ALGORITHM:

Binary Search Tree

Step 1: Start

Step 2: Declare a structure and structure pointers for insertion, deletion and search operation and also declare a function for inorder traversal.

Step 3: Declare a pointer as root and also the required variable.

Step 4: Read the choice from the user to perform insertion, deletion, searching and inorder traversal.

Step 5: If the user choose to perform insertion operation then read the value which is to be inserted to the ~~root~~ tree from the user.

Step 5.1: The value to the insert pointer and also the root pointer.

Step 5.2: check if ! root then allocate memory for the root

Step 5.3: Set the value to the info part of the root and then set left and right part of the root to null and return root.

Step 5.4 : check if root → info > x then call the insert pointer to insert to left of the root

Step 5.5 : check if root → info < x then call the insert pointer to insert to the right of the root

Step 5.6 : Return the root.

Step 6 : If the user choose to perform deletion operation then read the element to be deleted from the tree pass the root pointer and the item to the delete pointer.

Step 6.1 : check if not ptr then print node not found

Step 6.2 : else if ptr → info < x the call delete pointer by passing the right pointer and the item.

Step 6.3 : else if ptr → info > x then call delete pointer by passing the left pointer and the item.

Step 6.4 : check if ptr → info == item then check if ptr → left == ptr → right then free ptr and return null.

Step 6.5 : else if ptr → left == null then set p1. ptr → right and free ptr, return p1

Step 6.6 : Else if ptr → right == null then Set
P1 = ptr → left and free ptr, return p1

Step 6.7 : Else set p1 = ptr → right and p2 = ptr → right

Step 6.8 : while p1 → left not equal to null, Set
P1 → left ptr → left and free ptr, return p2

Step 6.9 : Return ptr

Step 7 : If the user choose to perform Search operation
the call the pointer to perform search oper

Step 7.1 : Declare the neccessary pointers and vanables

Step 7.2 : Read the element to be Searched.

Step 7.3 : while ptr check if item > ptr → info then
ptr = ptr → right

Step 7.4 : Else if item < ptr → info then ptr = ptr → left

Step 7.5 : Else break

Step 7.6 : check if ptr then print that the element
is found.

Step 7.7 : Else print element not found in tree
and return root

Step 8 : If the user choose to perform traversal then call the traversal function and pass the root pointers.

Step 8.1 : If root not equals to null recursively all the functions by passing root → left

Step 8.2 : print root → info

Step 8.3 : call the traversal function recursively by passing root → right.

Step 9 : End