VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

DEEPINI S (1BM21CS050)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **Deepini S (1BM21CS050),** who is a bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Radhika A D
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

WEEK 1

Write program to do the following:

a) Print all the nodes reachable from a given starting node in a digraph using BFS method.

b) Check whether a given graph is connected or not using DFS method.

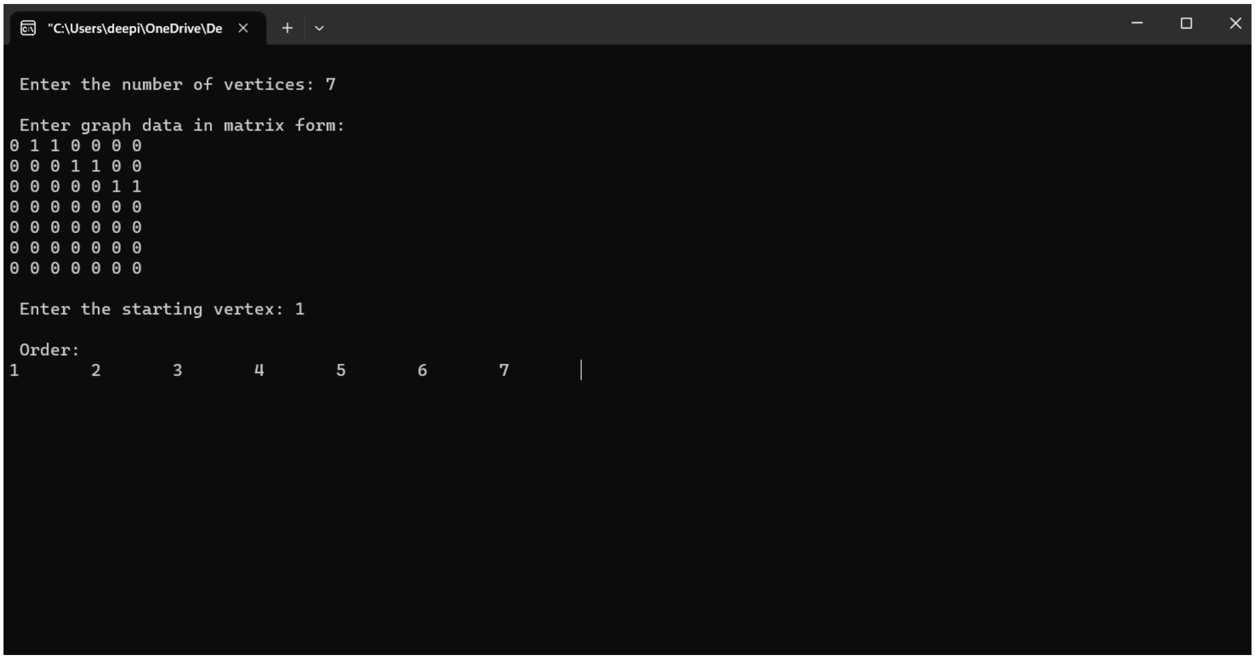
BFS

CODE:

```
 #include<stdio.h>

 #include<math.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 Order:\n");
 printf("%d\t",v);
```

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

if(visited[i])

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

}
```

OUTPUT :

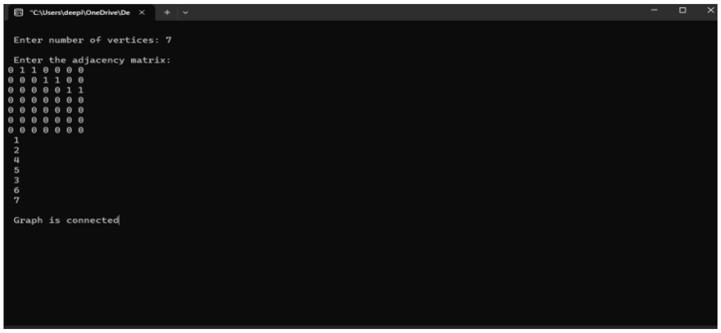


```
 Enter the number of vertices: 7

 Enter graph data in matrix form:
0 1 1 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 0 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

 Enter the starting vertex: 1

 Order:
1       2       3       4       5       6       7
```

DFS

```
#include<stdio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
        int i;
        reach[v]=1;
        for(i=1;i<=n;i++)
        if(a[v][i] && !reach[i])
        {
        printf("\n %d",i);
        dfs(i);
        }
}
void main()
{
        int i,j,v,count=0, f=1;
        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
           reach[i]=0;
          for(j=1;j<=n;j++)
             a[i][j]=0;}
        printf("\n Enter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
        printf(" %d",f);
        dfs(1);
        for(i=1;i<=n;i++)
        {
           if(reach[i])
           count++;
        }
        if(count==n)
        printf("\n Graph is connected");
        else
        printf("\n Graph is not connected");
}
```

OUTPUT :



```
 Enter number of vertices: 7

 Enter the adjacency matrix:
0 1 1 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 0 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
 1
 2
 4
 5
 3
 6
 7

 Graph is connected
```

WEEK 2

Write a program to obtain the Topological ordering of vertices in a given digraph.

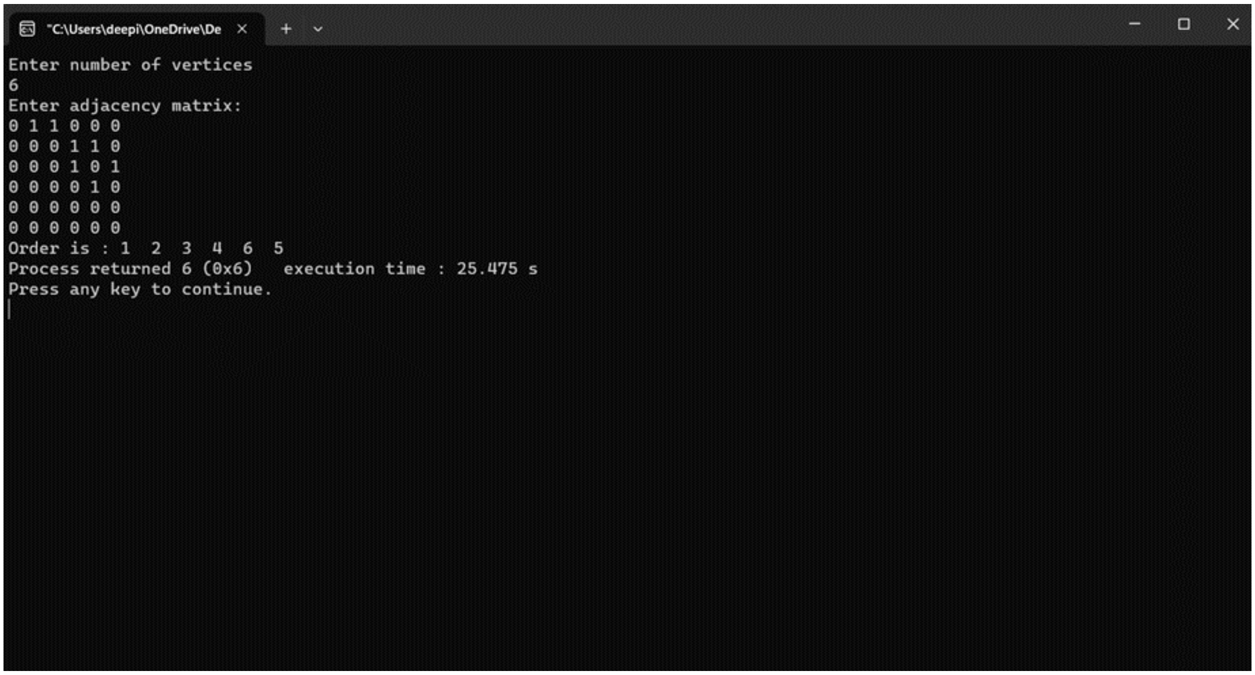
CODE:

```
#include<stdio.h>

 #include<conio.h>

void main()

{

int a[10][10],n,i,j;

int indeg[10],flag[10],c=0;

printf("Enter number of vertices \n");

scanf("%d",&n);

printf("Enter adjacency matrix: \n");

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

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

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

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

   indeg[i]=0;

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

```
   flag[i]=0;
for(i=0;i<n;i++)
     for(j=0;j<n;j++)
         if(a[i][j]==1)
            indeg[j]+=1;
printf("Order is : ");
 while(c<=n)
{
for(i=0;i<n;i++)
{
if(indeg[i]==0 && flag[i]==0)
{
printf("%d ",i+1);
flag[i]=1;
}
}
for(i=0;i<n;i++)
{
if(flag[i]==1)
{
for(j=0;j<n;j++)
{
```

```
if(a[i][j]==1)
{
indeg[j]-=1;
a[i][j]=0;
}
c++;
}
}
```

OUTPUT :



```
Enter number of vertices
6
Enter adjacency matrix:
0 1 1 0 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 1 0
0 0 0 0 0 0
0 0 0 0 0 0
Order is : 1  2  3  4  6  5
Process returned 6 (0x6)   execution time : 25.475 s
Press any key to continue.
```

WEEK 3

Implement Johnson Trotter algorithm for permutations

CODE :

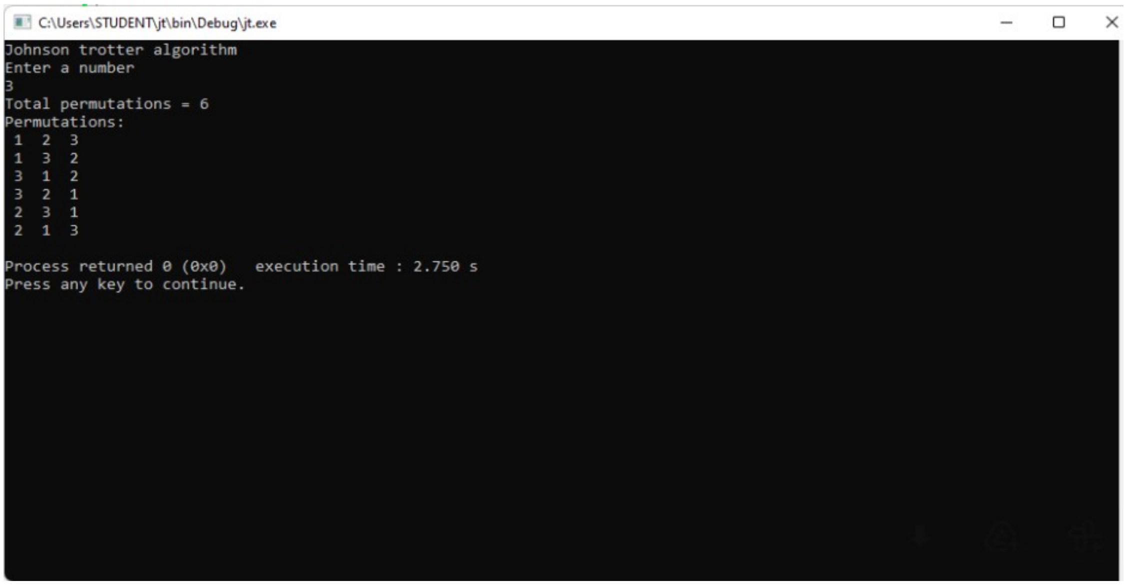
```
#include <stdio.h>
 #include <stdlib.h>
 int flag = 0;
int swap(int *a,int *b)
{
int t = *a;
*a = *b;
*b = t;
}

int search(int arr[],int n,int mobile)
{
int g;
for(g=0;g<n;g++)
{
 if(arr[g] == mobile)
return g+1;
else
flag++;
}
return -1;
}
int fm(int arr[],int d[],int n)
{
int mobile = 0; int mp = 0;
int i;
for(i=0;i<n;i++)
```

```
{
if((d[arr[i]-1] == 0) && i != 0)
{
if(arr[i]>arr[i-1] && arr[i]>mp)
{
mobile = arr[i]; mp = mobile;
}
else
flag++;
}
else if((d[arr[i]-1] == 1) && i != n-1)
{
if(arr[i]>arr[i+1] && arr[i]>mp)
    mobile = arr[i]; mp = mobile;
else
flag++;
}
else
flag++;

}
if((mp == 0) && (mobile == 0))
return 0;
else
return mobile;
}

void permut(int arr[],int d[],int n)
{
int i;
int mobile = fm(arr,d,n);
int pos = search(arr,n,mobile);
if(d[arr[pos-1]-1]==0)
swap(&arr[pos-1],&arr[pos-2]);
 else
swap(&arr[pos-1],&arr[pos]);
for(int i=0;i<n;i++)
```

```
{
if(arr[i] > mobile)
{
if(d[arr[i]-1]==0)
d[arr[i]-1] = 1;
 else
d[arr[i]-1] = 0;
}
}
for(i=0;i<n;i++)
{
printf(" %d ",arr[i]);
} }
int fact(int k)
{
int f = 1; int i = 0;
for(i=1;i<k+1;i++)
   f = f*i;
return f;
}

int main()
{
int n = 0; int i;
int j;
int z = 0;
printf("Johnson trotter algorithm \n");
printf("Enter a number\n"); scanf("%d",&n);
int arr[n],d[n];
z = fact(n);
printf("Total permutations = %d",z);
printf("\nPermutations: \n"); for(i=0;i<n;i++)
{
d[i] = 0;
arr[i] = i+1;
printf(" %d ",arr[i]);
}
```

```
printf("\n");
for(j=1;j<z;j++)
{
permut(arr,d,n);
printf("\n");
}
return 0;
}
```

OUTPUT :



```
Johnson trotter algorithm
Enter a number
3
Total permutations = 6
Permutations:
 1  2  3
 1  3  2
 3  1  2
 3  2  1
 2  3  1
 2  1  3

Process returned 0 (0x0)   execution time : 2.750 s
Press any key to continue.
```

WEEK 4

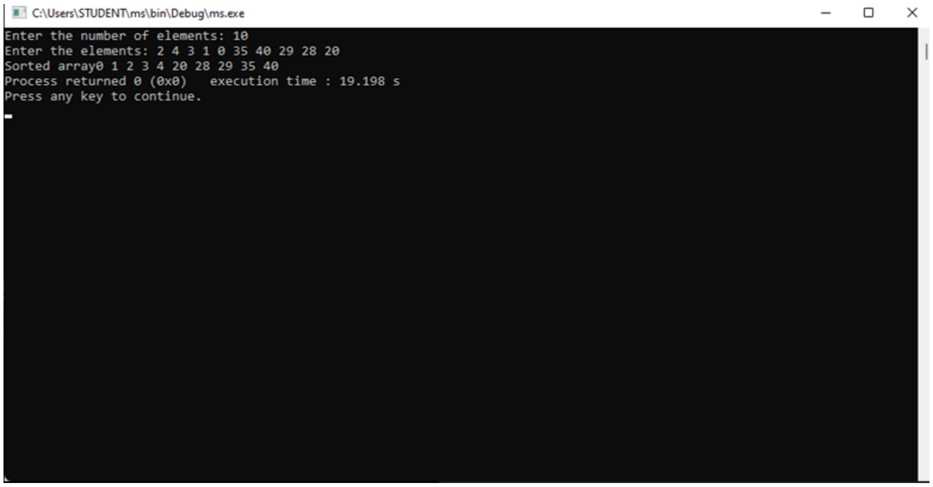
Sort a given set of N integer elements using Merge Sort technique

CODE:

```
#include <stdio.h>
 #include <stdlib.h>
void merge(int low,int mid,int high,int a[20],int m[20])
{
int i = low; int j = mid+1; int k = 0;
while(i<=mid && j<=high)
{
if(a[i]<a[j])
{
m[k] = a[i];
 i++;
k++;
}
else
{
m[k] = a[j];
j++;
k++;
}
}
while (i <= mid)
{
m[k] = a[i];
i++;
k++;
}
while (j <= high)
{
m[k] = a[j];
j++;
k++;
}
for(int i=0;i<k;i++)
    a[low+i] = m[i];
}
void merge_sort(int low,int high,int a[20],int merged[20])
{
if(low<high)
{
int mid = (low+high)/2;
merge_sort(low,mid,a,merged);
merge_sort(mid+1,high,a,merged);
 merge(low,mid,high,a,merged);
}
}
```

```
int main()
{
int n,a[30];
printf("Enter the number of elements:");
scanf("%d",&n);
printf("Enter the elements:");
for(int i=0;i<n;i++)
scanf("%d",&a[i]);
int merged[30];
merge_sort(0,n-1,a,merged);
printf("Sorted array");
for(int i=0;i<n;i++)
   printf("%d ",a[i]);
}
```

OUTPUT :



```
Enter the number of elements: 10
Enter the elements: 2 4 3 1 0 35 40 29 28 20
Sorted array0 1 2 3 4 20 28 29 35 40
Process returned 0 (0x0)   execution time : 19.198 s
Press any key to continue.
```

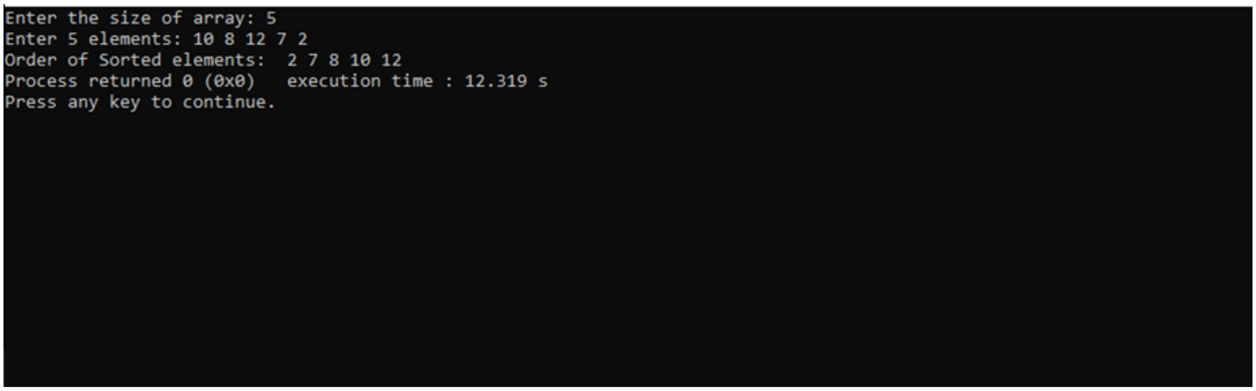
WEEK 5

Sort a given set of N integer elements using Quick Sort technique

CODE :

```
#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 the size of array: ");
 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 :



```
Enter the size of array: 5
Enter 5 elements: 10 8 12 7 2
Order of Sorted elements:  2 7 8 10 12
Process returned 0 (0x0)   execution time : 12.319 s
Press any key to continue.
```

WEEK 6

Implement 0/1 Knapsack problem using dynamic programming

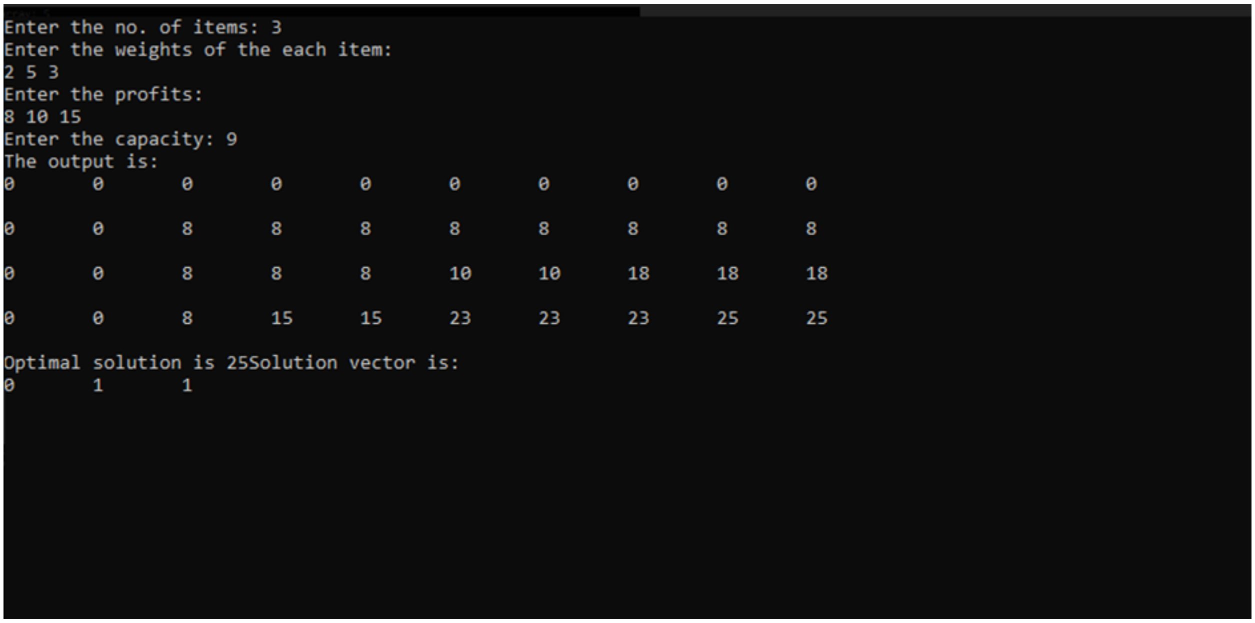
CODE :

```
#include<stdio.h>
#include<conio.h>
int i,j,n,m,p[10],w[10],v[10][10];
void main()
{
printf("Enter the no. of items:\t");
scanf("%d",&n);
printf("Enter the weights of the each item:\n");
 for(i=1;i<=n;i++)
scanf("%d",&w[i]);
printf("Enter the profits:\n");
for(i=1;i<=n;i++)
scanf("%d",&p[i]);
printf("Enter the capacity:");
scanf("%d",&m);
knapsack();
getch();
}
void knapsack()
{
int x[10];
for(i=0;i<=n;i++)
{

for(j=0;j<=m;j++)
{
```

```
if(i==0||j==0)
{
v[i][j]=0;
}
else if(j-w[i]<0)
{
v[i][j]=v[i-1][j];
}
else
{
v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);
}
}
}
printf("The output is:\n");
for(i=0;i<=n;i++)
{
for(j=0;j<=m;j++)
  printf("%d\t",v[i][j]);
printf("\n\n");
}
printf("Optimal solution is %d",v[n][m]);
 printf("Solution vector is:\n");
for(i=n;i>=1;i--)
{
if(v[i][m]!=v[i-1][m])
{
x[i]=1;
m=m-w[i];
}
else
x[i]=0;
}
for(i=1;i<=n;i++)
```

```
printf("%d\t",x[i]);
}
int max(int x,int y)
{
if(x>y)
return x;
else
return y;
}
```

OUTPUT :



```
Enter the no. of items: 3
Enter the weights of the each item:
2 5 3
Enter the profits:
8 10 15
Enter the capacity: 9
The output is:
0       0       0       0       0       0       0       0       0       0

0       0       8       8       8       8       8       8       8       8

0       0       8       8       8       10      10      18      18      18

0       0       8       15      15      23      23      23      25      25

Optimal solution is 25Solution vector is:
0       1       1
```

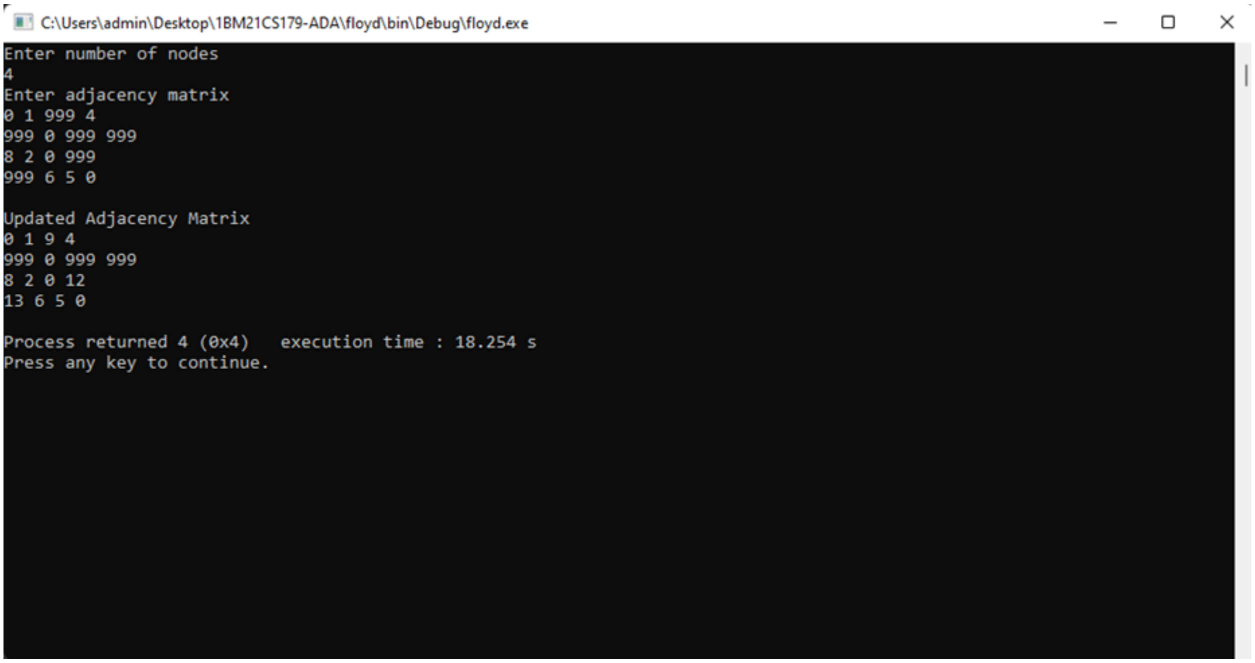
WEEK 7

Implement All Pair Shortest paths problem using Floyd's algorithm.

CODE :

```
#include<stdio.h>
 void main()
{
int i,j,k,n,adj[10][10],ori[10][10];
 printf("Enter number of nodes \n");
scanf("%d",&n);
printf("Enter adjacency matrix \n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
scanf("%d",&p[i][j]);
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
ori[i][j]=adj[i][j];
 for(k=0;k<n;k++)
 for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(adj[i][j] > adj[k][j]+adj[i][k])
adj[i][j]=adj[k][j]+adj[i][k];
printf("\nUpdated Matrix \n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
 printf("%d ",adj[i][j]);
}
}
```

OUTPUT :


C:\Users\admin\Desktop\1BM21CS179-ADA\floyd\bin\Debug\floyd.exe

```
Enter number of nodes
4
Enter adjacency matrix
0 1 999 4
999 0 999 999
8 2 0 999
999 6 5 0

Updated Adjacency Matrix
0 1 9 4
999 0 999 999
8 2 0 12
13 6 5 0

Process returned 4 (0x4)   execution time : 18.254 s
Press any key to continue.
```

WEEK 8

Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm

Prim’s

CODE :

```
 #include<stdio.h>
 float cost[10][10];
int vt[10],et[10][10],vis[10],j,n, x=1, e=0;
float sum=0;
void main()
{
int i;
printf("Enter the number of vertices\n");
scanf("%d",&n);
printf("Enter the cost adjacency matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
    scanf("%f",&cost[i][j]);
```

```
vis[i]=0;
prims();
printf("Edges of spanning tree\n");
for(i=1;i<=e;i++)
printf("%d,%d\t",et[i][0],et[i][1]);
printf("Weight=%f\n",sum);
}
void prims()
{
int s,m,k,u,v; float min; vt[x]=1;
vis[x]=1;
for(s=1;s<n;s++)
{
j=x;
min=999;
while(j>0)
{
k=vt[j];
for(m=2;m<=n;m++)
{
if(vis[m]==0)
{
if(cost[k][m]<min)
{
min=cost[k][m];
u=k;
v=m;
}
}
}
j--;
}
vt[++x]=v;
et[s][0]=u;
et[s][1]=v;
e++;
vis[v]=1;
sum=sum+min;
}
}
```

Kruskal’s

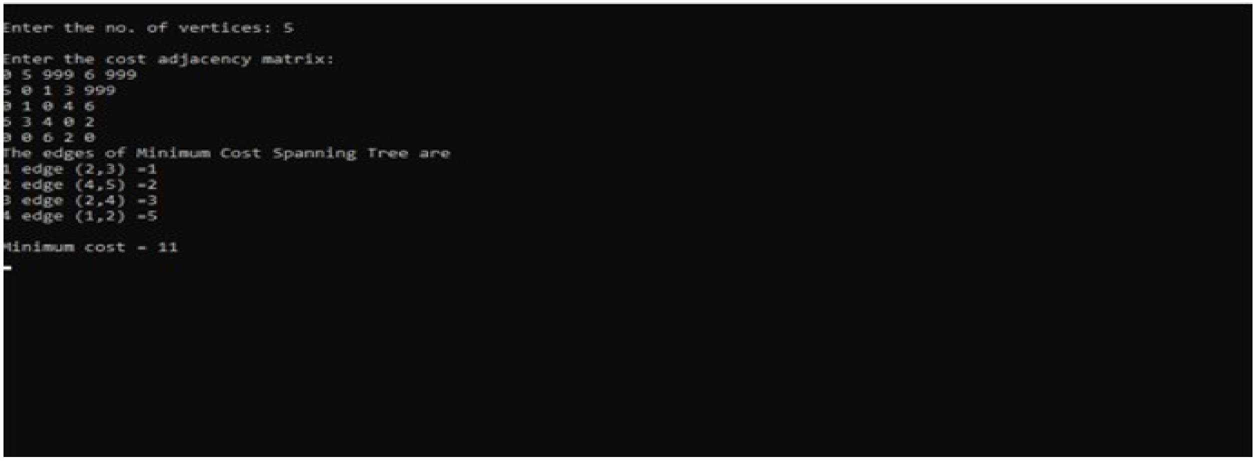
CODE :

```
#include <stdio.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
void main()
{
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
   cost[i][j]=999;
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
for(i=1,min=999;i<=n;i++)
{
```

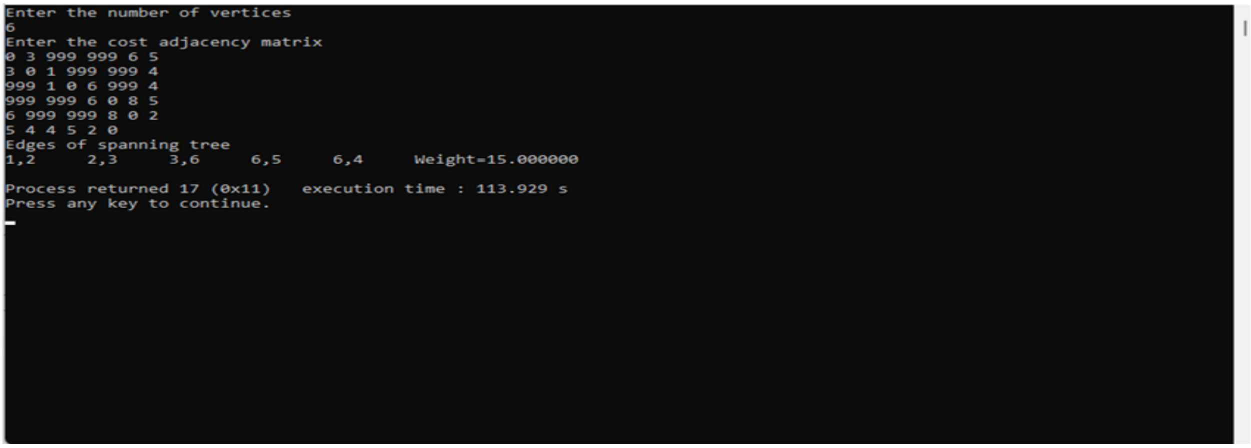
```
for(j=1;j <= n;j++)
{
if(cost[i][j] < min)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
}
}
 u=find(u);
 v=find(v);
 if(uni(u,v))
       printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
       mincost +=min;
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
int find(int i)
{
while(parent[i])
i=parent[i];
```

```
return i;
}
int uni(int i,int j)
{
if(i!=j)
parent[j]=i;
}
```

OUTPUT :



```
Enter the no. of vertices: 5

Enter the cost adjacency matrix:
0 5 999 6 999
5 0 1 3 999
0 1 0 4 6
6 3 4 0 2
0 0 6 2 0
The edges of Minimum Cost Spanning Tree are
1 edge (2,3) =1
2 edge (4,5) =2
3 edge (2,4) =3
4 edge (1,2) =5

Minimum cost = 11
```



```
Enter the number of vertices
6
Enter the cost adjacency matrix
0 3 999 999 6 5
3 0 1 999 999 4
999 1 0 6 999 4
999 999 6 0 8 5
6 999 999 8 0 2
5 4 4 5 2 0
Edges of spanning tree
1,2     2,3     3,6     6,5     6,4     Weight=15.000000

Process returned 17 (0x11)   execution time : 113.929 s
Press any key to continue.
```

WEEK 9

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijikstra's algorithm

CODE :

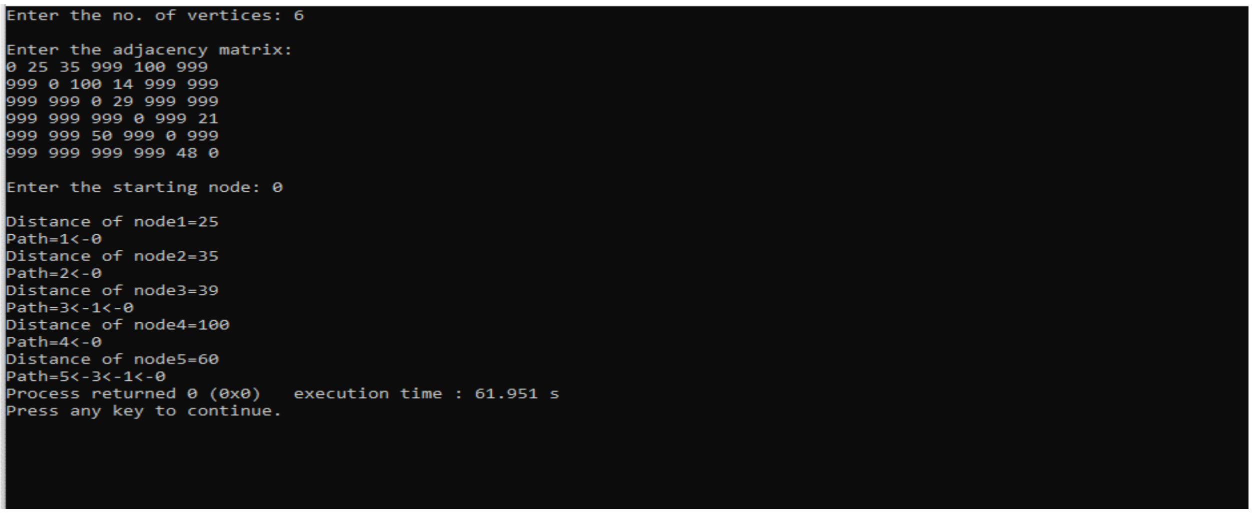
```
#include<stdio.h>
 #include<conio.h>
#define INFINITY 9999
#define MAX 10
int main()
{
int G[MAX][MAX],i,j,n,u;
 printf("Enter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
 printf("\nEnter the starting node:");
scanf("%d",&u);
```

```
 dijkstra(G,n,u);
 }

 void dijkstra(int G[MAX][MAX],int n,int start)
 {
 int cost[MAX][MAX],distance[MAX],pred[MAX];
 int visited[MAX],count,mindistance,nextnode,i,j;
  for(i=0;i<n;i++)
 for(j=0;j<n;j++)
 if(G[i][j]==0)
  cost[i][j]=INFINITY;

 else
 cost[i][j]=G[i][j];
  for(i=0;i<n;i++)
 distance[i]=cost[start][i];
 pred[i]=start;
  visited[i]=0;
 distance[start]=0;
 visited[start]=1;
count=1;
while(count<n-1)
```

```
{
mindistance=INFINITY;

for(i=0;i<n;i++)
 if(distance[i]<mindistance&&!visited[i])
mindistance=distance[i];
nextnode=i;
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
   distance[i]=mindistance+cost[nextnode][i]; pred[i]=nextnode;
count++;
}

for(i=0;i<n;i++)
if(i!=start)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
```

```
 do
{
j=pred[j]; printf("<-%d",j);
}while(j!=start);
}
}
```

OUTPUT :



```
Enter the no. of vertices: 6

Enter the adjacency matrix:
0 25 35 999 100 999
999 0 100 14 999 999
999 999 0 29 999 999
999 999 999 0 999 21
999 999 50 999 0 999
999 999 999 999 48 0

Enter the starting node: 0

Distance of node1=25
Path=1<-0
Distance of node2=35
Path=2<-0
Distance of node3=39
Path=3<-1<-0
Distance of node4=100
Path=4<-0
Distance of node5=60
Path=5<-3<-1<-0
Process returned 0 (0x0)   execution time : 61.951 s
Press any key to continue.
```

WEEK 10

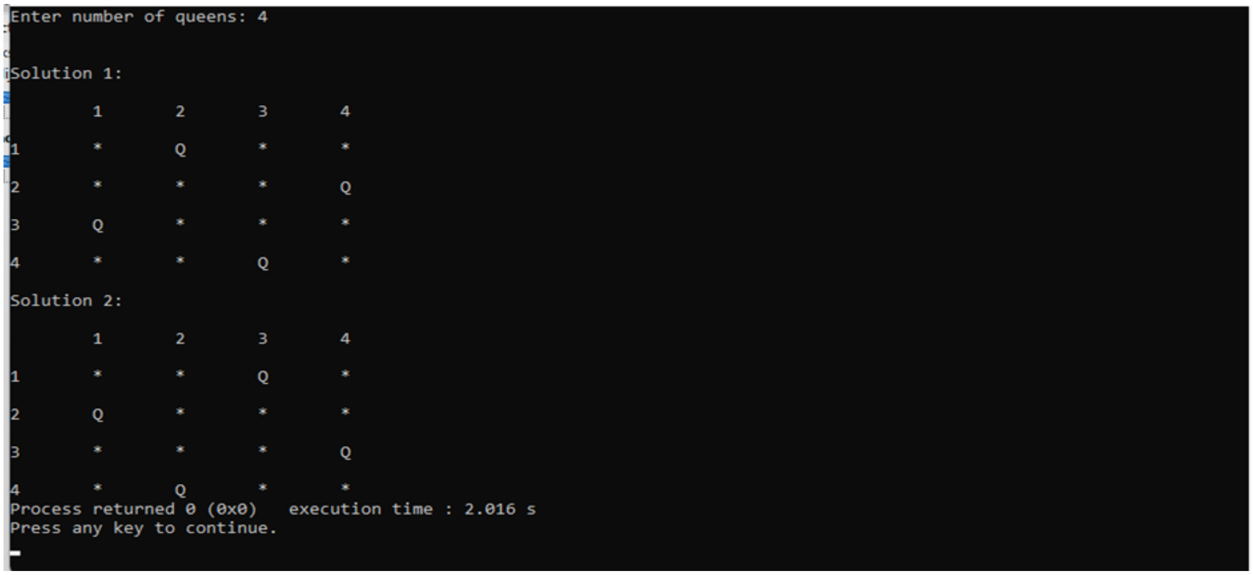
Implement N-Queen problem

CODE :

```
#include<stdio.h>
#include<math.h>
int board[20],count;
 int main()
{
int n,i,j;
 printf("Enter number of queens:");
 scanf("%d",&n);
queen(1,n);
 return 0;
}

void print(int n)
{
int i,j;
printf("\n\nSolution %d:\n\n",++count);
```

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

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

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

{

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

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

{

if(board[i]==j)

printf("\tQ");

else

printf("\t*");

}

}

}

int place(int row,int column)

{

int i;

for(i=1;i<=row-1;++i)

{

if(board[i]==column)

return 0;

else
if(abs(board[i]-column)==abs(i-row))

return 0;

}

return 1;

}


void queen(int row,int n)

{

int column;

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

{

if(place(row,column))

{

board[row]=column;

if(row==n)

print(n);

else

queen(row+1,n);

}

}

}
```

OUTPUT :



```
Enter number of queens: 4


Solution 1:

        1       2       3       4

1       *       Q       *       *

2       *       *       *       Q

3       Q       *       *       *

4       *       *       Q       *

Solution 2:

        1       2       3       4

1       *       *       Q       *

2       Q       *       *       *

3       *       *       *       Q

4       *       Q       *       *
Process returned 0 (0x0)   execution time : 2.016 s
Press any key to continue.
```

WEEK 11

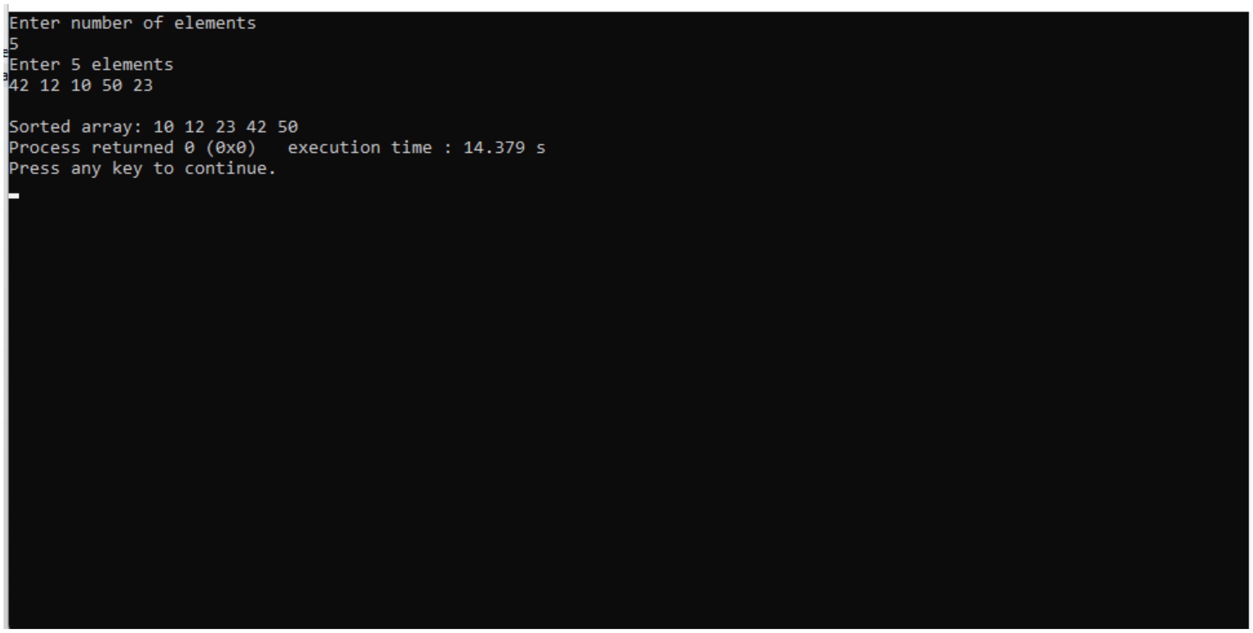
Sort a given set of N integer elements using Heap Sort technique

CODE :

```
#include <stdio.h>
void heapify(int arr[], int n, int i)
{
    int largest = i, left = 2 * i + 1, right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i)
    {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}
```

```
void heapsort(int arr[], int n)

{

    for (int i = n / 2 - 1; i >= 0; i--)

        heapify(arr, n, i);

    for (int i = n - 1; i >= 0; i--) {

        int temp = arr[0];

        arr[0] = arr[i];

        arr[i] = temp;

        heapify(arr, i, 0);

    }

}

int main()

{

    int arr[10], n, i;

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

    scanf("%d", &n);

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

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

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

    heapsort(arr, n);


    printf("\nSorted array: ");
    for (i = 0; i < n; i++)

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


    return 0;

}
```

OUTPUT :



```
Enter number of elements
5
Enter 5 elements
42 12 10 50 23

Sorted array: 10 12 23 42 50
Process returned 0 (0x0)   execution time : 14.379 s
Press any key to continue.
```