Q. Insertion and Deletion in a Binary Search Tree

Algorithm :

Step 1: Start

Step 2: Define a structure 'Node' with 'data', 'left' and 'right' pointers.

Step 3: Create a 'createNode(data)' function to create a new node with 'data'.

Step 4: Implement 'insert(root, key)' to insert 'key' into the BST

Step 5: In 'insert', if 'root' is NULL, return 'createNode(key)'.

Step 6: If 'key' is less, call 'insert(root→left, key)'; else 'insert(root→right, key);'.

Step 7: Implement 'minValueNode(node)' to find the node with minimum value.

Step 8: In 'deleteNode(root, key)', if 'root' is NULL, return 'root'.

Step 9: If 'key' is less, call 'deleteNode(root→left, key);'.

Step 10: If 'key' is greater, call 'deleteNode(root→right, key);'.

Step 11: In 'deleteNode' handle deletion based on the number of children or replace with a minimum value node if needed. Return 'root'.

Step 12: Stop

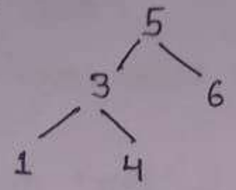
Pseudo Code :

```
Struct Node :
    Int data
    Node * left
    Node * right

Function createNode (data) :
    Node * n = New Node
    n → data = data
    n → left = NULL
    n → right = NULL
    Return n
Function insert (root, key) :
    If root = = NULL :
        Return createNode (key)
    If key < root → data :
        root → left = insert (root → left , key )
    Else If key > root → data :
        root → right = insert (root → right , key)

    Return root

Function minValueNode (node)
    Node * current = node
    While current → left ! = NULL :
        current = current → left
    Return current

Function deleteNode (root, key):
    If root == NULL :
        Return root

    If key < root → data :
        root → left = delete Node (root → left , key)
    Else If key > root → data :
        root → right = delete Node (root → right , key)
```

```
Else:
    If root → left == NULL:
        // Handle deletion with one child or no child
    Else If root → right == NULL:
        // Handle deletion with one child or no child
    Else:
        // Handle deletion with two children
Return root
```

Output:

Initial BST :



Insert 2 :

```
Inserting 2 :
In - order traversal after insertion :
1 2 3 4 5 6
```

~~Delete~~

Delete 3 :

```
Deleting 3 :
In-order traversal after deletion :
1 2 4 5 6
```