**BCSE202P -**

**Data Structures and Algorithms**

Digital Assignment – 3

**Name:** Dhruv Rajeshkumar Shah

**Registration No –** 21BCE0611

1. Write a program to perform the following operations:

a) Create a binary search tree

b) Insert an element into a binary search tree

c) Deletion of an element(all the options)

d) Sort the elements of the BST.

e) Search for an element in the BST

f) Display the leaf nodes alone.

f) Find the minimum and maximum element in the BST

g) Find the kth minimum and maximum element in the BST

**CODE**

// DA-3

// Dhruv Rajeshkumar Shah

// 21BCE0611

// Binary Search Tree

#include <stdio.h>

#include <stdlib.h>

// Declaring node

struct node

{

    int value;

    struct node \*left;

    struct node \*right;

};

struct node \*root = NULL;

// Function to create a new node

struct node \*createNode(int value)

{

    struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

    newNode->value = value;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

// Function to insert a node in BST

struct node \*insert(struct node \*root, int value)

{

    if (root == NULL)

    {

        root = createNode(value);

    }

    else if (value < root->value)

    {

        root->left = insert(root->left, value);

    }

    else

    {

        root->right = insert(root->right, value);

    }

    return root;

}

// Function to delete a node from BST

struct node \*delete (struct node \*root, int value)

{

    if (root == NULL)

    {

        printf("Element not found in BST");

    }

    else if (value < root->value)

    {

        root->left = delete (root->left, value);

    }

    else if (value > root->value)

    {

        root->right = delete (root->right, value);

    }

    else

    {

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

        {

            free(root);

            root = NULL;

        }

        else if (root->left == NULL)

        {

            struct node \*temp = root;

            root = root->right;

            free(temp);

        }

        else if (root->right == NULL)

        {

            struct node \*temp = root;

            root = root->left;

            free(temp);

        }

        else

        {

            struct node \*temp = root->right;

            while (temp->left != NULL)

            {

                temp = temp->left;

            }

            root->value = temp->value;

            root->right = delete (root->right, temp->value);

        }

    }

    return root;

}

// Function to sort the BST

void sort(struct node \*root)

{

    if (root != NULL)

    {

        sort(root->left);

        printf("%d ", root->value);

        sort(root->right);

    }

}

// Function to search a node in BST

struct node \*search(struct node \*root, int value)

{

    if (root == NULL)

    {

        printf("Element not found in BST");

    }

    else if (value < root->value)

    {

        root->left = search(root->left, value);

    }

    else if (value > root->value)

    {

        root->right = search(root->right, value);

    }

    else

    {

        printf("Element found in BST");

    }

    return root;

}

// Function to display leaf nodes

void displayLeafNodes(struct node \*root)

{

    if (root != NULL)

    {

        displayLeafNodes(root->left);

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

        {

            printf("%d ", root->value);

        }

        displayLeafNodes(root->right);

    }

}

// Function to display largest element in BST

void largestElement(struct node \*root)

{

    if (root != NULL)

    {

        while (root->right != NULL)

        {

            root = root->right;

        }

        printf("Largest element in BST is %d", root->value);

    }

}

// Function to display smallest element in BST

void smallestElement(struct node \*root)

{

    if (root != NULL)

    {

        while (root->left != NULL)

        {

            root = root->left;

        }

        printf("Smallest element in BST is %d", root->value);

    }

}

// Function to display kth largest element in BST

void kthLargestElement(struct node \*root, int k)

{

    if (root != NULL)

    {

        kthLargestElement(root->right, k);

        k--;

        if (k == 0)

        {

            printf("Kth largest element in BST is %d", root->value);

        }

        kthLargestElement(root->left, k);

    }

}

// Function to display kth smallest element in BST

void kthSmallestElement(struct node \*root, int k)

{

    if (root != NULL)

    {

        kthSmallestElement(root->left, k);

        k--;

        if (k == 0)

        {

            printf("Kth smallest element in BST is %d", root->value);

        }

        kthSmallestElement(root->right, k);

    }

}

// Main function

int main()

{

    int opt = 0;

    printf("Binary Search Tree\n");

    while (opt != 8)

    {

        printf("Choose an option:\n");

        printf("1. Insert\n");

        printf("2. Delete\n");

        printf("3. Sort\n");

        printf("4. Search\n");

        printf("5. Display leaf nodes\n");

        printf("6. Display largest and smallest element\n");

        printf("7. Display kth largest and smallest element\n");

        printf("8. Exit\n");

        scanf("%d", &opt);

        switch (opt)

        {

        case 1:

        {

            int value;

            printf("Enter value to insert: ");

            scanf("%d", &value);

            root = insert(root, value);

            break;

        }

        case 2:

        {

            int value;

            printf("Enter value to delete: ");

            scanf("%d", &value);

            root = delete (root, value);

            break;

        }

        case 3:

        {

            printf("Sorted BST: ");

            sort(root);

            printf("\n");

            break;

        }

        case 4:

        {

            int value;

            printf("Enter value to search: ");

            scanf("%d", &value);

            root = search(root, value);

            printf("\n");

            break;

        }

        case 5:

        {

            printf("Leaf nodes: ");

            displayLeafNodes(root);

            printf("\n");

            break;

        }

        case 6:

        {

            largestElement(root);

            printf("\n");

            smallestElement(root);

            printf("\n");

            break;

        }

        case 7:

        {

            int k;

            printf("Enter k: ");

            scanf("%d", &k);

            kthLargestElement(root, k);

            printf("\n");

            kthSmallestElement(root, k);

            printf("\n");

            break;

        }

        case 8:

        {

            printf("Exiting...");

            break;

        }

        default:

        {

            printf("Invalid option\n");

            break;

        }

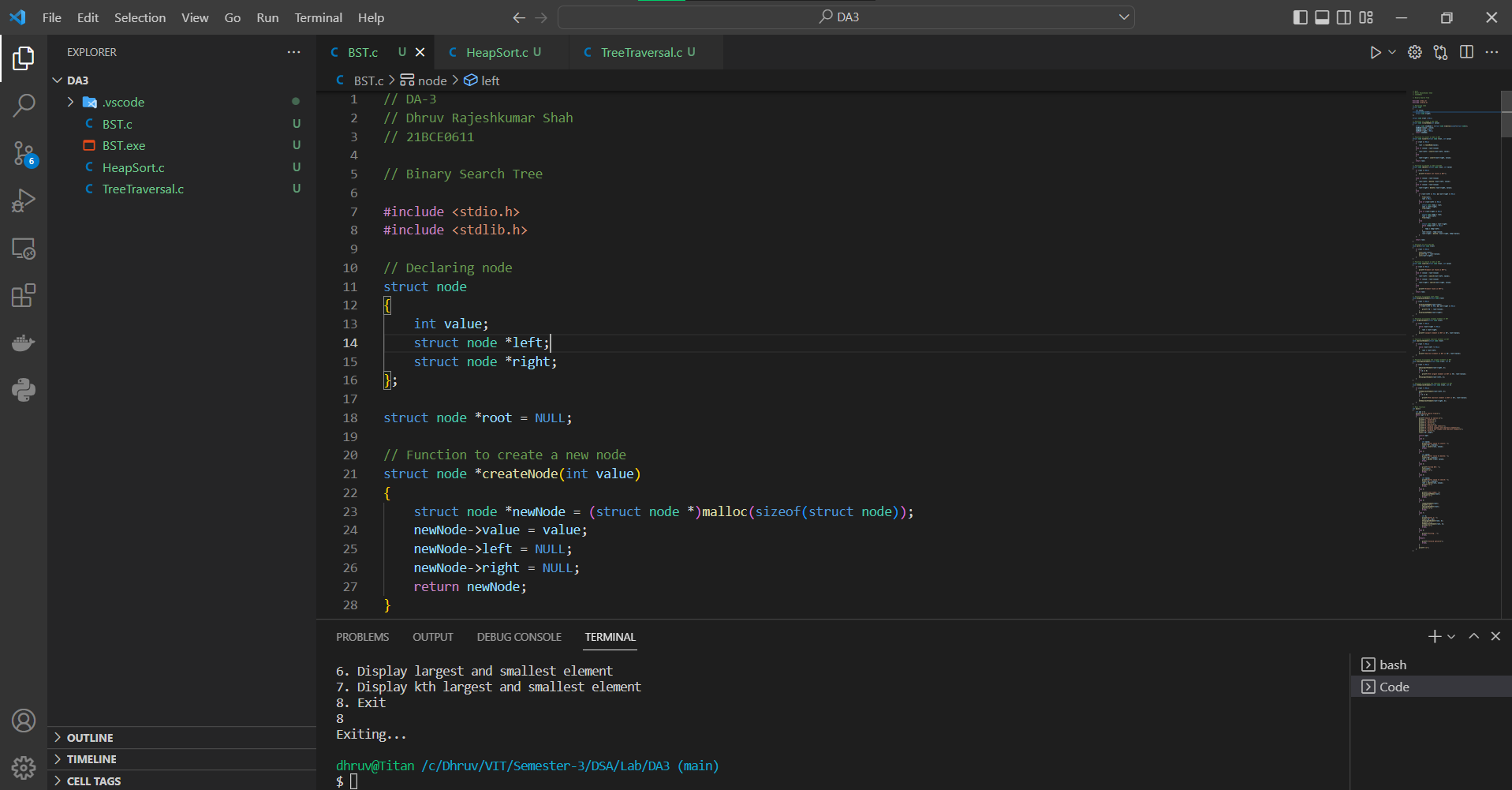
        }

        printf("\n");

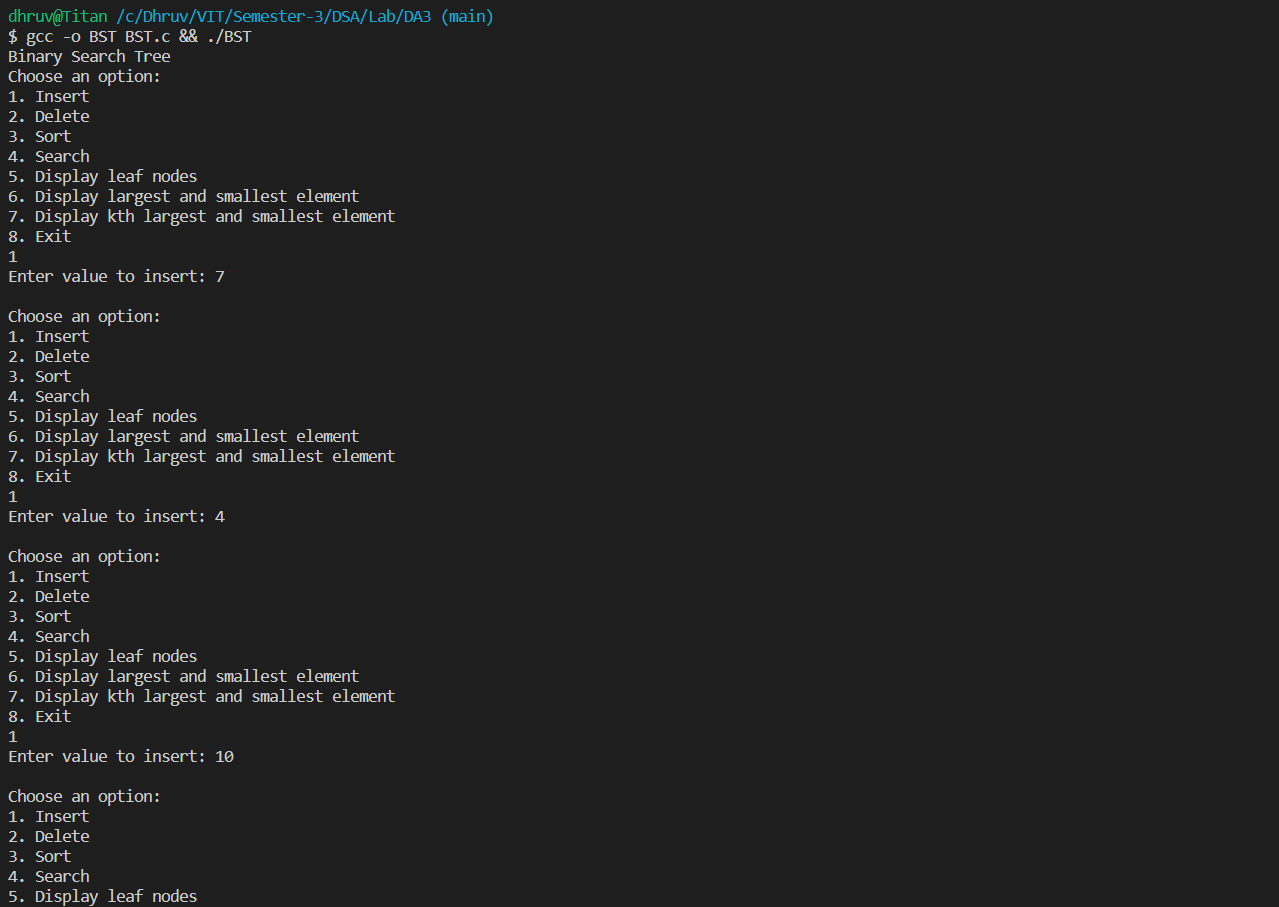
    }

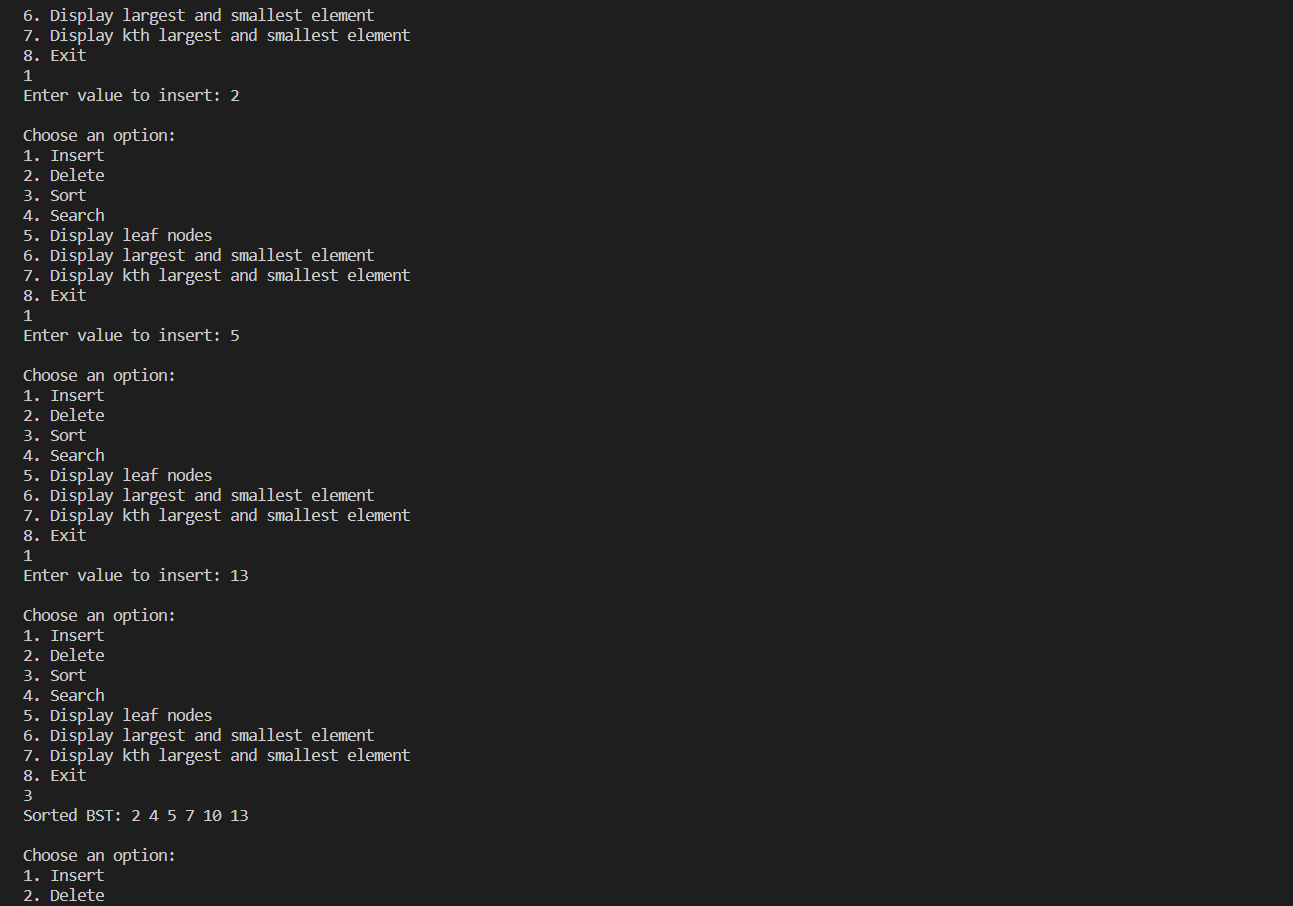
}

**SCREENSHOT**

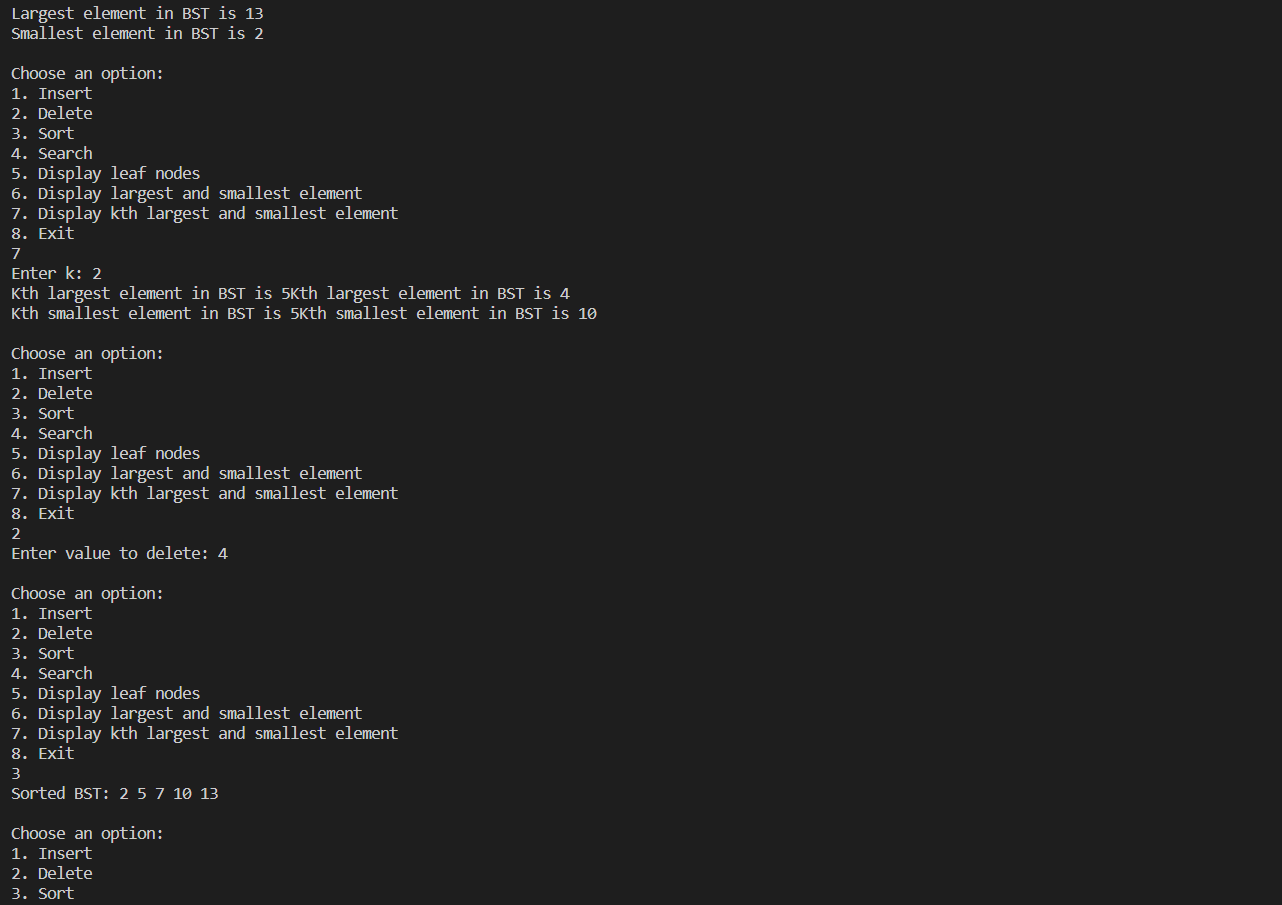
****

**OUTPUT**

****







2. Write a program to visit the nodes of a binary tree in all possible ways. Display the order of visiting the nodes.

**CODE**

// DA-3

// Dhruv Rajeshkumar Shah

// 21BCE0611

// Different types of tree traversals

#include <stdio.h>

#include <stdlib.h>

// Declaring node

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

};

struct node \*root = NULL;

// Function to create a new node

struct node \*createNode(int data)

{

    struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

    newNode->data = data;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

// Inorder Traversal

void inorder(struct node \*root)

{

    if (root != NULL)

    {

        inorder(root->left);

        printf("%d ", root->data);

        inorder(root->right);

    }

}

// Preorder Traversal

void preorder(struct node \*root)

{

    if (root != NULL)

    {

        printf("%d ", root->data);

        preorder(root->left);

        preorder(root->right);

    }

}

// Postorder Traversal

void postorder(struct node \*root)

{

    if (root != NULL)

    {

        postorder(root->left);

        postorder(root->right);

        printf("%d ", root->data);

    }

}

// Main

int main()

{

    root = createNode(10);

    root->left = createNode(5);

    root->right = createNode(15);

    root->left->left = createNode(2);

    root->left->right = createNode(7);

    root->right->left = createNode(12);

    root->right->right = createNode(17);

    printf("Inorder Traversal: ");

    inorder(root);

    printf("\n");

    printf("Preorder Traversal: ");

    preorder(root);

    printf("\n");

    printf("Postorder Traversal: ");

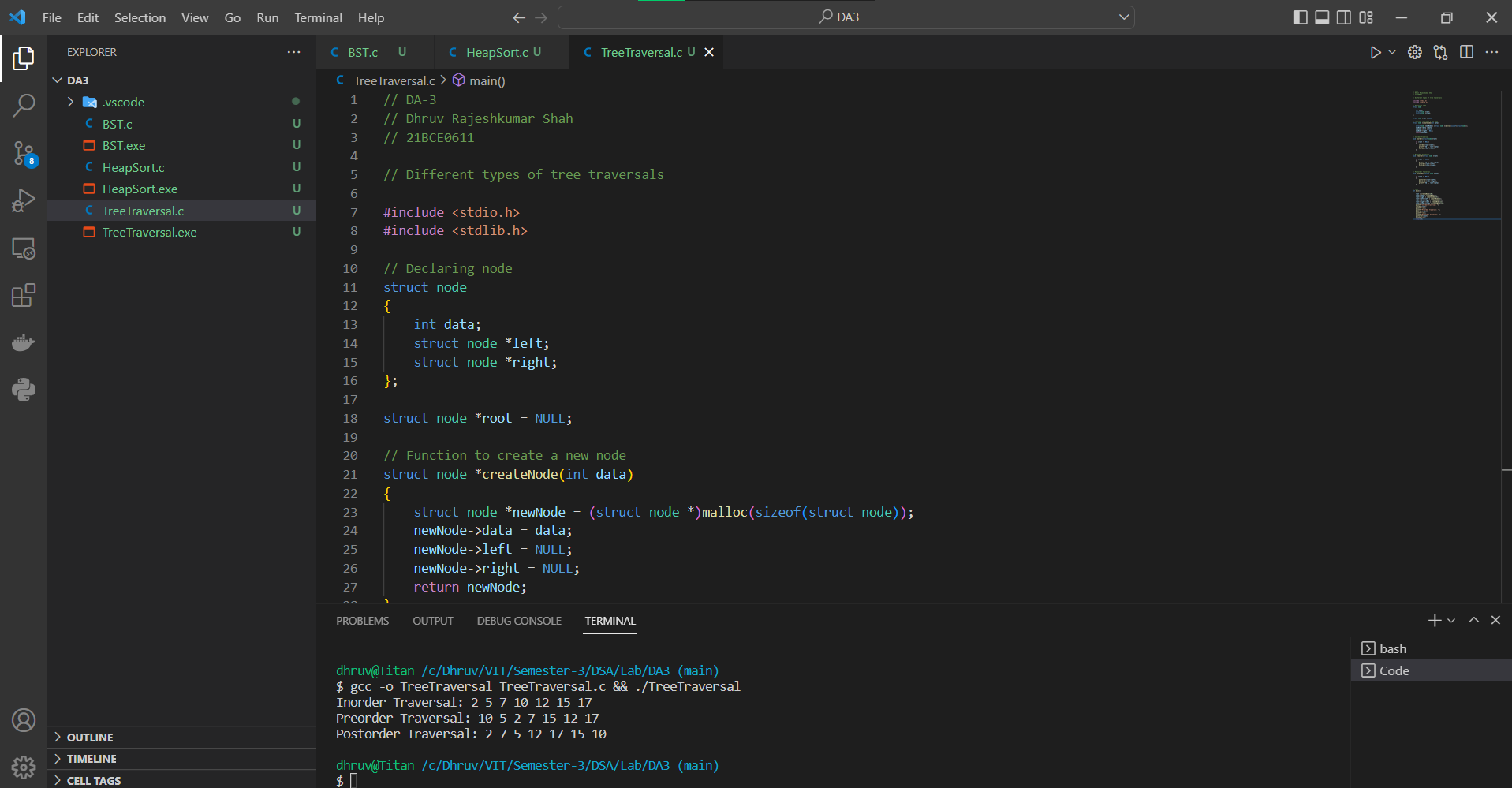
    postorder(root);

    printf("\n");

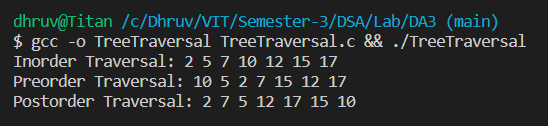
    return 0;

}

**SCREENSHOT**

****

**OUTPUT**

****

3. Given an array of elements, construct a min Heap and perform in-place sorting (ascending order) using heap sort.

**CODE**

// DA-3

// Dhruv Rajeshkumar Shah

// 21BCE0611

#include <stdio.h>

#include <stdlib.h>

// Swap

void swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

// Function to print the array

void printArray(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

// Function to heapify the tree using min heap

void heapify(int arr[], int n, int i)

{

    int smallest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < n && arr[left] < arr[smallest])

    {

        smallest = left;

    }

    if (right < n && arr[right] < arr[smallest])

    {

        smallest = right;

    }

    if (smallest != i)

    {

        swap(&arr[i], &arr[smallest]);

        heapify(arr, n, smallest);

    }

}

// Function to implement heap sort for ascending order

void heapSort(int arr[], int n, int sorted\_arr[])

{

    for (int i = n / 2 - 1; i >= 0; i--)

    {

        heapify(arr, n, i);

    }

    for (int i = n - 1; i >= 0; i--)

    {

        sorted\_arr[n - 1 - i] = arr[0];

        swap(&arr[0], &arr[i]);

        heapify(arr, i, 0);

    }

}

// Main

int main()

{

    int arr[] = {12, 11, 13, 5, 7, 6};

    int n = sizeof(arr) / sizeof(arr[0]);

    int sorted\_arr[n];

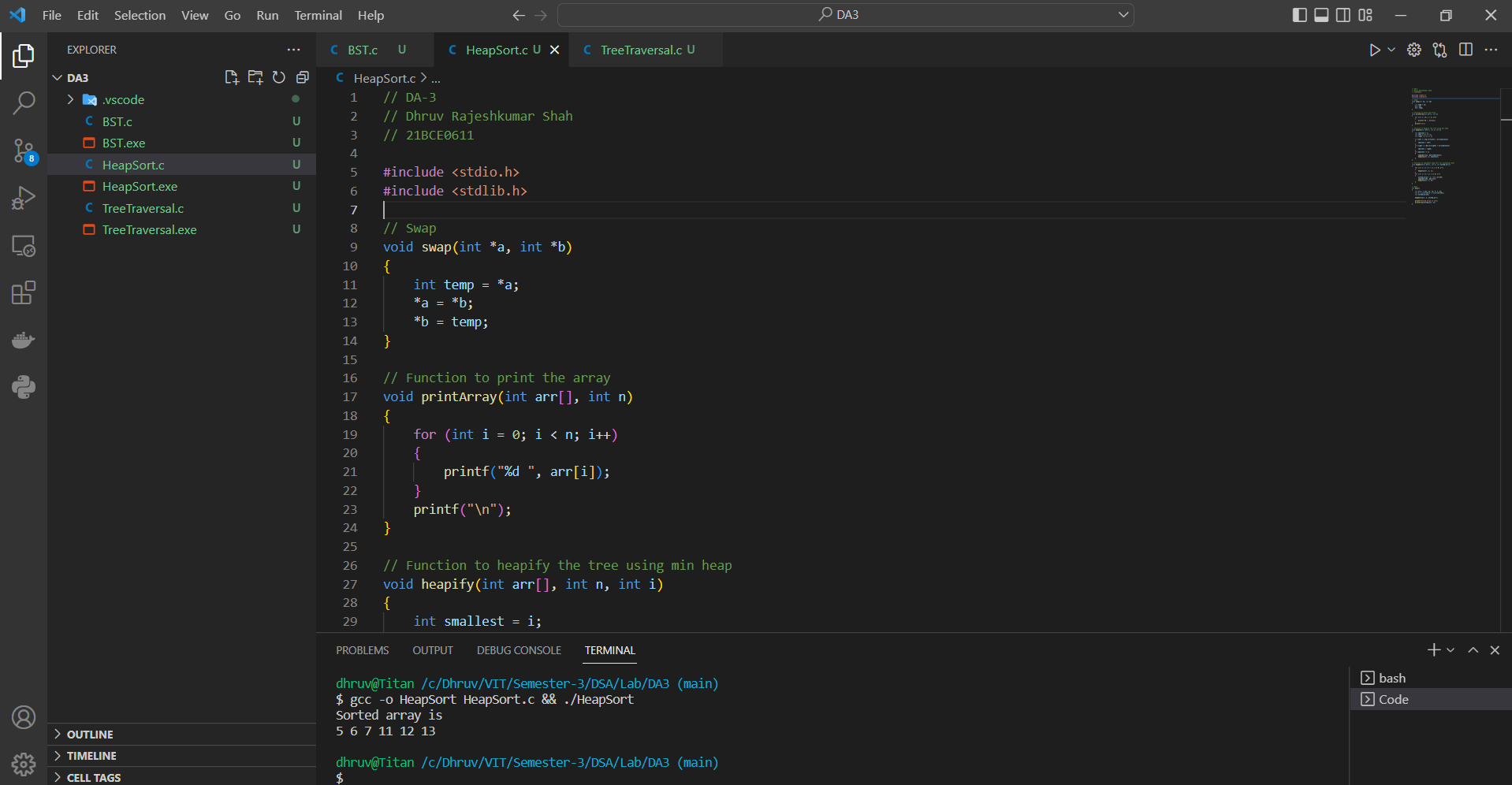
    heapSort(arr, n, sorted\_arr);

    printf("Sorted array is \n");

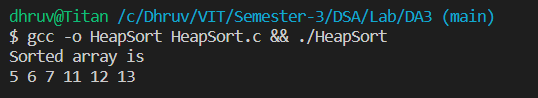
    printArray(sorted\_arr, n);

}

**SCREENSHOT**

****

**OUTPUT**

****