

## Design Analysis and Algorithm – Lab Work

### Week 4

#### 1. Balancing using AVL Tree

##### CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};
int height(struct Node *N) {
    if (N == NULL) return 0;
    return N->height;
}
int max(int a, int b) {
    return (a > b) ? a : b;
}

struct Node* newNode(int key) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    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;
}

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) {
    if (N == NULL) return 0;
    return height(N->left) - height(N->right);
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return(newNode(key));

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
}
```

```

        node->right = insert(node->right, key);
    }
    else
        return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}

void inOrder(struct Node *root) {
    if(root != NULL) {
        inOrder(root->left);
        printf("%d ", root->key);
        inOrder(root->right);
    }
}

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

    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &temp);
        root = insert(root, temp);
    }
    printf("\nSorted array :\n");
    inOrder(root);
    printf("\n");
    return 0;
}

```

OUTPUT:

```

Enter number of elements: 6
Enter 6 elements:
29 71 63 13 9 83

Sorted array :
9 13 29 63 71 83

```

**Time Complexity:**  $O(n \log n)$  for both average and worst cases.

**SPACE COMPLEXITY:**  $O(n)$  to store the nodes and recursion stack.