VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

CHINMAYI (1BM21CS045)

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 **CHINMAYI (1BM21CS045),** 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.

Dr. B S Rajeshwari
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.

Q1)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.

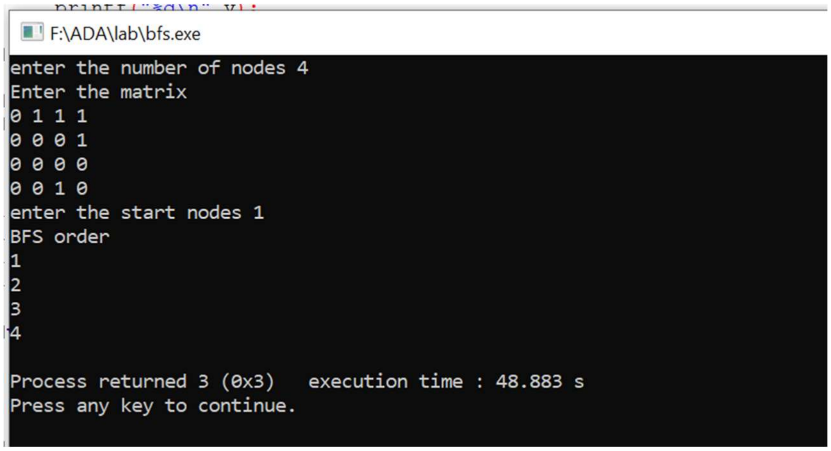
Program:

a. BFS method

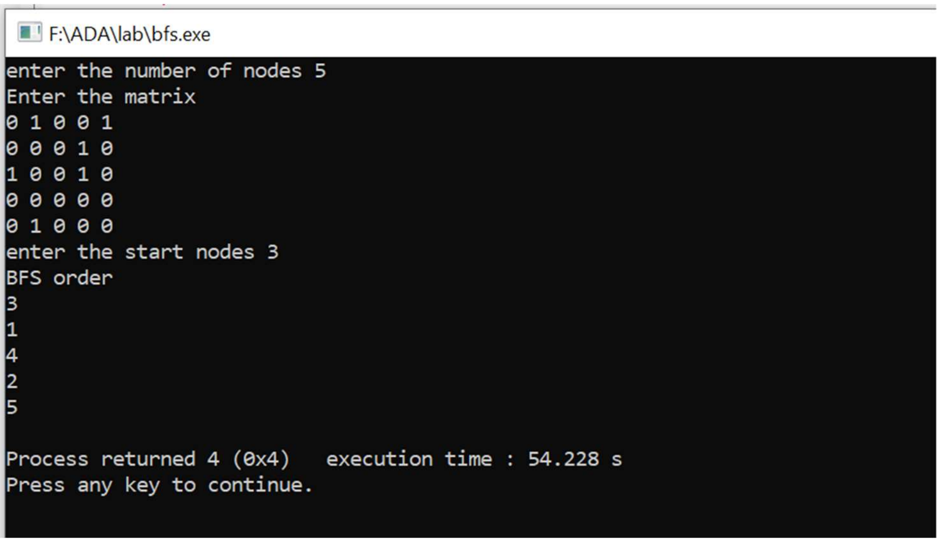
```
#include<stdio.h>

int front = 0, rear =-1;

void bfs(int v, int n, int g[n][n], int visited [n], int queue[n]){

   int i;

   visited[v]=1;

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

   queue[++rear]=v;

   while(front<=rear){

   int temp = queue[front++];

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

   if(g[temp][i]&& !visited[i]){

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

   visited[i]=1;

   queue[++rear]=i;

   }

   }

   }

}

void main(){

   int i,j,n,snode;

   printf("enter the number of nodes ");

   scanf("%d",&n);
```

```
    int g[n][n], visited[n],queue[n];
    for(i=1;i<=n;i++){
            visited[i]=0;
            queue[i]=0;
        for(j=1;j<=n;j++)
            g[i][j]=0;
    }
    printf("Enter the matrix\n");
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++)
          scanf("%d",&g[i][j]);
    }
    printf("enter the start nodes ");
    scanf("%d",&snode);
    printf("BFS order\n");
    bfs(snode,n,g, visited, queue);
}
```

OUTPUT



```
enter the number of nodes 4
Enter the matrix
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
enter the start nodes 1
BFS order
1
2
3
4

Process returned 3 (0x3)   execution time : 48.883 s
Press any key to continue.
```



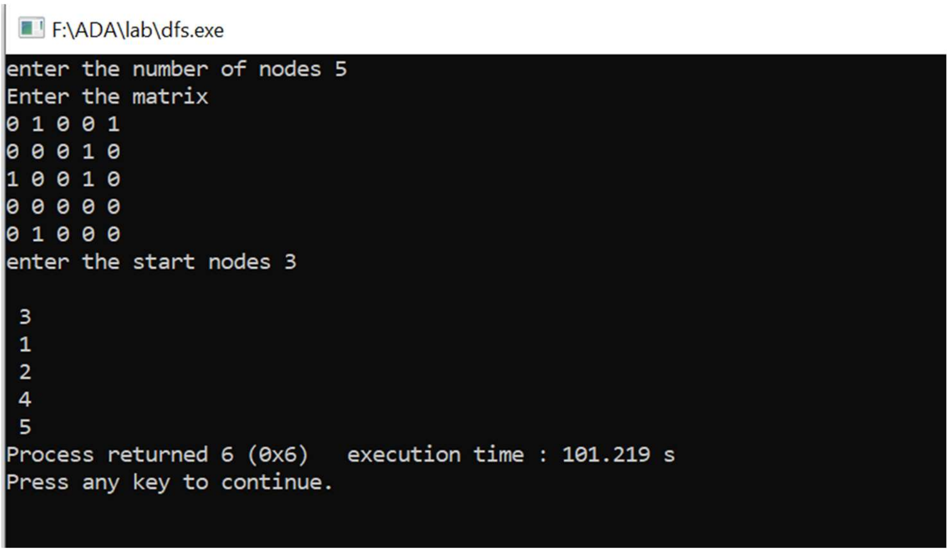
```
enter the number of nodes 5
Enter the matrix
0 1 0 0 1
0 0 0 1 0
1 0 0 1 0
0 0 0 0 0
0 1 0 0 0
enter the start nodes 3
BFS order
3
1
4
2
5

Process returned 4 (0x4)   execution time : 54.228 s
Press any key to continue.
```

b. DFS method

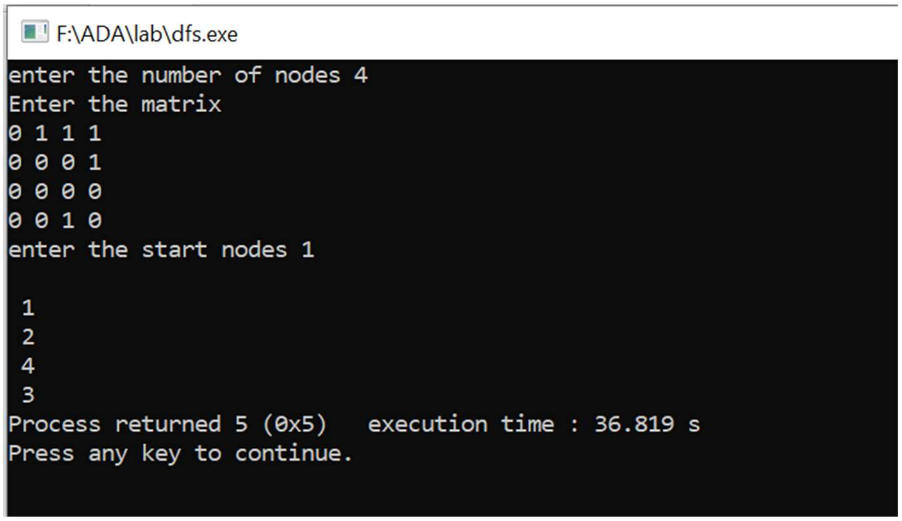
```
#include<stdio.h>

void dfs(int v, int n, int g[n][n], int visited [n]){
    int i;
    printf("\n %d",v);
    visited[v]=1;
    for(i=1;i<=n;i++){
        if(g[v][i]&& !visited[i]){
            dfs(i,n,g, visited);
        }
    }
}

void main(){
    int i,j,n,snode;
    printf("enter the number of nodes ");
    scanf("%d",&n);
    int g[n][n], visited[n];
    for(i=1;i<=n;i++){
            visited[i]=0;
        for(j=1;j<=n;j++)
            g[i][j]=0;
    }
    printf("Enter the matrix\n");
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++)
```

```
            scanf("%d",&g[i][j]);
        }
        printf("enter the start nodes ");
        scanf("%d",&snode);
        dfs(snode,n,g, visited);
}
```

OUTPUT:



```
enter the number of nodes 5
Enter the matrix
0 1 0 0 1
0 0 0 1 0
1 0 0 1 0
0 0 0 0 0
0 1 0 0 0
enter the start nodes 3

 3
 1
 2
 4
 5
Process returned 6 (0x6)   execution time : 101.219 s
Press any key to continue.
```



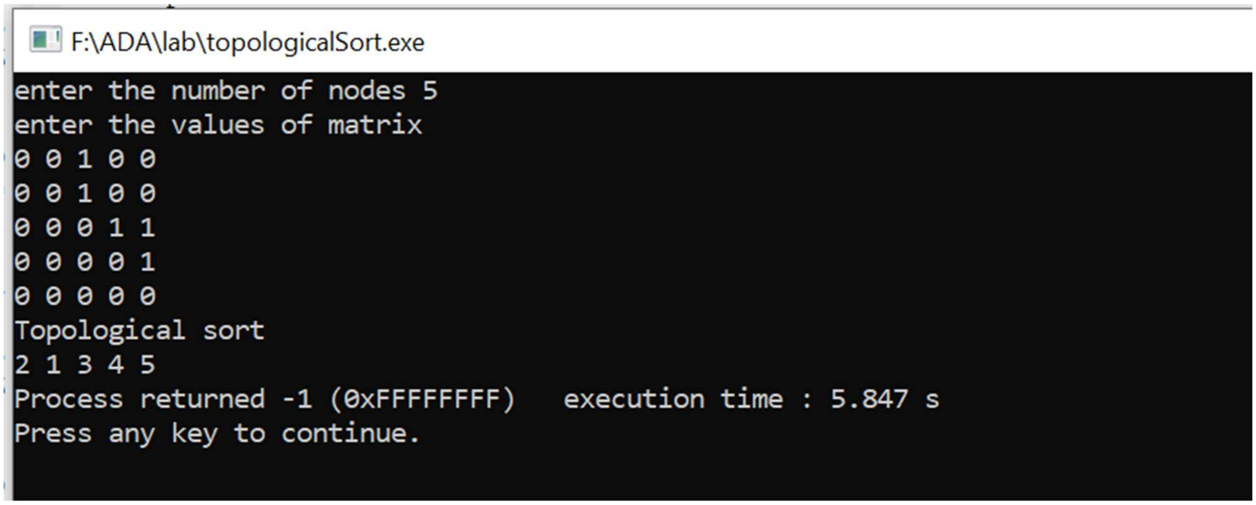
```
enter the number of nodes 4
Enter the matrix
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
enter the start nodes 1

 1
 2
 4
 3
Process returned 5 (0x5)   execution time : 36.819 s
Press any key to continue.
```

Q2) Write program to obtain the Topological ordering of vertices in a given digraph.

Program:

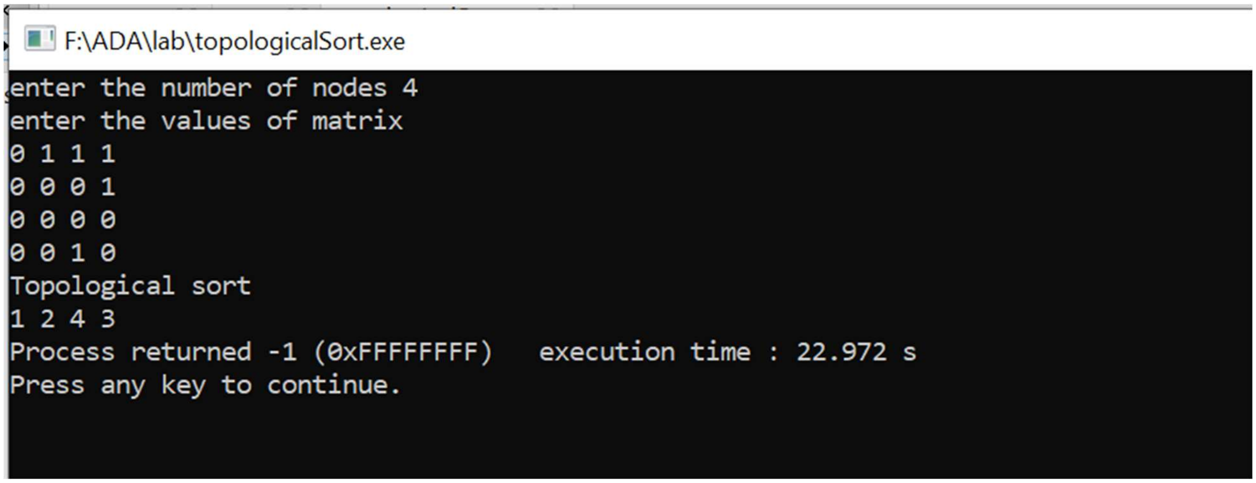
```
#include<stdio.h>
#define MAX 7
int nodes[MAX];
int top=-1;
void main(){
    int i,j,n;
    printf("enter the number of nodes ");
    scanf("%d",&n);
    int matrix[n][n];
    printf("enter the values of matrix \n");
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++){
            scanf("%d",&matrix[i][j]);
        }
    }
    topologicalSort(n,matrix);
}

void topologicalSort(int n, int matrix[n][n]){
    int i,j;
    int visited[MAX]={0};
    for(i=1;i<=n;i++){
        if(!visited[i])
            dfs(i,n,matrix,visited);
    }
    printf("Topological sort\n");
    while (top >= 0)
    {
        printf("%d ", nodes[top--]);
    }
}

void dfs(int v,int n,int matrix[n][n],int visited[n]){
    int i;
    visited[v]=1;
    for(i=1;i<=n;i++){
        if(matrix[v][i]==1 && !visited[i]){
            dfs(i,n,matrix,visited);
        }
    }
    nodes[++top]=v;
}
```

OUTPUT:



F:\ADA\lab\topologicalSort.exe

```
enter the number of nodes 5
enter the values of matrix
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0
Topological sort
2 1 3 4 5
Process returned -1 (0xFFFFFFFF)   execution time : 5.847 s
Press any key to continue.
```



F:\ADA\lab\topologicalSort.exe

```
enter the number of nodes 4
enter the values of matrix
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Topological sort
1 2 4 3
Process returned -1 (0xFFFFFFFF)   execution time : 22.972 s
Press any key to continue.
```

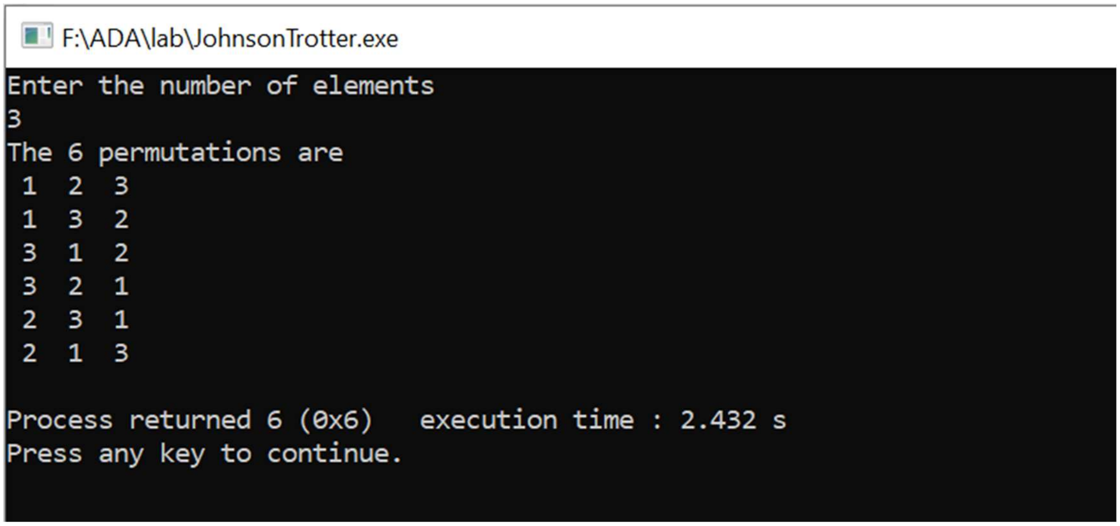
3.Implement Johnson Trotter algorithm to generate permutations.

Program:

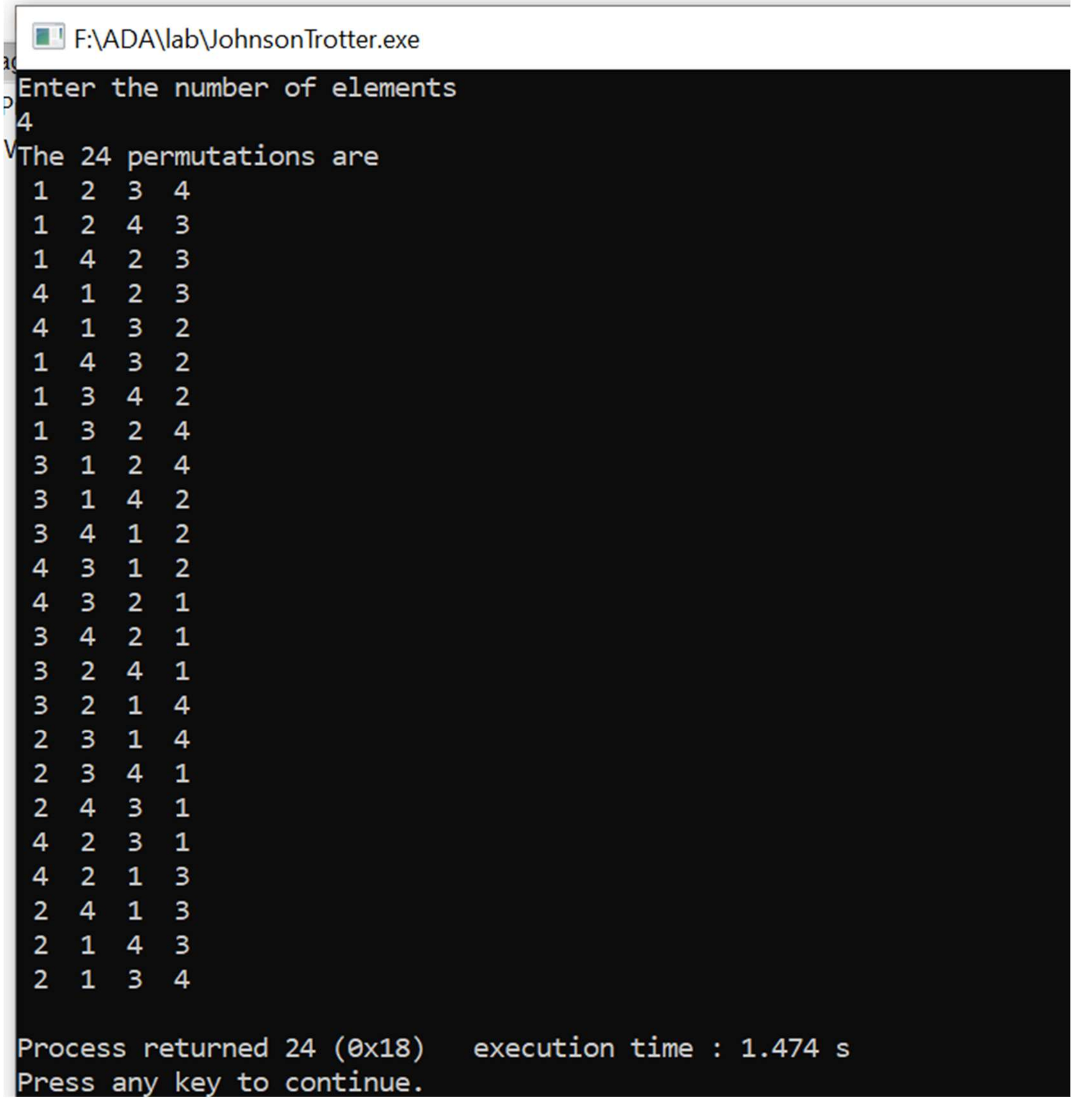
```
#include<stdio.h>
#define LtoR 0
#define RtoL 1

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

void main(){
    int n;
    printf("Enter the number of elements\n");
    scanf("%d",&n);
    printf("The %d permutations are \n",fact(n));
    permutation(n);
}
int fact(int n) {
    int factorial = 1;
    for (int i = 1; i <= n; i++) {
        factorial = factorial * i;
    }
    return factorial;
}
```

```
int searchArray(int arr[], int n, int mob) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == mob) {
            return i + 1;
        }
    }
    return -1;
}
int getMobileComp(int a[], int dir[], int n) {
    int prevMob = 0, mob = 0;
    for (int i = 0; i < n; i++) {
        if (dir[a[i] - 1] == RtoL && i != 0) {
            if (a[i] > a[i - 1] && a[i] > prevMob) {
                mob = a[i];
                prevMob = mob;
            }
        }

        if (dir[a[i] - 1] == LtoR && i != n - 1) {
            if (a[i] > a[i + 1] && a[i] > prevMob) {
                mob = a[i];
                prevMob = mob;
            }
        }
    }

    if (mob == 0 && prevMob == 0) {
        return 0;
    } else {
        return mob;
    }
}

void printOnePerm(int arr[], int dir[], int n, int pnum) {
    int mob = getMobileComp(arr, dir, n);
    int pos = searchArray(arr, n, mob);

    if (dir[arr[pos - 1] - 1] == RtoL) {
        swap(&arr[pos - 1], &arr[pos - 2]);
    } else if (dir[arr[pos - 1] - 1] == LtoR) {
        swap(&arr[pos], &arr[pos - 1]);
    }

    for (int i = 0; i < n; i++) {
        if (arr[i] > mob) {
            if (dir[arr[i] - 1] == LtoR) {
                dir[arr[i] - 1] = RtoL;
            } else if (dir[arr[i] - 1] == RtoL) {
                dir[arr[i] - 1] = LtoR;
            }
        }
    }
    for (int i = 0; i < n; i++) {
        printf(" %d ",arr[i]);
    }
    printf("\n");
}



void permutation(int n){
    int arr[n];
    int dir[n];
    for (int i = 0; i < n; i++) {
        arr[i] = i + 1;
        printf(" %d ", arr[i]);
    }
    printf("\n");

    for (int i = 0; i < n; i++) {
        dir[i] = RtoL;
    }

    for (int i = 1; i < fact(n); i++) {
        printOnePerm(arr, dir, n,i);
    }
}
```

OUTPUT:



```
Enter the number of elements
3
The 6 permutations are
 1  2  3
 1  3  2
 3  1  2
 3  2  1
 2  3  1
 2  1  3

Process returned 6 (0x6)   execution time : 2.432 s
Press any key to continue.
```



```
Enter the number of elements
4
The 24 permutations are
 1  2  3  4
 1  2  4  3
 1  4  2  3
 4  1  2  3
 4  1  3  2
 1  4  3  2
 1  3  4  2
 1  3  2  4
 3  1  2  4
 3  1  4  2
 3  4  1  2
 4  3  1  2
 4  3  2  1
 3  4  2  1
 3  2  4  1
 3  2  1  4
 2  3  1  4
 2  3  4  1
 2  4  3  1
 4  2  3  1
 4  2  1  3
 2  4  1  3
 2  1  4  3
 2  1  3  4

Process returned 24 (0x18)   execution time : 1.474 s
Press any key to continue.
```

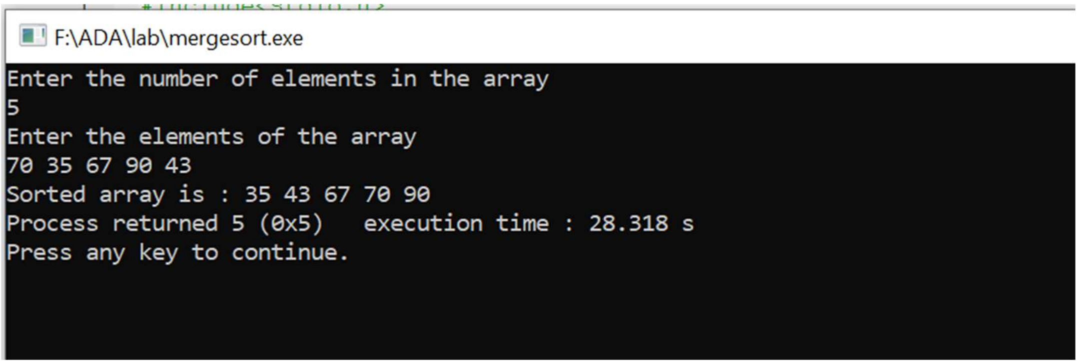
4. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Program:

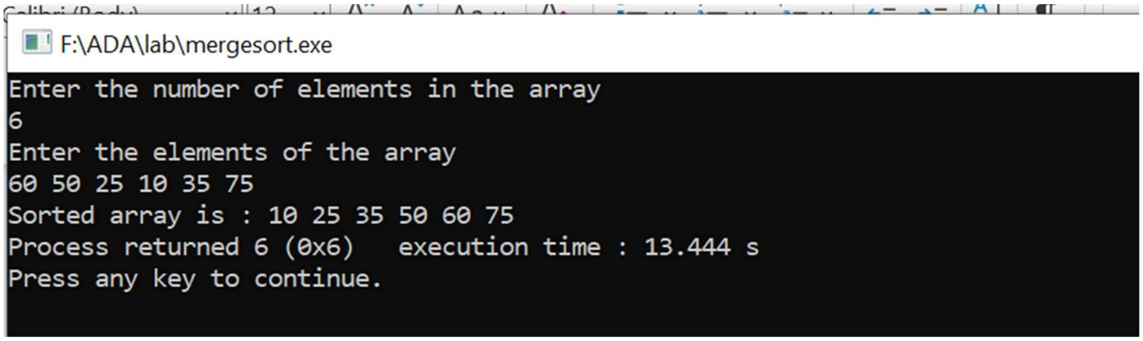
```
#include<stdio.h>
void main(){
   int low=0,high,i,n;
   int arr[15];
   printf("Enter the number of elements in the array\n");
   scanf("%d",&n);
   high=n-1;
   printf("Enter the elements of the array\n");
   for(i=0;i<n;i++){
     scanf("%d",&arr[i]);
   }
   mergeSort(low,high,n,arr);
   printf("Sorted array is : ");
   for(i=0;i<n;i++){
      printf("%d ",arr[i]);
   }
}

void mergeSort(int low,int high,int n,int arr[n])
{
   int mid;
   if(low<high)
   {
      mid=(low+high)/2;
```

```
        mergeSort(low,mid,n,arr);
        mergeSort(mid+1,high,n,arr);
        merge(low,mid,high,n,arr);
    }
}

void merge(int low,int mid,int high,int n,int arr[n])
{
    int i=low,j=mid+1,k=low,c[n];
    while(i<=mid&&j<=high)
    {
        if(arr[i]<arr[j])
        {
            c[k]=arr[i];
            i++;
            k++;
        }
        else
        {
            c[k]=arr[j];
            j++;
            k++;
        }
    }
    while(i<=mid)
    {
        c[k]=arr[i];
        i++;

        k++;

    }

    while(j<=high)

    {

        c[k]=arr[j];

        j++;

        k++;

    }

    for (i = low; i <= high; i++)

    {

        arr[i]=c[i];

    }

}
```

OUTPUT

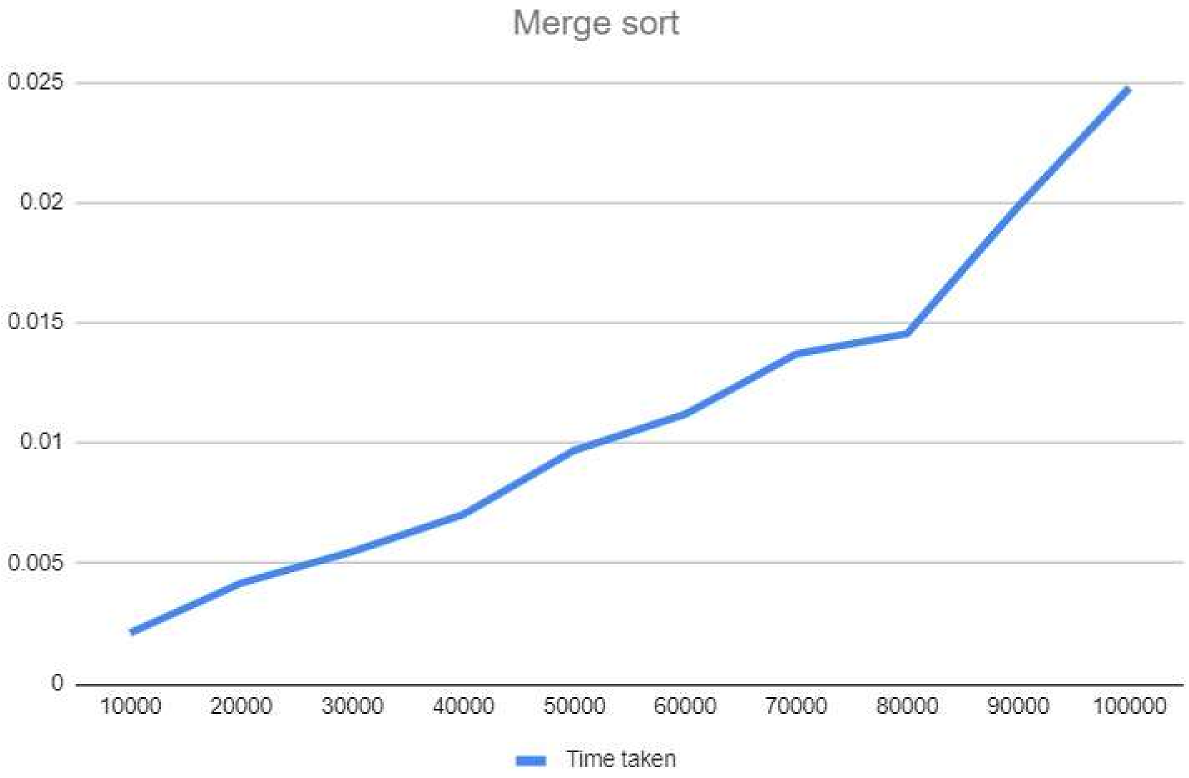


```
Enter the number of elements in the array
5
Enter the elements of the array
70 35 67 90 43
Sorted array is : 35 43 67 70 90
Process returned 5 (0x5)   execution time : 28.318 s
Press any key to continue.
```



```
Enter the number of elements in the array
6
Enter the elements of the array
60 50 25 10 35 75
Sorted array is : 10 25 35 50 60 75
Process returned 6 (0x6)   execution time : 13.444 s
Press any key to continue.
```

Graph:


Merge sort
0.025
0.02
0.015
0.01
0.005
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Time taken

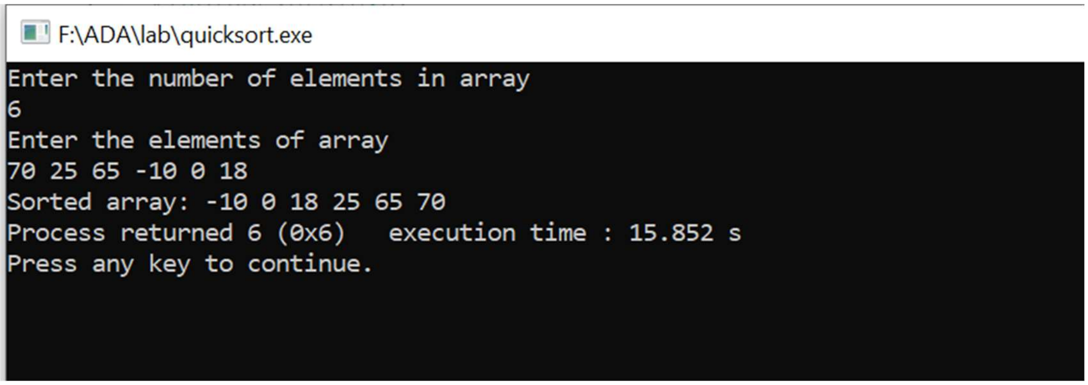
5. Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

Program:

```
#include<stdio.h>

void main(){

   int arr[20],low,high,n,i;

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

   scanf("%d",&n);

   printf("Enter the elements of array\n");

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

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

   }

   low=0;high=n-1;

   quickSort(low, high, arr);

   printf("Sorted array: ");

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

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

   }

}


void quickSort(int low,int high,int a[]){

   int j;

   if(low<high){

      j=partition(low,high,a);

      quickSort(low,j-1,a);

      quickSort(j+1,high,a);

   }

}
```

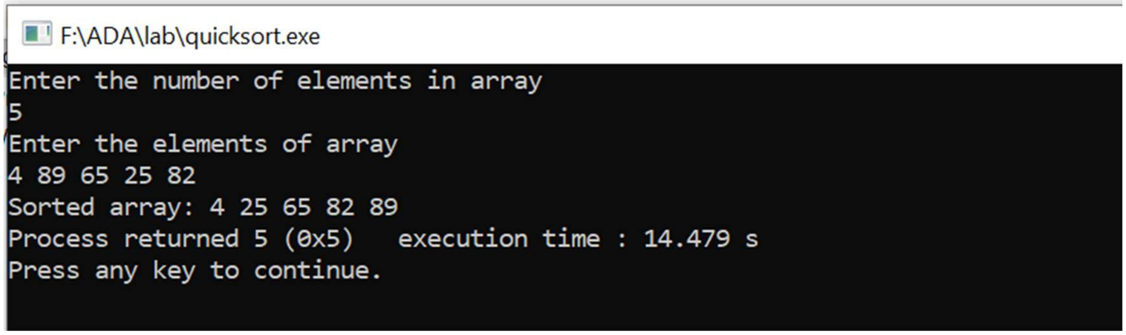
```
int partition(int low, int high, int a[]){
    int i,j,pivot,temp;
    i=low;
    j=high+1;
    pivot=a[low];
    while(i<j){
        do{
            i=i+1;
        }while(pivot>=a[i]);
        do{
            j=j-1;
        }while(pivot<a[j]);
        if(i<j){
            temp = a[i];
            a[i]=a[j];
            a[j]=temp;
        }
        if(i>j)
    {
        temp = a[low];
        a[low]=a[j];
        a[j]=temp;
    }
    }
    return j;
}
```

OUTPUT:



F:\ADA\lab\quicksort.exe

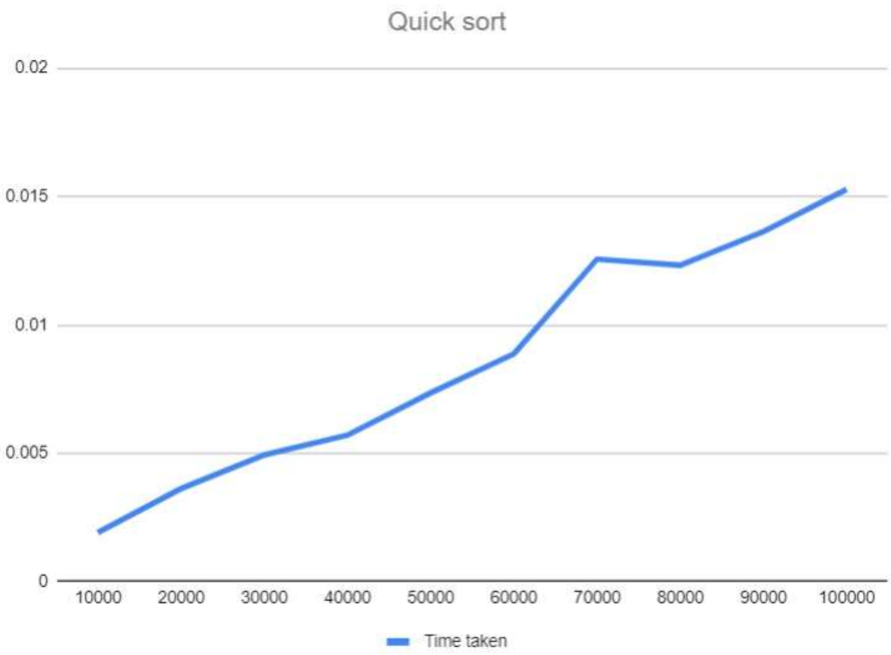
```
Enter the number of elements in array
6
Enter the elements of array
70 25 65 -10 0 18
Sorted array: -10 0 18 25 65 70
Process returned 6 (0x6)   execution time : 15.852 s
Press any key to continue.
```



F:\ADA\lab\quicksort.exe

```
Enter the number of elements in array
5
Enter the elements of array
4 89 65 25 82
Sorted array: 4 25 65 82 89
Process returned 5 (0x5)   execution time : 14.479 s
Press any key to continue.
```

Graph



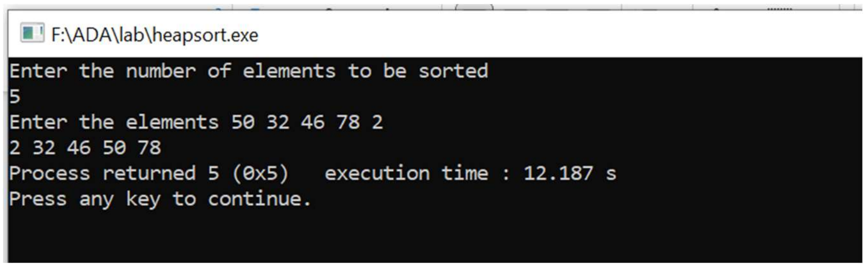
6. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

Program:

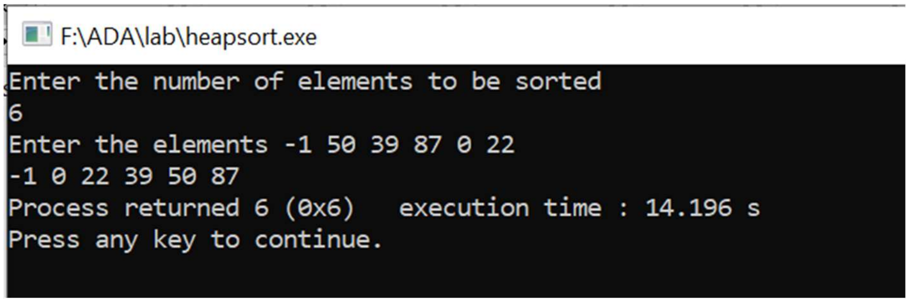
```
#include<stdio.h>
void main(){
   int n;
   printf("Enter the number of elements to be sorted\n");
   scanf("%d",&n);
   int arr[n];
   printf("Enter the elements ");
   for(int i=0;i<n;i++){
      scanf("%d",&arr[i]);
   }
   heapsort(n,arr);
   for(int i=0;i<n;i++){
      printf("%d ",arr[i]);
   }
}
void heapify(int n,int arr[n], int i){
   int largest =i;
   int left = 2*i+1;
   int 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(n,arr,largest);
    }
}
void heapsort(int n,int arr[n]){
    for(int i= n/2 -1;i>=0;i--){
        heapify(n,arr,i);
    }
    for(int i=n-1;i>0;i--){
        int temp = arr[0];
        arr[0]=arr[i];
        arr[i]=temp;
        heapify(i,arr,0);
    }
}
```

OUTPUT:

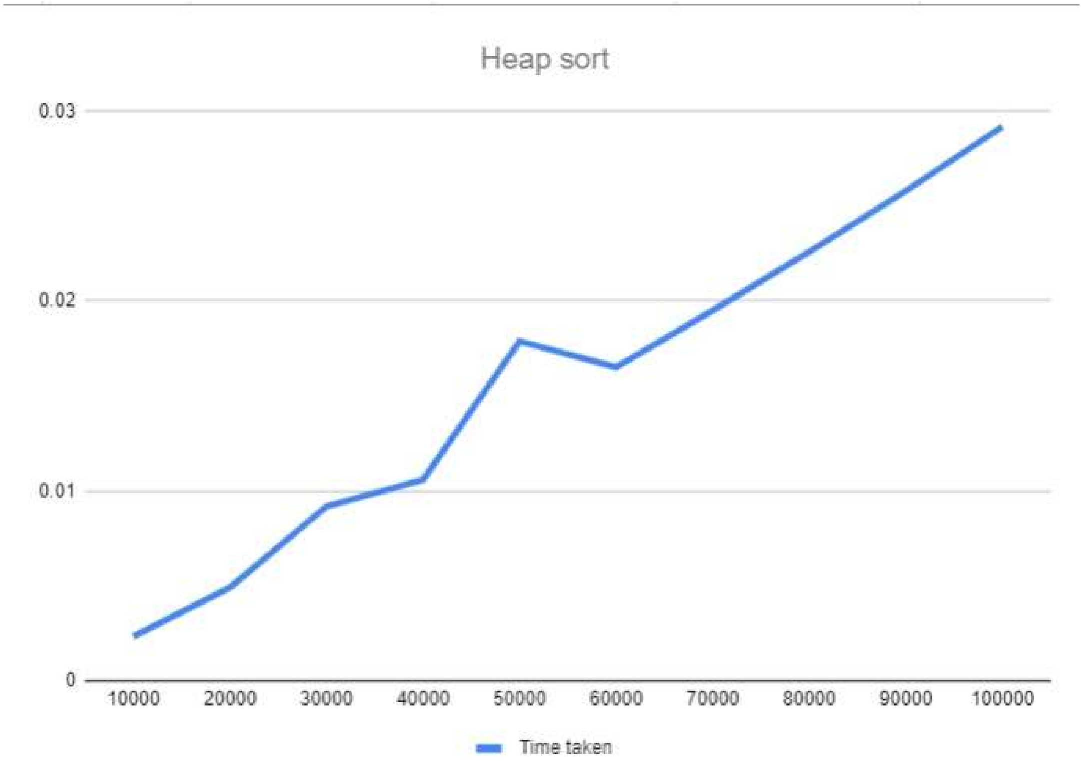


```
Enter the number of elements to be sorted
5
Enter the elements 50 32 46 78 2
2 32 46 50 78
Process returned 5 (0x5)   execution time : 12.187 s
Press any key to continue.
```



```
Enter the number of elements to be sorted
6
Enter the elements -1 50 39 87 0 22
-1 0 22 39 50 87
Process returned 6 (0x6)   execution time : 14.196 s
Press any key to continue.
```

Graph:


Heap sort
0.03
0.02
0.01
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Time taken

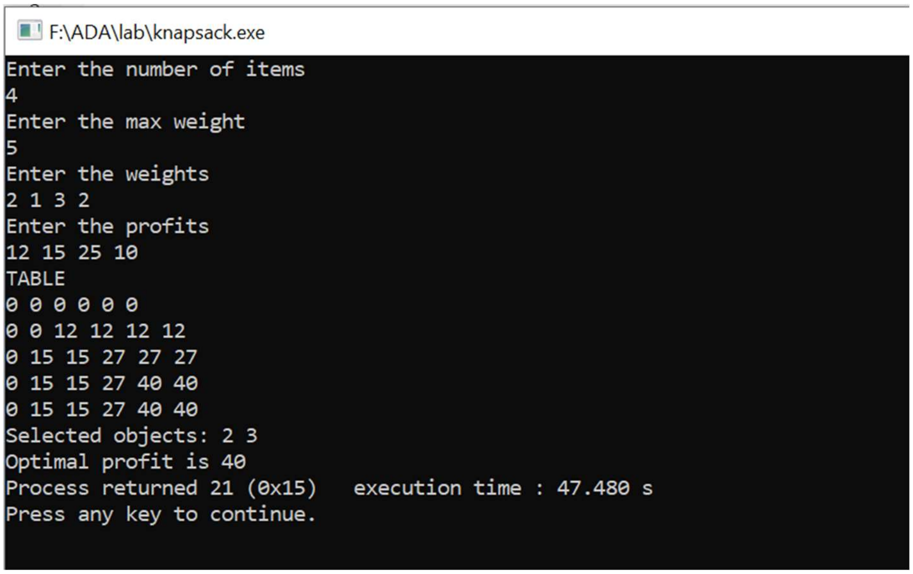
7. Implement 0/1 Knapsack problem using dynamic programming.

Program

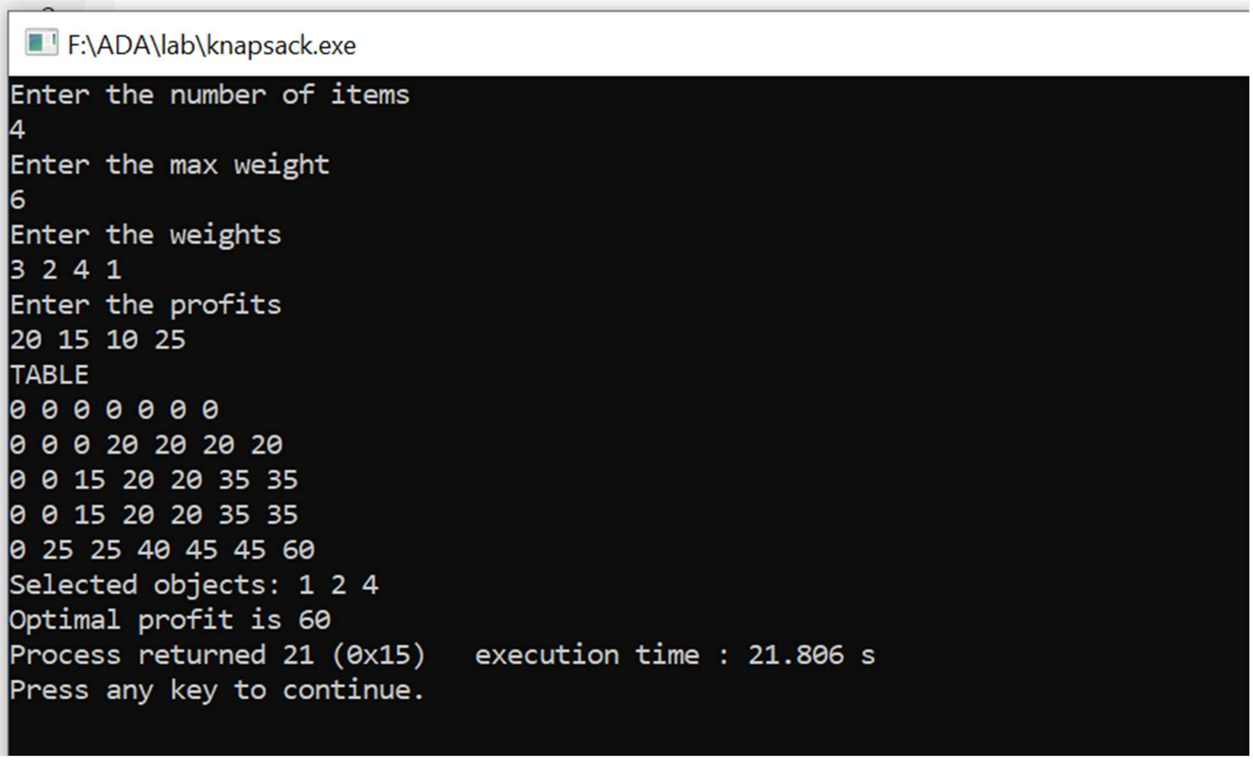
```
#include <stdio.h>

#include <stdbool.h>


int p[15],w[15],maxW;

void main(){

   int n,i,j,maxP;

   printf("Enter the number of items\n");

   scanf("%d",&n);

   printf("Enter the max weight\n");

   scanf("%d",&maxW);

   printf("Enter the weights\n");

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

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

   printf("Enter the profits\n");

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

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

   maxP=knapsack(n);

   printf("Optimal profit is %d ",maxP);

}




int knapsack(int n) {

   int v[n+1][maxW+1],i,j;
```

```
for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= maxW; j++) {
        if (i == 0 || j == 0)
            v[i][j] = 0;
        else if (w[i-1] <= j)
            v[i][j] = max(p[i - 1] + v[i - 1][j - w[i - 1]], v[i - 1][j]);
        else
            v[i][j] = v[i - 1][j];
    }
}

int selected[n];
i = n; j = maxW;
int count = 0;

while (i > 0 && j > 0) {
    if (v[i][j] != v[i - 1][j]) {
        selected[count++] = i;
        j -= w[i - 1];
        i--;
    } else {
        i--;
    }
}
printf("TABLE \n");
for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= maxW; j++) {
            printf("%d ",v[i][j]);
        }
        printf("\n");
    }
    printf("Selected objects: ");
    for (int j = count - 1; j >= 0; j--)
        printf("%d ", selected[j]);
    printf("\n");
    return v[n][maxW];
}

int max(int a, int b) {
    return (a > b) ? a : b;
}
```

OUTPUT:



```
Enter the number of items
4
Enter the max weight
5
Enter the weights
2 1 3 2
Enter the profits
12 15 25 10
TABLE
0 0 0 0 0 0
0 0 12 12 12 12
0 15 15 27 27 27
0 15 15 27 40 40
0 15 15 27 40 40
Selected objects: 2 3
Optimal profit is 40
Process returned 21 (0x15)   execution time : 47.480 s
Press any key to continue.
```



```
Enter the number of items
4
Enter the max weight
6
Enter the weights
3 2 4 1
Enter the profits
20 15 10 25
TABLE
0 0 0 0 0 0 0
0 0 0 20 20 20 20
0 0 15 20 20 35 35
0 0 15 20 20 35 35
0 25 25 40 45 45 60
Selected objects: 1 2 4
Optimal profit is 60
Process returned 21 (0x15)   execution time : 21.806 s
Press any key to continue.
```

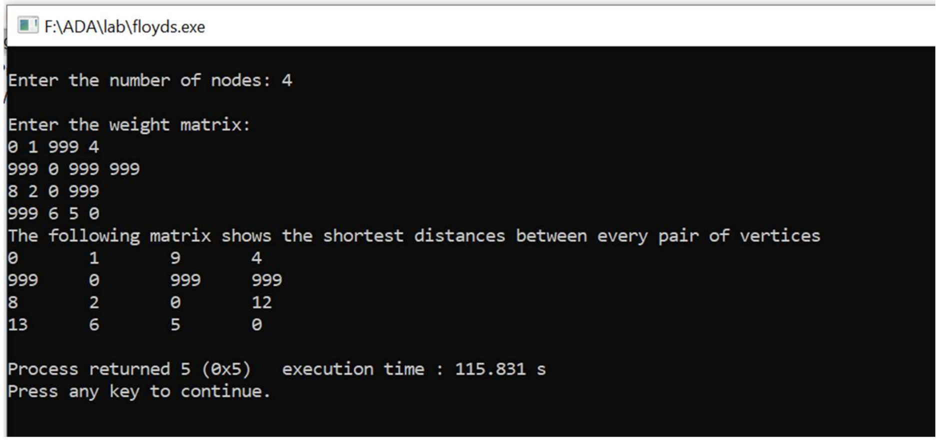
Q8)Implement All Pair Shortest paths problem using Floyd’s algorithm.

Program:

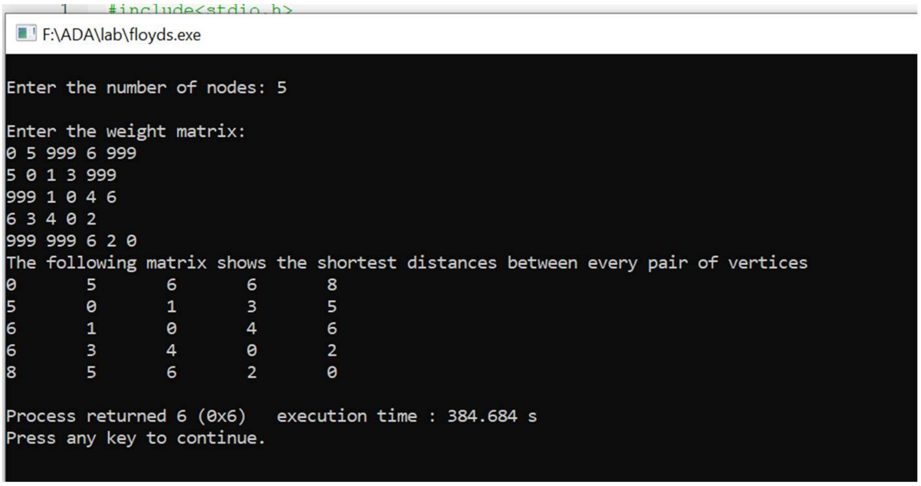
```
#include<stdio.h>
#define MAX 10


void display(int n,int w[MAX][MAX])
{
   int i,j;
   printf("The following matrix shows the shortest distances between every pair of vertices \n");
   for (int i = 1; i <= n; i++)
   {
      for (int j = 1; j <= n; j++)
      {
            printf("%d\t", w[i][j]);
      }
      printf("\n");
   }

   //printf("\n The shortest paths are:\n");
      //for(i=1;i<=n;i++)
         //for(j=1;j<=n;j++)
         //{
           // if(i!=j)
               //printf("\n <%d,%d>=%d",i,j,w[i][j]);
         //}
}
```

```
void floyds(int n,int w[MAX][MAX])
{
   int i, j, k;

   for (k = 1; k <= n; k++)
   {
      for (i = 1; i <= n; i++)
      {
         for (j = 1; j <= n; j++)
         {
            if (w[i][k] + w[k][j] < w[i][j])
               w[i][j] = w[i][k] + w[k][j];
         }
      }
   }
   display(n,w);
}

void main()
{
   int i,n,W,j;
   int w[MAX][MAX], dist[MAX][MAX] ;

   printf("\nEnter the number of nodes: ");
   scanf("%d",&n);
   printf("\nEnter the weight matrix:\n");
   for (i = 1; i <= n; i++)
   {
      for (j = 1; j <= n; j++)
      {
         scanf("%d",&w[i][j]);
      }
   }
   floyds(n,w);
}
```

OUTPUT:



```
Enter the number of nodes: 4

Enter the weight matrix:
0 1 999 4
999 0 999 999
8 2 0 999
999 6 5 0
The following matrix shows the shortest distances between every pair of vertices
0       1       9       4
999     0       999     999
8       2       0       12
13      6       5       0

Process returned 5 (0x5)   execution time : 115.831 s
Press any key to continue.
```



```
Enter the number of nodes: 5

Enter the weight matrix:
0 5 999 6 999
5 0 1 3 999
999 1 0 4 6
6 3 4 0 2
999 999 6 2 0
The following matrix shows the shortest distances between every pair of vertices
0       5       6       6       8
5       0       1       3       5
6       1       0       4       6
6       3       4       0       2
8       5       6       2       0

Process returned 6 (0x6)   execution time : 384.684 s
Press any key to continue.
```

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

Prim’s

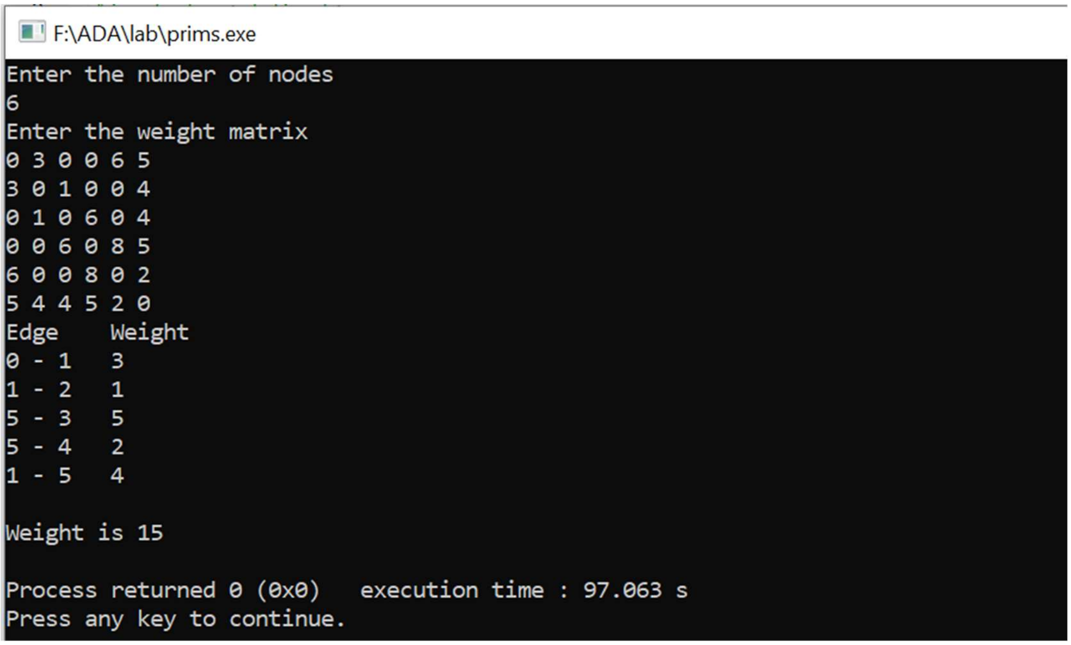
Program:

```
#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 10


int minKey(int key[], bool mstSet[], int n)

{

        int min = INT_MAX, min_index;


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

                if (mstSet[v] == false && key[v] < min)

                        min = key[v], min_index = v;


        return min_index;

}


int printMST(int parent[], int graph[V][V], int n)

{

  int weight = 0;

        printf("Edge \tWeight\n");

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

                printf("%d - %d \t%d \n", parent[i], i,

                        graph[i][parent[i]]);
```

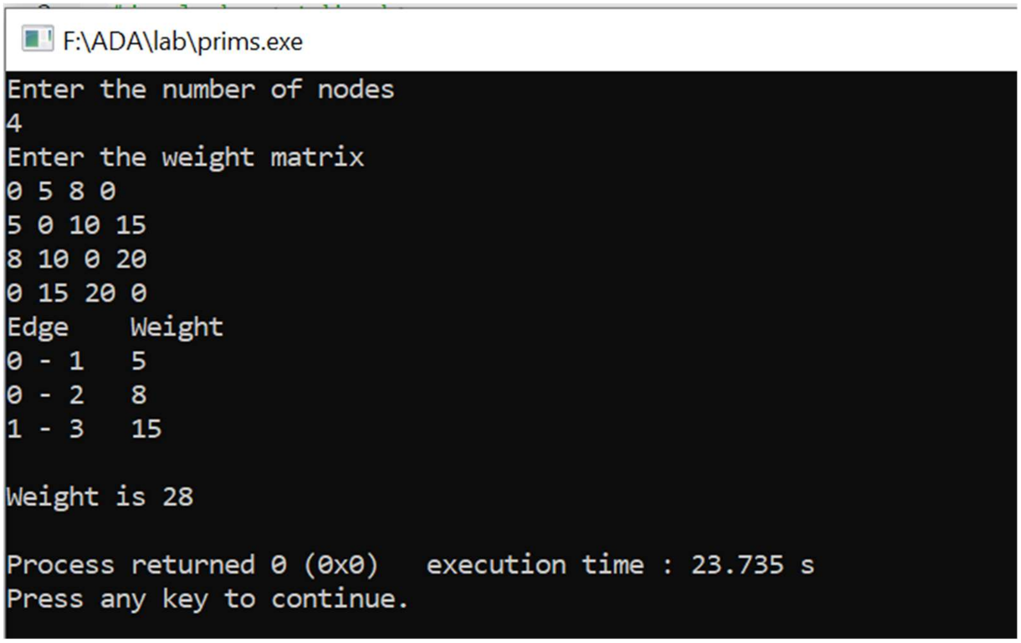
```
    for(int i=1;i<n;i++){
        weight += graph[i][parent[i]];
    }
    printf("\nWeight is %d \n",weight);
}

void prims(int graph[V][V],int n)
{
	int parent[V];
	int key[V];
	bool mstSet[V];

	for (int i = 0; i < n; i++)
		key[i] = INT_MAX, mstSet[i] = false;

	key[0] = 0;

	parent[0] = -1;

	for (int count = 0; count < n - 1; count++) {

		int u = minKey(key, mstSet, n);

		mstSet[u] = true;

		for (int v = 0; v < n; v++)
```

```
                    if (graph[u][v] && mstSet[v] == false
                        && graph[u][v] < key[v])
                        parent[v] = u, key[v] = graph[u][v];
        }

        printMST(parent, graph, n);
}

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

        prims(graph,n);
        return 0;
}
```

OUTPUT:



```
Enter the number of nodes
6
Enter the weight matrix
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 2
5 4 4 5 2 0
Edge    Weight
0 - 1   3
1 - 2   1
5 - 3   5
5 - 4   2
1 - 5   4

Weight is 15

Process returned 0 (0x0)   execution time : 97.063 s
Press any key to continue.
```



```
Enter the number of nodes
4
Enter the weight matrix
0 5 8 0
5 0 10 15
8 10 0 20
0 15 20 0
Edge    Weight
0 - 1   5
0 - 2   8
1 - 3   15

Weight is 28

Process returned 0 (0x0)   execution time : 23.735 s
Press any key to continue.
```

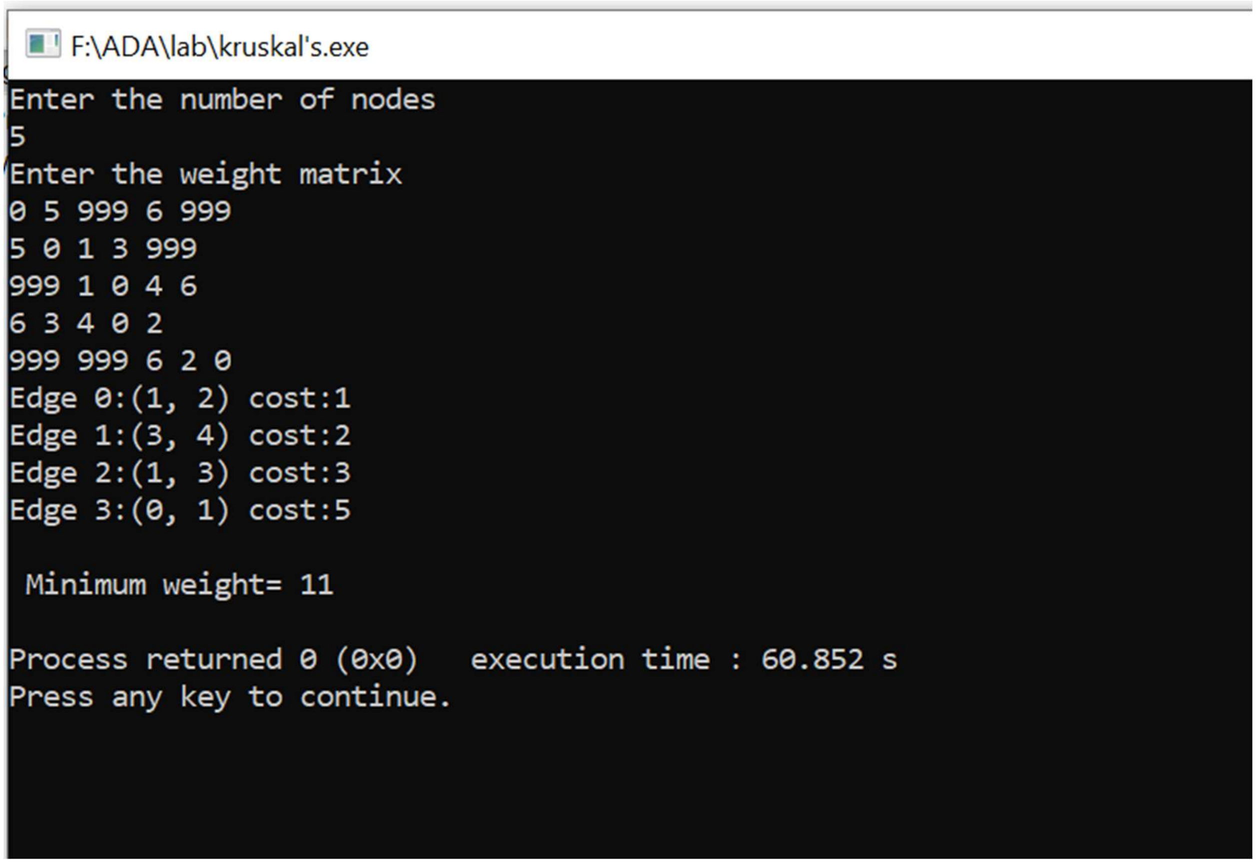
Kruskal's

Program:

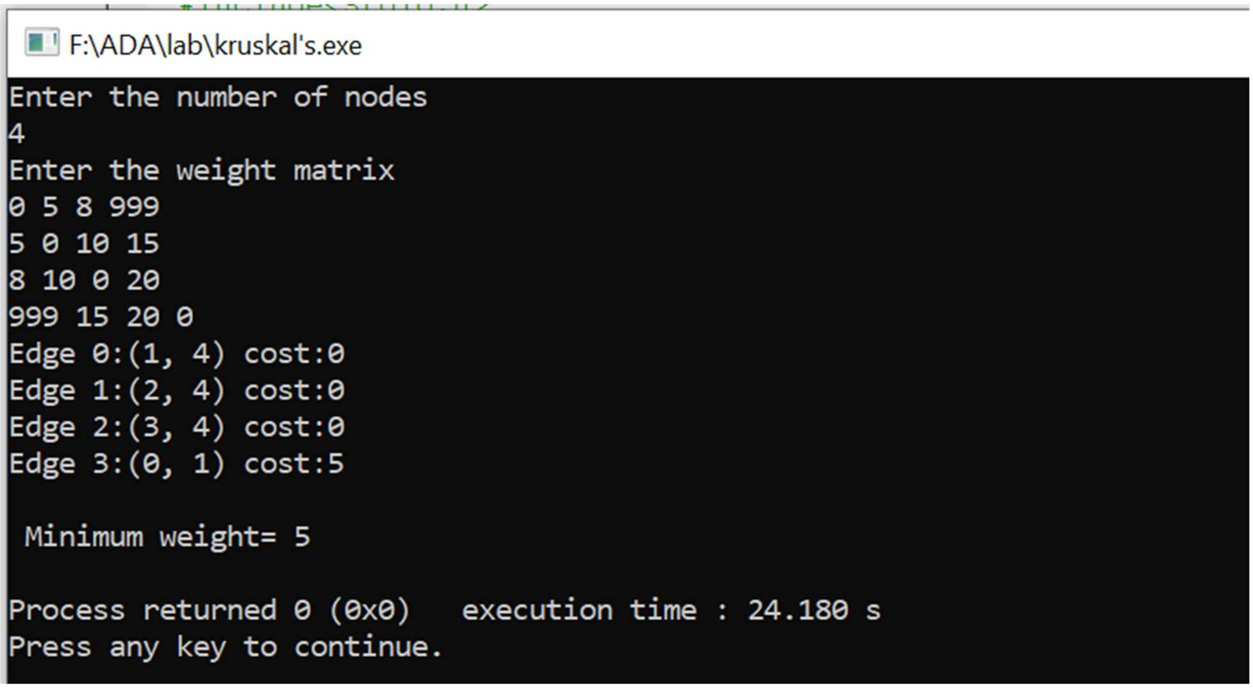
```
#include<stdio.h>
```

```
#include <stdbool.h>
#define INT_MAX 99
#define V 5
int n;
int parent[V];
int find(int i)
{
    while (parent[i] != i)
        i = parent[i];
    return i;
}

void union1(int i, int j)
{
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}

void kruskalMST(int cost[][V])
{
    int mincost = 0;

    for (int i = 0; i < V; i++)
        parent[i] = i;

    int edge_count = 0;
    while (edge_count < V - 1) {
        int min = INT_MAX, a = -1, b = -1;
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (find(i) != find(j) && cost[i][j] < min) {
                    min = cost[i][j];
                    a = i;
                    b = j;
                }
            }
        }

        union1(a, b);
        printf("Edge %d:(%d, %d) cost:%d \n",
               edge_count++, a, b, min);
        mincost += min;
    }
    printf("\n Minimum weight= %d \n", mincost);

}

int main()
{

    int cost[V][V];
    printf("Enter the number of nodes\n");
    scanf("%d",&n);
    printf("Enter the weight matrix\n");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++)
            scanf("%d",&cost[i][j]);
    }

    kruskalMST(cost);

    return 0;
}
```

Output:



```
Enter the number of nodes
5
Enter the weight matrix
0 5 999 6 999
5 0 1 3 999
999 1 0 4 6
6 3 4 0 2
999 999 6 2 0
Edge 0:(1, 2) cost:1
Edge 1:(3, 4) cost:2
Edge 2:(1, 3) cost:3
Edge 3:(0, 1) cost:5

 Minimum weight= 11

Process returned 0 (0x0)   execution time : 60.852 s
Press any key to continue.
```



```
Enter the number of nodes
4
Enter the weight matrix
0 5 8 999
5 0 10 15
8 10 0 20
999 15 20 0
Edge 0:(1, 4) cost:0
Edge 1:(2, 4) cost:0
Edge 2:(3, 4) cost:0
Edge 3:(0, 1) cost:5

 Minimum weight= 5

Process returned 0 (0x0)   execution time : 24.180 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.

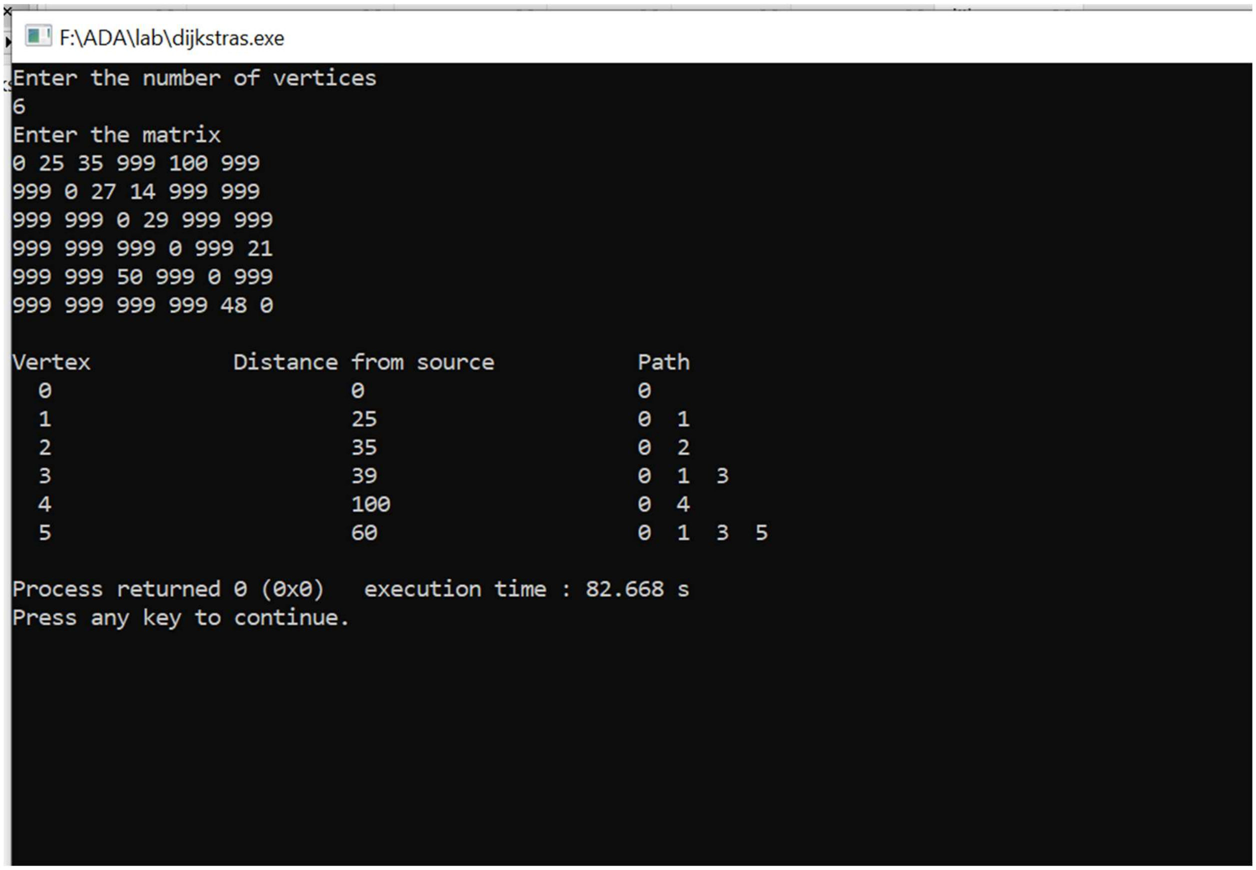
Program:

```
#include <stdbool.h>
#include <stdio.h>
#define MAX 999
int V;
int parents[50], noParent=-1;

int minDistance(int dist[], bool picked[])
{
	int min = MAX, min_index;

	for (int v = 0; v < V; v++)
		if (picked[v] == false && dist[v] <= min)
			min = dist[v], min_index = v;

	return min_index;
}
void printPath(int vertx,int parents[V])
{
	if (vertx == noParent) {
		return;
	}
	printPath(parents[vertx], parents);
	printf("%d  ",vertx);
}
void printSolution(int dist[])
{
	printf("\nVertex \t\t Distance from source\t\tPath\n");
	for (int i = 0; i < V; i++){
      printf("  %d \t\t\t  %d \t\t\t", i, dist[i]);
		printPath(i,parents);
		printf("\n");
	}
}

void dijkstra(int graph[V][V], int src)
```

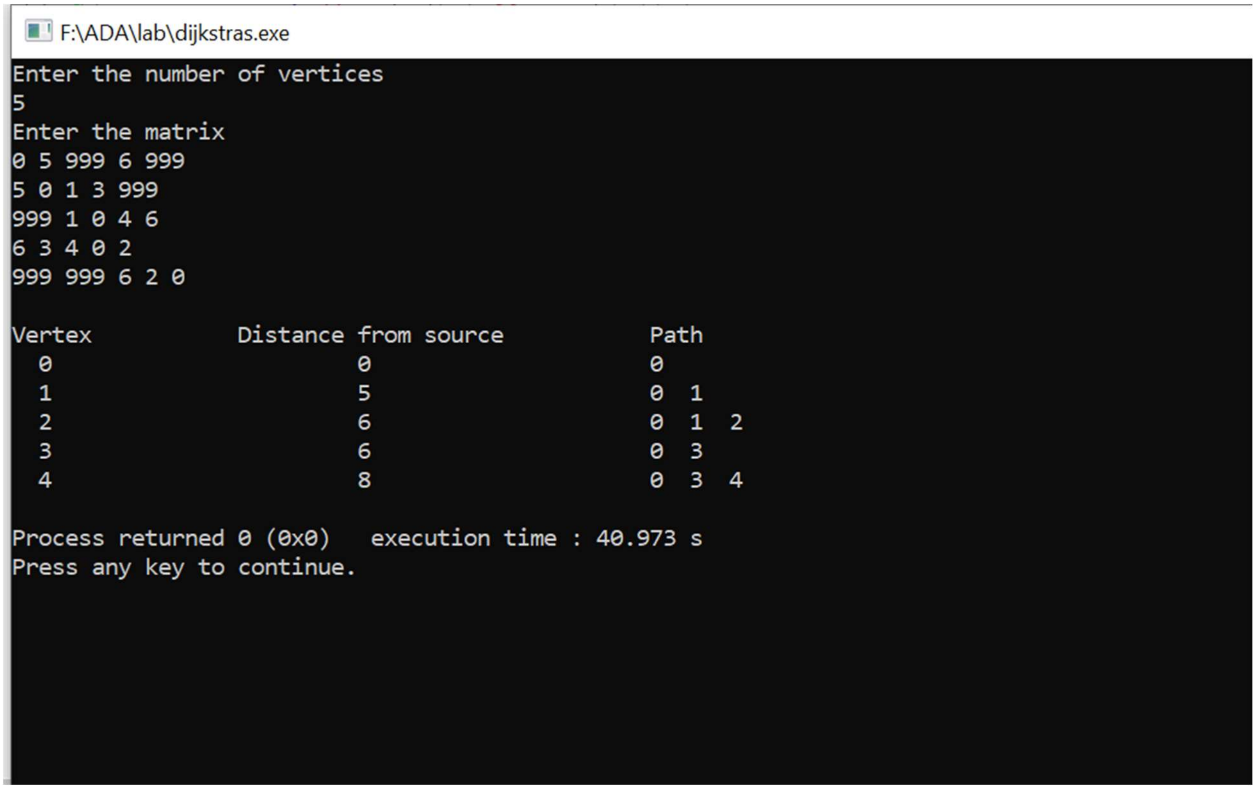
```
{
            int dist[V];

            bool picked[V];
            for (int i = 0; i < V; i++)
                        dist[i] = MAX, picked[i] = false;

            dist[src] = 0;
   parents[0]=noParent;
            for (int count = 0; count < V - 1; count++) {
                        int u = minDistance(dist, picked);

                        picked[u] = true;

                        for (int v = 0; v < V; v++){
           if (!picked[v] && graph[u][v]
                                    && dist[u] != MAX
                                    && dist[u] + graph[u][v] < dist[v]){
                                       dist[v] = dist[u] + graph[u][v];
                                    parents[v]=u;
                                    }
                        }
            }

            printSolution(dist);
}

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

```
        dijkstra(graph, 0);

        return 0;
}
```

Output:



```
Enter the number of vertices
6
Enter the 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

Vertex           Distance from source           Path
  0                       0                     0
  1                       25                    0  1
  2                       35                    0  2
  3                       39                    0  1  3
  4                       100                   0  4
  5                       60                    0  1  3  5

Process returned 0 (0x0)   execution time : 82.668 s
Press any key to continue.
```



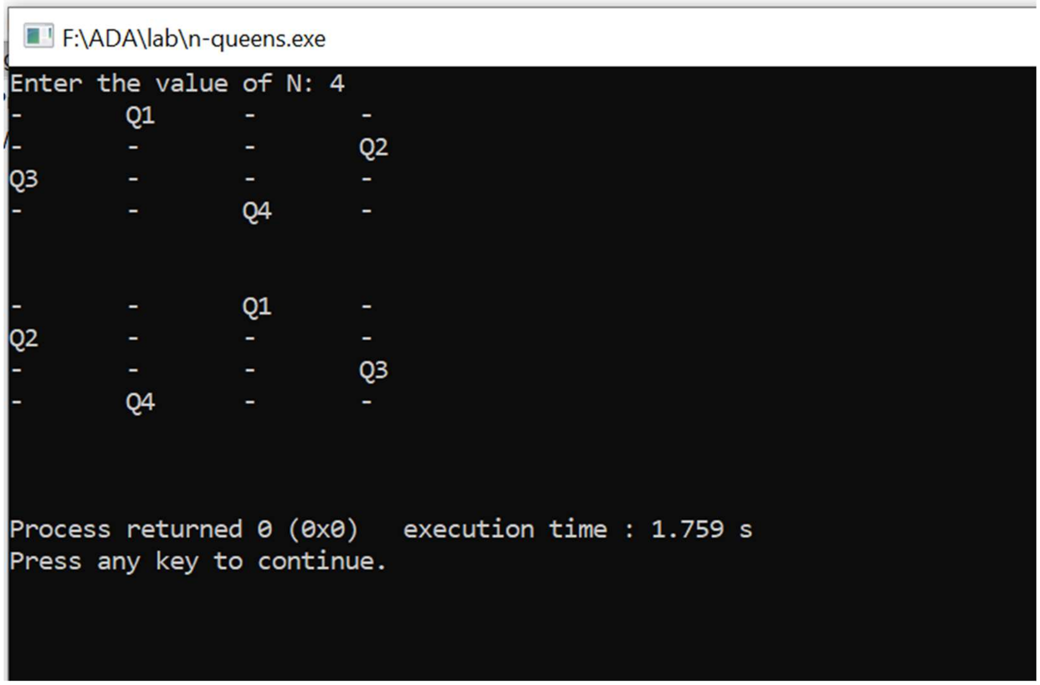
F:\ADA\lab\dijkstras.exe

```
Enter the number of vertices
5
Enter the matrix
0 5 999 6 999
5 0 1 3 999
999 1 0 4 6
6 3 4 0 2
999 999 6 2 0

Vertex           Distance from source           Path
  0                       0                     0
  1                       5                     0  1
  2                       6                     0  1  2
  3                       6                     0  3
  4                       8                     0  3  4

Process returned 0 (0x0)   execution time : 40.973 s
Press any key to continue.
```

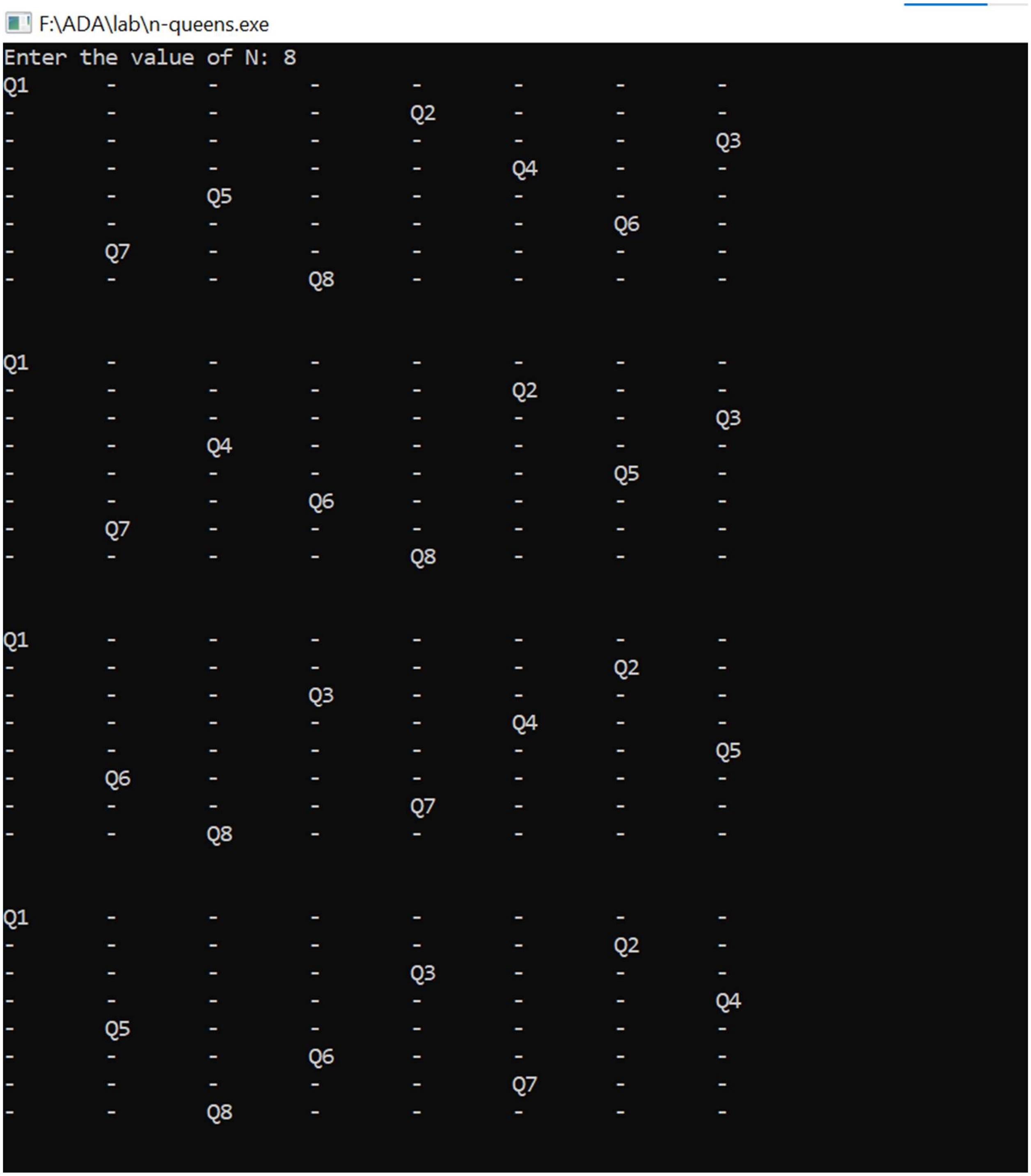
11. Implement “N-Queens Problem” using Backtracking.

```
#include <stdio.h>
#define MAX 10
int x[MAX];

int place(int k) {
    int i;
    for (i = 1; i < k; i++) {
        if (x[i] == x[k] || i - x[i] == k - x[k] || i + x[i] == k + x[k]) {
            return 0;
        }
    }
    return 1;
}

void write(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (j == x[i])
                printf("Q%d\t",i);
            else
                printf("-\t");
        }
        printf("\n");
    }
    printf("\n\n");
}

void nqueens(int n) {
    int k = 1;
    x[k] = 0;

    while (k != 0) {
        x[k] = x[k] + 1;

        while (x[k] <= n && !place(k)) {
            x[k] = x[k] + 1;
        }
        if (x[k] <= n) {
            if (k == n) {
                write(n);
            } else {
                k = k + 1;
                x[k] = 0;
            }
        } else {
            k = k - 1;
        }
    }
}

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

Output:



```
Enter the value of N: 4
-       Q1      -       -
-       -       -       Q2
Q3      -       -       -
-       -       Q4      -


-       -       Q1      -
Q2      -       -       -
-       -       -       Q3
-       Q4      -       -



Process returned 0 (0x0)   execution time : 1.759 s
Press any key to continue.
```



```
Enter the value of N: 8
Q1      -       -       -       -       -       -       -
-       -       -       -       Q2      -       -       -
-       -       -       -       -       -       -       Q3
-       -       -       -       -       Q4      -       -
-       -       Q5      -       -       -       -       -
-       -       -       -       -       -       Q6      -
-       Q7      -       -       -       -       -       -
-       -       -       Q8      -       -       -       -


Q1      -       -       -       -       -       -       -
-       -       -       -       -       Q2      -       -
-       -       -       -       -       -       -       Q3
-       -       Q4      -       -       -       -       -
-       -       -       -       -       -       Q5      -
-       -       -       Q6      -       -       -       -
-       Q7      -       -       -       -       -       -
-       -       -       -       Q8      -       -       -


Q1      -       -       -       -       -       -       -
-       -       -       -       -       -       Q2      -
-       -       -       Q3      -       -       -       -
-       -       -       -       -       Q4      -       -
-       -       -       -       -       -       -       Q5
-       Q6      -       -       -       -       -       -
-       -       -       -       Q7      -       -       -
-       -       Q8      -       -       -       -       -


Q1      -       -       -       -       -       -       -
-       -       -       -       -       -       Q2      -
-       -       -       -       Q3      -       -       -
-       -       -       -       -       -       -       Q4
-       Q5      -       -       -       -       -       -
-       -       -       Q6      -       -       -       -
-       -       -       -       -       Q7      -       -
-       -       Q8      -       -       -       -       -
```