DAA LAB ASSIGNMENT-5

1.QUICK SORT:

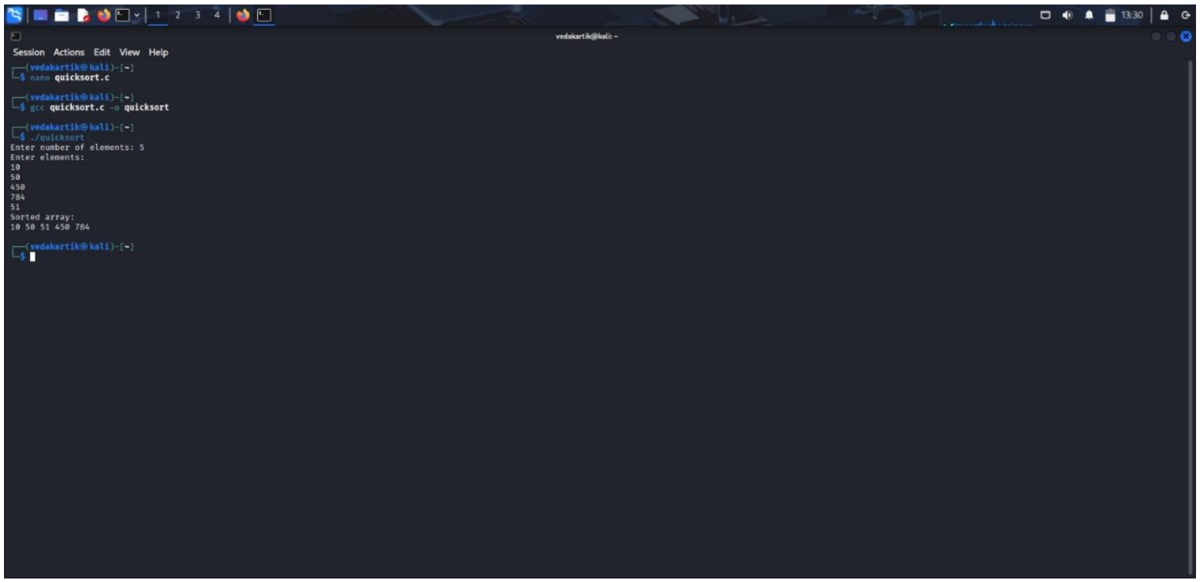
CODE:

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node *left;

    struct Node *right;

};

struct Node* createNode(int value) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = value;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

struct Node* insert(struct Node* root, int value) {

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

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

    return root;
}
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}
int search(struct Node* root, int key) {
    if (root == NULL)
        return 0;

    if (root->data == key)
        return 1;

    if (key < root->data)
        return search(root->left, key);
    else
```

```
        return search(root->right, key);
}
void freeTree(struct Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

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

    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter values:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }
    printf("Inorder traversal: ");

    inorder(root);

    printf("\n");

    printf("Preorder traversal: ");

    preorder(root);

    printf("\n");

    printf("Postorder traversal: ");

    postorder(root);

    printf("\n");

    printf("Enter value to search: ");

    scanf("%d", &key);


    if (search(root, key))

        printf("Value found in BST\n");

    else

        printf("Value not found in BST\n");

    freeTree(root);

    return 0;

}
```

OUTPUT:

};



```
┌──(vedakartik㉿kali)-[~]
└─$ nano quicksort.c

┌──(vedakartik㉿kali)-[~]
└─$ gcc quicksort.c -o quicksort

┌──(vedakartik㉿kali)-[~]
└─$ ./quicksort
Enter number of elements: 5
Enter elements:
10
50
450
784
51
Sorted array:
10 50 51 450 784

┌──(vedakartik㉿kali)-[~]
└─$ 
```

<u>*2.MERGE SORT*</u>

<u>***CODE:***</u>

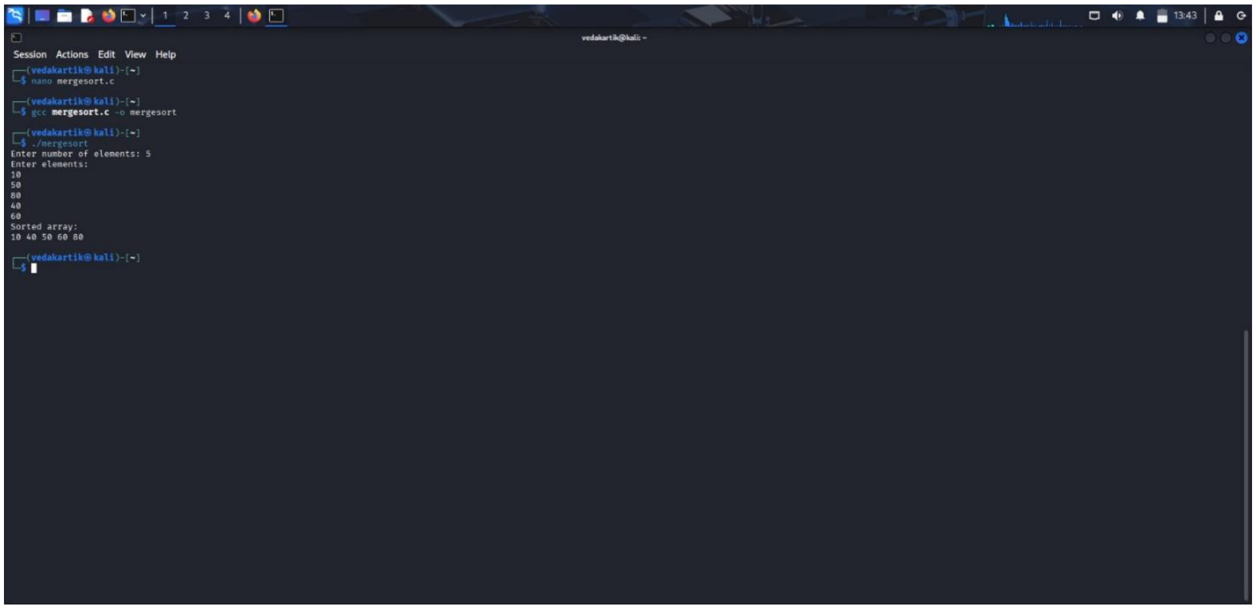
```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node *left;

    struct Node *right;

};
```

```
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }

    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);

        printf("%d ", root->data);

        inorder(root->right);

    }

}

)

void preorder(struct Node* root) {

    if (root != NULL) {

        printf("%d ", root->data);

        preorder(root->left);

        preorder(root->right);

    }

}


void postorder(struct Node* root) {

    if (root != NULL) {

        postorder(root->left);

        postorder(root->right);

        printf("%d ", root->data);

    }

}
```

```
int search(struct Node* root, int key) {
    if (root == NULL)
        return 0;
    if (root->data == key)
        return 1;
    if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}
void freeTree(struct Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

int main() {
    struct Node* root = NULL;
    int n, value, key;
```

```
printf("Enter number of nodes: ");

scanf("%d", &n);

printf("Enter values:\n");

for (int i = 0; i < n; i++) {

    scanf("%d", &value);

    root = insert(root, value);

}

printf("Inorder traversal: ");

inorder(root);

printf("\n");

printf("Preorder traversal: ");

preorder(root);

printf("\n");

printf("Postorder traversal: ");

postorder(root);

printf("\n");

printf("Enter value to search: ");

scanf("%d", &key);


if (search(root, key))

    printf("Value found in BST\n");
    else
        printf("Value not found in BST\n");
    freeTree(root);
    return 0;
}
```

OUTPUT:



```
┌──(vedakartik㉿kali)-[~]
└─$ nano mergesort.c

┌──(vedakartik㉿kali)-[~]
└─$ gcc mergesort.c -o mergesort

┌──(vedakartik㉿kali)-[~]
└─$ ./mergesort
Enter number of elements: 5
Enter elements:
10
50
80
40
60
Sorted array:
10 40 50 60 80

┌──(vedakartik㉿kali)-[~]
└─$ 
```

3)BST

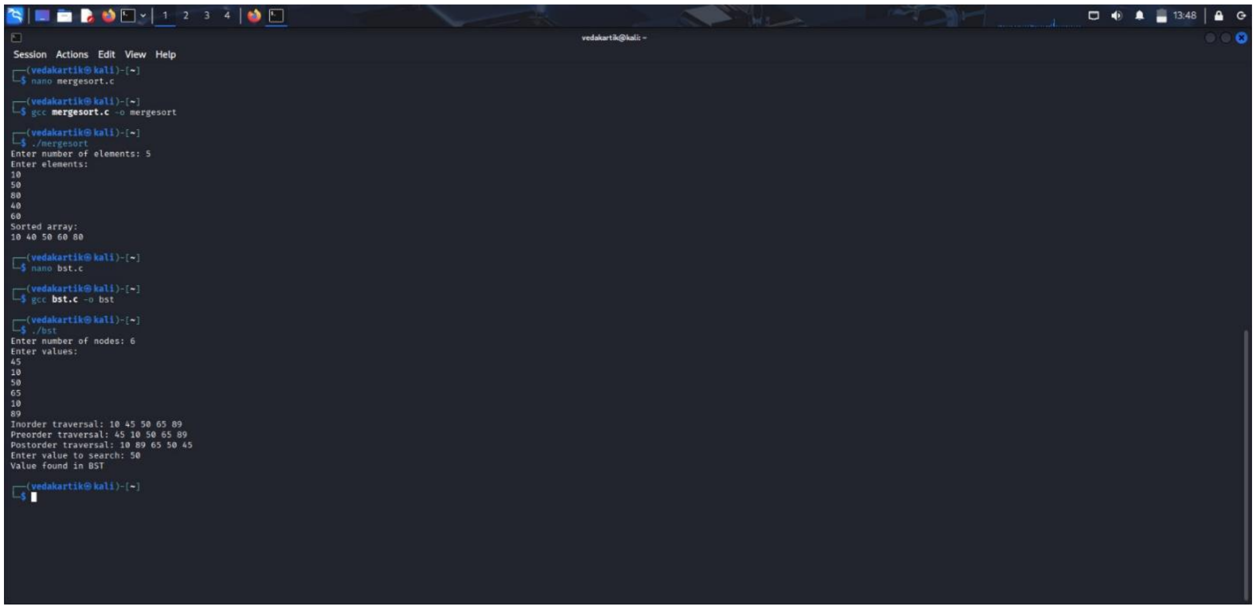
CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *left;
    struct Node *right;

};

struct Node* createNode(int value) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = value;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

struct Node* insert(struct Node* root, int value) {

    if (root == NULL) {

        return createNode(value);

    }

    if (value < root->data) {

        root->left = insert(root->left, value);

    } else if (value > root->data) {

        root->right = insert(root->right, value);

    }

    return root;

}

void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}
int search(struct Node* root, int key) {
    if (root == NULL)
        return 0;
    if (root->data == key)
        return 1;
    if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}
void freeTree(struct Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
int main() {
    struct Node* root = NULL;
    int n, value, key;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
```

```
printf("Enter values:\n");
for (int i = 0; i < n; i++) {
    scanf("%d", &value);
    root = insert(root, value);
}
printf("Inorder traversal: ");
inorder(root);
printf("\n");
printf("Preorder traversal: ");
preorder(root);
printf("\n");
printf("Postorder traversal: ");
postorder(root);
printf("\n")
printf("Enter value to search: ");
scanf("%d", &key);
if (search(root, key))
    printf("Value found in BST\n");
else
    printf("Value not found in BST\n")
freeTree(root);
```

```
    return 0;

}
```

OUTPUT:



```
┌──(vedakartik㉿kali)-[~]
└─$ nano mergesort.c

┌──(vedakartik㉿kali)-[~]
└─$ gcc mergesort.c -o mergesort

┌──(vedakartik㉿kali)-[~]
└─$ ./mergesort
Enter number of elements: 5
Enter elements:
10
50
80
40
60
Sorted array:
10 40 50 60 80

┌──(vedakartik㉿kali)-[~]
└─$ nano bst.c

┌──(vedakartik㉿kali)-[~]
└─$ gcc bst.c -o bst

┌──(vedakartik㉿kali)-[~]
└─$ ./bst
Enter number of nodes: 6
Enter values:
45
10
50
65
10
89
Inorder traversal: 10 45 50 65 89
Preorder traversal: 45 10 50 65 89
Postorder traversal: 10 89 65 50 45
Enter value to search: 50
Value found in BST

┌──(vedakartik㉿kali)-[~]
└─$
```

G VEDA KARTIK

CH.SC.U4CSE24212