AMRITA SCHOOL OF COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS
(23CSE211)

Name: P JEEVAN SANDEEP

Roll No.: CH.SC.U4CSE24134

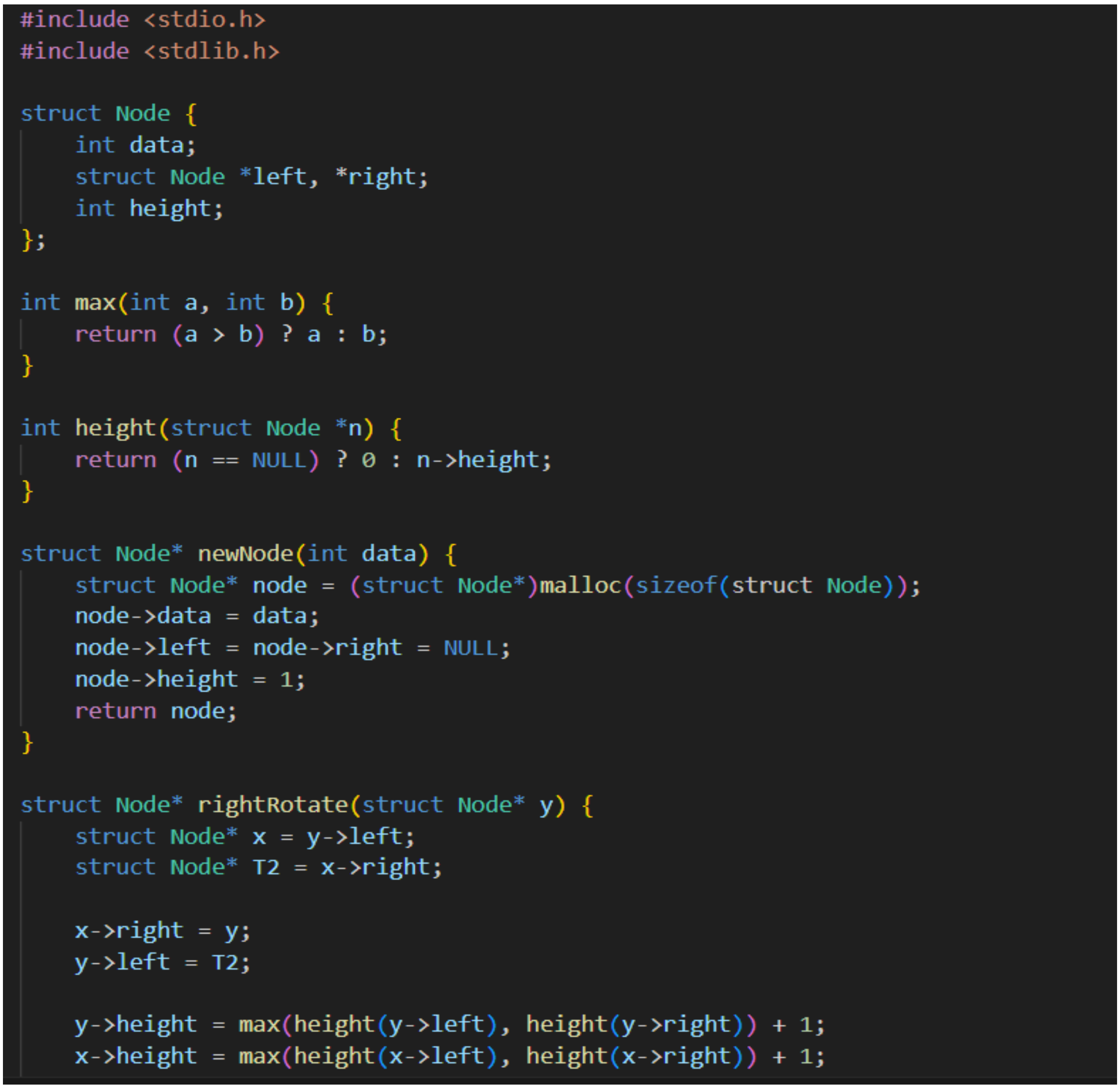
Class: BTech (CSE-B)

School: Amrita School of Computing,
Chennai Campus.

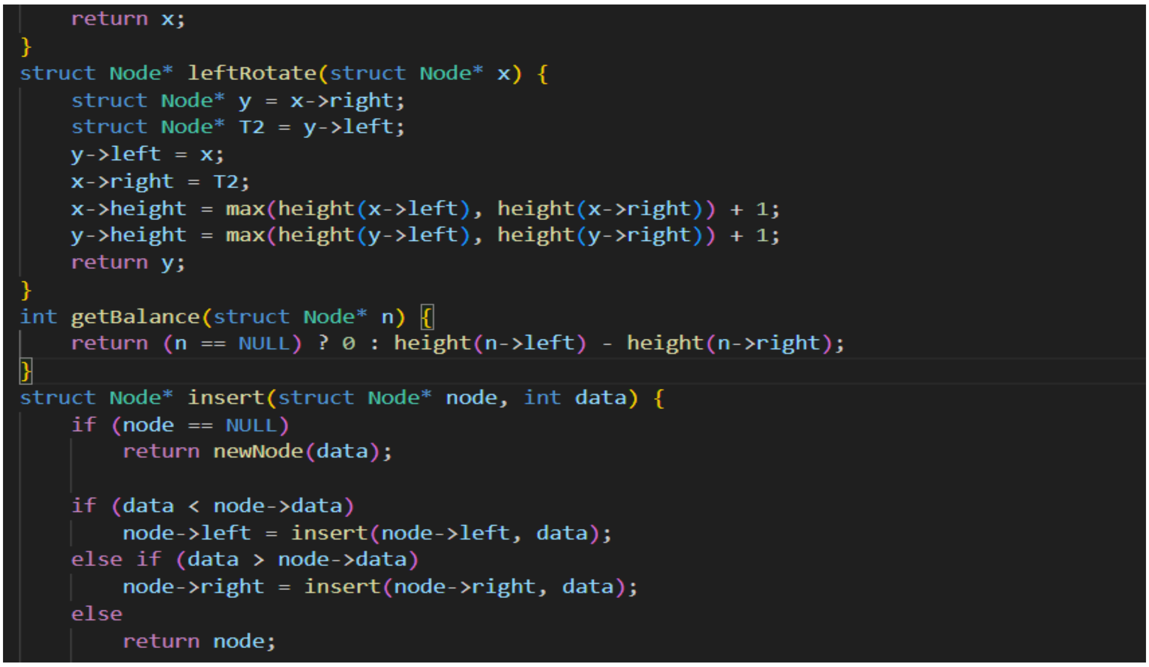
LAB-5

1) Solving an array of integers using AVL-Tree Method.

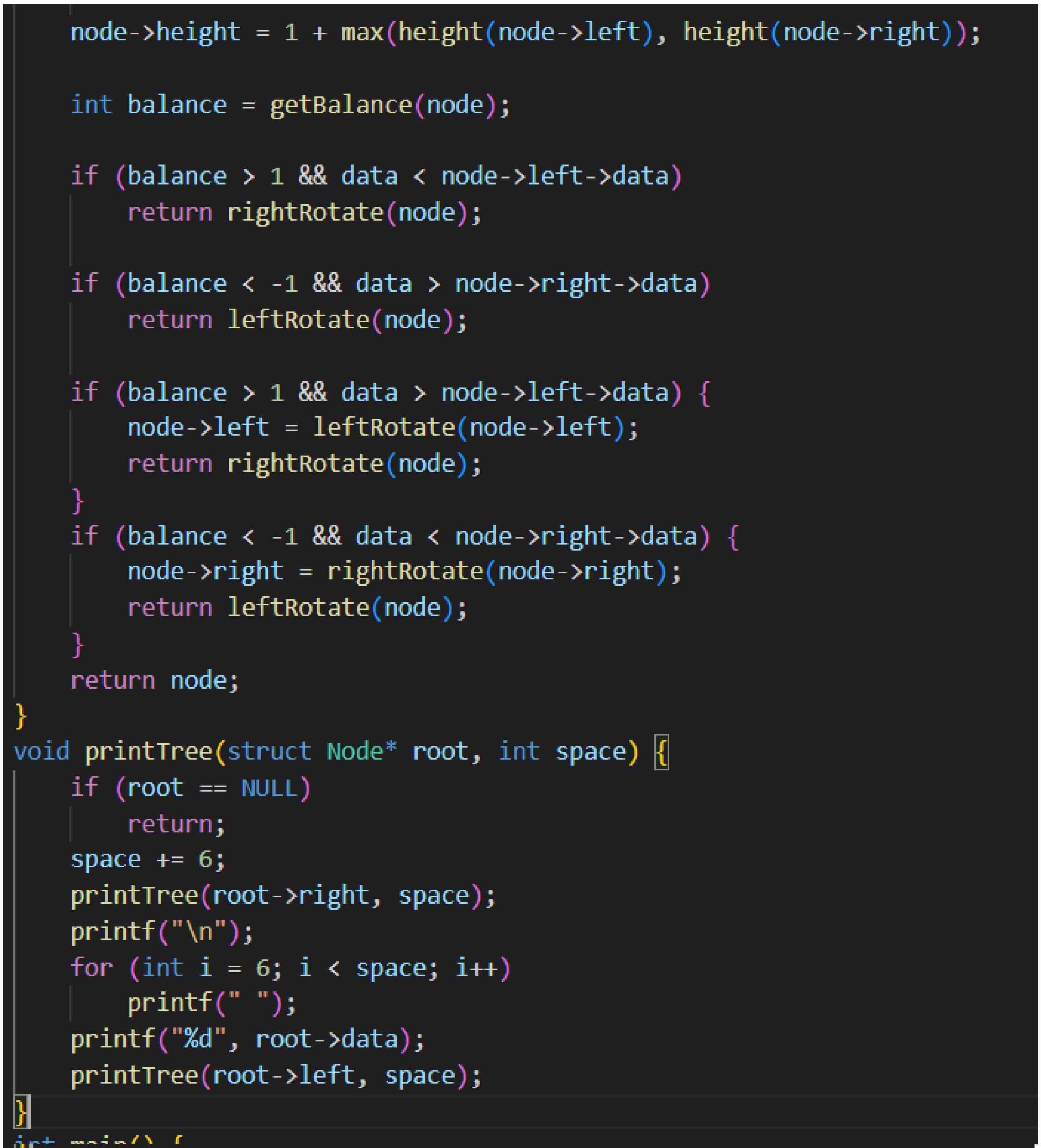
Code:



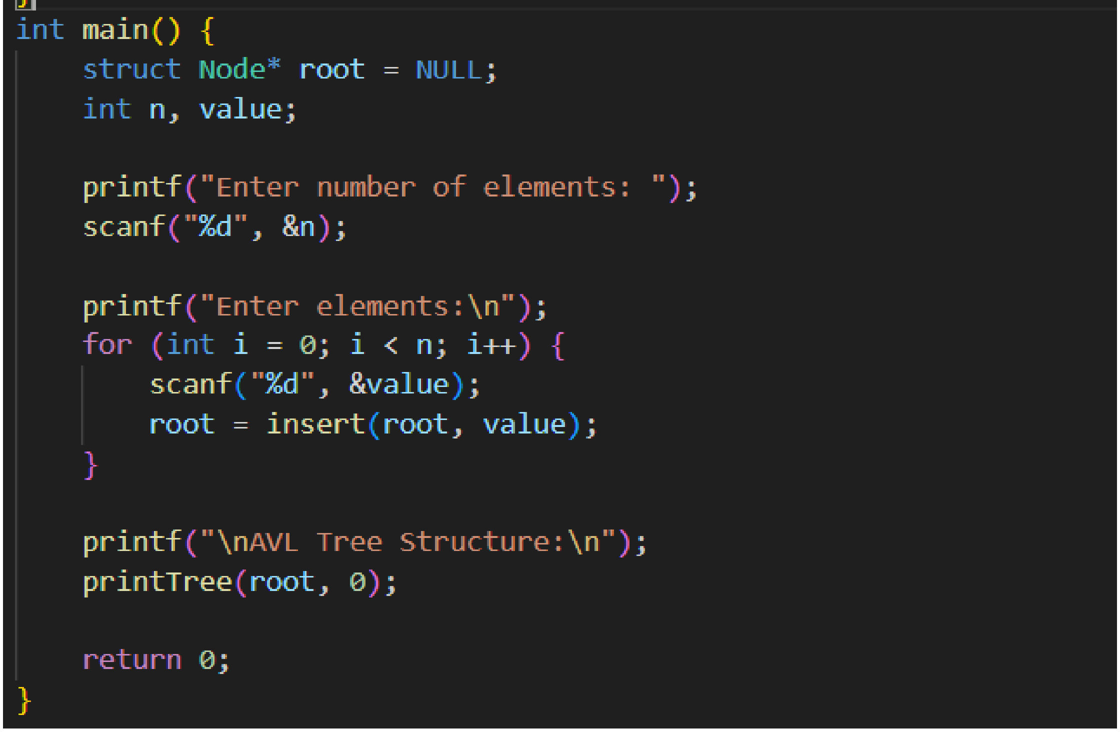
```c
#include <stdio.h>
#include <stdlib.h>

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

int max(int a, int b) {
    return (a > b) ? a : b;
}

int height(struct Node *n) {
    return (n == NULL) ? 0 : n->height;
}

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    node->height = 1;
    return node;
}

struct Node* rightRotate(struct Node* y) {
    struct Node* x = y->left;
    struct Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
```



```c
    return x;
}
struct Node* leftRotate(struct Node* x) {
    struct Node* y = x->right;
    struct Node* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}
int getBalance(struct Node* n) {
    return (n == NULL) ? 0 : height(n->left) - height(n->right);
}
struct Node* insert(struct Node* node, int data) {
    if (node == NULL)
        return newNode(data);

    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
    else
        return node;
```

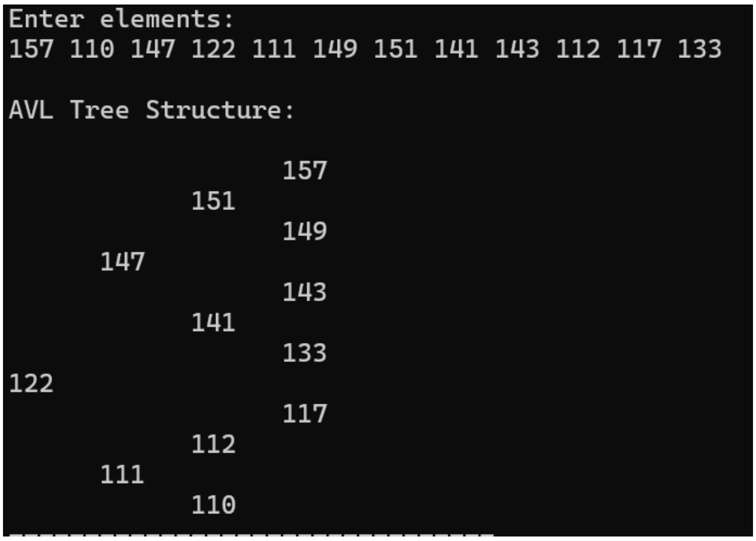


```
    node->height = 1 + max(height(node->left), height(node->right));

    int balance = getBalance(node);

    if (balance > 1 && data < node->left->data)
        return rightRotate(node);

    if (balance < -1 && data > node->right->data)
        return leftRotate(node);

    if (balance > 1 && data > node->left->data) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && data < node->right->data) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
void printTree(struct Node* root, int space) {
    if (root == NULL)
        return;
    space += 6;
    printTree(root->right, space);
    printf("\n");
    for (int i = 6; i < space; i++)
        printf(" ");
    printf("%d", root->data);
    printTree(root->left, space);
}
```



```
int main() {
    struct Node* root = NULL;
    int n, value;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    printf("\nAVL Tree Structure:\n");
    printTree(root, 0);

    return 0;
}
```

Output:



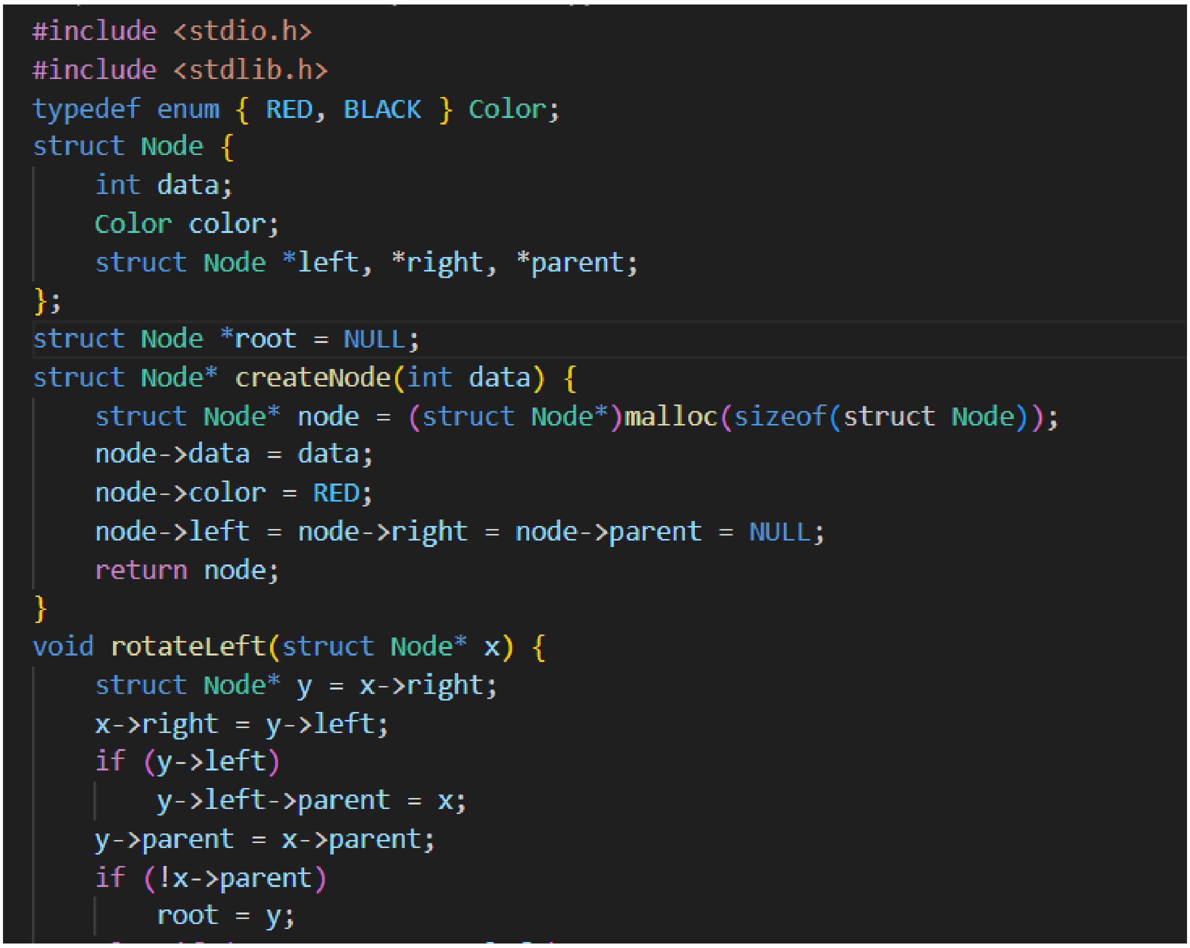
```
Enter elements:
157 110 147 122 111 149 151 141 143 112 117 133

AVL Tree Structure:

                  157
            151
                  149
      147
                  143
            141
                  133
122
                  117
            112
      111
            110
--------------------------------
```

Space Complexity: O (n)

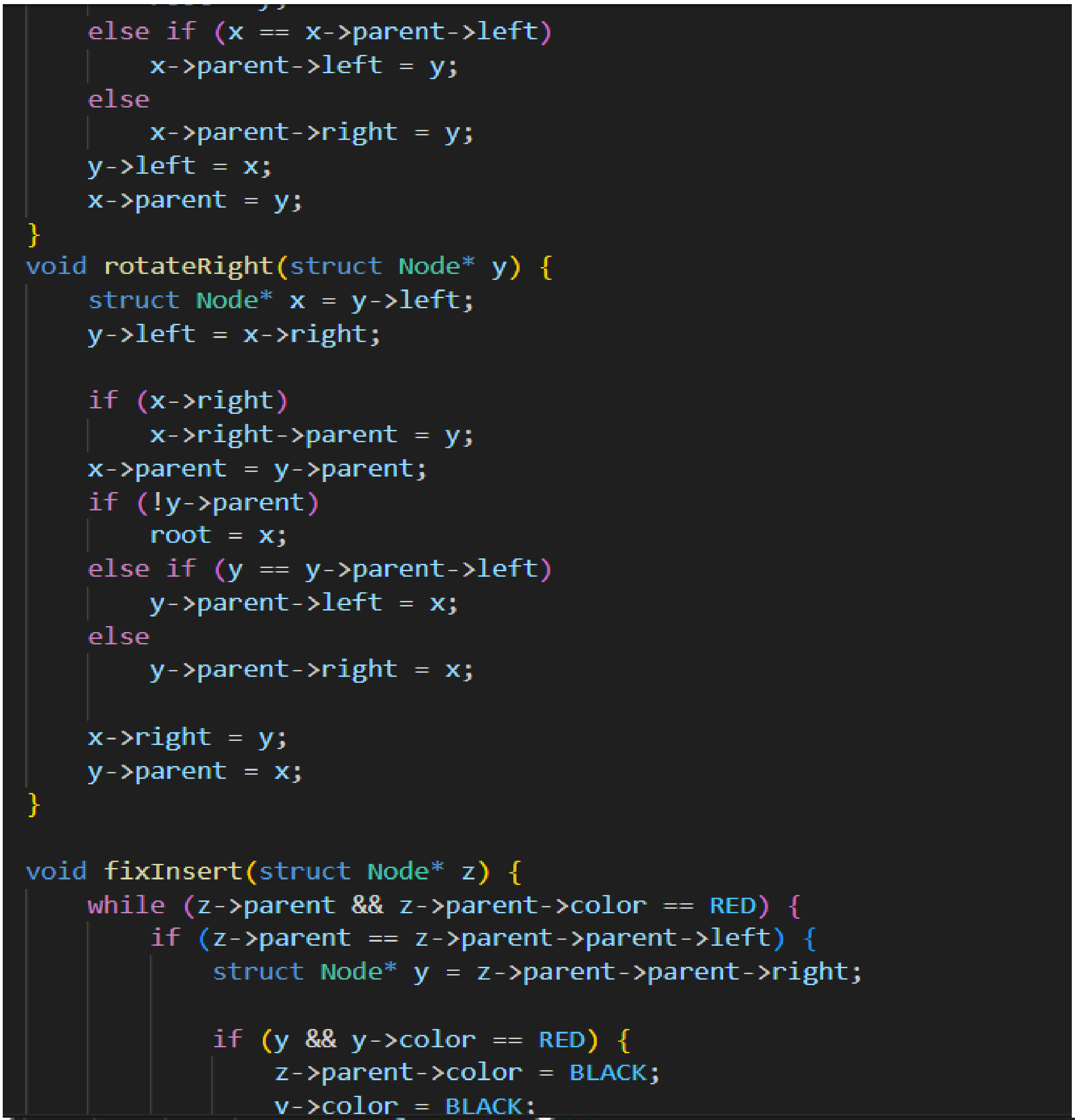
Time Complexity: O (n log n)

2) Solving an array of integers using Red-Black Algorithm.

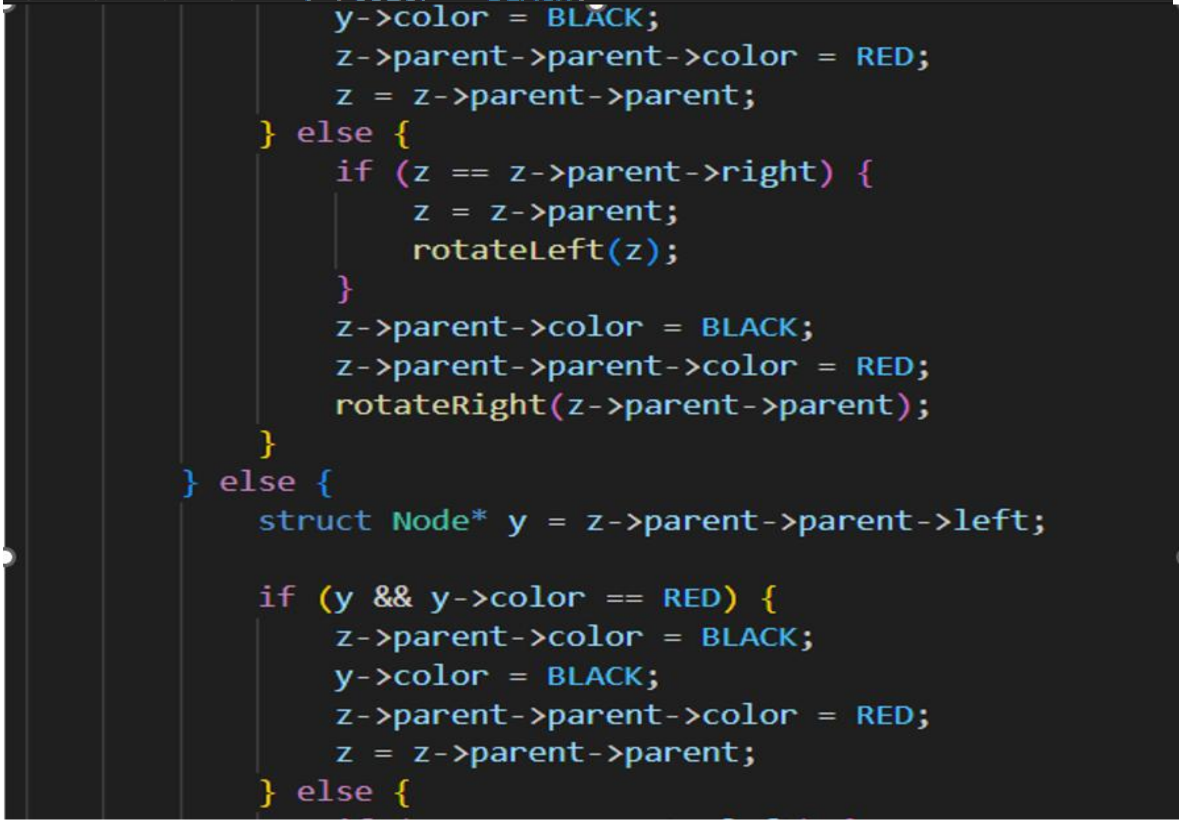
Code:



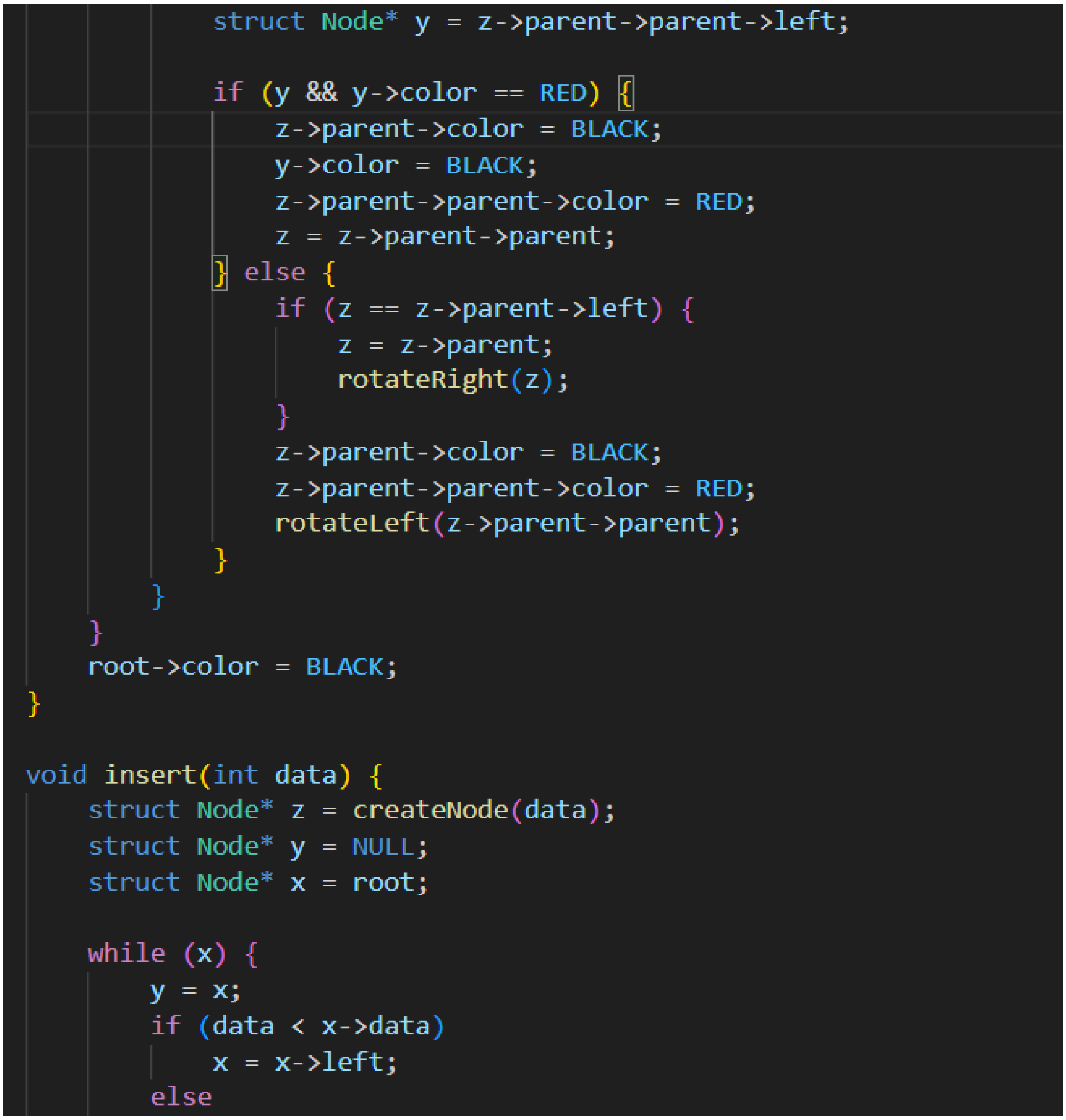
```c
#include <stdio.h>
#include <stdlib.h>
typedef enum { RED, BLACK } Color;
struct Node {
    int data;
    Color color;
    struct Node *left, *right, *parent;
};
struct Node *root = NULL;
struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->color = RED;
    node->left = node->right = node->parent = NULL;
    return node;
}
void rotateLeft(struct Node* x) {
    struct Node* y = x->right;
    x->right = y->left;
    if (y->left)
        y->left->parent = x;
    y->parent = x->parent;
    if (!x->parent)
        root = y;
```



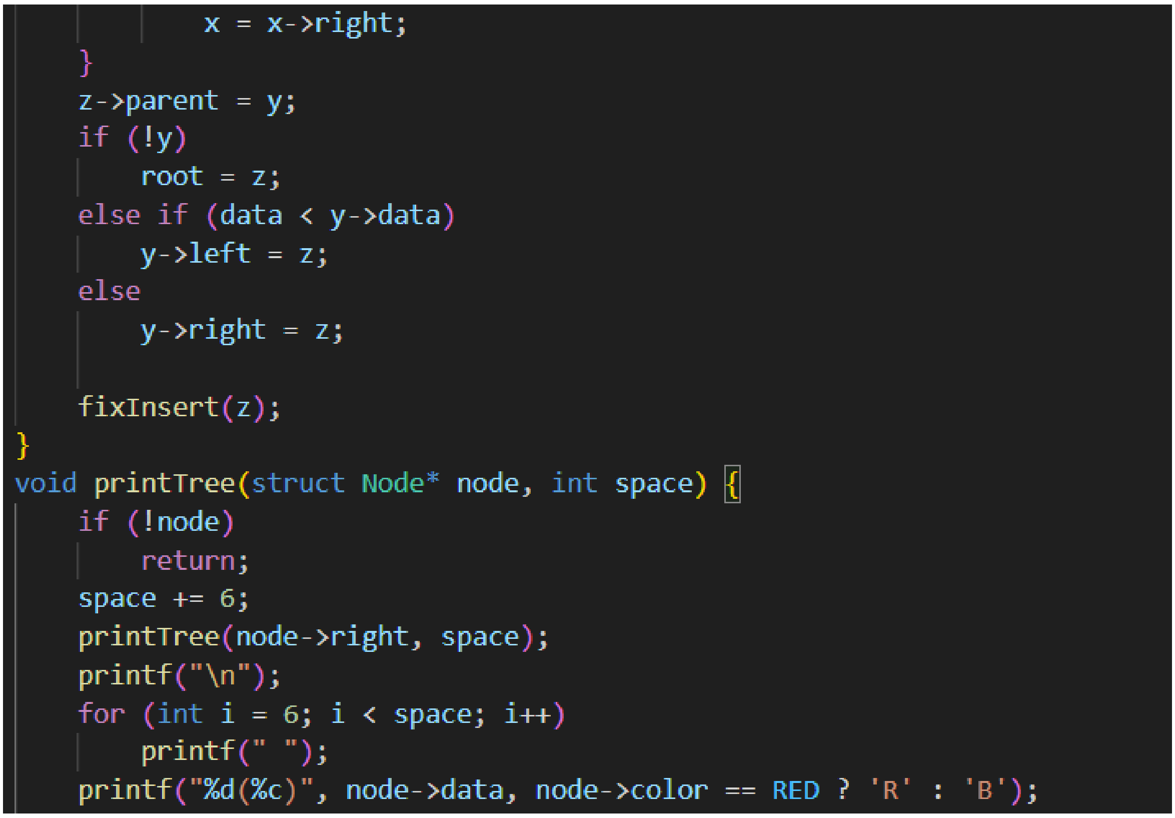
```
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}
void rotateRight(struct Node* y) {
    struct Node* x = y->left;
    y->left = x->right;

    if (x->right)
        x->right->parent = y;
    x->parent = y->parent;
    if (!y->parent)
        root = x;
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;

    x->right = y;
    y->parent = x;
}

void fixInsert(struct Node* z) {
    while (z->parent && z->parent->color == RED) {
        if (z->parent == z->parent->parent->left) {
            struct Node* y = z->parent->parent->right;

            if (y && y->color == RED) {
                z->parent->color = BLACK;
```



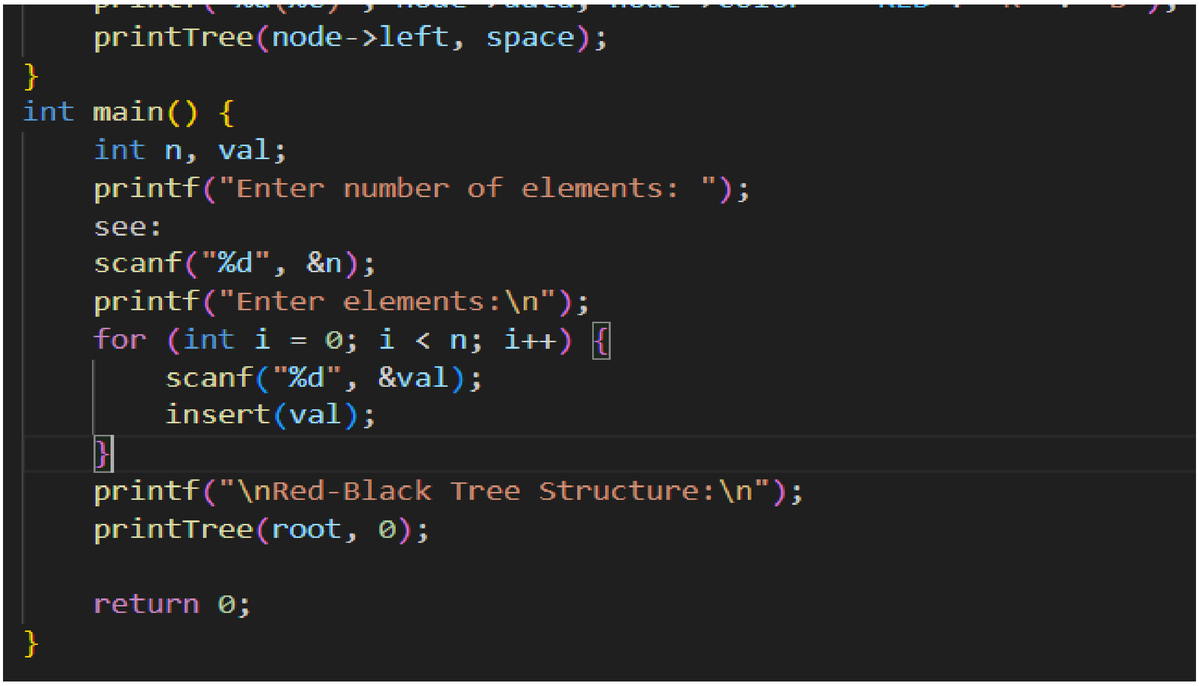
```
                y->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            } else {
                if (z == z->parent->right) {
                    z = z->parent;
                    rotateLeft(z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                rotateRight(z->parent->parent);
            }
        } else {
            struct Node* y = z->parent->parent->left;

            if (y && y->color == RED) {
                z->parent->color = BLACK;
                y->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            } else {
```



```c
            struct Node* y = z->parent->parent->left;

            if (y && y->color == RED) {
                z->parent->color = BLACK;
                y->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            } else {
                if (z == z->parent->left) {
                    z = z->parent;
                    rotateRight(z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                rotateLeft(z->parent->parent);
            }
        }
    }
    root->color = BLACK;
}

void insert(int data) {
    struct Node* z = createNode(data);
    struct Node* y = NULL;
    struct Node* x = root;

    while (x) {
        y = x;
        if (data < x->data)
            x = x->left;
        else
```

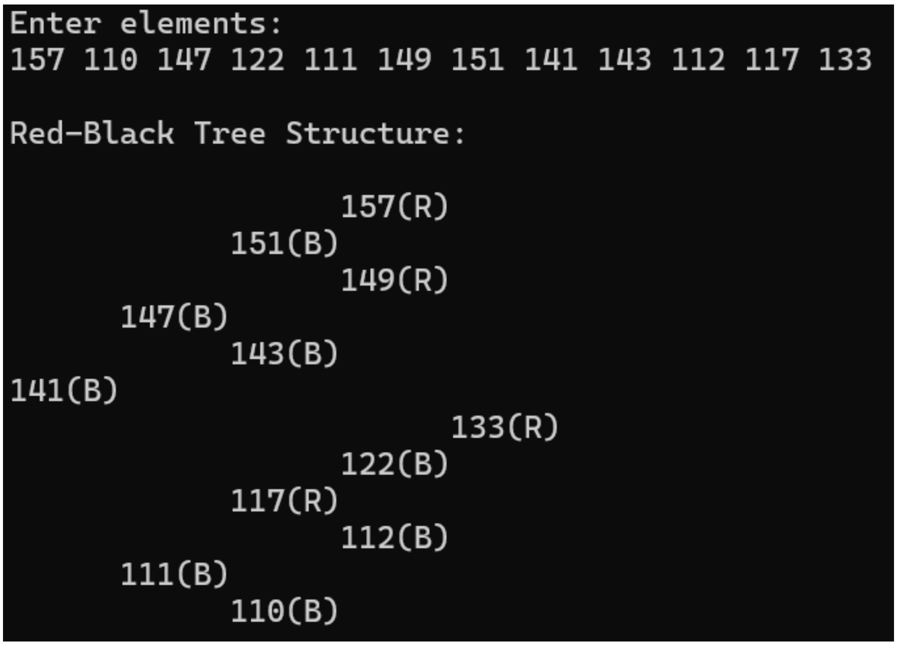


```c
            x = x->right;
    }
    z->parent = y;
    if (!y)
        root = z;
    else if (data < y->data)
        y->left = z;
    else
        y->right = z;

    fixInsert(z);
}
void printTree(struct Node* node, int space) {
    if (!node)
        return;
    space += 6;
    printTree(node->right, space);
    printf("\n");
    for (int i = 6; i < space; i++)
        printf(" ");
    printf("%d(%c)", node->data, node->color == RED ? 'R' : 'B');
```



```
    printf("%d(%c) ", node->data, node->color == RED ? 'R' : 'B');
    printTree(node->left, space);
}
int main() {
    int n, val;
    printf("Enter number of elements: ");
    see:
    scanf("%d", &n);
    printf("Enter elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        insert(val);
    }
    printf("\nRed-Black Tree Structure:\n");
    printTree(root, 0);

    return 0;
}
```

OUTPUT :



```
Enter elements:
157 110 147 122 111 149 151 141 143 112 117 133

Red-Black Tree Structure:

                  157(R)
            151(B)
                  149(R)
      147(B)
            143(B)
141(B)
                        133(R)
                  122(B)
            117(R)
                  112(B)
      111(B)
            110(B)
```

Space Complexity:

O(n)

Time Complexity:

O(log n)