**LAB – 7**

**NAME – TUSHAR MAROO**

**SECTION – A**

**REGISTRATION NUMBER – 190905238**

**ROLL NUMBER – 40**

**Q1.**

**CODE**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int info;

struct node \*left, \*right;

} NODE;

NODE \*create(NODE \*root, int x)

{

NODE \*getnode;

if (root == NULL)

{

root = (NODE \*)malloc(sizeof(NODE));

root->info = x;

root->left = root->right = NULL;

}

else if (x > root->info)

root->right = create(root->right, x);

else if (x < root->info)

root->left = create(root->left, x);

else

{

printf("Duplicate node\n");

exit(0);

}

return (root);

}

void inorder(NODE \*root)

{

if (root != NULL)

{

inorder(root->left);

printf("%5d", root->info);

inorder(root->right);

}

}

void postorder(NODE \*root)

{

if (root != NULL)

{

postorder(root->left);

postorder(root->right);

printf("%5d", root->info);

}

}

void preorder(NODE \*root)

{

if (root != NULL)

{

printf("%5d", root->info);

preorder(root->left);

preorder(root->right);

}

}

int max(int x, int y)

{

return x > y ? x : y;

}

int height(NODE \*root)

{

if (root == NULL)

return 0;

return 1 + max(height(root->left), height(root->right));

}

int printBalanceFactor(NODE \*root)

{

if (root != NULL)

{

printf("\nBalance factor of node with value %d : %d", root->info, height(root->left) - height(root->right));

printBalanceFactor(root->left);

printBalanceFactor(root->right);

}

}

void main()

{

int n, x, ch, i;

NODE \*root;

root = NULL;

printf("-----------Menu-----------\n");

printf(" 1. Insert\n 2. All traversals\n 3. Get Balance Factor\n 4. Exit\n");

while (1)

{

printf("Enter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter node (do not enter duplicate nodes) : ");

scanf("%d", &x);

root = create(root, x);

break;

case 2:

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nInorder traversal : ");

inorder(root);

printf("\nPreorder traversal : ");

preorder(root);

printf("\nPostorder traversal : ");

postorder(root);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 3:

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printBalanceFactor(root);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 4:

exit(0);

default:

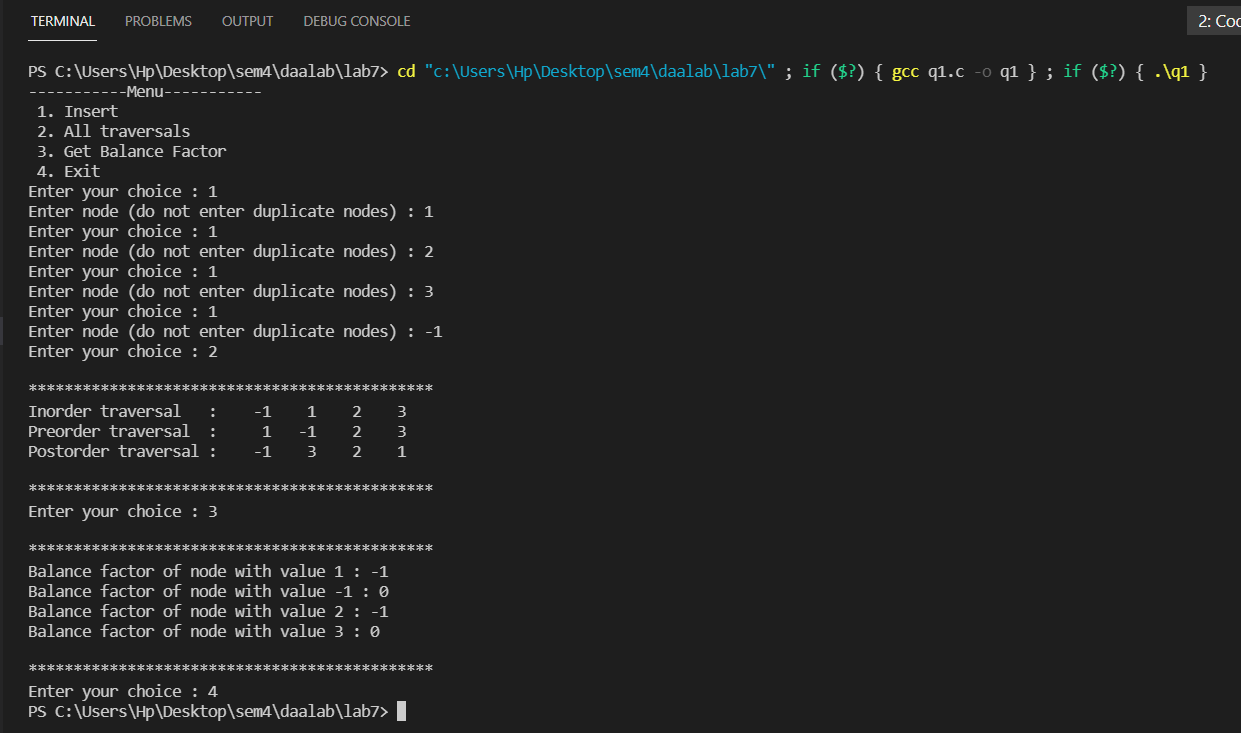
printf("Invalid Choice\n");

}

}

}

**OUTPUT**

****

**ANALYSIS**

Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST. The cost of these operations may become O(n) for a skewed Binary tree. So best case time case time complexity is O(logn) and worst case time complexity is O(n).

**Q2.**

**CODE**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int info;

struct node \*left, \*right;

} NODE;

struct Stack

{

int top;

unsigned capacity;

NODE \*\*array;

};

struct Stack \*createStack(unsigned capacity)

{

struct Stack \*stack = (struct Stack \*)malloc(sizeof(struct Stack));

stack->capacity = capacity;

stack->top = -1;

stack->array = (NODE \*\*)malloc(stack->capacity \* sizeof(NODE \*));

return stack;

}

int isFull(struct Stack \*stack)

{

return stack->top == stack->capacity - 1;

}

int isEmpty(struct Stack \*stack)

{

return stack->top == -1;

}

void push(struct Stack \*stack, NODE \*item)

{

if (isFull(stack))

return;

stack->array[++stack->top] = item;

// printf("%d pushed to stack\n", item);

}

NODE \*pop(struct Stack \*stack)

{

if (isEmpty(stack))

return NULL;

return stack->array[stack->top--];

}

NODE \*peek(struct Stack \*stack)

{

if (isEmpty(stack))

return NULL;

return stack->array[stack->top];

}

int max(int x, int y)

{

return x > y ? x : y;

}

int height(NODE \*root)

{

if (root == NULL)

return 0;

return 1 + max(height(root->left), height(root->right));

}

int getBalFactor(NODE \*root)

{

return height(root->left) - height(root->right);

}

NODE \*rightRotate(NODE \*y)

{

NODE \*x = y->left;

NODE \*T2 = x->right;

x->right = y;

y->left = T2;

return x;

}

NODE \*leftRotate(NODE \*x)

{

NODE \*y = x->right;

NODE \*T2 = y->left;

y->left = x;

x->right = T2;

return y;

}

NODE \*create(NODE \*root, int x)

{

struct Stack \*stack = createStack(100);

NODE \*newnode = (NODE \*)malloc(sizeof(NODE));

newnode->info = x;

newnode->right = NULL;

newnode->left = NULL;

NODE \*curr = root;

NODE \*trail = NULL;

while (curr != NULL)

{

trail = curr;

push(stack, trail);

if (x < curr->info)

curr = curr->left;

else if (x > curr->info)

curr = curr->right;

else

{

printf("Duplicate element\n");

exit(0);

}

}

if (trail == NULL)

{

trail = newnode;

return trail;

}

else if (x < trail->info)

trail->left = newnode;

else

trail->right = newnode;

NODE \*newRoot = root;

while (!isEmpty(stack))

{

NODE \*toBalance = pop(stack);

NODE \*prev = peek(stack);

int balance = getBalFactor(toBalance);

if (balance > 1 && x < toBalance->left->info)

{

toBalance = rightRotate(toBalance);

}

else if (balance < -1 && x > toBalance->right->info)

{

toBalance = leftRotate(toBalance);

}

else if (balance > 1 && x > toBalance->left->info)

{

toBalance->left = leftRotate(toBalance->left);

toBalance = rightRotate(toBalance);

}

else if (balance < -1 && x < toBalance->right->info)

{

toBalance->right = rightRotate(toBalance->right);

toBalance = leftRotate(toBalance);

}

if (prev != NULL && prev->info > toBalance->info)

{

prev->left = toBalance;

}

else if (prev != NULL)

{

prev->right = toBalance;

}

newRoot = toBalance;

}

return newRoot;

}

void inorder(NODE \*root)

{

if (root != NULL)

{

inorder(root->left);

printf("%5d", root->info);

inorder(root->right);

}

}

void postorder(NODE \*root)

{

if (root != NULL)

{

postorder(root->left);

postorder(root->right);

printf("%5d", root->info);

}

}

void preorder(NODE \*root)

{

if (root != NULL)

{

printf("%5d", root->info);

preorder(root->left);

preorder(root->right);

}

}

int printBalanceFactor(NODE \*root)

{

if (root != NULL)

{

printf("\nBalance factor of node with value %d : %d", root->info, getBalFactor(root));

printBalanceFactor(root->left);

printBalanceFactor(root->right);

}

}

void main()

{

int n, x, ch, i;

NODE \*root;

root = NULL;

printf("-----------Menu-----------\n");

printf(" 1. Insert\n 2. All traversals\n 3. Get Balance Factor\n 4. Exit\n");

while (1)

{

printf("Enter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter node (do not enter duplicate nodes) : ");

scanf("%d", &x);

root = create(root, x);

break;

case 2:

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nInorder traversal : ");

inorder(root);

printf("\nPreorder traversal : ");

preorder(root);

printf("\nPostorder traversal : ");

postorder(root);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 3:

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printBalanceFactor(root);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 4:

exit(0);

default:

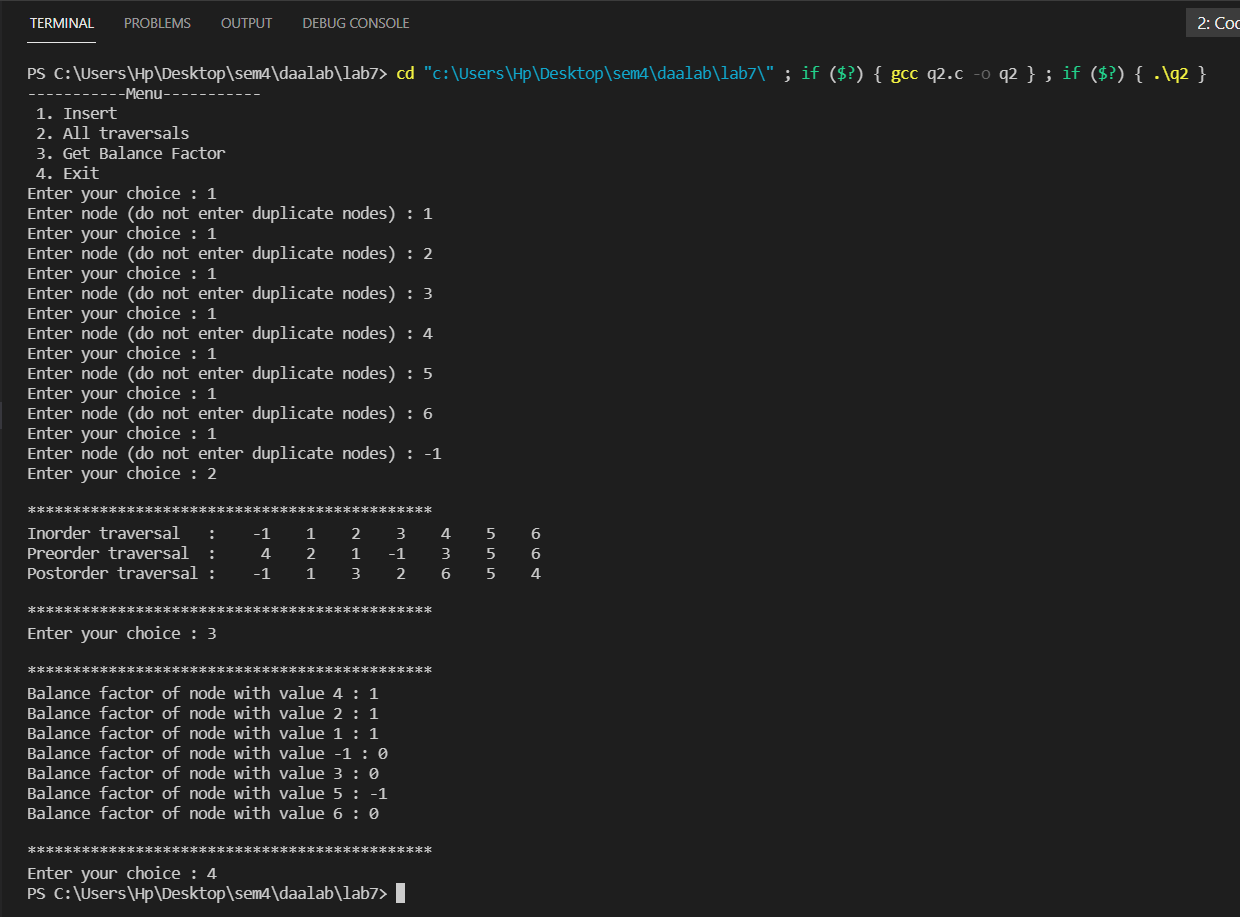
printf("Invalid Choice\n");

}

}

}

**OUTPUT**

****

**ANALYSIS**

The height of an AVL tree is always O(logn) where n is the number of nodes in the tree. Thus most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(logn) time.