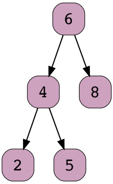
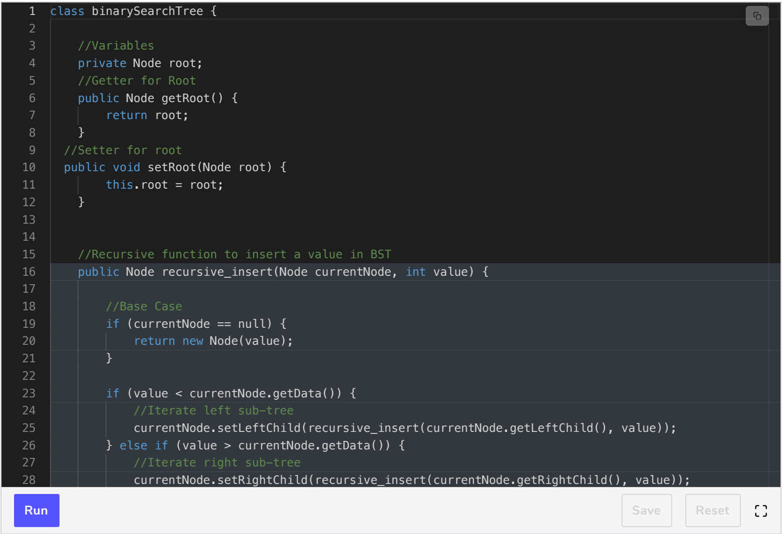
What is a Binary Search Tree?

A Binary Search Tree (BST) is a hierarchical data structure that consists of vertices connected through edges. The value of the left node is less than the value of the parent node, and the value of the right node is greater than the value of the parent node.



Code Implementation



```
class binarySearchTree {

    //Variables
    private Node root;
    //Getter for Root
    public Node getRoot() {
        return root;
    }
  //Setter for root
  public void setRoot(Node root) {
        this.root = root;
    }


    //Recursive function to insert a value in BST
    public Node recursive_insert(Node currentNode, int value) {

        //Base Case
        if (currentNode == null) {
            return new Node(value);
        }

        if (value < currentNode.getData()) {
            //Iterate left sub-tree
            currentNode.setLeftChild(recursive_insert(currentNode.getLeftChild(), value));
        } else if (value > currentNode.getData()) {
            //Iterate right sub-tree
            currentNode.setRightChild(recursive_insert(currentNode.getRightChild(), value));
```

Recursive Insert

Understanding the Code

In the code above, the function `insert` is a recursive method, since it makes a **recursive call**. Below is an explanation of the above code:

Driver Method

- In the `main` code, we create a new BST, named `bsT`.
- The `insert()` method is subsequently called to insert nodes in the BST.
- The `printTree()` method prints the BST using the in order traversal. It takes the **root** of the tree to begin the traversal. This process just prints the BST in order.

Recursive Method

- The return type of the `insert()` function is `boolean` and takes one integer type of input parameter `value` which depicts the value of the node. This method calls the actual recursive method `recursive_insert` that takes two input parameters. The first is `currentNode` of type `Node`. The second parameter `value` is the integer to be inserted in the BST. We will discuss the `recursive_insert` method below.

Base Case

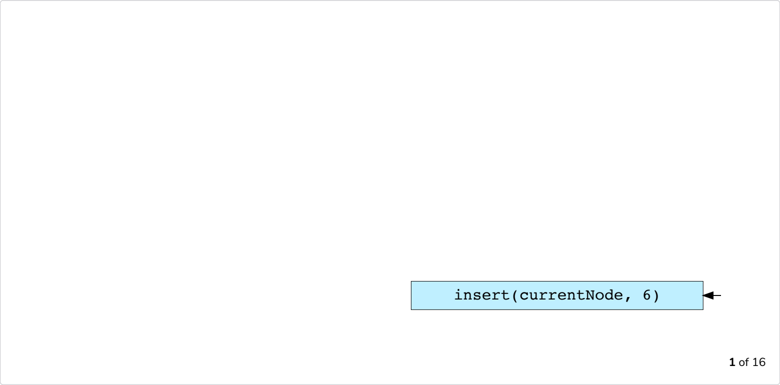
- We have defined a base case for the method in first `if` condition between **lines 19 to 21**.
- If the value of the `currentNode` is `null`, meaning there is an available space for the child node to be inserted, a `new Node()` is created with the `value`.

Recursive Case

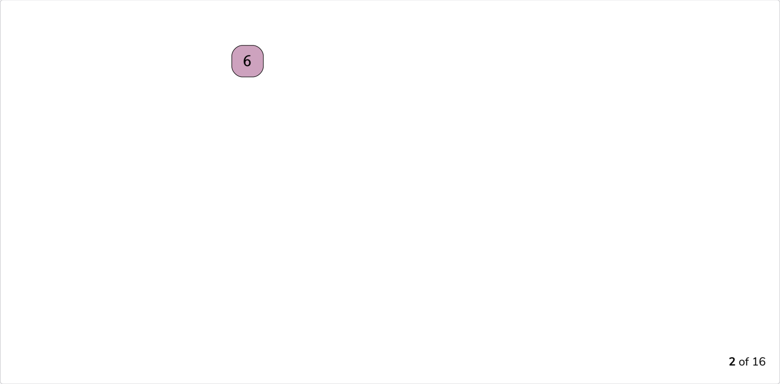
- If the base case condition is not met and the `value` to be inserted is **less** than the value of `currentNode.getData()`, the function enters the second `if` condition, **line 25**, where it makes a **recursive call**.
- In this process, the `recursive_insert` method is recursively .The first parameter is `currentNode.getLeftChild()` and the second parameter is `value`.
- If the method has not reached the base and the inserted `value` is **greater** than the value of `currentNode.getData()`, the function enters the other `else if` condition, in **line 26-28**, where, it makes a recursive call.
- In this `else if` condition, the `recursive_insert` method is called. The first parameter is `currentNode.getRightChild()` and the second parameter is `value`.
- If all the above conditions are not met, we reach the `else` condition from **lines 29 to 31** which returns the `currentNode` since the defined `value` already exists.
- **Line 34** returns the `currentNode` after being inserted into its position.
- Subsequent recursive calls will continually be made until the first parameter equals `null`, which ensures the availability of a position for the new node. The new node is then added to its corresponding position.

For a better explanation of the insert method, see the section below.

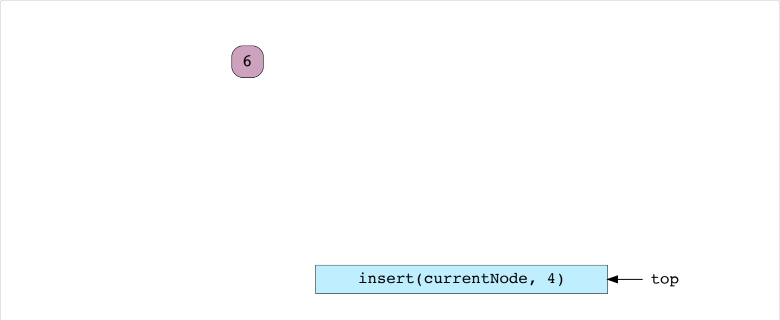
Understanding through a Stack

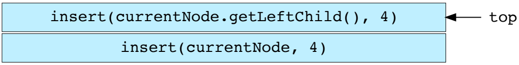


1 of 16

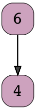


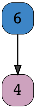
2 of 16

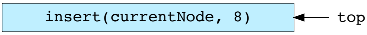


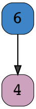

insert(currentNode.getLeftChild(), 4)

insert(currentNode, 4)



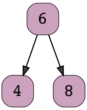


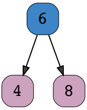

insert(currentNode, 8)



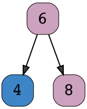
`insert(currentNode.getRightChild(), 8)` ← top

`insert(currentNode, 8)`



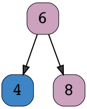


`insert(currentNode, 2)` ← top



insert(currentNode.getLeftChild(), 2) ← top

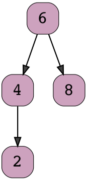
insert(currentNode, 2)

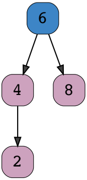


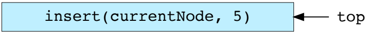
insert(currentNode. getLeftChild(), 2) ← top

insert(currentNode.getLeftChild(), 2)

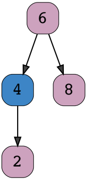
insert(currentNode, 2)

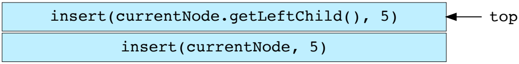





`insert(currentNode, 5)`

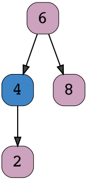
top

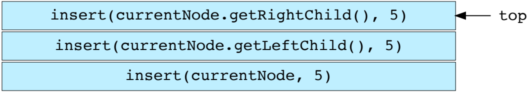



`insert(currentNode.getLeftChild(), 5)`

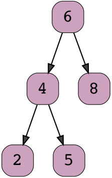
`insert(currentNode, 5)`

top





```
insert(currentNode.getRightChild(), 5)
insert(currentNode.getLeftChild(), 5)
insert(currentNode, 5)
```

top