D.Tarun venkat sai
AP19110010504

1.Write a C program to print preorder, inorder, and postorder traversal on Binary Tree.

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

void inorder(struct node* root){
	if(root == NULL) return;
	inorder(root->left);
	printf("%d ->", root->data);
	inorder(root->right);
}

void preorder(struct node* root){
	if(root == NULL) return;
	printf("%d ->", root->data);
	preorder(root->left);
	preorder(root->right);
}

void postorder(struct node* root) {
	if(root == NULL) return;
	postorder(root->left);
	postorder(root->right);
	printf("%d ->", root->data);
}
struct node *createNode(value)
{
	struct node* newNode = malloc(sizeof(struct node));
	newNode->data = value;
	newNode->left = NULL;
	newNode->right = NULL;

	return newNode;
}
void main()
{
	struct node* root = createNode(1);
	root->left=createNode(12);
	root->right=createNode(9);

	root->left->left=createNode(10);
	root->left->right=createNode(15);

	root->right->left=createNode(11);
	root->right->right=createNode(16);


	printf("Inorder traversal \n");
```

```
	inorder(root);

	printf("\nPreorder traversal \n");
	preorder(root);

	printf("\nPostorder traversal \n");
	postorder(root);
}
```

2.Write a C program to create (or insert) and inorder traversal on Binary Search Tree.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
	int key;
	struct node *left, *right;
};
struct node *newNode(int item)
{
	struct node *temp =  (struct node *)malloc(sizeof(struct node));
	temp->key = item;
	temp->left = temp->right = NULL;
	return temp;
}
void inorder(struct node *root)
{
	if (root != NULL)
	{
	inorder(root->left);
	printf("%d \n", root->key);
	inorder(root->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);
	return node;
}
int main()
{
	struct node *root = NULL;
	root = insert(root, 3);
	insert(root, 12);
	insert(root, 51);
	insert(root, 43);
	insert(root, 37);
	insert(root, 98);
	insert(root, 5);
	inorder(root);
	return 0;
}
```

3.Write a C program for linear search algorithm.

```
#include<stdio.h>
main()
{
	int a[10],n,i,se;
	printf("enter the number of variables to be used");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("enter the value of a[%d]",i);
		scanf("%d",&a[i]);
	}
	printf("enter the searching element ");
	scanf("%d",&se);
	for(i=0;i<n;i++)
	{
		if(a[i]==se)
		{
			printf("the element is found at position %d",i);
			break;
		}
	}
	return 0;
}
```

4.Write a C program for binary search algorithm.

```
#include<stdio.h>
main()
{
	int a[10],n,i,se,top,mid,bottom;
	printf("enter the number of variables to be used");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("enter the value of a[%d]",i);
		scanf("%d",&a[i]);
	}
	printf("enter the searching element ");
	scanf("%d",&se);
	top=0;
	bottom=n-1;
	while(top<=bottom)
	{
		mid=top+bottom/2;
		if(a[i]==se)
		{
			printf("the number is found at %d",mid);
		}
		else if(a[mid]>se)
		{
			bottom=mid-1;
		}
		else
		{
			top=mid+1;
			mid=top+bottom;
```

```
            }
        }
        if(top>bottom)
        {
            printf("the element is not found");
        }
        return 0;
}
```

5.Write a C program depth first search (DFS) using array.

```
#include<stdio.h>
int G[10][10],visited[10],n;
void DFS(int i)
{
    int j;
    printf("\n%d",i);
    visited[i]=1;
    for(j=0;j<n;j++)
    {
        if(!visited[j]&&G[i][j]==1)
        {
           DFS(j);
        }
    }
}
void main()
{
    int i,j;
    printf("Enter number of vertices:");
    scanf("%d",&n);
    printf("\nEnter adjacency matrix of the graph:");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&G[i][j]);
            for(i=0;i<n;i++)
            {
                visited[i]=0;
                DFS(0);
            }
        }

    }
}
```

6.Write a C program breath first search (BFS) using array.

```
#include<stdio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v)
{
    for (i=1;i<=n;i++)
    {
        if(a[v][i] && !visited[i])
```

```
            {
                q[++r]=i;
            }
            if(f<=r)
            {
                visited[q[f]]=1;
                bfs(q[f++]);
            }

        }
}
void main()
{
            int v;
            printf("\n Enter the number of vertices:");
            scanf("%d",&n);
            for (i=1;i<=n;i++)
            {                   q[i]=0;
                visited[i]=0;
            }
            printf("\n Enter graph data in matrix form:\n");
            for (i=1;i<=n;i++)
            {
               for (j=1;j<=n;j++)
               {
                  scanf("%d",&a[i][j]);
                  printf("\n Enter the starting vertex:");
                  scanf("%d",&v);
                  bfs(v);
                  printf("\n The node which are reachable are:\n");
               }
            }
            for (i=1;i<=n;i++)
            {
               if(visited[i])
               {
                 printf("%d\t",i);
               }
               else
               {
                 printf("\n Bfs is not possible");
               }
            }
}
```