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

Code:

```
#include <stdio.h>

#include <stdlib.h>


struct node

{

int data;

struct node* left;

struct node* right;

};


struct node* newNode(int data)

{

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

node->data = data;

node->left = NULL;

        node->right = NULL;


return(node);

}


int postorder(struct node* node)

{

        if (node == NULL)
        {
                return;
        }

postorder(node->left);

postorder(node->right);

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

return 0;
}


int inorder(struct node* node)
{
if (node == NULL)
{
    return;
}

inorder(node->left);

printf("%d ", node->data);
```

```
inorder(node->right);

return 0;
}

int preorder(struct node* node)
{
if (node == NULL)
{
return;
}

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

preorder(node->left);

preorder(node->right);

return 0;
}

int main()
{
struct node *root = newNode(3);
root->left = newNode(0);
```

```
root->right = newNode(1);

root->left->left = newNode(1);

root->left->right = newNode(2);



printf("\nPreorder traversal of binary tree is \n");

preorder(root);


printf("\nInorder traversal of binary tree is \n");

inorder(root);


printf("\nPostorder traversal of binary tree is \n");

postorder(root);


getchar();

return 0;

}
```

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

   Code:

```
#include <stdio.h>

#include <stdlib.h>

struct btnode {
```

```
int val;

struct btnode *leaf;

struct btnode *r;

}*root = NULL, *temp = NULL, *t2, *t1;


int insert();

int create();

int inorder(struct btnode *t);

int finding(struct btnode *t);


int flag = 1;

int main()

{

int choice;

printf("\n1 - Insert an element into tree\n2 - Inorder Traversal\n3 - exit\n");

while(1)

{

printf("\nEnter your choice : ");

scanf("%d", &choice);

if (choice==1){

insert();

}

else if (choice==2){

inorder(root);

}
else if(choice==3){

exit(0);

}

else{

printf("Invalid input");

}

}

return 0;

}

int insert() {

create();

if (root == NULL)

            root = temp;

else

            finding(root);

    return 0;

}

 int inorder(struct btnode *t) {

if (root == NULL)

{

printf("No elements in a tree to display");

}

if (t->leaf != NULL)

inorder(t->leaf);

printf("%d -> ", t->val);
```

```
if (t->r != NULL)

inorder(t->r);

return 0;

}




int finding(struct btnode *t) {

if ((temp->val > t->val) && (t->r != NULL))

finding(t->r);

else if ((temp->val > t->val) && (t->r == NULL))

t->r = temp;

else if ((temp->val < t->val) && (t->leaf != NULL))

finding(t->leaf);

else if ((temp->val < t->val) && (t->leaf == NULL))

t->leaf = temp;

return 0;

}

int create() {

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

temp = (struct btnode *)malloc(1*sizeof(struct btnode));

temp->val = data;

temp->leaf = temp->r = NULL;
```

```
return 0;

}
```

3. Write a C program for the linear search algorithm.

Code:

```
#include <stdio.h>

int main() {

int a[100], search, i, n;

printf("Enter numbers in array\n");

scanf("%d", &n);

printf("Enter %d’s number \n", n);

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

scanf("%d", &a[i]);



printf("Enter a number to search\n");

scanf("%d", &number);

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

   if (a[i] == number){

        printf("%d is there in the array and at location %d.\n", search, i+1);

        break;

          }

   }

if (i == n)

        printf("%d isn't there in the  in the array.\n", search);

  return 0;

}
```

4.Write a C program for binary search algorithm

Code:

```
#include<stdio.h>

int main() {

int n,k;

printf("Enter no. of elements in the array\n");

scanf("%d",&n);

printf("enter %d the numbers",n);

int a[50],i,temp,j,l,h,m,flag;

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

scanf("%d ",&a[i]);

}

printf("enter the element to search:");

scanf("%d",&k);

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

for(j=i+1;j<n;j++)      {

        if(a[i]>a[j])           {

        temp=a[i];

        a[i]=a[j];

        a[j]=temp;

        }       }    }

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

        printf("%d ", a[i]);

}

l=0;
h=n-1;

while(l<=h)      {

m=(l+h)/2;

if(k==a[m])      {

            flag=1;

            break;

            }

            else if(k<a[m])            {

 h=m-1;

}

else     {

l=m+1;

printf("%d",l);

}

}

if(flag==0)       {

            printf("%d  value not found\n",k);

}

else     {

            printf("%d value  found at %d position\n",k,m+1);

}

}
```