RED BLACK TREE:

CODE:

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

```
void rightRotate(Node** root, Node* y) {
    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(Node** root, Node* node) {
    Node* parent = NULL;
    Node* grandparent = NULL;
    while (node != *root && node->parent->color == RED) {
        parent = node->parent;
        grandparent = parent->parent;
        if (parent == grandparent->left) {
            Node* uncle = grandparent->right;
            if (uncle != NULL && uncle->color == RED) {
                parent->color = BLACK;
                uncle->color = BLACK;
                grandparent->color = RED;
                node = grandparent;
            } else {
                if (node == parent->right) {
                    node = parent;
                    leftRotate(root, node);
                    parent = node->parent;
                }
                parent->color = BLACK;
                grandparent->color = RED;
                rightRotate(root, grandparent);
```

```
            }
        } else {
            Node* uncle = grandparent->left;
            if (uncle != NULL && uncle->color == RED) {
                parent->color = BLACK;
                uncle->color = BLACK;
                grandparent->color = RED;
                node = grandparent;
            } else {
                if (node == parent->left) {
                    node = parent;
                    rightRotate(root, node);
                    parent = node->parent;
                }
                parent->color = BLACK;
                grandparent->color = RED;
                leftRotate(root, grandparent);
            }
        }
    }
    (*root)->color = BLACK;
}
void insert(Node** root, int data) {
    Node* newNode = createNode(data);
    Node* y = NULL;
    Node* x = *root;
    while (x != NULL) {
        y = x;
        if (newNode->data < x->data) {
            x = x->left;
        } else {
            x = x->right;
        }
    }
    newNode->parent = y;
    if (y == NULL) {
        *root = newNode;
    } else if (newNode->data < y->data) {
        y->left = newNode;
```

```
    } else {
        y->right = newNode;
    }
    fixInsert(root, newNode);
}
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d(%s) ", root->data, root->color == RED ? "RED" : "BLACK");
        inorderTraversal(root->right);
    }
}
void freeTree(Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
int main() {
    Node* root = NULL;
    insert(&root, 1);
    insert(&root, 2);
    insert(&root, 3);
    insert(&root, 4);
    insert(&root, 5);
    insert(&root, 6);
    insert(&root, 7);
    printf("Inorder traversal of the Red-Black Tree:\n");
    inorderTraversal(root);
    printf("\n");
    freeTree(root);
    return 0;
}
```

OUTPUT:

```
Inorder traversal of the Red-Black Tree:
1(BLACK) 2(BLACK) 3(BLACK) 4(RED) 5(RED) 6(BLACK) 7(RED)
```

SPLAY TREE:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
void rightRotate(Node** root) {
    Node* leftChild = (*root)->left;
    (*root)->left = leftChild->right;
    leftChild->right = *root;
    *root = leftChild;
}
void leftRotate(Node** root) {
    Node* rightChild = (*root)->right;
    (*root)->right = rightChild->left;
    rightChild->left = *root;
    *root = rightChild;
}
void splay(Node** root, Node* x) {
    if (*root == NULL || x == NULL) return;
    while (x != *root) {
        if (x->left == *root) {
            rightRotate(root);
        } else if (x->right == *root) {
            leftRotate(root);
        } else {
            if (x->left == (*root)->left) {
                rightRotate(root);
                rightRotate(root);
```

```
            } else {
                leftRotate(root);
                rightRotate(root);
            }
        }
    }
}
Node* insert(Node* root, int data) {
    Node* newNode = createNode(data);
    if (root == NULL) return newNode;

    Node* x = root;
    Node* y = NULL;
    while (x != NULL) {
        y = x;
        if (data < x->data) x = x->left;
        else x = x->right;
    }
    if (data < y->data) y->left = newNode;
    else y->right = newNode;

    splay(&root, newNode);
    return root;
}
Node* search(Node* root, int key) {
    Node* x = root;
    while (x != NULL) {
        if (key == x->data) {
            splay(&root, x);
            return x;
        } else if (key < x->data) {
            x = x->left;
        } else {
            x = x->right;
        }
    }
    return NULL;
}
void inorderTraversal(Node* root) {
```

```
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
void freeTree(Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
int main() {
    Node* root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 15);
    root = insert(root, 25);
    root = insert(root, 5);
    printf("Inorder traversal of the Splay Tree:\n");
    inorderTraversal(root);
    printf("\n");
    freeTree(root);
    return 0;
}
```

OUTPUT:

Inorder traversal of the Splay Tree:

5 10 15 20 25 30