50 VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

ARYAN MADHAN PILLAI (1BM21CS033)

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 ARYAN MADHAN PILLAI**(1BM21CS033),** 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 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.

Name of the Lab-In charge:	
Dr.Sunayana S	Dr. Jyothi Nayak
Designation	Professor and Head
Assistant Professor	Department of CSE
Department of CSE	BMSCE, Bengaluru
BMSCE, Bengaluru	

Index Sheet

Lab Program No.	Program Details	Page No.
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.	5
2	Write program to obtain the Topological ordering of vertices in a given digraph.	10
3	Implement Johnson Trotter algorithm to generate permutations.	13
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.	17
5	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	20
6	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	23
7	Implement 0/1 Knapsack problem using dynamic programming.	27
8	Implement All Pair Shortest paths problem using Floyd’s algorithm.	32
9	Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm.	35
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.	43

Course Outcome

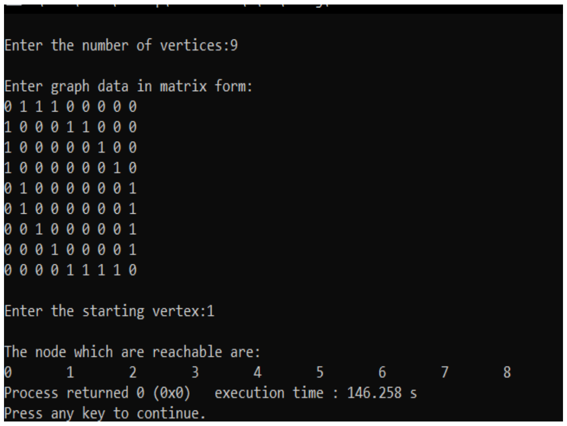
CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms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

1(a).Write program to print all the nodes reachable from a given starting node in a digraph using BFS method.

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

int a[10][20], q[10], visited[10], n, i, j, f = 0, r = -1;

void bfs(int k) {
   for(i = 0; i < n; i++){
      if(a[k][i] && visited[i]==0){
         q[++r] = i;
      }
   }
   if(f <= r){
      visited[q[f]] = 1;
      bfs(q[f++]);
   }
}

int main() {
   int v;
   printf("\nEnter 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=0; i<n; i++) {
      for(j=0;j<n;j++) {
         scanf("%d", &a[i][j]);
      }
   }

   printf("\nEnter the starting vertex:");
   scanf("%d", &v);
   bfs(v);
   printf("\nThe node which are reachable are:\n");

   for(i=0; i < n; i++) {
      if(visited[i])
         printf("%d\t", i);
      else {
         printf("\nBfs is not possible. Not all nodes are reachable!\n");
         break;
      }
   }
   return 0;
}
```

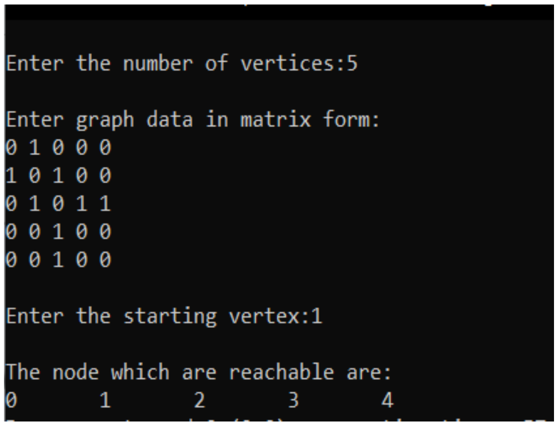
OUTPUT

CASE1:



```
Enter the number of vertices:9

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

Enter the starting vertex:1

The node which are reachable are:
0       1       2       3       4       5       6       7       8
Process returned 0 (0x0)   execution time : 146.258 s
Press any key to continue.
```

CASE 2:



```
Enter the number of vertices:5

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

Enter the starting vertex:1

The node which are reachable are:
0       1       2       3       4
```

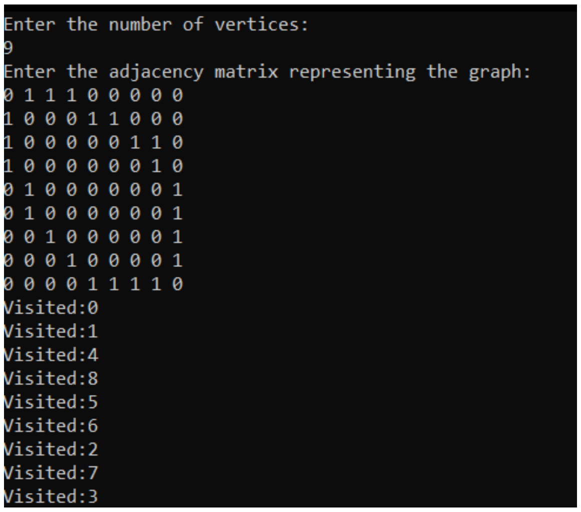
1(b).Write program to do the following check whether a given graph is connected or not using DFS method.

```
#include<stdio.h>

int graph[20][20], vis[10];

void DFS(int i,int n){
   int j;
   printf("Visited:%d\n",i);
   vis[i]=1;
   for(j=0;j<n;j++){
      if(graph[i][j]==1 && vis[j]==0){
         DFS(j,n);
      }
   }
}
void main(){

   int n,i,j,top=-1;
   printf("Enter the number of vertices:\n");
   scanf("%d",&n);
   printf("Enter the adjacency matrix representing the graph:\n");

   int vis[n],st[n];

   for(int i=0;i<n;i++){
      for(int j=0;j<n;j++){
         scanf("%d",&graph[i][j]);
      }
   }
   for(int i=0;i<n;i++){
      vis[i]=0;
   }
```

```
    DFS(0,n);
}
```

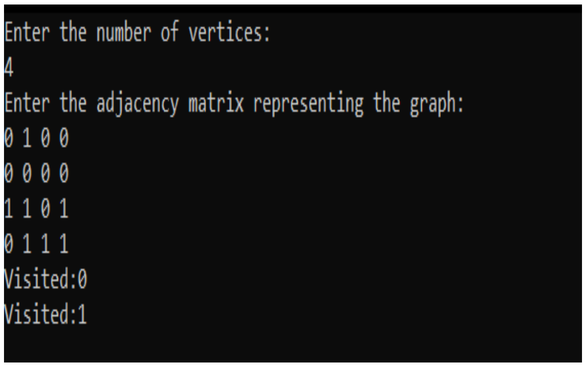
OUTPUT

CASE1:



```
Enter the number of vertices:
9
Enter the adjacency matrix representing the graph:
0 1 1 1 0 0 0 0 0
1 0 0 0 1 1 0 0 0
1 0 0 0 0 0 1 1 0
1 0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 1
0 0 0 1 0 0 0 0 1
0 0 0 0 1 1 1 1 0
Visited:0
Visited:1
Visited:4
Visited:8
Visited:5
Visited:6
Visited:2
Visited:7
Visited:3
```

CASE2:



```
Enter the number of vertices:
4
Enter the adjacency matrix representing the graph:
0 1 0 0
0 0 0 0
1 1 0 1
0 1 1 1
Visited:0
Visited:1
```

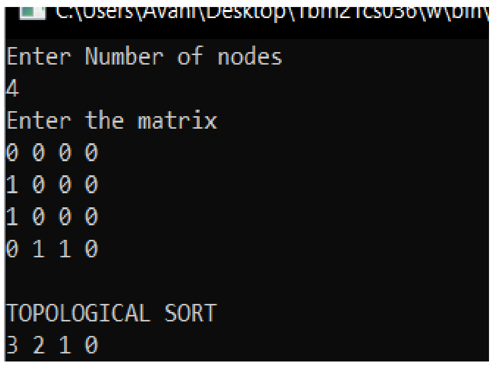
2. Write program to obtain the Topological ordering of vertices in a given digraph

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

int visited[50],graph[10][10],n,stack[10],top=-1;

void topological_sort(int node){
   visited[node]=1;

   for(int j=0;j<n;j++){
      if(graph[node][j]==1 && visited[j]!=1){
         topological_sort(j);
      }
   }

   stack[++top]=node;
}

int main(){
printf("Enter Number of nodes\n");
scanf("%d",&n);
printf("Enter the matrix\n");
for(int i=0;i<n;i++){
   for(int j=0;j<n;j++){
      int key;
      scanf("%d",&key);
      graph[i][j]=key;
   }
}for(int i=0;i<n;i++){
      topological_sort(i);
   }
```

```
    printf("\nTOPOLOGICAL SORT\n");
    while(top!=-1){
        printf("%d ",stack[top--]);
    }

return 0;
}
```

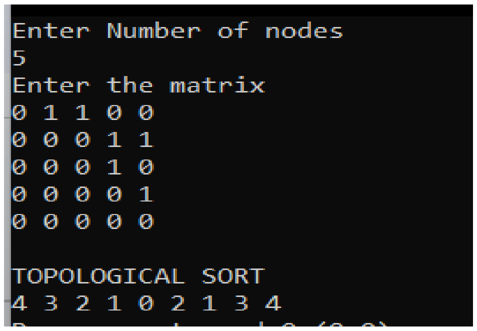
OUTPUT

CASE1:



```
C:\Users\Avani\Desktop\1bm21cs036\w\bin\

Enter Number of nodes
4
Enter the matrix
0 0 0 0
1 0 0 0
1 0 0 0
0 1 1 0

TOPOLOGICAL SORT
3 2 1 0
```

CASE 2:



```
Enter Number of nodes
5
Enter the matrix
0 1 1 0 0
0 0 0 1 1
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0

TOPOLOGICAL SORT
4 3 2 1 0 2 1 3 4
```

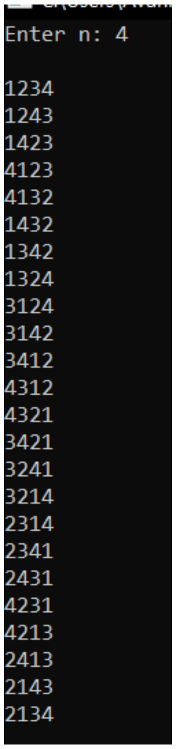
3. Implement Johnson Trotter algorithm to generate permutations.

```
#include  <stdio.h>

#include<conio.h>


int NN, i, count=0;

int p[100], pi[100];

int dir[100];


void PrintPerm()

{

  int i;

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

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

   printf("\n");

}


void Move( int x, int d )

{

  int z;

  z = p[pi[x]+d];

  p[pi[x]] = z;

  p[pi[x]+d] = x;

  pi[z] = pi[x];
  pi[x] = pi[x]+d;
}

void Perm ( int n )
{
  int i;
  if (n > NN)
    PrintPerm();
  else
  {
    Perm( n+1 );
    for (i=1; i<=n-1; ++i)
    {
      Move( n, dir[n] );
      Perm( n+1 );
    }
    dir[n] = -dir[n];
  }
}

void main ()
{
  printf( "Enter n: " );
  scanf( "%d", &NN );
```

```
   printf( "\n" );

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

   {

      dir[i] = -1; p[i] = i;

      pi[i] = i;

   }

   Perm ( 1 );

   printf( "\n" );

}
```

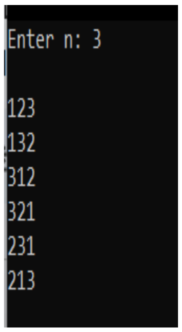
OUTPUT

CASE1:



```
Enter n: 4

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134
```

CASE2:



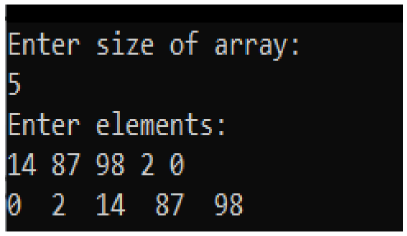
```
Enter n: 3

123
132
312
321
231
213
```

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>
#include <stdlib.h>

int res[10], n;
void SortedMerge(int arr[], int l, int m, int h) {
   int i=l, j=m+1, k=0;

   while (i<=m && j<=h){
      if (arr[i]<arr[j]){
         res[k]=arr[i];
         i++;
      }
      else{
         res[k]=arr[j];
         j++;
      }
      k++;
   }
   while (i<=m){
      res[k++]=arr[i++];
   }
   while (j<=h){
      res[k]=arr[j];
      k++;
      j++;
   }
   for (int i=0; i<(h-l)+1; i++) {
      arr[l+i]=res[i];
   }
}
void Merge(int arr[], int l, int h){
   if (l<h){
        int m = (l+h)/2;
        Merge(arr, l, m);
        Merge(arr, m+1, h);
        SortedMerge(arr, l, m , h);
    }
}

int main(){
    int a[10],i;
    printf ("Enter size of array:\n");
    scanf("%d",&n);
    printf ("Enter elements:\n");
    for (i=0; i<n; i++){
        scanf("%d",&a[i]);
    }
    Merge(a,0,n-1);

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

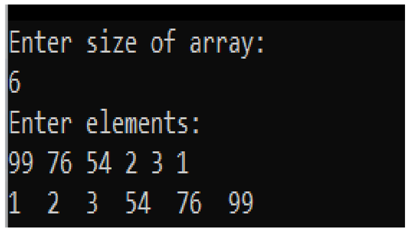
OUTPUT

CASE1:



```
Enter size of array:
5
Enter elements:
14 87 98 2 0
0  2  14  87  98
```

CASE2:



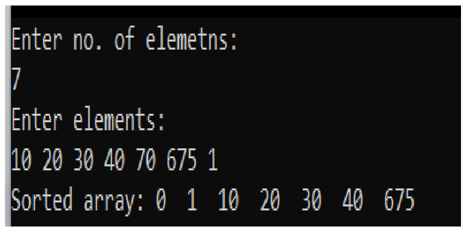
```
Enter size of array:
6
Enter elements:
99 76 54 2 3 1
1  2  3  54  76  99
```

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

```
#include <stdio.h>

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

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

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

    if (i<j) swap(&arr[i], &arr[j]);
  }
  swap (&arr[low], &arr[j]);
  return j;
}

void quickSort(int arr[], int low, int high) {
  if (low < high) {
    int j = partition(arr, low, high);

    quickSort(arr, low, j-1);
    quickSort(arr, j+1, high);
  }
}

int main() {
  int arr[10], n, i;
```

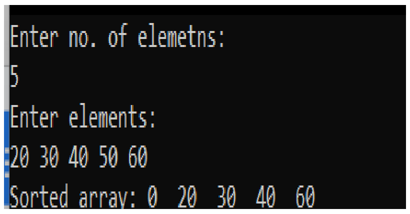
```
    printf("Enter no. of elemetns:\n");
    scanf ("%d", &n);

    printf ("Enter elements:\n");
    for (i=0; i<n; i++){
        scanf ("%d",&arr[i]);
    }
    quickSort(arr, 0, n-1);

    printf("Sorted array: ");
    for (i=0; i<n; i++){
        printf ("%d  ", arr[i]);
    }

    return 0;
}
```

OUTPUT
CASE1:



```
Enter no. of elemetns:
7
Enter elements:
10 20 30 40 70 675 1
Sorted array: 0  1  10  20  30  40  675
```

CASE2:



```
Enter no. of elemetns:
5
Enter elements:
20 30 40 50 60
Sorted array: 0  20  30  40  60
```

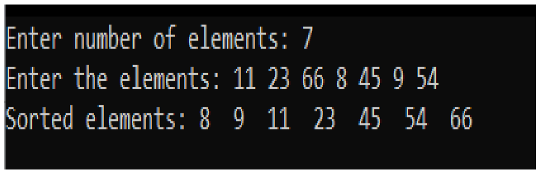
6. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

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

void swap (int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}

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){
        swap (&arr[i], &arr[largest]);
        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--){
      swap (&arr[0], &arr[i]);
      heapify (arr, i, 0);
   }
}

int main (){
   int n;

   printf ("Enter number of elements: ");
   scanf ("%d", &n);
   int arr[n];

   printf ("Enter the elements: ");
   for (int i = 0; i < n; i++){
      scanf ("%d", &arr[i]);
   }
   heapsort (arr, n);
```

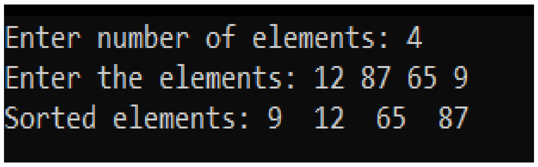
```
    printf ("Sorted elements: ");
    for (int i=0; i<n; i++){
        printf ("%d  ", arr[i]);
    }
    printf ("\n");
    return 0;
}
```

OUTPUT
CASE1:



```
Enter number of elements: 7
Enter the elements: 11 23 66 8 45 9 54
Sorted elements: 8  9  11  23  45  54  66
```

CASE2:



```
Enter number of elements: 4
Enter the elements: 12 87 65 9
Sorted elements: 9  12  65  87
```

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

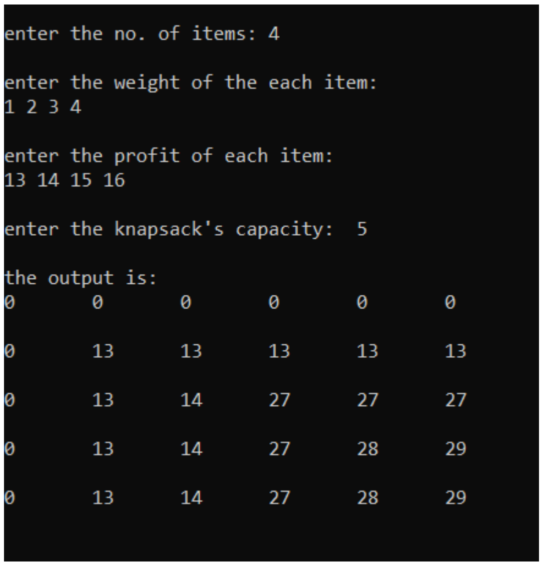
```
#include<stdio.h>

#include<conio.h>

void knapsack();

int max(int,int);

int i,j,n,m,p[10],w[10],v[10][10];

void main()

{

 printf("\nenter the no. of items:\t");

 scanf("%d",&n);

 printf("\nenter the weight of the each item:\n");

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

 {

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

 }

 printf("\nenter the profit of each item:\n");

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

 {

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

 }

 printf("\nenter the knapsack's capacity:\t");

 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("\nthe 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("\nthe optimal solution is %d",v[n][m]);
printf("\nthe 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;
 }
}
```

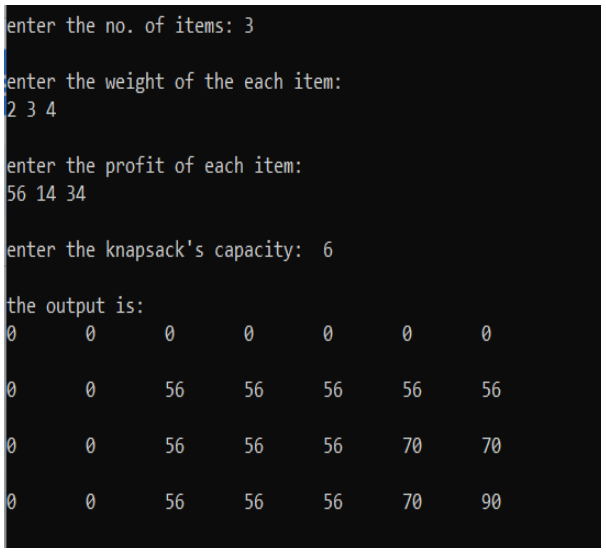
OUTPUT

CASE1:



```
enter the no. of items: 4

enter the weight of the each item:
1 2 3 4

enter the profit of each item:
13 14 15 16

enter the knapsack's capacity:  5

the output is:
0       0       0       0       0       0

0       13      13      13      13      13

0       13      14      27      27      27

0       13      14      27      28      29

0       13      14      27      28      29
```

CASE 2:



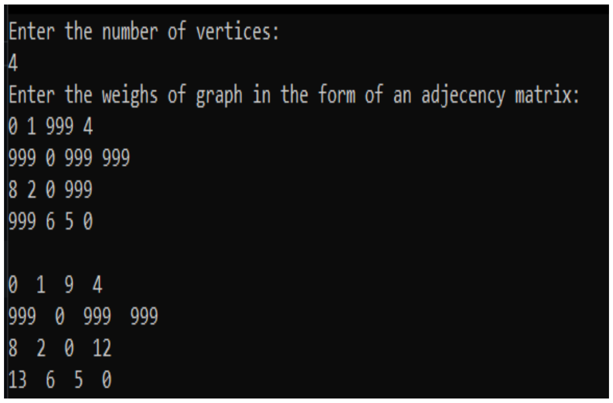
```
enter the no. of items: 3

enter the weight of the each item:
2 3 4

enter the profit of each item:
56 14 34

enter the knapsack's capacity:  6

the output is:
0       0       0       0       0       0       0

0       0       56      56      56      56      56

0       0       56      56      56      70      70

0       0       56      56      56      70      90
```

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

```
#include  <stdio.h>

#include <stdlib.h>

#include <limits.h>


int min (int a, int b) {

   return a < b ? a : b;

}


int main(){

   int n, graph[10][10], i, j, k;

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

   scanf("%d",&n);

   printf("Enter the weighs of graph in the form of an adjecency matrix:\n");

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

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

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


         if (i==j) graph[i][j]=0;


         else if (graph[i][j]==0) graph[i][j] = INT_MAX/3;

      }

   }

   for (k=0; k<n; k++) {
        for (i=0; i<n; i++){
            for (j=0; j<n; j++){
                graph[i][j] = min(graph[i][j], graph[i][k]+graph[k][j]);
            }
        }
    }

    for (i=0; i<n; i++){
        printf ("\n");
        for (j=0; j<n; j++) {
            printf ("%d  ", graph[i][j]);
        }
    }

    return 0;
}
```

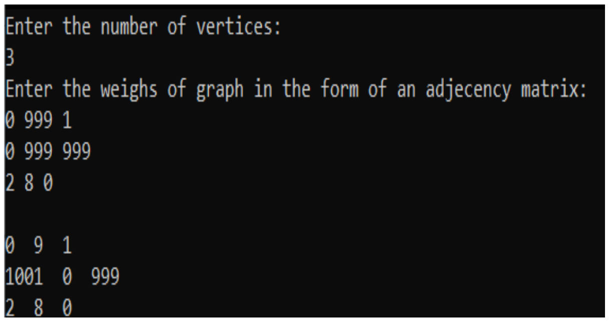
OUTPUT

CASE 1:



```
Enter the number of vertices:
4
Enter the weighs of graph in the form of an adjecency matrix:
0 1 999 4
999 0 999 999
8 2 0 999
999 6 5 0

0  1  9  4
999  0  999  999
8  2  0  12
13  6  5  0
```

CASE 2:



```
Enter the number of vertices:
3
Enter the weighs of graph in the form of an adjecency matrix:
0 999 1
0 999 999
2 8 0

0  9  1
1001  0  999
2  8  0
```

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

PRIMS ALGORITHM

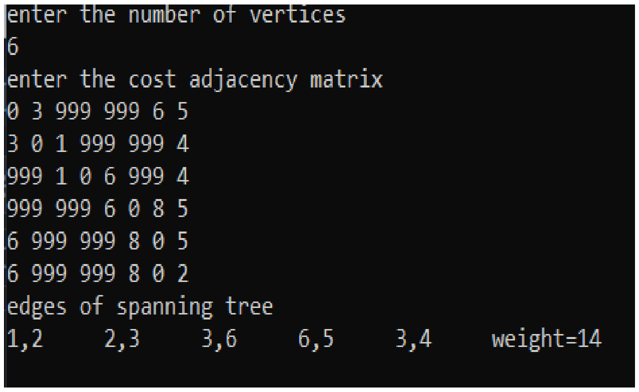
```
#include<stdio.h>


int cost[10][10],vt[10],et[10][10],vis[10],j,n;

int sum=0;

int x=1;

int e=0;

void prims();


int 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("%d",&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=%d\n",sum);
  return 0;
}

void prims()
{
  int s,min,m,k,u,v;
  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;
  }
}
```

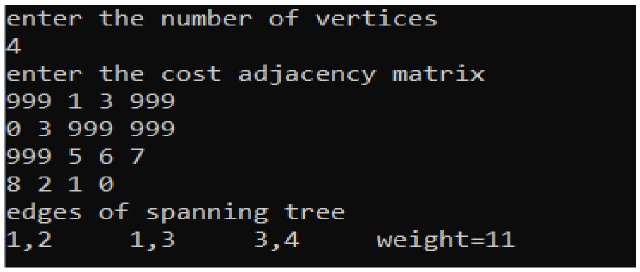
OUTPUT

CASE1:



```
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 5
6 999 999 8 0 2
edges of spanning tree
1,2     2,3     3,6     6,5     3,4     weight=14
```

CASE 2:



```
enter the number of vertices
4
enter the cost adjacency matrix
999 1 3 999
0 3 999 999
999 5 6 7
8 2 1 0
edges of spanning tree
1,2     1,3     3,4     weight=11
```

KRUSHKALS ALGORITHM

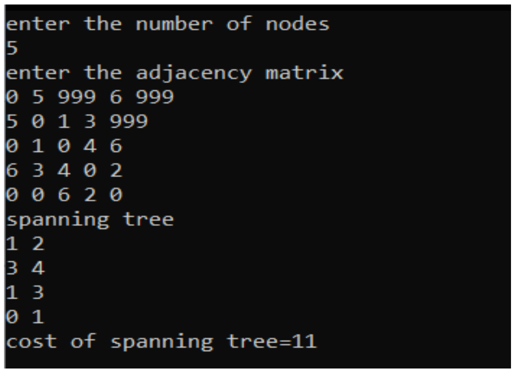
```
#include <stdio.h>

int find(int v, int parent[10])
{
  while (parent[v] != v)
  {
    v = parent[v];
  }
  return v;
}

void union1(int i, int j, int parent[10])
{
  if (i < j)
    parent[j] = i;
  else
    parent[i] = j;
}

void kruskal(int n, int a[10][10])
{
  int count, k, min, sum, i, j, t[10][10], u, v, parent[10];
  count = 0;
  k = 0;
  sum = 0;
  for (i = 0; i < n; i++)
    parent[i] = i;
  while (count != n - 1)
  {
    min = 999;
    for (i = 0; i < n; i++)
    {
      for (j = 0; j < n; j++)
      {

        if (a[i][j] < min && a[i][j] != 0)
```

```
            {
               min = a[i][j];
               u = i;
               v = j;
            }
         }
      }
      i = find(u, parent);
      j = find(v, parent);
      if (i != j)
      {
         union1(i, j, parent);
         t[k][0] = u;
         t[k][1] = v;
         k++;
         count++;
         sum = sum + a[u][v];
      }
      a[u][v] = a[v][u] = 999;
   }
   if (count == n - 1)
   {
      printf("spanning tree\n");
      for (i = 0; i < n - 1; i++)
      {
         printf("%d %d\n", t[i][0], t[i][1]);
      }
      printf("cost of spanning tree=%d\n", sum);
   }
   else
      printf("spanning tree does not exist\n");
}

int main()
{
   int n, i, j, a[10][10];
   printf("enter the number of nodes\n");
   scanf("%d", &n);
```

```
  printf("enter the adjacency matrix\n");
  for (i = 0; i < n; i++)
  {
    for (j = 0; j < n; j++)
      scanf("%d", &a[i][j]);
  }
  kruskal(n, a);
  return 0;
}
```

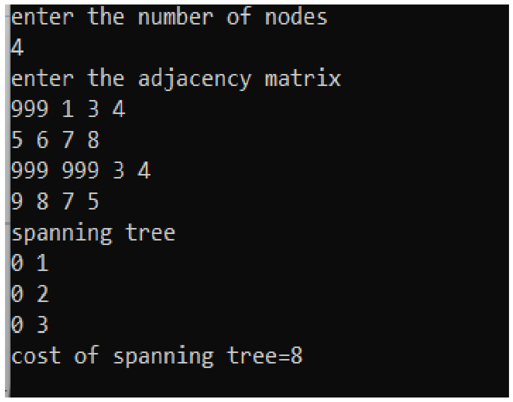
OUTPUT

CASE 1:



```
enter the number of nodes
5
enter the 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
spanning tree
1 2
3 4
1 3
0 1
cost of spanning tree=11
```

CASE 2:



```
enter the number of nodes
4
enter the adjacency matrix
999 1 3 4
5 6 7 8
999 999 3 4
9 8 7 5
spanning tree
0 1
0 2
0 3
cost of spanning tree=8
```

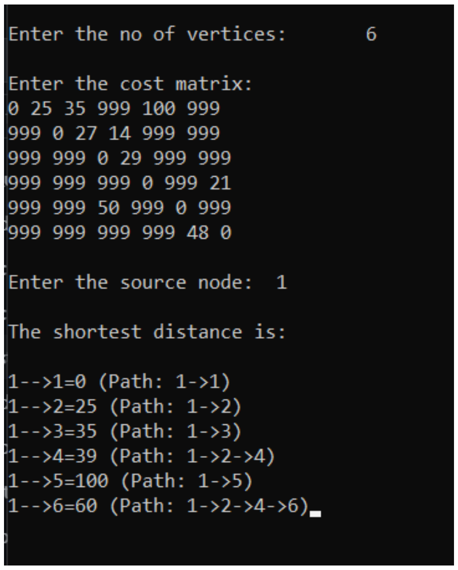
10. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

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

void dijkstras();
int c[10][10], n, src;
void printPath(int parent[], int node);

void main()
{
    int i, j;
    printf("\nEnter the no of vertices:\t");
    scanf("%d", &n);
    printf("\nEnter the cost matrix:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            scanf("%d", &c[i][j]);
        }
    }
    printf("\nEnter the source node:\t");
    scanf("%d", &src);
    dijkstras();
    getch();
}

void dijkstras()
{
    int vis[10], dist[10], parent[10], u, j, count, min;
    for (j = 1; j <= n; j++)
    {
        dist[j] = c[src][j];
        parent[j] = src;
    }
    for (j = 1; j <= n; j++)
    {
        vis[j] = 0;
    }
    dist[src] = 0;
    vis[src] = 1;
    count = 1;
    while (count != n)
    {
        min = 9999;
        for (j = 1; j <= n; j++)
        {
            if (dist[j] < min && vis[j] != 1)
            {
                min = dist[j];
                u = j;
            }
        }
        vis[u] = 1;
        count++;
        for (j = 1; j <= n; j++)
        {
            if (min + c[u][j] < dist[j] && vis[j] != 1)
            {
                dist[j] = min + c[u][j];
                parent[j] = u;
            }
        }
    }
    printf("\nThe shortest distance is:\n");
    for (j = 1; j <= n; j++)
    {
        printf("\n%d-->%d=%d (Path: %d", src, j, dist[j], src);
        printPath(parent, j);
        printf(")");
    }
}

void printPath(int parent[], int node)
```

```
{
   if (parent[node] == src)
   {
      printf("->%d", node);
      return;
   }
   printPath(parent, parent[node]);
   printf("->%d", node);
}
```

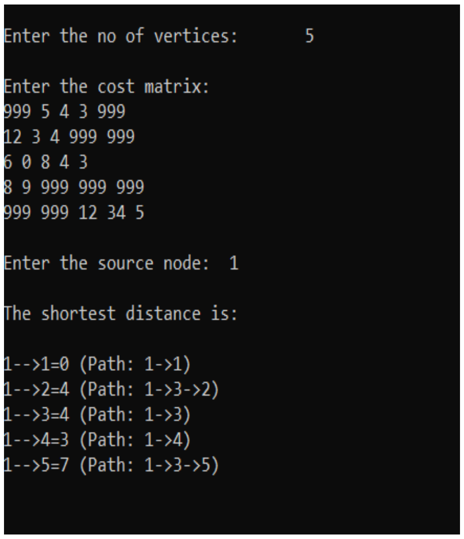
OUTPUT

CASE 1:



```
Enter the no of vertices:       6

Enter the cost matrix:
0 25 35 999 100 999
999 0 27 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 source node:  1

The shortest distance is:

1-->1=0 (Path: 1->1)
1-->2=25 (Path: 1->2)
1-->3=35 (Path: 1->3)
1-->4=39 (Path: 1->2->4)
1-->5=100 (Path: 1->5)
1-->6=60 (Path: 1->2->4->6)
```

CASE 2:



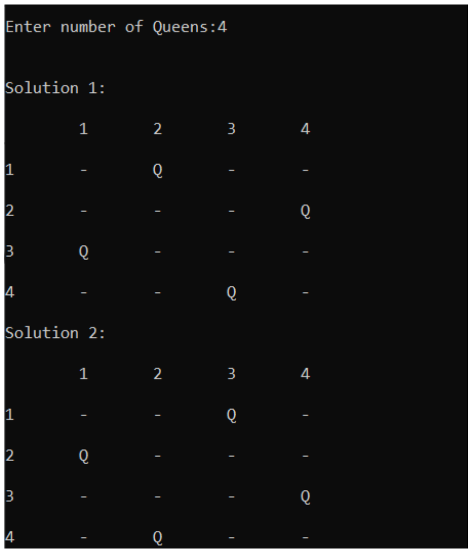
```
Enter the no of vertices:       5

Enter the cost matrix:
999 5 4 3 999
12 3 4 999 999
6 0 8 4 3
8 9 999 999 999
999 999 12 34 5

Enter the source node:  1

The shortest distance is:

1-->1=0 (Path: 1->1)
1-->2=4 (Path: 1->3->2)
1-->3=4 (Path: 1->3)
1-->4=3 (Path: 1->4)
1-->5=7 (Path: 1->3->5)
```

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;
}


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);
  }
}
}
```

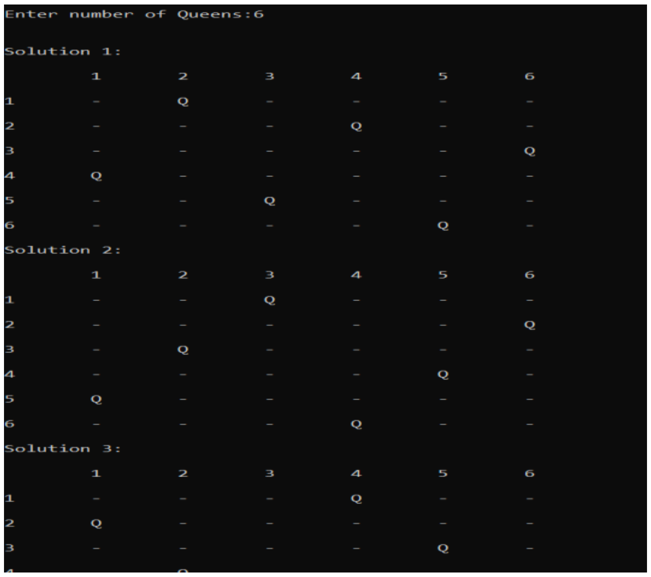
OUTPUT

CASE 1:

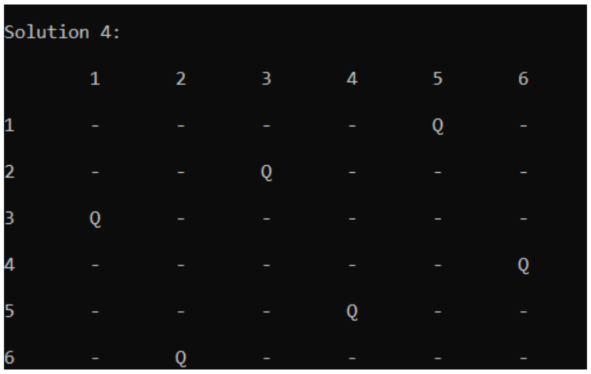


```
Enter number of Queens:4

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       -       -
```

CASE 2:



```
Enter number of Queens:6

Solution 1:
        1       2       3       4       5       6
1       -       Q       -       -       -       -
2       -       -       -       Q       -       -
3       -       -       -       -       -       Q
4       Q       -       -       -       -       -
5       -       -       Q       -       -       -
6       -       -       -       -       Q       -
Solution 2:
        1       2       3       4       5       6
1       -       -       Q       -       -       -
2       -       -       -       -       -       Q
3       -       Q       -       -       -       -
4       -       -       -       -       Q       -
5       Q       -       -       -       -       -
6       -       -       -       Q       -       -
Solution 3:
        1       2       3       4       5       6
1       -       -       -       Q       -       -
2       Q       -       -       -       -       -
3       -       -       -       -       Q       -
```



```
Solution 4:
        1       2       3       4       5       6
1       -       -       -       -       Q       -
2       -       -       Q       -       -       -
3       Q       -       -       -       -       -
4       -       -       -       -       -       Q
5       -       -       -       Q       -       -
6       -       Q       -       -       -       -
```