# DSA Assignment 8

## Q1. Binary Tree Traversals (Recursive)

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

struct Node {
   int data;
   Node* left;
   Node* right;
};

Node* newNode(int val) {
   Node* n = new Node();
   n->data = val;
   n->left = n->right = NULL;
   return n;
}

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

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

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

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

   cout << "Preorder: “;
```

```
    preorder(root);
    cout << "\nInorder: ";
    inorder(root);
    cout << "\nPostorder: ";
    postorder(root);

    return 0;
}
```

**OUTPUT**

```
Preorder: 1 2 4 5 3
Inorder: 4 2 5 1 3
Postorder: 4 5 2 3 1 %
```

## Q2. BST Functions (Search, Min, Max, Successor, Predecessor)

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

struct Node {
    int data;
    Node* left;
    Node* right;
};

Node* newNode(int val) {
    Node* temp = new Node();
    temp->data = val;
    temp->left = temp->right = NULL;
    return temp;
}

Node* insert(Node* root, int val) {
    if (root == NULL) return newNode(val);
    if (val < root->data) root->left = insert(root->left, val);
    else if (val > root->data) root->right = insert(root->right, val);
    return root;
}

Node* searchRec(Node* root, int key) {
    if (root == NULL || root->data == key) return root;
    if (key < root->data) return searchRec(root->left, key);
    return searchRec(root->right, key);
}

Node* searchNonRec(Node* root, int key) {
    while (root != NULL) {
        if (root->data == key) return root;
        if (key < root->data) root = root->left;
        else root = root->right;
```

```
  }
  return NULL;
}

Node* minNode(Node* root) {
  while (root && root->left != NULL)
    root = root->left;
  return root;
}

Node* maxNode(Node* root) {
  while (root && root->right != NULL)
    root = root->right;
  return root;
}

Node* inorderSuccessor(Node* root, Node* key) {
  Node* succ = NULL;
  while (root != NULL) {
    if (key->data < root->data) {
      succ = root;
      root = root->left;
    } else root = root->right;
  }
  return succ;
}

Node* inorderPredecessor(Node* root, Node* key) {
  Node* pred = NULL;
  while (root != NULL) {
    if (key->data > root->data) {
      pred = root;
      root = root->right;
    } else root = root->left;
  }
  return pred;
}

int main() {
  Node* root = NULL;
  root = insert(root, 50);
  insert(root, 30);
  insert(root, 70);
  insert(root, 20);
  insert(root, 40);

  cout << "Searching 40 Recursively: "
    << (searchRec(root, 40) ? "Found" : "Not found") << endl;

  cout << "Searching 90 Non-Recursively: "
    << (searchNonRec(root, 90) ? "Found" : "Not found") << endl;
```

```
    cout << "Min element: " << minNode(root)->data << endl;
    cout << "Max element: " << maxNode(root)->data << endl;

    Node* key = searchRec(root, 30);
    cout << "Inorder Successor of 30: "
         << inorderSuccessor(root, key)->data << endl;

    cout << "Inorder Predecessor of 30: "
         << inorderPredecessor(root, key)->data << endl;

    return 0;
}
```

**OUTPUT**

```
Searching 40 Recursively: Found
Searching 90 Non-Recursively: Not found
Min element: 20
Max element: 70
Inorder Successor of 30: 40
Inorder Predecessor of 30: 20
```

## Q3. BST Insert, Delete, Min/Max Depth

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

struct Node {
    int data;
    Node* left;
    Node* right;
};

Node* newNode(int val) {
    Node* t = new Node();
    t->data = val;
    t->left = t->right = NULL;
    return t;
}

Node* insert(Node* root, int val) {
    if (root == NULL) return newNode(val);
    if (val < root->data) root->left = insert(root->left, val);
    else if (val > root->data) root->right = insert(root->right, val);
    return root;
}
```

```
Node* minNode(Node* root) {
  while (root->left != NULL)
    root = root->left;
  return root;
}

Node* deleteNode(Node* root, int key) {
  if (root == NULL) 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 == NULL) return root->right;
    if (root->right == NULL) return root->left;

    Node* temp = minNode(root->right);
    root->data = temp->data;
    root->right = deleteNode(root->right, temp->data);
  }
  return root;
}

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

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

int main() {
  Node* root = NULL;
  root = insert(root, 40);
  insert(root, 20);
  insert(root, 10);
  insert(root, 30);
  insert(root, 60);
  insert(root, 50);

  cout << "Max depth: " << maxDepth(root) << endl;
  cout << "Min depth: " << minDepth(root) << endl;

  cout << "Deleting 20...\n";
  root = deleteNode(root, 20);

  cout << "Max depth after deletion: " << maxDepth(root) << endl;
```

```
    cout << "Min depth after deletion: " << minDepth(root) << endl;

    return 0;
}
```

**OUTPUT**

```
Max depth: 3
Min depth: 3
Deleting 20...
Max depth after deletion: 3
Min depth after deletion: 3
```

## Q4. Check if Binary Tree is BST

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

struct Node {
    int data;
    Node* left;
    Node* right;
};

Node* newNode(int val) {
    Node* n = new Node();
    n->data = val;
    n->left = n->right = NULL;
    return n;
}

bool isBSTUtil(Node* root, int min, int max) {
    if (root == NULL) return true;
    if (root->data <= min || root->data >= max) return false;
    return isBSTUtil(root->left, min, root->data) &&
           isBSTUtil(root->right, root->data, max);
}

bool isBST(Node* root) {
    return isBSTUtil(root, INT_MIN, INT_MAX);
}

int main() {
    Node* root = newNode(10);
    root->left = newNode(5);
    root->right = newNode(20);
    root->left->left = newNode(2);
    root->left->right = newNode(25);
```

```
    if (isBST(root)) cout << "Tree is a BST";
    else cout << "Tree is NOT a BST";

    return 0;
}
```

**OUTPUT**

```
Tree is NOT a BST
```