Pre-Order Traversal

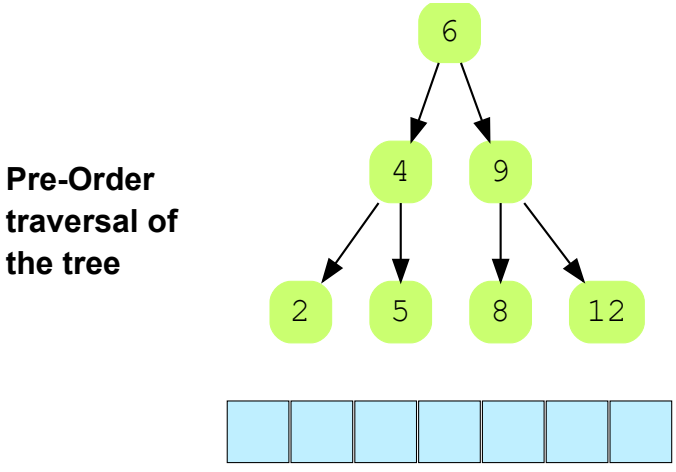
In this lesson, we will cover the traversal strategy, 'Pre-Order Traversal' in a Binary Search Tree, and its implementation it in Python

We'll cover the following

Introduction

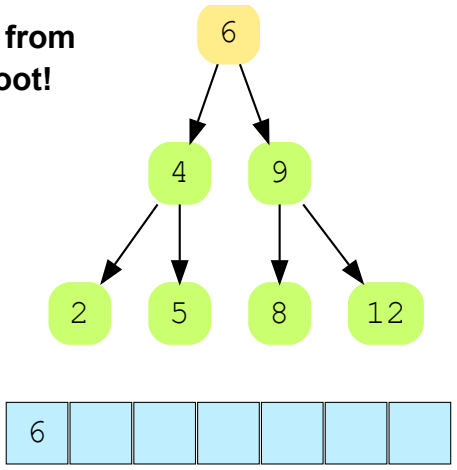
In this traversal, the elements are traversed in “root-left-right” order. We first visit the root/parent node, then the left child, and then the right child. Here is a high-level description of the algorithm for *Pre-Order* traversal, starting from the root node:

1. Visit the current node, i.e., print the value stored at the node
2. Call the `preOrderPrint()` function on the left sub-tree of the ‘current Node’.
3. Call the `preOrderPrint()` function on the right sub-tree of the ‘current Node’.



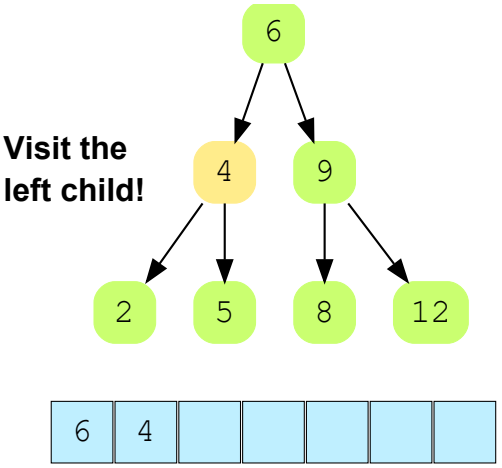
1 of 9

Start from the root!



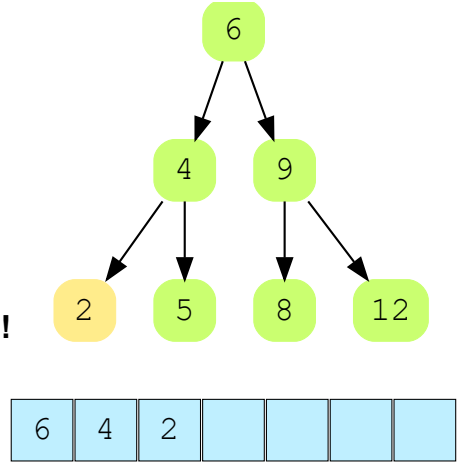
2 of 9

Visit the left child!

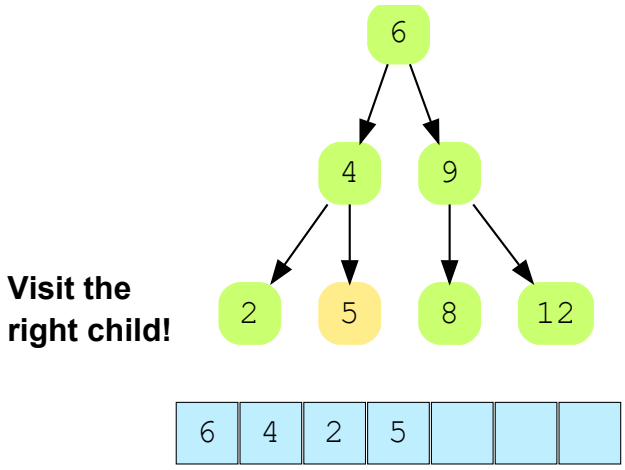


3 of 9

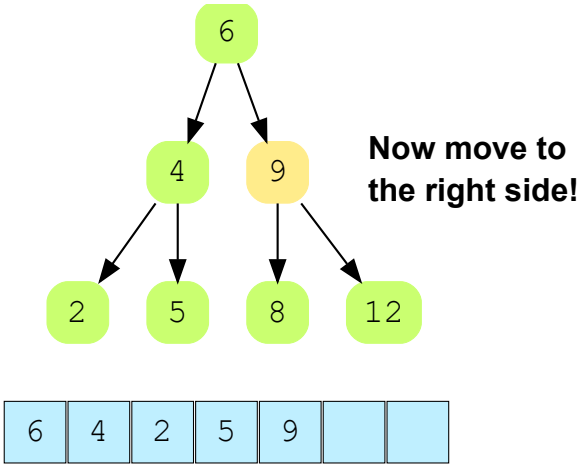
Visit its left child!



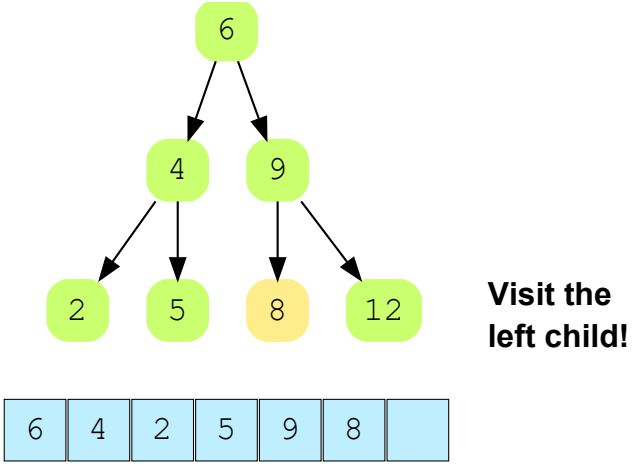
4 of 9



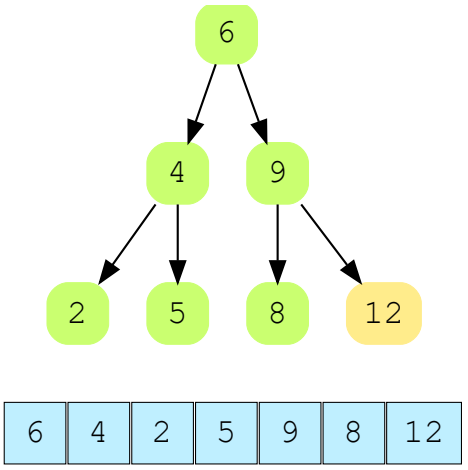
5 of 9



6 of 9

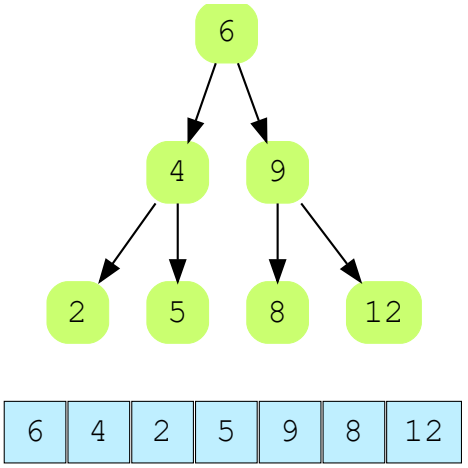


7 of 9



Now visit the right child!

8 of 9



9 of 9

Implementation in Python

main.py

BinarySearchTree.py

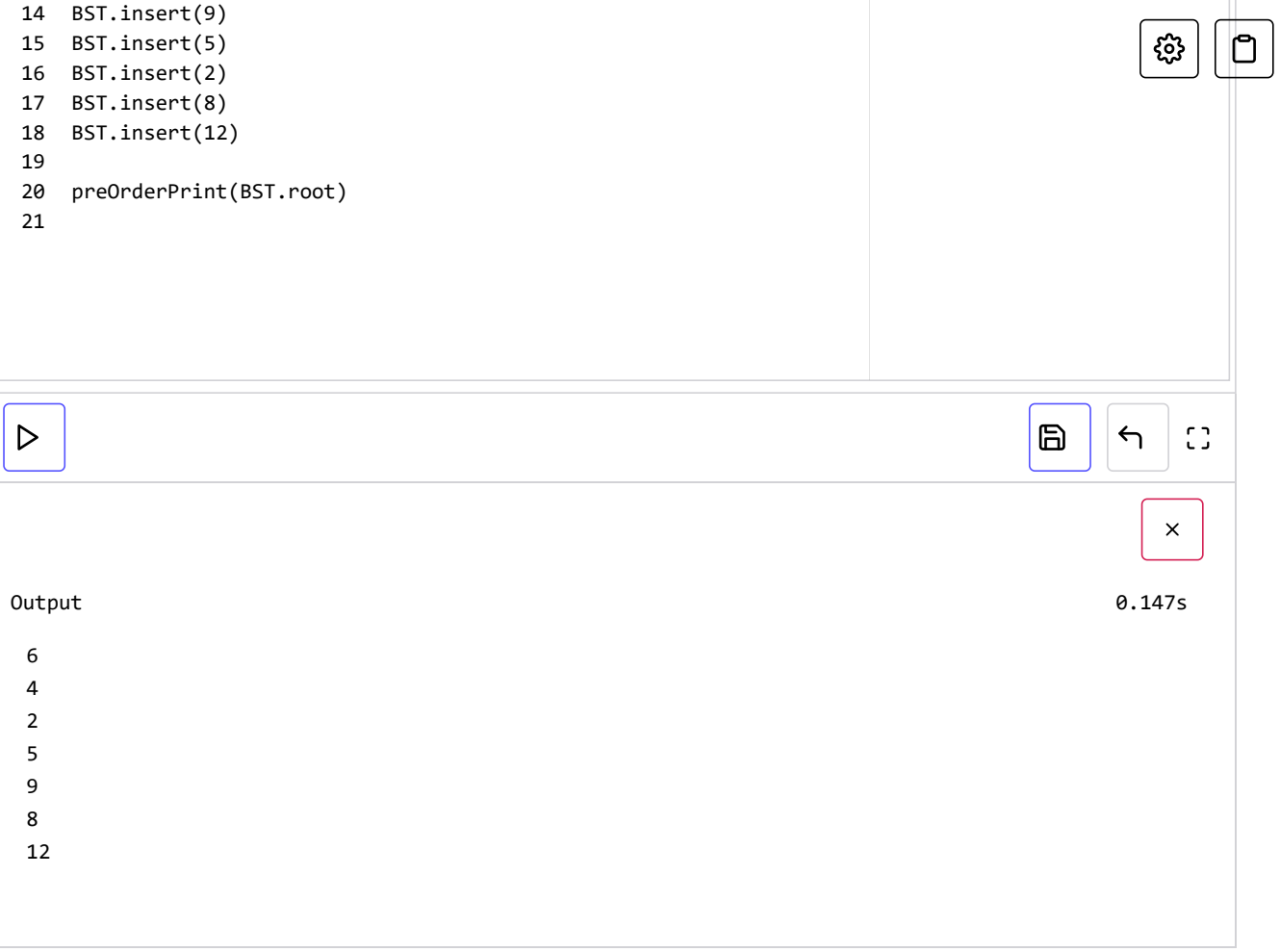
Node.py

```
from Node import Node
from BinarySearchTree import BinarySearchTree


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


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

```
14  BST.insert(9)
15  BST.insert(5)
16  BST.insert(2)
17  BST.insert(8)
18  BST.insert(12)
19
20  preOrderPrint(BST.root)
21
```



```
Output                                                    0.147s

6
4
2
5
9
8
12
```

Explanation

First, we create an object of the `BinarySearchTree` class and insert some values into it. We will then pass the tree's root to the `preOrderPrint()` function. If the node given is *not* `None`, this function prints the value at the node and calls `preOrderPrint()` on the left child first and then on the right child. Also note that we have hidden the `Node` class to make the code shorter! We have done the same for the next couple of chapters.

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

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

Time Complexity

This is a linear time algorithm, i.e., the time complexity of is in $O(n)$ because a total of n recursive calls occur.

If you have understood Pre-Order Traversal clearly, it will be a piece of cake for you to understand the rest of the traversals, as all three of them are similar to one another. In the next lesson, we are going to study another type of BST Traversal known as Post-Order Traversal.