VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

Analysis and Design Of Algorithms
LAB REPORT

(19CS4PCADA)

Submitted by

RAVI SAJJANAR (1BM19CS127)

Under the Guidance of
Dr. Nagarathna N
Professor, BMSCE

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

Mar-2021 to Jun-2022

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 Assignment work entitled "**Database Management System** " carried out by **RAVI SAJJANAR (1BM19CS127)** who is the bonafide students 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 Visveswaraiah Technological University, Belgaum during the year 2021-2022. The Lab report has been approved as it satisfies the academic requirements in respect of **Analysis and Design Of Algorithms (19CS4PADA) LAB** work prescribed for the said degree.

Signature of the Guide Dr.
Dr.Nagarathna N
Professor
BMSCE, Bengaluru

Signature of the HOD
Dr. Umadevi V
Associate Prof.& Head, Dept. of CSE
BMSCE, Bengaluru

External Viva

Name of the Examiner — Signature with date

1._____ _____

2. _____ _____

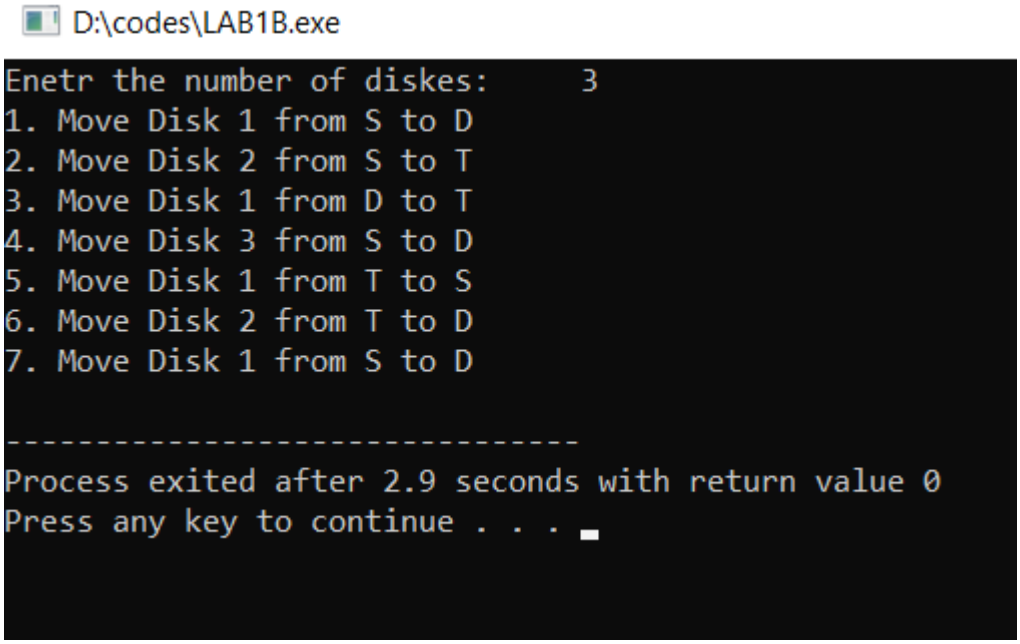
INDEX

PROGRAM 1:

Write a recursive program to

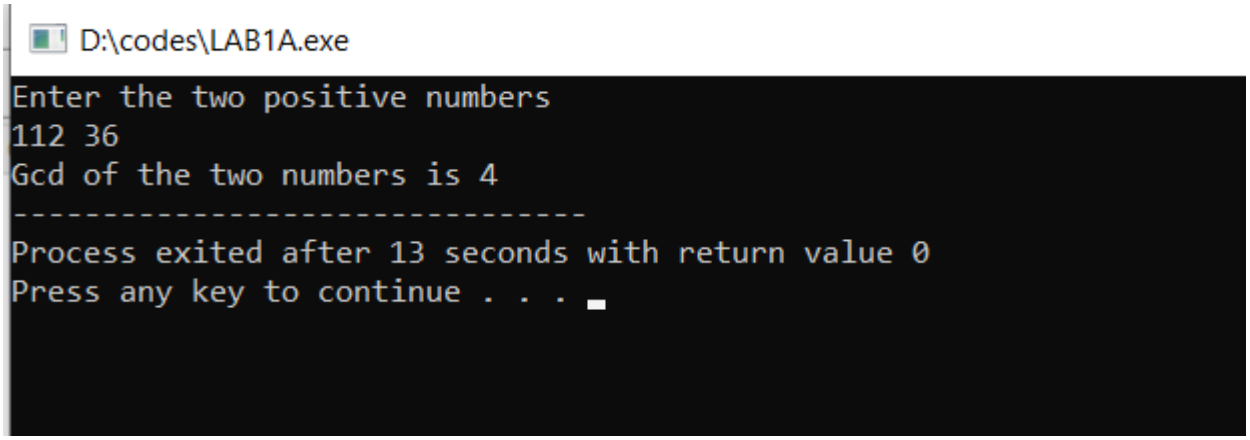
a) Solve Towers-of-Hanoi problem b) To find GCD

```
// Tower of hennoi
#include<stdio.h>

int count=1;

void TowerOfHanoi(int n,char src,char temp,char des){
	if(n==1){
	printf("%d. Move Disk %d from %c to %c\n",count++,n,src,des);
	return;
	}

	TowerOfHanoi(n-1,src,des,temp);
	printf("%d. Move Disk %d from %c to %c\n",count++,n,src,des);
	TowerOfHanoi(n-1,temp,src,des);
}
void main(){
	int n=0;
	printf("Enetr the number of diskes:\t");
	scanf("%d",&n);

	TowerOfHanoi(n,'S','T','D');
}
```

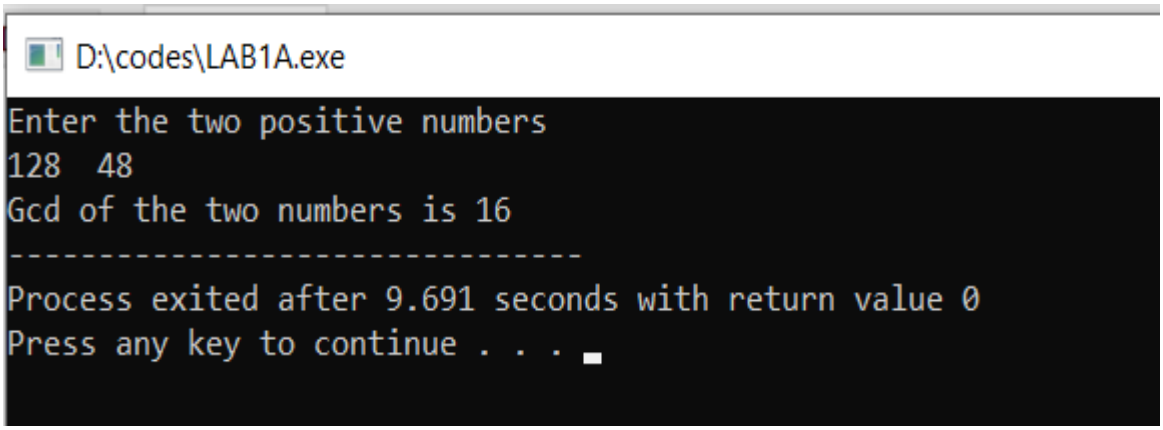



```
// Iterative GCD

#include <stdio.h>

int gcd(int a,int b){
   int r;
   while (b!=0){
      r=a%b;
      a=b;
      b=r;
   }
   return a;
}

int main()
{
   int a,b,res;
```

```
    printf("Enter the two positive numbers\n");
    scanf("%d%d",&a,&b);
    res=gcd(a,b);
    printf("Gcd of the two numbers is %d",res);
    return 0;
}
```



```
//Recersive GCD

#include <stdio.h>

int gcd(int a,int b){
   if (b==0) return a;
   else return gcd(b,(a%b));
}

int main()
{
   int a,b,res;
   printf("Enter the two positive numbers\n");
   scanf("%d%d",&a,&b);
   res=gcd(a,b);
   printf("Gcd of the two numbers is %d",res);
   return 0;
}
```



PROGRAM 2: Linear & Binary Search

Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.

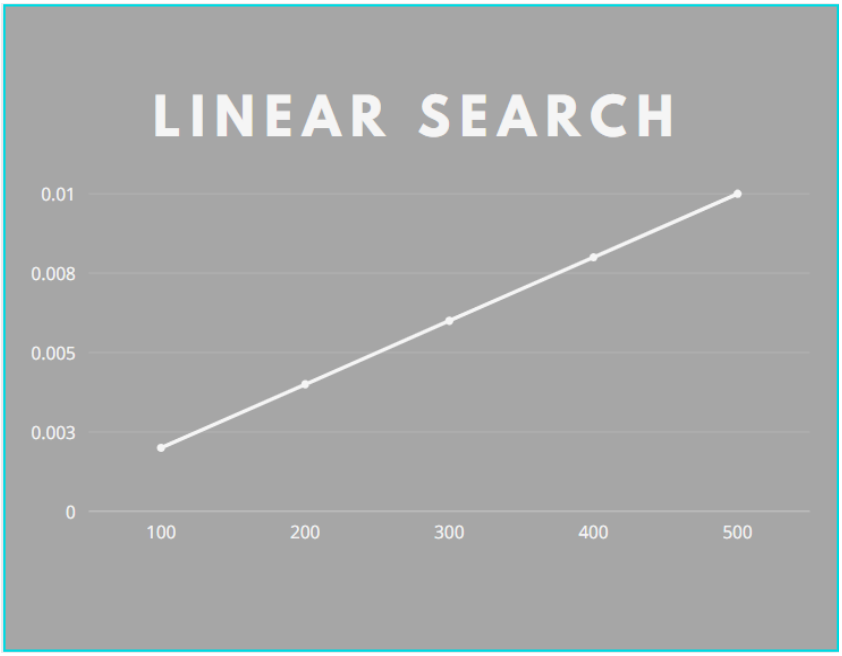
Linear Search

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

int arr[100000];

int linear_search(int arr[], int key,int i,int n)
{ int a,b;
  for(a=0;a<1000;a++)
  {
    for(b=0;b<10000;b++)
    {

    }
  }

  if(i>n)
  return -1;
  if(arr[i]==key)
  return i;
  else
  {
    linear_search(arr,key,i+1,n);
  }
}

int main()
```

```
{
  int key,res,n,i;
  time_t start,end;

  printf("Enter the no of elements \n");
  scanf("%d",&n);

  for(i=0;i<n;i++)
  {
    arr[i]=rand();
  }

  printf("The elements are :\n");
   for(i=0;i<n;i++)
  {
    printf("%d\t",arr[i]);
  }
  printf("\n");

  printf("Enter element to be searched\n");
  scanf("%d",&key);



  start=time(NULL);
  res=linear_search(arr,key,0,n);
  end=time(NULL);

  if(res==-1)
    printf("Element not found in linear search\n");
  else
  printf("Element found in linear search at pos %d\n",res);

  printf("The time taken is %.10f",difftime(end,start)/CLOCKS_PER_SEC);

  return 0;
}
```

```
C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 2 ( Recursive Binary and Linear Search)\linearsearchrecursive.exe
Enter the no of elements
100
The elements are :
41      18467   6334    26500   19169   15724   11478   29358   26962   24464   5705    28145   23281   16827   9961
        491     2995    11942   4827    5436    32391   14604   3902    153     292     12382   17421   18716   19718
        19895   5447    21726   14771   11538   1869    19912   25667   26299   17035   9894    28703   23811   31322
        30333   17673   4664    15141   7711    28253   6868    25547   27644   32662   32757   20037   12859   8723
        9741    27529   778     12316   3035    22190   1842    288     30106   9040    8942    19264   22648   27446
        23805   15890   6729    24370   15350   15006   31101   24393   3548    19629   12623   24084   19954   18756
        11840   4966    7376    13931   26308   16944   32439   24626   11323   5537    21538   16118   2082    22929
        16541
Enter element to be searched
1
Element not found in linear search
The time taken is 0.0020000000
--------------------------------
Process exited after 4.752 seconds with return value 0
Press any key to continue . . .
```

Linear Search Array size	Time in sec
100	0.002
200	0.004
300	0.006
400	0.008
500	0.010

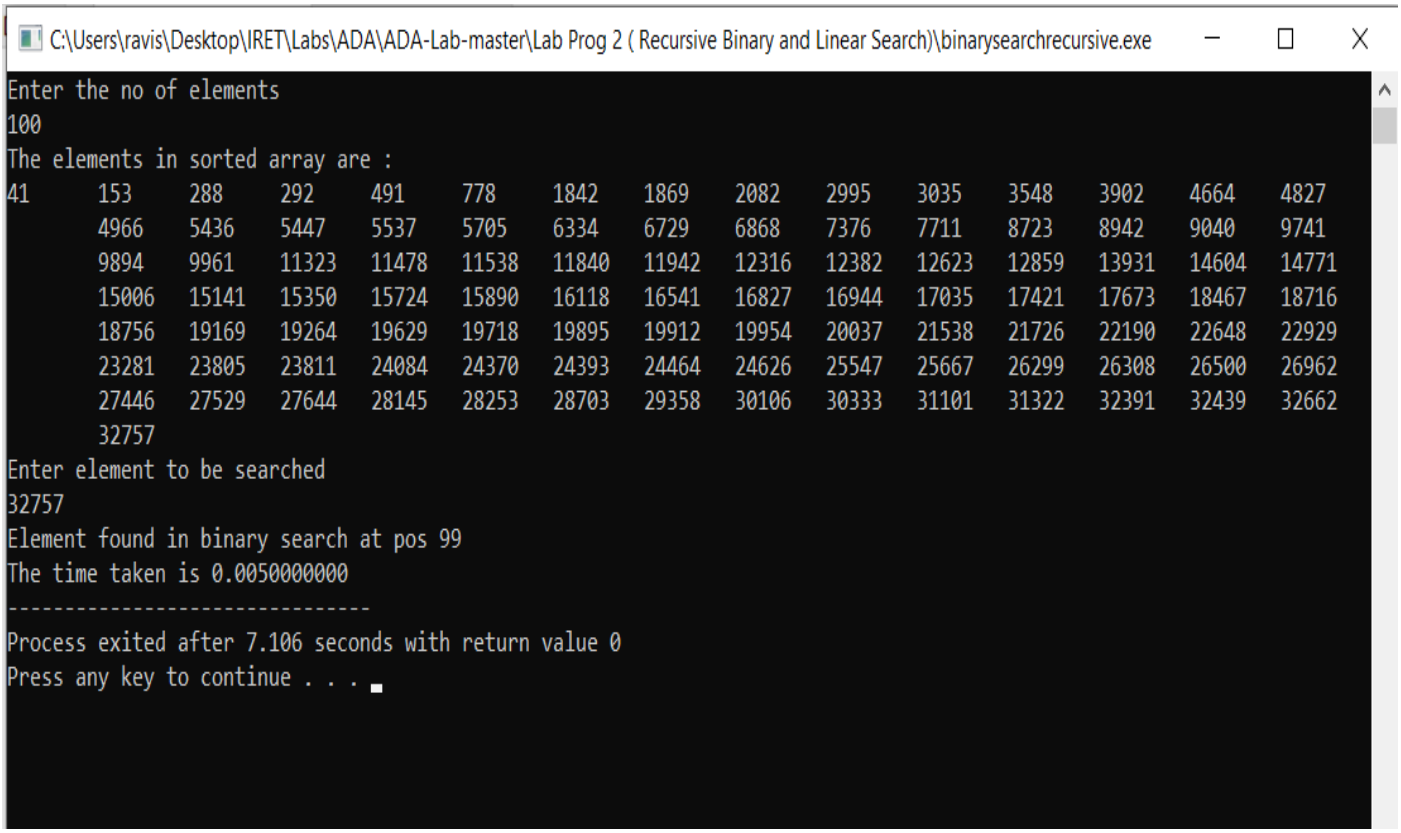


Binary Search

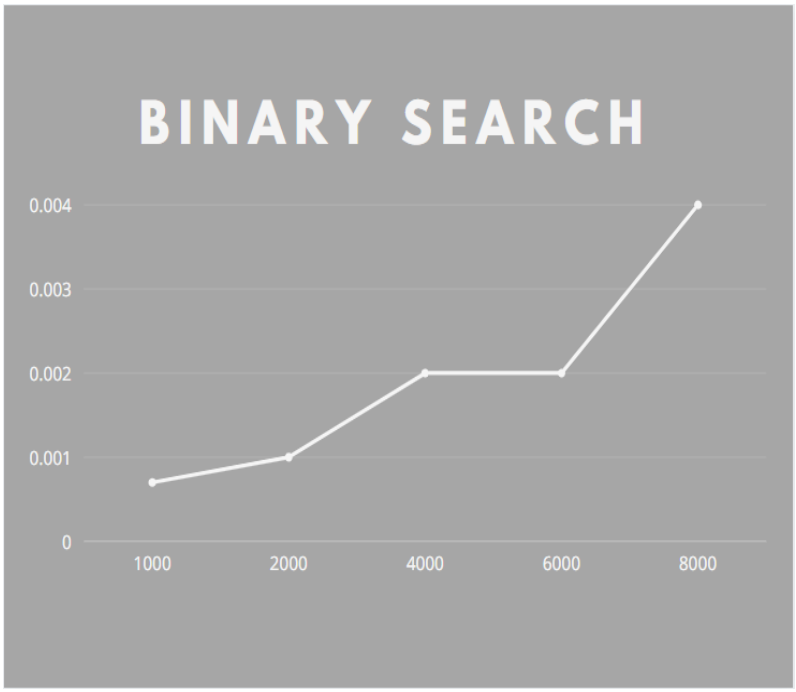
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int arr[100000];

int binarySearch(int arr[], int l, int r, int x)
{
        int a,b;
  for(a=0;a<1000;a++)
  {
    for(b=0;b<10000;b++)
    {

    }
  }
   if (r >= l)
   {
       int mid = l + (r - l)/2;

       if (arr[mid] == x)
       return mid;
       if (arr[mid] > x)
      return binarySearch(arr, l, mid-1, x);


       return binarySearch(arr, mid+1, r, x);
   }
   return -1;
}

void sort(int arr[],int n)
{
    int temp;
    int i,j;
    for(i=0;i<n;i++)
```

```
    {
       for(j=i+1;j<n;j++)
       {
          if(arr[i]>=arr[j])
          {
            temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
          }
       }
    }
}


int main()
{
  int key,res,n,i;
  time_t start,end;

  printf("Enter the no of elements \n");
  scanf("%d",&n);

  for(i=0;i<n;i++)
  {
    arr[i]=rand();
  }

  start=time(NULL);
  sort(arr,n);
  printf("The elements in sorted array are :\n");
   for(i=0;i<n;i++)
  {
    printf("%d\t",arr[i]);
  }
  printf("\n");


  printf("Enter element to be searched\n");
```

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

  res=binarySearch(arr,0,n-1,key);
  end=time(NULL);

  if(res==-1)
    printf("Element not found in binary search\n");
  else
  printf("Element found in binary search at pos %d\n",res);


  printf("The time taken is %.10f",difftime(end,start)/CLOCKS_PER_SEC);

  return 0;
}
```



Binary Search Array size	Time in sec
1000	0.0007
2000	0.001
4000	0.002
6000	0.002
8000	0.004



PROGRAM 3: Selection Sort

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int arr[1000000];

int swap(int arr[], int index1, int index2)
{
   int temp = arr[index2];
   arr[index2] = arr[index1];
   arr[index1] = temp;
}

int FindMin(int arr[], int n, int i)
{
   int minindex = i, temp;

   for (int j = i; j < n; j++)
   {
      if (arr[j] < arr[minindex])
      {
         minindex = j;
      }
   }
   return minindex;
}

void SelectionSort(int arr[], int n)
{

   for (int i = 0; i < n; i++)
   {
      int min = FindMin(arr, n, i);
```

```
        swap(arr, min, i);
    }
}

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

int main()
{
    time_t start, end;
    int n;
    srand(time(0));
    printf("Enter the no of elements \n");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        arr[i] = rand();
    }

    start = time(NULL);
    SelectionSort(arr, n);
    end = time(NULL);

    printf("The array is sorted\n");
    // printf("The sorted array is: \n");
    // printArray(arr, n);

    printf("The time taken is %.10f\n", difftime(end, start) / CLOCKS_PER_SEC);
    return 0;
}
```

```
C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 3 ( Selection Sort and bubble sort  )\selectionSort.exe
Enter the no of elements
20000
The array is sorted
The time taken is 0.0010000000

--------------------------------
Process exited after 9.846 seconds with return value 0
Press any key to continue . . .
```

Bubble Sort Array size	Time in sec
20000	0.001
40000	0.002
60000	0.004
80000	0.007
100000	0.011

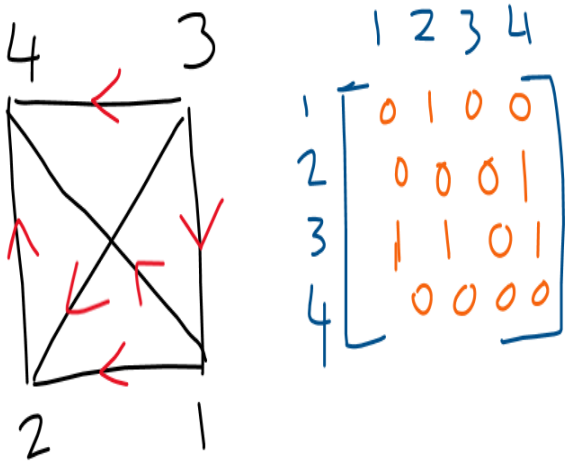
PROGRAM 4: BFS & DFS

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.

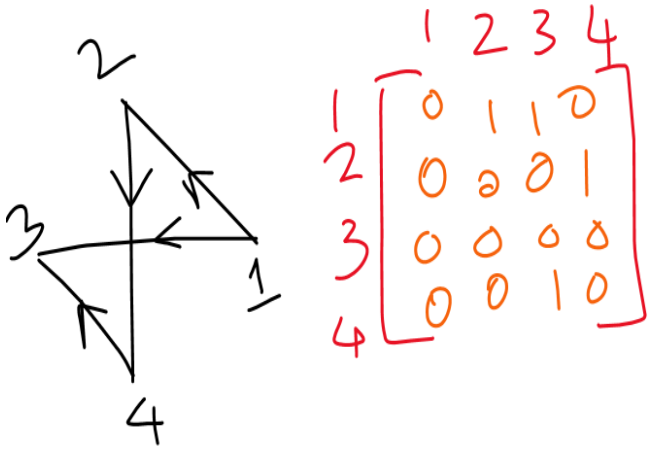
BFS:

```
#include <stdio.h>
#include <conio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

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

void main()
{
   int v;

   printf("\n Enter the number of vertices:");
   scanf("%d", &n);
```

```
    for (i = 1; i <= n; i++)
    {
       q[i] = 0;
       visited[i] = 0;
    }
    printf("\n Enter graph data in matrix form:\n");
    for (i = 1; i <= n; i++)
       for (j = 1; j <= n; j++)
          scanf("%d", &a[i][j]);
    printf("\n Enter the starting vertex:");
    scanf("%d", &v);
    bfs(v);
    printf("\n The node which are reachable are:\n");
    for (i = 1; i <= n; i++)
       if (visited[i])
          printf("%d\t", i);
    getch();
}
```




```
C:\softwares_sajjan\c code\lab_4_ada.exe
enter number of vertices
4
Enter adjacency matrix
0 1 0 1
0 0 0 1
1 1 0 1
0 0 0 0
enter starting vertex
1
The nodes reachable from src are
124
--------------------------------
Process exited after 39.12 seconds with return value 0
Press any key to continue . . .
```

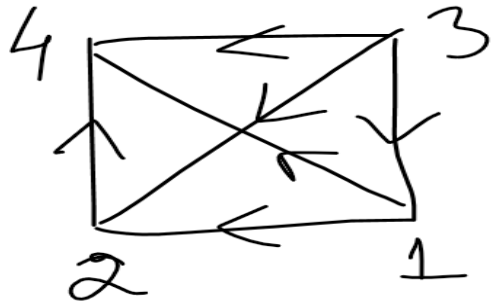


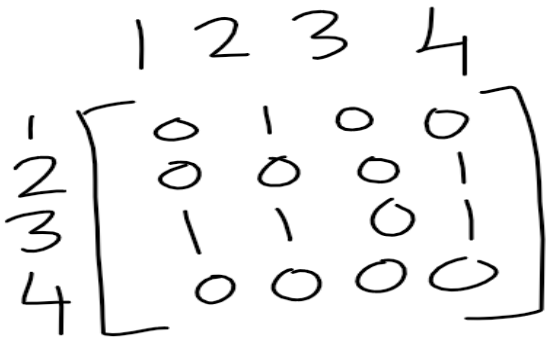

```
C:\softwares_sajjan\c code\lab_4_ada.exe
enter number of vertices
4
Enter adjacency matrix
0 1 1 0
0 0 0 1
0 0 0 0
0 0 1 0
enter starting vertex
1
The nodes reachable from src are
1234
--------------------------------
Process exited after 35.31 seconds with return value 0
Press any key to continue . . .
```

DFS:

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

int a[20][20], reach[20], n;

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

void main()
{
   int i, j, count = 0;
```

```
    printf("\n Enter number of vertices:");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
    {
       reach[i] = 0;
       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]);

    dfs(1);
    printf("\n");
    for (i = 1; i <= n; i++)
    {
       if (reach[i])
          count++;
    }

    if (count == n)
       printf("\n Graph is connected");
    else
       printf("\n Graph is not connected");
    getch();
}
```



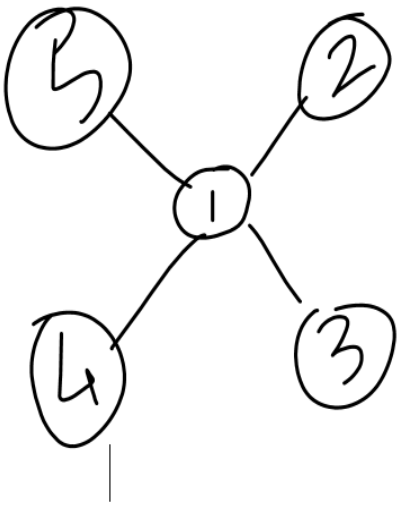


C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 4 (BFS and DFS)\ConnectedDFS.exe

```
Enter number of vertices:4

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

1->2
2->4

Graph is not connected
```

C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 4 (BFS and DFS)\ConnectedDFS.exe

```
Enter number of vertices:5

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

1->2
1->3
1->4
1->5

Graph is connected
```



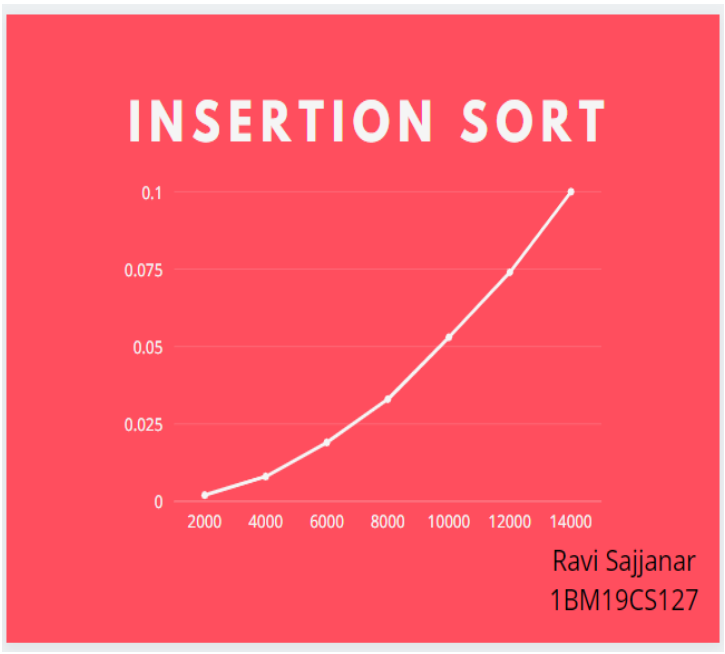
PROGRAM 5: Insertion Sort

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

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

int arr[1000000];
void insetionSort(int arr[], int n)
{
   int i,j,curr;
   for ( i = 1; i < n; i++)
   {
       curr = arr[i];
       j = i - 1;
      while (j >= 0 && curr < arr[j])
      {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = curr;
   }
}

void printArray(int arr[], int n)
{
   int i;
```

```
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    time_t start, end;
    int n;
    srand(time(0));
    printf("Enter the no of elements \n");
    scanf("%d", &n);

    int i;
    for ( i = 0; i < n; i++)
    {
        arr[i] = rand();
    }
    start = time(NULL);
    insetionSort(arr, n);
    end = time(NULL);

    printf("The array is sorted\n");
    // printf("The sorted array is: \n");
    // printArray(arr, n);

    printf("The time taken is %.10f\n", difftime(end, start) / CLOCKS_PER_SEC);
    return 0;
}
```

Array Size	Time (in Sec)
2000	0.002
4000	0.008
6000	0.019
8000	0.033
10000	0.053
12000	0.074
14000	0.100

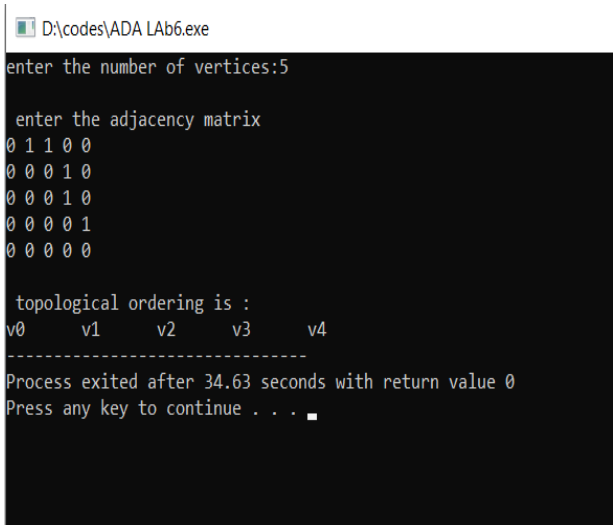


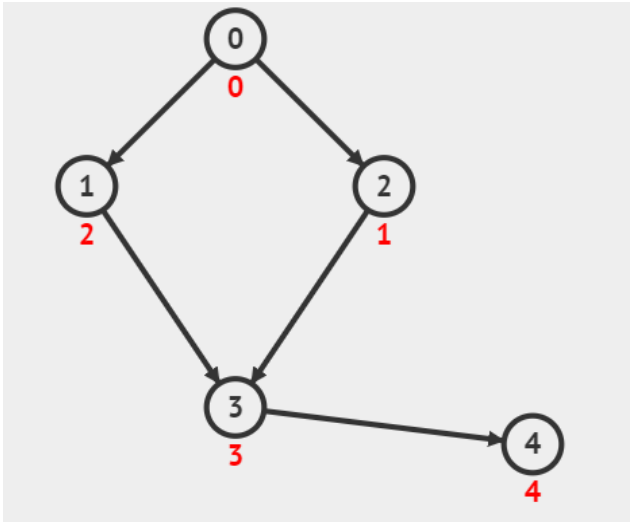
PROGRAM 6: Topological Ordering

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

```
#include<stdio.h>
  int temp[10],k=0;

void topo(int n,int indegree[10],int a[10][10])
  {
  int i,j;

  for(i=1;i<=n;i++)
    {
          if(indegree[i]==0)
           {
           indegree[i]=1;
             temp[++k]=i;
                 for(j=1;j<=n;j++)
                   {
                    if(a[i][j]==1&&indegree[j]!=-1)
                     indegree[j]--;
                   }
                   i=0;
             }
     }
  }

 void main()
  {
  int i,j,n,indegree[10],a[10][10];
  printf("enter the number of vertices:");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
  indegree[i]=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]);
    if(a[i][j]==1)
    indegree[j]++;
  }

  topo(n,indegree,a);

  if(k!=n)
  printf("topological ordering is not possible\n");

else
  {
    printf("\n topological ordering is :\n");
    for(i=1;i<=k;i++)
    printf("v%d\t",temp[i]);
   }
 }
```



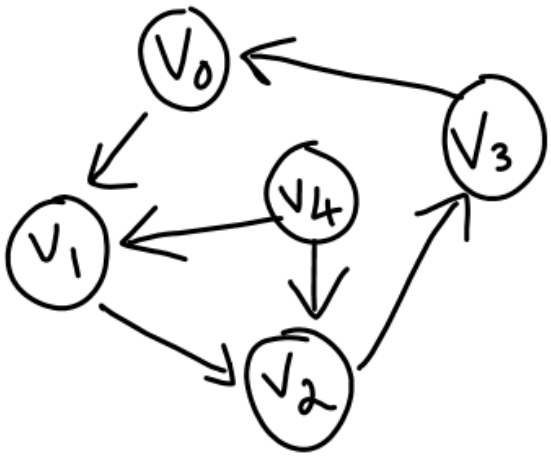


Using DFS Technique

```
#include<stdio.h>
  int res[10],top=0,s[10];
  void dfs(int v,int n,int a[10][10]){
    s[v]=1;
    for(int i=1;i<=n;i++)
    if(s[i]==0&&a[v][i]==1) dfs(i,n,a);
  top++;
  res[top]=v;
  }

void topo(int n,int a[10][10])
  {
  for(int i=1;i<=n;i++){
    s[i]=0;
  }
  top=0;
  for(int i=1;i<=n;i++){
    if(s[i]==0)dfs(i,n,a);
  }


  }

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

```
  for(i=1;i<=n;i++)
  vertices[i]=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]);

  }

  topo(n,a);

  if(top!=n)
  printf("topological ordering is not possible\n");

else
  {
    printf("\n topological ordering is :\n");
    for(int i=n;i>0;i--){
      printf("v%d\t",res[i]);
    }
   }
 }
```

```
enter the number of vertices:5

 enter the adjacency matrix
0 1 1 0 0
0 0 0 1 0
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0

 topological ordering is :
v1      v3      v2      v4      v5
--------------------------------
Process exited after 23.07 seconds with return value 0
Press any key to continue . . .
```



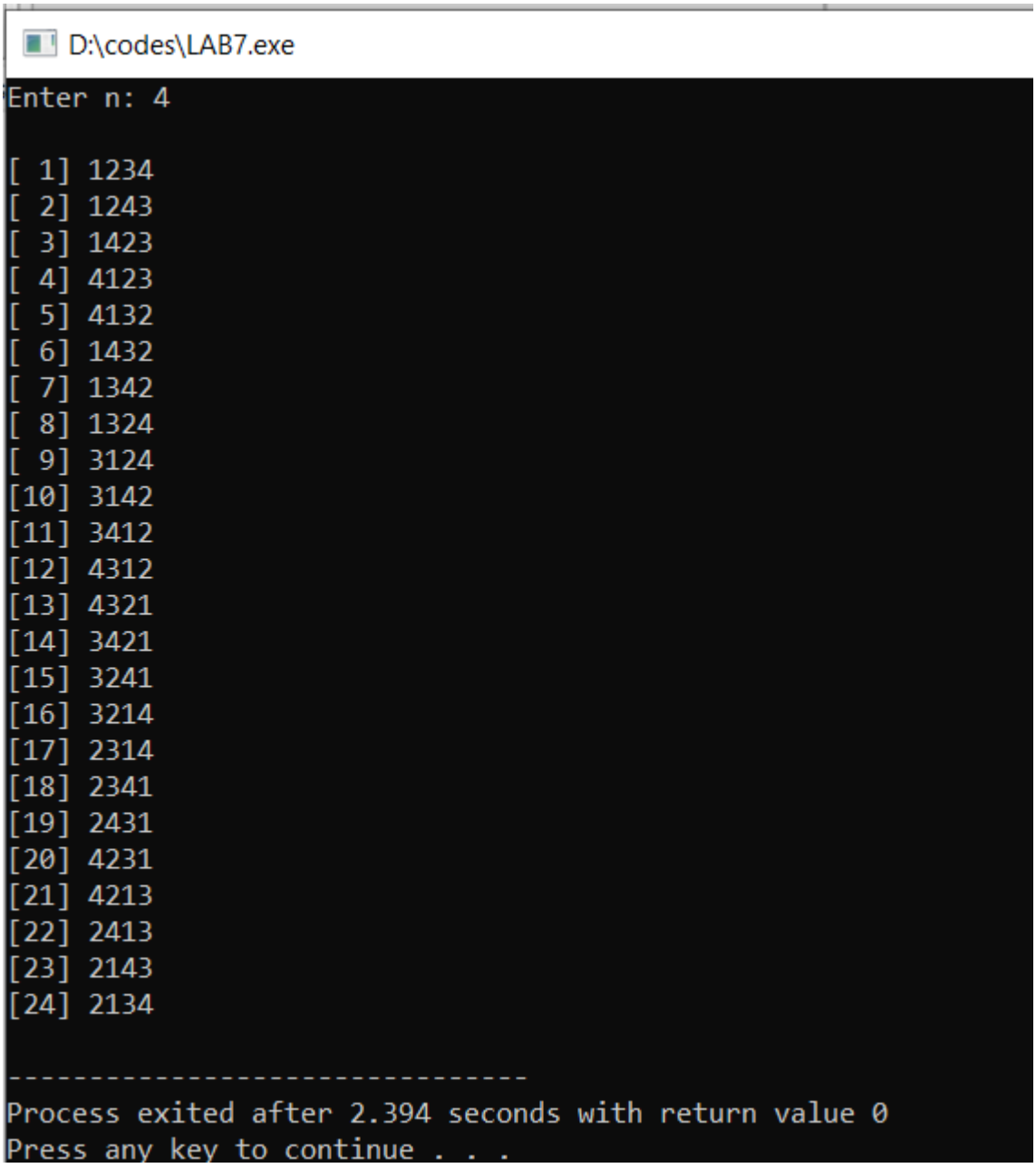
PROGRAM 7: Johnson Trotter

Implement Johnson Trotter algorithm to generate permutations.

```
#include <stdio.h>

int NN, i, count=0;
int p[100], pi[100];
int dir[100];

void PrintPerm()
{
  int i;

  count = count + 1;
  printf( "[%2d] ", count );

  for (i=1; i <= NN; ++i)
    printf( "%d", p[i] );
}



void PrintTrans( int x, int y )
{
  // printf( " (%d %d)", x, y );
  printf( "\n" );
}




void Move( int x, int d )
```

```
{
  int z;
  PrintTrans( pi[x], pi[x]+d );
  z = p[pi[x]+d];
  p[pi[x]] = z;
  p[pi[x]+d] = x;
  pi[z] = pi[x];
  pi[x] = pi[x]+d;
}



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

```
{
  printf( "Enter n: " );
  scanf( "%d", &NN );
  printf( "\n" );
  for (i=1; i<=NN; ++i)
  {
    dir[i] = -1; p[i] = i;
    pi[i] = i;
  }
  Perm ( 1 );
  printf( "\n" );

}
```


D:\codes\LAB7.exe
Enter n: 4
[1] 1234
[2] 1243
[3] 1423
[4] 4123
[5] 4132
[6] 1432
[7] 1342
[8] 1324
[9] 3124
[10] 3142
[11] 3412
[12] 4312
[13] 4321
[14] 3421
[15] 3241
[16] 3214
[17] 2314
[18] 2341
[19] 2431
[20] 4231
[21] 4213
[22] 2413
[23] 2143
[24] 2134

Process exited after 2.394 seconds with return value 0
Press any key to continue . . .

PROGRAM 8: Merge Sort

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

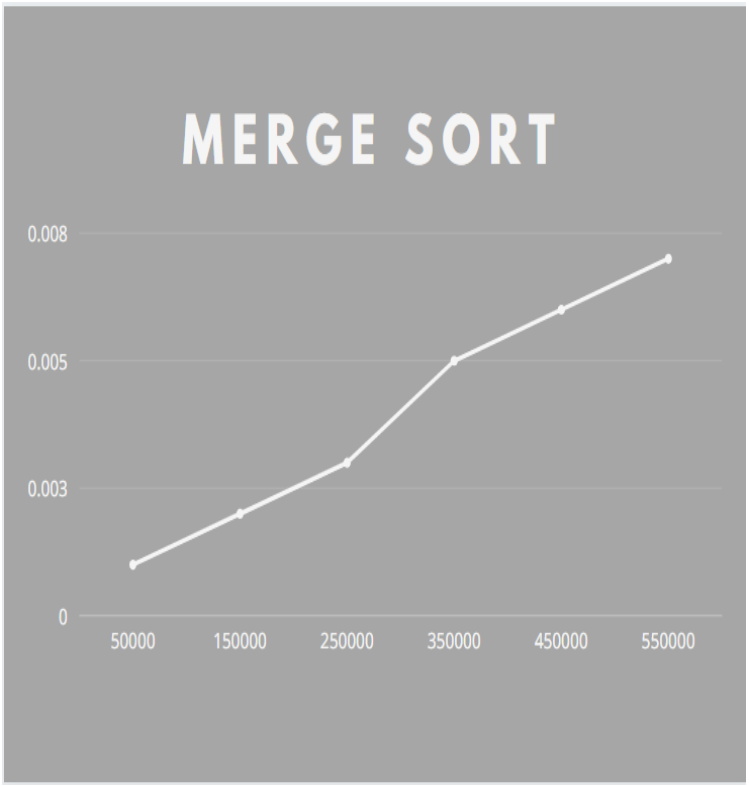
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int arr[1000000];
void merge(int arr[], int p, int q, int r)
{

   int n1 = q - p + 1;
   int n2 = r - q;

   int L[n1], M[n2];
        int i,j,k;
   for ( i = 0; i < n1; i++)
      L[i] = arr[p + i];
   for ( j = 0; j < n2; j++)
      M[j] = arr[q + 1 + j];

   i = 0;
   j = 0;
   k = p;

   while (i < n1 && j < n2)
   {
      if (L[i] <= M[j])
      {
         arr[k] = L[i];
         i++;
      }
      else
      {
```

```
            arr[k] = M[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = M[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    int i,j;
    for( i=0;i<80;i++)
    {
        for( j=0;j<40 ; j++)
        {

        }
    }
    if (l < r)
    {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
```

```
        merge(arr, l, m, r);
    }
}

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

int main()
{
    time_t start, end;
    int n,i;
    srand(time(0));
    printf("Enter the no of elements \n");
    scanf("%d", &n);

    for ( i = 0; i < n; i++)
    {
        arr[i] = rand();
    }

    start = time(NULL);
    mergeSort(arr,0,n-1);
    end = time(NULL);

    printf("The array is sorted\n");
    // printf("The sorted array is: \n");
    // printArray(arr, n);

    printf("The time taken is %.10f\n", difftime(end, start) / CLOCKS_PER_SEC);
    return 0;
}
```

```
C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 8 ( Merge Sort )\mergeSort.exe
Enter the no of elements
50000
The array is sorted
The time taken is 0.0010000000

--------------------------------
Process exited after 3.004 seconds with return value 0
Press any key to continue . . .
```

Merge Sort Array size	Time in sec
50000	0.001
150000	0.002
250000	0.003
350000	0.005
450000	0.006
550000	0.007

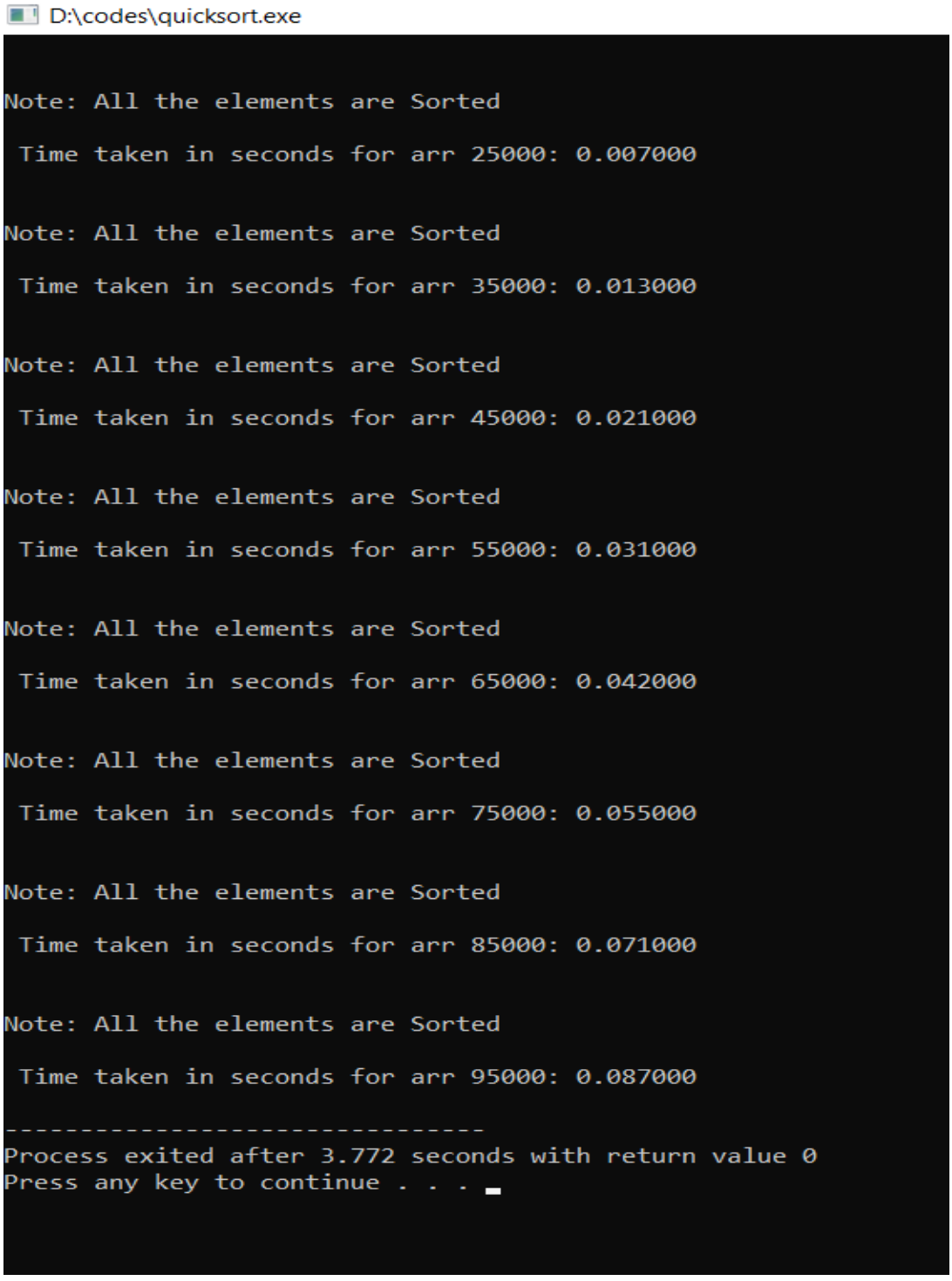


PROGRAM 9: Quick Sort

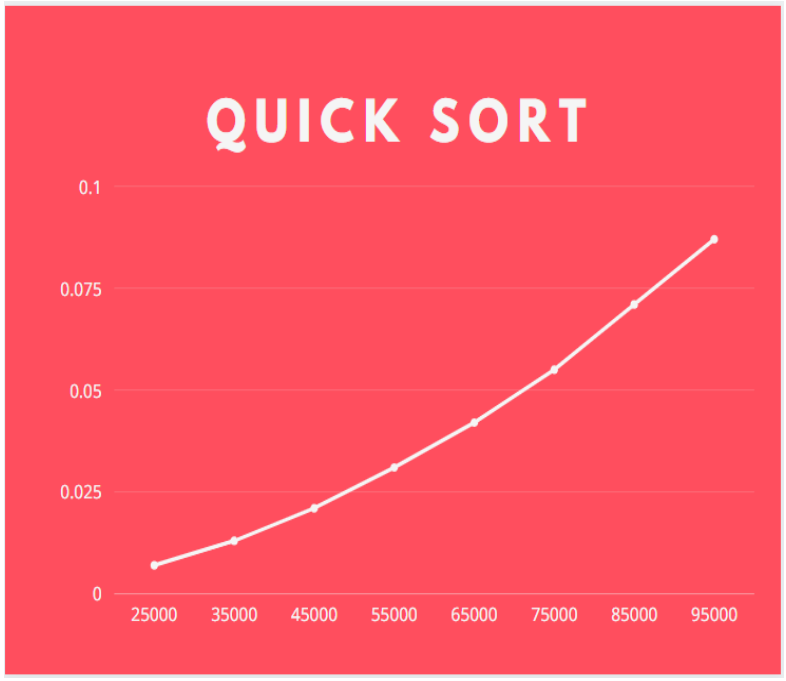
Sort a given set of N integer elements using Quick Sort technique and compute its time taken

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

void swap(int* a, int* b)
{
   int t = *a;
   *a = *b;
   *b = t;
}
int partition (int arr[], int low, int high)
{
   int pivot = arr[high];
   int i = (low - 1);

   for (int j = low; j <= high- 1; j++)
   {
      if (arr[j] < pivot)
      {
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i + 1], &arr[high]);
   return (i + 1);
}
void quickSort(int arr[], int low, int high)
{
   if (low < high)
   {
      int pi = partition(arr, low, high);
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
```

```
    }
}
int main()
{
    int arr[100000],size,i;
    clock_t timereq;
    float cpu_time;

   //printf("Enter size\n");
   //scanf("%d",&size);
  for(size=25000;size<=100000;size=size+10000){
          i=0;

   srand(time(NULL));
          for(int i=0;i<size;i++){
          arr[i]=rand()%100;
          // printf("%d\t",arr[i]);
          }
          printf("\n");

   timereq= clock();
    quickSort(arr, 0, size-1);
    timereq=clock()-timereq;
    cpu_time=((float)(timereq))/CLK_TCK;
    printf("Check: ");
    for(i=0;arr[i]<=arr[i+1] && i<size ;i++);
    if(size-1 == i)
          printf("All the elements are Sorted\n");
    else
          printf("Elements are NOT  Sorted\n");

    printf("Time taken to sort an arr[%d] is %f seconds\n ",size,cpu_time);

          }
    return 0;
}
```


D:\codes\quicksort.exe
Note: All the elements are Sorted
Time taken in seconds for arr 25000: 0.007000
Note: All the elements are Sorted
Time taken in seconds for arr 35000: 0.013000
Note: All the elements are Sorted
Time taken in seconds for arr 45000: 0.021000
Note: All the elements are Sorted
Time taken in seconds for arr 55000: 0.031000
Note: All the elements are Sorted
Time taken in seconds for arr 65000: 0.042000
Note: All the elements are Sorted
Time taken in seconds for arr 75000: 0.055000
Note: All the elements are Sorted
Time taken in seconds for arr 85000: 0.071000
Note: All the elements are Sorted
Time taken in seconds for arr 95000: 0.087000
Process exited after 3.772 seconds with return value 0
Press any key to continue . . .

25000	0.007
35000	0.013
45000	0.021
55000	0.031
65000	0.042
75000	0.055
85000	0.071
95000	0.087



PROGRAM 10: Heap Sort

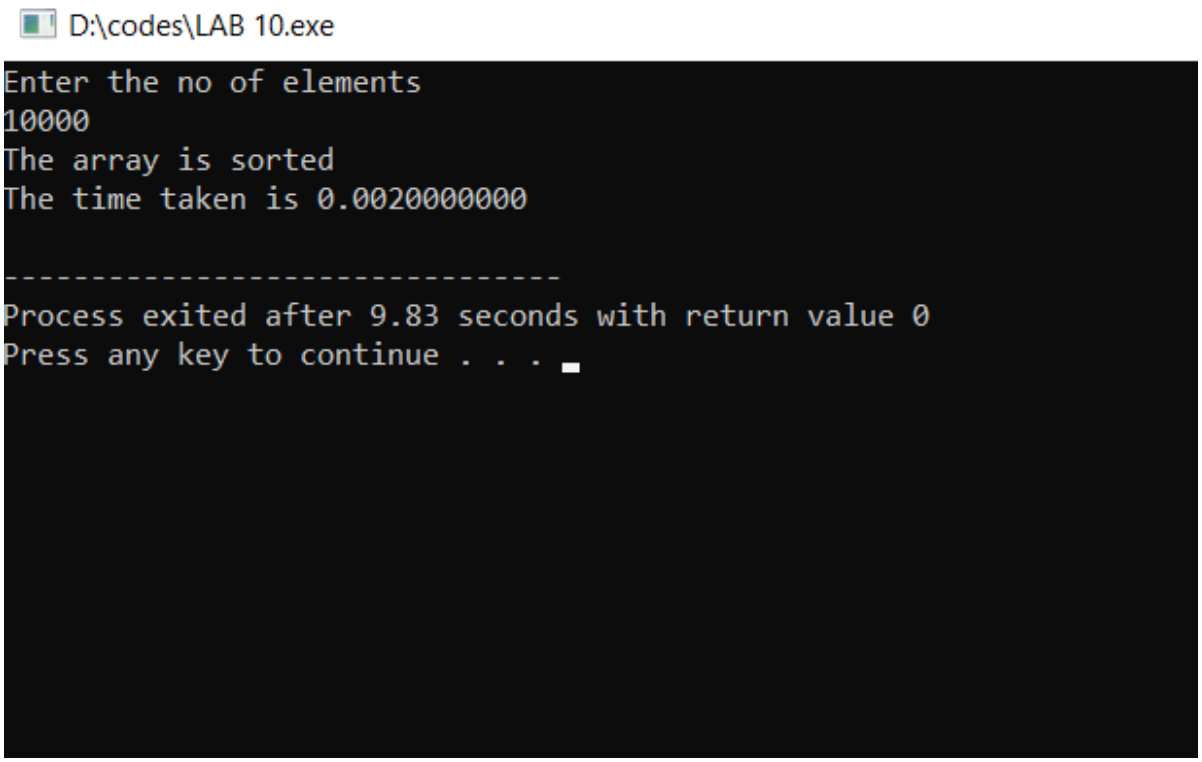
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int arr[1000000];

int temp;

void maxheap(int arr[], int size, int i)
{
   int j,k;

   for (k = 0; k < 180; k++)
   {
      for (j = 0; j < 40; j++)
      {

      }
   }

   int largest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;
```

```
    if (left < size && arr[left] > arr[largest])
        largest = left;

    if (right < size && arr[right] > arr[largest])
        largest = right;

    if (largest != i)
    {
        temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        maxheap(arr, size, largest);
    }
}

void heapSort(int arr[], int size)
{

    int i;
    for (i = size / 2 - 1; i >= 0; i--)
        maxheap(arr, size, i);
    for (i = size - 1; i >= 0; i--)
    {
        temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        maxheap(arr, i, 0);
```

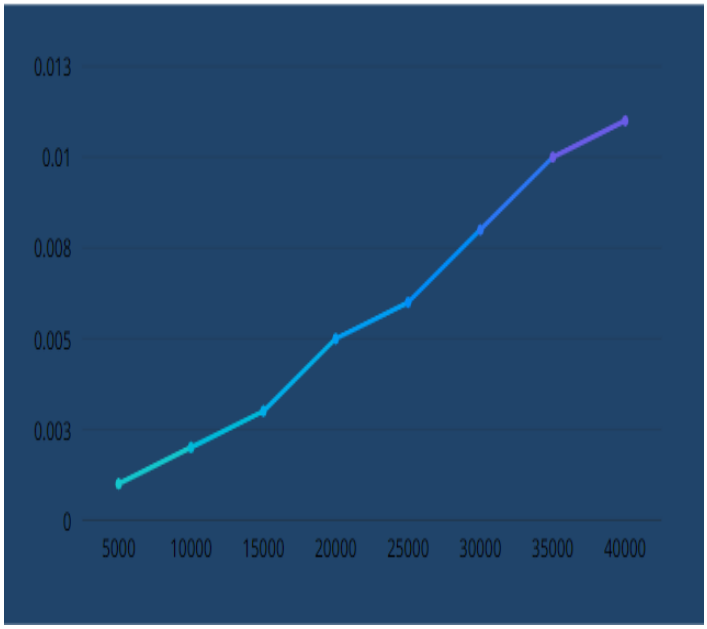
```
    }
}

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

int main()
{
    time_t start, end;
    int n;
    srand(time(0));
    printf("Enter the no of elements \n");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        arr[i] = rand();
    }

    start = time(NULL);
    heapSort(arr, n);
    end = time(NULL);
```

```
    printf("The array is sorted\n");
    // printf("The sorted array is: \n");
    // printArray(arr, n);

    printf("The time taken is %.10f\n", difftime(end, start) /
CLOCKS_PER_SEC);
    return 0;
}
```



HEAP SORT

1BM19CS127



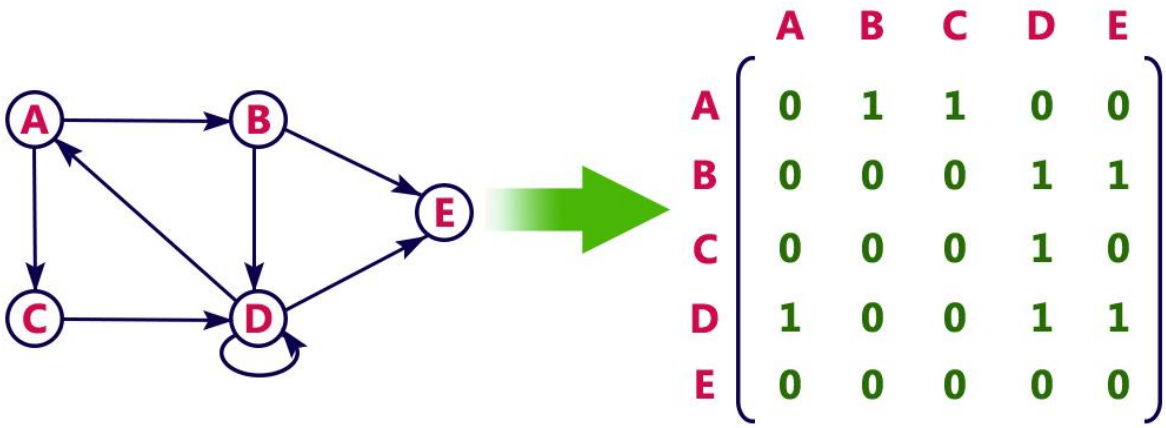
n	t
5000	0.001
10000	0.002
15000	0.003
20000	0.005
25000	0.006
30000	0.008
35000	0.01
40000	0.011

PROGRAM 11: Warshall's algorithm

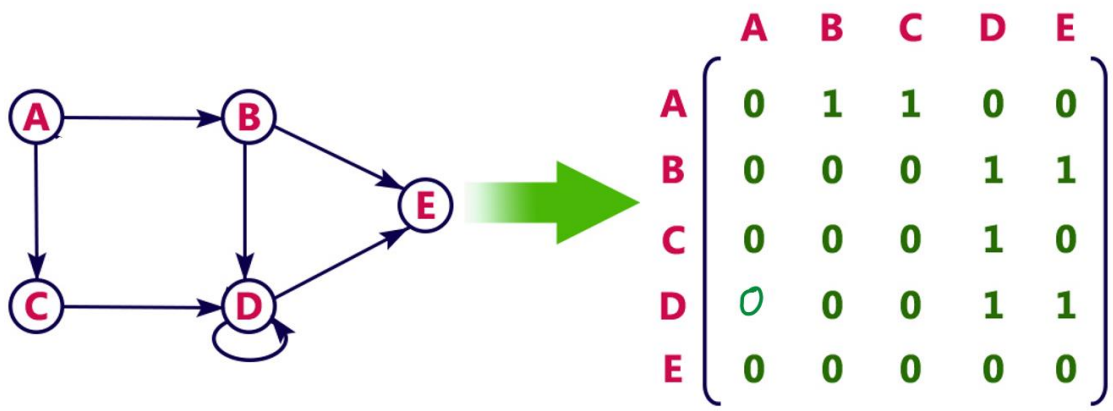
Implement Warshall's algorithm using dynamic programming.

```
#include<stdio.h>
#include<conio.h>
int n,a[10][10],p[10][10];

void warshall(int n,int a[10][10],int p[10][10])
{
int i,j,k;
for(i=0;i<n;i++)
     for(j=0;j<n;j++)
          p[i][j]=a[i][j];

   for(k=0;k<n;k++)
      for(i=0;i<n;i++)
          for(j=0;j<n;j++)
            if((p[i][j]==0) && (p[i][k]==1 && p[k][j]==1))
               p[i][j]=1;
}
int main()
{
   int i,j;
   printf("enter the number of vertices\n");
   scanf("%d",&n);
   printf("enter the adjacency matrix\n");
   for(i=0;i<n;i++)
```

```
{
  for(j=0;j<n;j++)
  {
    scanf("%d",&a[i][j]);
  }
 }
 warshall(n,a,p);
 printf("trasitive closure\n");
 for(i=0;i<n;i++)
 {
    for(j=0;j<n;j++)
    {
        printf("%d\t",p[i][j]);
    }
    printf("\n");
 }
 }
```




```
D:\codes\LAB 11.exe
enter the number of vertices
5
enter the adjacency matrix
0 1 1 0 0
0 0 0 1 1
0 0 0 1 0
1 0 0 1 1
0 0 0 0 0
trasitive closure
1       1       1       1       1
1       1       1       1       1
1       1       1       1       1
1       1       1       1       1
0       0       0       0       0

--------------------------------
Process exited after 31.78 seconds with return value 0
Press any key to continue . . .
```




```
D:\codes\LAB 11.exe
enter the number of vertices
5
enter the adjacency matrix
0 1 1 0 0
0 0 0 1 1
0 0 0 1 0
0 0 0 1 1
0 0 0 0 0
trasitive closure
0       1       1       1       1
0       0       0       1       1
0       0       0       1       1
0       0       0       1       1
0       0       0       0       0

--------------------------------
Process exited after 68.15 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 12: Knapsack problem

Implement 0/1 Knapsack problem using dynamic programming.

Program:

```
#include<stdio.h>
#include<conio.h>
void knapsack();
int max(int,int);
int i,j,n,m,p[10],w[10],v[10][10];
void main()
{
 clrscr();
 printf("\nenter the no. of items:\t");
 scanf("%d",&n);
 printf("\nenter the weight of the each item:\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&w[i]);
 }
 printf("\nenter the profit of each item:\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&p[i]);
 }
 printf("\nenter the knapsack's capacity:\t");
 scanf("%d",&m);
 knapsack();
 getch();
}

void knapsack()
{
```

```
int x[10];
for(i=0;i<=n;i++)
{
 for(j=0;j<=m;j++)
 {
  if(i==0||j==0)
  {
   v[i][j]=0;
  }
  else if(j-w[i]<0)
  {
   v[i][j]=v[i-1][j];
  }
  else
  {
   v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);
  }
 }
}
printf("\nthe output is:\n");
for(i=0;i<=n;i++)
{
 for(j=0;j<=m;j++)
 {
  printf("%d\t",v[i][j]);
 }
 printf("\n\n");
}
printf("\nthe optimal solution is %d",v[n][m]);
printf("\nthe solution vector is:\n");
for(i=n;i>=1;i--)
{
 if(v[i][m]!=v[i-1][m])
 {
  x[i]=1;
  m=m-w[i];
```

```
  }
  else
  {
   x[i]=0;
  }
 }
 for(i=1;i<=n;i++)
 {
  printf("%d\t",x[i]);
 }
}

int max(int x,int y)
{
 if(x>y)
 {
  return x;
 }
 else
 {
  return y;
 }
}
```

Output:

Enter the no. of items: 4

Enter the weight of each item:
2 1 3 2

Enter the profit of the each item:
12 10 20 15

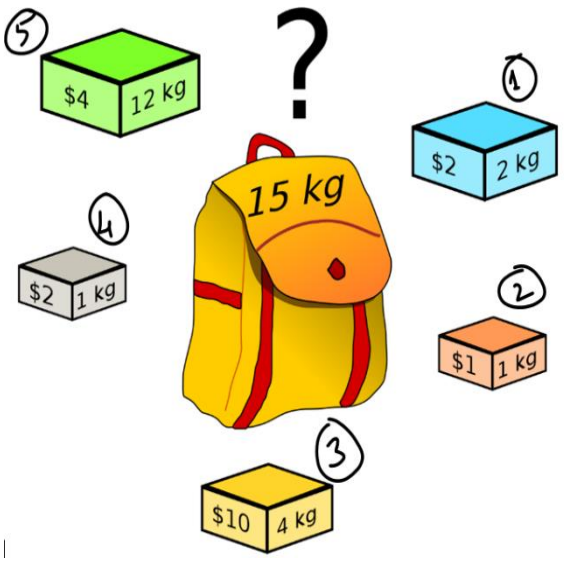
Enter the Knapsack's capacity: 5

The output is:
0 0 0 0 0 0
0 0 12 12 12 12
0 10 12 22 22 22
0 10 12 22 30 32
0 10 15 25 30 37
The optimal solution is: 37

The solution vector is:
1 1 0 1

D:\codes\LAB 12.exe

```
enter the no. of items: 4

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

enter the profit of each item:
12  10  20  15

enter the knapsack's capacity:  5

the output is:
0       0       0       0       0       0

0       0       12      12      12      12

0       10      12      22      22      22

0       10      12      22      30      32

0       10      15      25      30      37


the optimal solution is 37
the solution vector is:
1       1       0       1
--------------------------------
Process exited after 27.09 seconds with return value 0
Press any key to continue . . .
```



D:\codes\LAB 12.exe

```
2  1  10  2  4

enter the knapsack's capacity:  15

the output is:
0       0       0       0       0       0       0       0       0       0       0       0       2       2       2
        2

0       0       2       2       2       2       2       2       2       2       0       1       2       3       3
        3

0       1       2       3       3       3       3       3       3       3       0       1       2       3       10
        11

0       1       2       3       10      11      12      13      13      13      0       2       3       4       10
        12

0       2       3       4       10      12      13      14      15      15      0       2       3       4       10
        12

0       2       3       4       10      12      13      14      15      15      0       2       4       6       10
        12


the optimal solution is 12
the solution vector is:
1       0       1       0       0
--------------------------------
Process exited after 205.4 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 13: Floyd's algorithm

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

Program:

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n;
void floyds();
int min(int,int);
void main()
{
 int i,j;
 clrscr();
 printf("\nenter the no. of vertices:\t");
 scanf("%d",&n);
 printf("\nenter the cost matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&a[i][j]);
  }
 }
 floyds();
 getch();
}

void floyds()
{
 int i,j,k;
 for(k=1;k<=n;k++)
 {
  for(i=1;i<=n;i++)
```

```
  {
   for(j=1;j<=n;j++)
   {
    a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
   }
  }
 }
 printf("\nall pair shortest path matrix is:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   printf("%d\t",a[i][j]);
  }
  printf("\n\n");
 }
}

int min(int x,int y)
{
 if(x<y)
 {
  return x;
 }
 else
 {
  return y;
 }
}
```

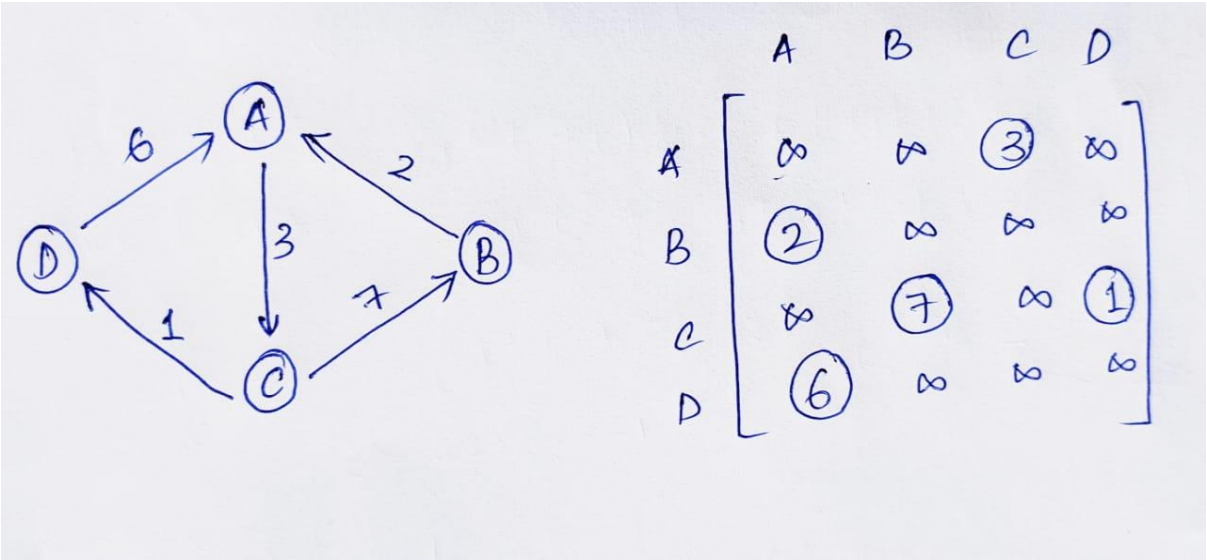
Output

```
Enter the no. of vertices:  4

Enter the cost matrix:
9999   9999       3   9999
    2   9999   9999   9999
9999       7   9999       1
```

6 9999 9999 9999

All pair shortest path matrix is:

10 10 3 4
2 12 5 6
7 7 10 1
6 16 9 10




```
D:\codes\LAB 13.exe
enter the no. of vertices:      4

enter the cost matrix:
999  999  3  999
2    999  999  999
999  7   999   1
6   999  999  999

all pair shortest path matrix is:
10      10      3       4

2       12      5       6

7       7       10      1

6       16      9       10


--------------------------------
Process exited after 56.49 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 14: Prim's algorithm.

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

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void prims();
int c[10][10],n;

int main()
{
 int i,j;
 printf("\n Enter the no. of vertices:  ");
 scanf("%d",&n);
 printf("\n Enter the cost matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&c[i][j]);
  }
 }
 prims();
}

void prims()
{
 int i,j,u,v,min;
 int ne=0,mincost=0;
 int elec[10];
 for(i=1;i<=n;i++)
 {
  elec[i]=0;
```

```
}
elec[1]=1;
while(ne!=n-1)
{
 min=9999;
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   if(elec[i]==1)
   {
    if(c[i][j]<min)
    {
     min=c[i][j];
     u=i;
     v=j;
    }
   }
  }
 }
 if(elec[v]!=1)
 {
  printf("\n\t%c -----> %c = %d\n",u+65,v+65,min);
  elec[v]=1;
  ne=ne+1;
  mincost=mincost+min;
 }
 c[u][v]=c[v][u]=9999;
}
printf("\n\n\t Minimum Cost is =%d\n",mincost);
}
```

==========Output==========

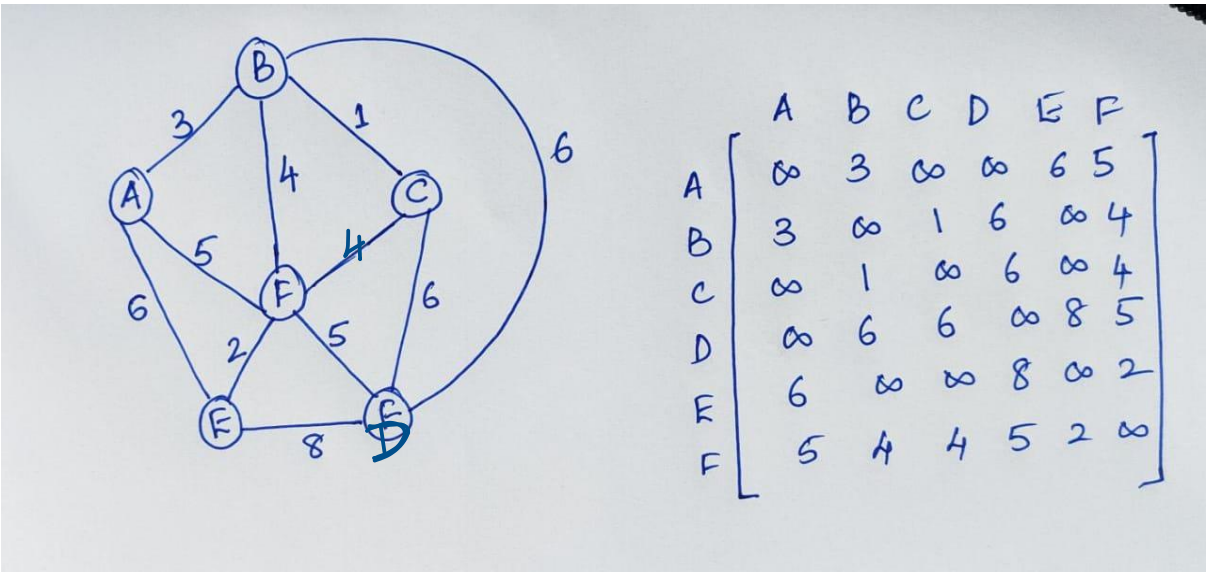
Enter the no. of vertices: 6

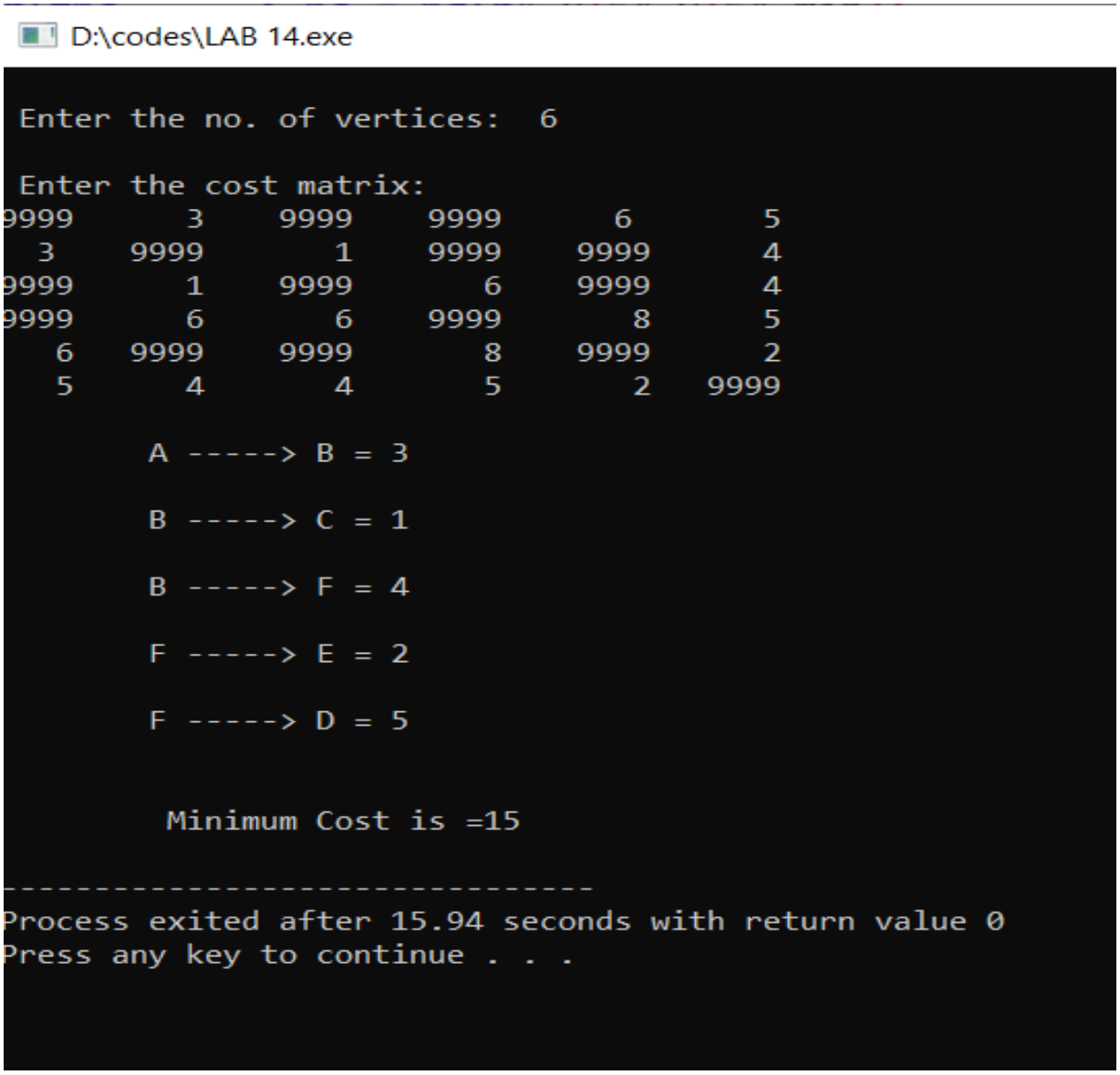
Enter the cost matrix:

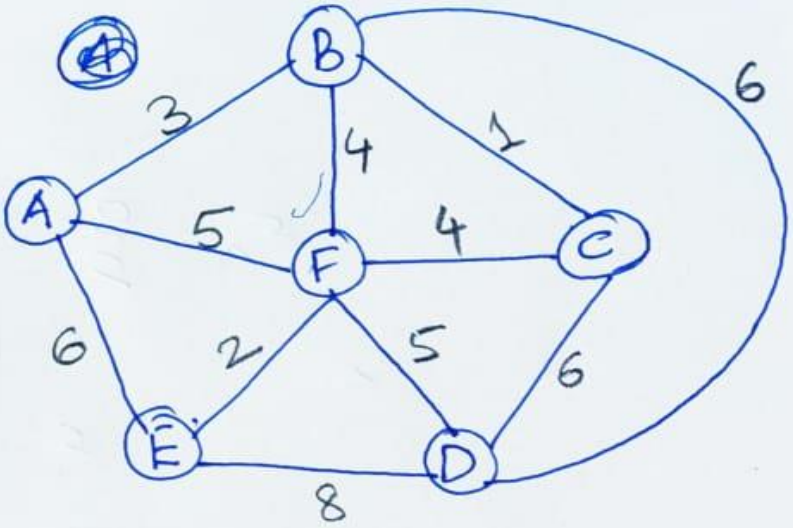
9999	3	9999	9999	6	5
3	9999	1	6	9999	4
9999	1	9999	6	9999	4
9999	6	6	9999	8	5
6	9999	9999	8	9999	2
5	4	4	5	2	9999

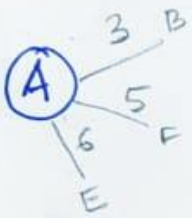
B----------> C = 1
E----------> F = 2
A----------> B = 3
B----------> F = 4
F----------> D = 5

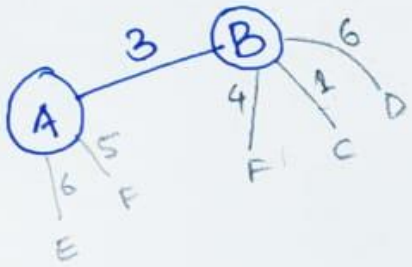
Minimum cost = 15

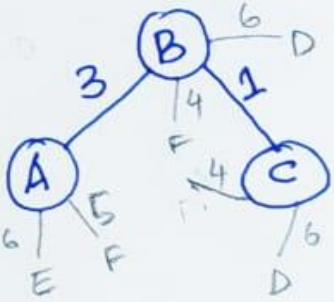


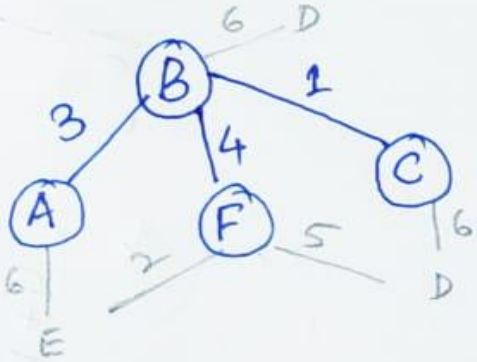

D:\codes\LAB 14.exe
Enter the no. of vertices: 6
Enter the cost matrix:
9999 3 9999 9999 6 5
3 9999 1 9999 9999 4
9999 1 9999 6 9999 4
9999 6 6 9999 8 5
6 9999 9999 8 9999 2
5 4 4 5 2 9999
A -----> B = 3
B -----> C = 1
B -----> F = 4
F -----> E = 2
F -----> D = 5
Minimum Cost is =15
Process exited after 15.94 seconds with return value 0
Press any key to continue . . .

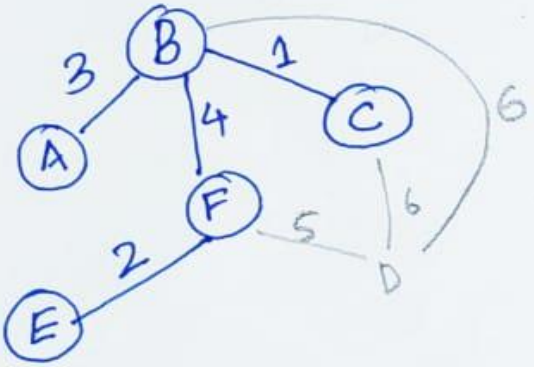


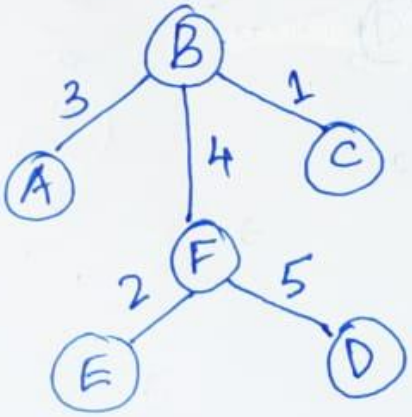












B ↔ C = 1

E ↔ F = ~~6~~ 2

A ↔ B = 3

B ↔ F = 4

F ↔ D = 5

15

PROGRAM 15: Kruskals algorithm

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

```
#include<stdio.h>
#include<conio.h>
void kruskals();
int c[10][10];
int n;

int main()
{
 int i,j;
 printf("\nEnter the no. of vertices: ");
 scanf("%d",&n);
 printf("\nEnter the cost matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&c[i][j]);
  }
 }
 kruskals();
}

void kruskals()
{
 int i,j,u,v,a,b,min;
 int ne=0,mincost=0;
 int parent[10];

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

```
 parent[i]=0;
}
while(ne!=n-1)
{
 min=9999;
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
    if(c[i][j]<min)
    {
     min=c[i][j];
     u=a=i;
     v=b=j;
    }
  }
 }
 while(parent[u]!=0)
 {
  u=parent[u];
 }
 while(parent[v]!=0)
 {
  v=parent[v];
 }
 if(u!=v)
 {
  printf("\n\t%C <---> %C = %d\n",a+65,b+65,min);
  parent[v]=u;
  ne=ne+1;
  mincost=mincost+min;
 }
 c[a][b]=c[b][a]=9999;
}
printf("\n Minimum cost is =%d",mincost);
}
```

==========Output==========

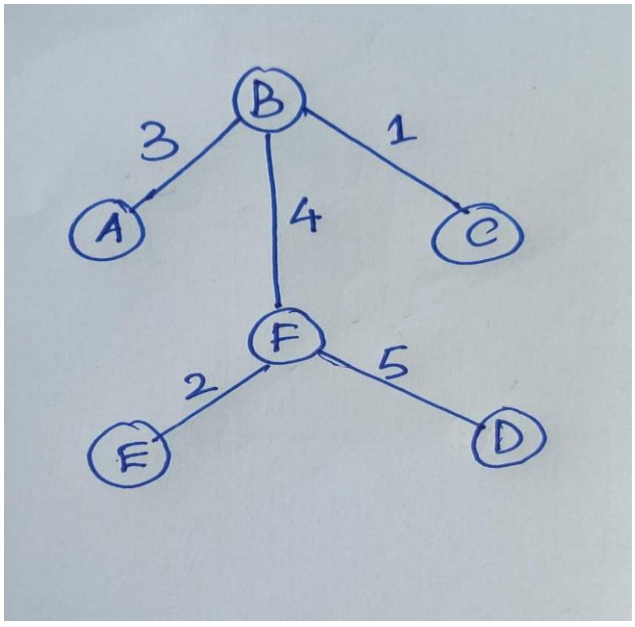
Enter the no. of vertices: 6

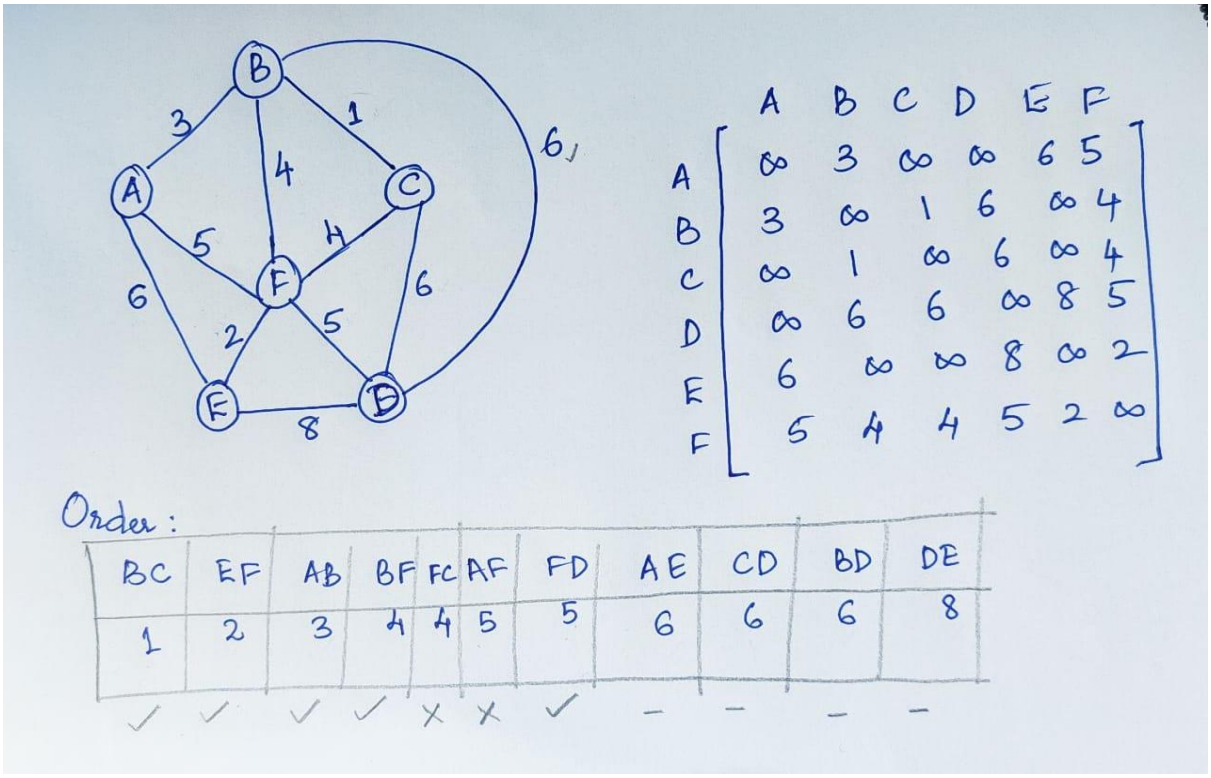
Enter the cost matrix:

9999	3	9999	9999	6	5
3	9999	1	9999	9999	4
9999	1	9999	6	9999	4
9999	6	6	9999	8	5
6	9999	9999	8	9999	2
5	4	4	5	2	9999

B----------> C = 1
E----------> F = 2
A----------> B = 3
B----------> F = 4
F----------> D = 5

Minimum cost = 15






```
D:\codes\LAB 15.exe

Enter the no. of vertices: 6

Enter the cost matrix:
9999       3    9999    9999       6       5
   3    9999       1    9999    9999       4
9999       1    9999       6    9999       4
9999       6       6    9999       8       5
   6    9999    9999       8    9999       2
   5       4       4       5       2    9999

         B <---> C = 1

         E <---> F = 2

         A <---> B = 3

         B <---> F = 4

         D <---> F = 5

 Minimum cost is =15
--------------------------------
Process exited after 21.38 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 16: Dijkstra's algorithm

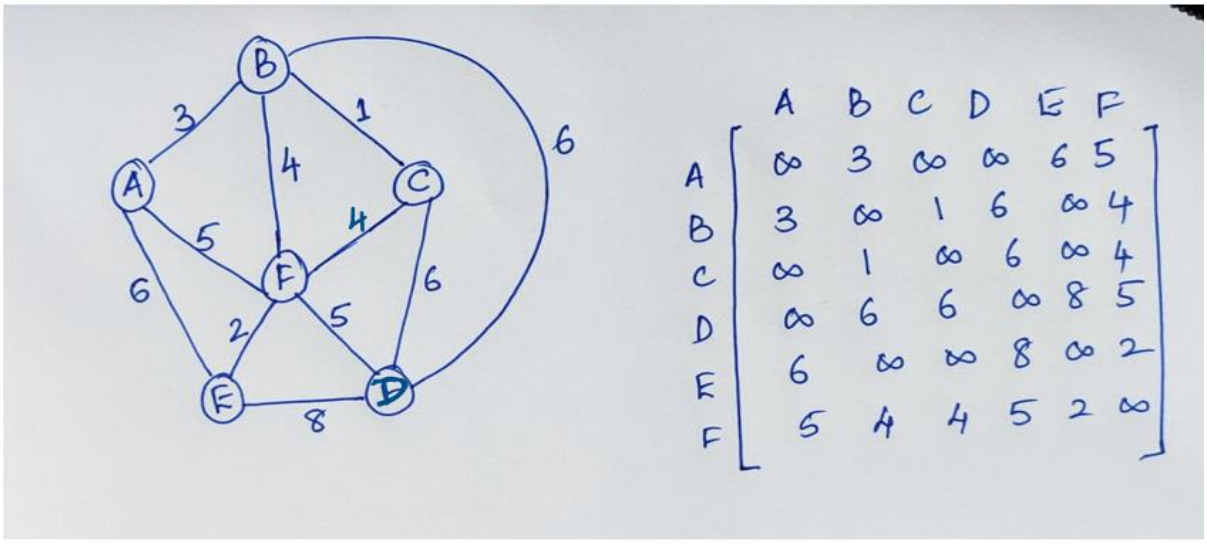
From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Program:

```
#include <stdio.h>
int minDistance(int dist[], int sptSet[],int V)
{
   int min = 999, min_index;
  int v;
   for ( v = 0; v < V; v++)
      if (sptSet[v] == 0 && dist[v] <= min)
         min = dist[v], min_index = v;

   return min_index;
}



int printSolution(int src,int dist[],int V )
{ int i;
   printf("\n\t Vertex \t\t Distance from Source\n\n");
   for (i = 0; i < V; i++)
      printf("\t%c ----> %c \t\t\t %d\n",src+65, i+65, dist[i]);
}



void dijkstra(int graph[10][10], int src,int V )
```

```
{
   int dist[V];
 int i,count,u,v;
   int sptSet[V];
   for ( i = 0; i < V; i++)
      dist[i] = 999, sptSet[i] = 0;

   dist[src] = 0;

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

      u = minDistance(dist, sptSet,V);
      sptSet[u] = 1;

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

         if (!sptSet[v] && graph[u][v] && dist[u] != 999
            && dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
   }

   printSolution(src,dist,V);
}

int main()
{
   int i,j,V;
```

```
    int graph[10][10];
    printf("Enter number of vertices\n");
    scanf("%d",&V);
    printf("Enter adjacency matrix\n");
    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
    scanf("%d",&graph[i][j]);
    }
  for(i=0;i<V;i++){
        dijkstra(graph,i,V );
  }
    // dijkstra(graph, 0,V );

    return 0;
}
```



	A	B	C	D	E	F
A	∞	3	∞	∞	6	5
B	3	∞	1	6	∞	4
C	∞	1	∞	6	∞	4
D	∞	6	6	∞	8	5
E	6	∞	∞	8	∞	2
F	5	4	4	5	2	∞

Dijkstra's Algorithm

(Single Source Shortest Path)

* Relaxation:

if $d(u) + c(u,v) < d(v)$

$d(v) = d(u) + c(u,v)$

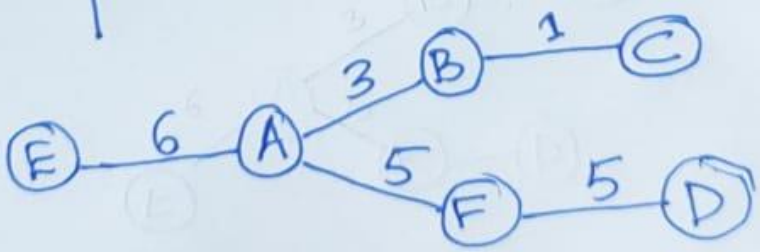
where $u \rightarrow$ source

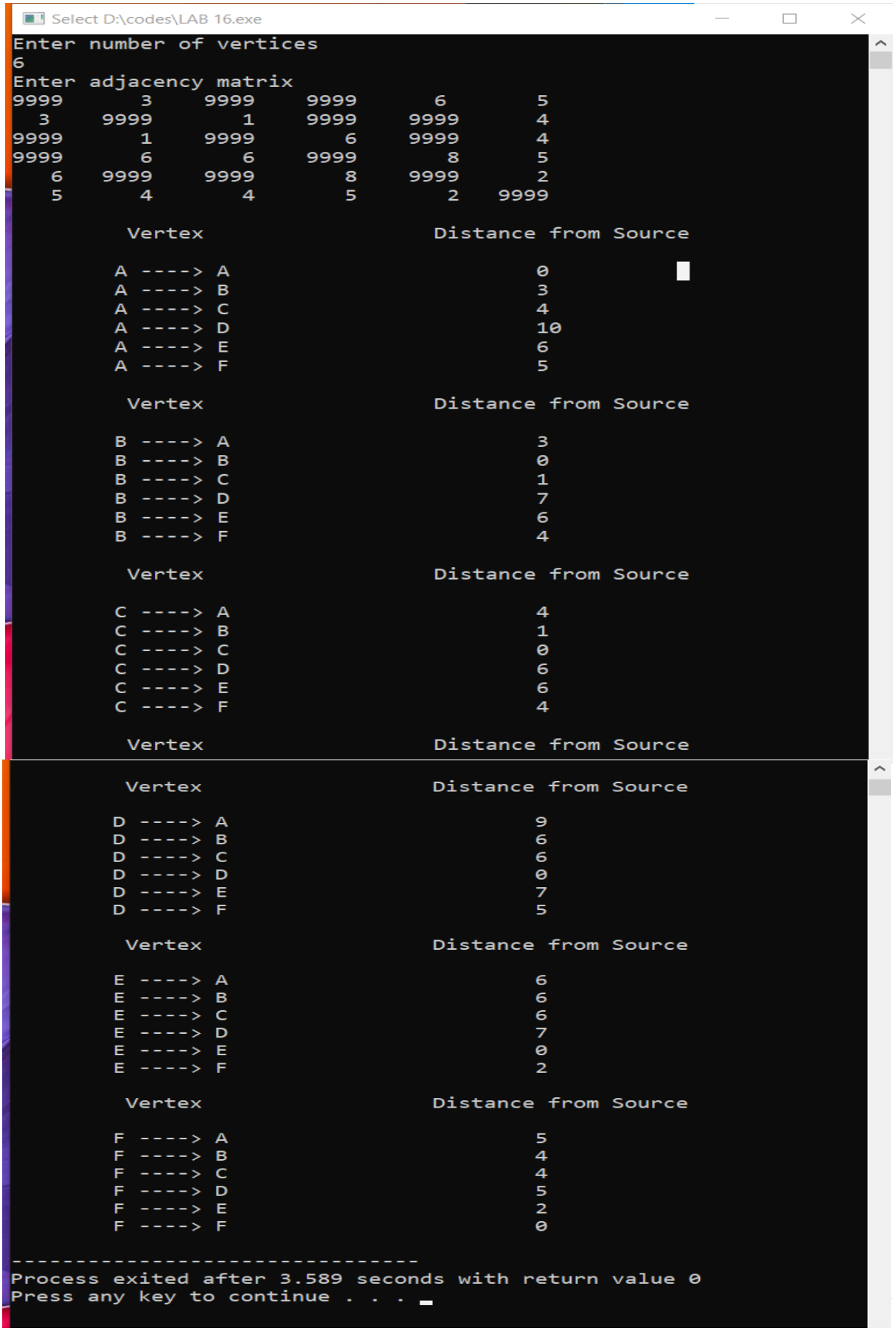
$v \rightarrow$ destination.

let Source = A

A Source	Destination B	C	D	E	F
(0)	∞	∞	∞	∞	∞
A	(3)	∞	∞	6	5
A,B	(3)	(4)	∞	6	5
A,B,C	(3)	(4)	10	6	(5)
A,B,C,F	(3)	(4)	10	(6)	(5)
A,B,C,F,E	(3)	(4)	(10)	(6)	(5)
A,B,C,F,E,D	(3)	(4)	(10)	(6)	(5)

Optimal




Select D:\codes\LAB 16.exe
Enter number of vertices
6
Enter adjacency matrix
9999 3 9999 9999 6 5
3 9999 1 9999 9999 4
9999 1 9999 6 9999 4
9999 6 6 9999 8 5
6 9999 9999 8 9999 2
5 4 4 5 2 9999
Vertex Distance from Source
A ----> A 0
A ----> B 3
A ----> C 4
A ----> D 10
A ----> E 6
A ----> F 5
Vertex Distance from Source
B ----> A 3
B ----> B 0
B ----> C 1
B ----> D 7
B ----> E 6
B ----> F 4
Vertex Distance from Source
C ----> A 4
C ----> B 1
C ----> C 0
C ----> D 6
C ----> E 6
C ----> F 4
Vertex Distance from Source
Vertex Distance from Source
D ----> A 9
D ----> B 6
D ----> C 6
D ----> D 0
D ----> E 7
D ----> F 5
Vertex Distance from Source
E ----> A 6
E ----> B 6
E ----> C 6
E ----> D 7
E ----> E 0
E ----> F 2
Vertex Distance from Source
F ----> A 5
F ----> B 4
F ----> C 4
F ----> D 5
F ----> E 2
F ----> F 0
Process exited after 3.589 seconds with return value 0
Press any key to continue . . .

PROGRAM 17: "Sum of Subsets" problem

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.

Program:

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

static int total_nodes;

void printSubset(int A[], int size)
{
   printf("{");
   for(int i = 0; i < size; i++)
   {
      printf(" %d ", A[i]);
   }

   printf(" }\n");

}

void subset_sum(int s[], int t[],  int s_size, int t_size, int sum, int ite, int const target_sum)
{
```

```
total_nodes++;

if( target_sum == sum )
{
   printSubset(t, t_size);

   if( ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum )
   {
      subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
   }
   return;

}
else
{
   if( ite < s_size && sum + s[ite] <= target_sum )
   {
      for( int i = ite; i < s_size; i++ )
      {
         t[t_size] = s[i];

         if( sum + s[i] <= target_sum )
         {

            subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
         }
```

```
            }

        }

    }

}
  void bsort(int s[],int size)
  {
     int i,j,temp;
     for (i = 0; i < size-1; i++)
     {
      for (j = 0; j < size-i-1; j++)
          {
             if (s[j] > s[j+1])
               {
                  temp=s[j];
                  s[j]=s[j+1];
                  s[j+1]=temp;
               }
          }
     }
  }

void generateSubsets(int s[], int size, int target_sum)
{
    int *tuplet_vector = (int *)malloc(size * sizeof(int));
```

```
    int total = 0;
int i;
    bsort(s, size);

    for( int i = 0; i < size; i++ )
    {
        total += s[i];
    }

    if( s[0] <= target_sum && total >= target_sum )
    {

        subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);

    }

    free(tuplet_vector);
}

int main()
{ int i,n;
    int sets[10] ;
    int target ;
    printf("Enter number of elements in array\n");
    scanf("%d",&n);
    printf("Enter elements of sets\n");
```

```
    for(i=0;i<n;i++)
    scanf("%d",&sets[i]);
    printf("Enter sum\n");
    scanf("%d",&target);
    generateSubsets(sets,n, target);


}
```

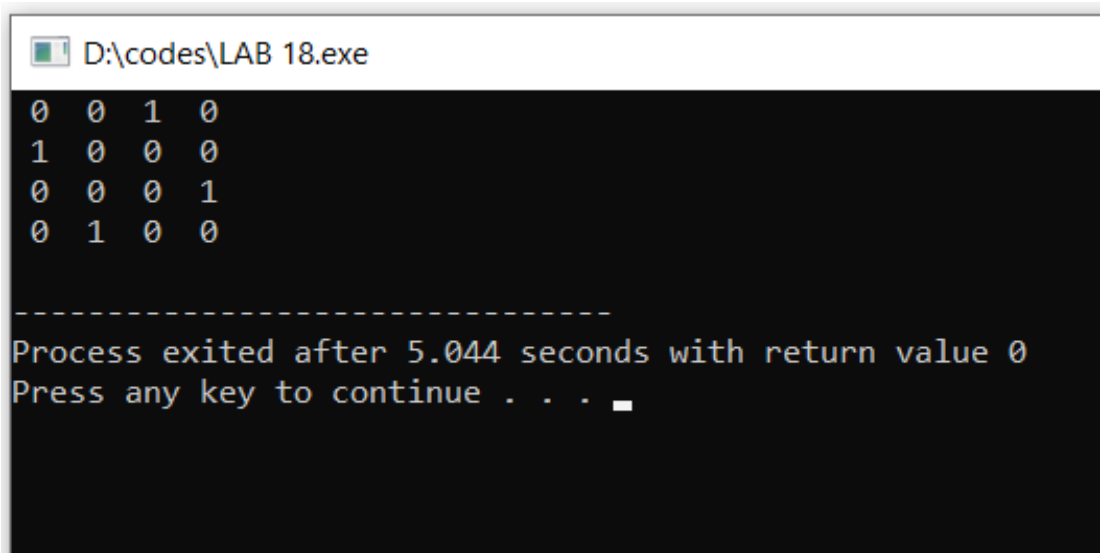
```
D:\codes\LAB 17.exe
Enter number of elements in array
10
Enter elements of sets
1 2 3 4 5 10 20 25 18 17
Enter sum
25
{ 1  2  3  4  5  10  }
{ 1  2  4  18  }
{ 1  2  5  17  }
{ 1  3  4  17  }
{ 1  4  20  }
{ 2  3  20  }
{ 2  5  18  }
{ 3  4  18  }
{ 3  5  17  }
{ 5  20  }
{ 25  }

--------------------------------
Process exited after 6.907 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 18: N-Queens Problem

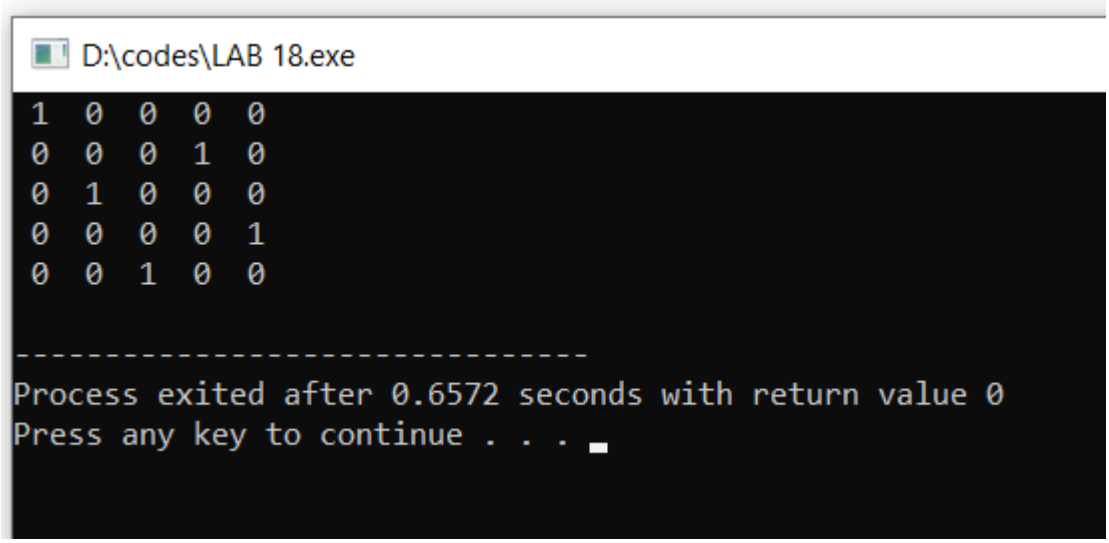
Implement “N-Queens Problem” using Backtracking.

```
#define N 4
#include <stdbool.h>
#include <stdio.h>

void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
}

bool isSafe(int board[N][N], int row, int col)
{
    int i, j;
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}
bool solveNQUtil(int board[N][N], int col)
{
    if (col >= N)
```

```
        return true;
    for (int i = 0; i < N; i++)
    {
        if (isSafe(board, i, col))
        {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1))
                return true;
            board[i][col] = 0;
        }
    }
    return false;
}
bool solveNQ()
{
    int board[N][N] = { { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };

    if (solveNQUtil(board, 0) == false) {
        printf("Solution does not exist");
        return false;
    }

    printSolution(board);
    return true;
}

int main()
{
    solveNQ();
    return 0;
}
```


D:\codes\LAB 18.exe
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

Process exited after 5.044 seconds with return value 0
Press any key to continue . . .


D:\codes\LAB 18.exe
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0

Process exited after 0.6572 seconds with return value 0
Press any key to continue . . .