
bt.root.left.left=TreeNode(4)
bt.root.left.right=TreeNode(5)
```



The Size of Binary Tree

- Live Demo

The Size of Binary Tree

```
def __len__(self):
    return self.len_aux(self.root)
def len_aux(self,current):
    if current is None:
        return 0
    else:
        return 1+self.len_aux(current.left)+self.len_aux(current.right)
```

```
print(bt.__len__())
```

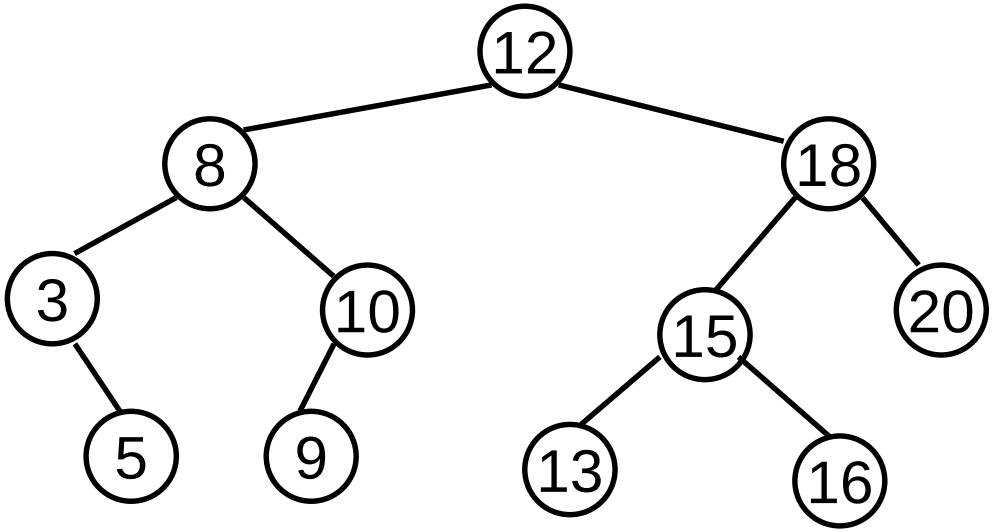
The output will be 5 for the given Tree.

Calculate the Height of the Binary Tree

```
def height(self):
     return self.height_aux(self.root)

def height_aux(self,current):
    if current is None:
        return -1
    else:
        return 1+max(self.height_aux(current.left),self.height_aux(current.right))
```

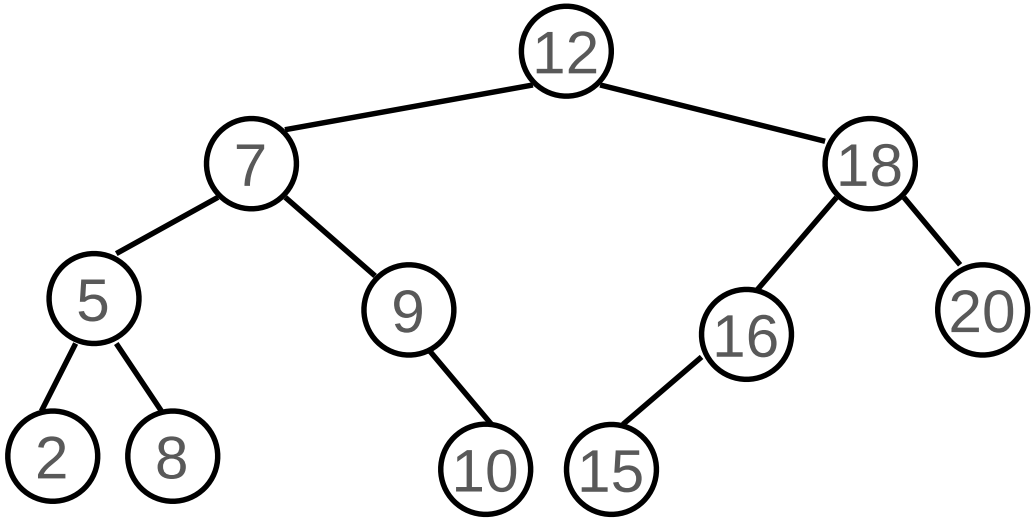
Binary Search Tree

- The values of each node in its left subtree is less than its value.
- The values of each node in its right subtree is greater than its value.



Note: Every subtree is a Binary Search Tree

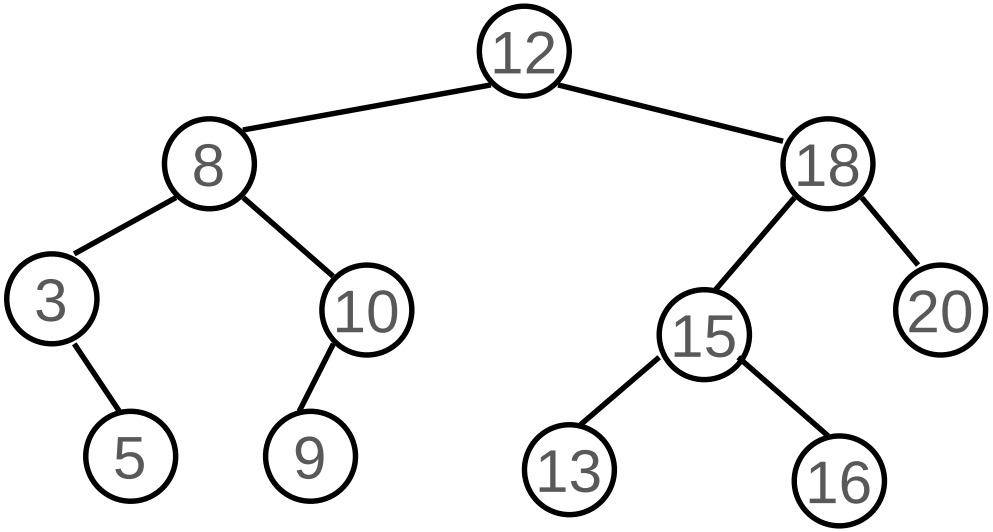
Is this a binary search tree?



A. Yes

B. No

Is this a binary search tree?



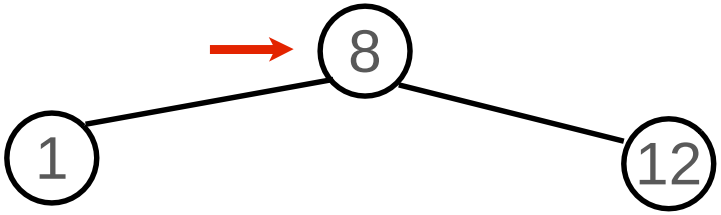
A. Yes

B. No

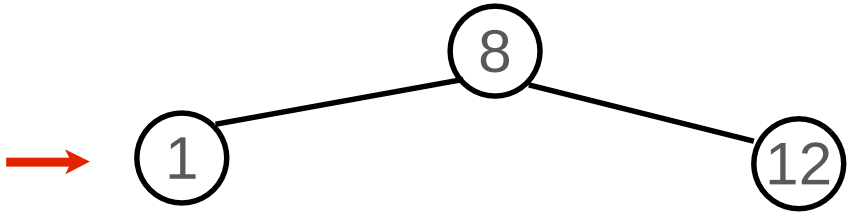
Binary Search Tree's Operations

- Searching a value
- Inserting a value

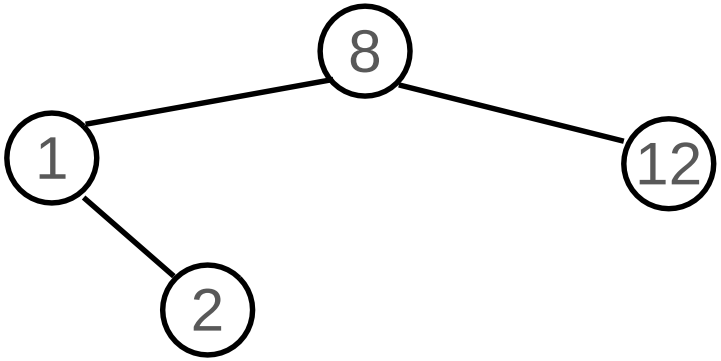
Insert 2



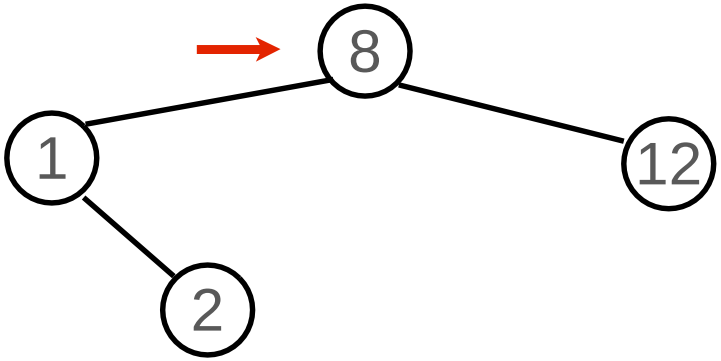
Insert 2



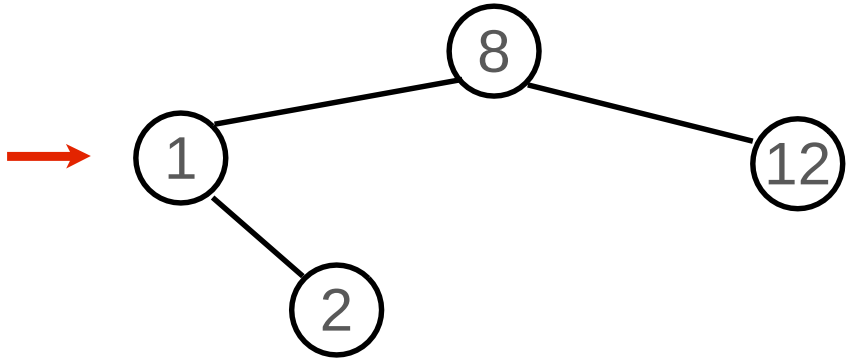
Insert 2



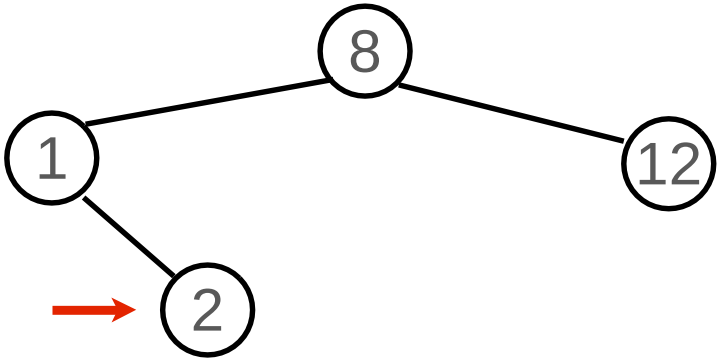
Insert 7



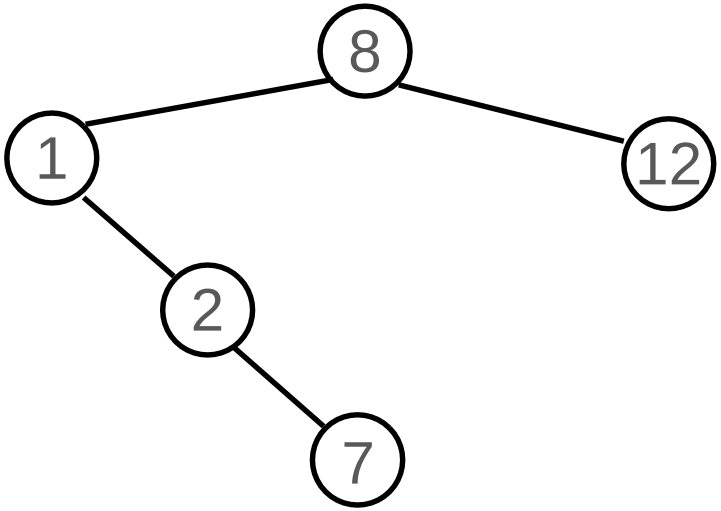
Insert 7



Insert 7



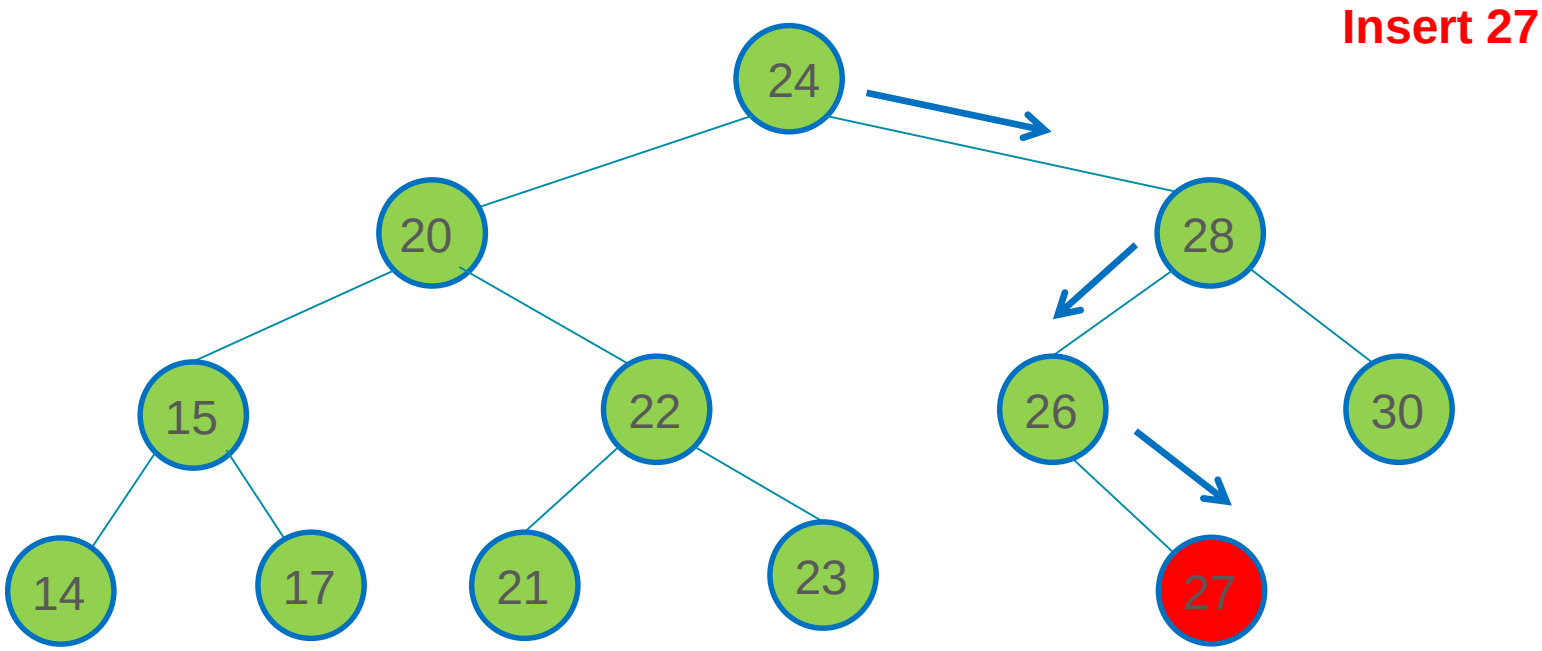
Insert 7



Our BST does not allow for duplicates, so we need to do something if we find the key in the tree…

Inserting a value in Binary Search Tree

1. Search for the value in Binary Search Tree
2. Insert when reached below leaf

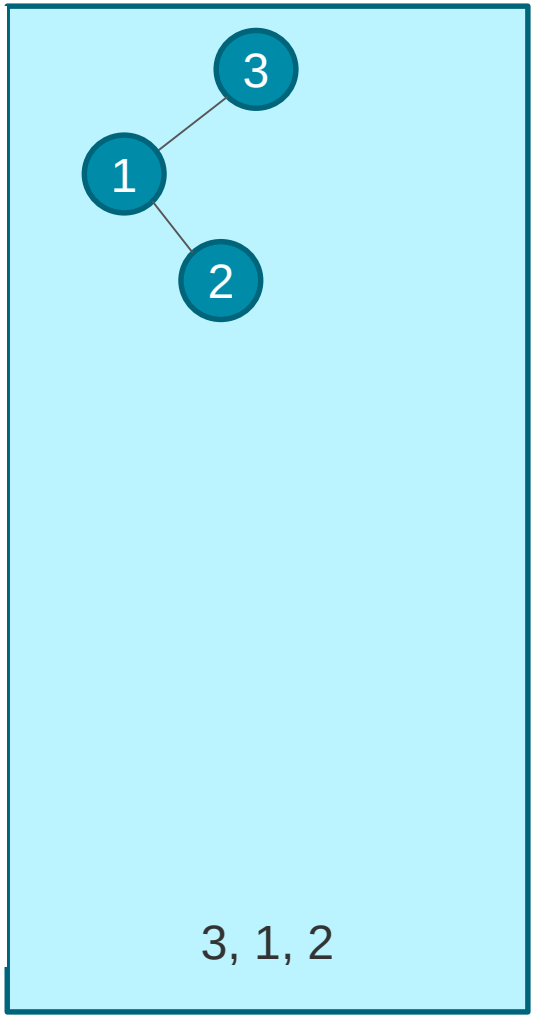


Insert into BST: Python Implementation

```
class TreeNode():
    def __init__(self,item):
        self.item=item
        self.left=None
        self.right=None


class binarySearchTree():
    def __init__(self):
        self.root=None

    #add insert function here ....
```

Insert into BST: Python Implementation

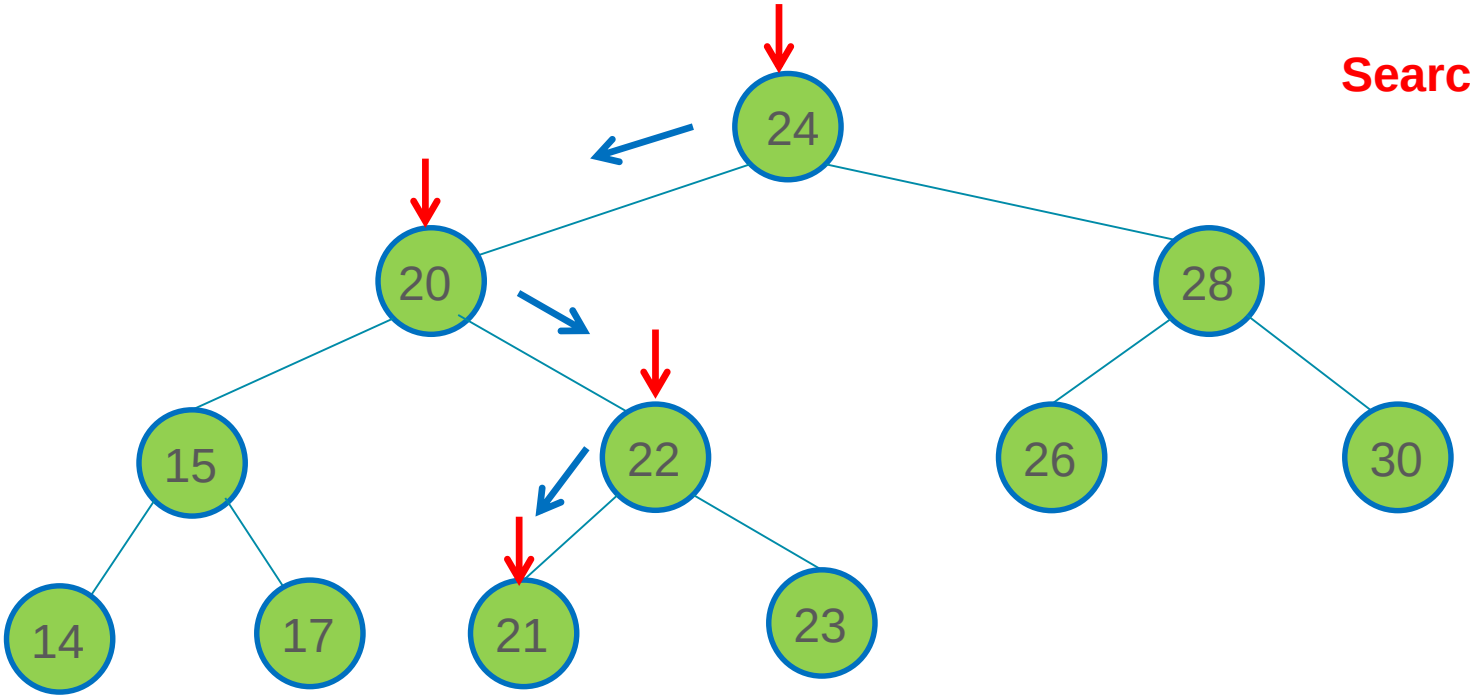
```
def insert(self,item):
    if self.root is None:
        self.root=TreeNode(item)
    else:
        self._insert(item,self.root)

def _insert(self,item,currentNode):
    if item < currentNode.item:
        if currentNode.left is None:
            currentNode.left=TreeNode(item)
        else:
            self._insert(item,currentNode.left)
    elif item > currentNode.item:
        if currentNode.right is None:
            currentNode.right=TreeNode(item)
        else:
            self._insert(item,currentNode.right)
    else:
        print("Item is already existed in the tree")
```



Searching Binary Search Tree

```
# pseudocode to search a target in a binary search tree
search(target,Tree)
        if (Tree is empty)
                return False # not present!
        if root == target
                return True
        if root > target   # search target in Left subtree
                search(target, LeftSubtree)
        else   #search target in Right subtree
                search(target, RightSubtree )
```

Searching Binary Search Tree



Lab Exercise

- Implement the Search Operation in a Binary Search Tree.