Program....

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

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

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

void insert(Node*& root, Node* temp) {
    if (root == nullptr) {
        root = temp;
        return;
    }

    Node* parent = nullptr;
    Node* current = root;

    while (current != nullptr) {
        parent = current;
        if (temp->data < current->data) {
            current = current->left;
        } else {
            current = current->right;
        }
    }

    if (temp->data < parent->data) {
        parent->left = temp;
    } else {
        parent->right = temp;
    }
}

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 << " ";
}

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

Node* smallest(Node* root) {
    if (root == nullptr) {
        return nullptr;
    }
    if (root->left == nullptr) {
        return root;
    }
    return smallest(root->left);
}

int longest_path(Node* root) {
    if (root == nullptr) {
        return 0;
    }
    int L = longest_path(root->left);
    int R = longest_path(root->right);
    return (L > R ? L : R) + 1;
}

void swapNodes(Node* root) {
    if (root == nullptr) {
        return;
    }
    Node* temp = root->left;
```

```
    root->left = root->right;
    root->right = temp;
    swapNodes(root->left);
    swapNodes(root->right);
}

int main() {
    Node* root = nullptr;
    int choice, num;
    while (true) {
        cout << "\nBinary Search Tree Menu:";
        cout << "\n1. Insert Node:";
        cout << "\n2. Preorder Traversal:";
        cout << "\n3. Inorder Traversal:";
        cout << "\n4. Postorder Traversal:";
        cout << "\n5. Search Node:";
        cout << "\n6. Smallest:";
        cout << "\n7. Longest Path:";
        cout << "\n8. Swapping:";
        cout << "\n9. Exit:";
        cout << "\nEnter your choice:";
        cin >> choice;

        switch (choice) {
            case 1: {
                cout << "Enter a number to insert: ";
                cin >> num;
                Node* temp = new Node(num);
                if (root == nullptr) {
                    root = temp;
                } else {
                    insert(root, temp);
                }
                cout << "Node inserted" << endl;
                break;
            }
            case 2: {
                if (root == nullptr)
                    cout << "Tree is empty!\n";
                else {
                    cout << "Preorder Traversal: ";
                    preorder(root);
                    cout << endl;
                }
                break;
            }
            case 3: {
                if (root == nullptr)
                    cout << "Tree is empty!\n";
                else {
                    cout << "Inorder Traversal: ";
                    inorder(root);
```

```
            cout << endl;
        }
        break;
    }
    case 4: {
        if (root == nullptr)
            cout << "Tree is empty!\n";
        else {
            cout << "Postorder Traversal: ";
            postorder(root);
            cout << endl;
        }
        break;
    }
    case 5: {
        int data;
        cout << "Enter value to search: ";
        cin >> data;
        Node* foundNode = search(root, data);
        if (foundNode) {
            cout << "Node " << data << " found in the tree\n";
        } else {
            cout << "Node is not found\n";
        }
        break;
    }
    case 6: {
        if (root == nullptr) {
            cout << "Tree is empty!\n";
        } else {
            Node* smallestNode = smallest(root);
            cout << "Smallest value in the tree is: " << smallestNode->data << endl;
        }
        break;
    }
    case 7: {
        cout << "Longest path is: " << longest_path(root) << endl;
        break;
    }
    case 8: {
        swapNodes(root);
        cout << "Postorder traversal after swapping: ";
        postorder(root);
        cout << endl;
        break;
    }
    case 9: {
        cout << "Exiting program" << endl;
        return 0;
    }
    default: {
        cout << "Invalid choice, try again.\n";
```

```
                break;
            }
        }
    }
    return 0;
}
```