NETAJI SUBHASH ENGINEERING COLLEGE

NAME: SOUVIK PAL

STREAM: CSE [4th SEMESTER]

SECTION: A [ROLL 60]

UNIVERSITY ROLL: 10900121064

COURSE NAME: DESIGN AND ANALYSIS OF ALGO. (DAA) LAB

COURSE CODE: PCC-CS494

Netaji Subhash Engineering College
Department of Computer Science & Engineering

INDEX

Course Name: **Student Name:**

Course Code: **University Roll No. :**

Sl. No.	Exp. No.	Problem Statement	D.O.E	D.O.S	Marks			Signature of the Faculty with Date	Remarks (if any)
					5	15	20		
1.	1.	Write a C program to find the position of an element from the set of integers using Binary Search Algorithm.							
2.	2.	Write a C program to find Maximum and Minimum element from an array of given integers using Divide and Conquer approach.							
3.	3.	Write a C program to sort given integer numbers in descending order using Merge Sort Algorithm.							
4.	4.	Write a C program to sort given integer numbers in ascending order using Quick Sort Algorithm.							
5.	5.	Write a C program to find all pair of shortest path for the following graph using Floyd- Warshall Algorithm.							
6.	6.	Write a C program to implement Single Source Shortest Path for the following graph using Dijkstra Algorithm. Consider vertex 1 as the source.							

7.	7.	Given the weight vector (2, 3, 5, 7, 1, 4, 1) and the profit vector (10, 5, 15, 7, 6, 18, 3) and a Knapsack of capacity 15, write a C program to find the optimal solution for the fractional knapsack problem of seven objects.							
8.	8.	Write a C program to implement Minimum Cost Spanning Tree for the following graph by Prim's Algorithm.							
9.	9.	Write a C program to implement Minimum Cost Spanning Tree for the following graph by Kruskal's Algorithm.							
10.	10.	Write a C program to find the optimal sequence of jobs when number of jobs n=5,profit (p1,p2,p3,p4,p5)=(20,15,10, 5,1) and deadline (d1,d2,d3,d4,d5)=(2,2,1,3,3) .							
11.	11.	Write a C program to implement Single Source Shortest Path for the following graph using Bellman-Ford Algorithm. Consider vertex 1 as the source.							
Average Marks									

Experiment No. 1:

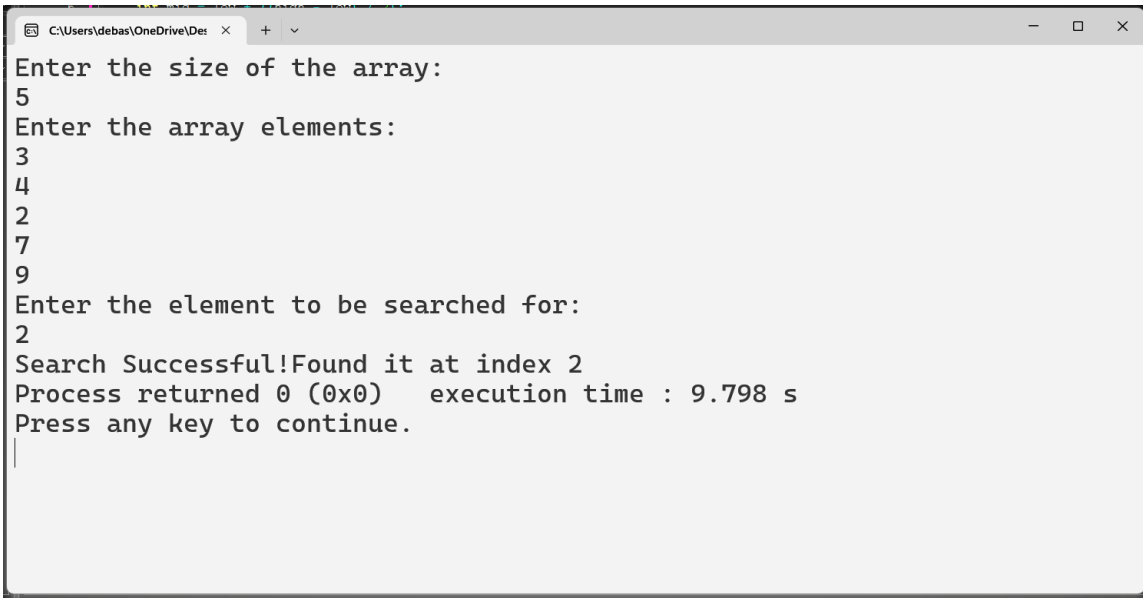
Problem Statement:

Write a C program to find the position of an element from the set of integers using Binary Search Algorithm.

Program Code:

```
#include<stdio.h>

int binary_search(int A[], int key, int len) {
  int low = 0;
  int high = len -1;
  while (low <= high) {
    int mid = low + ((high - low) / 2);
    if (A[mid] == key) {
      return mid;
    }
    if (key < A[mid]) {
      high = mid - 1;
    }
    else {
      low = mid + 1;
    }
  }
  return -1;
}

int main() {
  int a[10]={1,3,5,7,9,11,13,15,17,21};
  int key = 3;
  int position = binary_search(a, key, 10);
  if (position == -1){
    printf("Not found");
    return 0;
  }
  printf("Found it at %d", position);
  return 0;
}
```

Output:


C:\Users\debas\OneDrive\Des
Enter the size of the array:
5
Enter the array elements:
3
4
2
7
9
Enter the element to be searched for:
2
Search Successful!Found it at index 2
Process returned 0 (0x0)   execution time : 9.798 s
Press any key to continue.

Experiment No. 2:

Problem Statement:

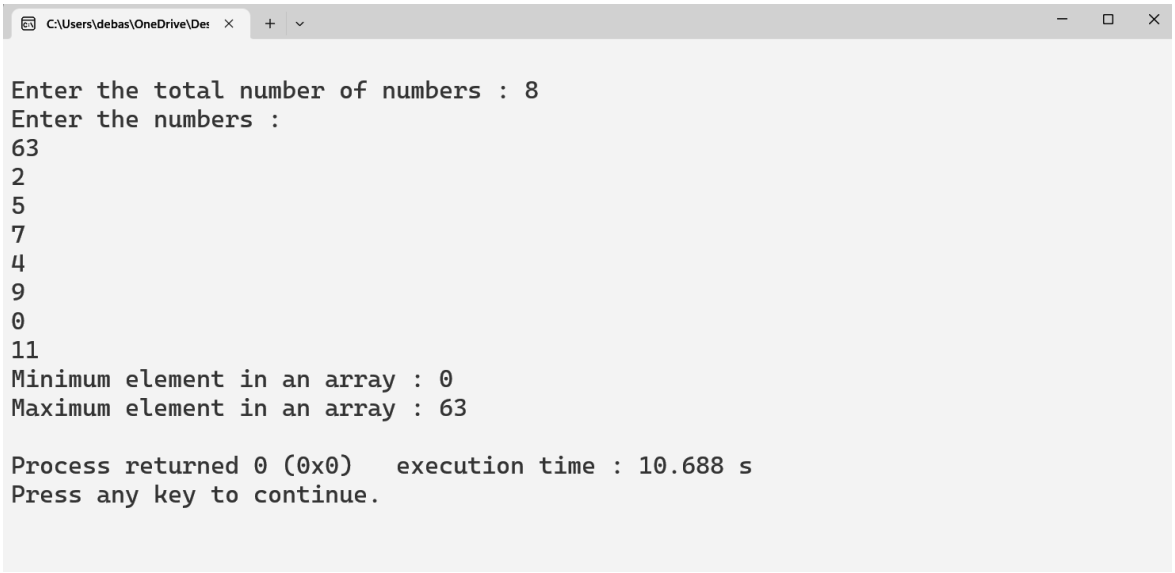
Write a C program to find Maximum and Minimum element from an array of given integers using Divide and Conquer approach.

Program Code:

```
#include<stdio.h>
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
 int max1, min1, mid;
 if(i==j)   {
  max = min = a[i];
 }
 else  {
  if(i == j-1)
  {
   if(a[i] <a[j])
   {
    max = a[j];
    min = a[i];
   }
   else
   {
    max = a[i];
    min = a[j];
   }
  }
  else
  {
   mid = (i+j)/2;
   maxmin(i, mid);
   max1 = max; min1 = min;
   maxmin(mid+1, j);
```

```
  if(max <max1)
   max = max1;
  if(min > min1)
   min = min1;    }   }    }
int main ()
{
 int i, num;
 printf ("\nEnter the total
number of numbers : ");
 scanf ("%d",&num);
 printf ("Enter the numbers :
\n");
 for (i=1;i<=num;i++)
  scanf ("%d",&a[i]);

 max = a[0];
 min = a[0];
 maxmin(1, num);
 printf ("Minimum element in an
array : %d\n", min);
 printf ("Maximum element in an
array : %d\n", max);
 return 0;
}
```

Output:



Experiment No. 3:

Problem Statement:

Write a C program to sort given integer numbers in descending order using Merge Sort Algorithm.

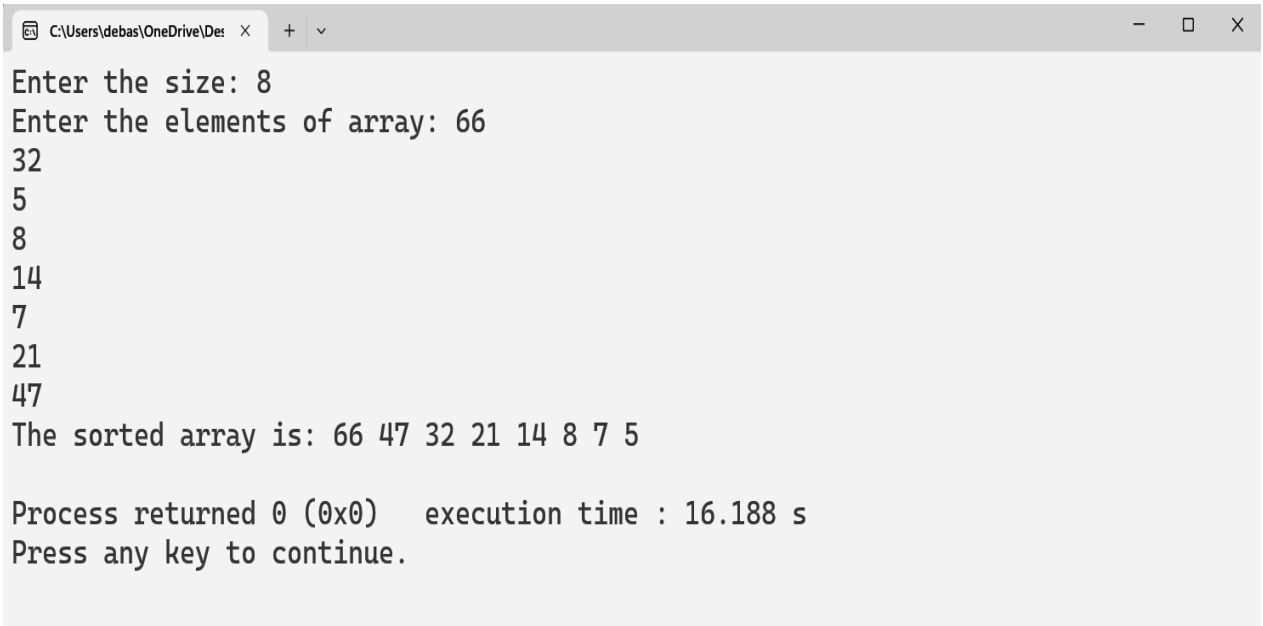
Program Code:

```
#include <stdio.h>

#include <stdlib.h>

void Merge(int arr[], int left, int mid, int right)

{

   int i, j, k;

   int size1 = mid - left + 1;

   int size2 = right - mid;

   int Left[size1], Right[size2];

   for (i = 0; i < size1; i++)

      Left[i] = arr[left + i];


   for (j = 0; j < size2; j++)

      Right[j] = arr[mid + 1 + j];

   i = 0;

   j = 0;

   k = left;

   while (i < size1 && j < size2)

   {

      if (Left[i] <= Right[j])

      {

         arr[k] = Left[i];

         i++;

      }

      else

      {

         arr[k] = Right[j];

         j++;

      }

      k++;

   }
```

```
    while (i < size1)
    {
        arr[k] = Left[i];
        i++;
        k++;
    }
    while (j < size2)
    {
        arr[k] = Right[j];
        j++;
        k++;
    }
}
void Merge_Sort(int arr[], int
left, int right)
{
    if (left < right)
    {
  int mid = left + (right - left) / 2;
        Merge_Sort(arr, left, mid);
        Merge_Sort(arr, mid + 1,
right);

        Merge(arr, left, mid, right);
    }
}
int main()
{
    int size;
    printf("Enter the size: ");
    scanf("%d", &size);

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

```
    Merge_Sort(arr, 0, size - 1);
    printf("The sorted array is: ");
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
}
   printf("\n");
    return 0;
}
```

Output:



Experiment No. 4:

Problem Statement:

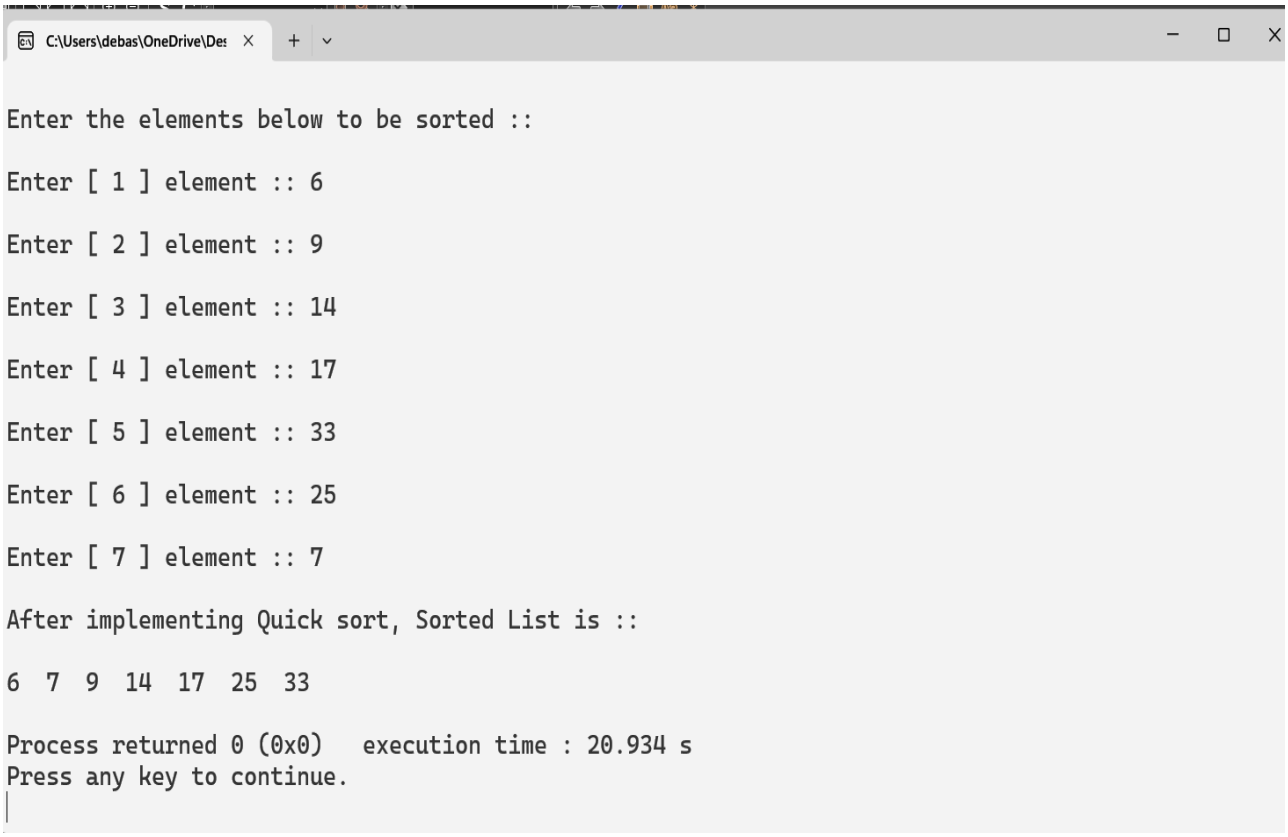
Write a C program to sort given integer numbers in ascending order using Quick Sort Algorithm.

Program Code:

```
#include <stdio.h>

void quicksort (int [], int, int);

int main()
{
   int list[50];
   int size, i;

   printf("How many elements u want to Sort :: ");
   scanf("%d", &size);

   printf("\nEnter the elements below to be sorted :: \n");
   for (i = 0; i < size; i++)
   {
      printf("\nEnter [ %d ] element :: ",i+1);
      scanf("%d", &list[i]);
   }

   quicksort(list, 0, size - 1);

   printf("\nAfter implementing Quick sort, Sorted List is :: \n\n");

   for (i = 0; i < size; i++)
   {
      printf("%d  ", list[i]);
```

```
    }

    printf("\n");
    return 0;
}
void quicksort(int list[], int low,
int high)
{
    int pivot, i, j, temp;
    if (low < high)
    {
        pivot = low;
        i = low;
        j = high;
        while (i < j)
        {
            while (list[i] <= list[pivot]
&& i <= high)
            {
                i++;
            }
            while (list[j] > list[pivot]
&& j >= low)
            {
                j--;
            }
            if (i < j)
            {
                temp = list[i];
                list[i] = list[j];
                list[j] = temp;
            }
        }
        temp = list[j];
        list[j] = list[pivot];
        list[pivot] = temp;
        quicksort(list, low, j - 1);
        quicksort(list, j + 1, high);
    }
}
```

Output:


C:\Users\debas\OneDrive\Des
Enter the elements below to be sorted ::
Enter [1] element :: 6
Enter [2] element :: 9
Enter [3] element :: 14
Enter [4] element :: 17
Enter [5] element :: 33
Enter [6] element :: 25
Enter [7] element :: 7
After implementing Quick sort, Sorted List is ::
6 7 9 14 17 25 33
Process returned 0 (0x0) execution time : 20.934 s
Press any key to continue.

Experiment No. 5:

Problem Statement:

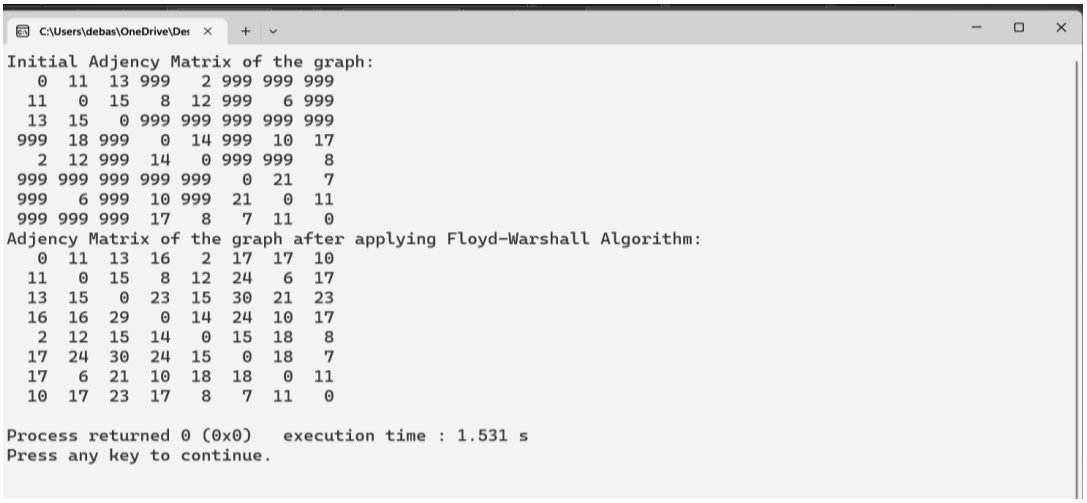
Write a C program to find all pair of shortest path for a graph using Floyd- Warshall Algorithm.

Program Code:

```
#include <stdio.h>

#define nV 8

#define INF 999


void printMatrix(int matrix[][nV]);


void floydWarshall(int graph[][nV]) {
  int matrix[nV][nV], i, j, k;

  for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
      matrix[i][j] = graph[i][j];

  for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
      for (j = 0; j < nV; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
  printMatrix(matrix);
}

void printMatrix(int matrix[][nV])
{
  for (int i = 0; i < nV; i++) {
    for (int j = 0; j < nV; j++) {
```

```
    if (matrix[i][j] == INF)
      printf("%4s", "INF");
    else
      printf("%4d", matrix[i][j]);
   }
   printf("\n");
 }
}

int main() {
 int graph[nV][nV] = {{0,   11,
13,  INF, 2,  INF,  INF, INF},
                {11,  0,  15,  8,
12,  INF,  6,  INF},
                {13,  15,  0,   INF,
INF, INF,  INF, INF},
                {INF,  18,  INF,  0,
14,  INF,  10,  17},
                {2,   12,  INF,  14,  0,
INF,  INF,  8},
                {INF,  INF, INF,  INF,
INF, 0,   21,   7},
                {INF,  6,  INF,  10,
INF, 21,  0,   11},
                {INF,  INF, INF,  17,
8,  7,   11,   0}
        };
 floydWarshall(graph);
}
```

Output:



Experiment No. 6:

Problem Statement:

Write a C program to implement Single Source Shortest Path for the following graph using Dijkstra Algorithm. Consider vertex 1 as the source.

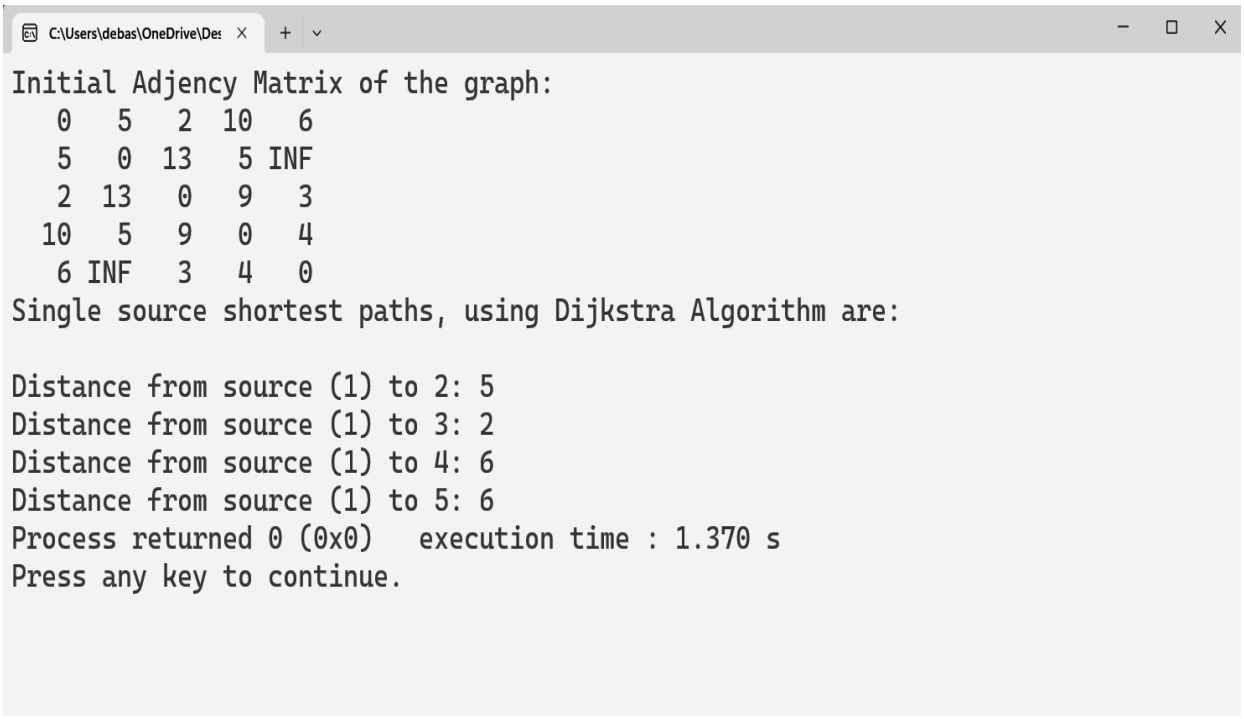
Program Code:

```
#include <stdio.h>

#define INF 9999

#define MAX 10


void Dijkstra(int
Graph[MAX][MAX], int n, int
start);


void Dijkstra(int
Graph[MAX][MAX], int n, int
start) {

  int cost[MAX][MAX],
distance[MAX], pred[MAX];

  int visited[MAX], count,
mindistance, nextnode, i, j;


// Creating cost matrix

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

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

    if (Graph[i][j] == 0)

      cost[i][j] = INF;

    else

      cost[i][j] = Graph[i][j];


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

  distance[i] = cost[start][i];

  pred[i] = start;

  visited[i] = 0;

}


distance[start] = 0;
```

```
 visited[start] = 1;
 count = 1;

 while (count < n - 1) {
  mindistance = INF;

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

  visited[nextnode] = 1;
  for (i = 0; i < n; i++)
   if (!visited[i])
    if (mindistance +
cost[nextnode][i] < distance[i]) {
     distance[i] = mindistance +
cost[nextnode][i];
     pred[i] = nextnode;
    }
  count++;
 }
 // Printing the distance
 printf("Single source shortest
paths, using Dijkstra Algorithm
are:\n");
 for (i = 0; i < n; i++)
  if (i != start) {
   printf("\nDistance from
source (1) to %d: %d", (i+1),
distance[i]);
  }
}
int main() {
 int graph[5][5] = {
            {0,  5,  2,  10,  6},
            {5,  0,  13,  5,  INF},
            {2,  13,  0,  9,  3},
            {10,  5,  9,  0,  4},
            {6,  INF,  3,  4,  0},
```

```
        };
    printf("Initial Adjency Matrix of the graph:\n");
     for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
       if (graph[i][j] == INF)
       printf("%4s", "INF");
      else
       printf("%4d", graph[i][j]);
     }
     printf("\n");
    }
    int u = 0;
    Dijkstra(graph, 5, u);

    return 0;
}
```

Output:



Experiment No. 7:

Problem Statement:

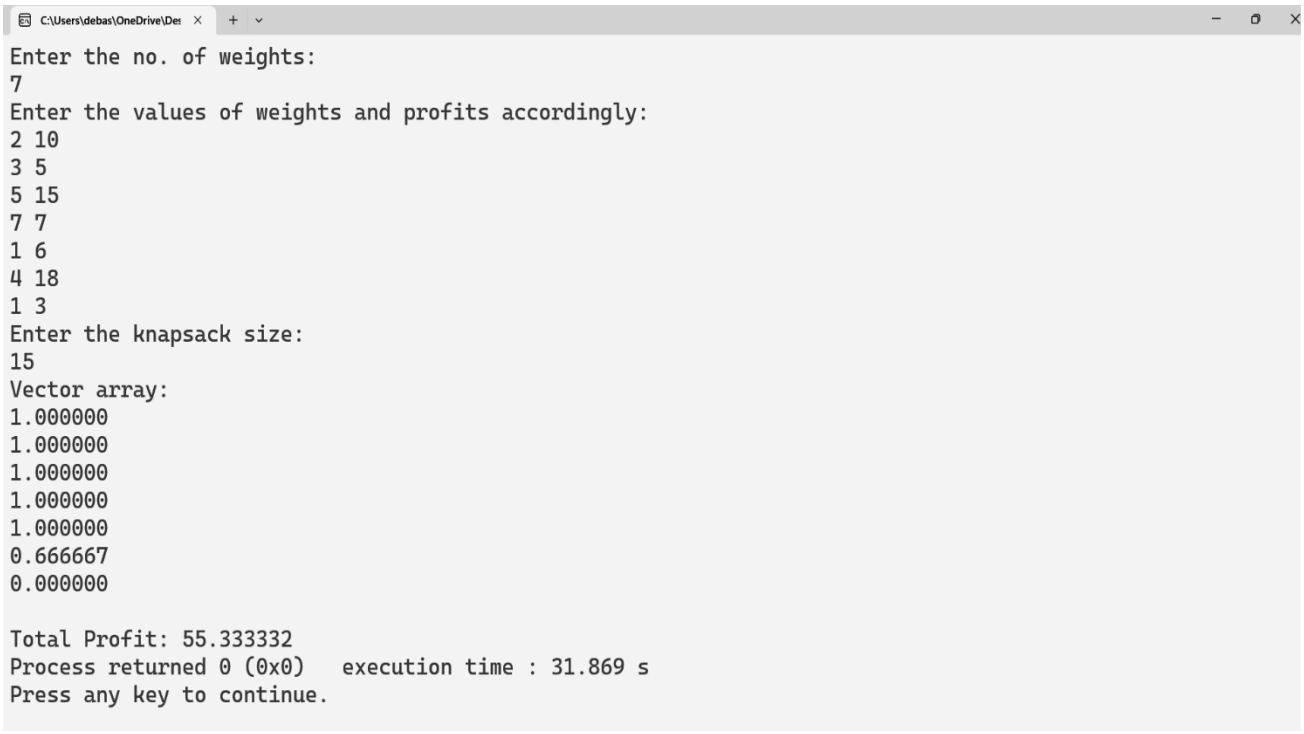
Given the weight vector (2, 3, 5, 7, 1, 4, 1) and the profit vector (10, 5, 15, 7, 6, 18, 3) and a Knapsack of capacity 15, write a C program to find the optimal solution for the fractional knapsack problem of seven objects.

Program Code:

```
#include <stdio.h>

#define MAX 100


int knapsack(int cp, int n, float wght[], float prft[])
{
   float vec[MAX], tp=0.0;
   int i;
   for(i=0;i<n;i++)
      vec[i]=0.0;
   for(i=0;i<n;i++)
   {
      if(wght[i]>cp)
         break;
      else
      {
         vec[i]=1.0;
         cp=cp-wght[i];
         tp=tp+prft[i];
      }
   }
if(i<n)
vec[i]=cp/wght[i];
tp=tp+(vec[i]*prft[i]);
printf("Vector array:\n");
for(i=0;i<n;i++)
   printf("%f\n", vec[i]);
printf("\nTotal Profit: %f", tp);
```

```
    return 0;
}
int main()
{
    float weight[MAX], profit[MAX], ratio[MAX];
    int n, capacity;
    printf("Enter the no. of weights:\n");
    scanf("%d", &n);
    printf("Enter the values of weights and profits accordingly:\n");
    for(int i=0;i<n;i++)
        scanf("%f  %f", &weight[i], &profit[i]);
    for(int i=0;i<n;i++)
        ratio[i]=profit[i]/weight[i];
    printf("Enter the knapsack size:\n");
    scanf("%d", &capacity);
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(ratio[i]<ratio[j]){
                float temp=ratio[i];
                ratio[i]=ratio[j];
                ratio[j]=temp;
                float temp1=weight[i];
                weight[i]=weight[j];
                weight[j]=temp1;
                float temp2=profit[i];
                profit[i]=profit[j];
                profit[j]=temp2;
            }
        }
    }
    knapsack(capacity, n, weight, profit);
    return 0;
}
```

Output:


C:\Users\debas\OneDrive\Des
Enter the no. of weights:
7
Enter the values of weights and profits accordingly:
2 10
3 5
5 15
7 7
1 6
4 18
1 3
Enter the knapsack size:
15
Vector array:
1.000000
1.000000
1.000000
1.000000
1.000000
0.666667
0.000000
Total Profit: 55.333332
Process returned 0 (0x0)   execution time : 31.869 s
Press any key to continue.

Experiment No. 8:

Problem Statement:

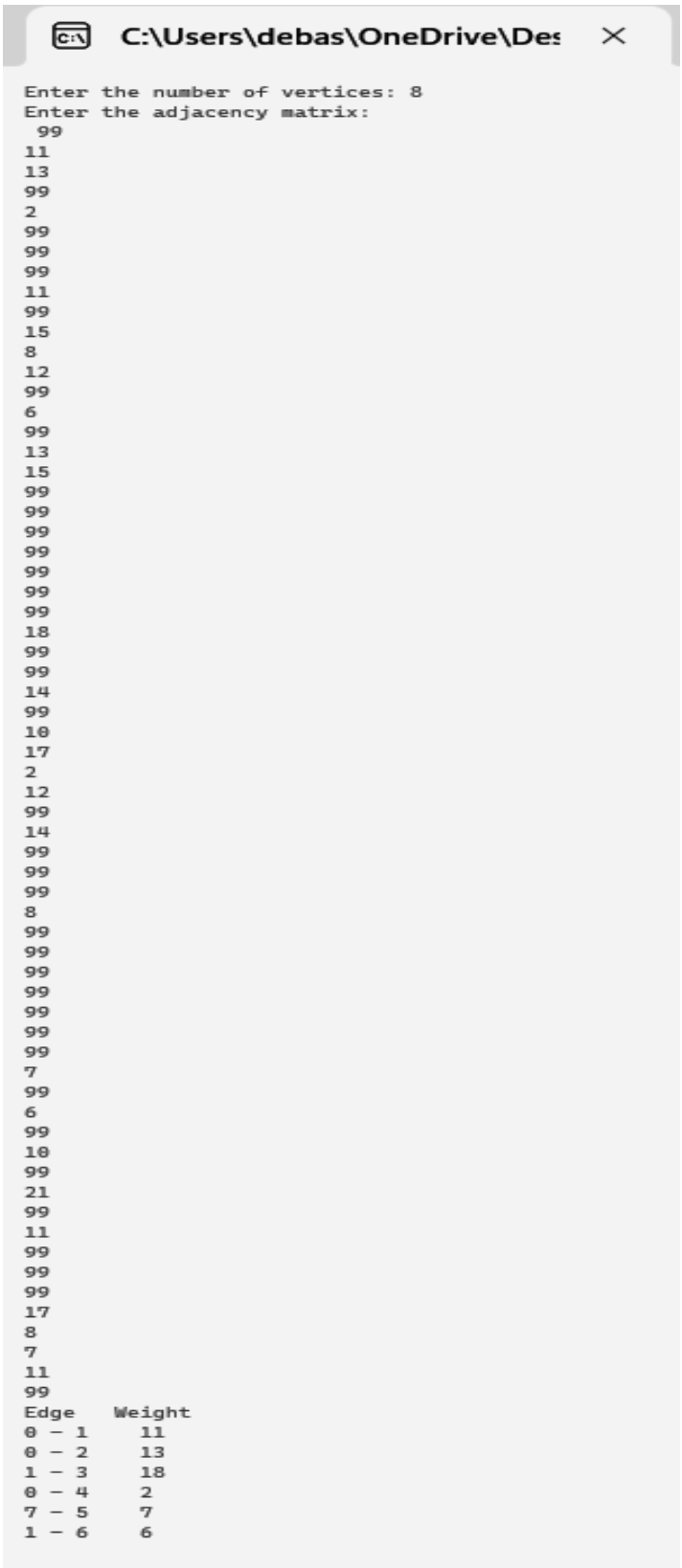
Write a C program to implement Minimum Cost Spanning Tree for the following graph by Prim's Algorithm.

Program Code:

```
#include <stdio.h>


int minKey(int key[], int mstSet[],
int V)
{
   int min = 99, min_index;
   for (int v = 0; v < V; v++)
      if (mstSet[v] == 0 && key[v]
< min)
         min = key[v], min_index =
v;
   return min_index;
}


void printMST(int parent[], int
graph[][100], int V)
{
   printf("Edge   Weight\n");
   for (int i = 1; i < V; i++)
      printf("%d - %d    %d \n",
parent[i], i, graph[i][parent[i]]);
}


void primMST(int graph[][100],
int V)
{
   int parent[V];
   int key[V];
   int mstSet[V];
   for (int i = 0; i < V; i++)
      key[i] = 99, mstSet[i] = 0;
   key[0] = 0;
```

```
    parent[0] = -1;

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

    {

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

        mstSet[u] = 1;

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

            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];

    }

    printMST(parent, graph, V);

}


int main()

{

    int V;

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

    scanf("%d", &V);

    int graph[100][100];

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

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

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

        }

    }

    primMST(graph, V);

    return 0;

}
```

Output:


C:\Users\debas\OneDrive\Des
Enter the number of vertices: 8
Enter the adjacency matrix:
99
11
13
99
2
99
99
99
11
99
15
8
12
99
6
99
13
15
99
99
99
99
99
99
99
99
18
99
99
14
99
10
17
2
12
99
14
99
99
99
8
99
99
99
99
99
99
99
7
99
6
99
10
99
21
99
11
99
99
99
17
8
7
11
99
Edge Weight
0 - 1 11
0 - 2 13
1 - 3 18
0 - 4 2
7 - 5 7
1 - 6 6

Experiment No. 9:

Problem Statement:

Write a C program to implement Minimum Cost Spanning Tree for the following graph by Kruskal's Algorithm.

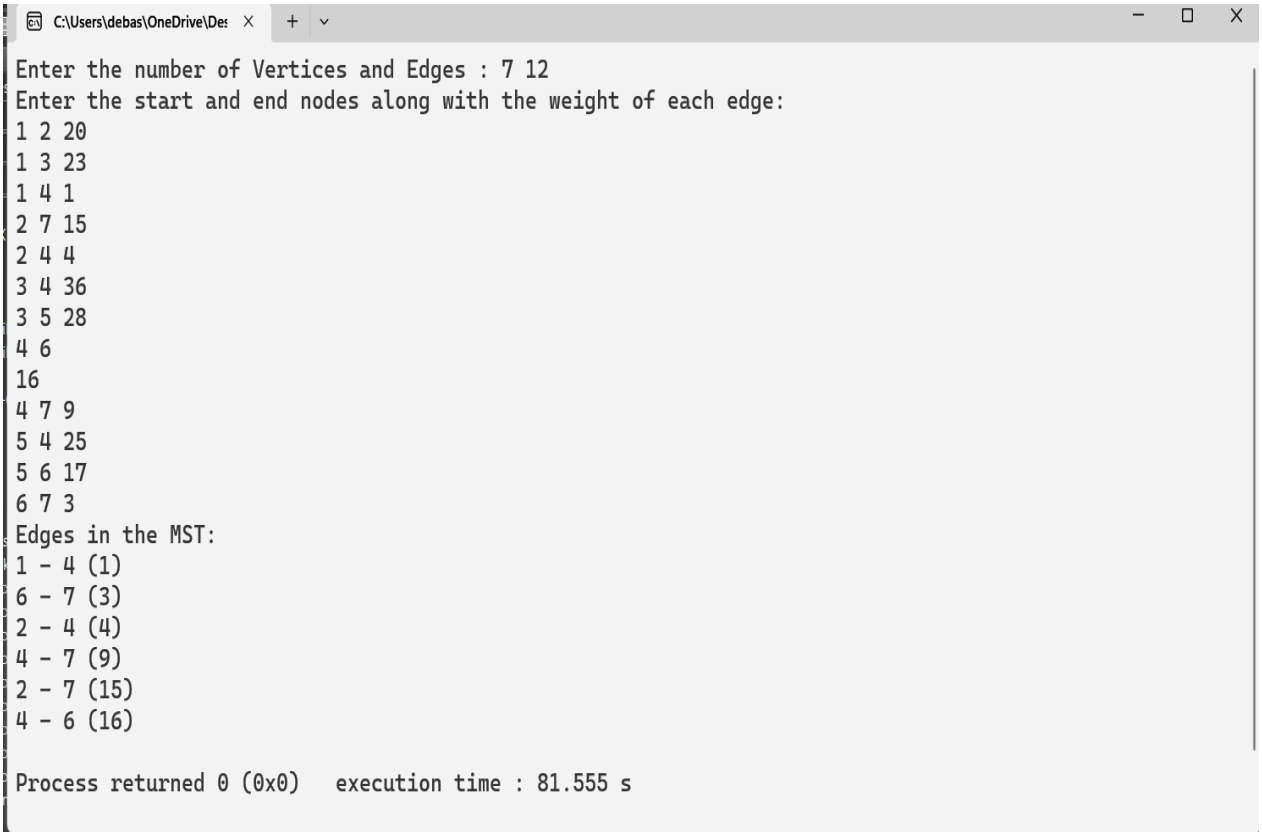
Program Code:

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

struct Edge {
   int node, dest, weight;
};
void kruskal(struct Edge T[], int V, int E)
{

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

   struct Edge result[V - 1];
   int count = 0;

   for (int i = 0; i < E - 1; i++)
   {
      for (int j = i + 1; j < E; j++)
      {
         if (T[i].weight > T[j].weight)
         {
            struct Edge temp = T[i];
            T[i] = T[j];
            T[j] = temp;
         }
      }
```

```
    }


    for (int i = 0; i < E && count < V
- 1; i++)
    {
        int nodeSubset =
subsets[T[i].node];
        int destSubset =
subsets[T[i].dest];

        if (nodeSubset !=
destSubset)
        {
            result[count++] = T[i];

            for (int j = 0; j < V; j++)
            {
                if (subsets[j] ==
destSubset)
                    subsets[j] =
nodeSubset;
            }
        }
    }

    printf("Edges in the MST:\n");
    for (int i = 0; i < count; i++)
        printf("%d - %d (%d)\n",
result[i].node, result[i].dest,
result[i].weight);
}

int main()
{
    int i;
    int E,V;

    printf("Enter the number of
Vertices and Edges : ");
    scanf("%d %d", &V, &E);

    struct Edge T[E];
```

```
   printf("Enter the start and end
nodes along with the weight of
each edge:\n");
   for(i = 0; i < E; i++)
   {
      {
         scanf("%d %d %d",
&T[i].node, &T[i].dest,
&T[i].weight);
      }
   }
   kruskal(T, V, E);
   return 0;
}
```

Output:



Experiment No. 10:

Problem Statement:

Write a C program to find the optimal sequence of jobs when number of jobs n=5,profit (p1,p2,p3,p4,p5)=(20,15,10,5,1) and deadline (d1,d2,d3,d4,d5)=(2,2,1,3,3).

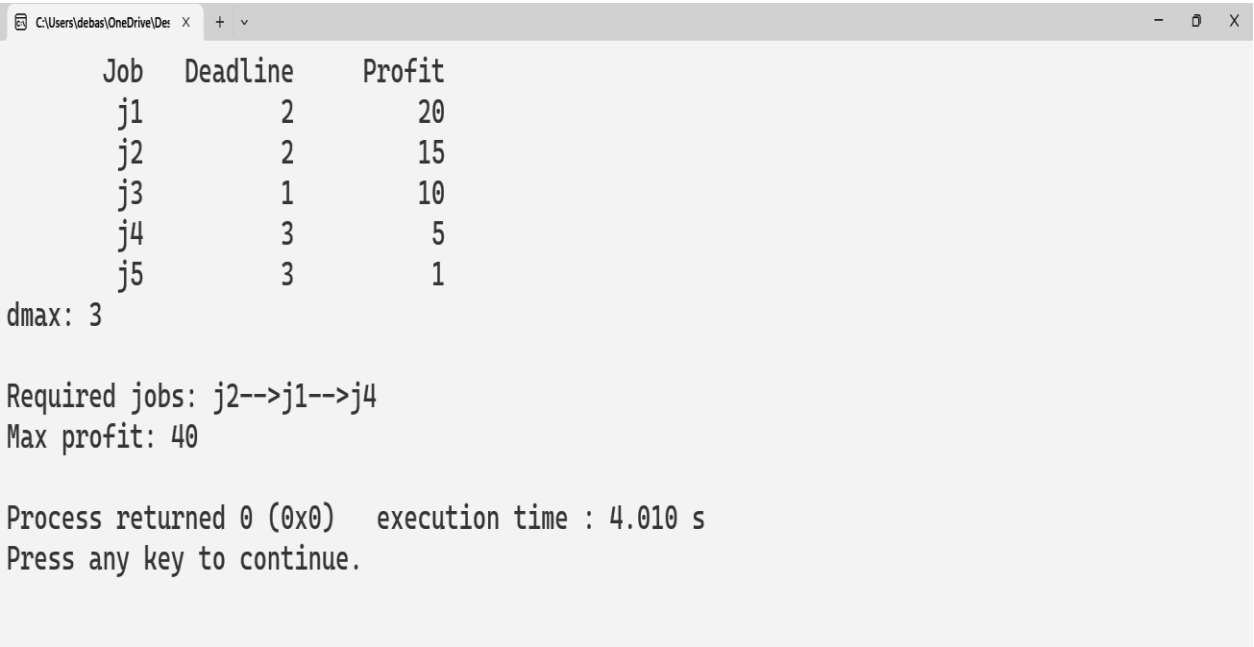
Program Code:

```
#include<stdio.h>

#define MAX 100

typedef struct Job{
        char id[5];
        int deadline;
        int profit;
}Job;

void jobseq(Job jobs[], int n);

int minvalue(int x,int y)

{
        if(x<y)
        return x;
        return y;
}

int main()
{
        int i,j;
        Job
jobs[5]={{"j1",2,20},{"j2",2,15},{"j
3",1,10},{"j4",3,5},{"j5",3,1}};
        Job temp;
        int n=5;
        for(i=1;i<n;i++)
        {
                for(j=0;j<n-i;j++)
                {

        if(jobs[j+1].profit>jobs[j].pr
ofit)
                        {
```

```
        temp=jobs[j+1];

        jobs[j+1]=jobs[j];

        jobs[j]=temp;
                    }
            }
        }
        printf("%10s %10s
%10s\n","Job","Deadline",
"Profit");
        for(i=0;i<n;i++)
        {
            printf("%10s %10i
%10i\n",
jobs[i].id,jobs[i].deadline,jobs[i].
profit);
        }
        jobseq(jobs, n);
}
void jobseq(Job jobs[],int n)
{
        int i,j,k,maxprofit;
        int timeslot[MAX];
        int filledslot=0;
        int dmax=0;
        for(i=0;i<n;i++)
        {

        if(jobs[i].deadline>dmax)

        dmax=jobs[i].deadline;
        }
        for(i=1;i<=n;i++)
        timeslot[i]=-1;
        printf("dmax: %d\n",
dmax);
        for(i=1;i<=n;i++)
        {

        k=minvalue(dmax,jobs[i-
1].deadline);
```

```
        while(k>=1)
        {
            if(timeslot[k]==-1)
            {
    timeslot[k]=i-1;
    filledslot++;
                    break;
            }
            k--;
        }
        if(filledslot==dmax)
        break;
    }
        printf("\nRrequired jobs: ");
        for(i=1;i<=dmax;i++)
        {
            printf("%s", jobs[timeslot[i]].id);
            if(i<dmax)
            printf("-->");
        }
        maxprofit=0;
        for(i=1;i<=dmax;i++)
        maxprofit+=jobs[timeslot[i]].profit;
        printf("\nMax profit: %d\n", maxprofit);
    }
```

Output:



Problem Statement:

Write a C program to implement Single Source Shortest Path for the following graph using Bellman-Ford Algorithm. Consider vertex 1 as the source.

Program Code:

```
#include <stdio.h>

#include <stdlib.h>

int Bellman_Ford(int G[20][20] , int V, int E,
int edge[20][2])

{

   int
i,u,v,k,distance[20],parent[20],S,flag=1;

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

      distance[i] = 1000 , parent[i] = -1 ;

      printf("Enter source: ");

      scanf("%d",&S);

      distance[S-1]=0 ;

   for(i=0;i<V-1;i++)

   {

      for(k=0;k<E;k++)

      {

         u = edge[k][0] , v = edge[k][1] ;

         if(distance[u]+G[u][v] < distance[v])

            distance[v] = distance[u] + G[u][v] ,
parent[v]=u ;
```

```
        }
    }
    for(k=0;k<E;k++)
        {
            u = edge[k][0] , v = edge[k][1] ;
            if(distance[u]+G[u][v] < distance[v])
                flag = 0 ;
        }
        if(flag)
            for(i=0;i<V;i++)
                printf("Vertex %d -> cost = %d
parent = %d\n",i+1,distance[i],parent[i]+1);


        return flag;
}
int main()
{
    int V,edge[20][2],G[20][20],i,j,k=0;
    printf("Enter no. of vertices: ");
    scanf("%d",&V);
    printf("Enter the adjency matrix of the
graph:\n");
    for(i=0;i<V;i++)
        for(j=0;j<V;j++)
        {
            scanf("%d",&G[i][j]);
            if(G[i][j]!=0)
                edge[k][0]=i,edge[k++][1]=j;


        }
    if(Bellman_Ford(G,V,k,edge))
        printf("\nNo negative weight cycle\n");
    else printf("\nNegative weight cycle
exists\n");
    return 0;
}
```

Output:


C:\Users\debas\OneDrive\Des
Enter no. of vertices: 7
Enter the adjency matrix of the graph:
0
6
5
5
0
0
0
0
0
-2
0
-1
0
0
0
-2
0
0
1
0
0
0
0
-2
0
0
-1
0
0
0
0
0
0
0
3
0
0
0
0
0
0
0
3
0
0
0
0
0
0
0
Enter source: 0
Negative weight cycle exists
Process returned 0 (0x0) execution time : 61.845 s