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

```
//C program for different tree transversals
#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);
}

void printPostorder(struct node* node)
{
    if (node == NULL)
```

```
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    printf("%d ", node->data);
}


void printInorder(struct node* node)
{
    if (node == NULL)
         return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(node->right);
}
```

```
void printPreorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    printf("%d ", node->data);

    /* then recur on left sutree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}

int main()
{
    struct node *root  = newNode(5);
    root->left        = newNode(6);
    root->right       = newNode(7);
    root->left->left  = newNode(8);
    root->left->right  = newNode(9);

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

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

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

    getchar();
    return 0;
}
```

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

```
// C program to demonstrate insert operation in 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 the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}
```

```
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         / \    / \
       20   40  60   80 */
    struct node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // print inoder traversal of the BST
    inorder(root);

    return 0;
}
```

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

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

```
#include <stdio.h>
#include <stdlib.h>
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
   int j;
   visited[i]=1;
   printf(" %d->",i+1);
   for(j=0;j<V;j++)
   {
      if(G[i][j]==1&&visited[j]==0)
         DFS(j);
   }
}
int main()
{
   int i,j,v1,v2;
   printf("\t\t\tGraphs\n");
   printf("Enter the no of edges:");
   scanf("%d",&E);
   printf("Enter the no of vertices:");
   scanf("%d",&V);
   for(i=0;i<V;i++)
   {
      for(j=0;j<V;j++)
         G[i][j]=0;
   }
```

```
    /*    creating edges :P    */
    for(i=0;i<E;i++)
    {
        printf("Enter the edges (format: V1 V2) : ");
        scanf("%d%d",&v1,&v2);
        G[v1-1][v2-1]=1;

    }

    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
            printf(" %d ",G[i][j]);
        printf("\n");
    }
    printf("Enter the source: ");
    scanf("%d",&source);
        DFS(source-1);
    return 0;
}
```

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

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

```
#include<stdio.h>
int G[20][20],q[20],visited[20],n,front = 1, rear = 0 ;
```

```
void bfs(int v)
{
    int i;
    visited[v] = 1;
 for(i=1;i<=n;i++)
  if(G[v][i] && !visited[i])
   q[++rear]=i;
   if(front <= rear)
    bfs(q[front++]);
 }

int main()
{
 int v,i,j;

 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",&G[i][j]);
 printf("\n Enter the starting vertex:");
 scanf("%d",&v);
 bfs(v);
 printf("\n The nodes which are reachable are:\n");
 for(i=1;i<=n;i++)
  if(visited[i])
   printf("%d\t",i);
  else
   printf("\n %d is not reachable",i);

return 0;
}
```

5)Write a C program for linear search algorithm.

```
#include <stdio.h>

int search(int arr[], int n, int x)
{
    int i;
```

```
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

int main(void)
{
    int arr[] = { 2, 13, 46, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = search(arr, n, x);
    (result == -1) ? printf("Element is not present in array")
                   : printf("Element is present at index %d",
                            result);
    return 0;
}
```

6) Write a C program for binary search algorithm.

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

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

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

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

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;
}
```