DAA WEEK7 SUBMISSION

Name: Davasam Karthikeya   Section: AIMLB   Reg No: 230962326

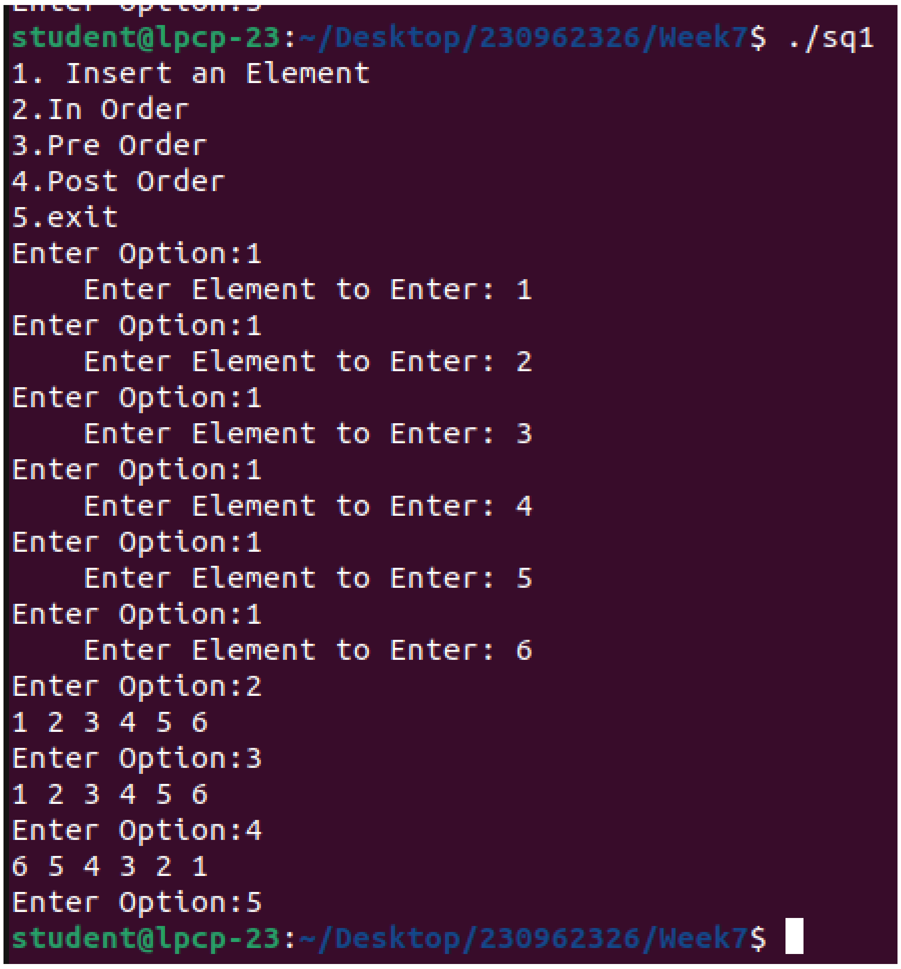
Date: 28/02/2025

1) Write a program to create a binary search tree and display its elements using all the traversal methods and analyse its time efficiency.

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{
    int data;
    struct Node *left;
    struct Node *right;
}*node_ptr;

node_ptr create_node(int data){
    node_ptr ptr = (node_ptr) malloc(sizeof(node_ptr));
    ptr->data = data;
    ptr->left = NULL;
    ptr->right = NULL;
}

node_ptr insert_BT(node_ptr head, int item){
    node_ptr new_node = create_node(item);
    if(head == NULL)return new_node;
    node_ptr ptr,ptr_prev;
    ptr = head; ptr_prev = NULL;
    while(ptr != NULL){
        ptr_prev = ptr;
        if(ptr->data > item)ptr = ptr->left;
        else if(ptr->data < item) ptr = ptr->right;
        else{
            printf("Key %d already Exists! \n",item);
            return head;
        }
    }
    if(ptr_prev->data > item)ptr_prev->left = new_node;
    else ptr_prev->right = new_node;
    return head;
}
void recInOrder(node_ptr ptr){
    if(ptr != NULL){
        recInOrder(ptr->left);
        printf("%d ",ptr->data);
        recInOrder(ptr->right);
    }
}
void recPreOrder(node_ptr ptr){
    if(ptr != NULL){
        printf("%d ",ptr->data);
        recPreOrder(ptr->left);
        recPreOrder(ptr->right);
    }
}
void recPostOrder(node_ptr ptr){
    if(ptr != NULL){
        recPostOrder(ptr->left);
        recPostOrder(ptr->right);
        printf("%d ",ptr->data);
    }
}
int main(){
    printf("1. Insert an Element\n2.In Order\n3.Pre Order\n4.Post Order\n5.exit\
n");
    int opt;
    node_ptr head = NULL;
    while(1){
        printf("Enter Option:");scanf("%d",&opt);
        switch(opt){
        case 1:
            int new;
            printf("    Enter Element to Enter: ");scanf("%d",&new);
            head = insert_BT(head,new);
            break;
        case 2:
            recInOrder(head);printf("\n");
            break;
        case 3:
            recPreOrder(head);printf("\n");
            break;
        case 4:
            recPostOrder(head);printf("\n");
            break;

        default:
            return 0;
        }

    }

    node_ptr root;
    root = NULL;
    root = insert_BT(root,2);
    root = insert_BT(root,3);
    root = insert_BT(root,4);
    root = insert_BT(root,4);
    printf("RecInOrder: ");recInOrder(root);
    printf("\nRecPreOrder: ");recPreOrder(root);
    printf("\nRecPostOrder: ");recPostOrder(root);printf("\n");
    return 0;
}
```

Sample Input/Output:



```
student@lpcp-23:~/Desktop/230962326/Week7$ ./sq1
1. Insert an Element
2.In Order
3.Pre Order
4.Post Order
5.exit
Enter Option:1
	Enter Element to Enter: 1
Enter Option:1
	Enter Element to Enter: 2
Enter Option:1
	Enter Element to Enter: 3
Enter Option:1
	Enter Element to Enter: 4
Enter Option:1
	Enter Element to Enter: 5
Enter Option:1
	Enter Element to Enter: 6
Enter Option:2
1 2 3 4 5 6
Enter Option:3
1 2 3 4 5 6
Enter Option:4
6 5 4 3 2 1
Enter Option:5
student@lpcp-23:~/Desktop/230962326/Week7$
```

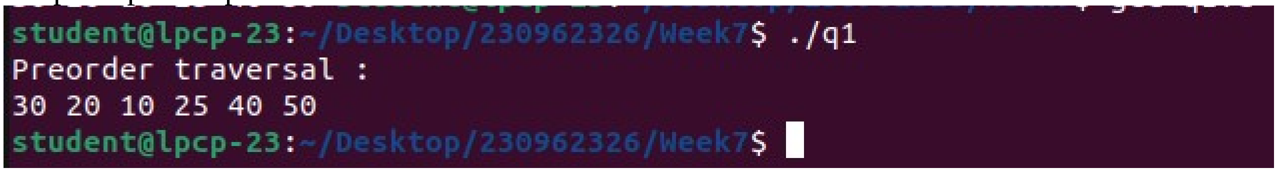
2) Write a program to create the AVL tree by iterative insertion.

Code:

```
#include<stdio.h>
#include<stdlib.h>

struct Node{
    int key;
    struct Node *left,*right;
    int height;
};
int height(struct Node *N){
    if (N == NULL)return 0;
    return N->height;
}

int max(int a, int b){
    return (a > b)? a : b;
}

struct Node* newNode(int key){
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key    = key;
    node->left   = NULL;
    node->right  = NULL;
    node->height = 1;
    return(node);
```

```
}

struct Node *rightRotate(struct Node *y){
    struct Node *x = y->left;
    struct Node *T2 = x->right;
    x->right = y;
    y->left = T2;

    y->height = max(height(y->left),height(y->right)) + 1;
    x->height = max(height(x->left),height(x->right)) + 1;

    return x;
}

struct Node *leftRotate(struct Node *x){
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(height(x->left),height(x->right)) + 1;
    y->height = max(height(y->left),height(y->right)) + 1;

    return y;
}

int getBalance(struct Node *N){
    if (N == NULL)return 0;
    return height(N->left) - height(N->right);
}

struct Node* insert(struct Node* node, int key){
    if (node == NULL)return(newNode(key));

    if (key < node->key)node->left  = insert(node->left, key);
    else if (key > node->key)node->right = insert(node->right, key);
    else return node;

    node->height = 1 + max(height(node->left),height(node->right));

    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)return rightRotate(node);
    if (balance < -1 && key > node->right->key)return leftRotate(node);
    if (balance > 1 && key > node->left->key){
        node->left =  leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key){
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}

void preOrder(struct Node *root){
    if(root != NULL){
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

int main(){
  struct Node *root = NULL;
  root = insert(root, 10);
  root = insert(root, 20);
  root = insert(root, 30);
  root = insert(root, 40);
  root = insert(root, 50);
  root = insert(root, 25);

  printf("Preorder traversal : \n");
  preOrder(root);
  return 0;
}
```

Sample Input/Output:



```
student@lpcp-23:~/Desktop/230962326/Week7$ ./q1
Preorder traversal :
30 20 10 25 40 50
student@lpcp-23:~/Desktop/230962326/Week7$
```

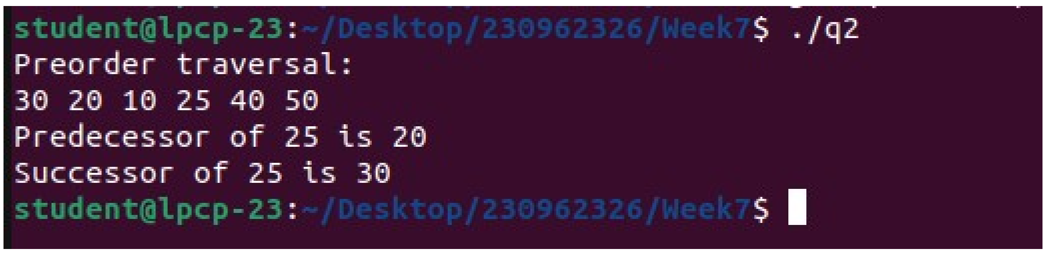
3) Using the AVL created in question 1, display the successor (next greater key) and predecessor (next smaller key) of a given key.

Code:

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int key;
    struct Node *left, *right;
    int height;
};

int height(struct Node *N) {
    if (N == NULL) return 0;
    return N->height;
}

int max(int a, int b) {
    return (a > b) ? a : b;
}

struct Node* newNode(int key) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return node;
}

struct Node *rightRotate(struct Node *y) {
    struct Node *x = y->left;
    struct Node *T2 = x->right;
    x->right = y;
    y->left = T2;

    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
    return x;
}

struct Node *leftRotate(struct Node *x) {
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    return y;
}

int getBalance(struct Node *N) {
    if (N == NULL) return 0;
    return height(N->left) - height(N->right);
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;

    node->height = 1 + max(height(node->left), height(node->right));

    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)return rightRotate(node);
    if (balance < -1 && key > node->right->key)return leftRotate(node);
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}

void preOrder(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)current = current->left;
    return current;
}
struct Node* maxValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->right != NULL)current = current->right;
    return current;
```

```
}
void findPreSuc(struct Node* root, struct Node** pre, struct Node** suc, int
key) {
    if (root == NULL) return;
    if (root->key == key) {
        if (root->left != NULL)*pre = maxValueNode(root->left);
        if (root->right != NULL)*suc = minValueNode(root->right);
        return;
    }
    if (root->key > key) {
        *suc = root;
        findPreSuc(root->left, pre, suc, key);
    } else {
        *pre = root;
        findPreSuc(root->right, pre, suc, key);
    }
}

int main() {
    struct Node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);

    printf("Preorder traversal: \n");
    preOrder(root);
    printf("\n");

    int key = 25;
    struct Node* pre = NULL, *suc = NULL;

    findPreSuc(root, &pre, &suc, key);

    if (pre != NULL)printf("Predecessor of %d is %d\n", key, pre->key);
    else printf("No predecessor found for %d\n", key);
    if(suc != NULL)printf("Successor of %d is %d\n", key, suc->key);
    else printf("No successor found for %d\n", key);

    return 0;
}
```

Sample Input/Output:



```
student@lpcp-23:~/Desktop/230962326/Week7$ ./q2
Preorder traversal:
30 20 10 25 40 50
Predecessor of 25 is 20
Successor of 25 is 30
student@lpcp-23:~/Desktop/230962326/Week7$
```