Aishwarya. M

1BM20CS401

CSE-4A

Program 10

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

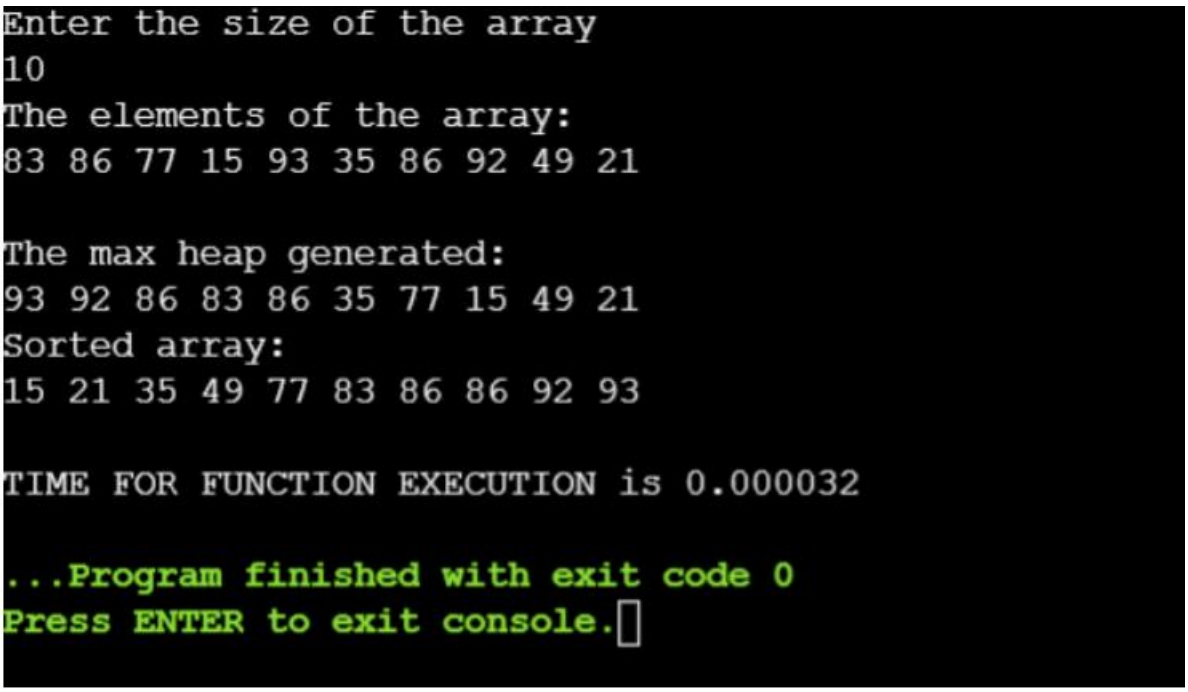
```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
 }
 void printArray(int arr[], int n) {
  for (int i = 0; i < n; ++i)
    printf("%d ", arr[i]);
  printf("\n");
 }
 void heapify(int arr[], int n, int i)
```

```
{
  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) {
    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);
  }
  printf("The max heap generated:\n");
  printArray(arr, n);
  for (int i = n - 1; i >= 0; i--) {
```

```
    swap(&arr[0], &arr[i]);
    heapify(arr, i, 0);
  }
 }
int main()
{
  int n;
       clock_t start, end;
  double cpu_time_used;
       printf("Enter the size of the array\n");
       scanf("%d",&n);
  int arr[n];
  printf("The elements of the array:\n");
  for(int i=0;i<n;i++)
  arr[i]=rand()%100;
  printArray(arr, n);
  printf("\n");
  start = clock();
  heapSort(arr, n);
  end = clock();
  cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

```
    printf("Sorted array: \n");
    printArray(arr, n);
    printf("\n");
    printf("TIME FOR FUNCTION EXECUTION is %f", cpu_time_used);
    return 0;
  }
```

Output :



Program 11:

Implement Warshall's algorithm using dynamic programming.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include <time.h>
int a[20][20];
int max(int,int);
void warshal(int p[20][20],int n)
{
        int i,j,k;
        for (k=1;k<=n;k++)
          for (i=1;i<=n;i++)
           for (j=1;j<=n;j++)
            p[i][j]=max(p[i][j],p[i][k]&&p[k][j]);
}
int max(int a,int b)
{
        if(a>b)
         return(a);
        else
```

```
        return(b);
}
void main() {
   int i,j,n;
   clock_t start, end;
   double cpu_time_used;
   printf("\n Enter number of vertices:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
     for(j=1;j<=n;j++)
     {
        a[i][j]=0;
     }
   }
   printf("\n Enter the adjacency matrix:\n");
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
      scanf("%d",&a[i][j]);
```

```
    }
  }
  start = clock();
  warshal(a,n);
  end = clock();
      printf("\n Transitive closure: \n");
      for (i=1;i<=n;i++) {
            for (j=1;j<=n;j++)
              printf("%d\t",a[i][j]);
            printf("\n");
      }
      cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
  printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);
  getch();
}
```

Output :

```
 Enter number of vertices:4

 Enter the adjacency matrix:
0 1 1 0
1 0  0 1
0 0  0 0
1 0 1 0

 Transitive closure:
1       1       1       1
1       1       1       1
0       0       0       0
1       1       1       1
TIME FOR FUNCTION EXECUTION is 0.000004


...Program finished with exit code 0
Press ENTER to exit console.
```

Program 12:

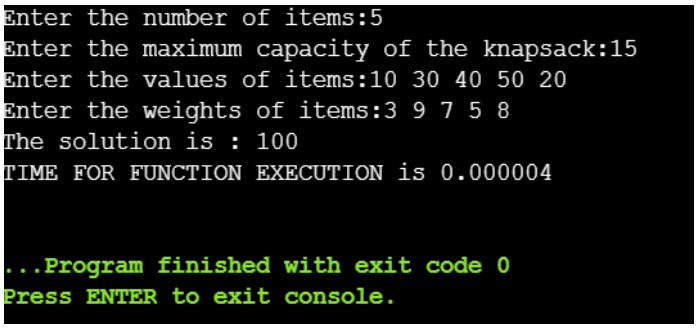
Implement 0/1 Knapsack problem using dynamic programming.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include <time.h>
int max(int a, int b) {
  if(a>b){
    return a;
  } else {
    return b;
  }
}
int knapsack(int W, int wt[], int val[], int n) {
  int i, w;
  int knap[n+1][W+1];
  for (i = 0; i <= n; i++) {
    for (w = 0; w <= W; w++) {
      if (i==0 || w==0)
        knap[i][w] = 0;
      else if (wt[i-1] <= w)
```

```
            knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
          else
            knap[i][w] = knap[i-1][w];
      }}
   return knap[n][W];
}
int main()
{
   int W;
   int n;
   clock_t start, end;
   double cpu_time_used;
   printf("Enter the number of items:");
   scanf("%d",&n);
   int val[n];
   int wt[n];
   printf("Enter the maximum capacity of the knapsack:");
   scanf("%d",&W);
   printf("Enter the values of items:");
   for(int i=0;i<n;i++)
   {
```

```
    scanf("%d",&val[i]);
  }
  printf("Enter the weights of items:");
  for(int i=0;i<n;i++)
  {
    scanf("%d",&wt[i]);
  }
  start = clock();
  int sol=knapsack(W, wt, val, n);
  end = clock();
  printf("The solution is : %d\n", sol);
  cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
  printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);
  getch();
  return 0;
}
```

Output



Program 13 :

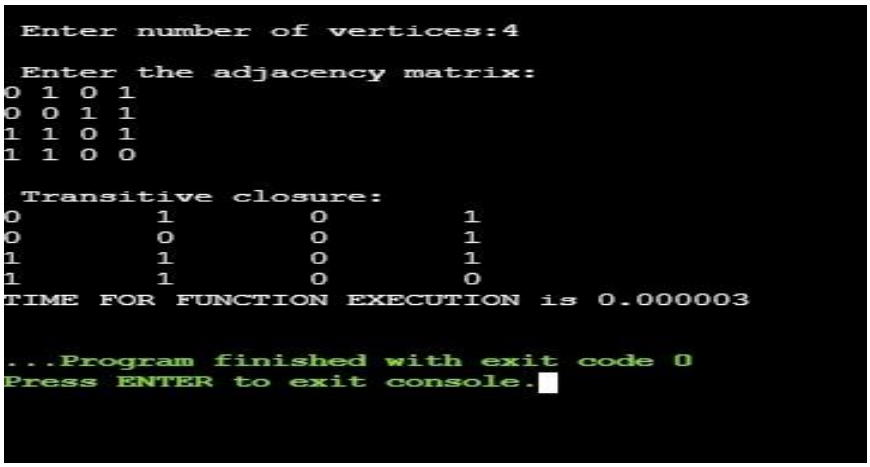
Implement All Pair Shortest paths problem using Floyd's algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include <time.h>
double inf=INFINITY;
int a[20][20];
int min(int,int);
void warshal(int p[20][20],int n)
{
        int i,j,k;
        for (k=1;k<=n;k++)
        {
          for (i=1;i<=n;i++)
           {
              for (j=1;j<=n;j++)
             {
               if(i==j)
            p[i][j]=0;
          else
```

```
            p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
             }}}}
int min(int a,int b)
{
 if(a<b)
  return(a);
 else
  return(b);
}
void main() {
    int i,j,n;
    clock_t start, end;
    double cpu_time_used;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    printf("\n Enter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
        scanf("%d",&a[i][j]);
```

```
    }
  }
  start = clock();
  warshal(a,n);
  end = clock();
      printf("\n Transitive closure: \n");
      for (i=1;i<=n;i++) {
            for (j=1;j<=n;j++)
              printf("%d\t",a[i][j]);
            printf("\n");
      }
      cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
  printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);
  getch();
}
```

Output:



Program 14:

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

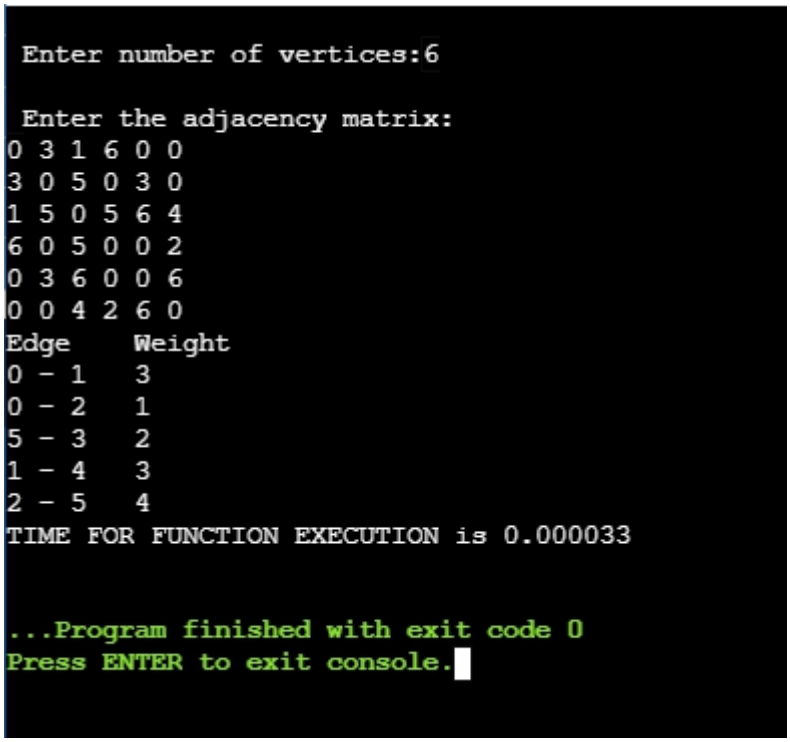
```
#include<stdio.h>

#include<conio.h>

#include <limits.h>

#include <time.h>

int a[20][20];

void printMST(int parent[],int n)

{

   printf("Edge \tWeight\n");

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

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

}

int findMinVertex(int visited[],int weight[],int n)

{

   int minVertex = -1; // Initialized to -1 means there is no vertex till now

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

    {

     if (!visited[i] && (minVertex == -1 || weight[i] < weight[minVertex]))
```

```
      {
      minVertex = i;
}}    return minVertex;
}
void prim(int n)
{
   int parent[n];
   int weight[n];
   int visited[n];
   for(int i=0;i<n;i++)
   {
      visited[i]=0;
      weight[i]=INT_MAX;
   }
   weight[0]=0;
   parent[0]=-1;
   for(int count=0;count<n-1;count++)
   {
      int minVertex=findMinVertex(visited,weight,n);
      visited[minVertex]=1;
         for (int j = 0; j < n; j++)
```

```
        {
            if(a[minVertex][j] != 0 && !visited[j])
            {
                if(a[minVertex][j] < weight[j])
                {
                // updating weight array and parent array
                weight[j] = a[minVertex][j];
                parent[j] = minVertex;
                }}}}
    printMST(parent,n);
}
void main() {
    int i,j,n;
    clock_t start, end;
    double cpu_time_used;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);

    printf("\n Enter the adjacency matrix:\n");
    for(i=0;i<n;i++)
    {
```

```
      for(j=0;j<n;j++)
      {
      scanf("%d",&a[i][j]);
      }
   }
   start = clock();
   prim(n);
   end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
   printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);
   getch();
}
```

Output :



Program 15 :

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.

```
#include<stdio.h>
#include<conio.h>
#include <limits.h>
#include <time.h>
int src[20];
int dest[20];
int wt[20];
int parent[20];
void sort(int wt[],int n)
{
   int i, j;
  for (i = 0; i < n-1; i++)
   {
      for (j = 0; j < n-i-1; j++)
      {
         if (wt[j] > wt[j+1])
         {
            int temp=wt[j];
      wt[j]=wt[j+1];
```

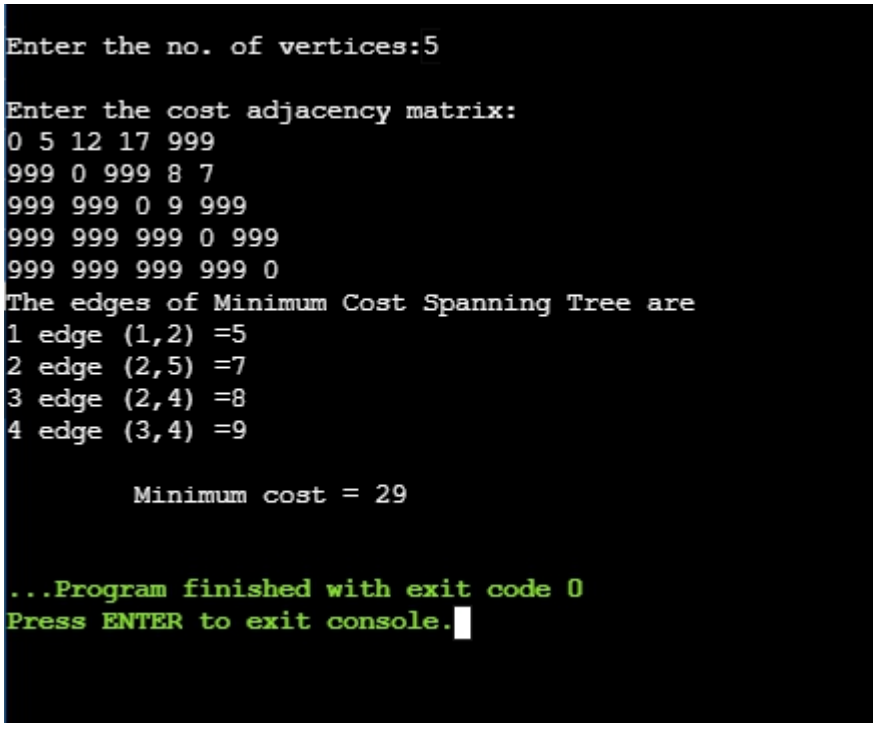
```
        wt[j+1]=temp;

        temp=src[j];
        src[j]=src[j+1];
        src[j+1]=temp;

        temp=dest[j];
        dest[j]=dest[j+1];
        dest[j+1]=temp;
           }
        }
    }

}
int findParent(int v,int parent[])
{
    if(parent[v]==v)
    {
        return v;
    }
    findParent(parent[v],parent);
```

```
}
void kruskal(int n,int e)
{
   int output[n-1];
   for(int i=0;i<n;i++)
   {
      parent[i]=i;
   }
   int count=0;
   int i=0;
   printf("\nEdge Weight");
   while(count!=n-1)
   {
      int srcParent=findParent(src[i],parent);
      int destParent=findParent(dest[i],parent);
      if(srcParent!=destParent)
      {
         output[count]=wt[i];
         printf("\n%d-%d=%d",src[i],dest[i],output[count]);
         count++;
         parent[srcParent]=destParent;
```

```
        }
        i++;
    }
}
void main() {
    int i,j,n,e;
    clock_t start, end;
    double cpu_time_used;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    printf("\n Enter number of edges:");
    scanf("%d",&e);
    int x=0;
    printf("\n Enter the source,destination,weight of edge
respectively:\n");
    for(i=0;i<e;i++)
    {
        scanf("%d%d%d",&src[i],&dest[i],&wt[i]);
    }

    start = clock();
    sort(wt,e);
```

```
   kruskal(n,e);
   end = clock();

        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
   printf("\nTIME FOR FUNCTION EXECUTION is %f\n",
cpu_time_used);
   getch();
}
```

Output:



Program 16:

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

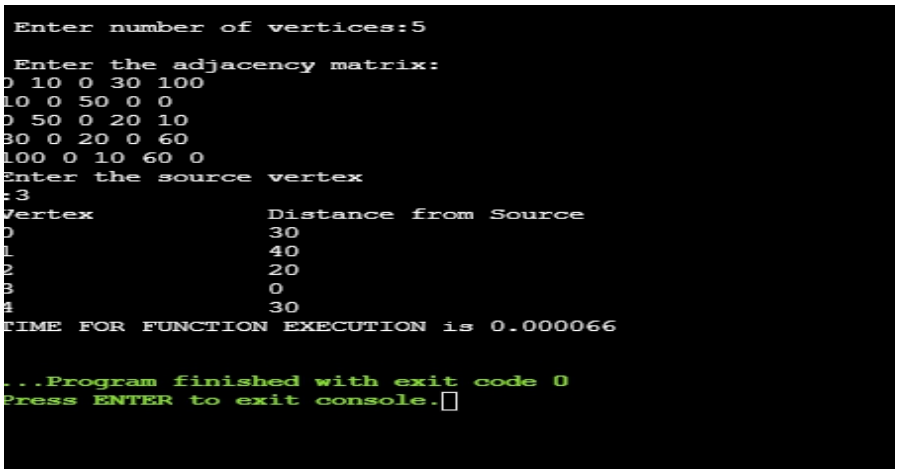
```
#include<stdio.h>
#include<conio.h>
#include <limits.h>
#include <time.h>
int a[20][20];
void printSolution(int dist[],int n)
{
   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < n; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}
int findMinVertex(int visited[],int dist[],int n)
{
   int minVertex = -1;
    for (int i = 0; i < n; i++)
    {
      if (!visited[i] && (minVertex == -1 || dist[i] < dist[minVertex]))
      {
         minVertex = i;
```

```
      }
    }
    return minVertex;
}

void dijsktra(int n,int src)
{
   int visited[n];
   int dist[n];
   for(int i=0;i<n;i++)
   {
      visited[i]=0;
      dist[i]=INT_MAX;
   }
   dist[src]=0;
   for(int i=0;i<n-1;i++)
   {
      int minVertex=findMinVertex(visited,dist,n);
      visited[minVertex]=1;
      for(int j=0;j<n;j++)
      {
```

```
            if(a[minVertex][j]!=0 && !visited[j])
            {
               if(a[minVertex][j]+dist[minVertex]<dist[j])
               {
                  dist[j]=a[minVertex][j]+dist[minVertex];
               }
            }
         }
      }
      printSolution(dist,n);
}
void main() {
      int i,j,n;
      clock_t start, end;
      double cpu_time_used;
      int src;
      printf("\n Enter number of vertices:");
      scanf("%d",&n);

      printf("\n Enter the adjacency matrix:\n");
      for(i=0;i<n;i++)
```

```
    {
       for(j=0;j<n;j++)
       {
       scanf("%d",&a[i][j]);
       }
    }
    printf("Enter the source vertex\n:");
    scanf("%d",&src);
    start = clock();
    dijsktra(n,src);
    end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);
    getch();
}
```

Output :



Program 17 :

Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set S =

{s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S =

{1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and

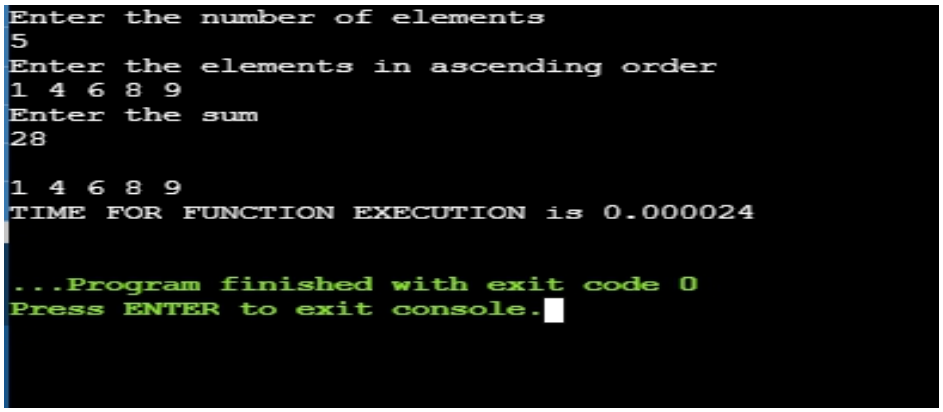
{1,8}. A suitable message is to be displayed if the given problem instance doesn’t have a solution.

```
#include<stdio.h>

#include<conio.h>

#include <limits.h>

#include <time.h>

int w[20],x[20],m,flag=0;

void sum_of_subsets(int s,int k,int r)

{

        x[k]=1;

        if(s+w[k]==m)

        {

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

                {

                        if(x[i]==1)

                        printf("%d ",w[i]);
```

```
                flag=1;
            }
            printf("\n");
        }
        else if(s+w[k]+w[k+1]<=m)
        sum_of_subsets(s+w[k],k+1,r-w[k]);
        if((s+r-w[k]>=m)&&(s+w[k+1]<=m))
        {
            x[k]=0;
            sum_of_subsets(s,k+1,r-w[k]);
        }
}
int main()
{
        int i,n,r=0;
        clock_t start, end;
    double cpu_time_used;
        printf("Enter the number of elements\n");
        scanf("%d",&n);
        printf("Enter the elements in ascending order\n");
        for(i=0;i<n;i++){
```

```
        scanf("%d",&w[i]);
        r=r+w[i];}
        printf("Enter the sum\n");
        scanf("%d",&m);
        printf("\n");
        start=clock();
        sum_of_subsets(0,0,r);
        end=clock();
        if(flag==0)
        printf("No solution\n");
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);
    getch();
        return 0;
}
```

Output:



Program 18:

Implement “N-Queens Problem” using Backtracking

```
#include<stdio.h>
#include<conio.h>
#include <limits.h>
#include <time.h>
int board[10][10];
int isSafe(int board[][10],int i,int j,int n)
{
   for(int row=0;row<i;row++)
   {
      if(board[row][j]==1)
      {
         return 0;
      }
   }
   int x=i;
   int y=j;
   while(x>=0 && y>=0)
   {
```

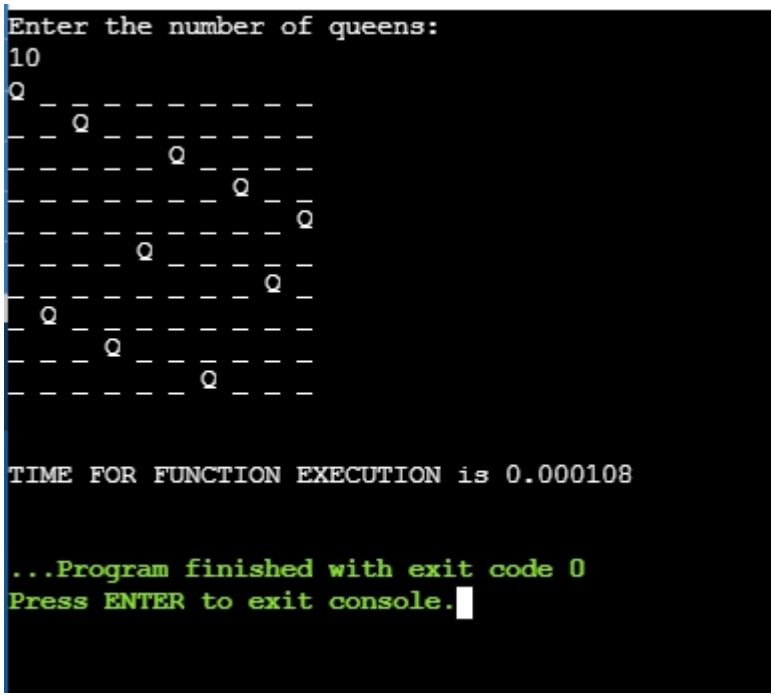
```
        if(board[x][y]==1)
        {
            return 0;
        }
        x--;
        y--;
    }
    x=i;
    y=j;
    while(x>=0 && y<n)
    {
        if(board[x][y]==1)
        {
            return 0;
        }
        x--;
        y++;
    }
    return 1;
}
int solveNQueen(int board[][10],int i, int n)
```

```
{
    if(i==n)
    {
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(board[i][j]==1)
                {
                    printf("Q ");
                }
                else
                {
                    printf("_ ");
                }
            }
            printf("\n");
        }
        printf("\n\n");
        return 1;
    }
```

```
    for(int j=0;j<n;j++)
    {
        if(isSafe(board,i,j,n))
        {
            board[i][j]=1;
            int nextQueenPossible= solveNQueen(board,i+1,n);
            if(nextQueenPossible==1)
            {
              return 1;
            }
            board[i][j]=0;
        }
    }
    return 0;
}
int main()
{
    int n;
    clock_t start, end;
    double cpu_time_used;
    printf("Enter the number of queens:\n");
```

```
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            board[i][j]=0;
        }
    }
    start=clock();
    int val=solveNQueen(board,0,n);
    end=clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);
    getch();
    return 0;
}
```

Output:


Enter the number of queens:
10
Q _ _ _ _ _ _ _ _ _
_ _ Q _ _ _ _ _ _ _
_ _ _ _ _ Q _ _ _ _
_ _ _ _ _ _ _ Q _ _
_ _ _ _ _ _ _ _ _ Q
_ _ _ _ Q _ _ _ _ _
_ _ _ _ _ _ _ _ Q _
_ Q _ _ _ _ _ _ _ _
_ _ _ Q _ _ _ _ _ _
_ _ _ _ _ _ Q _ _ _
TIME FOR FUNCTION EXECUTION is 0.000108
...Program finished with exit code 0
Press ENTER to exit console.