Binary Search Tree

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

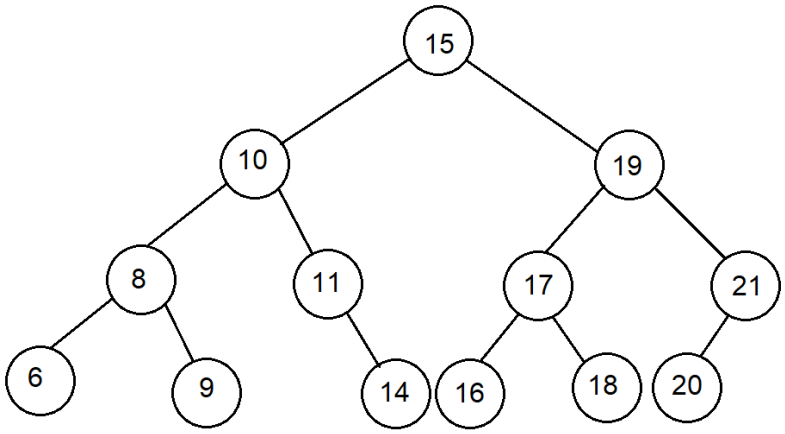
Advantages of using binary search tree

- Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
- The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes $O(\log_2 n)$ time. In worst case, the time it takes to search an element is O(n).
- It also speeds up the insertion and deletion operations as compare to that in array and linked list.

Problem description: A program in java to check whether a tree is binary search tree or not.

Output: If the tree is binary search tree, it gives the output "Is a binary search tree", else, it gives the output "Not a binary search tree".

Tree used in this code:



Time complexity of the code: O(n)

Program in java:

```java
// Java program to check if a given tree is BST.

public class BST {

    static int prev = Integer.MIN_VALUE;
    /* A binary tree node has data, pointer to
    left child and a pointer to right child */
    static class Node {
        int data;
        Node left, right;

        Node(int data)
        {
            this.data = data;
            left = right = null;
        }
    };

    // Utility function to check if Binary Tree is BST
    public static boolean checkBST(Node root)
    {
        // traverse the tree in inorder fashion and
        // keep track of prev node
        if (root != null) {
            if (!checkBST(root.left))
                return false;

            // Allows only distinct valued nodes
            if (root.data <= prev)
                return false;

            // Initialize prev to current
            prev = root.data;

            return checkBST(root.right);
        }

        return true;
    }

    // Function to check if Binary Tree is BST
    public static boolean isBST(Node root)
    {
        return checkBST(root);
    }
```

```
/* Driver code*/
    public static void main(String[] args){

        //entering a valid BST

        /*
                        15
                      /    \
                    10      19
                   /  \     /  \
                  8   11   17   21
                 / \   \   / \  /
                6  9   14 16 18 20

         This is the tree passed as input


         */
        Node root = null;
        root = new Node(15);
        root.left = new Node(10);
        root.right = new Node(19);
        root.left.left = new Node(8);
        root.left.right = new Node(11);
        root.left.left.left = new Node(6);
        root.left.left.right = new Node(9);
        root.left.right.right = new Node(14);
        root.right.left = new Node(17);
        root.right.right = new Node(21);
        root.right.left.left = new Node(16);
        root.right.left.right = new Node(18);
        root.right.right.left = new Node(20);


        if (isBST(root))
            System.out.println("Is a binary search tree");
        else
            System.out.println("Not a binary search tree");
    }
}
```

Output:

```
"C:\Program Files\Java\jdk-11.0.3\bin\java.exe" "-javaagent:C:\Pr
Is a binary search tree

Process finished with exit code 0
```