VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

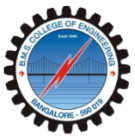
LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Deeksha S (1BM21CS048)

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 **Deeksha S (1BM21CS048),** 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	Dr. Jyothi S Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	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.

PROGRAM 1

Write program to do the following:

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

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

CODE:

a. BFS:

```
#include <stdio.h>
int am[100][100], q[100], visit[100], front = 1, rear = 0;
void bfs(int v,int n)
{
   int i;
   printf("%d ", v);
   visit[v] = 1;
   q[++rear] = v;

   while (front <= rear)
   {
      v = q[front++];
      for (i = 1; i <= n; i++)
      {
         if (am[v][i] && !visit[i])
         {
            printf("%d ", i);
            q[++rear] = i;
            visit[i] = 1;
         }
      }
```

```
    }
}

int main()
{
    int i, j,n,start;
    printf("Enter the number of nodes:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++)
    {
        visit[i] = 0;
        q[i] = 0;
        for (j = 1; j <= n; j++)
        {
            scanf("%d", &am[i][j]);
        }
    }
    printf("Enter the starting node:\n");
    scanf("%d",&start);
    printf("BFS Traversal: ");
    bfs(start,n);
    printf("\n");
    return 0;
}
```

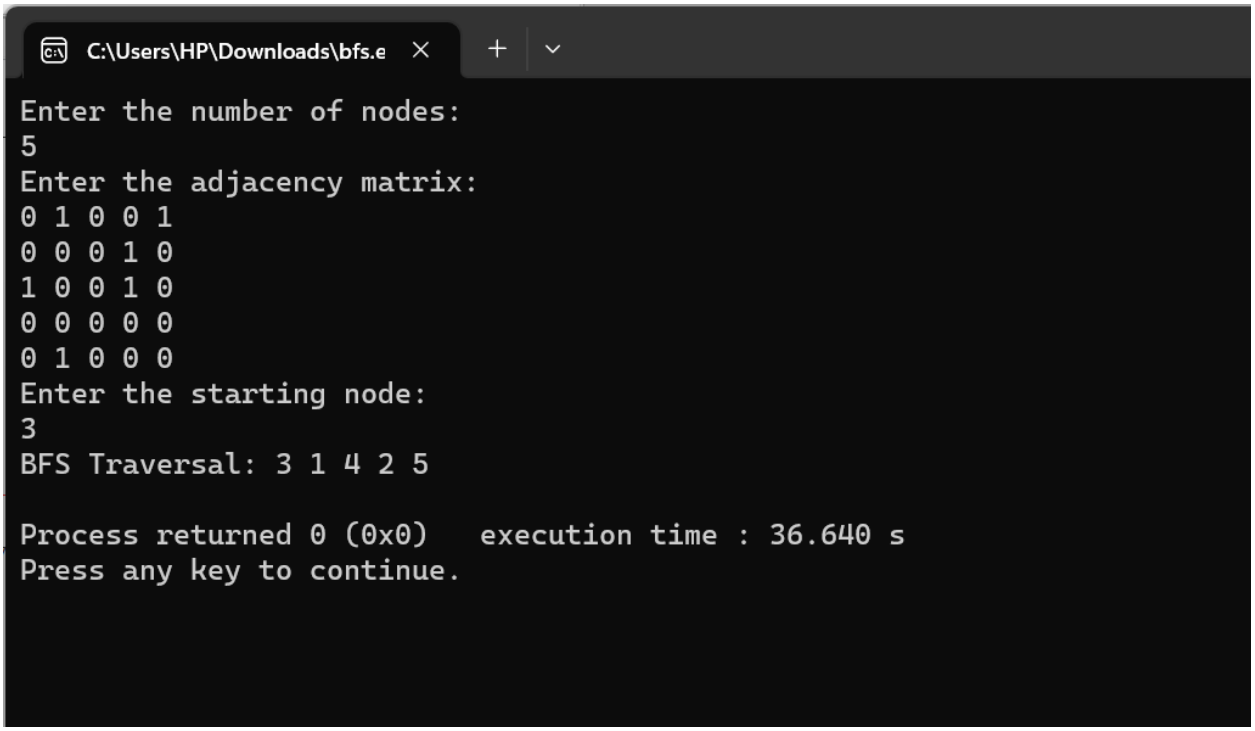
b. DFS:

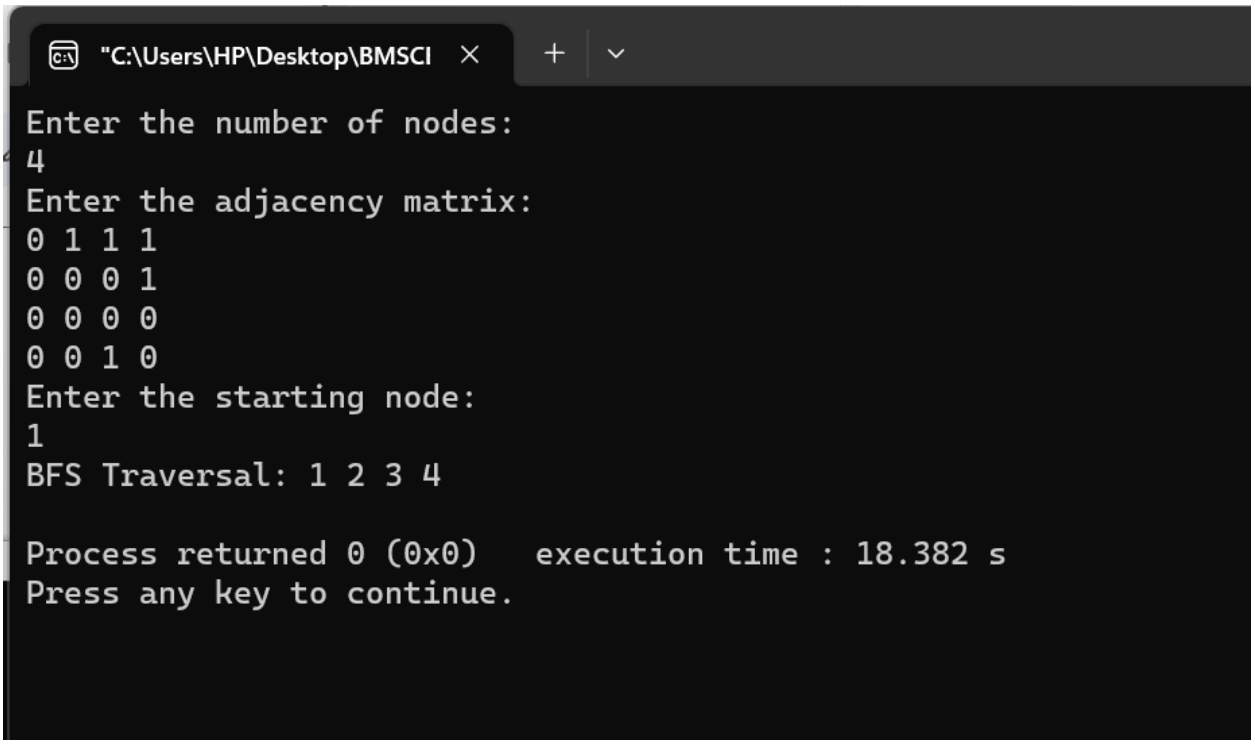
```
#include <stdio.h>
//define n 4
int am[100][100],visit[100];

void dfs(int v,int n)
{
   int i;
   printf("\n %d ",v);
   visit[v]=1;
   for(i=1;i<=n;i++)
   {
      if(am[v][i]&&!visit[i])
      {
         dfs(i,n);//printf("\n %d -> %d ",v,i);
      }
   }
}

void main()
{
   int i,j,n,start;
   printf("Enter the number of nodes:\n");
   scanf("%d",&n);
   printf("Enter the adjacency matrix:\n");
   for(i=1;i<=n;i++)
   {
      visit[i]=0;
      for(j=1;j<=n;j++)
```

```
        {
            scanf("%d",&am[i][j]);
        }
    }
    printf("Enter the starting node:\n");
    scanf("%d",&start);
    printf("DFS Traversal: ");
    dfs(start,n);
}
```

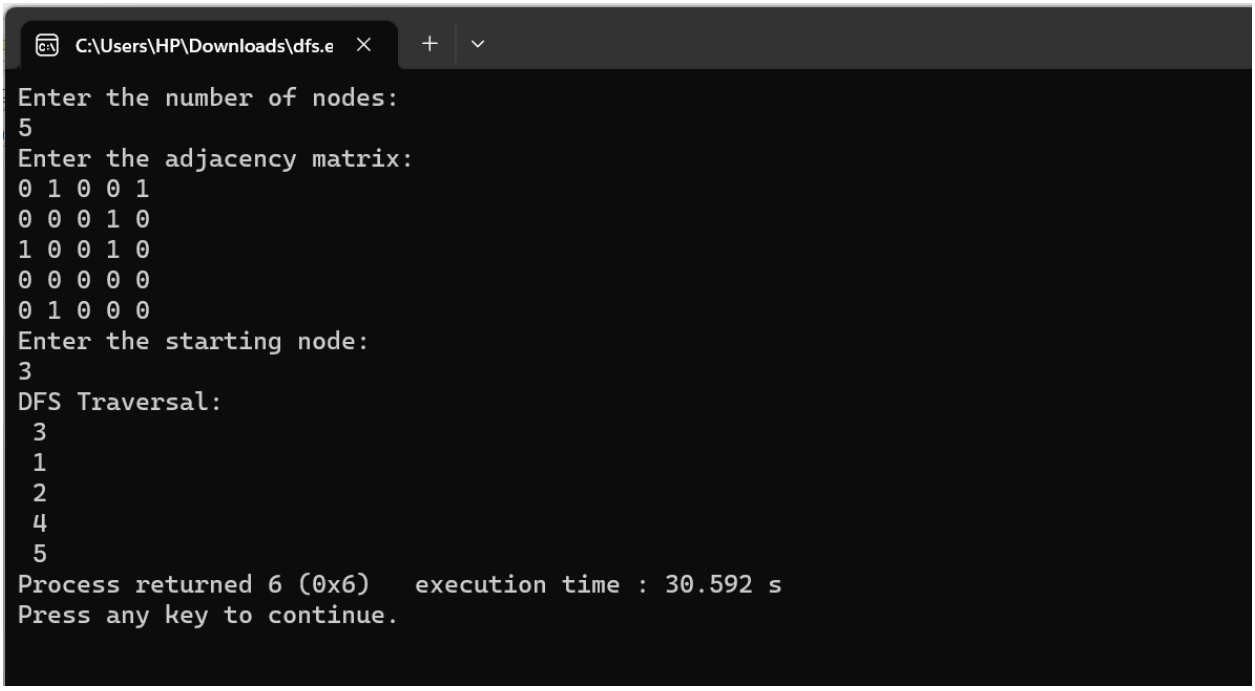
OUTPUT:

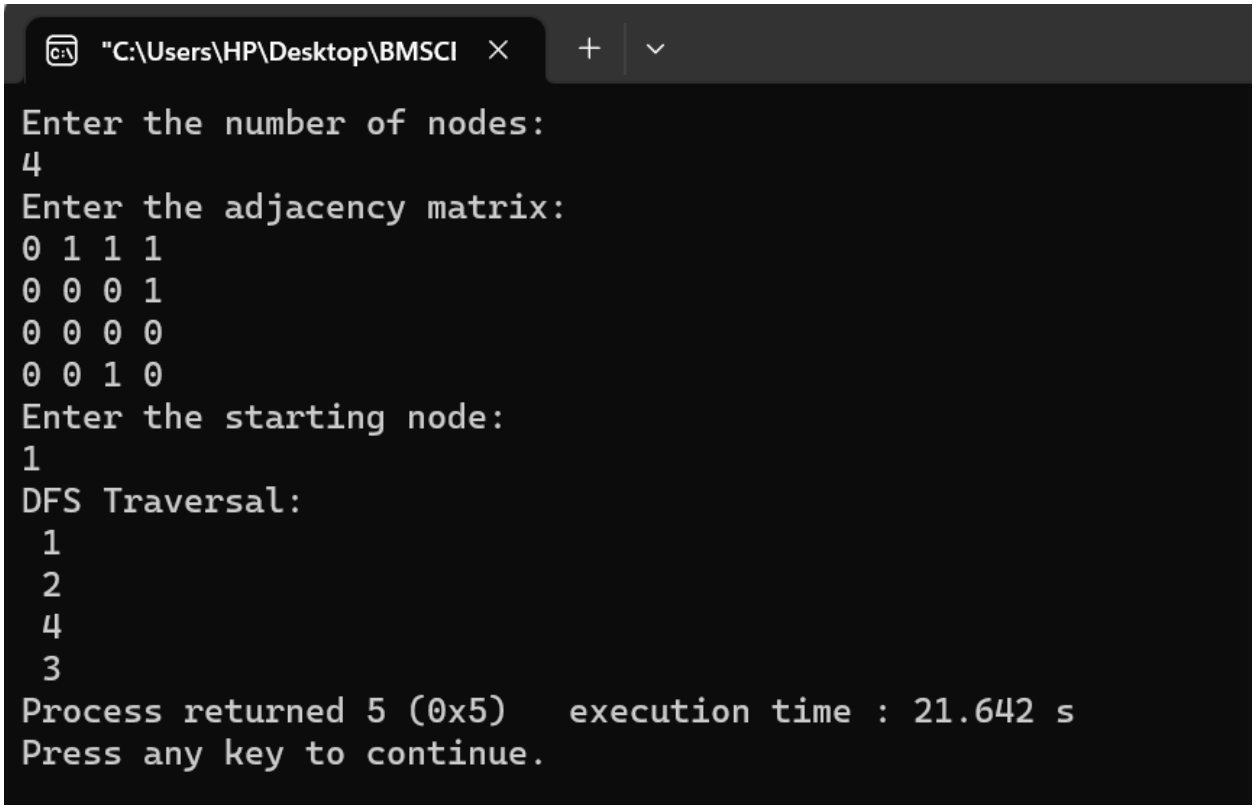
BFS:


C:\Users\HP\Downloads\bfs.e
Enter the number of nodes:
5
Enter the adjacency 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 starting node:
3
BFS Traversal: 3 1 4 2 5
Process returned 0 (0x0) execution time : 36.640 s
Press any key to continue.


"C:\Users\HP\Desktop\BMSCI
Enter the number of nodes:
4
Enter the adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Enter the starting node:
1
BFS Traversal: 1 2 3 4
Process returned 0 (0x0) execution time : 18.382 s
Press any key to continue.

DFS :


C:\Users\HP\Downloads\dfs.e
Enter the number of nodes:
5
Enter the adjacency 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 starting node:
3
DFS Traversal:
3
1
2
4
5
Process returned 6 (0x6) execution time : 30.592 s
Press any key to continue.


"C:\Users\HP\Desktop\BMSCI
Enter the number of nodes:
4
Enter the adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Enter the starting node:
1
DFS Traversal:
1
2
4
3
Process returned 5 (0x5) execution time : 21.642 s
Press any key to continue.

PROGRAM 2

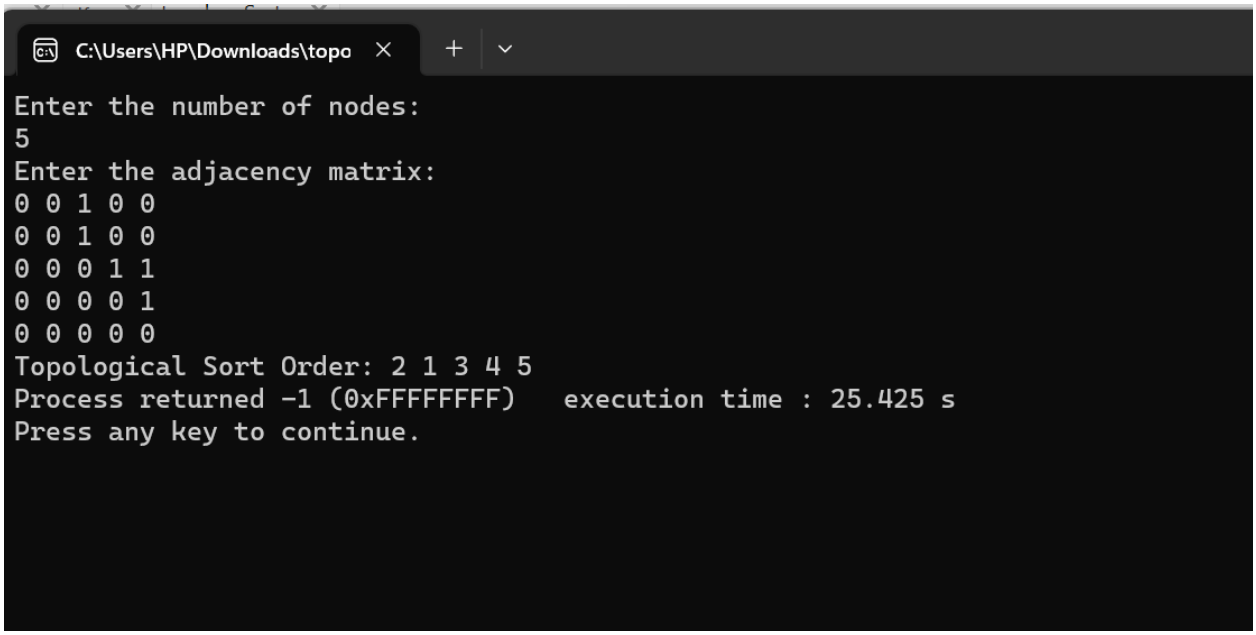
Write a program to obtain the Topological ordering of vertices in a given Digraph.

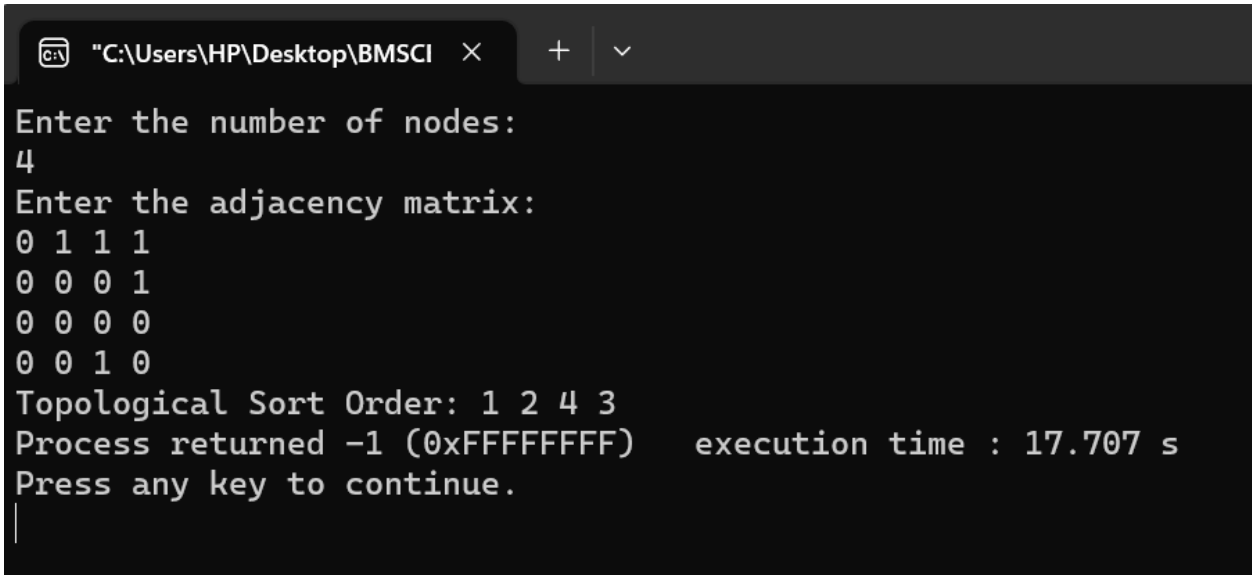
CODE:

```
#include <stdio.h>
#define max 100
int visit[max],stack[max],top=-1;

void dfs(int v,int n,int am[max][max])
{
   int i;
   //printf("\n %d ",v);
   visit[v]=1;
   for(i=1;i<=n;i++)
   {
      if(am[v][i]&&!visit[i])
      {
         dfs(i,n,am);//printf("\n %d -> %d ",v,i);
      }
   }
   stack[++(top)] = v;
}

void topologicalSort(int n,int am[max][max])
{
   int i,j;
   for (i = 1; i <=n ; i++)
   {
```

```
        if (!visit[i])//if not visited
            dfs(i, n, am);
    }
    printf("Topological Sort Order: ");
    while (top >= 0)
    {
        printf("%d ", stack[top--]);
    }
}

void main()
{
    int am[max][max];
    int i,j,n;
    printf("Enter the number of nodes:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
    {
        visit[i]=0;
        for(j=1;j<=n;j++)
        {
            scanf("%d",&am[i][j]);
        }
    }
    topologicalSort(n,am);
}
```

OUTPUT:


C:\Users\HP\Downloads\topo
Enter the number of nodes:
5
Enter the adjacency 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 Order: 2 1 3 4 5
Process returned -1 (0xFFFFFFFF)   execution time : 25.425 s
Press any key to continue.


"C:\Users\HP\Desktop\BMSCI
Enter the number of nodes:
4
Enter the adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Topological Sort Order: 1 2 4 3
Process returned -1 (0xFFFFFFFF)   execution time : 17.707 s
Press any key to continue.

PROGRAM 3

Implement Johnson Trotter algorithm to generate permutations.

CODE:

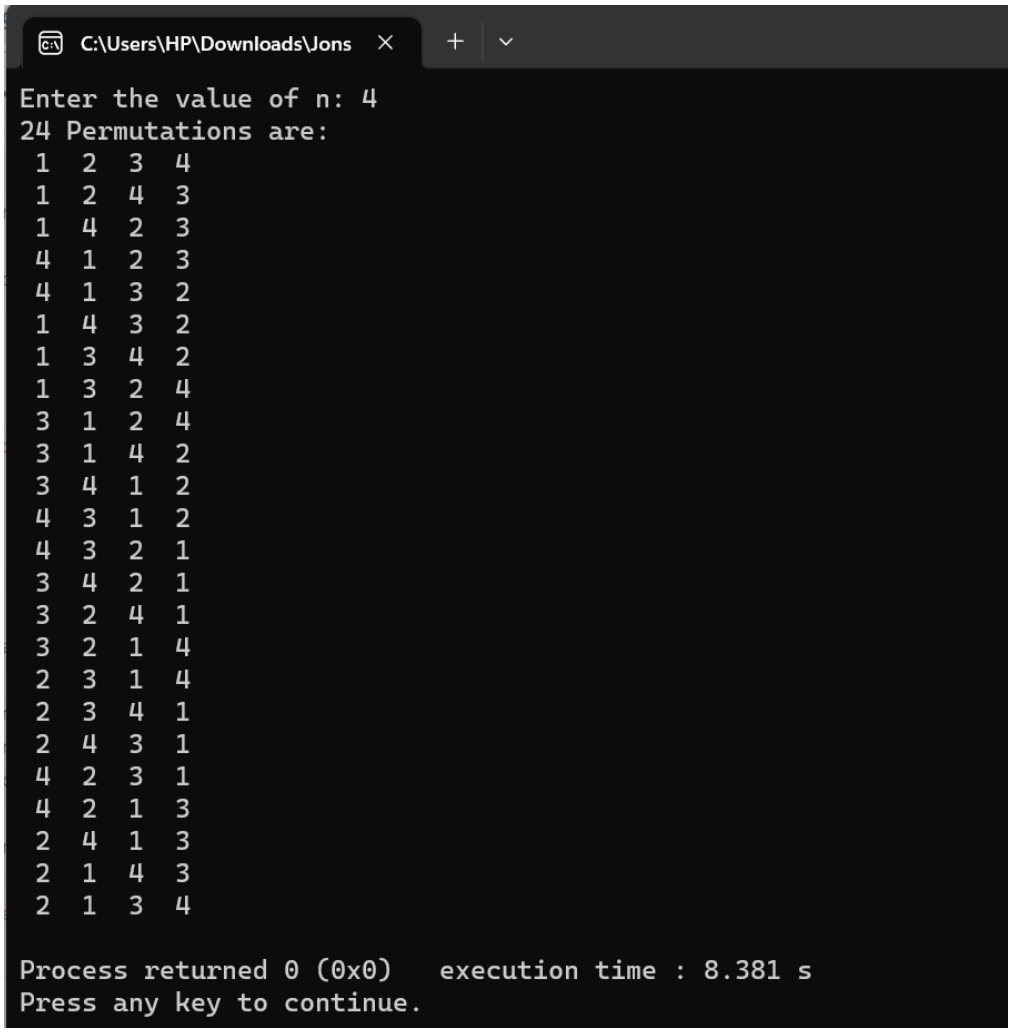
```
#include<stdio.h>
#include<stdlib.h>
#define RIGHT_TO_LEFT 0
#define LEFT_TO_RIGHT 1
void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

int searchArray(int array[], int n, int mobile)
{
   for (int i = 0; i < n; i++)
   {
      if (array[i] == mobile)
         return i + 1;
   }
   return -1;  // Mobile not found
}

int getMobile(int array[], int dir[], int n)
{
   int mobile_prev = 0, mobile = 0;
```

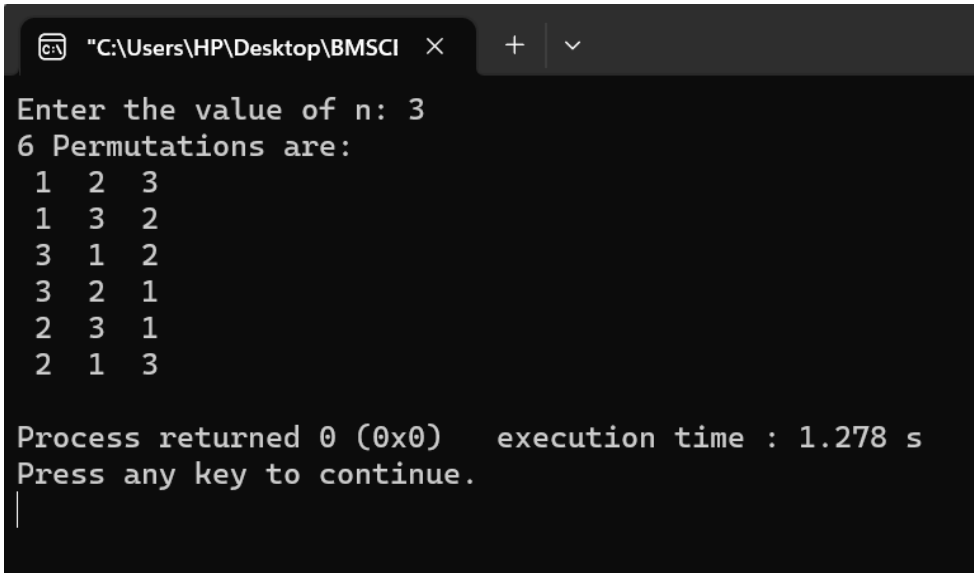
```
    for (int i = 0; i < n; i++)
        {
        // Direction 0 represents RIGHT TO LEFT.
        if (dir[array[i] - 1] == RIGHT_TO_LEFT && i != 0)
            {
            if (array[i] > array[i - 1] && array[i] > mobile_prev)
            {
                mobile = array[i];
                mobile_prev = mobile;
            }
        }
        if (dir[array[i] - 1] == LEFT_TO_RIGHT && i != n - 1)
        {
            if (array[i] > array[i + 1] && array[i] > mobile_prev)
            {
                mobile = array[i];
                mobile_prev = mobile;
            }
        }
    }
    if (mobile == 0 && mobile_prev == 0)
        return 0;  // No mobile element found
    else
        return mobile;
}

void printOnePermutation(int array[], int dir[], int n, int pnum)
{
    int mobile = getMobile(array, dir, n);
```

```
    int pos = searchArray(array, n, mobile);
    if (dir[array[pos - 1] - 1] == RIGHT_TO_LEFT)
        swap(&array[pos - 1], &array[pos - 2]);
    else if (dir[array[pos - 1] - 1] == LEFT_TO_RIGHT)
        swap(&array[pos], &array[pos - 1]);

    for (int i = 0; i < n; i++)
    {
        if (array[i] > mobile)
            {
            if (dir[array[i] - 1] == LEFT_TO_RIGHT)
                dir[array[i] - 1] = RIGHT_TO_LEFT;
            else if (dir[array[i] - 1] == RIGHT_TO_LEFT)
                dir[array[i] - 1] = LEFT_TO_RIGHT;
        }
    }
    for (int i = 0; i < n; i++)
        printf(" %d ",array[i]);
    printf("\n");
}
int factorial(int n)
{
    int result = 1;
    for (int i = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

```
void printPermutation(int n)
{
   int array[n],dir[n];
   printf("%d Permutations are:\n",factorial(n));
   for (int i = 0; i < n; i++)
   {
      array[i] = i + 1;
      printf(" %d ", array[i]);
   }
   printf("\n");

   for (int i = 0; i < n; i++)
   {
      dir[i] = RIGHT_TO_LEFT;//initialize right to left for all
   }
   for (int i = 1; i < factorial(n); i++)
   {
      printOnePermutation(array, dir, n,i);
   }
}

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

OUTPUT:


C:\Users\HP\Downloads\Jons
Enter the value of n: 4
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 0 (0x0) execution time : 8.381 s
Press any key to continue.


"C:\Users\HP\Desktop\BMSCI
Enter the value of n: 3
6 Permutations are:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
Process returned 0 (0x0) execution time : 1.278 s
Press any key to continue.

PROGRAM 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.

CODE:

```
#include<stdio.h>
void merge_sort(int low,int high,int n,int array[n])
{
   int mid;
   if(low<high)
   {
      mid=(low+high)/2;
      merge_sort(low,mid,n,array);
      merge_sort(mid+1,high,n,array);
      merge(low,mid,high,n,array);
   }
}

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

```
            i++;
            k++;
        }
        else
        {
            c[k]=array[j];
            j++;
            k++;
        }
    }
    while(i<=mid)
    {
        c[k]=array[i];
        i++;
        k++;
    }
    while(j<=high)
    {
        c[k]=array[j];
        j++;
        k++;
    }
    for (i = low; i <= high; i++)
    {
        array[i]=c[i];
    }
}

void main()
```

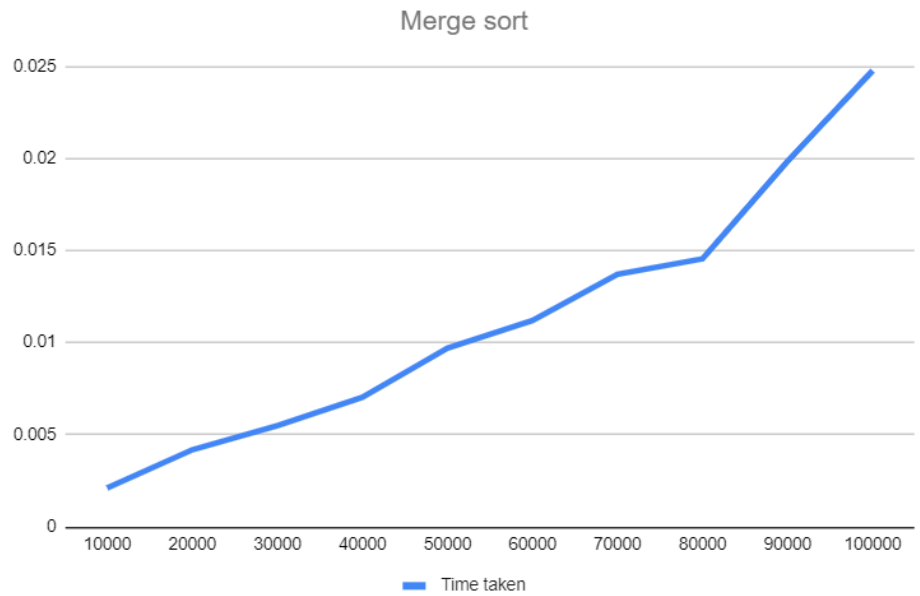
```
{
   int i;
   int array[20],n;
   printf("Enter the number of elements\n");
   scanf("%d",&n);
   printf("Enter the elements of the array\n");
   for (i = 0; i < n; i++)
   {
      scanf("%d", &array[i]);
   }
   merge_sort(0,n-1,n,array);
   printf("Sorted array is ");
   for (i = 0; i < n; i++)
   {
     // array[i]=c[i];
      printf("%d ", array[i]);
   }
}
```

OUTPUT:

```
F:\ADA\lab\mergesort.exe
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.
```

```
F:\ADA\lab\mergesort.exe
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:



PROGRAM 5

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

CODE:

```
#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;
```

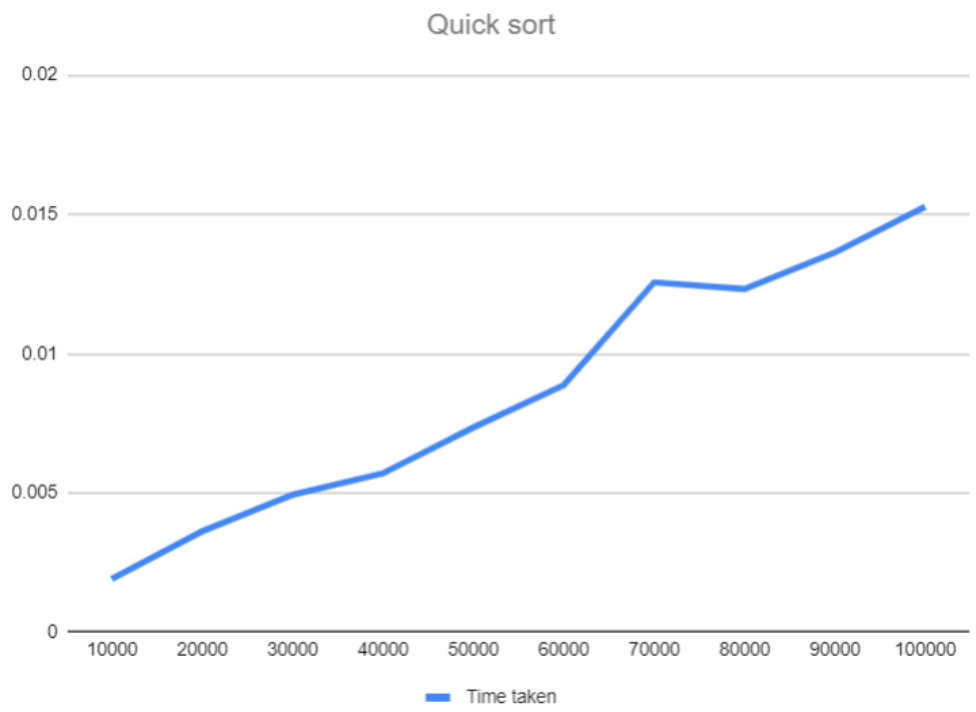
}

OUTPUT:

```
C:\Users\HP\Downloads\quicl
Enter the number of elements in array
9
Enter the elements of array
54 26 93 17 77 31 44 55 20
Sorted array: 17 20 26 31 44 54 55 77 93
Process returned 9 (0x9)   execution time : 97.035 s
Press any key to continue.
```

```
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.
```

GRAPH:



PROGRAM 6

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

CODE:

```
#include<stdio.h>

void heapsort(int n, int arr[])
{
   for (int i = n / 2 - 1; i >= 0; i--)
      heapify(n,arr,i);
   for (int i = n - 1; i > 0; i--)
   {
      swap(&arr[0], &arr[i]);
      heapify(i, arr, 0);
   }
}

void heapify(int n, int arr[], 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;
```

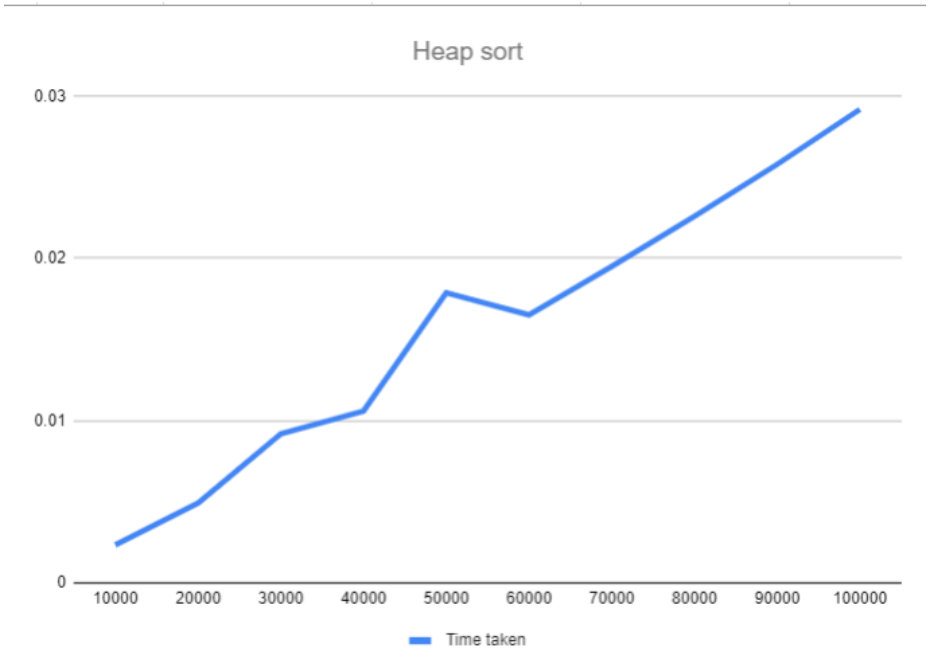
```
    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);
        heapify(n, arr, largest);
    }
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void main()
{
    int n;
    printf("HEAP SORT \n ");
    printf("\nEnter the number of elements to be sorted: ");
    scanf("%d", &n);
    int arr[n];
    printf("\nEnter the elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    heapsort(n, arr);
    printf("\nSorted array in ascending order:\n ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
}
```

OUTPUT:

```
"C:\Users\HP\Desktop\BMSCI
HEAP SORT

Enter the number of elements to be sorted: 5

Enter the elements: 59 32 12 67 34

Sorted array in ascending order:
 12 32 34 59 67
Process returned 5 (0x5)   execution time : 9.608 s
Press any key to continue.
```

```
"C:\Users\Admin\Desktop\1BM21CS048\Sem4\ADA Lab\heap_ggg.exe"
HEAP SORT

Enter the number of elements to be sorted: 8

Enter the elements: 89 81 74 22 14 9 54 11

Sorted array in ascending order: 9 11 14 22 54 74 81 89
Process returned 8 (0x8)   execution time : 15.018 s
Press any key to continue.
```

GRAPH :



PROGRAM 7

Implement 0/1 Knapsack problem using dynamic programming.

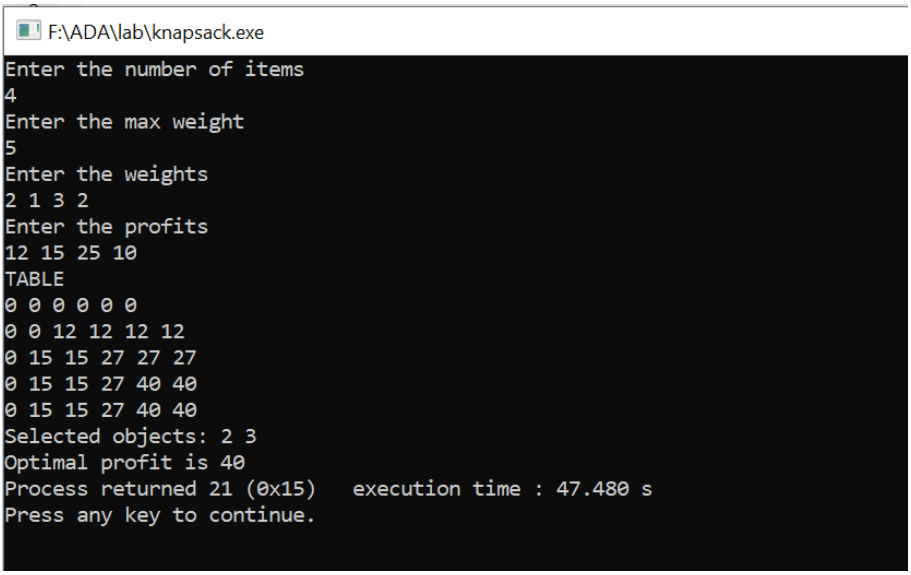
CODE:

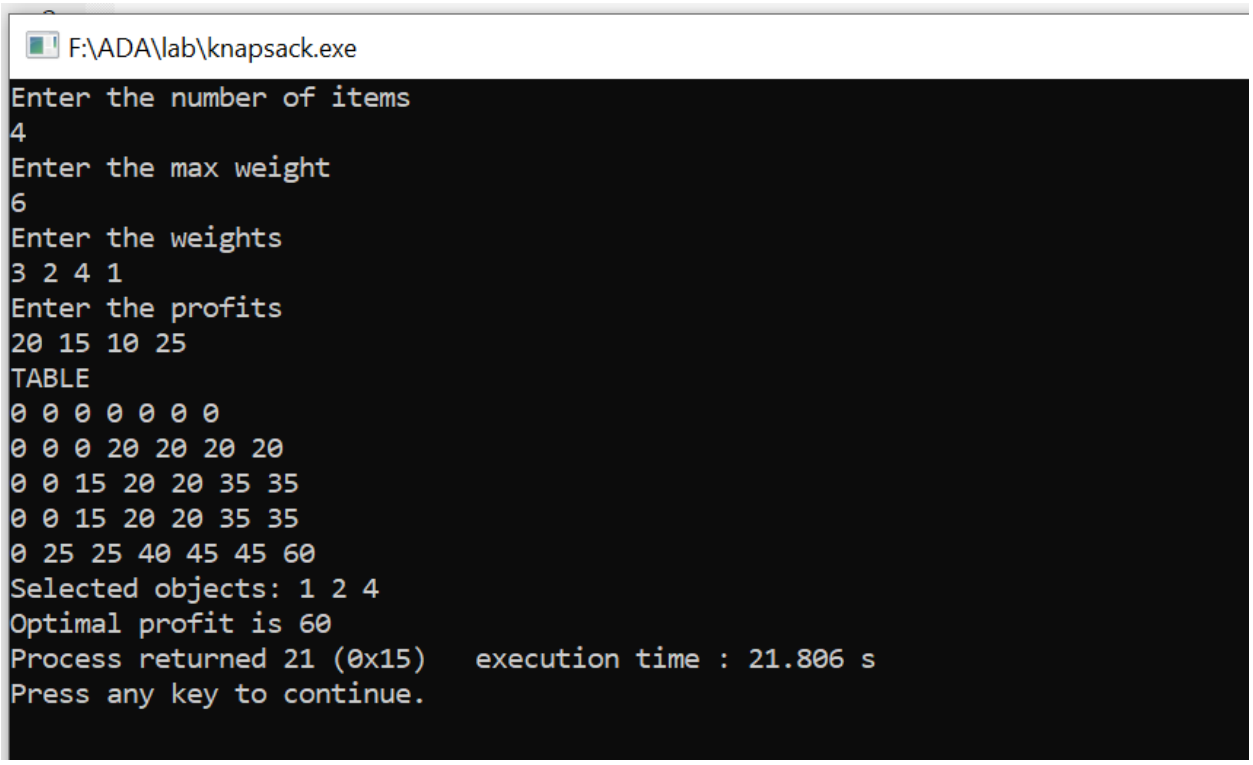
```
#include<stdio.h>
#include <stdio.h>
#include <stdbool.h>

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:




F:\ADA\lab\knapsack.exe
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.

PROGRAM 8

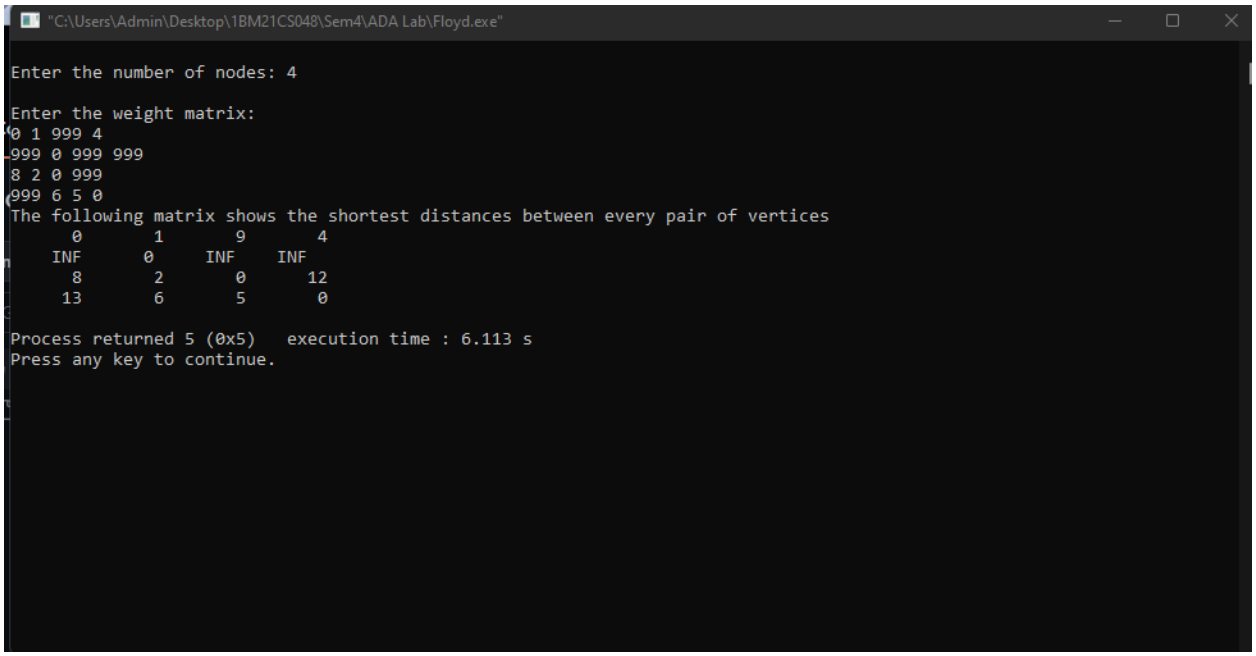
Implement All Pair Shortest paths problem using Floyd's algorithm.

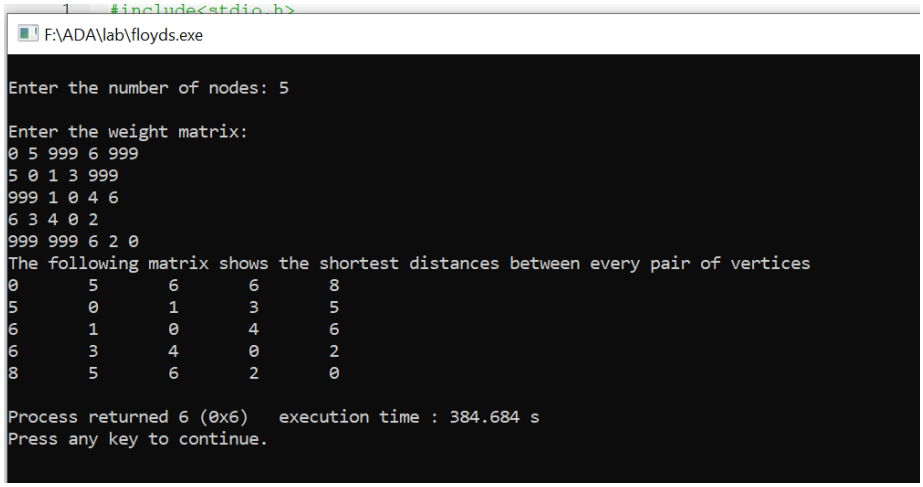
CODE:

```
#include<stdio.h>
#define MAX 10
#define INF 999

void printSolution(int n,int dist[MAX][MAX])
{
   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++)
      {
         if (dist[i][j] == INF)
            printf("%7s", "INF");
         else
            printf("%7d ", dist[i][j]);
      }
      printf("\n");
   }
  }

void floyd(int n,int dist[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 (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printSolution(n,dist);
}

void main()
{
    int i,n,W,j;
    int w[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]);
    }
    floyd(n,w);
}
```

OUTPUT:


"C:\Users\Admin\Desktop\1BM21CS048\Sem4\ADA Lab\Floyd.exe"
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
INF 0 INF INF
8 2 0 12
13 6 5 0
Process returned 5 (0x5) execution time : 6.113 s
Press any key to continue.


F:\ADA\lab\floyds.exe
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.

PROGRAM 9

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

KRUSKALS ALGORITHM

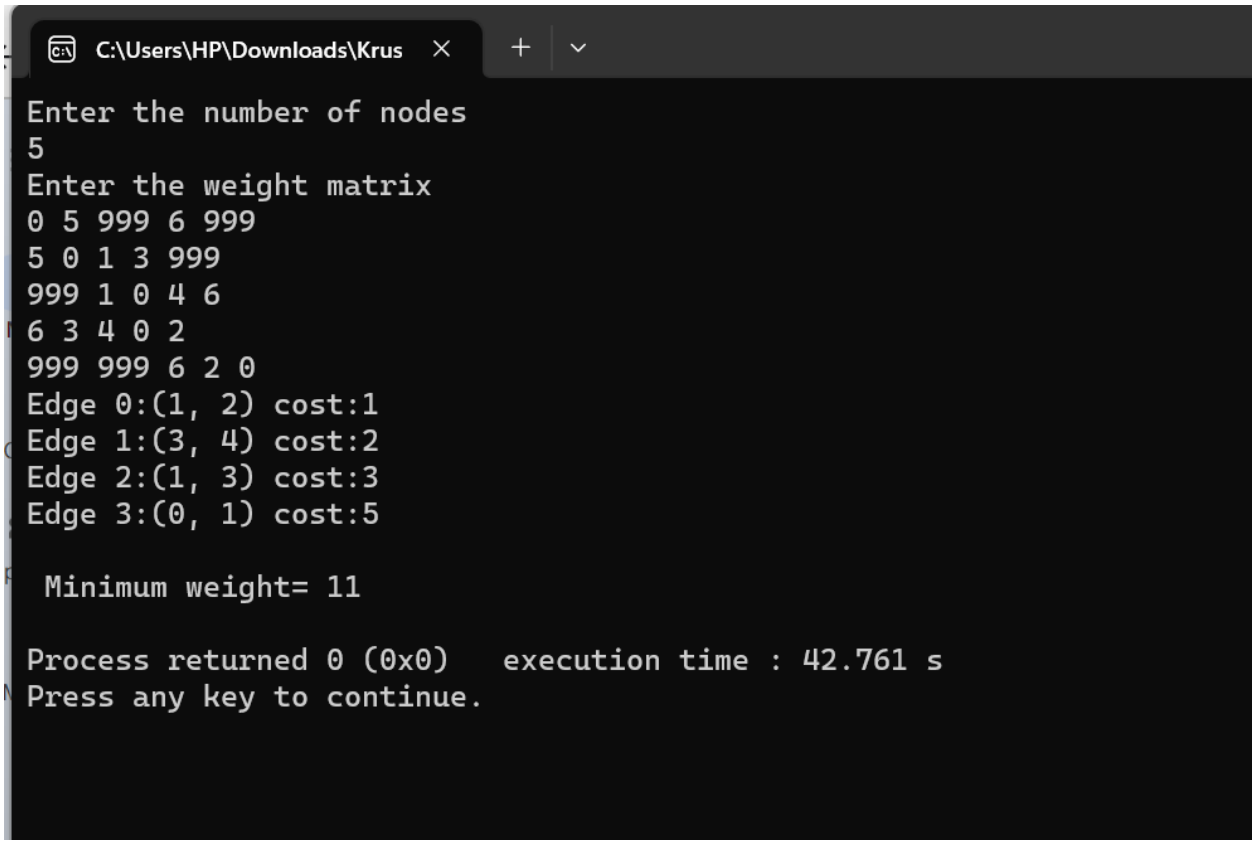
CODE:

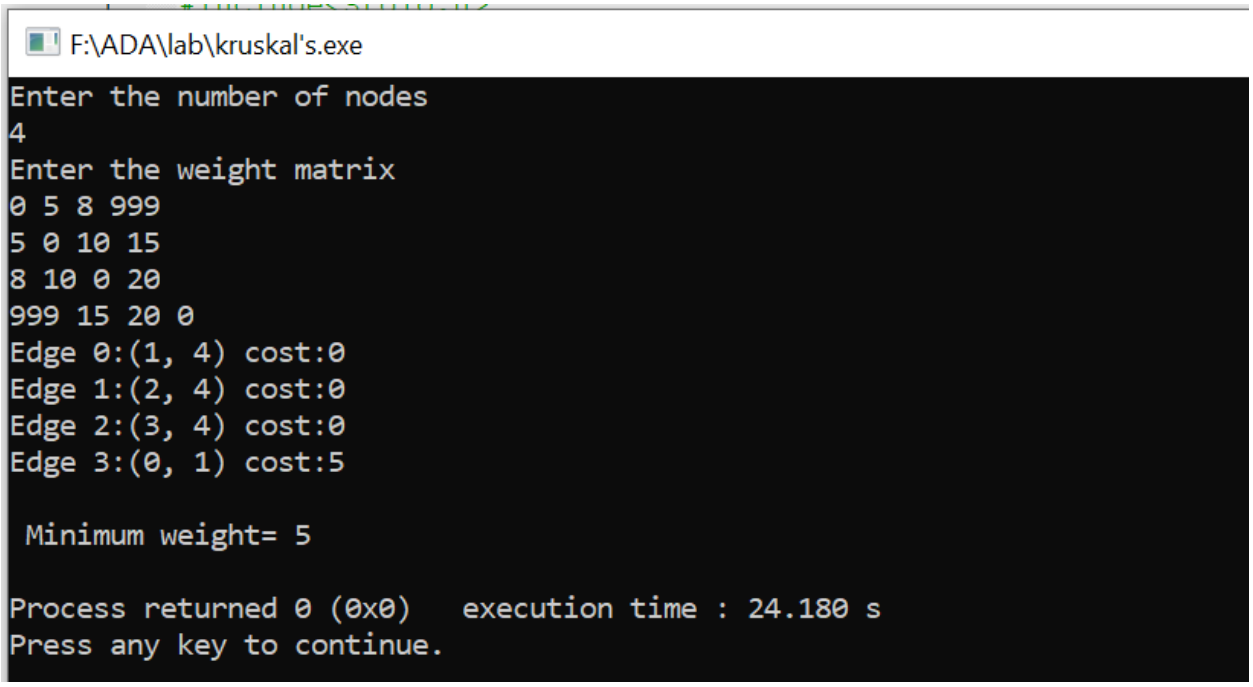
```
#include<stdio.h>
#include <stdbool.h>
#define INT_MAX 999
#define V 10
int n;
int parent[V];
int find(int i)
{
   while (parent[i] != i)
      i = parent[i];
   return i;
}

void merge1(int i, int j)
{
   int a = find(i);
   int b = find(j);
   parent[a] = b;
}

void kruskalMST(int cost[][V])
{
   int mincost = 0; // Cost of min MST.
```

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

    int edge_count = 0;
    while (edge_count < n - 1)
    {
        int min = INT_MAX, a = -1, b = -1;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (find(i) != find(j) && cost[i][j] < min)
                {
                    min = cost[i][j];
                    a = i;
                    b = j;
                }
            }
        }
        merge1(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:





PRIMS ALGORITHM

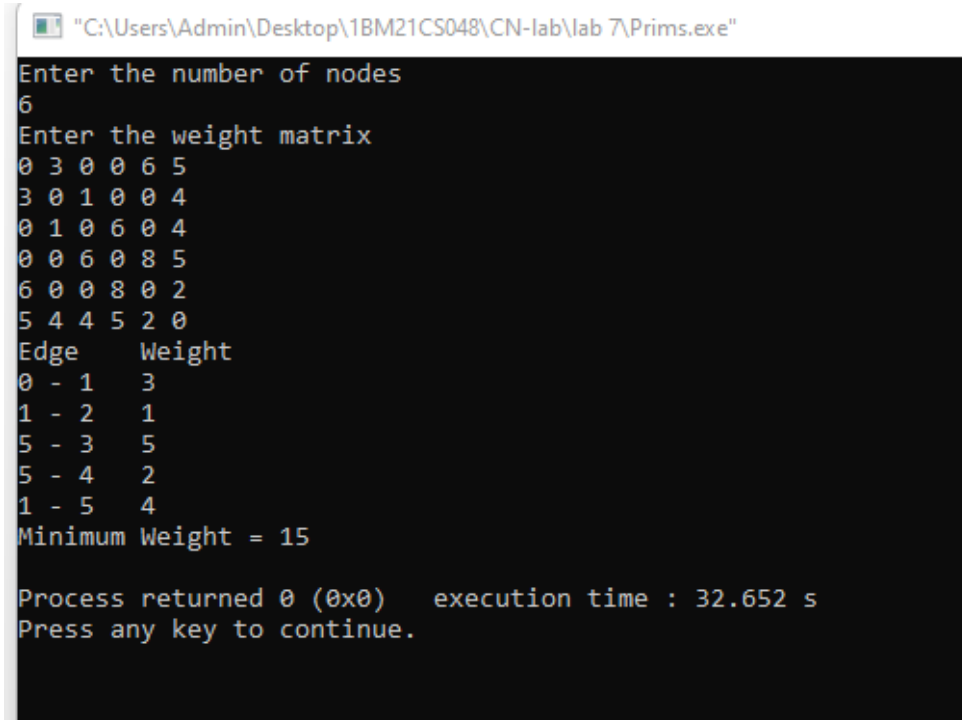
CODE:

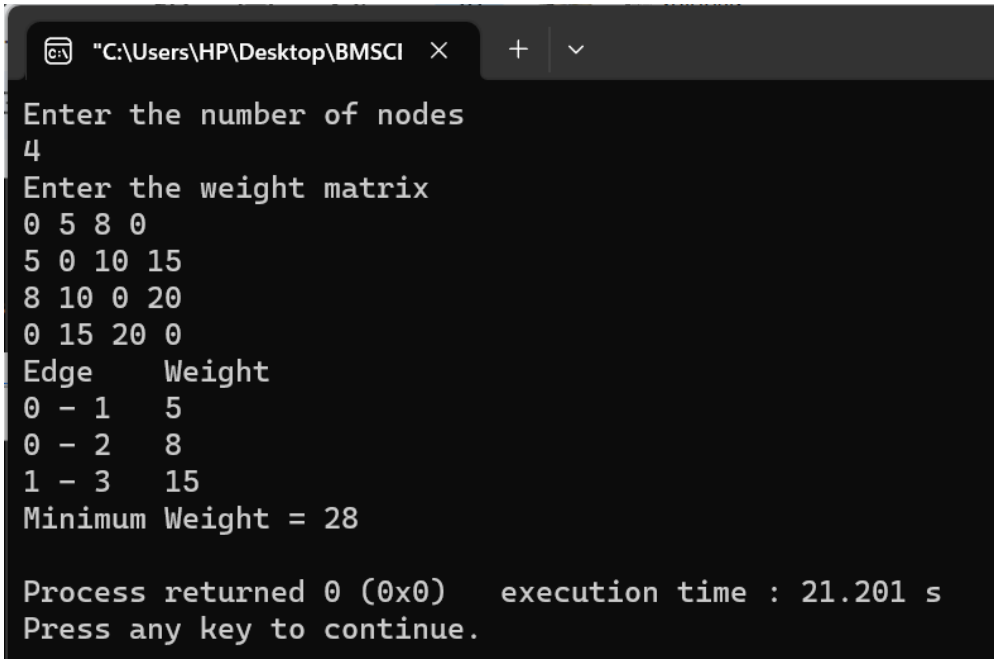
```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 10

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 sum=0;
    printf("Edge \tWeight\n");
    for (int i = 1; i < n; i++)
    {
        sum+=graph[i][parent[i]];
         printf("%d - %d \t%d \n", parent[i], i,
    graph[i][parent[i]]);
    }
    printf("Minimum Weight = %d \n",sum);
}

void MSTPrims(int graph[V][V],int n)
{
    int parent[V];
    int key[V];
    bool mstSet[V];//stores node included in mst

    for (int i = 0; i < n; i++)
        key[i] = INT_MAX, mstSet[i] = false;//all key set to infinte and all nodes not added

    key[0] = 0;//Make key 0 so that this vertex is picked as first
    parent[0] = -1;
```

```
    for (int count = 0; count < n - 1; count++)
    {
        int u = minKey(key, mstSet, n);//stores min index
        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);
}

void main()
{
    int w[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",&w[i][j]);
    }
    MSTPrims(w,n);
}
```

OUTPUT:


"C:\Users\Admin\Desktop\1BM21CS048\CN-lab\lab 7\Prims.exe"
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
Minimum Weight = 15
Process returned 0 (0x0) execution time : 32.652 s
Press any key to continue.


"C:\Users\HP\Desktop\BMSCI
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
Minimum Weight = 28
Process returned 0 (0x0) execution time : 21.201 s
Press any key to continue.

PROGRAM 10

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

CODE:

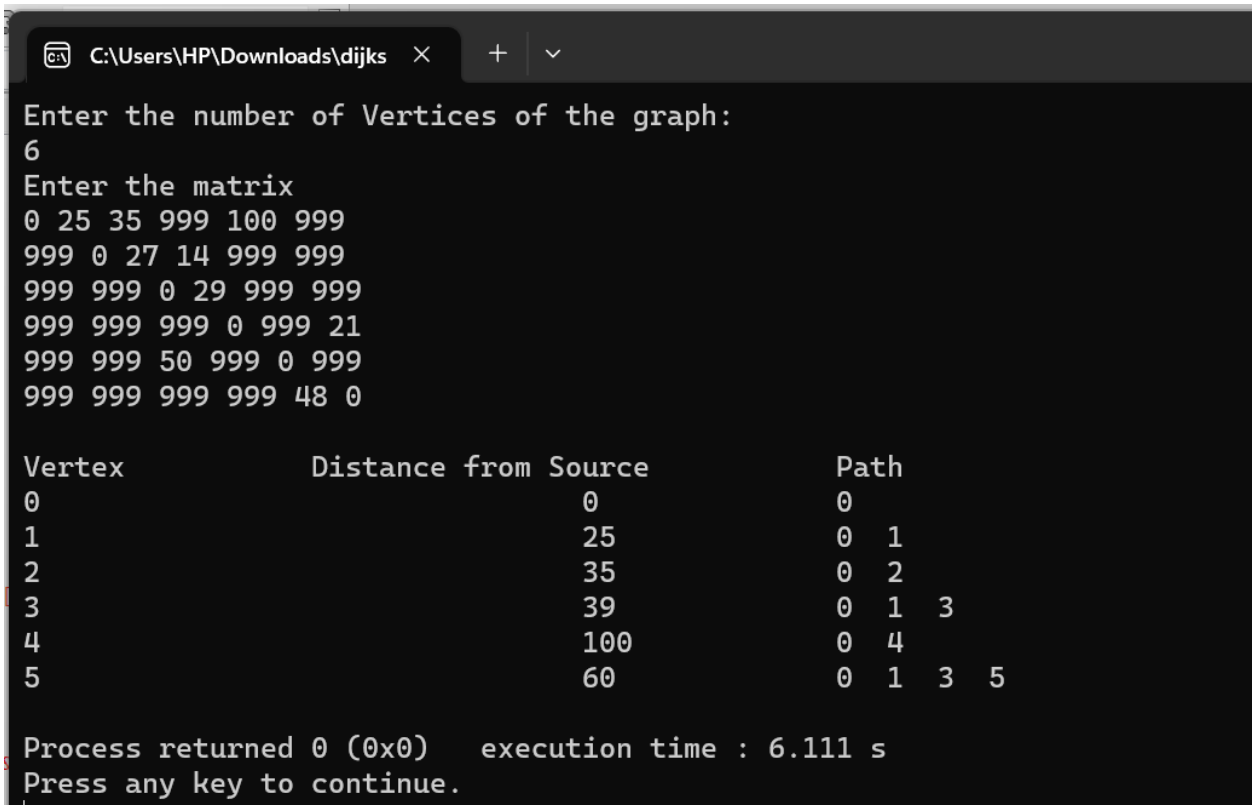
```
#include <stdbool.h>
#include <stdio.h>
#define MAX 999
int V;
int parents[50];
int noParent= -1;

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

	for (int v = 0; v < V; v++)
	{
	   if (picked[v] == false && totalWeight[v] <= min)
			{
			   min = totalWeight[v];
	   min_index = v;
	}
	}

	return min_index;
}
```

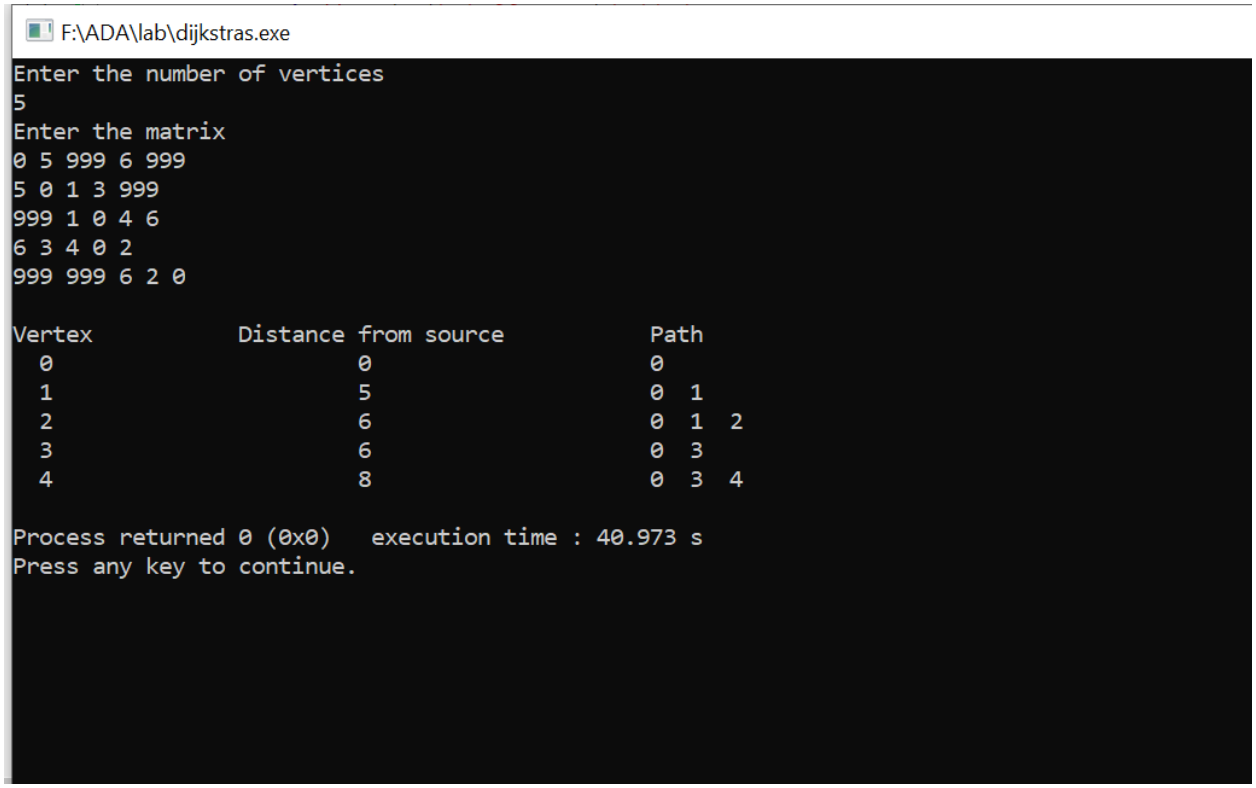
```
void printPath(int currentVertex,int parents[V])
{
        if (currentVertex == noParent)
   {
                return;
        }
        printPath(parents[currentVertex], parents);
        printf("%d  ",currentVertex);
}
void printSolution(int totalWeight[])
{
        printf("\nVertex \t\t Distance from Source\t\tPath\n");
        for (int i = 0; i < V; i++)
   {
      printf("%d \t\t\t\t %d \t\t", i, totalWeight[i]);
                printPath(i,parents);//for each node we print the shortest from from root node
                printf("\n");
        }
}

void dijkstra(int graph[V][V], int src)
{
        int totalWeight[V];//total weight from source to each node

        bool picked[V];//node picked or not
        for (int i = 0; i < V; i++)
   {
      totalWeight[i] = MAX;
      picked[i] = false;
```

```
    }

        totalWeight[src] = 0;
    parents[0]=noParent;//no parent yet
        for (int count = 0; count < V - 1; count++)
    {
                int u = minDistance(totalWeight, picked);//find node with min distance

                picked[u] = true;//pick that node

                for (int v = 0; v < V; v++)
       {
        if (!picked[v] && graph[u][v]
                                && totalWeight[u] != MAX
                                && totalWeight[u] + graph[u][v] < totalWeight[v])//find min one
                                {
              totalWeight[v] = totalWeight[u] + graph[u][v];
              parents[v]=u;//u is parent of v
                                }
                }
        }

        printSolution(totalWeight);
}

int main()
{
    printf("Enter the number of Vertices of the graph:\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:




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.

PROGRAM 11

Implement “N-Queens Problem” using Backtracking

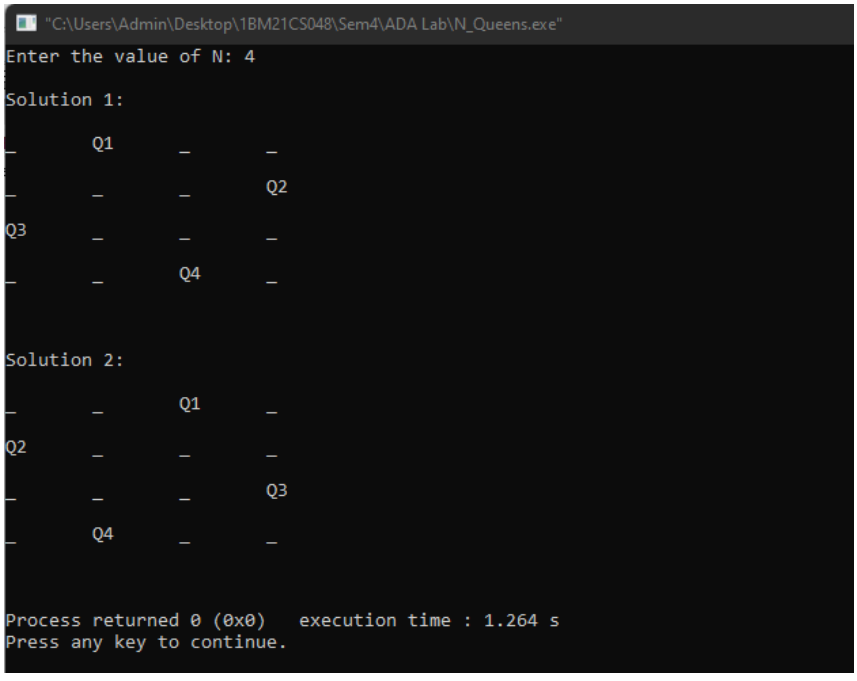
CODE:

```
#include <stdio.h>
#define MAX 10
int x[MAX],c=0;
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)
{
   c++;
   printf("\nSolution %d: \n\n",c);
   for (int i = 1; i <= n; i++)//i-> queen number
      {
      for (int j = 1; j <= n; j++)//j-> position
      {
         if (j == x[i])
```

```
            printf("Q%d\t",i);
        else
            printf("_\t");
    }
    printf("\n\n");
  }
  printf("\n");
}

void nqueens(int n)
{
  int k = 1; //Select the first queen
  x[k] = 0;//But not placed on chess board

  while (k != 0) //A queen exists ?
  {
    x[k] = x[k] + 1;//Place the kth queen in next column

    while (x[k] <= n && !place(k))
    {
      x[k] = x[k] + 1;
    }
    if (x[k] <= n)
    {
      if (k == n)  //If all Queens are placed
      {
        write(n);
      }
      else
```

```
        {
            k = k + 1;  //Select next Queen
            x[k] = 0; //But do not place
        }
    }
    else
        k = k - 1; //Backtrack and select previous queen
  }
}
void main()
{
  int n;
  printf("Enter the value of N: ");
  scanf("%d", &n);
  nqueens(n);
}
```

OUTPUT:



92 solutions for n=8