**Assignment No. 5**

**Name: Bhavin Ratansing Patil**

**Roll No.: 26 SEDA**

**Q.1 Create a Binary Search Tree and perform recursive and non-recursive, insert and search operations.**

**Binary Search Tree**

A binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

Binary search tree is a node-based binary tree data structure which has the following properties:

1. The left subtree of a node contains only nodes with keys lesser than the node’s key.

2. The right subtree of a node contains only nodes with keys greater than the node’s key.

3. The left and right subtree each must also be a binary search tree.

**Algorithm:**

**Insert:**

**Step 1**: Create a new BST node and assign values to it.

**Step 2**: insert (node, data)

     i) If root == NULL,

         return the new node to the calling function.

     ii) if root->data > data

         call the insert function with root=>right and assign the return value in root=>right.

        root->left = insert(root->left, data)

     iii) else

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

         Root->right = insert(root->right, data)

**Step 3**: Finally, return the original root pointer to the calling function.

**Search:**

**Step 1**: Repeat Steps 2,3 & 4 Until element Not find && Root! = NULL

**Step 2**: If item Equal to Root Data Then print message item present.

**Step 3**: Else If item Greater that Equal that Root Data Then Move Root to Right.

**Step 4**: Else Move Root to Left.

**Step 5**: Stop

**Recursive Traversal:**

**inorder(temp):**

**Step 1**: If temp != NULL

**Step 2**: inorder( temp->Left );

**Step 3**: Print temp->Data

**Step 4**: inorder( temp->Right );

**preorder(temp):**

**Step 1**: If temp != NULL

**Step 2**: Print temp->Data;

**Step 3**: preorder( temp->Left );

**Step 4**: preorder( temp->Right );

**postorder(temp):**

**Step 1**: If temp != NULL

**Step 2**: postorder( temp->Left );

**Step 3**: postorder( temp->Right );

**Step 4**: Print temp->Data;

**Non-Recursive Traversal:**

**inorder(temp):**

**Step 1:** while temp!=NULL

     i) push(temp),

ii) temp=temp->left

**Step 2:** while top! = -1

i) r = pop()

ii) print r->data

iii) r = r->right

iv) while r! = NULL

i) push(r)

ii) r = r->left

**preorder(temp):**

**Step 1:** while temp! =NULL

i) print temp->data,

     ii) push(temp),

iii) temp=temp->left

**Step 2:** while top=-1

i) r = pop(),

ii) r = r->right,

iv) while r! = NULL

i) print r->data,

ii) push(r),

ii) r = r->left

**Program:**

#include <stdio.h>

#include <malloc.h>

struct node \*st[100];

int top = -1;

// BST operation = create , insert ,traversing and search

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

} \* root;

// ---------------------------------------------

struct node \*insert(struct node \*temp, int data)

{

    struct node \*r;

    r = malloc(sizeof(struct node));

    r->data = data;

    r->left = r->right = NULL;

    if (temp == NULL)

    {

        return r;

    }

    if (temp->data > data)

    {

        temp->left = insert(temp->left, data);

    }

    else

    {

        temp->right = insert(temp->right, data);

    }

    return temp;

}

// ---------------------------------------------

struct node \*create(struct node \*temp)

{

    struct node \*r;

    int i, n, x;

    printf("\n How many nodes you want to insert: ");

    scanf("%d", &n);

    for (i = 0; i < n; i++)

    {

        printf("\nEnter Data for Node: ");

        scanf("%d", &x);

        temp = insert(temp, x);

    }

    return temp;

}

// ---------------------------------------------

struct node \*search(struct node \*temp, int data)

{

    if (temp == NULL)

    {

        printf("\nData is not present");

        return NULL;

    }

    if (temp->data == data)

    {

        printf("\nData is present\n");

        return temp;

    }

    if (temp->data > data)

    {

        return search(temp->left, data);

    }

    else

    {

        return search(temp->right, data);

    }

}

// ---------------------------------------------

void inorder(struct node \*temp)

{

    if (temp != NULL)

    {

        inorder(temp->left);

        printf("\t%d", temp->data);

        inorder(temp->right);

    }

}

void preorder(struct node \*temp)

{

    if (temp != NULL)

    {

        printf("\t%d", temp->data);

        preorder(temp->left);

        preorder(temp->right);

    }

}

void postorder(struct node \*temp)

{

    if (temp != NULL)

    {

        postorder(temp->left);

        postorder(temp->right);

        printf("\t%d", temp->data);

    }

}

// ---------------------------------------------

void push(struct node \*temp)

{

    st[++top] = temp;

}

struct node \*pop()

{

    return st[top--];

}

void inordernr(struct node \*temp)

{

    struct node \*r;

    while (temp != NULL)

    {

        push(temp);

        temp = temp->left;

    }

    while (top != -1)

    {

        r = pop();

        printf("\t%d", r->data);

        r = r->right;

        while (r != NULL)

        {

            push(r);

            r = r->left;

        }

    }

}

void preordernr(struct node \*temp)

{

    struct node \*r;

    while (temp != NULL)

    {

        printf("\t%d", temp->data);

        push(temp);

        temp = temp->left;

    }

    while (top != -1)

    {

        r = pop();

        r = r->right;

        while (r != NULL)

        {

            printf("\t%d", r->data);

            push(r);

            r = r->left;

        }

    }

}

void main()

{

    int ch, choice, ele, ins;

    struct node \*temp, \*temp1;

    root = NULL;

    do

    {

        printf("\n1)Create\n2)Insert\n3)Inorder(recursive)\n4)Preorder(recursive)\n5)Postorder(recursive)\n6)Inorder(Non-recursive)\n7)Preorder(Non-recursive)\n8)Search\n0)Quit\n\nEnter Your Choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            temp = root;

            root = create(root);

            temp = root;

            break;

        case 2:

            printf("Enter the data you want to insert: ");

            scanf("%d", &ins);

            insert(temp, ins);

            break;

        case 3:

            printf("\n=====Inorder (Recursive)=====\n");

            inorder(temp);

            printf("\n================================\n");

            break;

        case 4:

            printf("\n=====Preorder (Recursive)=====\n");

            preorder(temp);

            printf("\n================================\n");

            break;

        case 5:

            printf("\n=====Postorder (Recursive)=====\n");

            postorder(temp);

            printf("\n================================\n");

            break;

        case 6:

            printf("\n=====Inorder (Non - Recursive)=====\n");

            inordernr(temp);

            printf("\n================================\n");

            break;

        case 7:

            printf("\n=====Preorder (Non - Recursive)=====\n");

            preordernr(temp);

            printf("\n================================\n");

            break;

        case 8:

            printf("\nEnter the data do you want to search: ");

            scanf("%d", &ele);

            printf("\n=====Search Result=====\n");

            search(root, ele);

            printf("\n================================\n");

            break;

        default:

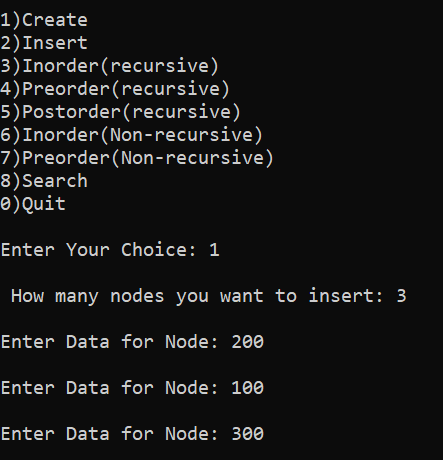
            break;

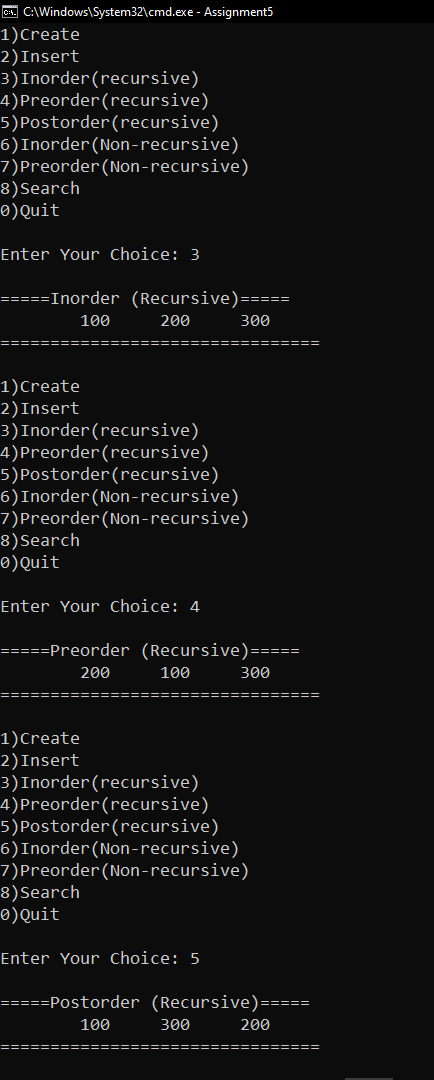
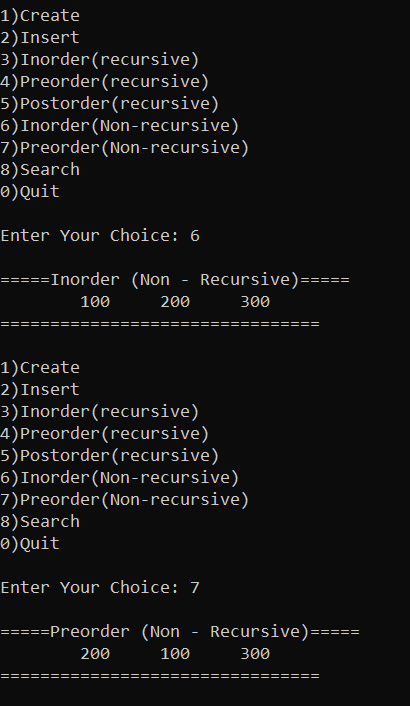
        }

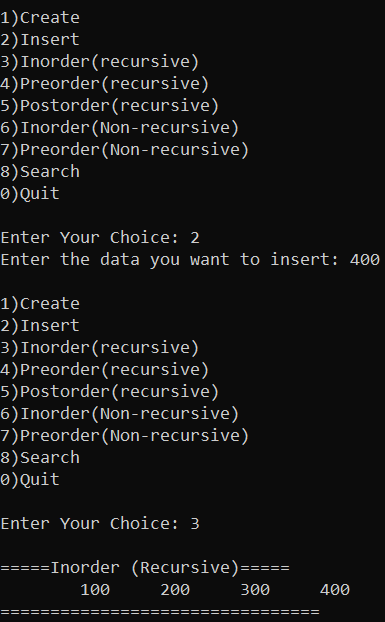
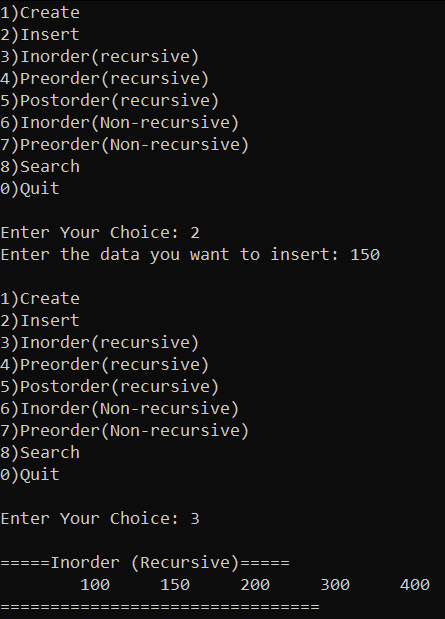
    } while (choice != 0);

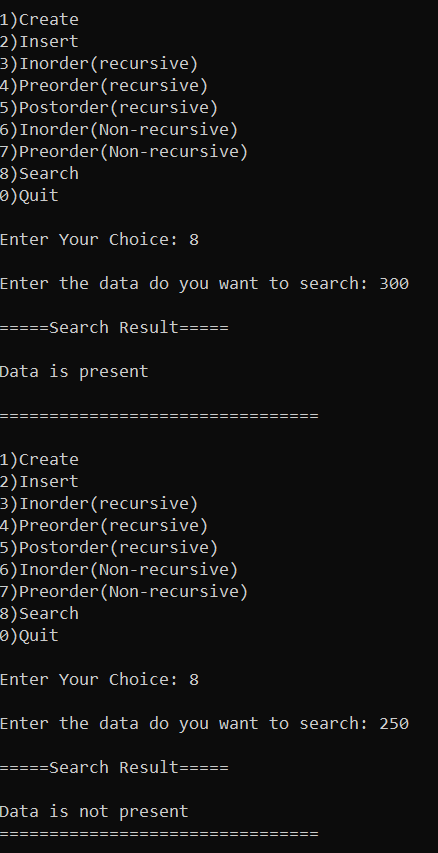
}

**Output:**

****

** **

** **

****