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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

typedef struct Binary_tree
{
        struct Binary_tree *left;
        int data;
        struct Binary_tree *right;
}node;

void crbtree(node *,int);
void preorder(node *);
void inorder(node *);
void postorder(node *);

void main()
{
        node *root=NULL;
        int e,ch;
        do
        {
                printf("\n....Binary Tree Traversals....");
                printf("\n1.Create\n2.Preorder\n3.Inorder\n4.Postorder\n5.Exit");
                printf("\nEnter ur Choice:");
                scanf("%d",&ch);

                if(ch==5)
                {
                        free(root);
                        exit(0);
                }

                switch(ch)
                {
                        case 1: printf("\nEnter Elements into Binary Tree:");
```

```
                        scanf("%d",&e);
                        root=(node *)malloc(sizeof(node));
                        crbtree(root,e);
                    break;
                    case 2: preorder(root);
                    break;
                    case 3: inorder(root);
                    break;
                    case 4: postorder(root);
                    break;
                    default:printf("\nYour Choice is Out of Range");
                }
        }while(1);
}

void preorder(node *root)
{
        if(root!=NULL)
        {
                printf("%3d",root->data);
                preorder(root->left);
                preorder(root->right);
        }
}

void inorder(node *root)
{
        if(root!=NULL)
        {
                inorder(root->left);
                printf("%3d",root->data);
                inorder(root->right);
        }
}

void postorder(node *root)
{
        if(root!=NULL)
        {
```

```
                postorder(root->left);
                postorder(root->right);
                printf("%3d",root->data);
        }
}

void crbtree(node *root,int e)
{
        char ch;
        if(root!=NULL)
        {
                root->data=e;
                printf("\nDo you want add as a Left Child(Y/N):");
                scanf("%c",&ch);
                if(ch=='y'||ch=='Y')
                {
                        root->left=(node *)malloc(sizeof(node));
                        printf("\nEnter Element:");
                        scanf("%d",&e);
                        crbtree(root->left,e);
                }
                else
                {
                    root->left=NULL;
                }
                printf("\nDo you want add as a Right Child(Y/N):");
                scanf("%c",&ch);
                if(ch=='y'||ch=='Y')
                {
                        root->right=(node *)malloc(sizeof(node));
                        printf("\nEnter Element:");
                        scanf("%d",&e);
                        crbtree(root->right,e);
                }
                else
                {
                    root->right=NULL;
                }
        }
```

```
}
```

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 a;
struct node* left;
struct node* right;
};
struct node *newNode(int item) {
struct node*temp = (struct node*)malloc(sizeof(struct node));
temp->a = item;
temp->left = temp->right = NULL;
return temp;
}
struct node* insert(struct node *node, int value){
if (node==NULL)return newNode(value);
if (value<node->a)
node->left = insert(node->left, value);
else if(value>node->a)
node->right = insert(node->right, value);
return node;
}
void inorder (struct node* root) {
if(root == NULL) return;
inorder(root->left);
printf("%d->", root->a);
inorder(root->right);
}
void main () {
struct node* root = NULL;
root = insert(root, 52);
insert(root, 35);
insert(root, 11);
insert(root, 45);
insert(root, 78);
```

```
insert(root, 89);
insert(root, 66);
printf("\n inorder traversal \n");
inorder(root);
}
```

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

```
#include<stdio.h>
int top=-1;
int x;
char stack[100];
void push(int x);
char pop();
int main()
{
int i,n,a,t,k,f,sum=0,count=1;
printf("Enter the number of elements in the stack");
scanf("%d",&n);
for(i=0;i<n;i++){
printf("Enter next element");
scanf("%d",&a);
push(a);
}
printf("Enter the sum to be checked");
scanf("%d",&k);
for(i=0;i<n;i++)
{
t=pop();
sum+=t;
count+=1;
if(sum==k){
for(int j=0;j<count;j++)
printf("%d",stack[j]);
f=1;
break;
}
push(t);
}
```

```
if(f!=1)
printf("The elements in the stack dont add up to the sum");
}

void push(int x)
{
if(top==99)
{
printf("\nStack is FULL !!!\n");
return;
}
top=top+1;
stack[top]=x;
}
char pop()
{
if(stack[top]==-1)
{
printf("\nStack is EMPTY!!!\n");
return 0;
}
x = stack[top];
top=top-1;
   return x;
}
```

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

```
#include<stdio.h>
#define SIZE 10
void insert(int);
void delete();
int queue[10], f=-1,r=-1;
void main() {
  int value, choice;
  while(1){
    printf("\n\n*** MENU ***\n");
    printf("1. Insertion\n2. Deletion\n3. Print Reverse\n4. Print Alternate\n5. Exit");
    printf("\nEnter your choice: ");
```

```
    scanf("%d",&choice);
    switch(choice){
case 1: printf("Enter the value to be insert: ");
scanf("%d",&value);
insert(value);
break;
case 2: delete();
break;
case 3:
            printf("The Reversed queue is:");
            for(int i=SIZE;i>=0;i--)
{
            if(queue[i]==0)
            continue;
            printf("%d ",queue[i]);
}
            break;
      case 4:
            printf("Alternate elements of the queue are:");
            for(int i=0;i<SIZE;i+=2)
{
            if(queue[i]==0)
            continue;
            printf("%d ",queue[i]);
}
            break;

case 5: exit(0);
default: printf("\nWrong selection!!! Try again!!!");
    }
}}
void insert(int value){
  if((f==0 &&r == SIZE-1) ||  f==r+1)
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else{
    if(f == -1)
f = 0;
    r=(r+1)%SIZE;
    queue[r] = value;
```

```
    printf("\nInsertion success!!!");
}}
void delete(){
  if(f == -1)
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", queue[f]);
    f=(f+1)%SIZE;
    if(f == r)
f = r = -1;
}
```

5. **Write a C program for linear search algorithm.**

```
#include <stdio.h>

long linear_search(long [], long, long);

int main()
{
  long array[100], search, b, n, position;

  printf("Enter number of elements in array\n");
  scanf("%ld", &n);

  printf("Input %d numbers\n", n);

  for (b = 0; b < n; b++)
    scanf("%ld", &array[b]);

  printf("Enter number to search\n");
  scanf("%ld", &search);

  position = linear_search(array, n, search);

  if (position == -1)
    printf("%d isn't present in the array.\n", search);
  else
    printf("%d is present at location %d.\n", search, position+1);
```

```
  return 0;
}

long linear_search(long a[], long n, long find) {
  long b;

  for (b = 0 ;b < n ; b++ ) {
    if (a[b] == find)
      return b;
  }

  return -1;
}
```

6. Write a C program for binary search algorithm.

```
#include <stdio.h>
int main()
{
  int a, first, last, middle, n, search, array[100];

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (a = 0; a < n; a++)
    scanf("%d", &array[a]);

  printf("Enter value to find\n");
  scanf("%d", &search);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
```

```
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);

  return 0;
}
```