VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Vibha Hugar (1BM21CS255)

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
May-2023 to July-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 **Vibha Hugar (1BM21CS255),** who is 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 May-2023 to July-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.

Name of the Lab-In charge: Radhika AD
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.

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.

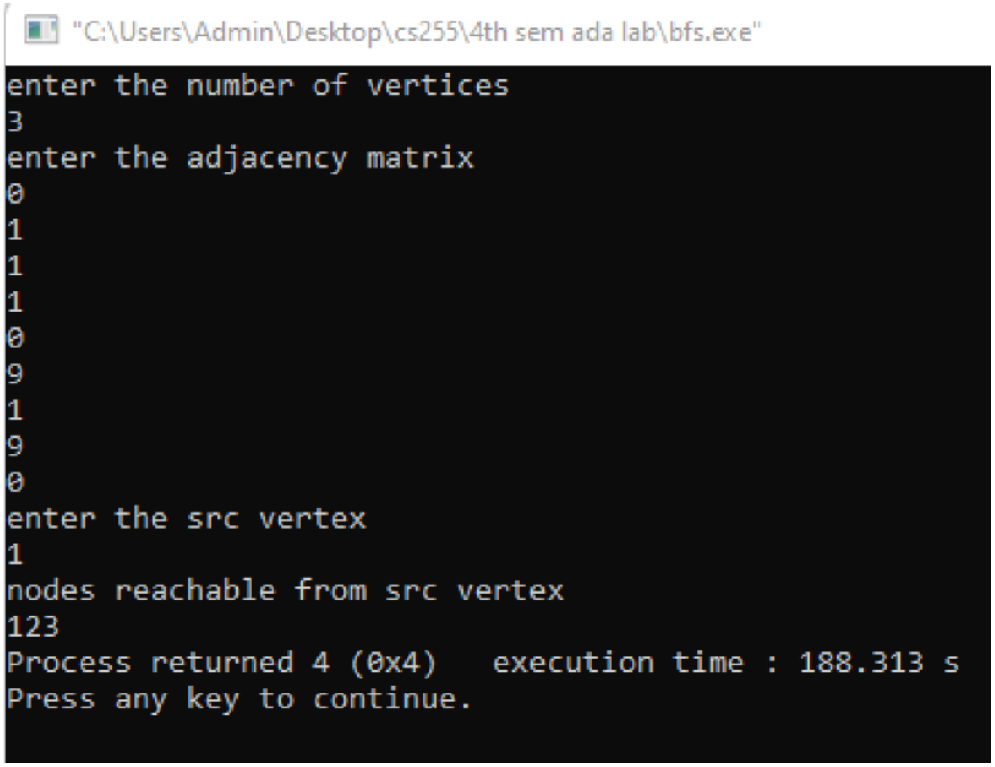
a.

```
#include<stdio.h
>
 void bfs(int);
 int
a[10][10],vis[10],
n;

 void main()
 {
   int i,j,src;

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

scanf("%d",&n);
   printf("enter
the adjacency
matrix\n");

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

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

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

```
      }

      vis[i]=0;
   }

   printf("enter
the src
vertex\n");

scanf("%d",&src);
   printf("nodes
reachable from
src vertex\n");
   bfs(src);

 }

 void bfs(int v)
 {
    int
q[10],f=1,r=1,u,i;
    q[r]=v;
    vis[v]=1;
    while(f<=r)
    {
      u=q[f];

printf("%d",u);

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

if(a[u][i]==1 &&
vis[i]==0)
          {
```

```
            vis[i]=1;
            r=r+1;
            q[r]=i;
          }
      }
      f=f+1;
    }
}
```

OUTPUT:

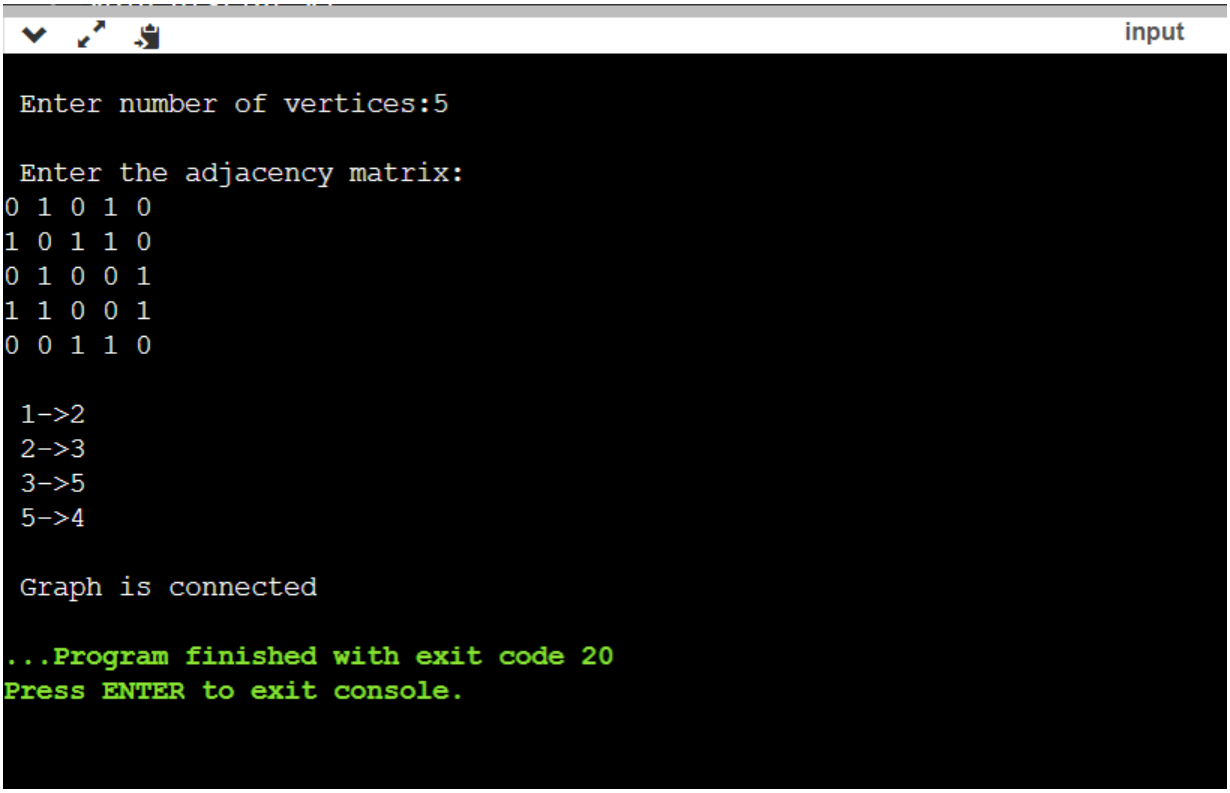


b.

```
#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->%d",v,i);
   dfs(i);
  }
}


void main()
{
 int i,j,count=0;
 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]);
 dfs(1);
 printf("\n");
 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:

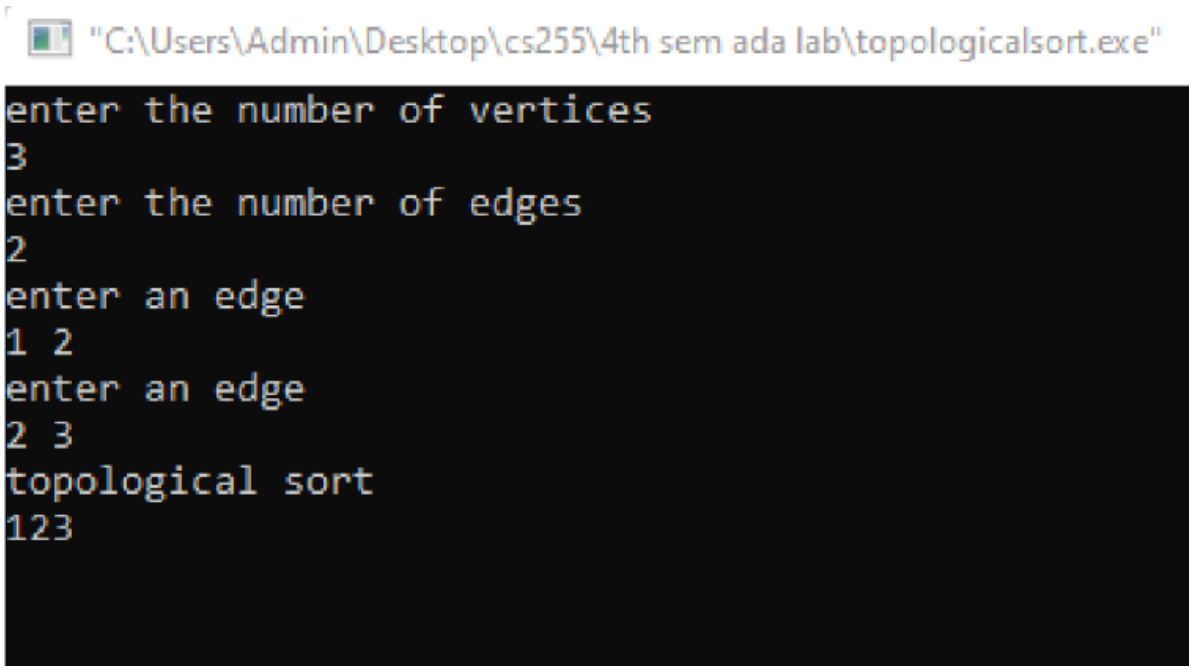


2. Write program to obtain the Topological ordering of vertices in a given digraph.

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

void dfs(int);
int a[10][10],vis[10],exp[10],n,j,m;

void main()
{
  int i,x,y;
  printf("enter the number of vertices\n");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
  {
    for(j=1;j<=n;j++)
    {
     a[i][j]=0;
    }
    vis[i]=0;
  }
  printf("enter the number of edges\n");
  scanf("%d",&m);
  for(i=1;i<=m;i++)
  {
    printf("enter an edge\n");
    scanf("%d %d",&x,&y);
```

```
    a[x][y]=1;
  }
  j=0;
  for(i=1;i<=n;i++)
  {
    if(vis[i]==0)
     dfs(i);
  }
  printf("topological sort\n");
  for(i=n-1;i>=0;i--)
  {
    printf("%d",exp[i]);
  }
  getch();
}

void dfs(int v)
{
  int i;
  vis[v]=1;
  for(i=1;i<=n;i++)
  {
    if(a[v][i]==1 && vis[i]==0)
    dfs(i);
  }
  exp[j++]=v;
}
```

OUTPUT:


"C:\Users\Admin\Desktop\cs255\4th sem ada lab\topologicalsort.exe"
enter the number of vertices
3
enter the number of edges
2
enter an edge
1 2
enter an edge
2 3
topological sort
123

3. Implement Johnson Trotter algorithm to generate permutations.

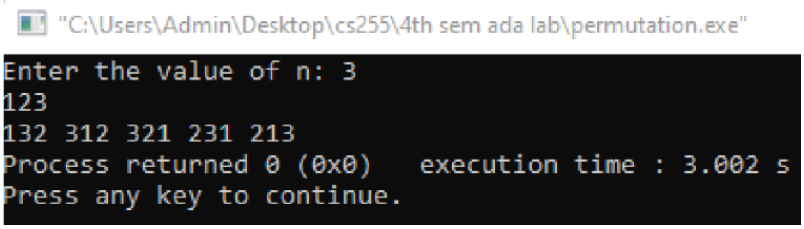
```
#include <stdio.h>

#define RIGHT_TO_LEFT 0
#define LEFT_TO_RIGHT 1

int searchArr(int a[], int n, int mobile) {
int i;
   for (i = 0; i < n; i++)
      if (a[i] == mobile)
         return i + 1;
   return -1;
}

int getMobile(int a[], int dir[], int n) {
int i;
   int mobile_prev = 0, mobile = 0;
   for (i = 0; i < n; i++) {
      if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
         if (a[i] > a[i - 1] && a[i] > mobile_prev) {
            mobile = a[i];
            mobile_prev = mobile;
         }
      }

      if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
         if (a[i] > a[i + 1] && a[i] > mobile_prev) {
```

```
            mobile = a[i];
            mobile_prev = mobile;
         }
      }
   }

   return mobile;
}

void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

void printOnePerm(int a[], int dir[], int n) {
int i;
   int mobile = getMobile(a, dir, n);
   int pos = searchArr(a, n, mobile);

   if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT)
      swap(&a[pos - 1], &a[pos - 2]);
   else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
      swap(&a[pos], &a[pos - 1]);

   for (i = 0; i < n; i++) {
      if (a[i] > mobile) {
```

```
            if (dir[a[i] - 1] == LEFT_TO_RIGHT)
                dir[a[i] - 1] = RIGHT_TO_LEFT;
            else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
                dir[a[i] - 1] = LEFT_TO_RIGHT;
        }
    }

    for (i = 0; i < n; i++)
        printf("%d", a[i]);
    printf(" ");
}

int factorial(int n) {
    int i,res = 1;
    for ( i = 1; i <= n; i++)
        res *= i;
    return res;
}

void printPermutation(int n) {
    int a[n];
    int dir[n];
int i;

    for (i = 0; i < n; i++) {
        a[i] = i + 1;
        printf("%d", a[i]);
```

```
    }
    printf("\n");

    for (i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;

    for (i = 1; i < factorial(n); i++)
        printOnePerm(a, dir, n);
}

int main() {
    int n;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    printf("Permutations:\n");
    printPermutation(n);
    return 0;
}
```

OUTPUT:

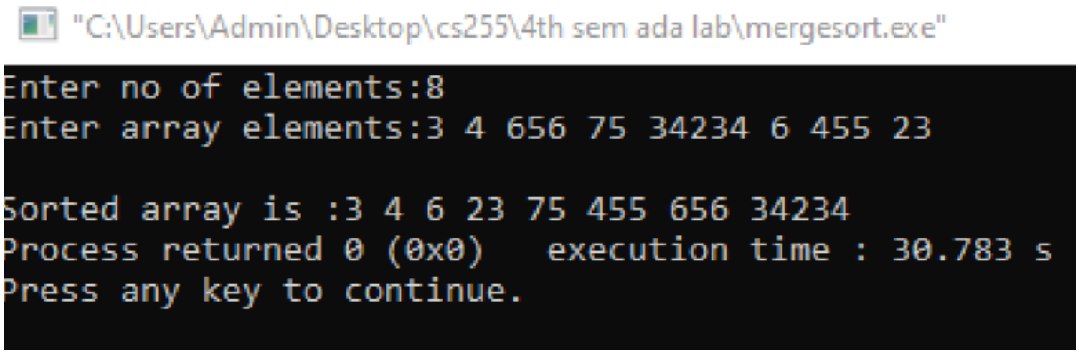


4. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include<stdio.h>

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
int a[30],n,i;
printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter array elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
mergesort(a,0,n-1);
printf("\nSorted array is :");
for(i=0;i<n;i++)
printf("%d ",a[i]);
return 0;
}

void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
mid=(i+j)/2;
mergesort(a,i,mid); //left recursion
mergesort(a,mid+1,j); //right recursion
merge(a,i,mid,mid+1,j); //merging of two sorted sub-arrays
}
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
int temp[50]; //array used for merging
int i,j,k;
i=i1; //beginning of the first list
j=i2; //beginning of the second list
k=0;
while(i<=j1 && j<=j2) //while elements in both lists
```

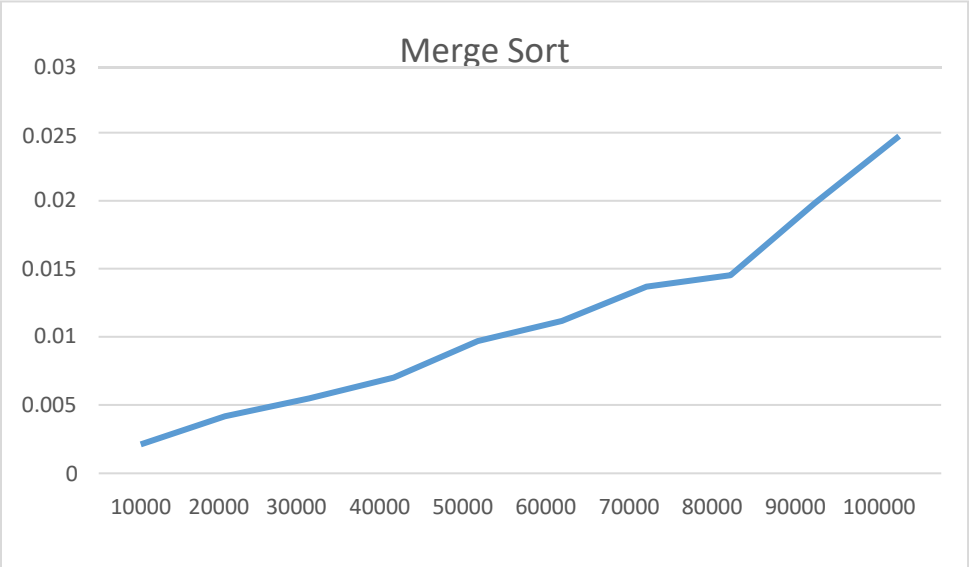
```
{
if(a[i]<a[j])
temp[k++]=a[i++];
else
temp[k++]=a[j++];
}
while(i<=j1) //copy remaining elements of the first list
temp[k++]=a[i++];
while(j<=j2) //copy remaining elements of the second list
temp[k++]=a[j++];
//Transfer elements from temp[] back to a[]
for(i=i1,j=0;i<=j2;i++,j++)
a[i]=temp[j];
}
```

OUTPUT:



GRAPH:

sizeofarray	timetaken
10000	0.002114
20000	0.00418
30000	0.005486
40000	0.007019
50000	0.00969
60000	0.011191
70000	0.013704
80000	0.014539
90000	0.019828
100000	0.024749

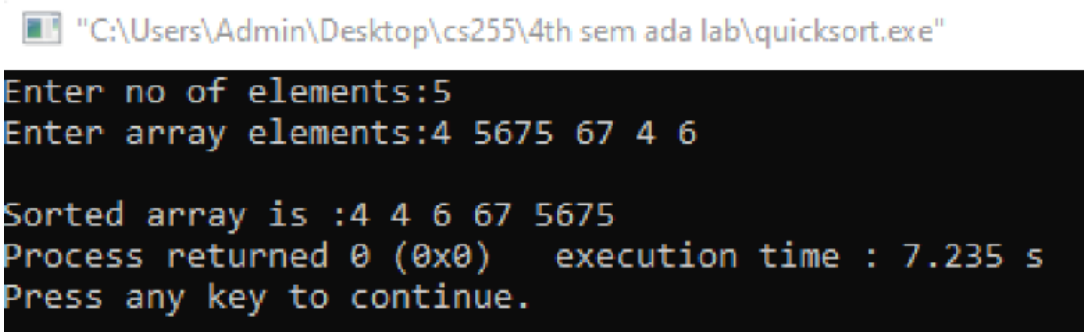


5. Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

```
#include<stdio.h>

void qsort(int a[], int low, int high)
{
   int mid;
   if(low<high)
   {
      mid=partition(a,low,high);
      qsort(a,low,mid-1);
      qsort(a,mid+1, high);
   }
}

int partition(int a[],int low, int high)
{
   int i,j,temp, pivot;
   pivot=a[low];
   i=low+1;
   j=high;

   while(i<=j)
   {
      while(a[i]<=pivot)
         i++;
      while(a[j]>pivot)
         j--;

      if(i<j)
      {
         temp=a[i];
         a[i]=a[j];
         a[j]=temp;
      }
   }

   temp=a[low];
   a[low]=a[j];
   a[j]=temp;
   return j;
}
```

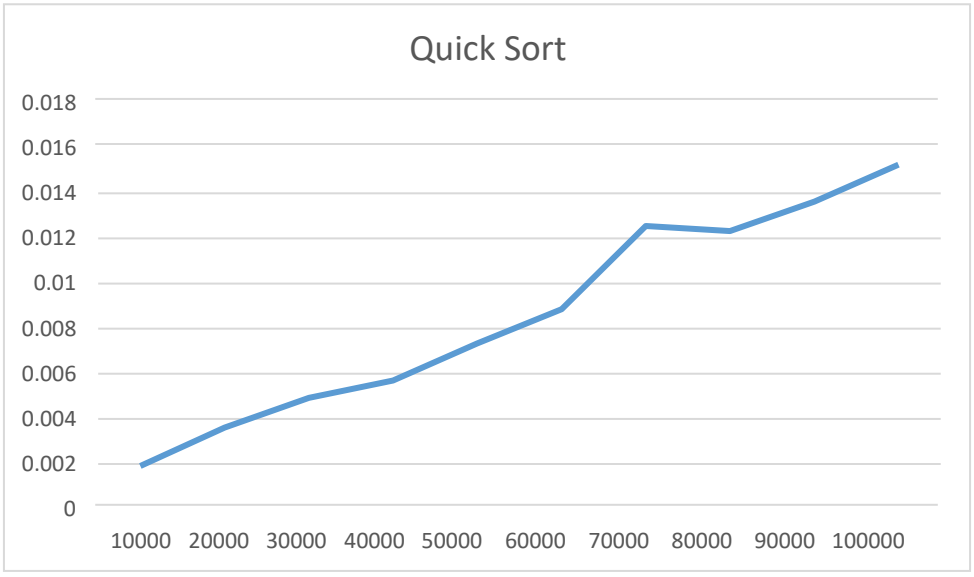
```
int main()
{
   int a[30],n,i;
   printf("Enter no of elements:");
   scanf("%d",&n);
   printf("Enter array elements:");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);

   qsort(a,0,n-1);
   printf("\nSorted array is :");
   for(i=0;i<n;i++)
   printf("%d ",a[i]);

   return 0;
}
```

OUTPUT:



GRAPH:

sizeofarray	timetaken
10000	0.001908
20000	0.003618
30000	0.004931
40000	0.005698
50000	0.00735
60000	0.008865
70000	0.012559
80000	0.012323
90000	0.013631
100000	0.015273

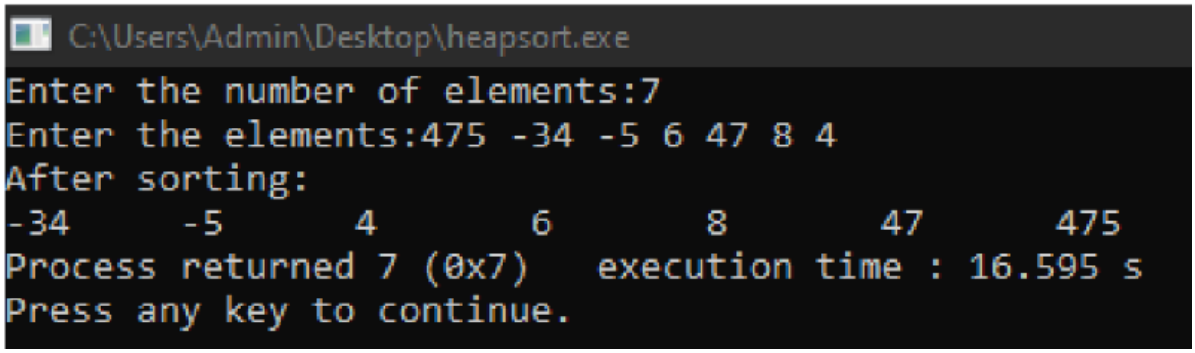

Quick Sort
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000

6. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include<stdio.h>
void heap_adj(int a[],int n)
{
   int i,j,item;
   j=0;
   item=a[j];
   i=2*j+1;
   while(i<n)
   {
      if((i+1)<=n-1)
      {
         if(a[i]<a[i+1])
         i++;
      }
      if(item<a[i])
      {
         a[j]=a[i];
         j=i;
         i=2*j+1;
      }
      else
      break;
   }
   a[j]=item;
}
void heap_const(int a[],int n)
{
   int i,j,k,item;
   for(k=0;k<n;k++)
   {
      item=a[k];
      i=k;
      j=(i-1)/2;
      while(i>0 && item>a[j])
      {
         a[i]=a[j];
         i=j;
         j=(i-1)/2;
      }
      a[i]=item;
   }
}
```

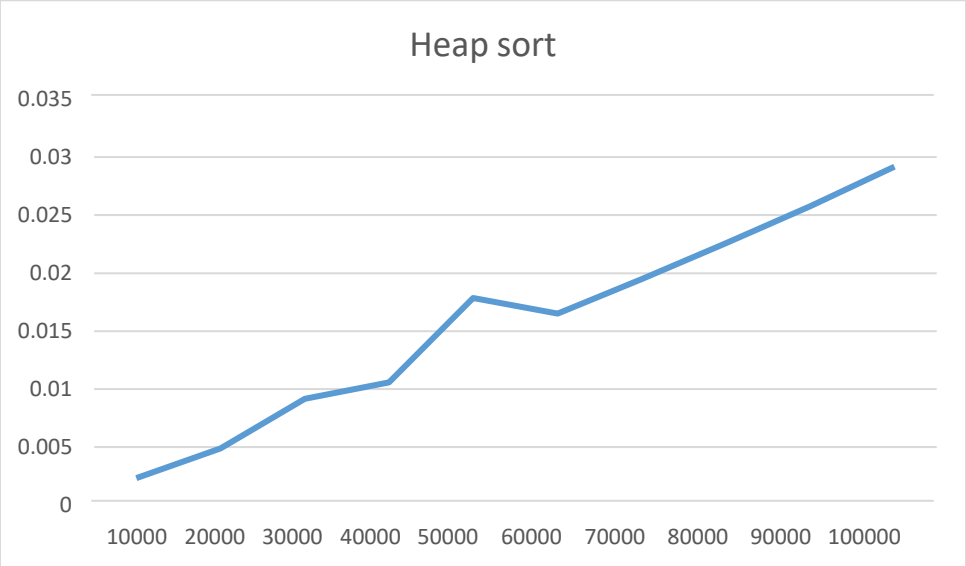
```
void heapsort(int a[],int n)
{
   int i,temp;
   heap_const(a,n);
   for(i=n-1;i>0;i--)
   {
      temp=a[i];
      a[i]=a[0];
      a[0]=temp;
      heap_adj(a,i);
   }
}
void main()
{
   int n,i;
   printf("Enter the number of elements:");
   scanf("%d",&n);
   int a[n];
   printf("Enter the elements:");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   heapsort(a,n);
   printf("After sorting:\n");
   for(i=0;i<n;i++)
   printf("%d\t",a[i]);
}
```

OUTPUT:



GRAPH:

sizeofarray	timetaken
10000	0.002324
20000	0.004903
30000	0.009185
40000	0.010584
50000	0.017871
60000	0.016515
70000	0.019496
80000	0.022587
90000	0.025799
100000	0.029185

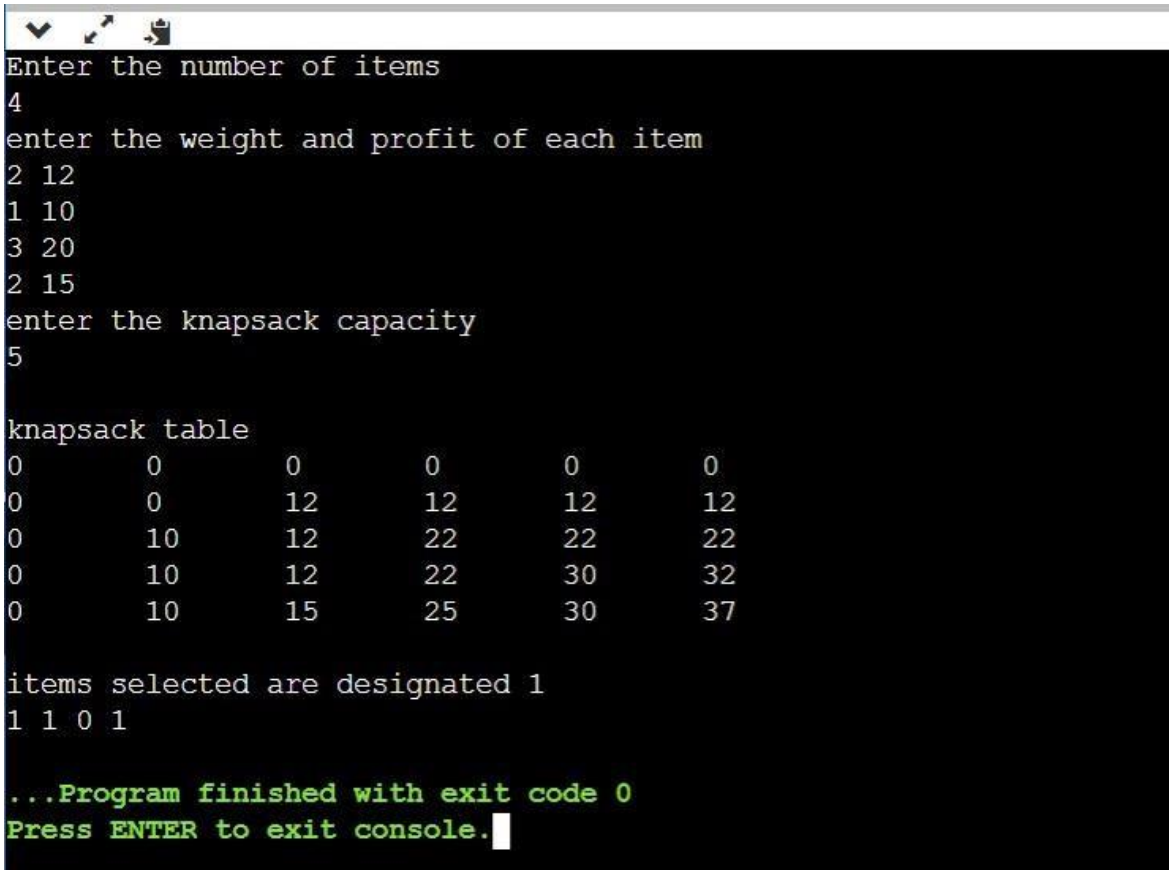

Heap sort
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000

7. Implement 0/1 Knapsack problem using dynamic programming.

```
#include<stdio.h>
void main()
{
   int i,j,w[10],p[10],opt[10][10],x[10],n,m;
   printf("Enter the number of items\n");
   scanf("%d",&n);
   printf("enter the weight and profit of each item\n");
   for(i=1;i<=n;i++)
   {
      scanf("%d %d",&w[i],&p[i]);
   }
   printf("enter the knapsack capacity\n");
   scanf("%d",&m);
   for(i=0;i<=n;i++)
   {
      for(j=0;j<=m;j++)
      {
         if(i==0 || j==0)
         {
            opt[i][j]=0;
         }
         else if(j-w[i]<0)
         {
            opt[i][j]=opt[i-1][j];
         }
         else
         {
            opt[i][j]=opt[i-1][j-w[i]]+p[i]>(opt[i-1][j])?opt[i-1][j-w[i]]+p[i]:(opt[i-1][j]);
         }
      }
   }
   //output
   printf("\nknapsack table\n");
   for(i=0;i<=n;i++)
   {
      for(j=0;j<=m;j++)
      {
```

```
            printf("%d\t",opt[i][j]);
        }
        printf("\n");
    }
    for(i=n;i>=1;i--)
    {
        if(opt[i][m]!=opt[i-1][m])
        {
            x[i]=1;
            m=m-w[i];
        }
        else
        {
            x[i]=0;
        }
    }
    printf("\nitems selected are designated 1\n");
    for(i=1;i<=n;i++)
    {
        printf("%d ",x[i]);
    }


}
```

OUTPUT:

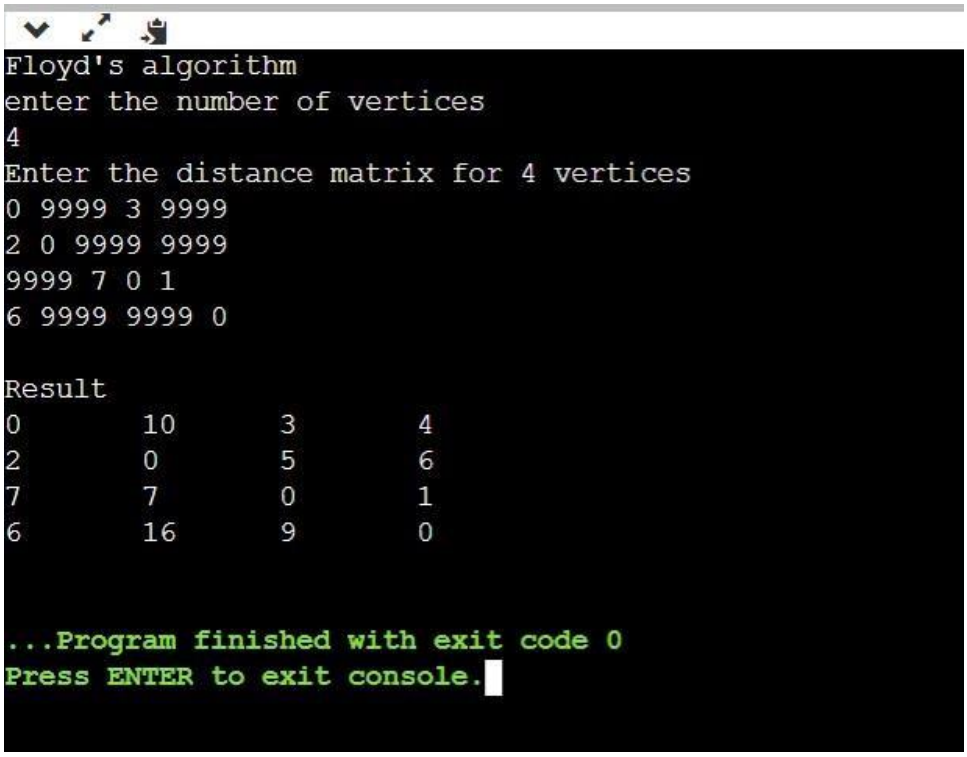

Enter the number of items
4
enter the weight and profit of each item
2 12
1 10
3 20
2 15
enter the knapsack capacity
5
knapsack table
0 0 0 0 0 0
0 0 12 12 12 12
0 10 12 22 22 22
0 10 12 22 30 32
0 10 15 25 30 37
items selected are designated 1
1 1 0 1
...Program finished with exit code 0
Press ENTER to exit console.

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

```
#include<stdio.h>
void main()
{
   int adj[10][10],n,i,j,k;
   int result[10][10];
   printf("Floyd's algorithm\n");
   printf("enter the number of vertices\n");
   scanf("%d",&n);
   printf("Enter the distance matrix for %d vertices\n",n);
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&adj[i][j]);
         result[i][j]=adj[i][j];
      }
   }
   for(k=0;k<n;k++)
   {
      for(j=0;j<n;j++)
      {
         for(i=0;i<n;i++)
         {
            result[i][j]=result[i][j]<(result[i][k]+result[k][j])?result[i][j]:(result[i][k]+result[k][j]);
         }
      }
   }
   printf("\nResult\n");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         printf("%d\t",result[i][j]);
      }
      printf("\n");
   }
```

}

OUTPUT:


Floyd's algorithm
enter the number of vertices
4
Enter the distance matrix for 4 vertices
0 9999 3 9999
2 0 9999 9999
9999 7 0 1
6 9999 9999 0
Result
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
...Program finished with exit code 0
Press ENTER to exit console.

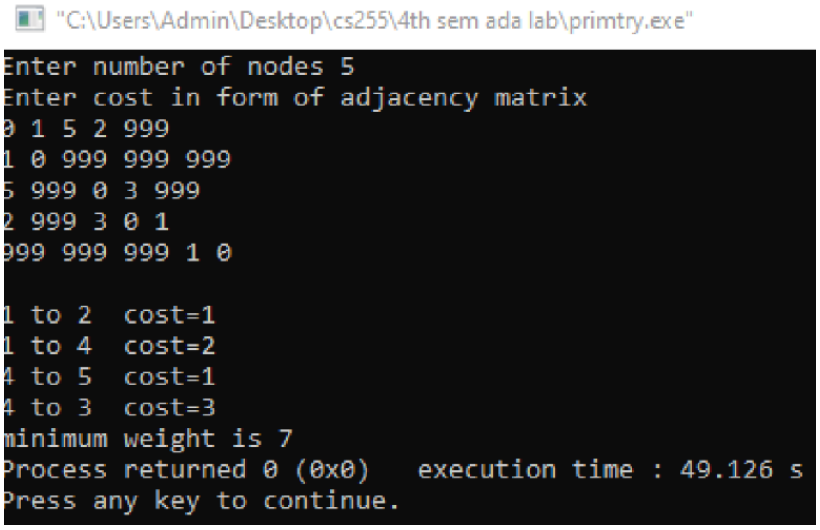
9. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.

OUTPUT:

Prim

```
#include<stdio.h>
int main()
{
   int cost[10][10],visited[10]={0},i,j,n,no_e=1,min,a,b,min_cost=0;
   printf("Enter the number of nodes:\n");
   scanf("%d",&n);
   printf("Enter the cost in form of 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]=1000;
      }
   }


   visited[1]=1;
   while(no_e<n)
   {
      min=1000;

      for(i=1;i<=n;i++)
      {
         for(j=1;j<=n;j++)
         {
            if(cost[i][j]<min)
            {
               if(visited[i]!=0)
               {
```

```
                min=cost[i][j];
                a=i;
                b=j;
              }
            }
          }
        }

        if(visited[b]==0)
        {
          printf("\n%d to %d  cost=%d",a,b,min);
          min_cost=min_cost+min;
          no_e++;
        }
        visited[b]=1;

        cost[a][b]=cost[b][a]=1000;
    }
    printf("\nminimum weight is %d",min_cost);
    return 0;
}
```



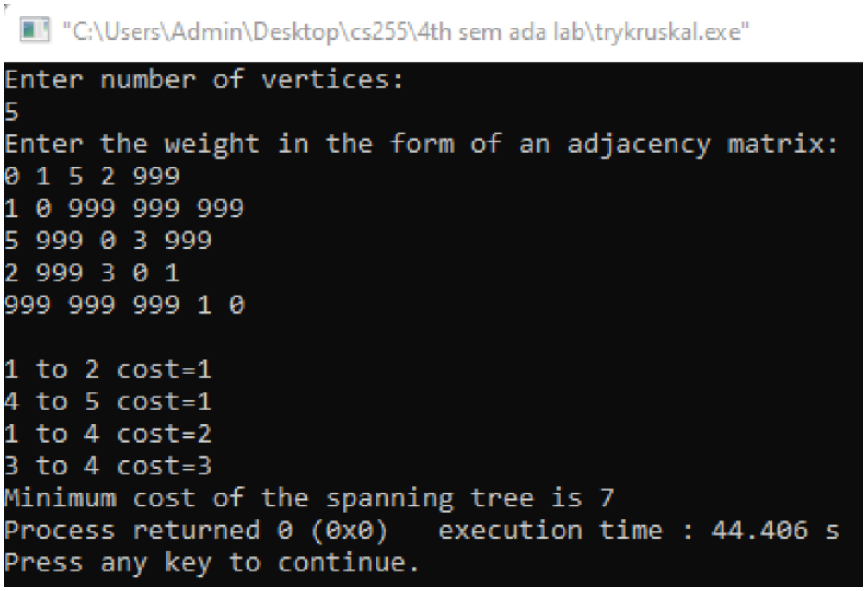
Kruskal

```
#include<stdio.h>
int parent[10]={0};
int find_parent(int);
int is_cyclic(int,int);
int main()
{
   int cost[10][10],min_cost=0,min,i,j,n,no_e=1,a,b,u,v,x;
   printf("Enter number of vertices:\n");
   scanf("%d",&n);
   printf("Enter the weight in the form of an 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;
      }
   }


   while(no_e<n)
   {
      min=999;
```

```
      for(i=1;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_parent(u);
      v=find_parent(v);
      x=is_cyclic(u,v);
      if(x==1)
      {
         printf("\n%d to %d cost=%d",a,b,min);
         no_e++;
         min_cost+=min;
      }
      cost[a][b]=cost[b][a]=999;
   }
   printf("\nMinimum cost of the spanning tree is %d",min_cost);
   return 0;
}


int find_parent(int a)
{
   while(parent[a]!=0)
     a=parent[a];
   return a;
}


int is_cyclic(int a ,int b)
{
   if(a!=b)
   {
      parent[b]=a;
      return 1;
   }
```

```
    return 0;
}
```

OUTPUT:


"C:\Users\Admin\Desktop\cs255\4th sem ada lab\trykruskal.exe"
Enter number of vertices:
5
Enter the weight in the form of an adjacency matrix:
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 1
999 999 999 1 0
1 to 2 cost=1
4 to 5 cost=1
1 to 4 cost=2
3 to 4 cost=3
Minimum cost of the spanning tree is 7
Process returned 0 (0x0) execution time : 44.406 s
Press any key to continue.

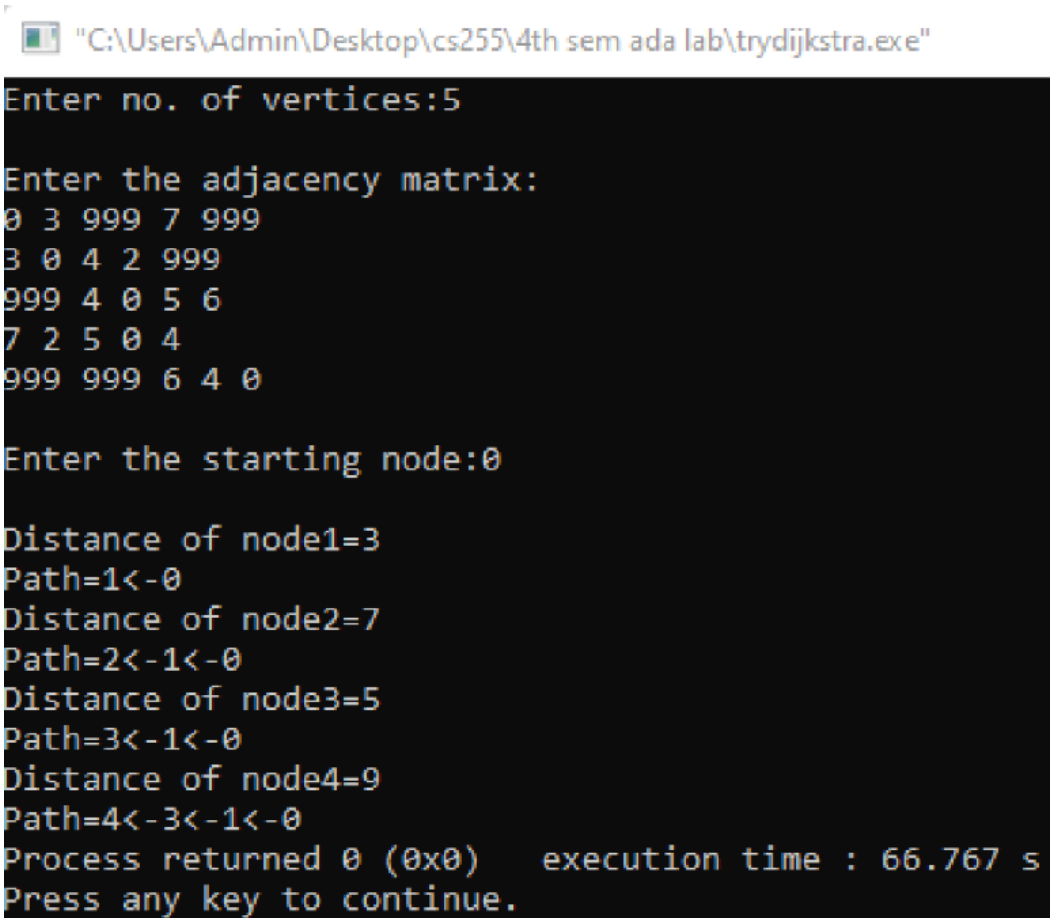
10. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

int main()
{
   int G[MAX][MAX], i, j, n, u;
   printf("Enter 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);
   return 0;
}

void dijkstra(int G[MAX][MAX], int n, int startnode)
{

   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[startnode][i];
   pred[i] = startnode;
   visited[i] = 0;
}
distance[startnode] = 0;
visited[startnode] = 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 != startnode)
   {
      printf("\nDistance of node%d = %d", i, distance[i]);
      printf("\nPath = %d", i);
      j = i;
      do
      {
         j = pred[j];
```

```
            printf("<-%d", j);
        } while (j != startnode);
    }
}
```

OUTPUT:



11. Implement “N-Queens Problem” using Backtracking.

```
#include<stdio.h>
#include<math.h>

int board[20],count;

int main()
{
int n,i,j;
void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
queen(1,n);
return 0;
}



//function for printing the solution
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) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
```

```
 }
}
}


/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row,int column)
{
int i;
for(i=1;i<=row-1;++i)
{
  //checking column and digonal conflicts
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}

return 1; //no conflicts
}




//function to check for proper positioning of queen
void queen(int row,int n)
{
int column;
for(column=1;column<=n;++column)
{
  if(place(row,column))
  {
   board[row]=column; //no conflicts so place queen
   if(row==n) //dead end
    print(n); //printing the board configuration
   else //try queen with next position
```

```
   queen(row+1,n);
  }
}
}
```

OUTPUT: