Title :- Binary search Tree (BST) operations

problem statement :- Beginning with an empty binary search tree, construct binary search tree by inserting the values in the order given After constructing a binary tree.

i] Insert new node. ii] find number of nodes in longest Path from the root

3] minimum data value found in the tree.

4] change a tree so that the roles of the right and left pointers are swapped at every node

5] search a value.

Objectives :- To construct and understand the concept the Binary search Tree (BST) and its operations.

Software Requirement :- g++ / gcc compiler, 64-bit Fedora eclipse IDE.

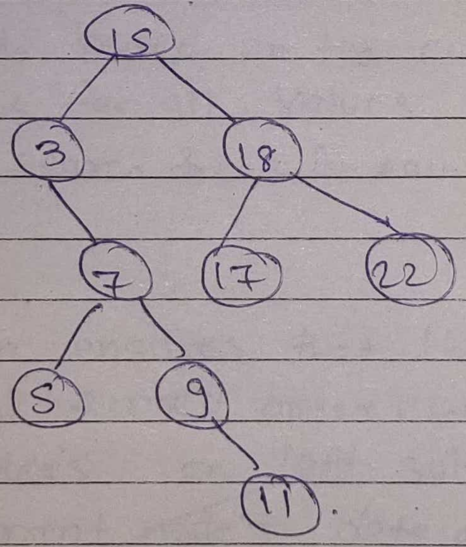
Theory :- A binary search tree is a binary tree where the value of each node is greater or equal to the values in its left subtree and less than or equal to the values in the right subtrees. This property allows for efficient searching and insertion of values in the tree.

- when inserting values into a binary search tree, the values are compared to the value of the root node. If the value is less than the root node, it is inserted in the left subtree otherwise is inserted in the right subtree.

This process is repeated recursively until a lead node is reached at which point the new node is inserted as a child of the leaf node.

The order in which values are inserted into the tree can affect the height and the balance of the tree In the worst case scenario, when the values are inserted in ascending or deconding order, the resulting tree can becomes degenerate tree, where all the nodes have only one child and the tree, where all that nodes have only one child. and the. tree become essentially a linked list. To avoid this, it is important to insert. values in a balanced order, such as randomly or by using a balanced binary search tree algorith.

Example : Keys :- 15, 3, 7, 9, 11, 18, 22, 17, 5



Binary search Tree (BST)

Algorithm :-

1. Start
2. create an empty binary search tree.
3. Read the values to be inserted in the binary search. tree in the order given
4. for each value to be inserted do this following
   a. If the binary search tree is empty, create a new nodes with the value as its data and make it the root node of the tree.
   b. If the binary search tree is not empty compare the value, to be inserted with the data of the current node.
   c. create a new node with the value as its data, and insert it as a child node. of the last visited leaf node based on the comparisons in step b.
5. Repeat steps for all values to be inserted
6. The binary search trees is now constructed
7. stop.

- This algorithm ensures that the binary search tree is always in the correct order. with values less than the current nodes's on left subtree and values greater than the current node's data on the right subtree.
- The time complexity of this algorithm is o(logn) in the average case, where n is the number of values to be inserted and worst-case time complexity is $O(n^2)$

Conclusion :- Binary search trees are useful data structures for searching data and their efficient construction and traversal algorithm