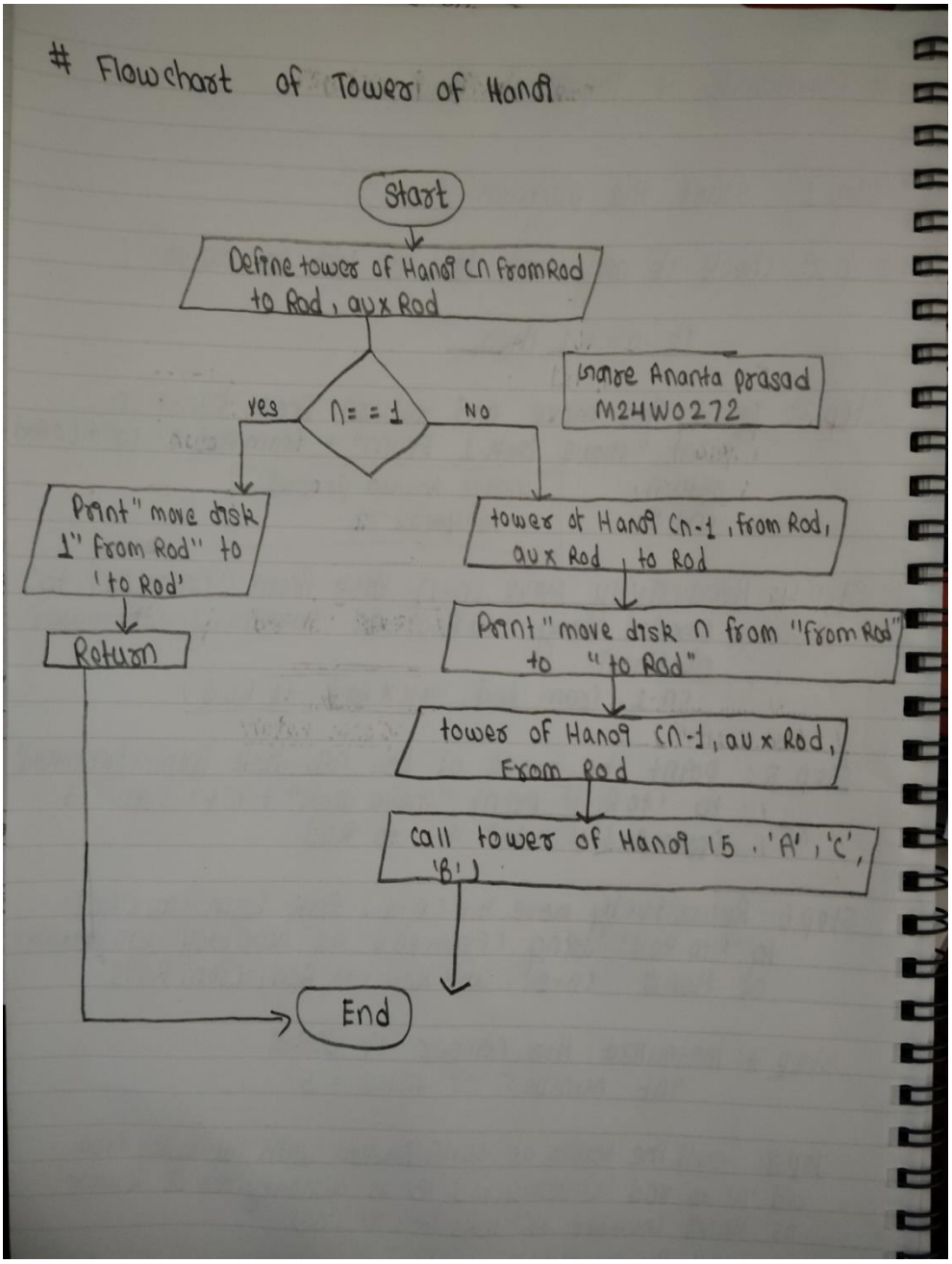
Gaire Ananta Prasad (M24W0272)

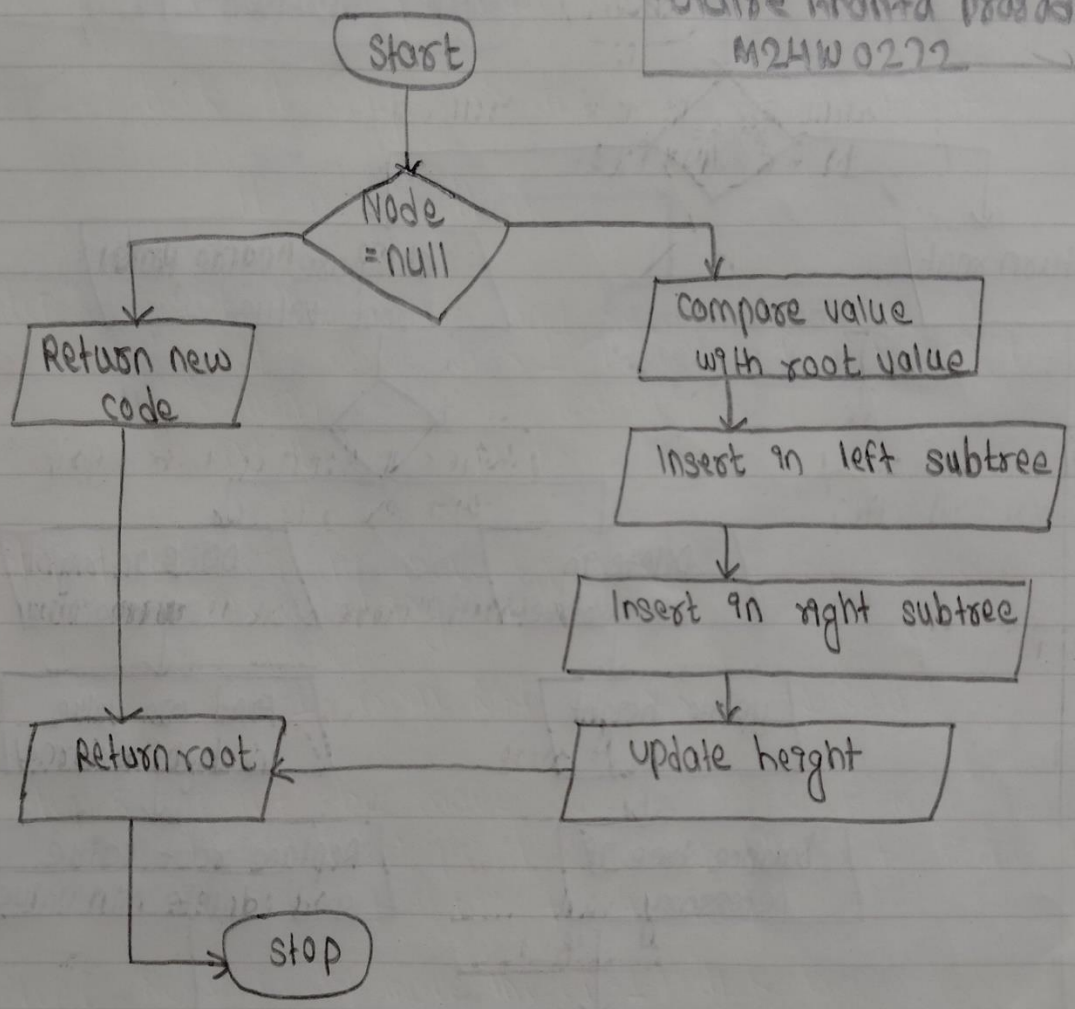
1. Binary Search Tree (BST) incorporating
2. Equilibrium Trees (AVL) incorporating rotations operations after



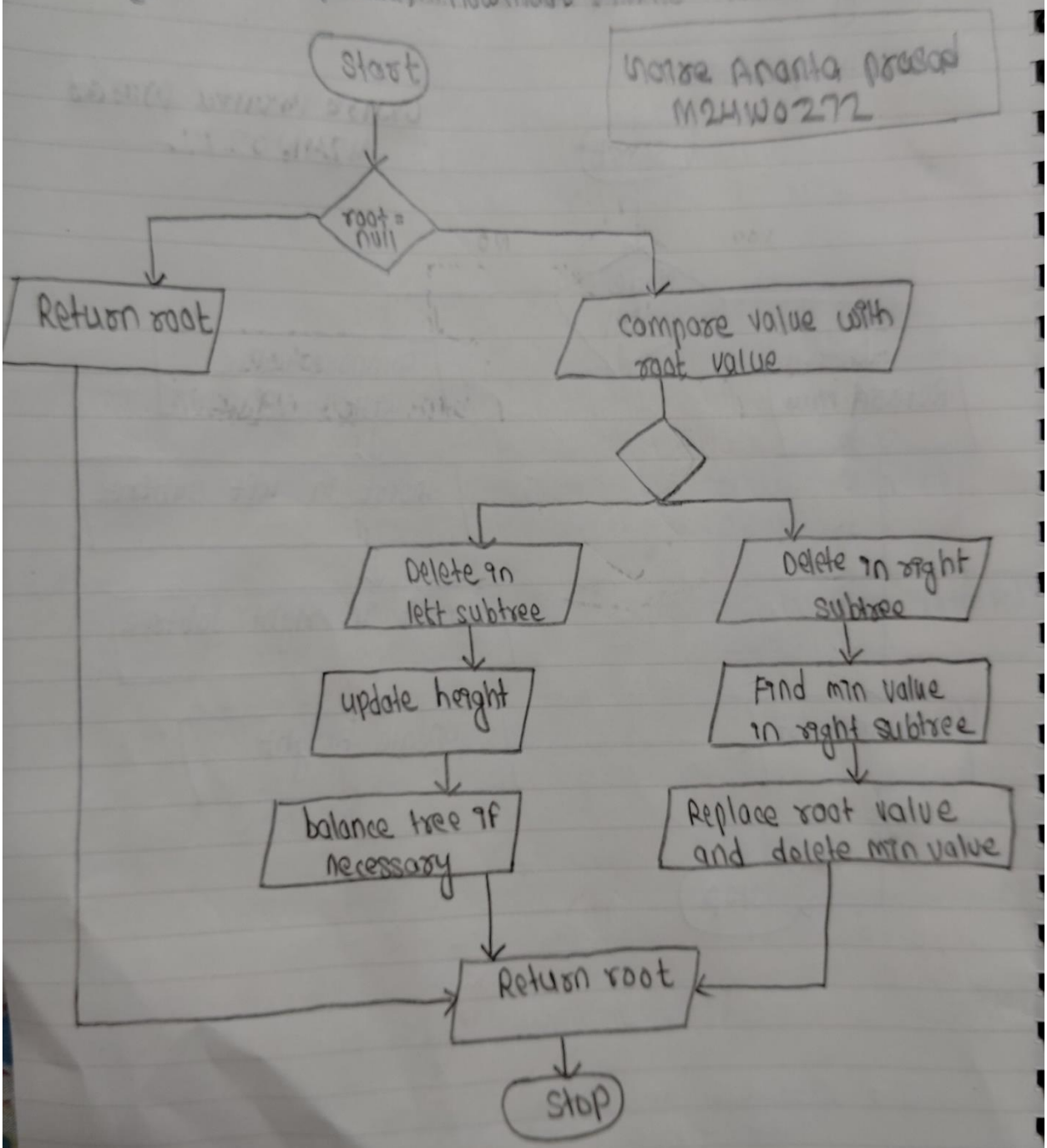
Binary search Tree (BST) Flowchart

* search operation :

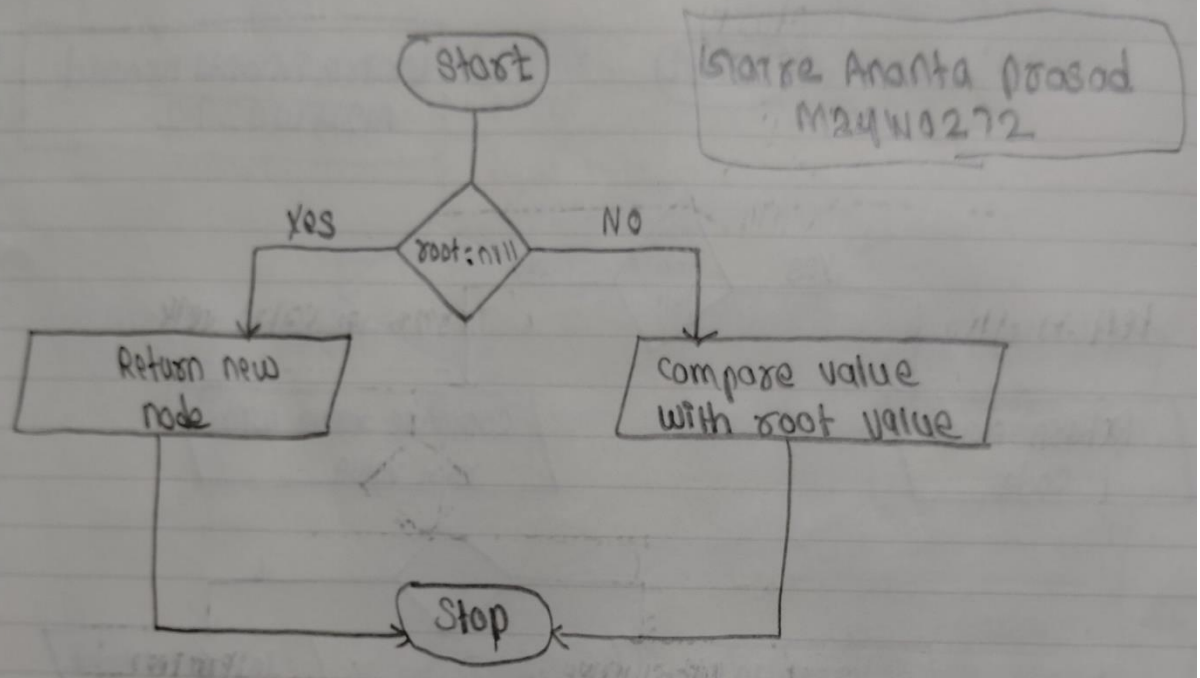
Gaire Ananta Prasad
M24W0272



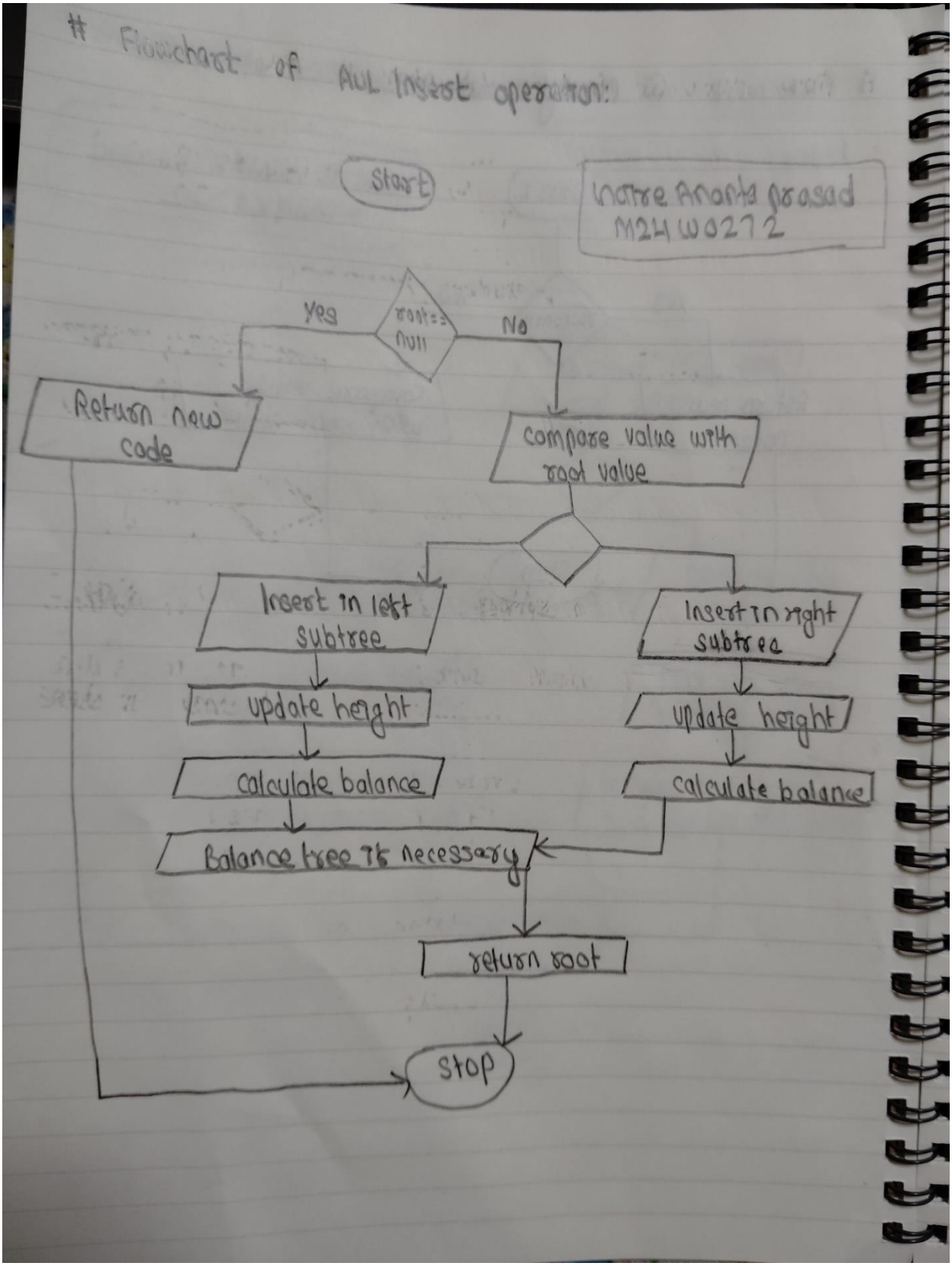
Binary delete operation flowchart



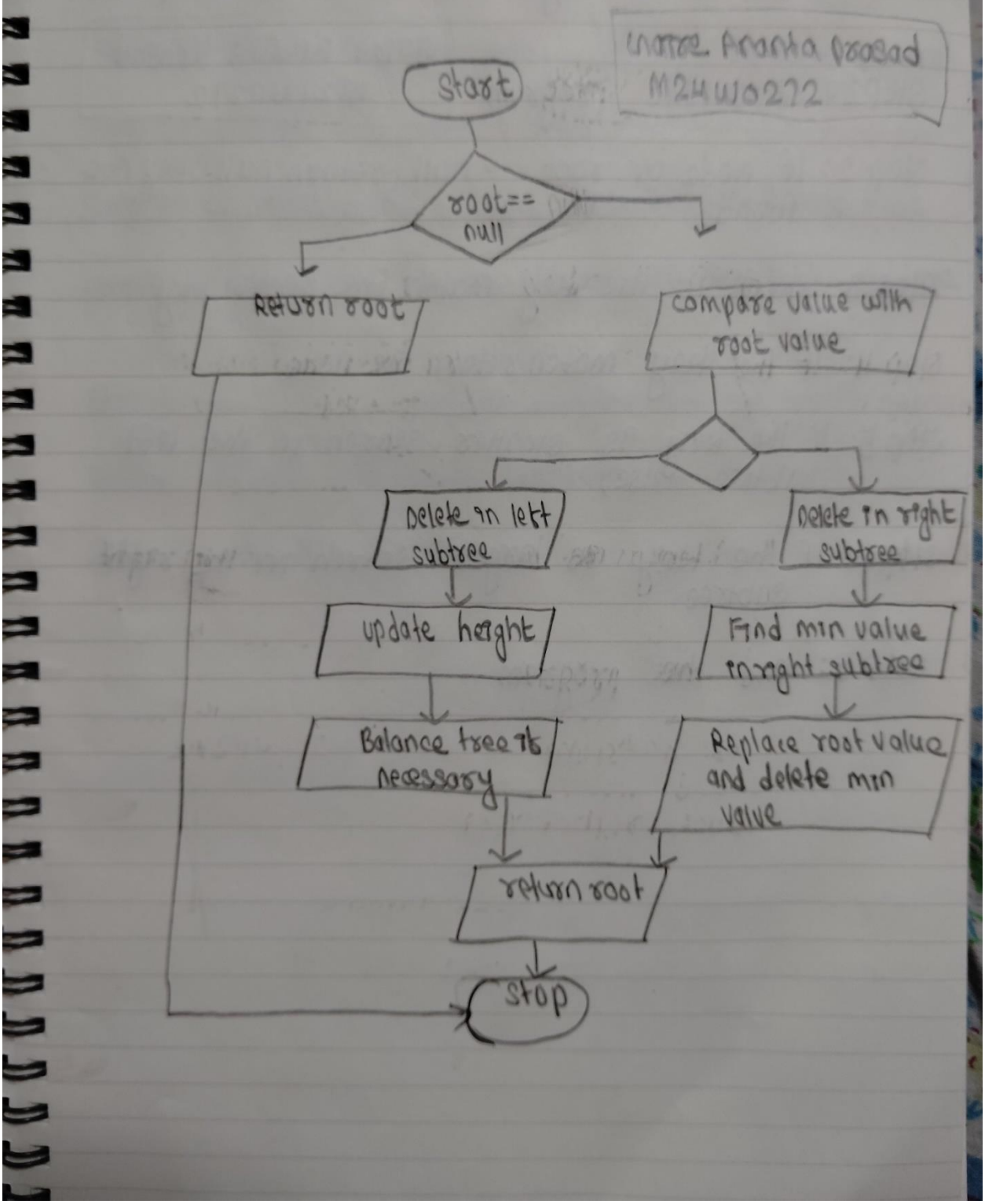
Flowchart of Binary search tree Insert operation



Flowchart of AVL Insert operation:



Flow chart of Delete operation of AVL



Pseudocode of Binary search tree (search operation)

Gaire Ananta Prasad
M24W0272

Step1: Start the program

Step 2: If node or root = = null, return null (key not found)

Step 3: check if the key match the root's key

Step 4: If the key match return the node

Step 5: If the key is smaller, search in the left subtree else,

Step 6: If the key is larger, search in the right subtree

Step 7: end the program

pseudocode of BTS (insert operation)

Gaire Ananta Prasad
M24W0272

Step 1: start the program.

Step 2: check if root = Null

Step 3: If root = null, create and return a new node

Step 4: If key is smaller, insert in the left subtree
else,

Step 5: If key is larger, insert in the right subtree

Step 6: Return the node pointer.

Step 7: end the program

pseudocode of BST delete operation)

Step 1: start the Program

Step 2: check if root = null

Step 3: If root = null, creat and return a new node

Step 4: If key is smaller, Insert in the left subtree

else,

Gaire Ananta Prasad
M24W0272

Step 5: If key is larger, Insert in the right subtree

Step 6: Return the node pointer.

Step 7: end the Program.

Pseudocode of BTS (delete operation)

Step 1: Start the program

Step 2: check, Node or root is null

Step 3: If root is null, return if (key not found)

Step 4: If key is smaller, delete in the left subtree
else,

Step 5: If key is larger, delete in the right subtree

Step 6: If key match delete the node

Step 7: Return the node pointer

Step 8: End the program.

Gaire Ananta prasad
M24WD272

pseuduced of AVL Insert operations.

Step 1: start the program

step 2: check, root is Null or not

step 3: It root is Null, create and return a new code

Step 4: It the key is larger, insert in right subtree
else

step 5: Insert in left subtree, if key is smaller

step 6: update the height of this ancestor node.

step 7: net the balance factor of this ancestor node

Step 8: It the Node become unbalance, then 4 cases

case 1: left, left
case 2: right, right
case 3: left, right
case 4: right, left

name Ananta prasad
M24W0272

step 9: return the pointer (unchange order)

step 10: End the program.

psecudocode of AVL delete operation with rotation.

step1: Start the program.
step 2: check if Node is null or not
step 3: If Node is null return it (key not found)
step 4: If the key is smaller, delete the left subtree
else,
step 5: If the key is larger, delete the right subtree

step 6: If the key match, delete the node
case 1: If Node > left = = null
temp = node > right
free (node)
return temp

else,
case 2 If Node > right = = null
temp = node > left
free (node)
return temp

Name Ananta prasad
M24W0272

step 7: Get the inorder successor (smallest in the right subtree
step 8: copy the inorder sucessor's key to this node.
step 9: Delete the inorder sucessor.
step 10: update the height of this ancessor node.
step 11: Get the balance factor.

step 12: end the program.

bst.h

```c
//Gaire Ananta Prasad (M24W0272)
#ifndef BST_H
#define BST_H

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

struct Node* createNode(int data);
struct Node* insert(struct Node* root, int data);
struct Node* search(struct Node* root, int data);
struct Node* deleteNode(struct Node* root, int data);
void inorderTraversal(struct Node* root);

#endif

```

bst.c

```c
//Gaire Ananta Prasad M24W0272
#include <stdio.h>
#include <stdlib.h>
#include "bst.h"

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

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }

    return root;
}

struct Node* search(struct Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }

    if (data < root->data) {
        return search(root->left, data);
    } else {
        return search(root->right, data);
    }
}
```

Gaire Ananta Prasad (M24W0272)

```c
struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

struct Node* deleteNode(struct Node* root, int data) {
    if (root == NULL) {
        return root;
    }

    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }

        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}

void inorderTraversal(struct Node* root) {
```

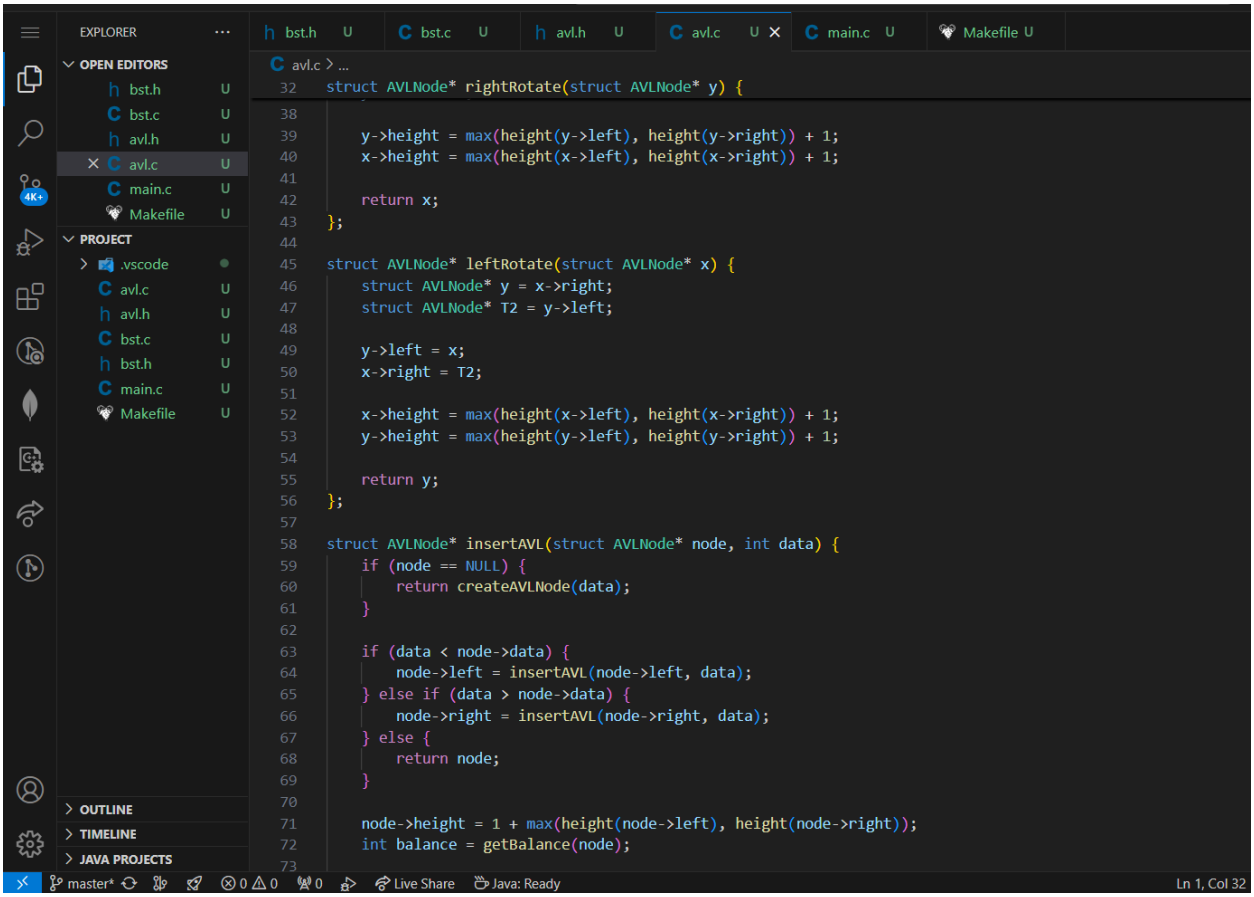
```c
}

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

Gaire Ananta Prasad (M24W0272)

```c
//Gaire Ananta Prasad M24W0272
#ifndef AVL_H
#define AVL_H

struct AVLNode {
    int data;
    struct AVLNode* left;
    struct AVLNode* right;
    int height;
};

struct AVLNode* createAVLNode(int data);
struct AVLNode* insertAVL(struct AVLNode* node, int data);
struct AVLNode* deleteAVLNode(struct AVLNode* root, int data);
void inorderTraversalAVL(struct AVLNode* root);

#endif
```

```c
//Gaire Ananta Prasad (M24W0272)
#include <stdio.h>
#include <stdlib.h>
#include "avl.h"

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

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

int height(struct AVLNode* node) {
    if (node == NULL) {
        return 0;
    };
    return node->height;
};

int getBalance(struct AVLNode* node) {
    if (node == NULL) {
        return 0;
    };
    return height(node->left) - height(node->right);
};

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

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

Gaire Ananta Prasad (M24W0272)



Gaire Ananta Prasad (M24W0272)

```c
struct AVLNode* insertAVL(struct AVLNode* node, int data) {
    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;
};

struct AVLNode* minValueNode(struct AVLNode* node) {
    struct AVLNode* current = node;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

struct AVLNode* deleteAVLNode(struct AVLNode* root, int data) {
    if (root == NULL) {
        return root;
    }

    if (data < root->data) {
        root->left = deleteAVLNode(root->left, data);
```

```c
struct AVLNode* deleteAVLNode(struct AVLNode* root, int data) {
        root->left = deleteAVLNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteAVLNode(root->right, data);
    } else {
        if ((root->left == NULL) || (root->right == NULL)) {
            struct AVLNode* temp = root->left ? root->left : root->right;

            if (temp == NULL) {
                temp = root;
                root = NULL;
            } else {
                *root = *temp;
            }
            free(temp);
        } else {
            struct AVLNode* temp = minValueNode(root->right);
            root->data = temp->data;
            root->right = deleteAVLNode(root->right, temp->data);
        }
    }

    if (root == NULL) {
        return root;
    }

    root->height = 1 + max(height(root->left), height(root->right));
    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0) {
        return rightRotate(root);
    }

    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
```

Gaire Ananta Prasad (M24W0272)

```c
struct AVLNode* deleteAVLNode(struct AVLNode* root, int data) {
    }

    if (balance < -1 && getBalance(root->right) <= 0) {
        return leftRotate(root);
    }

    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}

void inorderTraversalAVL(struct AVLNode* root) {
    if (root != NULL) {
        inorderTraversalAVL(root->left);
        printf("%d ", root->data);
        inorderTraversalAVL(root->right);
    }
}
```

```c
//Gaire Ananta Prasad (M24W0272)
#include <stdio.h>
#include "bst.h"
#include "avl.h"

int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    printf("BST Inorder Traversal: ");
    inorderTraversal(root);
    printf("\n");

    root = deleteNode(root, 20);
    printf("BST Inorder Traversal after deleting 20: ");
    inorderTraversal(root);
    printf("\n");

    root = deleteNode(root, 30);
    printf("BST Inorder Traversal after deleting 30: ");
    inorderTraversal(root);
    printf("\n");

    root = deleteNode(root, 50);
    printf("BST Inorder Traversal after deleting 50: ");
    inorderTraversal(root);
    printf("\n");

    // AVL Tree
    struct AVLNode* avlRoot = NULL;
    avlRoot = insertAVL(avlRoot, 50);
```

Gaire Ananta Prasad (M24W0272)

```c
int main() {
    avlRoot = insertAVL(avlRoot, 50);
    avlRoot = insertAVL(avlRoot, 30);
    avlRoot = insertAVL(avlRoot, 20);
    avlRoot = insertAVL(avlRoot, 40);
    avlRoot = insertAVL(avlRoot, 70);
    avlRoot = insertAVL(avlRoot, 60);
    avlRoot = insertAVL(avlRoot, 80);

    printf("AVL Tree Inorder Traversal: ");
    inorderTraversalAVL(avlRoot);
    printf("\n");

    avlRoot = deleteAVLNode(avlRoot, 20);
    printf("AVL Tree Inorder Traversal after deleting 20: ");
    inorderTraversalAVL(avlRoot);
    printf("\n");

    avlRoot = deleteAVLNode(avlRoot, 30);
    printf("AVL Tree Inorder Traversal after deleting 30: ");
    inorderTraversalAVL(avlRoot);
    printf("\n");

    avlRoot = deleteAVLNode(avlRoot, 50);
    printf("AVL Tree Inorder Traversal after deleting 50: ");
    inorderTraversalAVL(avlRoot);
    printf("\n");

    return 0;
}
```

```makefile
# Gaire Ananta Prasad (M24W0272)
CC = gcc
CFLAGS = -Wall -Wextra

all: bst_avl

bst_avl: main.o bst.o avl.o
	$(CC) $(CFLAGS) -o bst_avl main.o bst.o avl.o

main.o: main.c bst.h avl.h
	$(CC) $(CFLAGS) -c main.c

bst.o: bst.c bst.h
	$(CC) $(CFLAGS) -c bst.c

avl.o: avl.c avl.h
	$(CC) $(CFLAGS) -c avl.c

clean:
	rm -f *.o bst_avl
```

Gaire Ananta Prasad (M24W0272)