**Name:** R. SHYAM **Register Number:** 191911031

**Subject code:** CSA0356 **Subject Name :** Data Structure for Stack Management

**Experiment 1**

**Write a C program to perform Matrix Multiplication**

**Program**

#include<stdio.h>

#include<stdlib.h>

int main(){

int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;

printf("enter the number of row=");

scanf("%d",&r);

printf("enter the number of column=");

scanf("%d",&c);

printf("enter the first matrix element=\n");

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

{

for(j=0;j<c;j++)

{

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

}

}

printf("enter the second matrix element=\n");

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

{

for(j=0;j<c;j++)

{

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

}

}

printf("multiply of the matrix=\n");

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

{

for(j=0;j<c;j++)

{

mul[i][j]=0;

for(k=0;k<c;k++)

{

mul[i][j]+=a[i][k]\*b[k][j];

}

}

}

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

{

for(j=0;j<c;j++)

{

printf("%d\t",mul[i][j]);

}

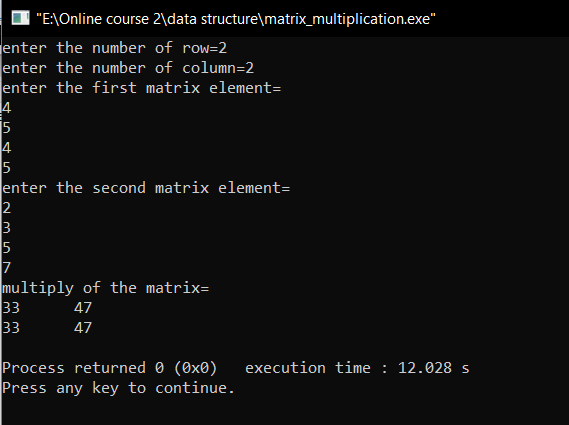
printf("\n");

}

return 0;

}

**Output**



**Experiment 2**

**Write a C program to find Odd or Even number from a given set of numbers**

**Program**

#include<stdio.h>

int main()

{

int n, a[20];

printf("Enter the size of the array: ");

scanf("%d", &n);

printf("Enter array elements: \n");

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

{

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

}

printf("Even numbers in the array are: \n");

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

{

if(a[i]%2==0)

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

}

printf("\nOdd numbers in the array are: \n");

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

{

if(a[i]%2!=0)

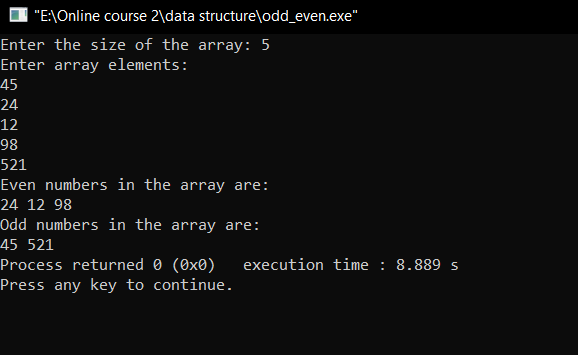
printf("%d ", a[i]);

}

return 0;

}

**Output**



**Experiment 3**

**Write a C program to find Factorial of a given number without using Recursion**

**Program**

#include<stdio.h>

int main()

{

int i, num, fact = 1;

printf("\nEnter the number: ");

scanf("%d", &num);

for (i=1;i<=num;i++)

{

fact=fact\*i;

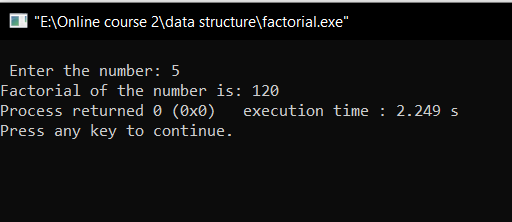
}

printf("Factorial of the number is: %d",fact);

return 0;

}

**Output**



**Experiment 4**

**Write a C program to find Fibonacci series without using Recursion**

**Program**

#include <stdio.h>

void main()

{

int i,n,t1=0,t2=1,nextterm;

printf("enter the number of terms \n");

scanf("%d",&n);

printf("fibonacci series are \n");

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

{

printf("%d \n",t1);

nextterm=t1+t2;

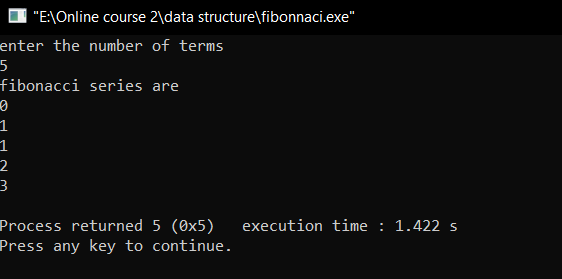
t1=t2;

t2=nextterm;

}

}

**Output**



**Experiment 5**

**Write a C program to find Factorial of a given number using Recursion**

**Program**

#include<stdio.h>

int fact(int n);

int main()

{

int n;

printf("\n Enter the number:");

scanf("%d", &n);

printf("\n Factorial of the given number is: %d",fact(n));

return 0;

}

int fact(int n)

{

if(n>=1)

{

return n\*fact(n-1);

}

else

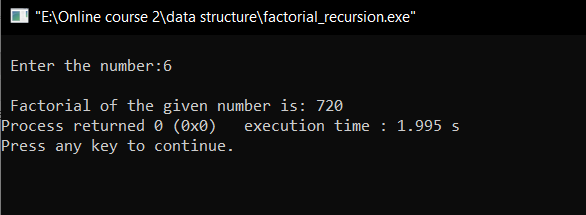
{

return 1;

}

}

**Output**



**Experiment 6**

**Write a C program to find Fibonacci series using Recursion**

**Program**

#include<stdio.h>

void Fibonacci(int n){

static int n1=0,n2=1,n3;

if(n>0){

n3 = n1 + n2;

n1 = n2;

n2 = n3;

printf("%d ",n3);

Fibonacci(n-1);

}

}

int main(){

int n;

printf("Enter the number of elements: ");

scanf("%d",&n);

printf("Fibonacci Series: ");

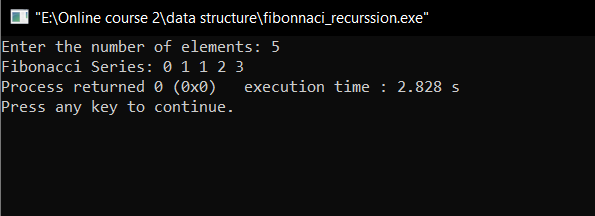
printf("%d %d ",0,1);

Fibonacci(n-2);

return 0;

}

**Output**



**Experiment 7**

**Write a C program to implement array operations such as Insert, Delete and Display.**

**Program**

#include<stdio.h>

int main(){

int arr[40],position,pos,i,size,value;

printf("Enter number of elements in array:");

scanf("%d",&size);

printf("Enter %d elements are:\n",size);

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

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

printf("Enter the position where you want to insert the element:");

scanf("%d",&pos);

printf("enter the value into that position:");

scanf("%d",&value);

for(i=size-1;i>=pos-1;i--)

arr[i+1]=arr[i];

arr[pos-1]= value;

printf("After inserting the value is\n");

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

printf("%d\n",arr[i]);

printf("Enter the location where you wish to delete element\n");

scanf("%d", &position);

if (position >= size+1)

printf("Deletion not possible.\n");

else

{

for (i = position - 1; i < size - 1; i++)

arr[i] = arr[i+1];

printf("After deleting the value:\n");

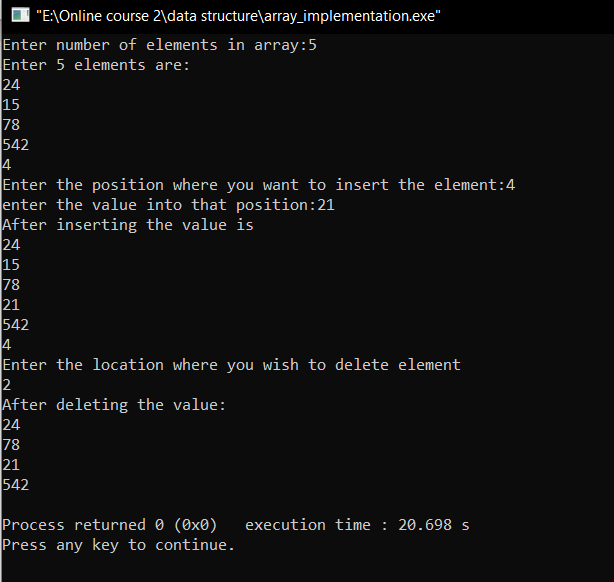
for (i = 0; i < size - 1; i++)

printf("%d\n", arr[i]);

}

return 0;

}

**Output**

**Experiment 8**

**Write a C program to search a number using Linear Search method**

**Program**

#include <stdio.h>

int main()

{

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

printf("Enter number of elements in array\n");

scanf("%d", &n);

printf("Enter %d integer(s)\n", n);

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

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

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

scanf("%d", &search);

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

{

if (array[i] == search)

{

printf("The element %d is present at location %d.\n", search, i+1);

break;

}

}

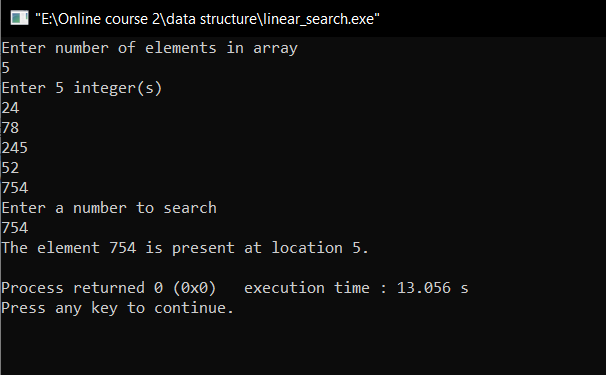
if (i == n)

printf("The element %d is not present in the array.\n", search);

return 0;

}

**Output**



**Experiment 9**

**Write a C program to search a number using Binary Search method**

**Program**

#include <stdio.h>

int main()

{

int i, low, high, mid, n, key, array[100];

printf("Enter number of elements:");

scanf("%d",&n);

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

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

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

printf("Enter the search element:");

scanf("%d", &key);

low = 0;

high = n - 1;

mid = (low+high)/2;

while (low <= high) {

if(array[mid] < key)

low = mid + 1;

else if (array[mid] == key) {

printf("The element %d found at location %d", key, mid+1);

break;

}

else

high = mid - 1;

mid = (low + high)/2;

}

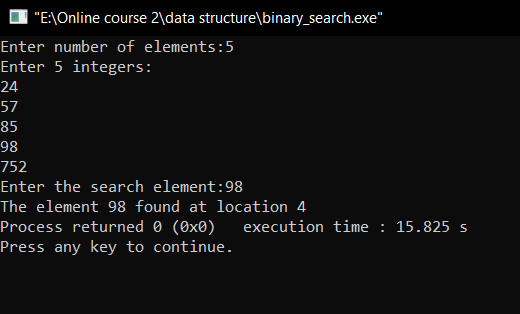
if(low > high)

printf("The element %d is not present in the given array", key);

return 0;

}

**Output**



**Experiment 10**

**Write a C program to implement Stack operations such as PUSH, POP and PEEK**

**Program**

#include<stdio.h>

#include<stdlib.h>

#define Size 4

int Top=-1, inp\_array[Size];

void Push();

void Pop();

void show();

int main()

{

int choice;

while(1)

{

printf("\nOperations performed by Stack");

printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");

printf("\n\nEnter the choice:");

scanf("%d",&choice);

switch(choice)

{

case 1: Push();

break;

case 2: Pop();

break;

case 3: show();

break;

case 4: exit(0);

default: printf("\nInvalid choice!!");

}

}

}

void Push()

{

int x;

if(Top==Size-1)

{

printf("\nOverflow!!");

}

else

{

printf("\nEnter element to be inserted to the stack:");

scanf("%d",&x);

Top=Top+1;

inp\_array[Top]=x;

}

}

void Pop()

{

if(Top==-1)

{

printf("\nUnderflow!!");

}

else

{

printf("\nPopped element: %d",inp\_array[Top]);

Top=Top-1;

}

}

void show()

{

if(Top==-1)

{

printf("\nUnderflow!!");

}

else

{

printf("\nElements present in the stack: \n");

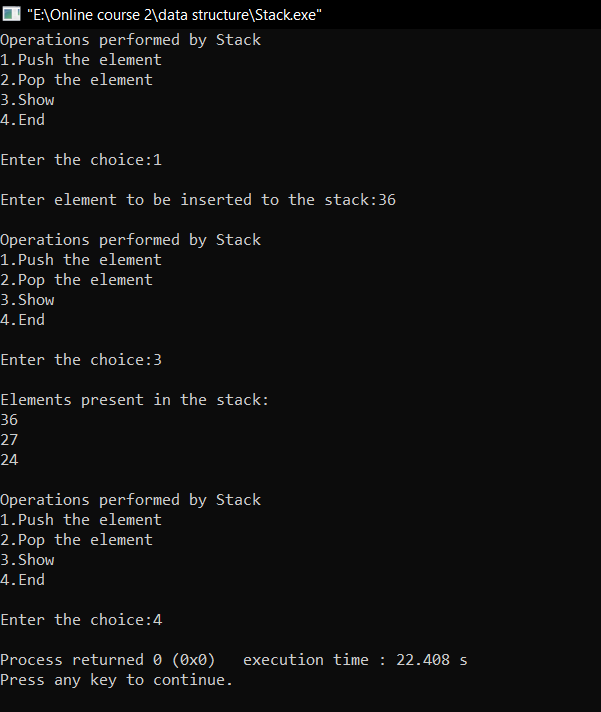
for(int i=Top;i>=0;--i)

printf("%d\n",inp\_array[i]);

}

}

**Output**



**Experiment 11**

**Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE and Display**

**Program**

#include <stdio.h>

#define SIZE 100

void enqueue();

void dequeue();

void show();

int inp\_arr[SIZE];

int Rear = - 1;

int Front = - 1;

main()

{

int ch;

while (1)

{

printf("1.Enqueue Operation\n");

printf("2.Dequeue Operation\n");

printf("3.Display the Queue\n");

printf("4.Exit\n");

printf("Enter your choice of operations : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

show();

break;

case 4:

exit(0);

default:

printf("Incorrect choice \n");

}

}

}

void enqueue()

{

int insert\_item;

if (Rear == SIZE - 1)

printf("Overflow \n");

else

{

if (Front == - 1)

Front = 0;

printf("Element to be inserted in the Queue\n : ");

scanf("%d", &insert\_item);

Rear = Rear + 1;

inp\_arr[Rear] = insert\_item;

}

}

void dequeue()

{

if (Front == - 1 || Front > Rear)

{

printf("Underflow \n");

return ;

}

else

{

printf("Element deleted from the Queue: %d\n", inp\_arr[Front]);

Front = Front + 1;

}

}

void show()

{

if (Front == - 1)

printf("Empty Queue \n");

else

{

printf("Queue: \n");

for (int i = Front; i <= Rear; i++)

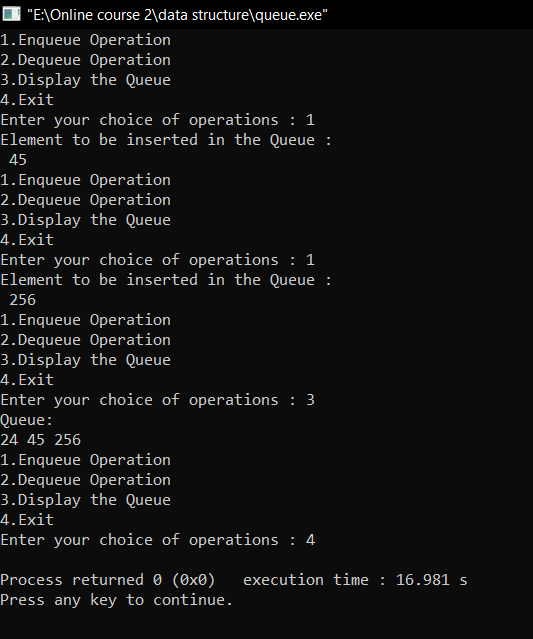
printf("%d ", inp\_arr[i]);

printf("\n");

}

}

**Output**



**Experiment 12**

**Write a C program to arrange a series of numbers using Insertion Sort**

**Program**

#include <stdio.h>

void printArray(int array[], int size) {

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

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

}

printf("\n");

}

void insertionSort(int array[], int size) {

for (int step = 1; step < size; step++) {

int key = array[step];

int j = step - 1;

while (key < array[j] && j >= 0) {

array[j + 1] = array[j];

--j;

}

array[j + 1] = key;

}

}

int main() {

int data[] = {10, 45, 7, 52, 58};

int size = sizeof(data) / sizeof(data[0]);

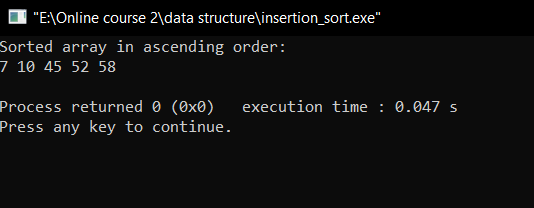
insertionSort(data, size);

printf("Sorted array in ascending order:\n");

printArray(data, size);

}

**Output**



**Experiment 13**

**Write a C program to implement Linked list operations**

**Program**

#include <stdio.h>

#include <stdlib.h>

struct node {

int value;

struct node \*next;

};

void printLinkedlist(struct node \*p) {

while (p != NULL) {

printf("%d ", p->value);

p = p->next;

}

}

int main() {

struct node \*head;

struct node \*one = NULL;

struct node \*two = NULL;

struct node \*three = NULL;

one = malloc(sizeof(struct node));

two = malloc(sizeof(struct node));

three = malloc(sizeof(struct node));

one->value = 1;

two->value = 2;

three->value = 3;

one->next = two;

two->next = three;

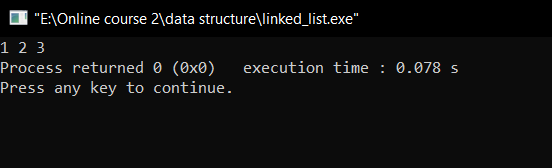
three->next = NULL;

head = one;

printLinkedlist(head);

}

**Output**



**Experiment 14**

**Write a C program to implement the tree Traversals (Inorder, Preorder, Postorder)**

**Program**

#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;

printPostorder(node->left);

printPostorder(node->right);

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

}

void printInorder(struct node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

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

printInorder(node->right);

}

void printPreorder(struct node\* node)

{

if (node == NULL)

return;

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

printPreorder(node->left);

printPreorder(node->right);

}

int main()

{

struct node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

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");

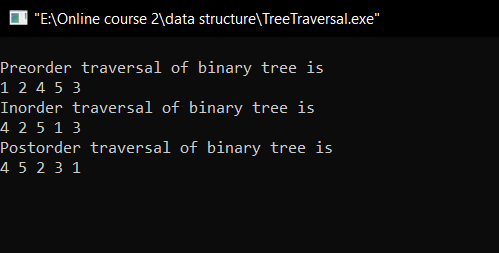
printPostorder(root);

getchar();

return 0;

}

**Output**



**Experiment 15**

**Write a C program to implement hashing using Linear Probing method**

**Program**

#include <stdio.h>

#include<stdlib.h>

#define TABLE\_SIZE 10

int h[TABLE\_SIZE]={NULL};

void insert()

{

int key,index,i,flag=0,hkey;

printf("\nenter a value to insert into hash table\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

for(i=0;i<TABLE\_SIZE;i++)

{

index=(hkey+i)%TABLE\_SIZE;

if(h[index] == NULL)

{

h[index]=key;

break;

}

}

if(i == TABLE\_SIZE)

printf("\nelement cannot be inserted\n");

}

void search()

{

int key,index,i,flag=0,hkey;

printf("\nenter search element\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

for(i=0;i<TABLE\_SIZE; i++)

{

index=(hkey+i)%TABLE\_SIZE;

if(h[index]==key)

{

printf("value is found at index %d",index);

break;

}

}

if(i == TABLE\_SIZE)

printf("\n value is not found\n");

}

void display()

{

int i;

printf("\nelements in the hash table are \n");

for(i=0;i< TABLE\_SIZE; i++)

printf("\nat index %d \t value = %d",i,h[i]);

}

main()

{

int opt,i;

while(1)

{

printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");

scanf("%d",&opt);

switch(opt)

{

case 1:

insert();

break;

case 2:

display();

break;

case 3:

search();

break;

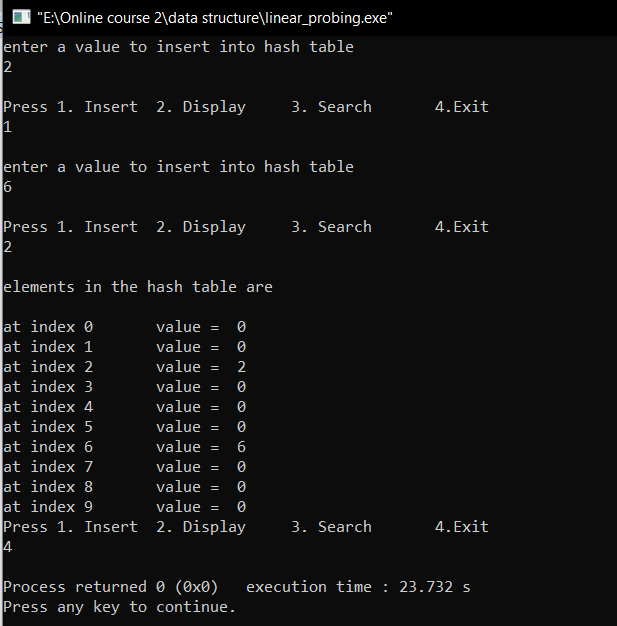
case 4:exit(0);

}

}

}

**Output**



**Experiment 16**

**Write a C program to arrange a series of numbers using Quick Sort**

**Program**

#include<stdio.h>

void quicksort(int number[25],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main(){

int i, count, number[25];

printf("Enter number of elements");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

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

quicksort(number,0,count-1);

printf("Order of Sorted elements: ");

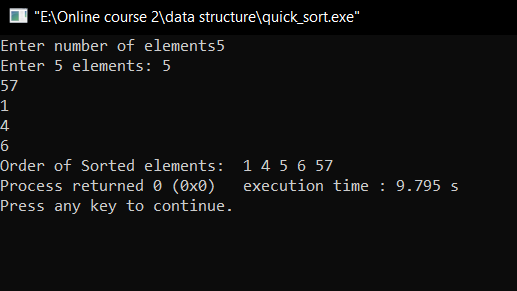
for(i=0;i<count;i++)

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

return 0;

}

**Output**



**Experiment 17**

**Write a C program to Graph traversal using Breadth First Search and Depth First Search**

**DFS Program**

#include<stdio.h>

void DFS(int);

int G[10][10],visited[10],n;

void main()

{

int i,j;

printf("Enter number of vertices:");

scanf("%d",&n);

printf("\nEnter adjecency 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);

}

void DFS(int i)

{

int j;

printf("\n%d",i);

visited[i]=1;

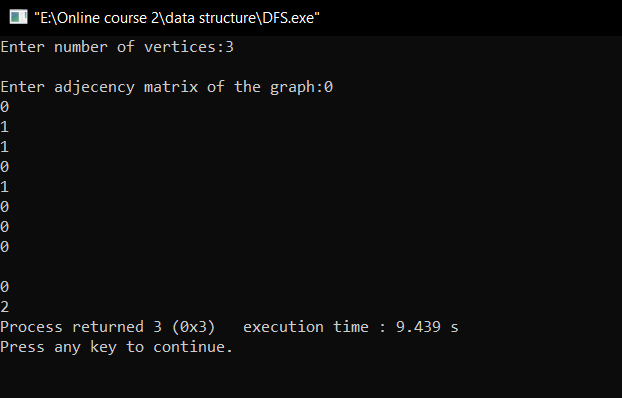
for(j=0;j<n;j++)

if(!visited[j]&&G[i][j]==1)

DFS(j);

}

**Output**



**BFS Program**

#include<stdio.h>

#include<conio.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("Enter the number of vertices: ");

scanf("%d",&n);

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

q[i] = 0;

visited[i] = 0;

}

printf("\nEnter graph data in matrix form:\n");

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

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

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

}

}

printf("Enter the starting vertex: ");

scanf("%d", &v);

bfs(v);

printf("\nThe node which are reachable are:");

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

if(visited[i])

printf(" %d", i);

else {

printf("\nBFS is not possible. All nodes are not reachable!");

break;

}

}

getch();

}

**Output**

