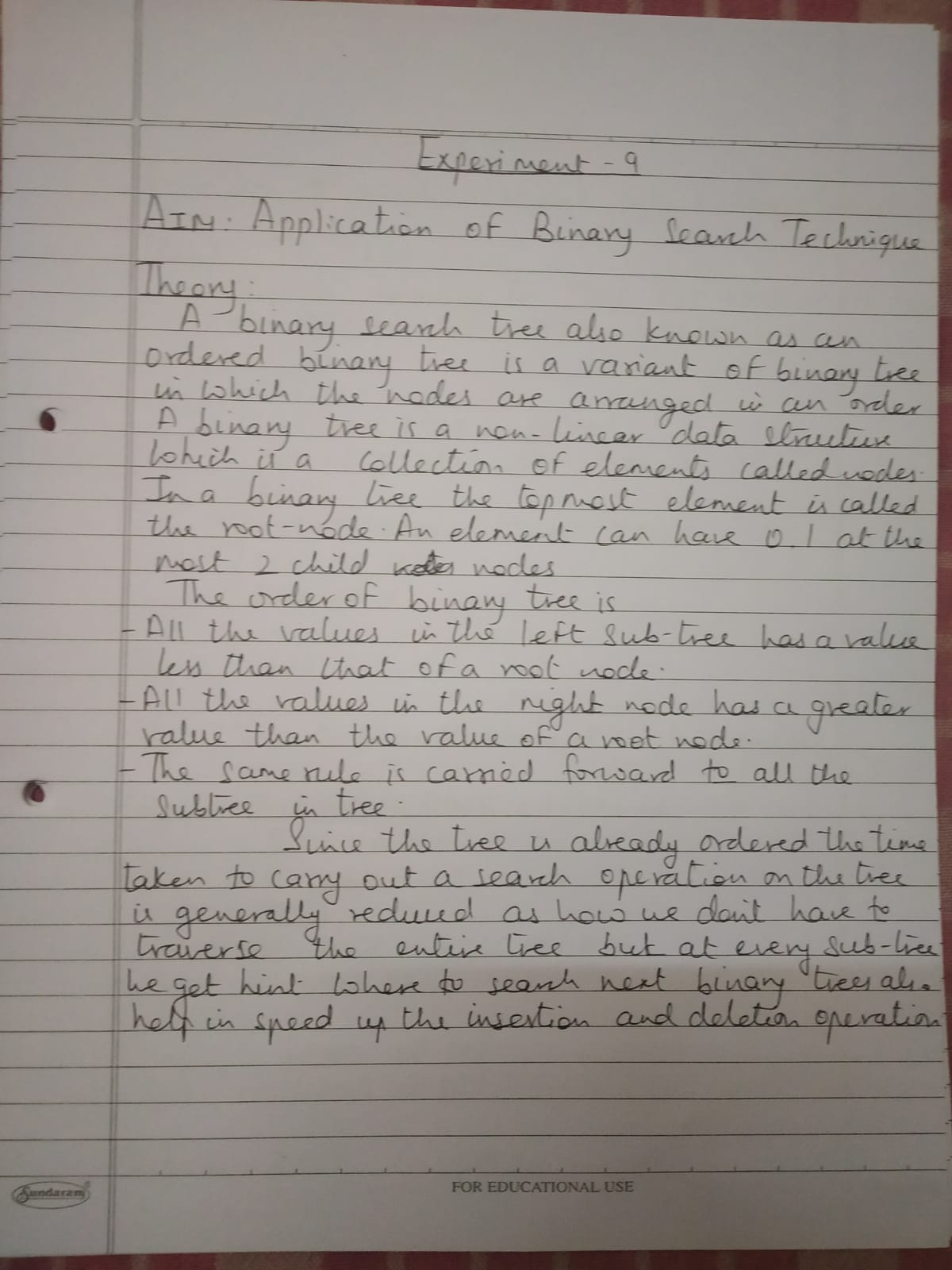
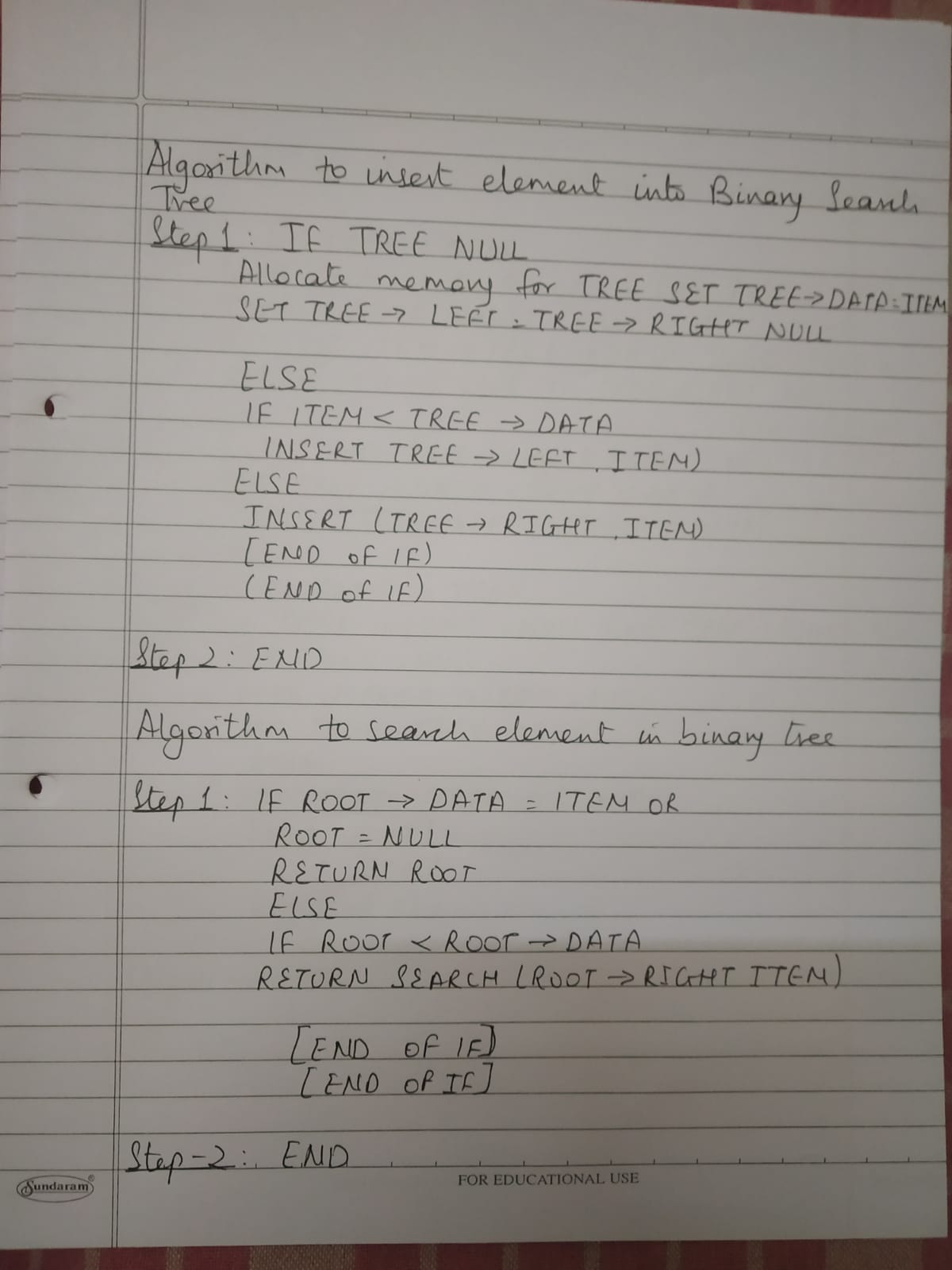
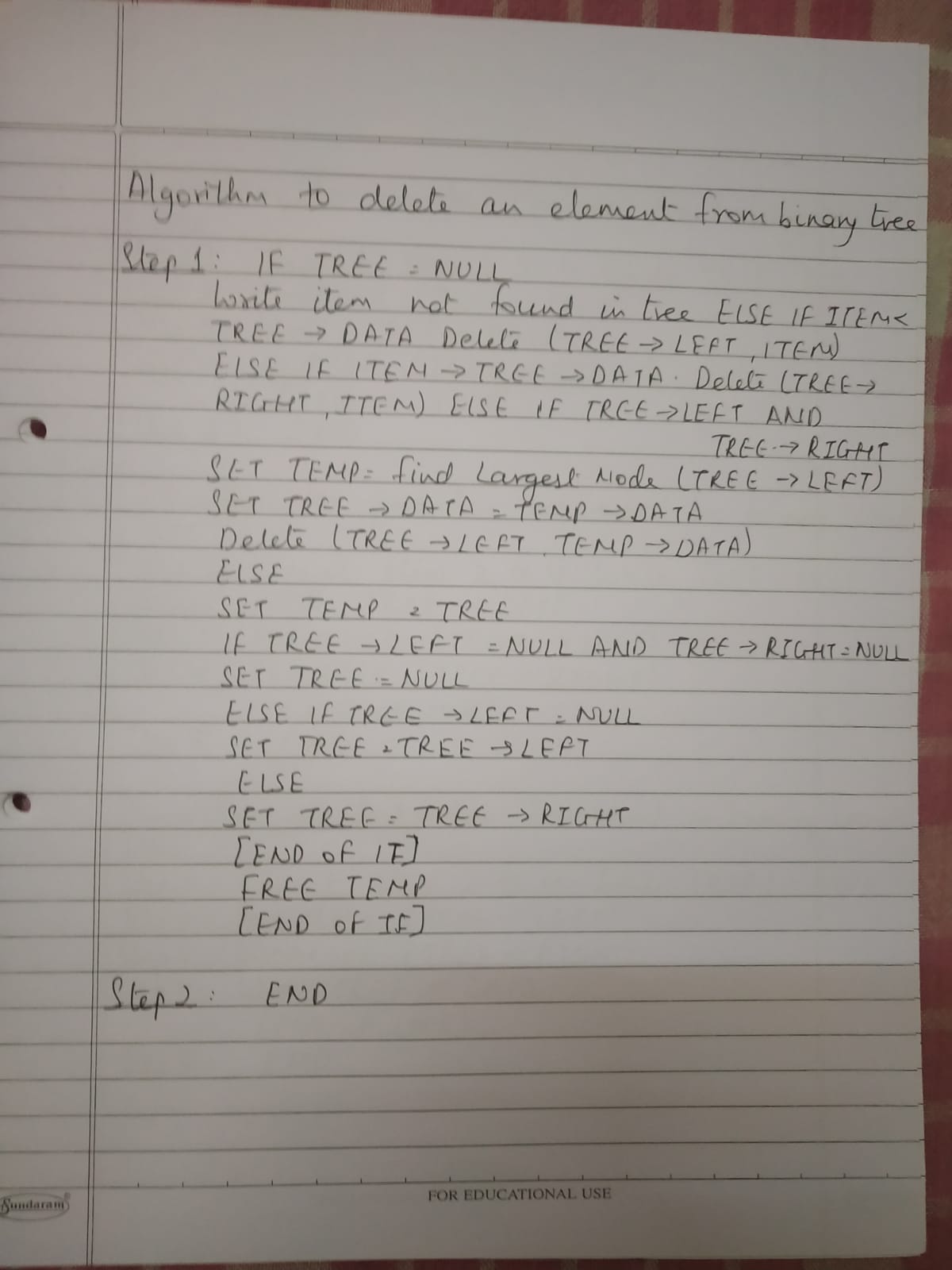


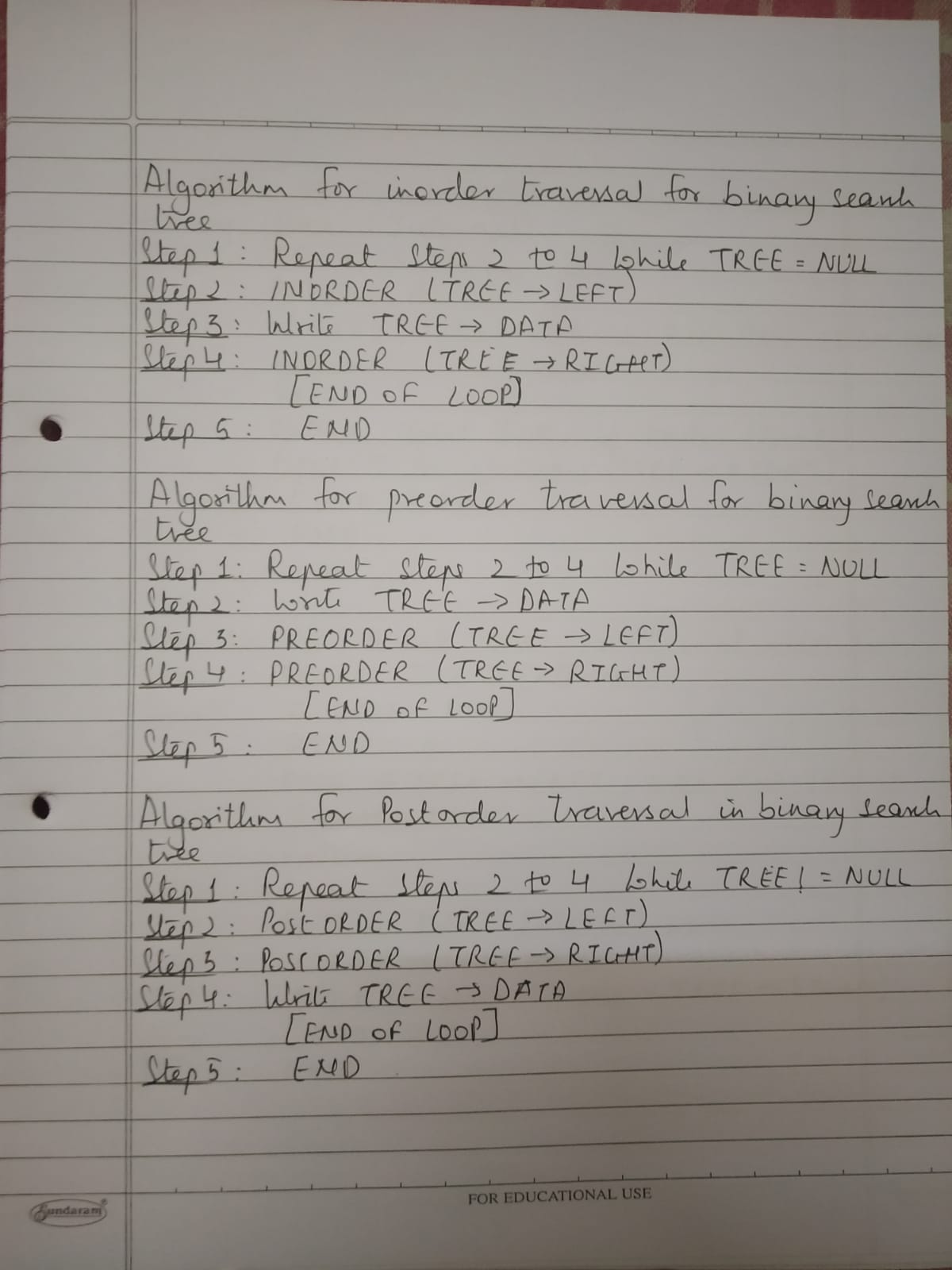
**COMPUTER ENGINEERING**

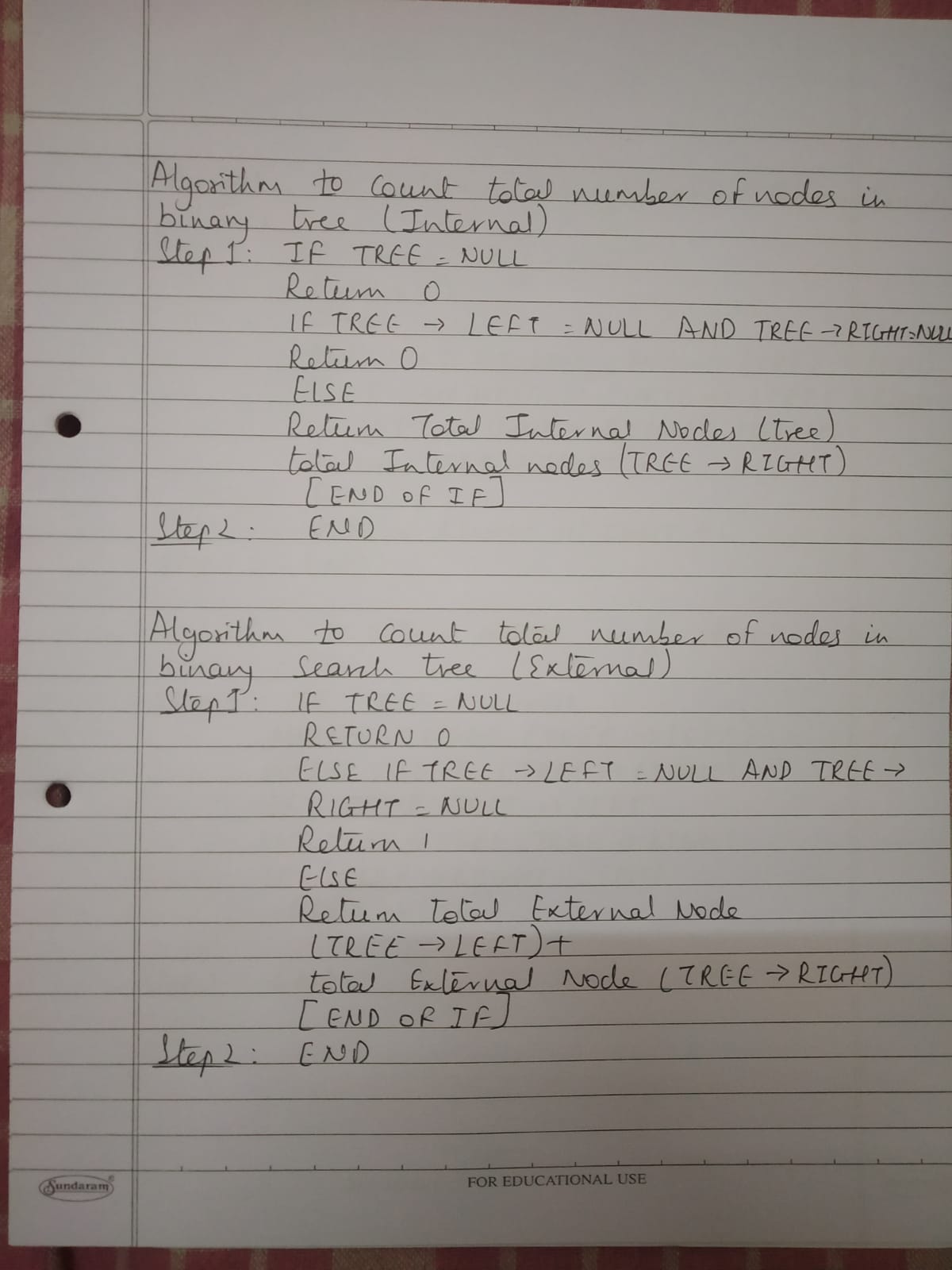
**DS ODD SEM 2021-22/EXPERIMENT 9 NAME:- GAURAV AMARNANI (D7A, 67)**





****

****

****

## 678

# Program:

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

struct node {

int data;

struct node \*left; struct node \*right;

};

struct node \*tree;

void create\_tree(struct node \*);

struct node \*insertElement(struct node \*, int);

void preorderTraversal(struct node \*);

void inorderTraversal(struct node \*);

void postorderTraversal(struct node \*);

struct node \*findSmallestElement(struct node \*);

struct node \*findLargestElement(struct node \*);

struct node \*deleteElement(struct node \*, int);

struct node \*mirrorImage(struct node \*);

struct node \*deleteTree(struct node \*);

int totalNodes(struct node \*);

int totalExternalNodes(struct node \*);

int totalInternalNodes(struct node \*);

int Height(struct node \*);

int main() {

int option, val;

struct node \*ptr;

create\_tree(tree);

do {

printf("\n \*\*\*MAIN MENU\*\* \n");

printf("\n 1. Insert Element");

printf("\n 2. Preorder Traversal");

printf("\n 3. Inorder Traversal");

printf("\n 4. Postorder Traversal");

printf("\n 5. Find the smallest element");

printf("\n 6. Find the largest element");

printf("\n 7. Delete an element");

printf("\n 8. Count the total number of nodes");

printf("\n 9. Determine the height of the tree");

printf("\n 10. Find the mirror image of the tree");

printf("\n 11. Delete the tree");

printf("\n 12. Exit");

printf("\n\n Enter your option : ");

scanf("%d", &option);

switch (option) {

case 1:

printf("\n Enter the value of the new node : ");

scanf("%d", &val);

tree = insertElement(tree, val);

break;

case 2:

printf("\n The elements of the tree are : \n");

preorderTraversal(tree);

break;

case 3:

printf("\n The elements of the tree are : \n");

inorderTraversal(tree);

break;

case 4:

printf("\n The elements of the tree are : \n");

postorderTraversal(tree);

break;

case 5:

ptr = findSmallestElement(tree);

printf("\n Smallest element is :%d", ptr->data);

break;

case 6:

ptr = findLargestElement(tree);

printf("\n Largest element is : %d", ptr->data);

break;

case 7:

printf("\n Enter the element to be deleted : ");

scanf("%d", &val);

tree = deleteElement(tree, val);

break;

case 8:

printf("\n Total no. of nodes = %d", totalNodes(tree));

break;

case 9:

printf("\n The height of the tree = %d", Height(tree));

break;

case 10:

tree = mirrorImage(tree);

break; case 11:

tree = deleteTree(tree);

break;

}

} while (option != 14);

getch();

return 0;

}

void create\_tree(struct node \*tree) {

tree = NULL;

}

struct node \*insertElement(struct node \*tree, int val) {

struct node \*ptr, \*nodeptr, \*parentptr;

ptr = (struct node \*)malloc(sizeof(struct node));

ptr->data = val;

ptr->left = NULL; ptr->right = NULL;

if (tree == NULL) {

tree = ptr;

tree->left = NULL;

tree->right = NULL;

}

else {

parentptr = NULL;

nodeptr = tree;

while (nodeptr != NULL) {

parentptr = nodeptr;

if (val< nodeptr->data) nodeptr = nodeptr->left;

else

}

nodeptr = nodeptr->right;

if (val<parentptr->data) parentptr->left = ptr;

else

}

parentptr->right = ptr;

return tree;

}

void preorderTraversal(struct node \*tree) {

if (tree != NULL) {

printf("%d\t", tree->data);

preorderTraversal(tree->left);

preorderTraversal(tree->right);

}

}

void inorderTraversal(struct node \*tree) {

if (tree != NULL) {

inorderTraversal(tree->left);

printf("%d\t", tree->data);

inorderTraversal(tree->right);

}

}

void postorderTraversal(struct node \*tree) {

if (tree != NULL) {

postorderTraversal(tree->left);

postorderTraversal(tree->right);

printf("%d\t", tree->data);

}

}

struct node \*findSmallestElement(struct node \*tree) {

if ((tree == NULL) || (tree->left == NULL)) return tree;

else

return findSmallestElement(tree->left);

}

struct node \*findLargestElement(struct node \*tree) {

if ((tree == NULL) || (tree->right == NULL))

return tree;

else

}

return findLargestElement(tree->right);

struct node \*deleteElement(struct node \*tree, int val) {

struct node \*cur, \*parent, \*suc, \*psuc, \*ptr;

if (tree->left == NULL) {

printf("\n The tree is empty ");

return (tree);

}

parent = tree; cur = tree->left;

while (cur != NULL && val != cur->data) {

parent = cur;

cur = (val< cur->data) ? cur->data : cur->right;

}

if (cur == NULL) {

printf("\n The value to be deleted is not present in the tree");

return (tree);

}

if (cur->left == NULL) ptr = cur->right;

else if (cur->right == NULL) ptr = cur->left;

else {

psuc = cur;

cur = cur->left;

while (suc->left != NULL) {

psuc = suc;

suc = suc->left;

}

if (cur == psuc) {

suc->left = cur->right;

}

suc->left = cur->right;

psuc->left = suc->right;

suc->right = cur->right;

ptr = suc;

}

parent->left = ptr;

else

parent->right = ptr;

free(cur);

return tree;

}

int totalNodes(struct node \*tree) {

if (tree == NULL)

return 0;

else

}

return (totalNodes(tree->left) + totalNodes(tree->right) + 1);

int Height(struct node \*tree) {

int leftheight, rightheight;

if (tree == NULL)

return 0;

else {

leftheight = Height(tree->left);

rightheight = Height(tree->right);

if (leftheight > rightheight)

return (leftheight + 1);

else

}

}

return (rightheight + 1);

struct node \*mirrorImage(struct node \*tree) {

struct node \*ptr;

if (tree != NULL) {

mirrorImage(tree->left);

mirrorImage(tree->right);

ptr = tree->left;

ptr->left = ptr->right;

tree->right = ptr;

}

}

struct node \*deleteTree(struct node \*tree) {

if (tree != NULL) {

deleteTree(tree->left);

deleteTree(tree->right);

free(tree);

}

}

# Output:

