**Experiment – 3.2**

**Student Name: Vivek UID:**

**Branch: CSE Section/Group:**

**Date of Performance: Semester: 3**

**Subject Name: OOPs using JAVA Subject Code: 21CSH-218**

**Aim of the practical:** Write a program to implement of different operation on a binary search tree

# Algorithm:

## Insertion-

1. Create a new BST node and assign values to it.
2. insert(node, key) if root == NULL,

return the new node to the calling function. if root=>data < key

call the insert function with root=>right and assign the return value in root=>right. root->right = insert(root=>right,key)

if root=>data > key

call the insert function with root->left and assign the return value in root=>left. root=>left = insert(root=>left,key)

1. Finally, return the original root pointer to the calling function.

## Deletion- 1.Leaf Node

If the node is leaf (both left and right will be NULL), remove the node directly and free its memory.

## Node with Right Child

If the node has only right child (left will be NULL), make the node points to the right node and free the node.

## Node with Left Child

If the node has only left child (right will be NULL), make the node points to the left node and free the node.

## Node has both left and right child

If the node has both left and right child,

find the smallest node in the right subtree. say min make node->data = min

Again delete the min node.

**Program code:** #include <iostream> using namespace std; struct Node {

int data;

Node \*left;

Node \*right;

};

Node\* create(int item)

{

Node\* node = new Node; node->data = item;

node->left = node->right = NULL; return node;

}

void inorder(Node \*root)

{

if (root == NULL) return;

inorder(root->left); cout<< root->data << " "; inorder(root->right);

}

Node\* findMinimum(Node\* cur)

{

while(cur->left != NULL) { cur = cur->left;

}

return cur;

}

Node\* insertion(Node\* root, int item)

{

if (root == NULL) return create(item);

if (item < root->data)

root->left = insertion(root->left, item); else

root->right = insertion(root->right, item); return root;

}

void search(Node\* &cur, int item, Node\* &parent)

{

while (cur != NULL && cur->data != item)

{

parent = cur;

if (item < cur->data) cur = cur->left;

else

cur = cur->right;

}

}

void deletion(Node\*& root, int item)

{

Node\* parent = NULL; Node\* cur = root; search(cur, item, parent); if (cur == NULL)

return;

if (cur->left == NULL && cur->right == NULL)

{

if (cur != root)

{

if (parent->left == cur) parent->left = NULL;

else

parent->right = NULL;

}

else

root = NULL; free(cur);

}

else if (cur->left && cur->right)

{

Node\* succ = findMinimum(cur->right); int val = succ->data;

deletion(root, succ->data); cur->data = val;

}

else

{

Node\* child = (cur->left)? cur->left: cur->right; if (cur != root)

{

if (cur == parent->left) parent->left = child;

else

parent->right = child;

}

else

root = child; free(cur);

}

}

int main()

{

Node\* root = NULL; root = insertion(root, 5); root = insertion(root, 9); root = insertion(root, 3); root = insertion(root, 7); root = insertion(root, 2); root = insertion(root, 5); root = insertion(root, 6); root = insertion(root, 4);

printf("The inorder traversal of the given binary tree is - \n"); inorder(root);

deletion(root, 7);

printf("\nAfter deleting node 25, the inorder traversal of the given binary tree is - \n"); inorder(root);

insertion(root, 1);

printf("\nAfter inserting node 2, the inorder traversal of the given binary tree is - \n"); inorder(root);

return 0;

}

# Output:

