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

class BST {
	int data;
	BST* left, * right;

public:
	BST();

	BST(int);

	BST* Insert(BST*, int);

	void Inorder(BST*);

	void Preorder(BST*);

	void Postorder(BST*);

	void printLevelOrder(BST* root);

	void printCurrentLevel(BST* root, int level);

	int height(BST* node);

	void mirror(BST* node);

	BST* search(BST* root, int key);

	void swapSubtrees(BST* root, int key);
};

// Default Constructor definition.
BST::BST()
	: data(0)
	, left(NULL)
	, right(NULL)
{
}

// Parameterized Constructor definition.
BST::BST(int value)
{
        data = value;
        left = right = NULL;
}

// Insert function definition.
BST* BST::Insert(BST* root, int value)
{
        if (!root) {
                // Insert the first node, if root is NULL.
                return new BST(value);
        }

        if (value > root->data) {

                root->right = Insert(root->right, value);
        }
        else {

                root->left = Insert(root->left, value);
        }

        return root;
}

void BST::Inorder(BST* root)
{
        if (!root) {
                return;
        }
        Inorder(root->left);
        cout << root->data << endl;
        Inorder(root->right);
}

void BST::Preorder(BST* root)
{
        if (!root) {
                return;
        }
        cout << root->data << endl;
        Inorder(root->left);
        Inorder(root->right);
}
```

```
// Postorder traversal function.
void BST::Postorder(BST* root)
{
        if (!root) {
                return;
        }
        Inorder(root->left);
        Inorder(root->right);
        cout << root->data << endl;
}
int BST::height(BST* node){
        if (node == NULL)
                return 0;

        else {

                int lheight = height(node->left);
                int rheight = height(node->right);

                if (lheight > rheight) {
                        return (lheight + 1);
                }
                else {
                        return (rheight + 1);
                }
        }
}


void BST::printLevelOrder(BST* root)
{
        int h = height(root);
        int i;
        for (i = 1; i <= h; i++)
                printCurrentLevel(root, i);
}

void BST::printCurrentLevel(BST* root, int level)
{
        if (root == NULL)
                return;
        if (level == 1)
                cout << root->data << " ";
        else if (level > 1) {
                printCurrentLevel(root->left, level - 1);
                printCurrentLevel(root->right, level - 1);
        }
}

void BST::mirror(BST* node)
{
        if (node == NULL)
                return;
        else
        {
                BST* temp;

                mirror(node->left);
                mirror(node->right);

                temp = node->left;
                node->left = node->right;
                node->right = temp;
        }
}

// Helper function
BST* BST::search(BST* root, int key)
{
        if (root == NULL || root->data == key)
                return root;

        if (root->data < key)
                return search(root->right, key);

        return search(root->left, key);
}

void BST::swapSubtrees(BST* root, int key)
{
        BST* node = search(root, key);
        if (node == NULL)
                return;

        mirror(node);
}

// Driver code
```

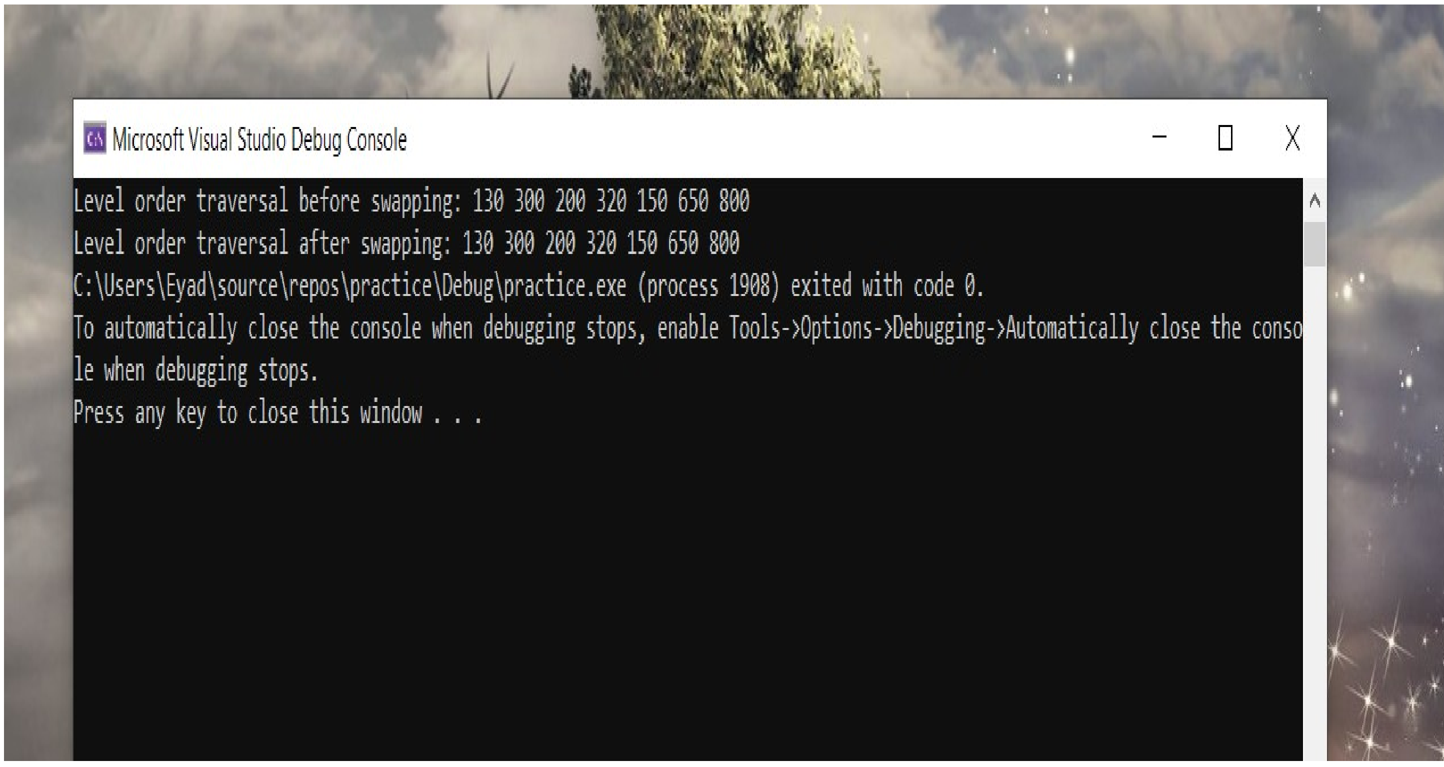
```
int main()
{
	BST lab, * root = NULL;
	root = lab.Insert(root, 130);
	lab.Insert(root, 300);
	lab.Insert(root, 200);
	lab.Insert(root, 320);
	lab.Insert(root, 150);
	lab.Insert(root, 650);
	lab.Insert(root, 800);

	cout << "Level order traversal before swapping: ";
	lab.printLevelOrder(root);

	lab.swapSubtrees(root, 50);

	cout << "\nLevel order traversal after swapping: ";
	lab.printLevelOrder(root);

	return 0;
}
```



```
Level order traversal before swapping: 130 300 200 320 150 650 800
Level order traversal after swapping: 130 300 200 320 150 650 800
C:\Users\Eyad\source\repos\practice\Debug\practice.exe (process 1908) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```