AMRITA SCHOOL OF COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS (23CSE211)

Name: Y.V.S.Likesh

Roll No.: CH.SC.U4CSE24152

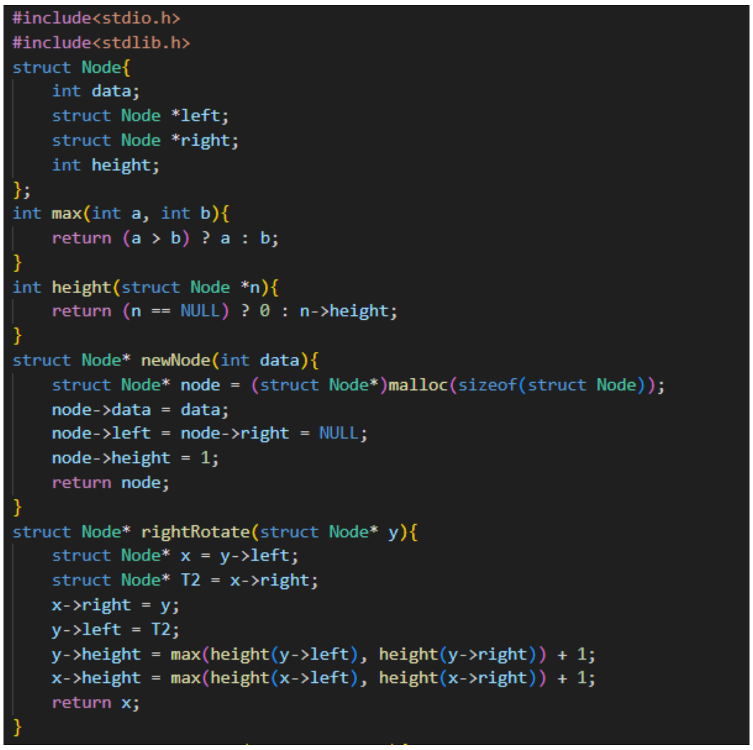
Class: BTech (CSE-B)

School: Amrita School of Computing,
Chennai Campus.

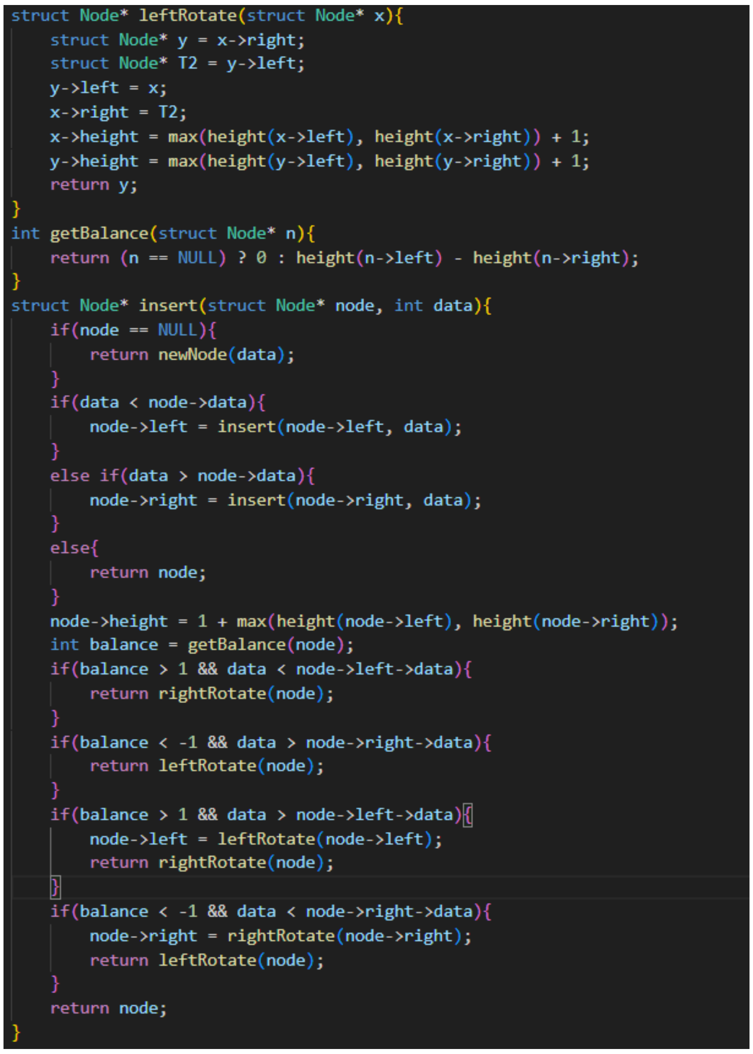
LAB-5

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

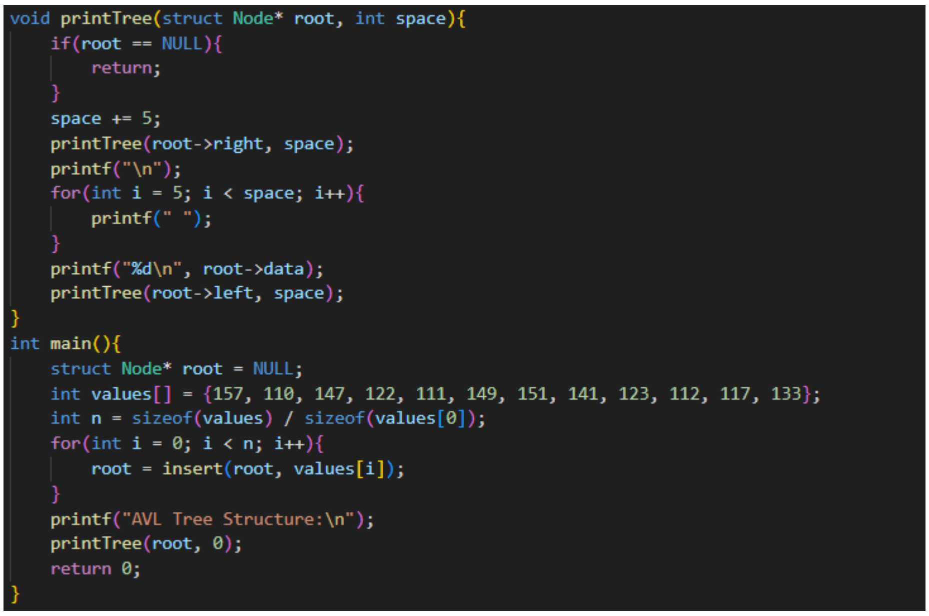
Code:



```c
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node *left;
    struct Node *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;
    return x;
}
```

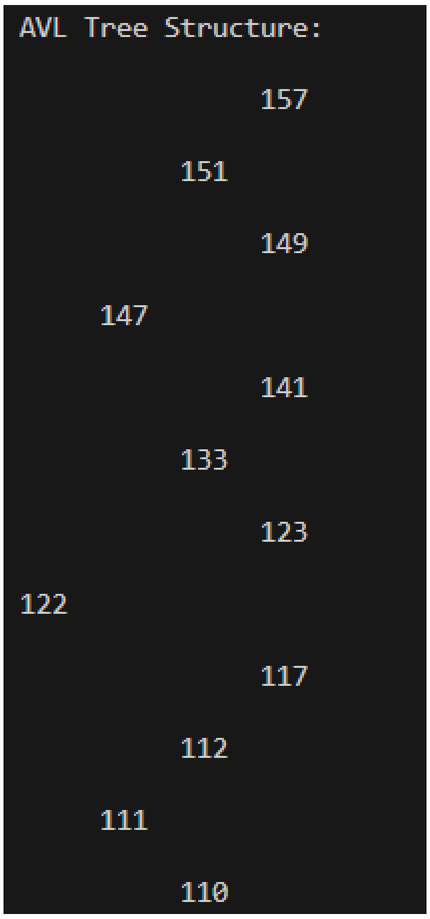


```c
struct Node* leftRotate(struct Node* x){
    struct Node* y = x->right;
    struct Node* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}
int getBalance(struct 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;
}
```



```c
void printTree(struct Node* root, int space){
    if(root == NULL){
        return;
    }
    space += 5;
    printTree(root->right, space);
    printf("\n");
    for(int i = 5; i < space; i++){
        printf(" ");
    }
    printf("%d\n", root->data);
    printTree(root->left, space);
}
int main(){
    struct Node* root = NULL;
    int values[] = {157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133};
    int n = sizeof(values) / sizeof(values[0]);
    for(int i = 0; i < n; i++){
        root = insert(root, values[i]);
    }
    printf("AVL Tree Structure:\n");
    printTree(root, 0);
    return 0;
}
```

Output:



```
AVL Tree Structure:

               157

          151

               149

     147

               141

          133

               123

122

               117

          112

     111

          110
```

Space Complexity:

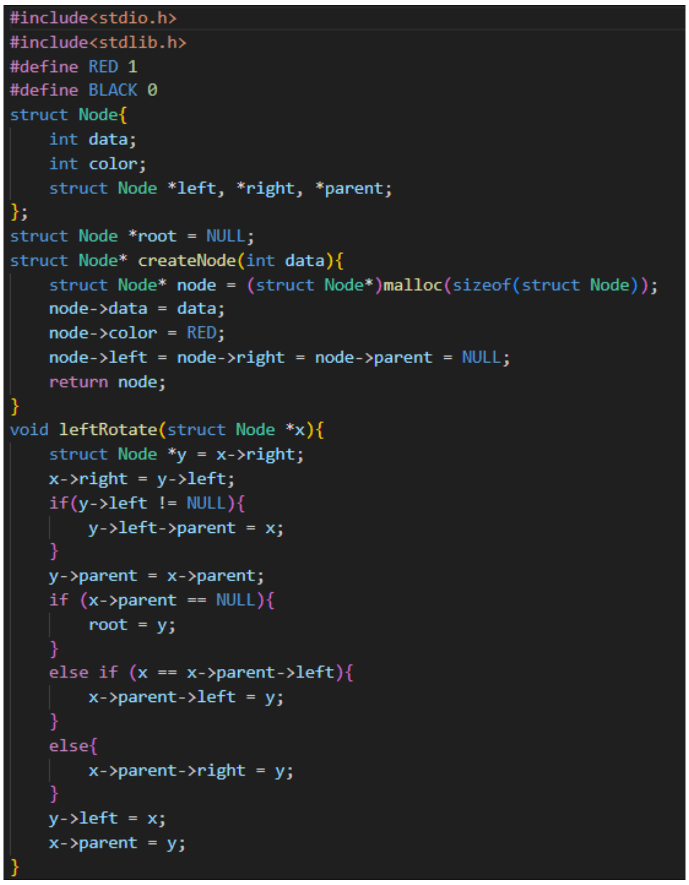
The space complexity of this program is O(n) for storing nodes and with O(1) extra space for height.

Time Complexity:

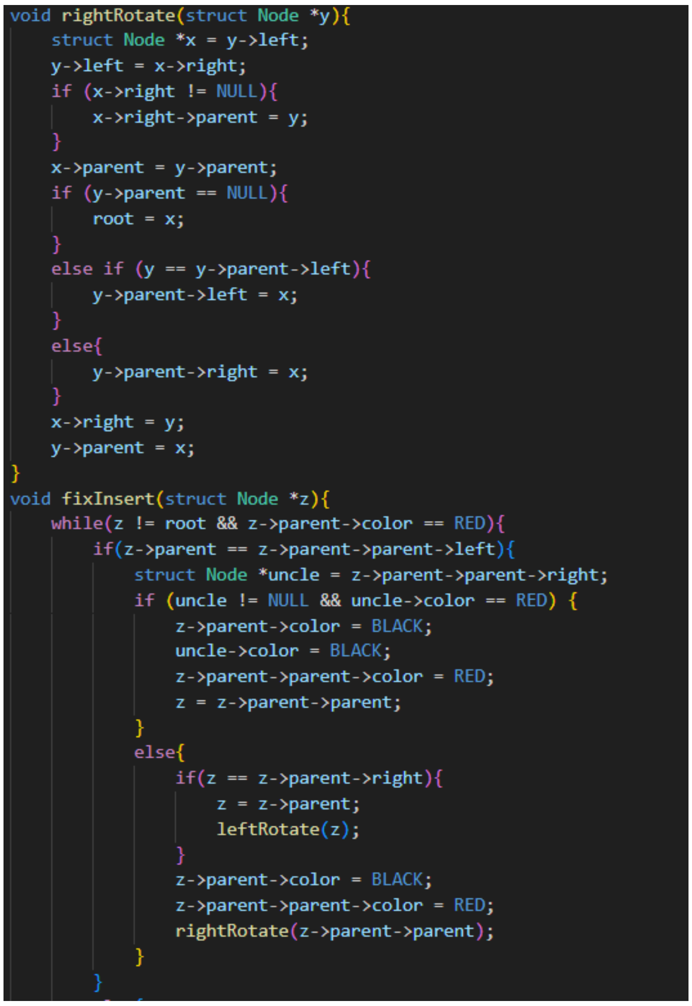
The time complexity of this program is O(log n). Because the tree is strictly height-balanced.

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

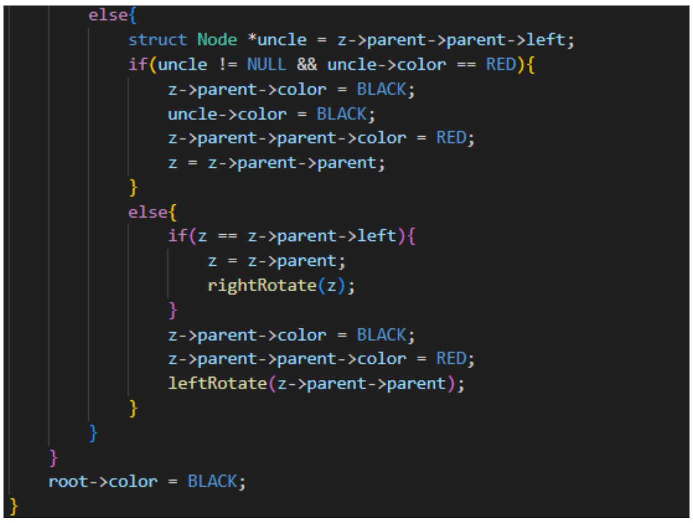
Code:



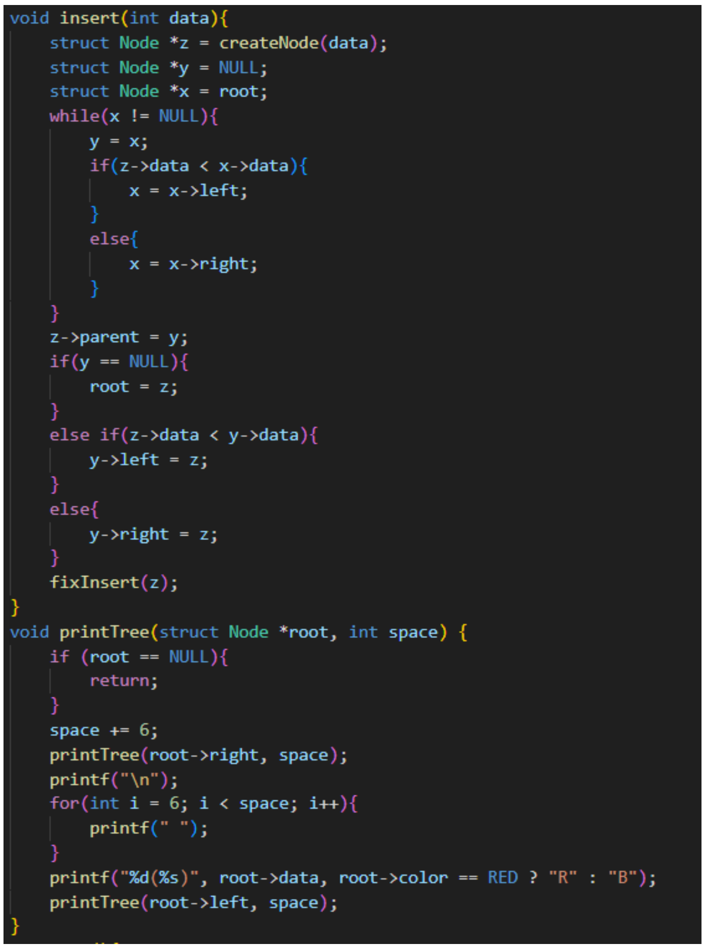
```c
#include<stdio.h>
#include<stdlib.h>
#define RED 1
#define BLACK 0
struct Node{
    int data;
    int color;
    struct Node *left, *right, *parent;
};
struct Node *root = NULL;
struct Node* createNode(int data){
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->color = RED;
    node->left = node->right = node->parent = NULL;
    return node;
}
void leftRotate(struct Node *x){
    struct Node *y = x->right;
    x->right = y->left;
    if(y->left != NULL){
        y->left->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == NULL){
        root = y;
    }
    else if (x == x->parent->left){
        x->parent->left = y;
    }
    else{
        x->parent->right = y;
    }
    y->left = x;
    x->parent = y;
}
```



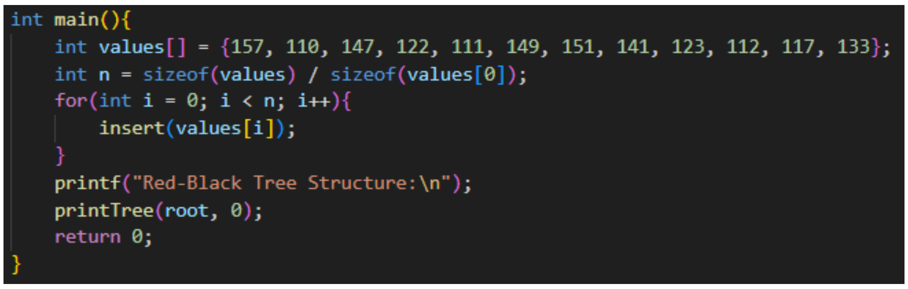
```c
void rightRotate(struct Node *y){
    struct Node *x = y->left;
    y->left = x->right;
    if (x->right != NULL){
        x->right->parent = y;
    }
    x->parent = y->parent;
    if (y->parent == NULL){
        root = x;
    }
    else if (y == y->parent->left){
        y->parent->left = x;
    }
    else{
        y->parent->right = x;
    }
    x->right = y;
    y->parent = x;
}
void fixInsert(struct Node *z){
    while(z != root && z->parent->color == RED){
        if(z->parent == z->parent->parent->left){
            struct Node *uncle = z->parent->parent->right;
            if (uncle != NULL && uncle->color == RED) {
                z->parent->color = BLACK;
                uncle->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            }
            else{
                if(z == z->parent->right){
                    z = z->parent;
                    leftRotate(z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                rightRotate(z->parent->parent);
            }
        }
```



```
        else{
            struct Node *uncle = z->parent->parent->left;
            if(uncle != NULL && uncle->color == RED){
                z->parent->color = BLACK;
                uncle->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            }
            else{
                if(z == z->parent->left){
                    z = z->parent;
                    rightRotate(z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                leftRotate(z->parent->parent);
            }
        }
    }
    root->color = BLACK;
}
```

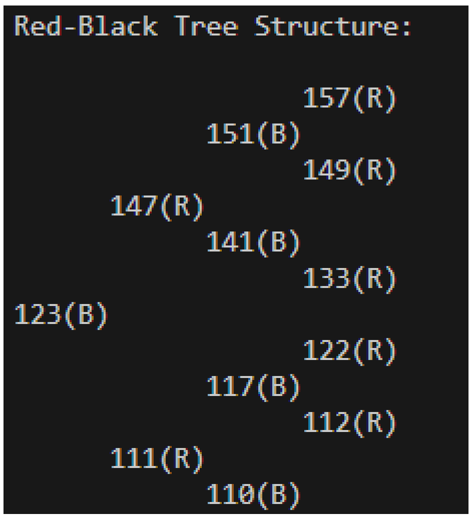


```
void insert(int data){
    struct Node *z = createNode(data);
    struct Node *y = NULL;
    struct Node *x = root;
    while(x != NULL){
        y = x;
        if(z->data < x->data){
            x = x->left;
        }
        else{
            x = x->right;
        }
    }
    z->parent = y;
    if(y == NULL){
        root = z;
    }
    else if(z->data < y->data){
        y->left = z;
    }
    else{
        y->right = z;
    }
    fixInsert(z);
}
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(%s)", root->data, root->color == RED ? "R" : "B");
    printTree(root->left, space);
}
```



```
int main(){
    int values[] = {157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133};
    int n = sizeof(values) / sizeof(values[0]);
    for(int i = 0; i < n; i++){
        insert(values[i]);
    }
    printf("Red-Black Tree Structure:\n");
    printTree(root, 0);
    return 0;
}
```

Output:



```
Red-Black Tree Structure:

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

Space Complexity:

The space complexity of this program is O(n) for storing nodes and with O(1) extra space per node for colour information.

Time Complexity:

The time complexity of this program is O(log n). Because the tree is strictly height-balanced.