**Lab Assignment 8**

**Binary Search Trees and Heap**

Name- Ujjwal Pant Batch- 2C24 Rollno. 1024030370

1. Write a program using functions for binary tree traversals: Pre-order, In-order and Post order using a recursive approach.

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

void preorder(Node* root) {
    if (root == nullptr) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}

void inorder(Node* root) {
    if (root == nullptr) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

void postorder(Node* root) {
    if (root == nullptr) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);

    cout << "Pre-order traversal: ";
    preorder(root);
    cout << endl;

    cout << "In-order traversal: ";
    inorder(root);
    cout << endl;

    cout << "Post-order traversal: ";
    postorder(root);
    cout << endl;

    return 0;
}
```

```
Pre-order traversal: 1 2 4 5 3 6
In-order traversal: 4 2 5 1 3 6
Post-order traversal: 4 5 2 6 3 1


=== Code Execution Successful ===
```

2.Implement following functions for Binary Search Trees
(a) Search a given item (Recursive & Non-Recursive)
(b) Maximum element of the BST
(c) Minimum element of the BST
(d) In-order successor of a given node the BST
(e) In-order predecessor of a given node the BST

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

Node* insert(Node* root, int value) {
    if (root == nullptr) return new Node(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    return root;
}

bool searchRecursive(Node* root, int key) {
    if (root == nullptr) return false;
    if (key == root->data) return true;
    if (key < root->data)
        return searchRecursive(root->left, key);
    else
        return searchRecursive(root->right, key);
}

bool searchIterative(Node* root, int key) {
    while (root != nullptr) {
        if (key == root->data) return true;
        else if (key < root->data) root = root->left;
        else root = root->right;
    }
    return false;
}

Node* findMax(Node* root) {
    if (root == nullptr) return nullptr;
    while (root->right != nullptr)
        root = root->right;
    return root;
}

Node* findMin(Node* root) {
    if (root == nullptr) return nullptr;
    while (root->left != nullptr)
        root = root->left;
    return root;
}

Node* inorderSuccessor(Node* root, Node* n) {
    if (n->right != nullptr)
        return findMin(n->right);

    Node* succ = nullptr;
    while (root != nullptr) {
        if (n->data < root->data) {
            succ = root;
            root = root->left;
        } else if (n->data > root->data)
            root = root->right;
        else
            break;
    }
    return succ;
}

Node* inorderPredecessor(Node* root, Node* n) {
    if (n->left != nullptr)
        return findMax(n->left);

    Node* pred = nullptr;
    while (root != nullptr) {
        if (n->data > root->data) {
            pred = root;
            root = root->right;
        } else if (n->data < root->data)
            root = root->left;
        else
            break;
    }
    return pred;
}
```

```cpp
int main() {
    Node* root = nullptr;

    int values[] = {5, 3, 2, 4, 8, 9, 10, 16};
    for (int v : values)
        root = insert(root, v);

    cout << "In-order Traversal of BST:\n";
    // (Assumed present in screenshot)
    // inorderTraversal(root);

    int key = 40;
    cout << "Recursive search for " << key << ": ";
    cout << (searchRecursive(root, key) ? "Found" : "Not Found") << endl
    ;

    key = 4;
    cout << "Recursive search: " << (searchRecursive(root, key) ?
        "Found" : "Not Found") << endl;
    cout << "Iterative search: " << (searchIterative(root, key) ?
        "Found" : "Not Found") << endl;

    Node* maxNode = findMax(root);
    if (maxNode) cout << "Maximum element: " << maxNode->data << endl;

    Node* minNode = findMin(root);
    if (minNode) cout << "Minimum element: " << minNode->data << endl;

    Node* node = root->left->right;
    Node* succ = inorderSuccessor(root, node);
    if (succ)
        cout << "In-order Successor of " << node->data << ": " << succ
            ->data << endl;

    Node* pred = inorderPredecessor(root, node);
    if (pred)
        cout << "In-order Predecessor of " << node->data << ": " << pred
            ->data << endl;

    return 0;
}
```

```
In-order Traversal of BST:
Recursive search for 40: Not Found
Recursive search: Found
Iterative search: Found
Maximum element: 16
Minimum element: 2
In-order Successor of 4: 5
In-order Predecessor of 4: 3
```

3. Write a program for binary search tree (BST) having functions for the following operations:
   (a) Insert an element (no duplicates are allowed),
   (b) Delete an existing element,
   (c) Maximum depth of BST
   (d) Minimum depth of BST

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int v) {
        data = v;
        left = right = nullptr;
    }
};

class BST {
public:
    Node* insert(Node* root, int key) {
        if (root == nullptr) return new Node(key);
        if (key < root->data)
            root->left = insert(root->left, key);
        else if (key > root->data)
            root->right = insert(root->right, key);
        return root;
    }

    Node* findMinNode(Node* root) {
        while (root->left != nullptr)
            root = root->left;
        return root;
    }

    Node* deleteNode(Node* root, int key) {
        if (root == nullptr) return root;

        if (key < root->data)
            root->left = deleteNode(root->left, key);
        else if (key > root->data)
            root->right = deleteNode(root->right, key);
        else {
            if (!root->left) return root->right;
            else if (!root->right) return root->left;

            Node* t = findMinNode(root->right);
            root->data = t->data;
            root->right = deleteNode(root->right, t->data);
        }
        return root;
    }

    int maxDepth(Node* root) {
        if (root == nullptr) return 0;
        int leftDepth = maxDepth(root->left);
        int rightDepth = maxDepth(root->right);
        return 1 + max(leftDepth, rightDepth);
    }

    int minDepth(Node* root) {
        if (root == nullptr) return 0;
        if (root->left == nullptr) return 1 + minDepth(root->right);
        if (root->right == nullptr) return 1 + minDepth(root->left);
        return 1 + min(minDepth(root->left), minDepth(root->right));
    }
};

int main() {
    BST t;
    Node* root = nullptr;

    int arr[] = {8, 3, 1, 6, 4, 7, 10, 14, 13};
    for (int x : arr)
        root = t.insert(root, x);

    cout << "Max Depth: " << t.maxDepth(root) << endl;
    cout << "Min Depth: " << t.minDepth(root) << endl;

    root = t.deleteNode(root, 6);
    cout << "After deletion, Max Depth: " << t.maxDepth(root) << endl;

    return 0;
}
```

```
Max Depth: 4
Min Depth: 3
After deletion, Max Depth: 4

=== Code Execution Successful ===
```

4.Write a program to determine whether a given binary tree is a BST or not.

```cpp
#include <iostream>
#include <climits>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int v) {
        data = v;
        left = right = nullptr;
    }
};

bool isBST(Node* root, int minval, int maxval) {
    if (root == nullptr) return true;
    if (root->data <= minval || root->data >= maxval)
        return false;
    return isBST(root->left, minval, root->data) &&
           isBST(root->right, root->data, maxval);
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(20);
    root->left->left = new Node(2);
    root->left->right = new Node(8);

    if (isBST(root, INT_MIN, INT_MAX))
        cout << "It IS a BST";
    else
        cout << "It is NOT a BST";

    return 0;
}
```

```
It IS a BST


=== Code Exec
```

5.Implement heap sort (increasing/decreasing)

```
#include <iostream>
using namespace std;

void heapifyMax(int arr[], int n, int i) {
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;

    if (left < n && arr[left] > arr[largest]) largest = left;
    if (right < n && arr[right] > arr[largest]) largest = right;

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapifyMax(arr, n, largest);
    }
}

void heapifyMin(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]);
        heapifyMin(arr, n, smallest);
    }
}

void heapSortIncreasing(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        heapifyMax(arr, n, i);

    for (int i = n-1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapifyMax(arr, i, 0);
    }
}

void heapSortDecreasing(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        heapifyMin(arr, n, i);

    for (int i = n-1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapifyMin(arr, i, 0);
    }
}

int main() {
    int arr[] = {3, 5, 2, 9, 1, 8, 10};
    int n = 7;

    cout << "Sorted (Increasing): ";
    heapSortIncreasing(arr, n);
    for (int i = 0; i < n; i++) cout << arr[i] << " ";

    cout << "\nSorted (Decreasing): ";
    heapSortDecreasing(arr, n);
    for (int i = 0; i < n; i++) cout << arr[i] << " ";

    return 0;
}
```

```
Sorted (Increasing): 1 2 3 5 8 9 10
Sorted (Decreasing): 10 9 8 5 3 2 1

=== Code Execution Successful ===
```

6.Implement priority queues using heaps

```cpp
#include <iostream>
using namespace std;

class PriorityQueue {
public:
    int arr[100];
    int size;

    PriorityQueue() { size = 0; }

    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return 2*i + 1; }
    int right(int i) { return 2*i + 2; }

    void insert(int x) {
        arr[size] = x;
        int i = size;
        size++;

        while (i > 0 && arr[parent(i)] < arr[i]) {
            swap(arr[i], arr[parent(i)]);
            i = parent(i);
        }
    }

    int getMax() {
        if (size == 0) return -1;
        return arr[0];
    }

    void heapify(int i) {
        int largest = i;
        int l = left(i);
        int r = right(i);

        if (l < size && arr[l] > arr[largest]) largest = l;
        if (r < size && arr[r] > arr[largest]) largest = r;

        if (largest != i) {
            swap(arr[i], arr[largest]);
            heapify(largest);
        }
    }

    int extractMax() {
        if (size <= 0) return -1;
        if (size == 1) return arr[--size];

        int root = arr[0];
        arr[0] = arr[size - 1];
        size--;
        heapify(0);

        return root;
    }

    bool isEmpty() {
        return size == 0;
    }
};

int main() {
    PriorityQueue pq;

    pq.insert(40);
    pq.insert(10);
    pq.insert(30);
    pq.insert(50);
    pq.insert(60);

    cout << "Max element: " << pq.getMax() << endl;

    cout << "Extracting elements: ";
    while (!pq.isEmpty())
        cout << pq.extractMax() << " ";
    cout << endl;

    return 0;
}
```

```
Max element: 60
Extracting elements: 60 50 40 30 10


=== Code Execution Successful ===
```