# In-Order Traversal

In this lesson, we will cover In-Order Traversal and implement it in Python

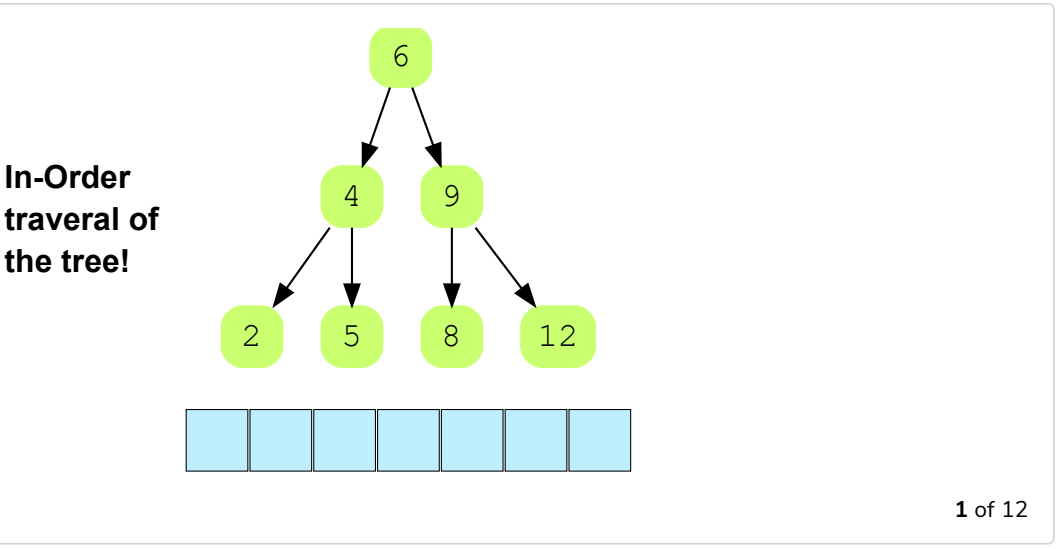
We'll cover the following



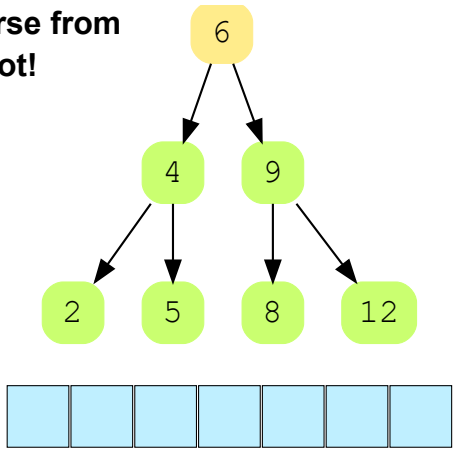
## Introduction #

In In-order traversal, the elements are traversed in “left-root-right” order so they are traversed *in order*. In other words, elements are printed in sorted ascending order with this traversal. We first visit the left child, then the root/parent node, and then the right child. Here is a high-level description of the in-order traversal algorithm,

1. Traverse the left sub-tree of the ‘currentNode’ recursively by calling the `inOrderPrint()` function on it.
2. Visit the current node and print its value
3. Traverse the right sub-tree of the ‘currentNode’ recursively by calling the `inOrderPrint()` function on it.

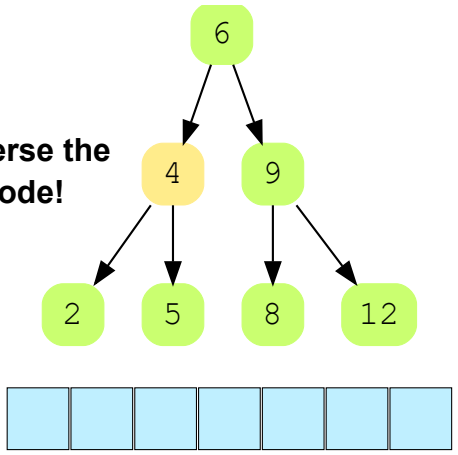


Traverse from the root!



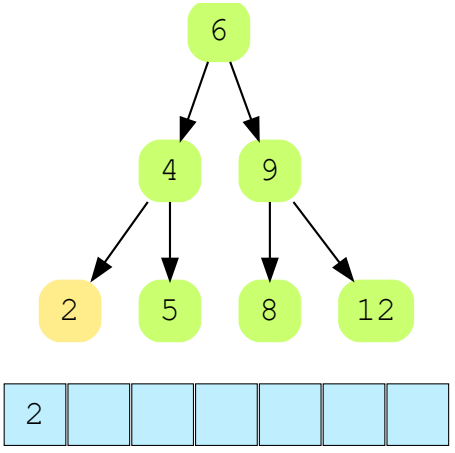



Traverse the left node!

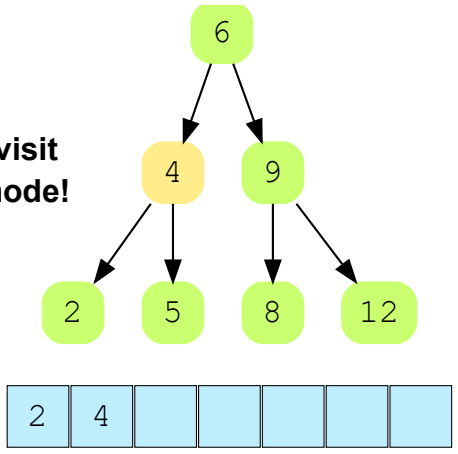





Now visit its left child!

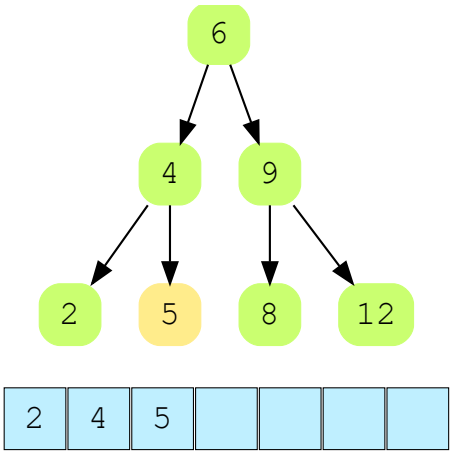


Now visit
this node!



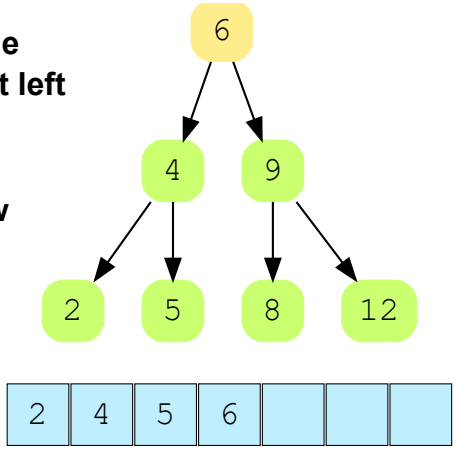


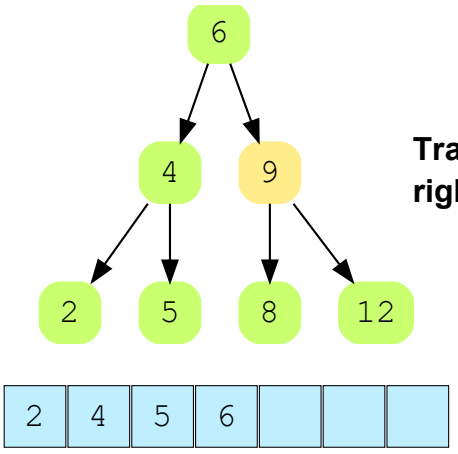

Now visit
its right child!





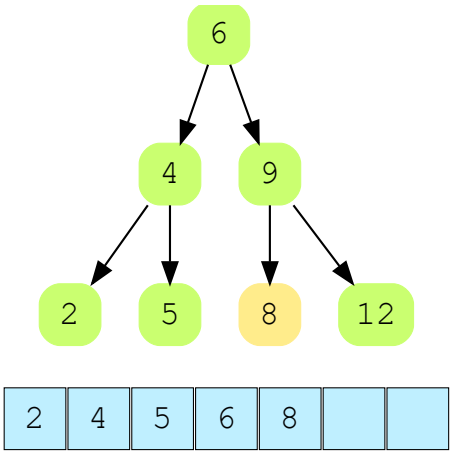

Since all the
elements at left
have been
visited,
we will now
visit the
root!





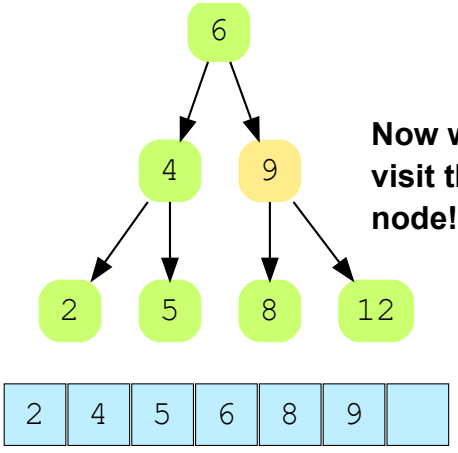

**Traverse the right child!**

**8** of 12



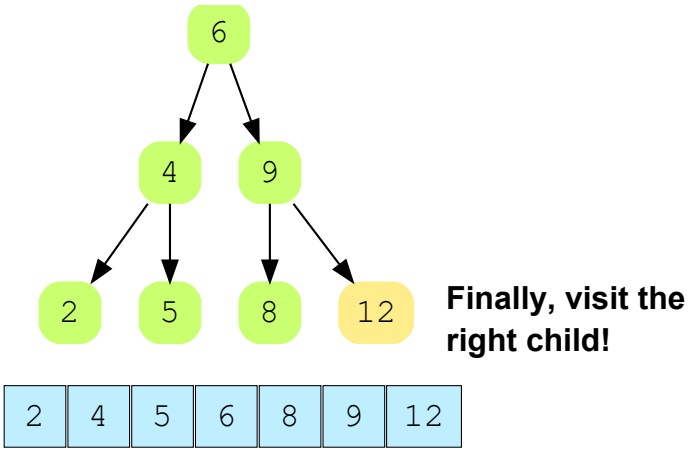

**Visit its left child!**

**9** of 12

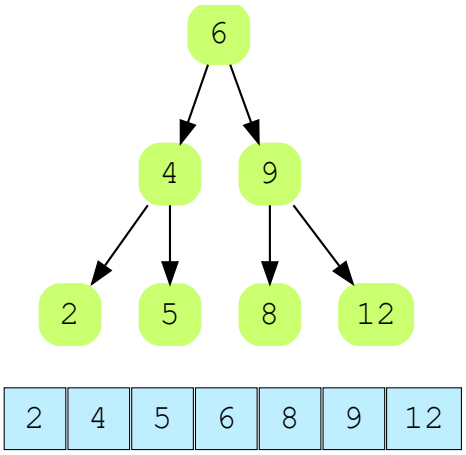



**Now we will visit this parent node!**

**10** of 12




**11** of 12




**12** of 12

# Implementation in Python #

main.py

BinarySearchTree.py

Node.py

```
from Node import Node
from BinarySearchTree import BinarySearchTree


def inOrderPrint(node):
    if node is not None:
        inOrderPrint(node.leftChild)
        print(node.val)
        inOrderPrint(node.rightChild)


BST = BinarySearchTree(6)
BST.insert(4)
BST.insert(9)
```

```
BST.insert(5)
BST.insert(2)
BST.insert(8)
BST.insert(12)

inOrderPrint(BST.root)

```

## Explanation #

First, we create an object of the `BinarySearchTree` class and insert some values into it. We then pass the tree's root to the `inOrderPrint()` function. If the node given is not `None`, this function calls `inOrderPrint()` on the left child first, then on the root, and then finally on the right child.

If you run the code for the BST above, it will print out the following

$$[2, 4, 5, 6, 8, 9, 12]$$

This lesson marks the end of our study of binary search trees. We will now move on to other kinds of trees.

Back

Post-Order Traversal

Next

What is an AVL Tree?

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/in-order-traversal__introduction-to-trees__data-structures-for-coding-interviews-in-python)