Practical 4 :

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

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

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

    friend class BST;
};

// Binary Search Tree class
class BST {
private:
    Node* root;

    // Recursive insert
    Node* insert(Node* n, int value) {
        if (!n)
            return new Node(value);
        if (value < n->data)
            n->left = insert(n->left, value);
        else
            n->right = insert(n->right, value);
        return n;
    }

    // Recursive in-order traversal
    void inorder(Node* n) {
        if (!n) return;
        inorder(n->left);
        cout << n->data << " ";
        inorder(n->right);
    }
```

```
    // Recursive height (longest path)
    int longestPath(Node* n) {
        if (!n) return 0;
        return max(longestPath(n->left), longestPath(n->right)) + 1;
    }

    // Recursive min value
    int findMinValue(Node* n) {
        if (!n) {
            cout << "Tree is empty!" << endl;
            return -1;
        }
        while (n->left)
            n = n->left;
        return n->data;
    }

    // Recursive swap
    void swapChildren(Node* n) {
        if (!n) return;
        swap(n->left, n->right);
        swapChildren(n->left);
        swapChildren(n->right);
    }

    // Recursive search
    bool search(Node* n, int value) {
        if (!n) return false;
        if (n->data == value) return true;
        return value < n->data ? search(n->left, value) : search(n->right, value);
    }

public:
    BST() {
        root = nullptr;
    }

    void insertValue(int value) {
        root = insert(root, value);
    }

    void display() {
        inorder(root);
        cout << endl;
```

```
    }

    int getLongestPath() {
        return longestPath(root);
    }

    int getMinValue() {
        return findMinValue(root);
    }

    void swapChildrenFromRoot() {
        swapChildren(root);
    }

    bool searchFromRoot(int value) {
        return search(root, value);
    }
};

// Main function
int main() {
    BST tree;
    int n, value;

    cout << "Enter number of nodes to insert into BST: ";
    cin >> n;

    cout << "Enter values: ";
    for (int i = 0; i < n; i++) {
        cin >> value;
        tree.insertValue(value);
    }

    cout << "\nBST (In-order Traversal): ";
    tree.display();

    cout << "Height of the BST (Longest Path from Root): " << tree.getLongestPath() << endl;
    cout << "Minimum Value in BST: " << tree.getMinValue() << endl;

    tree.swapChildrenFromRoot();
    cout << "BST after swapping left and right children: ";
    tree.display();

    cout << "Enter value to search: ";
```

```
    cin >> value;
    if (tree.searchFromRoot(value))
        cout << "Value " << value << " found in the tree." << endl;
    else
        cout << "Value " << value << " not found in the tree." << endl;

    return 0;
}
```