VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

OMAR ABDULLA SHERIEF (1BM20CS209)

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
May-2022 to July-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 **OMAR ABDULLA SHERIEF (1BM20CS209),** 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 year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge: Rekha GS
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

`

Index Sheet

	and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.	
18	Implement "N-Queens Problem" using Backtracking.	

Course Outcome

CO1	Ability to **analyze** time complexity of Recursive and Non-Recursive algorithms using asymptotic notations.
CO2	Ability to **design** efficient algorithms using various design techniques.
CO3	Ability to **apply** the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Ability to **conduct** practical experiments to solve problems using an appropriate designing method and find time efficiency.

Program Title

Complete executed program

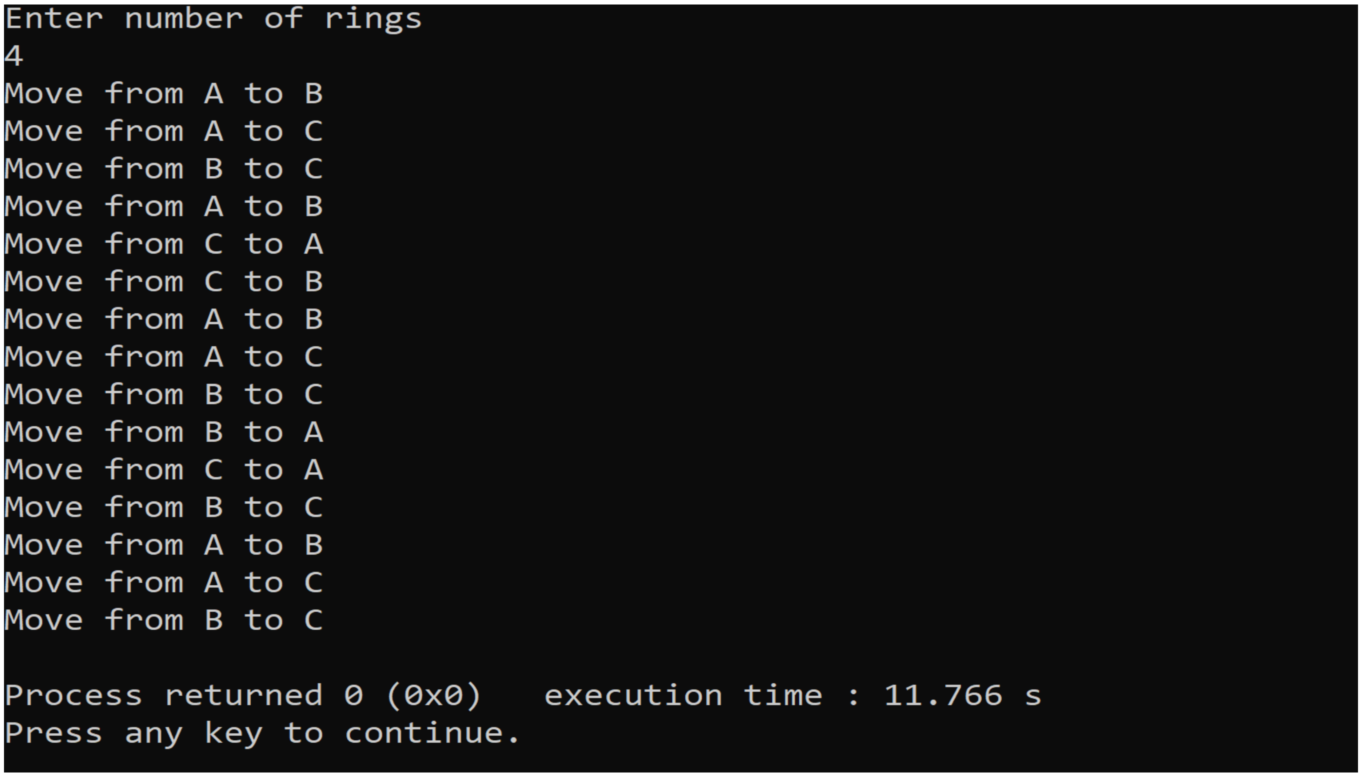
Result Screen shot

Graph (only for Searching and Sorting programs)

LAB PROGRAM 1: Write a recursive program to Solve

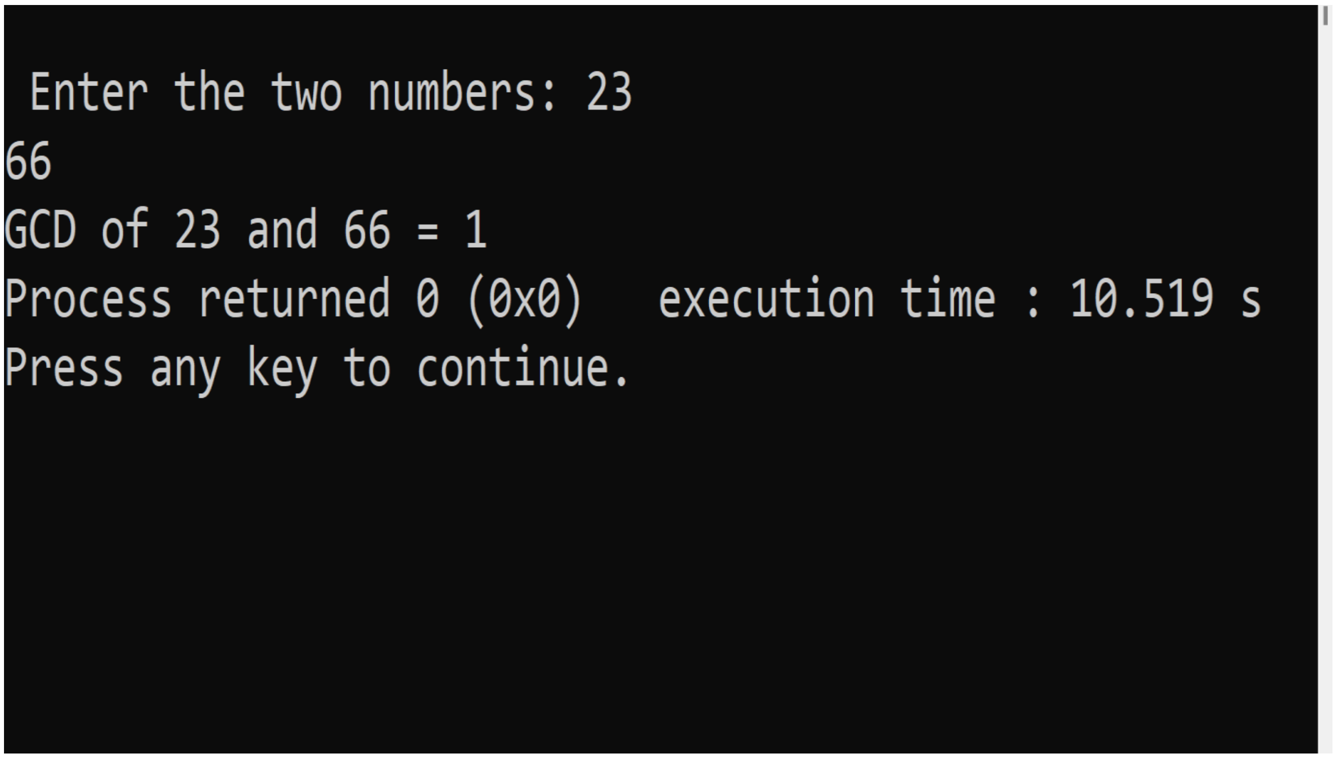
a) Towers-of-Hanoi problem **b)** To find GCD

```
#include(stdio.h)
#include(conio.h)
#include(math.h)
void hanoi(int x, char from, char to, char aux)
{
if(x==1)
printf("Move Disk From %c to %c\n",from,to);
else
{
hanoi(x-1,from,aux,to);
printf("Move Disk From %c to %c\n",from,to);
hanoi(x-1,aux,to,from);
}
}
void main( )
{
int disk;
int moves;
clrscr();
printf("Enter the number of disks you want to play with:");
scanf("%d",&disk);
moves=pow(2,disk)-1;
printf("\nThe No of moves required is=%d \n",moves);
hanoi(disk,'A','C','B');
getch( );
}
```



```
Enter number of rings
4
Move from A to B
Move from A to C
Move from B to C
Move from A to B
Move from C to A
Move from C to B
Move from A to B
Move from A to C
Move from B to C
Move from B to A
Move from C to A
Move from B to C
Move from A to B
Move from A to C
Move from B to C

Process returned 0 (0x0)   execution time : 11.766 s
Press any key to continue.
```

```
#include<stdio.h>
                  int GCD(int x, int y)
                  {
                  if(x%y==0)
                   return y;
                  else
                   return (GCD(y, x%y));
                  }

                  int main()
                  {
                  int num1, num2, result;
                  printf("\n Enter the two numbers: ");
                  scanf("%d %d", &num1, &num2);
                  result = GCD(num1, num2);
                  printf("GCD of %d and %d = %d", num1, num2, result);
                  return 0;
                  }
```



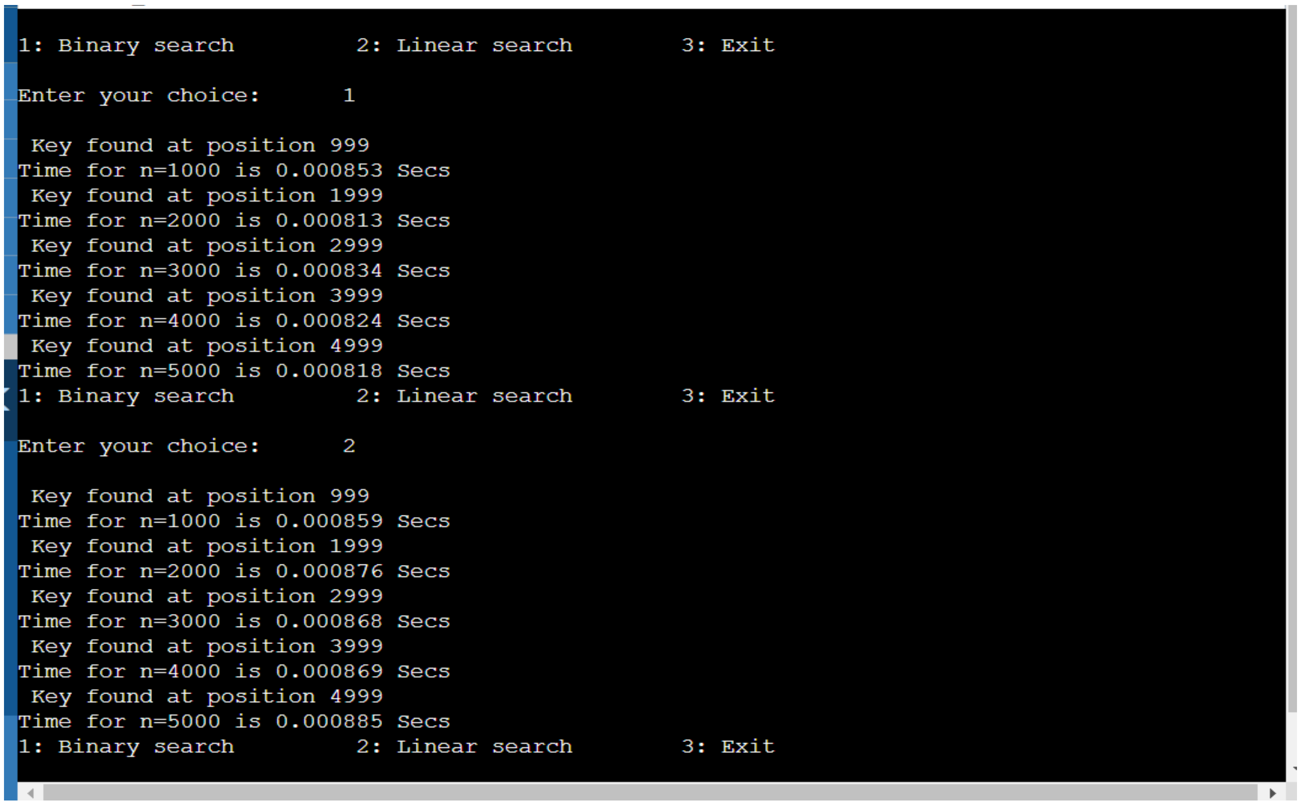
```
 Enter the two numbers: 23
66
GCD of 23 and 66 = 1
Process returned 0 (0x0)   execution time : 10.519 s
Press any key to continue.
```

LAB PROGRAM 2: 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.

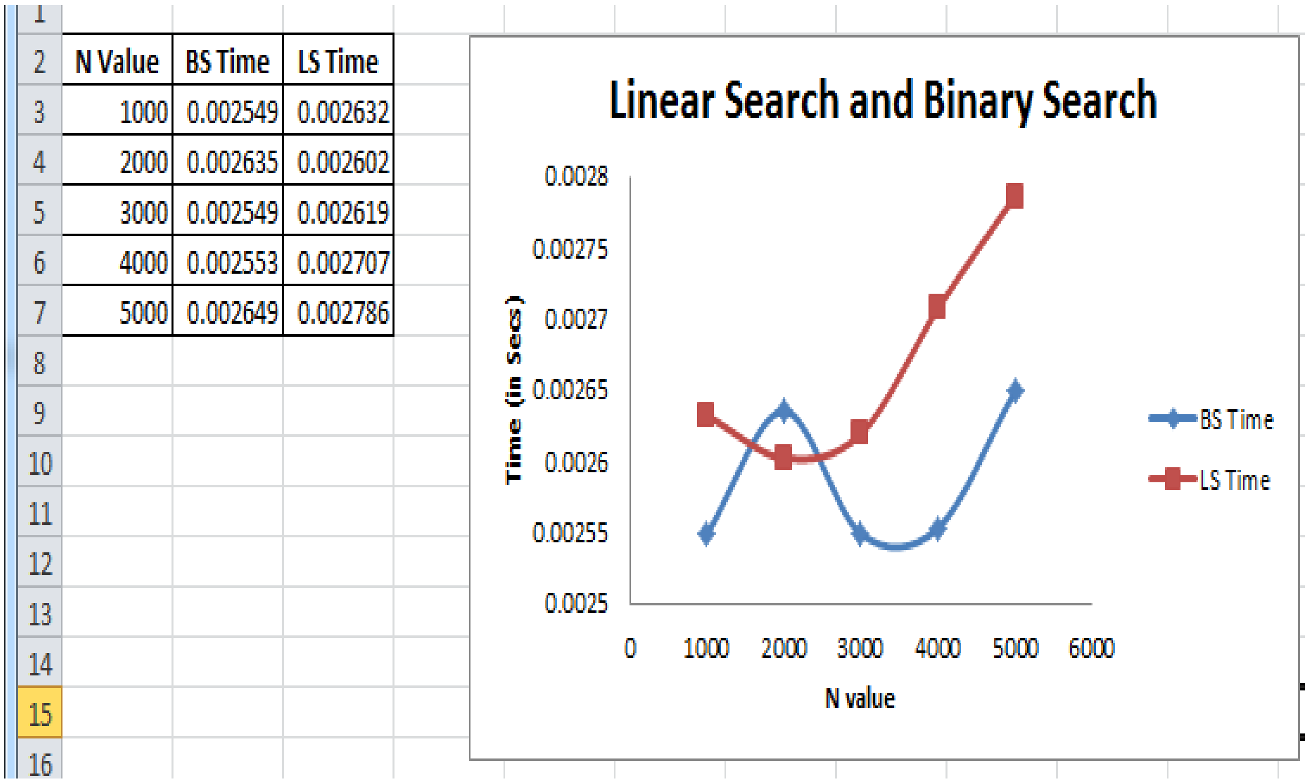
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
int bin_srch(int [],int,int,int);
int lin_srch(int [],int,int,int);
void bub_sort(int[],int);
int n,a[10000];
int main()
{
 int ch,key,search_status,temp;
 clock_t end,start;
unsigned long int i, j;

 while(1)
 {
  printf("\n1: Binary search\t 2: Linear search\t 3: Exit\n");
  printf("\nEnter your choice:\t");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:
      n=1000;
              while(n<=5000)
              {
               for(i=0;i<n;i++)
               {
               //a[i]=random(1000);
               a[i]=i;  //Insering numbers in Ascending order
               }
               key=a[n-1]; //Last element of the aray
               start=clock();
           //bub_sort(a,n); //Sorting numbers in Ascending order using Bubble sort
               search_status=bin_srch(a,0,n-1,key);
               if(search_status==-1)
              printf("\nKey Not Found");
           else
              printf("\n Key found at position %d",search_status);
           //Dummy loop to create delay
```

```
               for(j=0;j<500000;j++){ temp=38/600;}
               end=clock();
               printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
               n=n+1000;
              }
              break;
   case  2:
         n=1000;
              while(n<=5000)
              {
           for(i=0;i<n;i++)
               {
                //a[i]=random(10000);
                a[i]=i;
               }
               key=a[n-1]; //Last element of the aray
               start=clock();
               search_status=lin_srch(a,0,n-1,key);
               if(search_status==-1)
              printf("\nKey Not Found");
           else
              printf("\n Key found at position %d",search_status);
           //Dummy loop to create delay
             for(j=0;j<500000;j++){ temp=38/600;}
               end=clock();
               printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
               n=n+1000;
              }
              break;
   default:
         exit(0);
  }
  getchar();
 }
}
void bub_sort(int a[],int n)
{
 int i,j,temp;
 for(i=0;i<=n-2;i++)
 {
  for(j=0;j<=n-2-i;j++)
  {
   if(a[j]>a[j+1])
   {
    temp=a[j];
    a[j]=a[j+1];
    a[j+1]=temp;
   }
  }
 }
}
int bin_srch(int a[],int low,int high,int key)
{
 int mid;
 if(low>high)
 {
  return -1;
 }
 mid=(low+high)/2;
 if(key==a[mid])
 {
  return mid;
 }
 if(key<a[mid])
 {
  return bin_srch(a,low,mid-1,key);
 }
 else
 {
  return bin_srch(a,mid+1,high,key);
 }
}

int lin_srch(int a[],int i,int high,int key)
{
 if(i>high)
 {
  return -1;
 }
 if(key==a[i])
 {
  return i;
 }
 else
 {
  return lin_srch(a,i+1,high,key);
 }
}
```

Result Screen short:



```
1: Binary search          2: Linear search        3: Exit

Enter your choice:      1

 Key found at position 999
Time for n=1000 is 0.000853 Secs
 Key found at position 1999
Time for n=2000 is 0.000813 Secs
 Key found at position 2999
Time for n=3000 is 0.000834 Secs
 Key found at position 3999
Time for n=4000 is 0.000824 Secs
 Key found at position 4999
Time for n=5000 is 0.000818 Secs
1: Binary search          2: Linear search        3: Exit

Enter your choice:      2

 Key found at position 999
Time for n=1000 is 0.000859 Secs
 Key found at position 1999
Time for n=2000 is 0.000876 Secs
 Key found at position 2999
Time for n=3000 is 0.000868 Secs
 Key found at position 3999
Time for n=4000 is 0.000869 Secs
 Key found at position 4999
Time for n=5000 is 0.000885 Secs
1: Binary search          2: Linear search        3: Exit
```

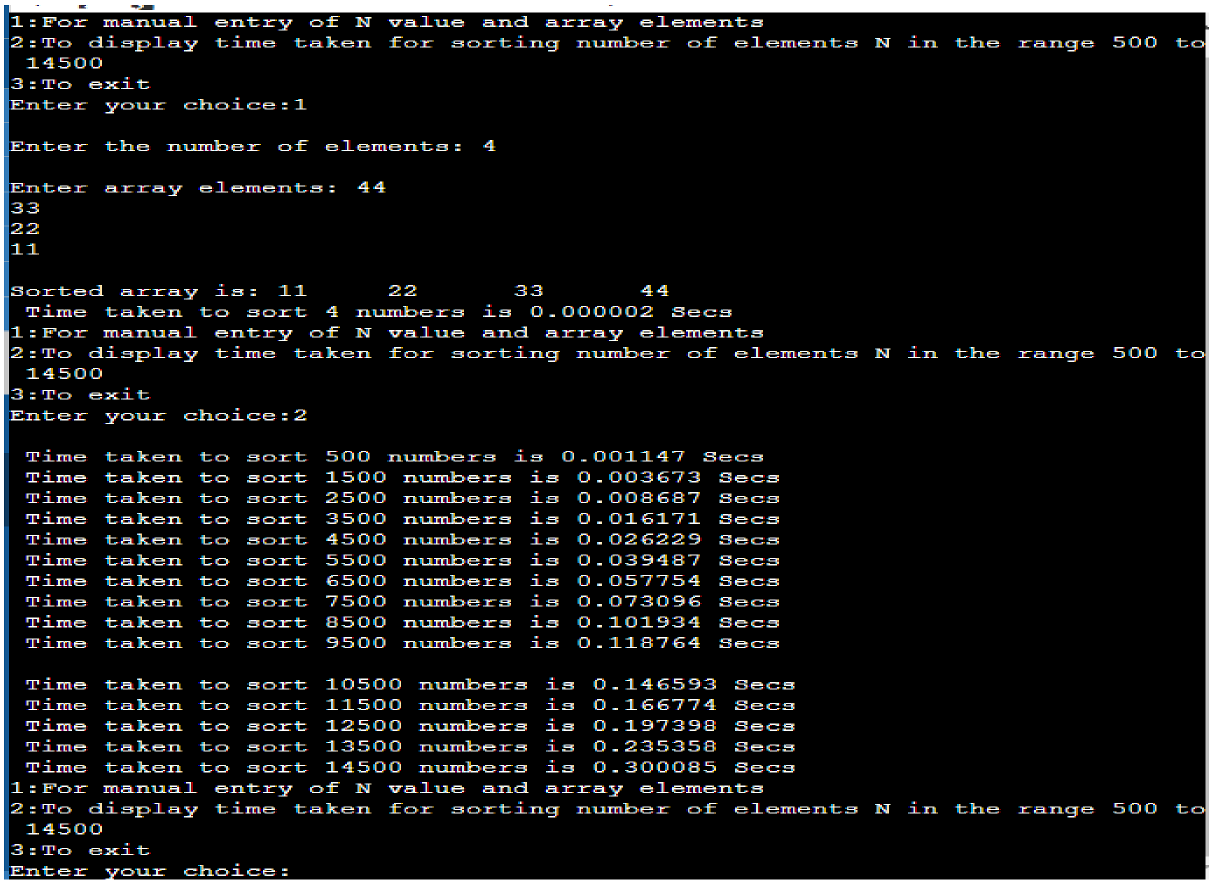


N Value	BS Time	LS Time
1000	0.002549	0.002632
2000	0.002635	0.002602
3000	0.002549	0.002619
4000	0.002553	0.002707
5000	0.002649	0.002786

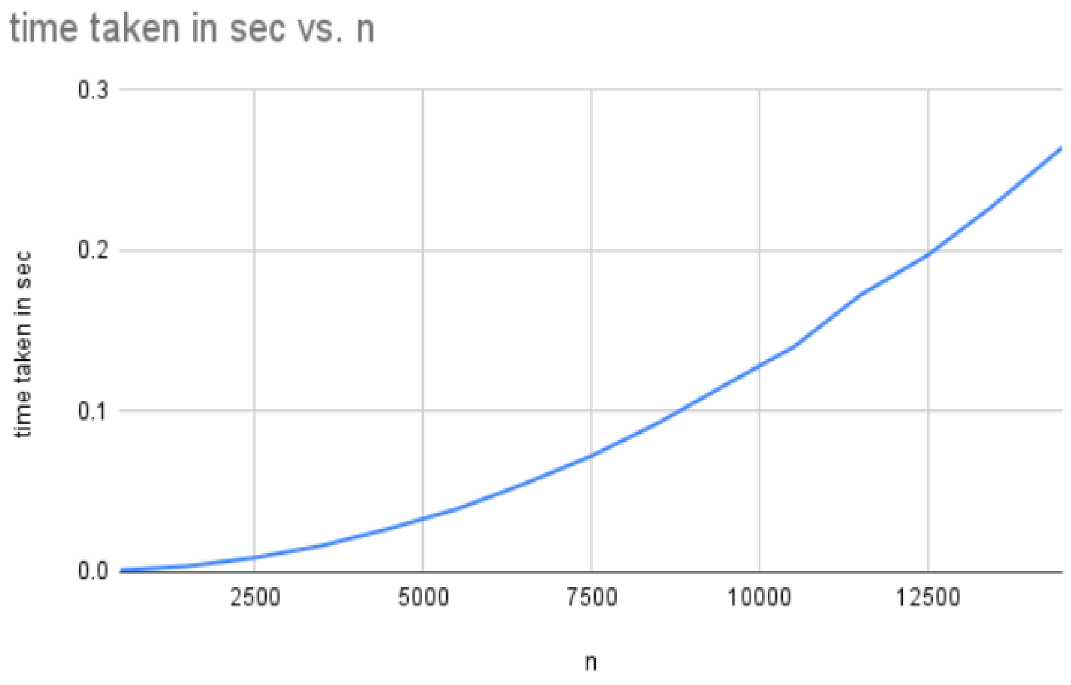
LAB PROGRAM 3: Sort a given set of N integer elements using **Selection Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void selsort(int n,int a[]);

void main()
{
   int a[15000],n,i,j,ch,temp;
   clock_t start,end;

   while(1)
   {
 printf("\n1:For manual entry of N value and array elements");
 printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");
 printf("\n3:To exit");
     printf("\nEnter your choice:");
     scanf("%d", &ch);
     switch(ch)
     {
       case 1:  printf("\nEnter the number of elements: ");
                        scanf("%d",&n);
                        printf("\nEnter array elements: ");
                        for(i=0;i<n;i++)
                         {
                          scanf("%d",&a[i]);
                         }
                        start=clock();
                        selsort(n,a);
                        end=clock();
                        printf("\nSorted array is: ");
                        for(i=0;i<n;i++)
                        printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                        break;
     case 2:
                   n=500;
                   while(n<=14500) {
                   for(i=0;i<n;i++)
                           {
                             //a[i]=random(1000);
                             a[i]=n-i;
                           }
                   start=clock();
                   selsort(n,a);
```

```
          //Dummy loop to create delay
              for(j=0;j<500000;j++){ temp=38/600;}
             end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                    n=n+1000;
                     }
               break;
   case 3: exit(0);
   }
   getchar();
    }
}


void selsort(int n,int a[])
{
     int i,j,t,small,pos;
     for(i=0;i<n-1;i++)
      {
       pos=i;
       small=a[i];
       for(j=i+1;j<n;j++)
       {
              if(a[j]<small)
              {
                small=a[j];
                pos=j;
              }
       }
       t=a[i];
       a[i]=a[pos];
       a[pos]=t;
    }
}
```



```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to
 14500
3:To exit
Enter your choice:1

Enter the number of elements: 4

Enter array elements: 44
33
22
11

Sorted array is: 11      22      33      44
 Time taken to sort 4 numbers is 0.000002 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to
 14500
3:To exit
Enter your choice:2

 Time taken to sort 500 numbers is 0.001147 Secs
 Time taken to sort 1500 numbers is 0.003673 Secs
 Time taken to sort 2500 numbers is 0.008687 Secs
 Time taken to sort 3500 numbers is 0.016171 Secs
 Time taken to sort 4500 numbers is 0.026229 Secs
 Time taken to sort 5500 numbers is 0.039487 Secs
 Time taken to sort 6500 numbers is 0.057754 Secs
 Time taken to sort 7500 numbers is 0.073096 Secs
 Time taken to sort 8500 numbers is 0.101934 Secs
 Time taken to sort 9500 numbers is 0.118764 Secs

 Time taken to sort 10500 numbers is 0.146593 Secs
 Time taken to sort 11500 numbers is 0.166774 Secs
 Time taken to sort 12500 numbers is 0.197398 Secs
 Time taken to sort 13500 numbers is 0.235358 Secs
 Time taken to sort 14500 numbers is 0.300085 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to
 14500
3:To exit
Enter your choice:
```



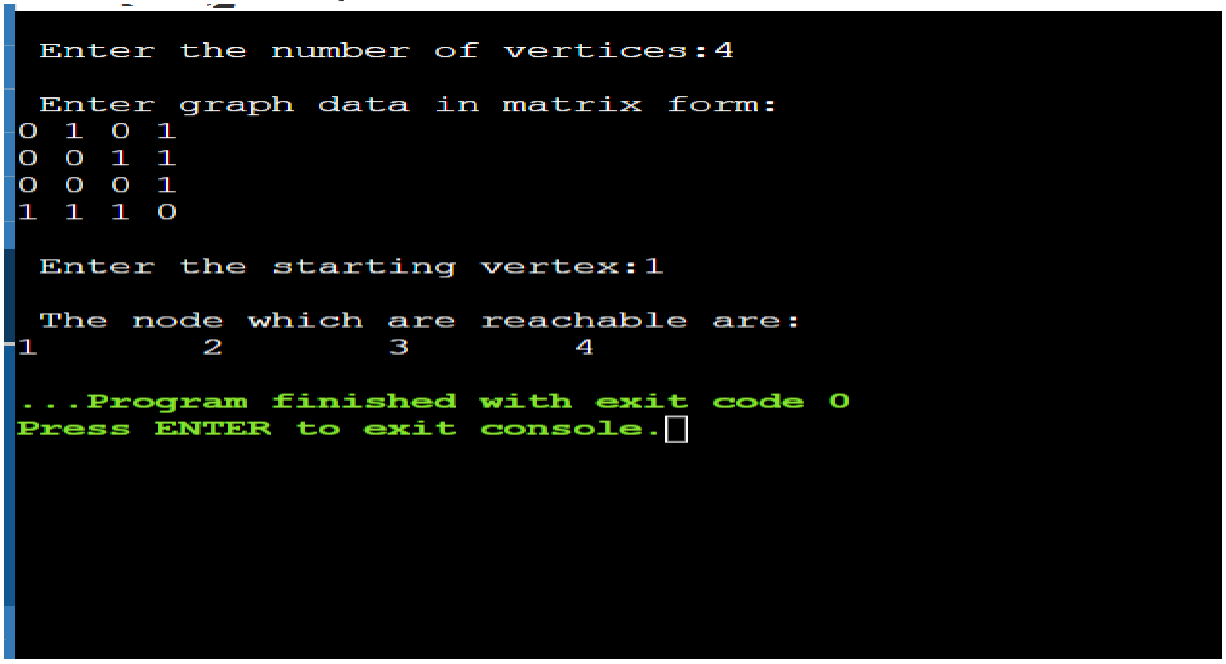
LAB PROGRAM 4: Write program to do the following:

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.

a)

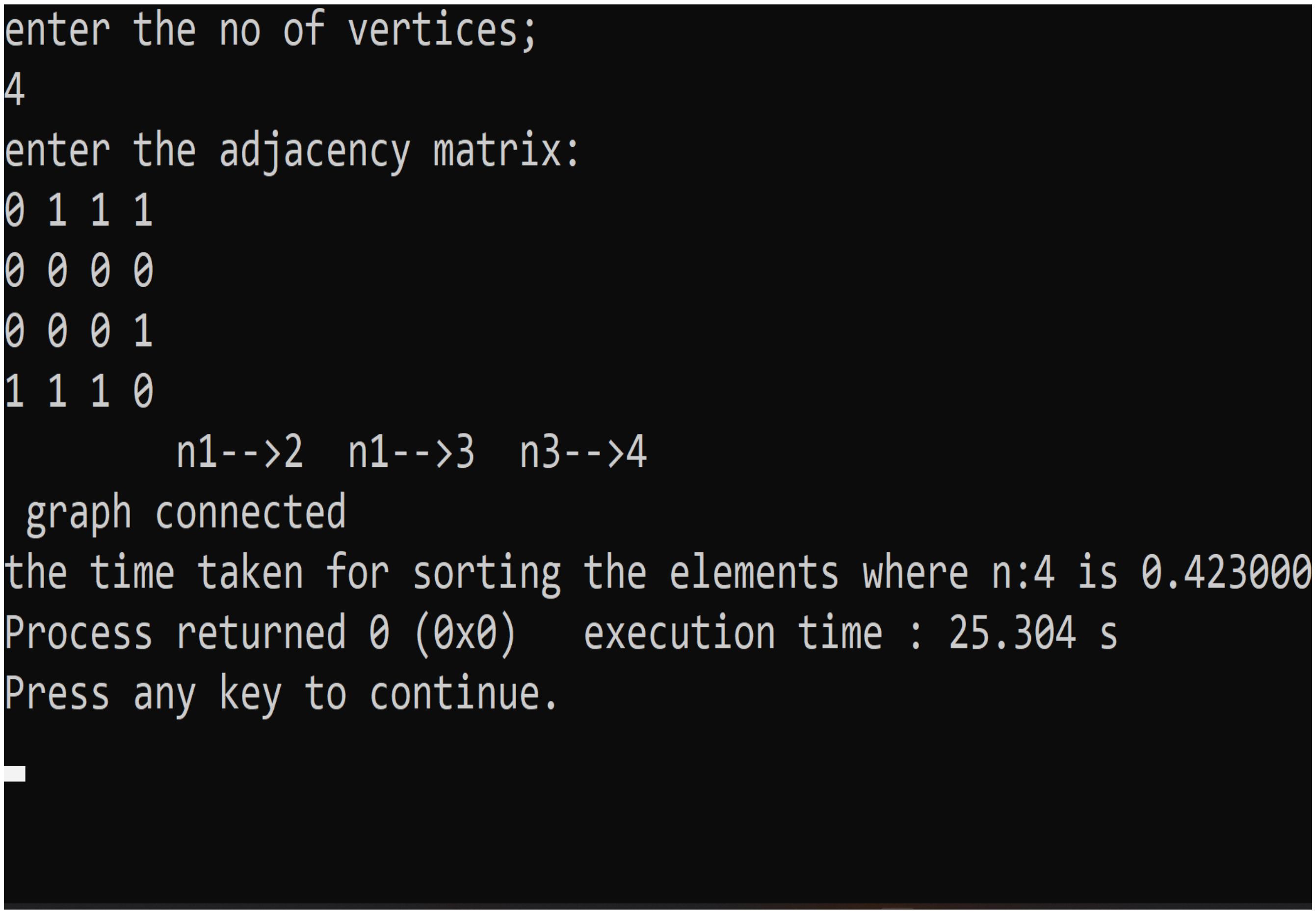
```
#include<stdio.h>
#include <conio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

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();
}
```



```
 Enter the number of vertices:4

 Enter graph data in matrix form:
0 1 0 1
0 0 1 1
0 0 0 1
1 1 1 0

 Enter the starting vertex:1

 The node which are reachable are:
1       2       3       4

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

b)

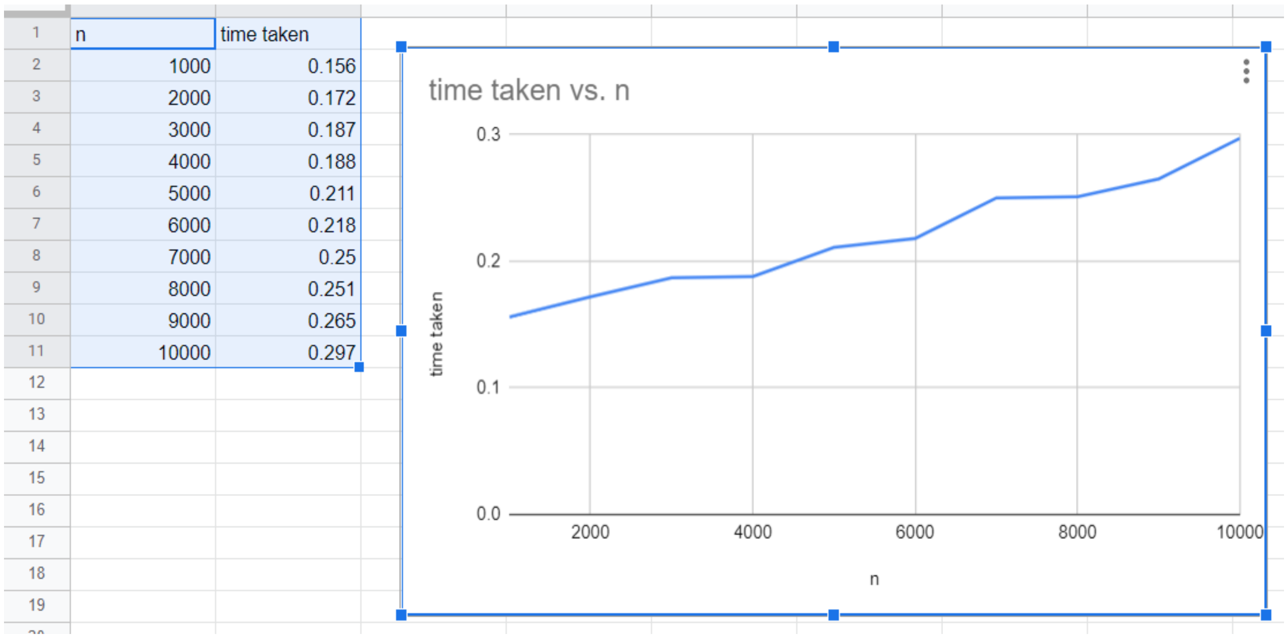
```
#include<stdio.h>
                                    #include<conio.h>
                                    #include<time.h>
                                    #include<windows.h>
                                    int a[20][20],reach[20],n;
                                    void dfs(int v){
                                    int i;
                                    reach[v]=1;
                                    for(i=1;i<=n;i++)
                                    if(a[v][i]&&!reach[i]){
                                    printf("\tn%d-->%d",v,i);
                                    dfs(i);
                                    }
                                    }

                                    int main(){
                                    clock_t s,e;
                                    int i,j,count=0;
                                    printf("enter the no of vertices;\n");
                                    scanf("%d",&n);
                                    printf("enter the adjacency matrix:\n");
                                    for(i=1;i<=n;i++)
                                    for(j=1;j<=n;j++)
                                    scanf("%d",&a[i][j]);
                                    s=clock();
                                    dfs(1);
                                    Sleep(400);
                                    e=clock();
                                    for(i=1;i<=n;i++)
                                    if(reach[i])
                                    count++;
                                    if(count==n)
                                    printf("\n graph connected");
                                    else
                                    printf("\n graph is disconnected");
                                    printf("\nthe time taken for sorting the elements where
                                    n:%d is %f",n,((double)(e-s))/CLK_TCK);
                                    return(0);
                                    }
```



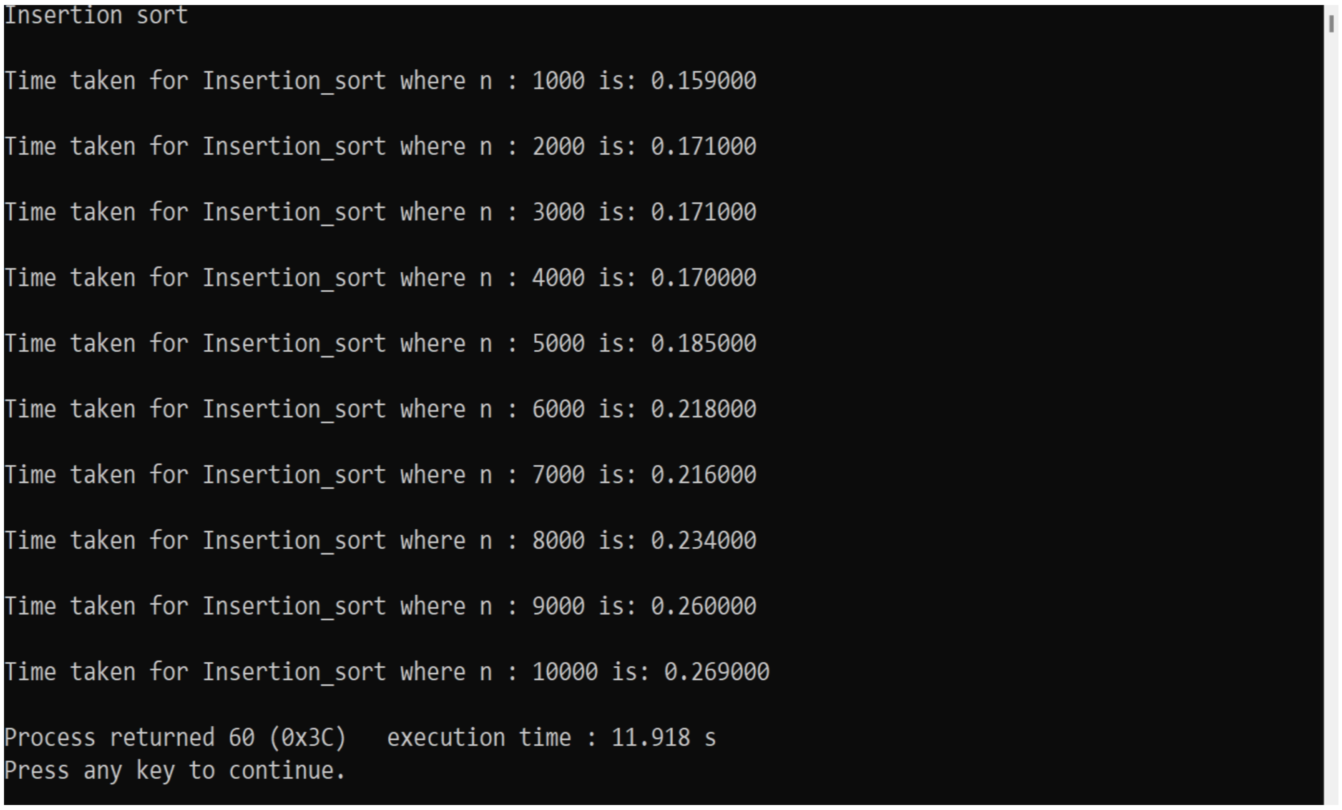
```
enter the no of vertices;
4
enter the adjacency matrix:
0 1 1 1
0 0 0 0
0 0 0 1
1 1 1 0
        n1-->2  n1-->3  n3-->4
 graph connected
the time taken for sorting the elements where n:4 is 0.423000
Process returned 0 (0x0)   execution time : 25.304 s
Press any key to continue.
```

LAB PROGRAM 5: Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.

```
#include<stdio.h>

#include<time.h>
#include<windows.h>
#include<stdlib.h>
int insertion_sort(int a[],int n){
int i,j,item;
for(i=0;i<n;i++){
    item=a[i];
    j=i-1;
    while(j>=0 && a[j]>item){
        a[j+1]=a[i];
        j=j-1;
    }
    a[j+1]=item;
}
}
void main(){
int a[20000],i,j,n,num,u=0,l=400,result  ;
clock_t s,e;
printf("Insertion sort\n");
n=1000;
while(n<=10000){
for(i=0;i<n;i++){
    a[i]=n-i;
}
s=clock();
result= insertion_sort(a,n);
Sleep(150);
e=clock();
printf("\nTime taken for Insertion_sort where n : %d is:
%f\n",n,((double)(e-s))/CLK_TCK);
n=n+1000;
}
}
```



n	time taken
1000	0.156
2000	0.172
3000	0.187
4000	0.188
5000	0.211
6000	0.218
7000	0.25
8000	0.251
9000	0.265
10000	0.297

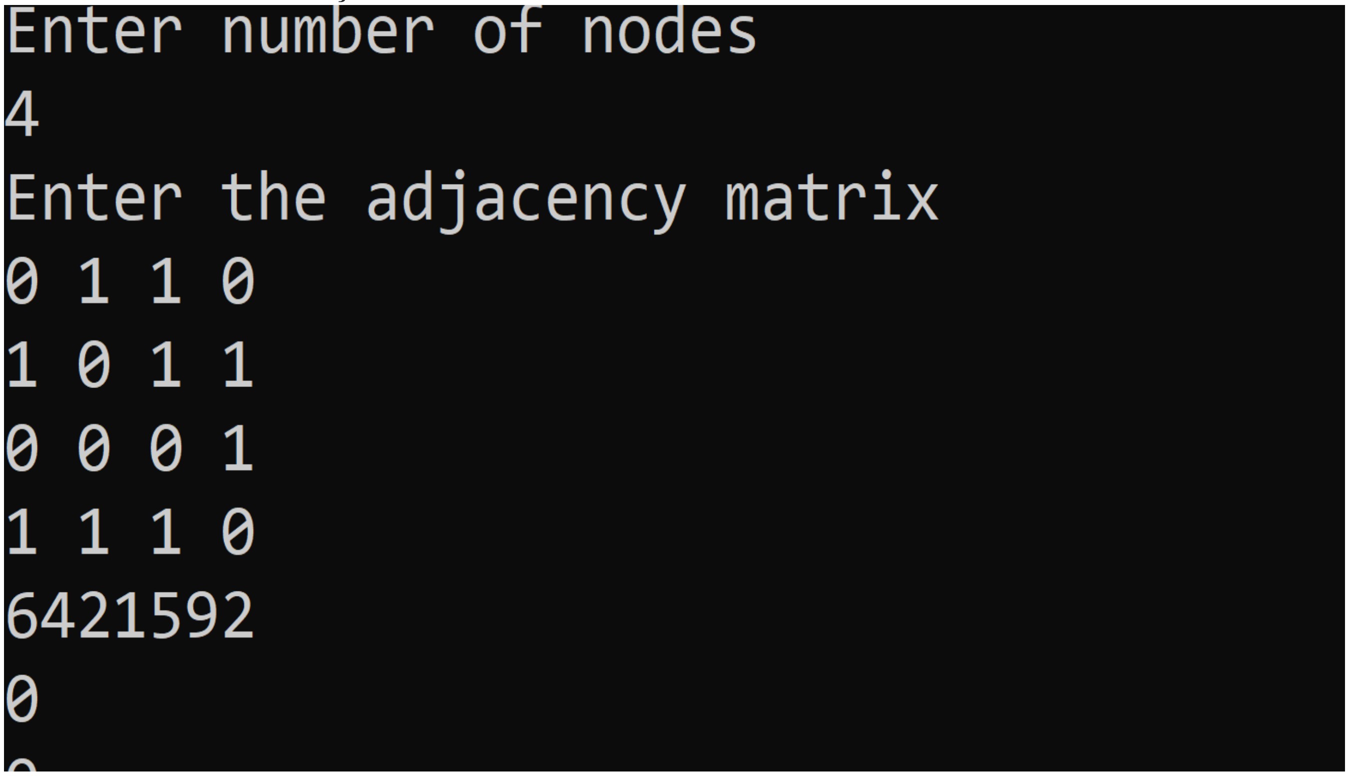


```
Insertion sort

Time taken for Insertion_sort where n : 1000 is: 0.159000

Time taken for Insertion_sort where n : 2000 is: 0.171000

Time taken for Insertion_sort where n : 3000 is: 0.171000

Time taken for Insertion_sort where n : 4000 is: 0.170000

Time taken for Insertion_sort where n : 5000 is: 0.185000

Time taken for Insertion_sort where n : 6000 is: 0.218000

Time taken for Insertion_sort where n : 7000 is: 0.216000

Time taken for Insertion_sort where n : 8000 is: 0.234000

Time taken for Insertion_sort where n : 9000 is: 0.260000

Time taken for Insertion_sort where n : 10000 is: 0.269000

Process returned 60 (0x3C)   execution time : 11.918 s
Press any key to continue.
```

LAB PROGRAM 6: Write program to obtain the Topological ordering of vertices in a given digraph

```
#include<stdio.h>
                #include<conio.h>
                void source_removal(int n, int a[10][10])
                {
                        inti,j,k,u,v,top,s[10],t[10],indeg[10],sum;
                        for(i=0;i<n;i++)
                        {
                                sum=0;
                                for(j=0;j<n;j++)
                                {
                                        sum+=a[j][i];
                                }
                                indeg[i]=sum;
                        }
                        top=-1;
                        for(i=0;i<n;i++)
                        {
                                if(indeg[i]==0)
                                {
                                        s[++top]=i;
                                }
                        }
                        k=0;
                        while(top!=-1)
                        {
                                u=s[top--];
                                t[k++]=u;
                                for(v=0;v<n;v++)
                                {
                                        if(a[u][v]==1)
                                        {
                                                indeg[v]=indeg[v]-1;
                                                if(indeg[v]==0)
                                                        s[++top]=v;
                                        }
                                }
                        }
```

```
          for(i=0;i<n;i++)
          {
                    printf("%d\n", t[i]);
          }
}
void main()
{
          inti,j,a[10][10],n;
          printf("Enter number of nodes\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]);
                    }
          }
          source_removal(n,a);
          getch();
}
```

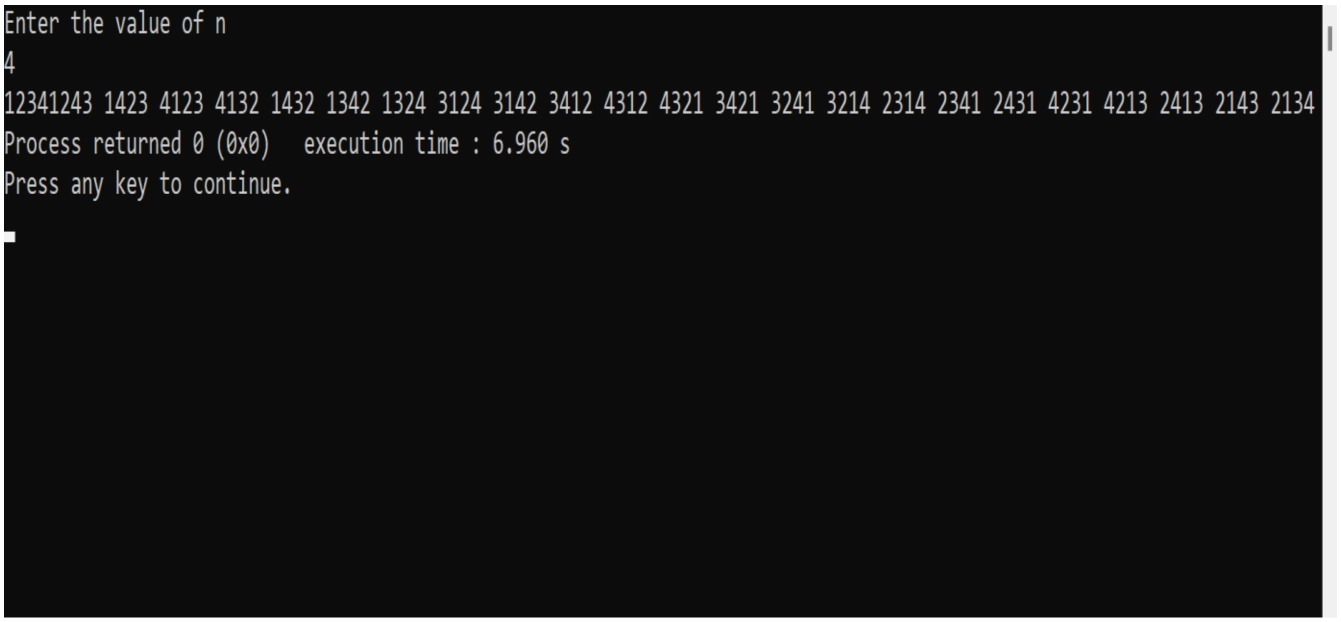


```
Enter number of nodes
4
Enter the adjacency matrix
0 1 1 0
1 0 1 1
0 0 0 1
1 1 1 0
6421592
0
```

LAB PROGRAM 7: Implement Johnson Trotter algorithm to generate permutations

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

int LEFT_TO_RIGHT = 1;
int RIGHT_TO_LEFT = 0;



int searchArr(int a[], int n, int mobile)
{
    for (int i = 0; i < n; i++)

        if (a[i] == mobile)
            return i + 1;

    return 0;
}




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

    for (int i = 0; i < n; i++)
    {
        //dir == 0 => Right to left
        if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0)
        {
            if (a[i] > a[i - 1] &&
                a[i] > mobile_prev)
            {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
```

```
            //dir == 1 => Left to right
            if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1)
            {
                if (a[i] > a[i + 1] &&
                    a[i] > mobile_prev)
                {
                    mobile = a[i];
                    mobile_prev = mobile;
                }
            }
        }

        if (mobile == 0 && mobile_prev == 0)
            return 0;
        else
            return mobile;
    }


    int printOnePerm(int a[], int dir[], int n)
    {
        int mobile = getMobile(a, dir, n);
        int pos = searchArr(a, n, mobile);

        // swapping the elements
        // according to the
        // direction
        if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT)
        {
            int temp = a[pos - 1];
            a[pos - 1] = a[pos - 2];
            a[pos - 2] = temp;
        }
        else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
        {
            int temp = a[pos];
            a[pos] = a[pos - 1];
            a[pos - 1] = temp;
        }
    // changing the directions
    // for elements greater
    // than largest mobile integer.
    for (int i = 0; i < n; i++)
    {
        if (a[i] > mobile)
        {
            if (dir[a[i] - 1] == LEFT_TO_RIGHT)
                dir[a[i] - 1] = RIGHT_TO_LEFT;

            else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
                dir[a[i] - 1] = LEFT_TO_RIGHT;
        }
    }

    for (int i = 0; i < n; i++)
        printf("%d" , a[i]);

    printf(" ");

    return 0;
}



     int fact(int n)
    {
        int res = 1;

        for (int i = 1; i <= n; i++)
            res = res * i;
        return res;
    }


    void printPermutation(int n)
    {
        // To store current
        // permutation
        int a[n];

         // To store current
         // directions
        int dir[n];

         // storing the elements
         // from 1 to n and
         // printing first permutation.
         for (int i = 0; i < n; i++)
         {
             a[i] = i + 1;
             printf("%d", a[i]);
         }
          for (int i = 0; i < n; i++)
             dir[i] = RIGHT_TO_LEFT;

         // for generating permutations
         // in the order.
         for (int i = 1; i < fact(n); i++)
             printOnePerm(a, dir, n);
     }

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

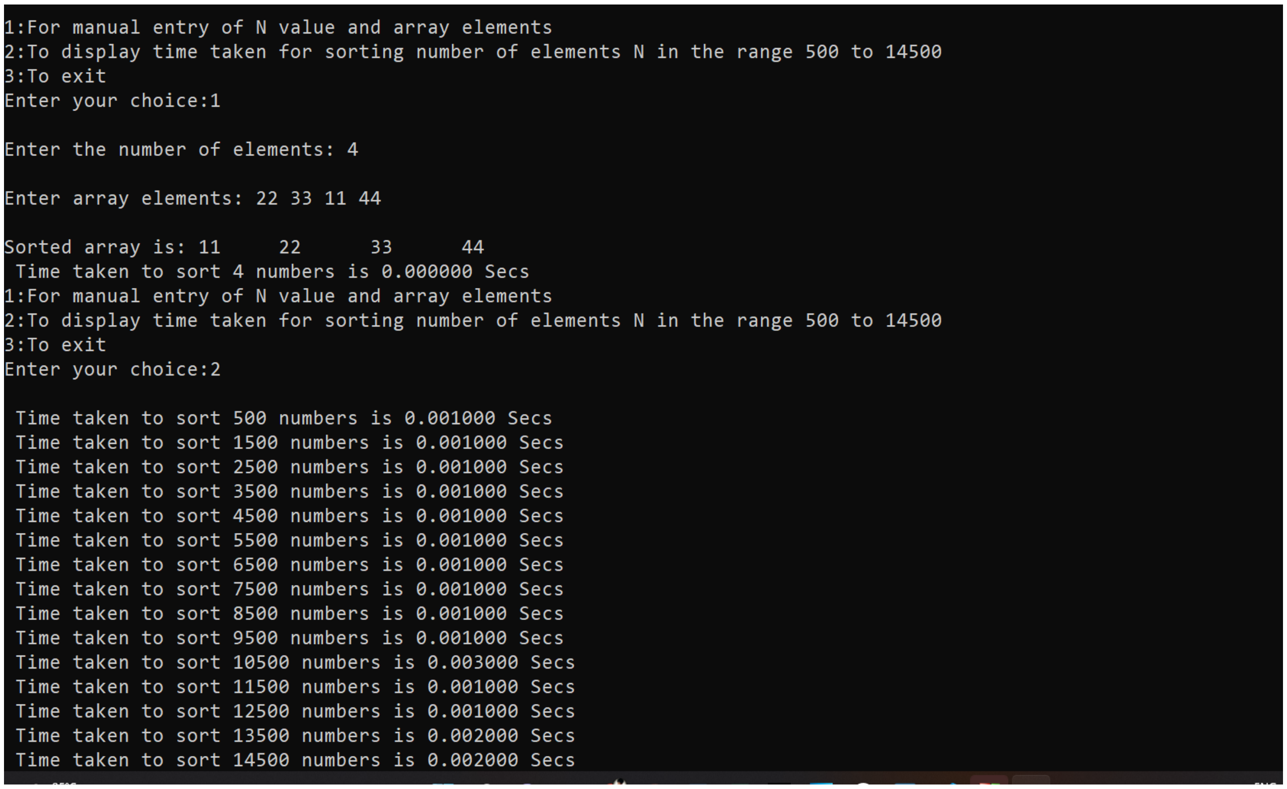


```
Enter the value of n
4
12341243 1423 4123 4132 1432 1342 1324 3124 3142 3412 4312 4321 3421 3241 3214 2314 2341 2431 4231 4213 2413 2143 2134
Process returned 0 (0x0)   execution time : 6.960 s
Press any key to continue.
```

LAB PROGRAM 8: 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.

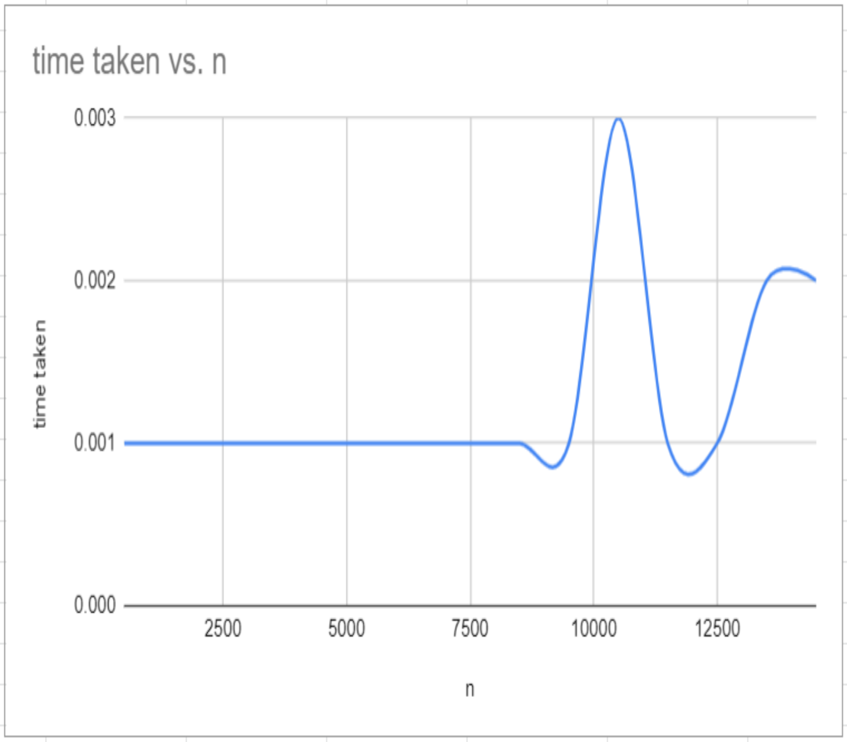
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void split(int[],int,int);
void combine(int[],int,int,int);
void main()
{
   int a[15000],n, i,j,ch, temp;
   clock_t start,end;

   while(1)
   {
 printf("\n1:For manual entry of N value and array elements");
 printf("\n2:To display time taken for sorting number of elements N in the range
500 to 14500");
 printf("\n3:To exit");
     printf("\nEnter your choice:");
     scanf("%d", &ch);
     switch(ch)
     {
       case 1:  printf("\nEnter the number of elements: ");
                    scanf("%d",&n);
                    printf("\nEnter array elements: ");
                    for(i=0;i<n;i++)
                     {
                      scanf("%d",&a[i]);
                     }
                    start=clock();
                    split(a,0,n-1);
                    end=clock();
                    printf("\nSorted array is: ");
                    for(i=0;i<n;i++)
                    printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                    break;
```

```
    case 2:
                n=500;
                while(n<=14500) {
                for(i=0;i<n;i++)
                     {
                       //a[i]=random(1000);
                       a[i]=n-i;
                     }
                start=clock();
                split(a,0,n-1);
          //Dummy loop to create delay
            for(j=0;j<500000;j++){ temp=38/600;}
                end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                     n=n+1000;
                     }
               break;
  case 3: exit(0);
  }
  getchar();
   }
}

void split(int a[],int low,int high)
{
 int mid;
 if(low<high)
 {
  mid=(low+high)/2;
  split(a,low,mid);
  split(a,mid+1,high);
  combine(a,low,mid,high);
 }
}

void combine(int a[],int low,int mid,int high)
{
 int c[15000],i,j,k;
 i=k=low;
 j=mid+1;
 while(i<=mid&&j<=high)
 {
  if(a[i]<a[j])
  {
   c[k]=a[i];
   ++k;
   ++i;
  }
  else
  {
   c[k]=a[j];
   ++k;
   ++j;
  }
 }
 if(i>mid)
 {
  while(j<=high)
  {
   c[k]=a[j];
   ++k;
   ++j;
  }
 }
 if(j>high)
 {
  while(i<=mid)
  {
   c[k]=a[i];
   ++k;
   ++i;
  }
 }
 for(i=low;i<=high;i++)
 {
  a[i]=c[i];
 }
}
```



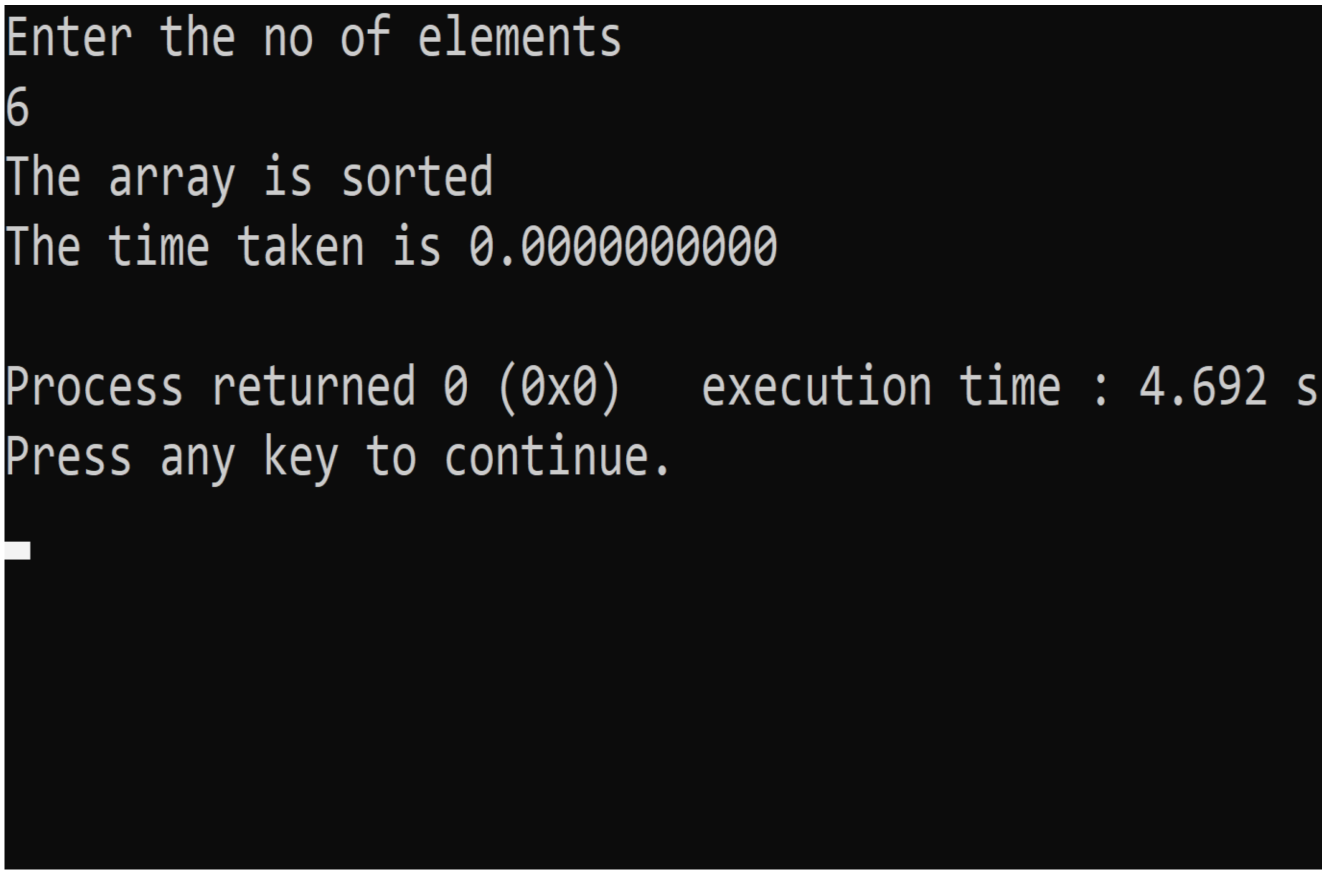
```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 4

Enter array elements: 22 33 11 44

Sorted array is: 11      22      33      44
 Time taken to sort 4 numbers is 0.000000 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2

 Time taken to sort 500 numbers is 0.001000 Secs
 Time taken to sort 1500 numbers is 0.001000 Secs
 Time taken to sort 2500 numbers is 0.001000 Secs
 Time taken to sort 3500 numbers is 0.001000 Secs
 Time taken to sort 4500 numbers is 0.001000 Secs
 Time taken to sort 5500 numbers is 0.001000 Secs
 Time taken to sort 6500 numbers is 0.001000 Secs
 Time taken to sort 7500 numbers is 0.001000 Secs
 Time taken to sort 8500 numbers is 0.001000 Secs
 Time taken to sort 9500 numbers is 0.001000 Secs
 Time taken to sort 10500 numbers is 0.003000 Secs
 Time taken to sort 11500 numbers is 0.001000 Secs
 Time taken to sort 12500 numbers is 0.001000 Secs
 Time taken to sort 13500 numbers is 0.002000 Secs
 Time taken to sort 14500 numbers is 0.002000 Secs
```

n	time taken
500	0.001
1500	0.001
2500	0.001
3500	0.001
4500	0.001
5500	0.001
6500	0.001
7500	0.001
8500	0.001
9500	0.001
10500	0.003
11500	0.001
12500	0.001
13500	0.002
14500	0.002



LAB PROGRAM 9: Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int arr[1000000];
void swap(int arr[], int index1,int index2){
  int temp= arr[index1];
  arr[index1] = arr[index2];
  arr[index2]=temp;
}
int partition(int arr[] ,int start , int end){
    int pivot = arr[end];
    int boundary = start-1;
    int i;
    for( i=start ; i<=end;i++){
        if(arr[i] <= pivot){
            boundary++;
            swap(arr,i,boundary);
        }
    }
    return boundary;
}
void quicksort(int arr[] , int start , int end){
 int i,j;
for( i=0;i<800;i++)
{
    for(j=0;j<400;j++)
    {
    }
}
    if(start>=end)
        return;
    int boundary = partition(arr, start , end);
    quicksort(arr ,start ,boundary-1);
    quicksort(arr ,boundary+1,end);
}
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);
    quicksort(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;
}
```

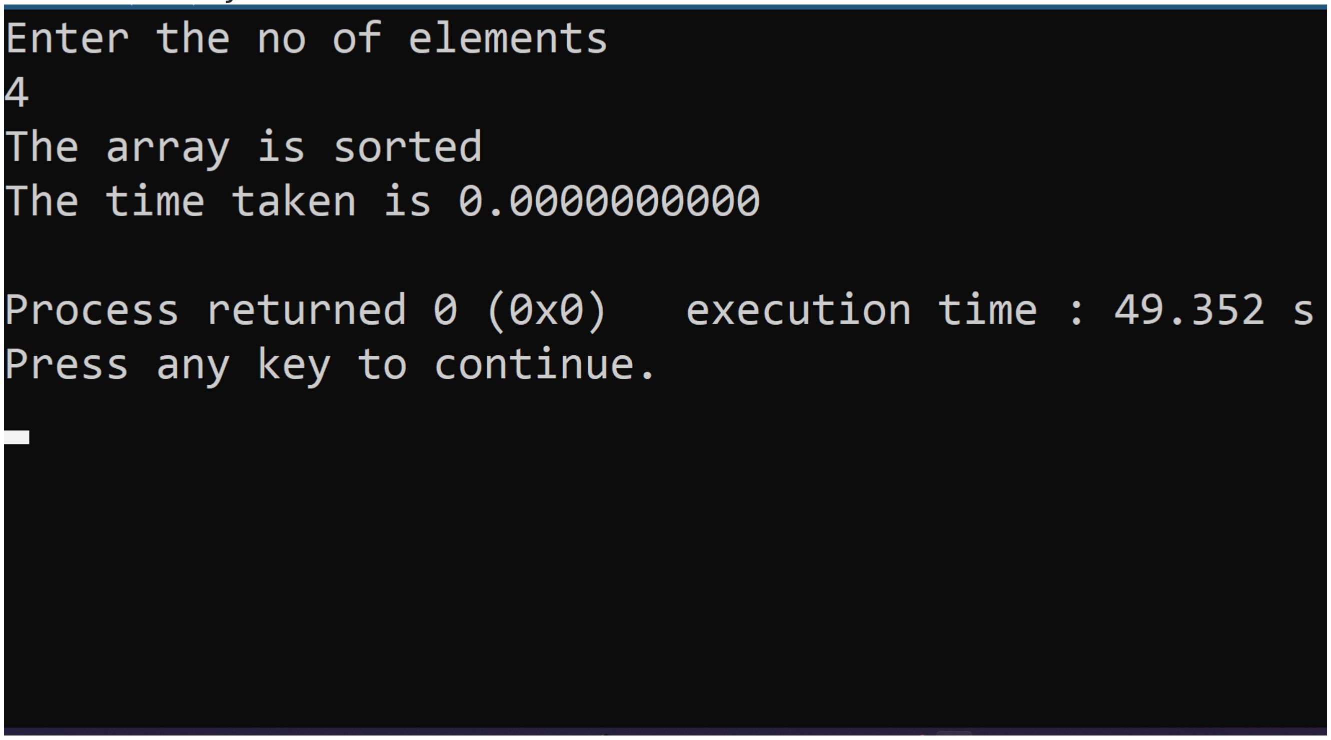


```
Enter the no of elements
6
The array is sorted
The time taken is 0.0000000000

Process returned 0 (0x0)   execution time : 4.692 s
Press any key to continue.
```

LAB PROGRAM 10: 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;
}
```

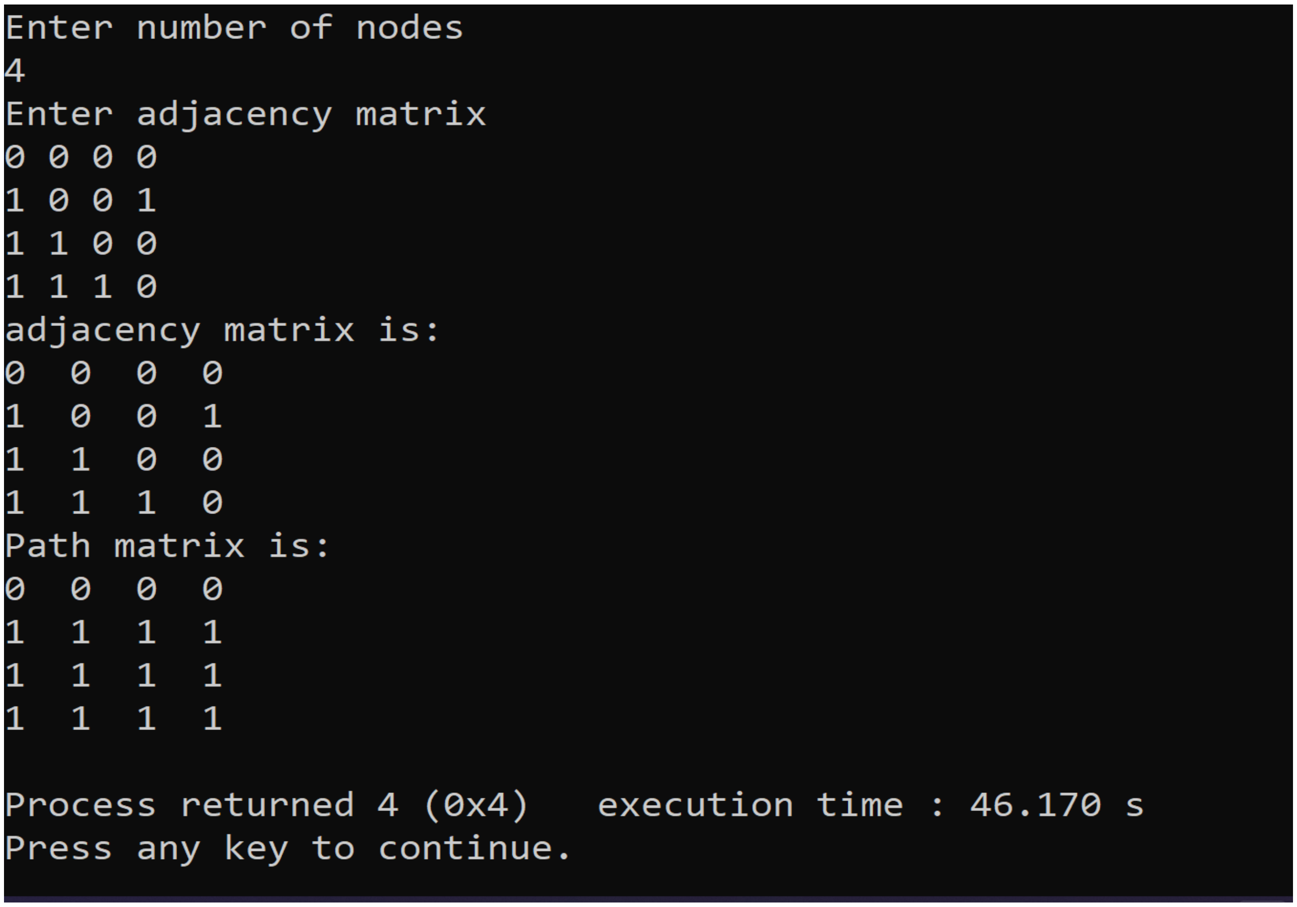


```
Enter the no of elements
4
The array is sorted
The time taken is 0.0000000000

Process returned 0 (0x0)   execution time : 49.352 s
Press any key to continue.
```

LAB PROGRAM 11: Implement Warshall's algorithm using dynamic programming

```
 #include<stdio.h>
                    void warshall(int a[10][10], int p[10][10],int n)
                    {

                        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]!=1 && p[i][k]==1 &&
                    p[k][j]==1)
                                    p[i][j]=1;
                                }
                            }
                        }
                    }

                    void main()
                    {
                        int a[10][10],p[10][10],n,i,j;
                        printf("Enter number of nodes\n");
                        scanf("%d",&n);
                        printf("Enter adjacency matrix\n");
                        for(i=0;i<n;i++)
                        {
                            for(j=0;j<n;j++)
                            scanf("%d",&a[i][j]);
                        }
                        printf("adjacency matrix is: \n");
```

```
          for(i=0;i<n;i++)
          {
              for(j=0;j<n;j++)
              printf("%d  ",a[i][j]);
              printf("\n");
          }
          warshall(a,p,n);
          printf("Path matrix is: \n");
          for(i=0;i<n;i++)
          {
              for(j=0;j<n;j++)
              printf("%d  ",p[i][j]);
              printf("\n");
          }

      }
```

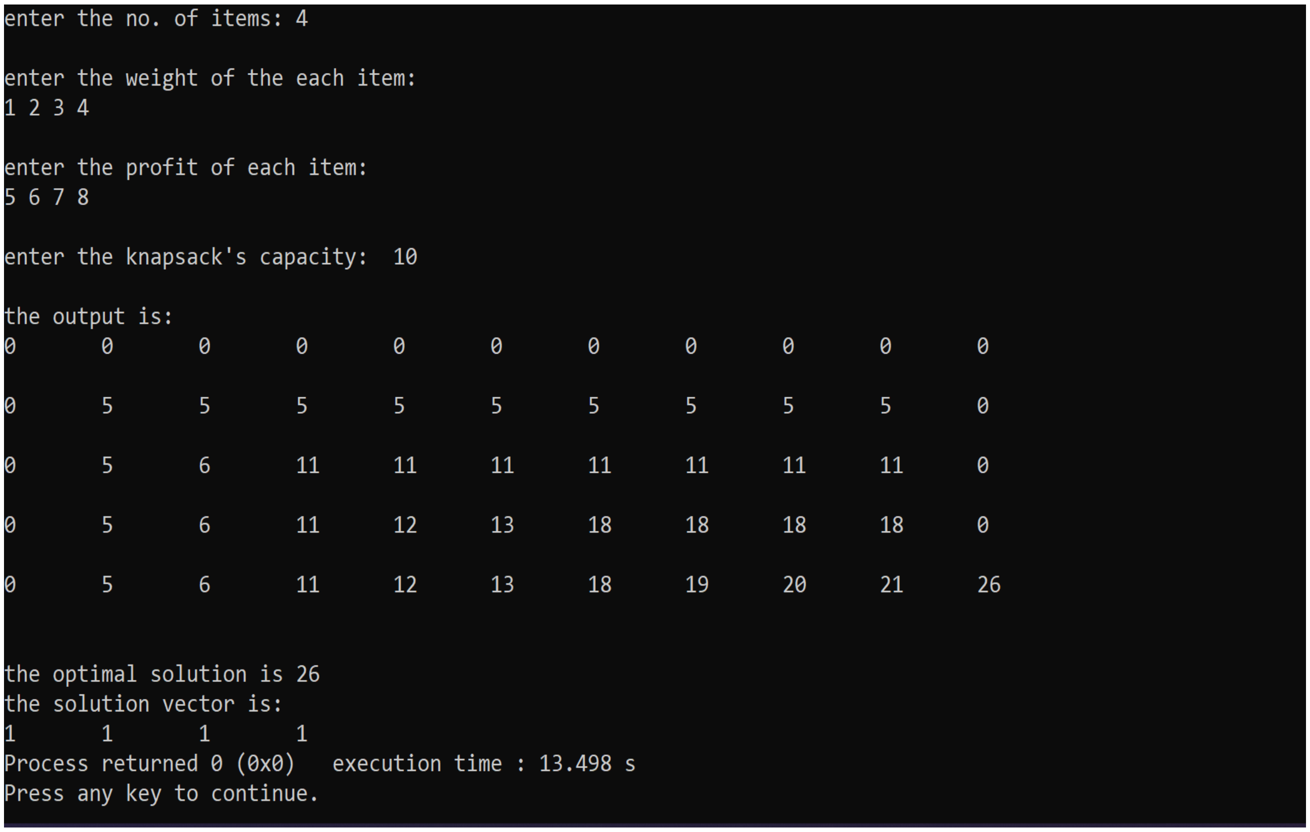


```
Enter number of nodes
4
Enter adjacency matrix
0 0 0 0
1 0 0 1
1 1 0 0
1 1 1 0
adjacency matrix is:
0  0  0  0
1  0  0  1
1  1  0  0
1  1  1  0
Path matrix is:
0  0  0  0
1  1  1  1
1  1  1  1
1  1  1  1

Process returned 4 (0x4)   execution time : 46.170 s
Press any key to continue.
```

LAB PROGRAM 12: Implement 0/1 Knapsack problem using dynamic programming.

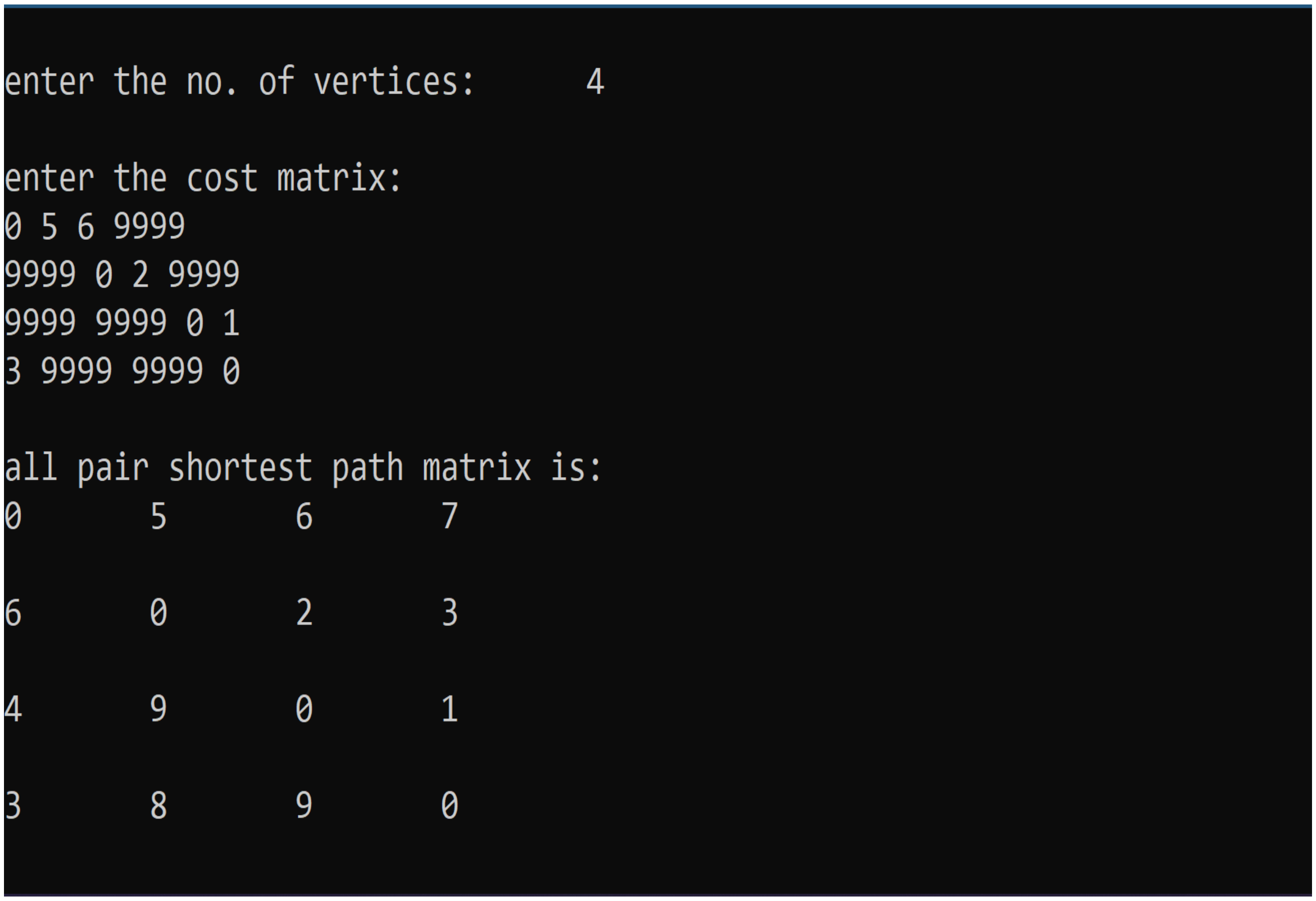
```
#include<stdio.h>
#include<conio.h>
void knapsack();
int max(int,int);
int i,j,n,m,p[10],w[10],v[10][10];
int  main()
{
 printf("enter the no. of items:\t");
 scanf("%d",&n);
 printf("\nenter the weight of the each item:\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&w[i]);
 }
 printf("\nenter the profit 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();
}
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;
 }
}
```


```
enter the no. of items: 4

enter the weight of the each item:
1 2 3 4

enter the profit of each item:
5 6 7 8

enter the knapsack's capacity:  10

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

0       5       5       5       5       5       5       5       5       5       0

0       5       6       11      11      11      11      11      11      11      0

0       5       6       11      12      13      18      18      18      18      0

0       5       6       11      12      13      18      19      20      21      26


the optimal solution is 26
the solution vector is:
1       1       1       1
Process returned 0 (0x0)   execution time : 13.498 s
Press any key to continue.
```

LAB PROGRAM 13: Implement All Pair Shortest paths problem using Floyd's algorithm.

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n;
void floyds();
int min(int,int);
void main()
{
 int i,j;

 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;
 }
}
```



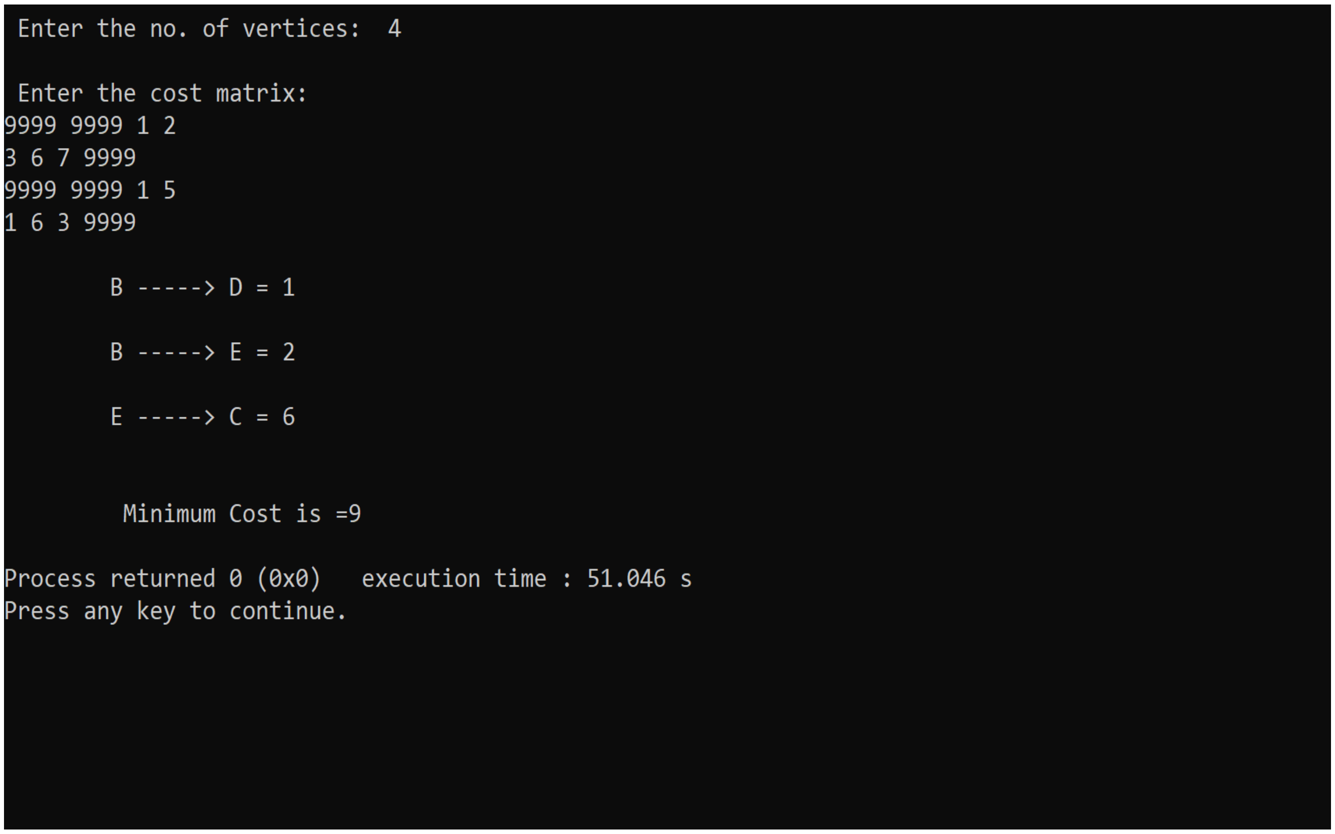
```
enter the no. of vertices:      4

enter the cost matrix:
0 5 6 9999
9999 0 2 9999
9999 9999 0 1
3 9999 9999 0

all pair shortest path matrix is:
0       5       6       7

6       0       2       3

4       9       0       1

3       8       9       0
```

LAB PROGRAM 14: Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
#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);
}
```

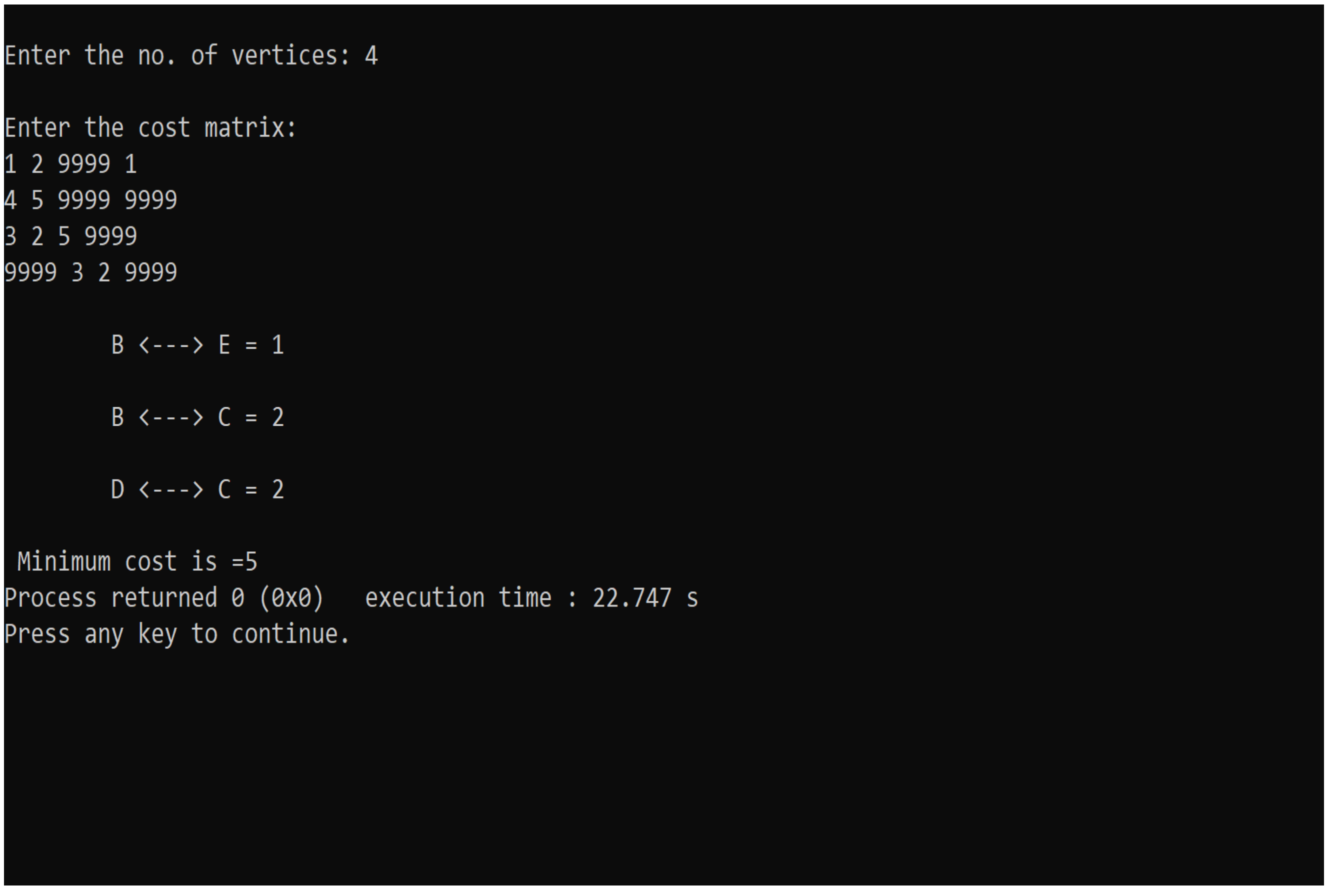


```
 Enter the no. of vertices:  4

 Enter the cost matrix:
9999 9999 1 2
3 6 7 9999
9999 9999 1 5
1 6 3 9999

        B -----> D = 1

        B -----> E = 2

        E -----> C = 6


         Minimum Cost is =9

Process returned 0 (0x0)   execution time : 51.046 s
Press any key to continue.
```

LAB PROGRAM 15: 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;                                  // No of edges
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);
}
```

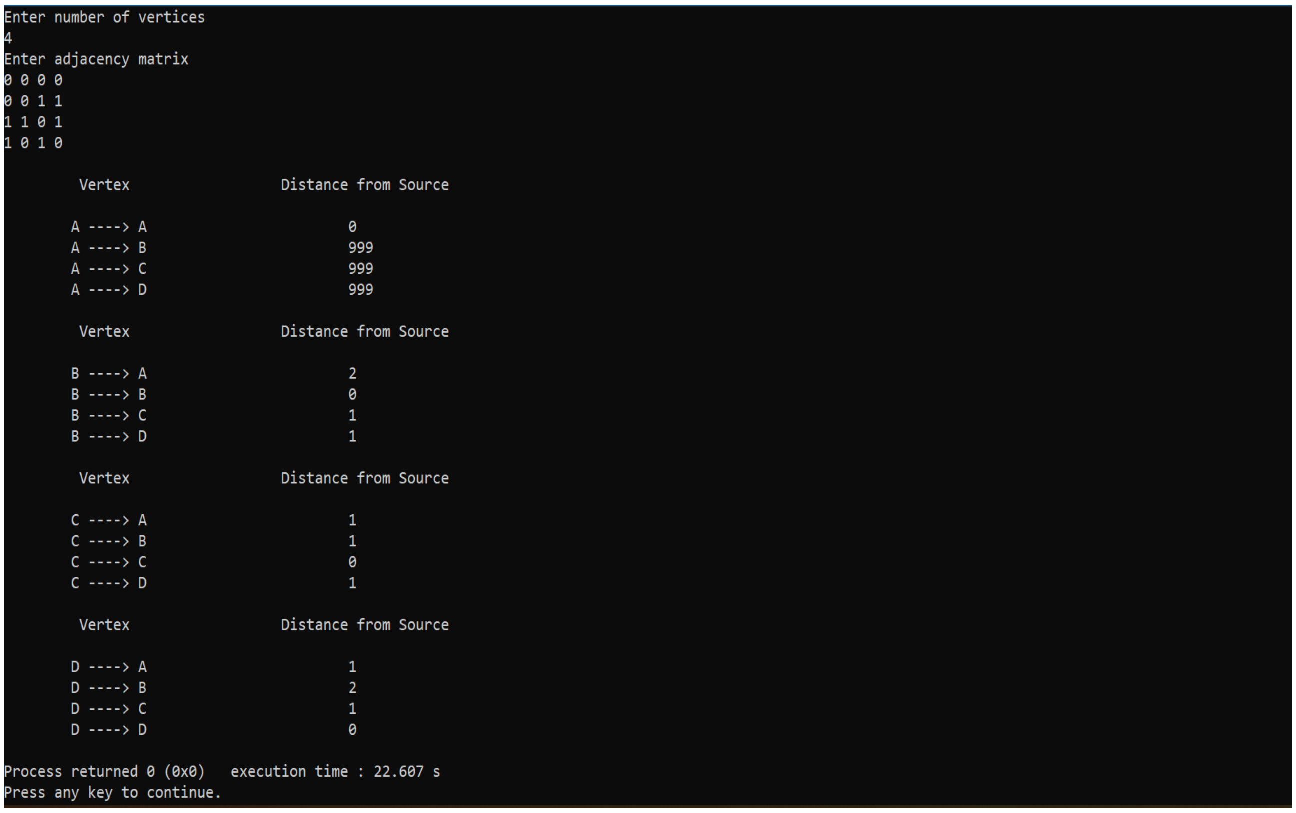


```
Enter the no. of vertices: 4

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

        B <---> E = 1

        B <---> C = 2

        D <---> C = 2

 Minimum cost is =5
Process returned 0 (0x0)   execution time : 22.747 s
Press any key to continue.
```

LAB PROGRAM 16: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

```
#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;
}
```



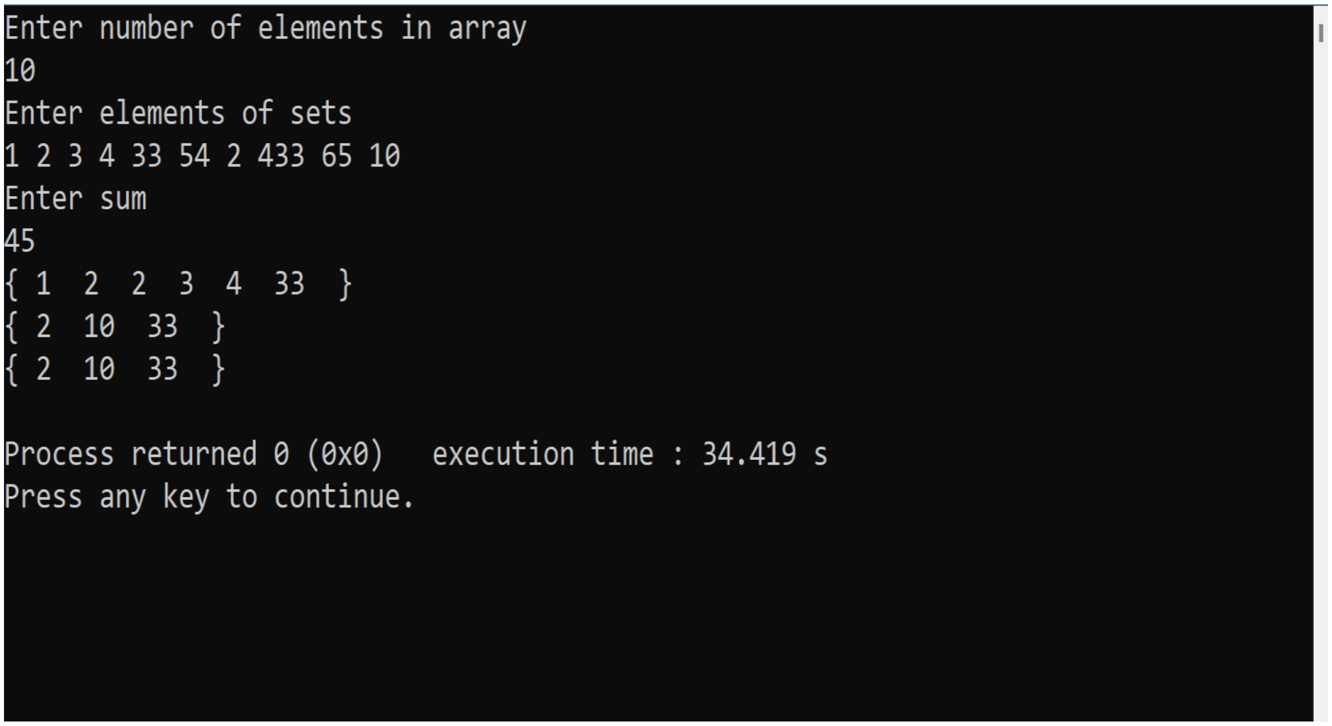
```
Enter number of vertices
4
Enter adjacency matrix
0 0 0 0
0 0 1 1
1 1 0 1
1 0 1 0

         Vertex                  Distance from Source

        A ----> A                        0
        A ----> B                        999
        A ----> C                        999
        A ----> D                        999

         Vertex                  Distance from Source

        B ----> A                        2
        B ----> B                        0
        B ----> C                        1
        B ----> D                        1

         Vertex                  Distance from Source

        C ----> A                        1
        C ----> B                        1
        C ----> C                        0
        C ----> D                        1

         Vertex                  Distance from Source

        D ----> A                        1
        D ----> B                        2
        D ----> C                        1
        D ----> D                        0

Process returned 0 (0x0)   execution time : 22.607 s
Press any key to continue.
```

a. **LAB 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 <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);
```

}

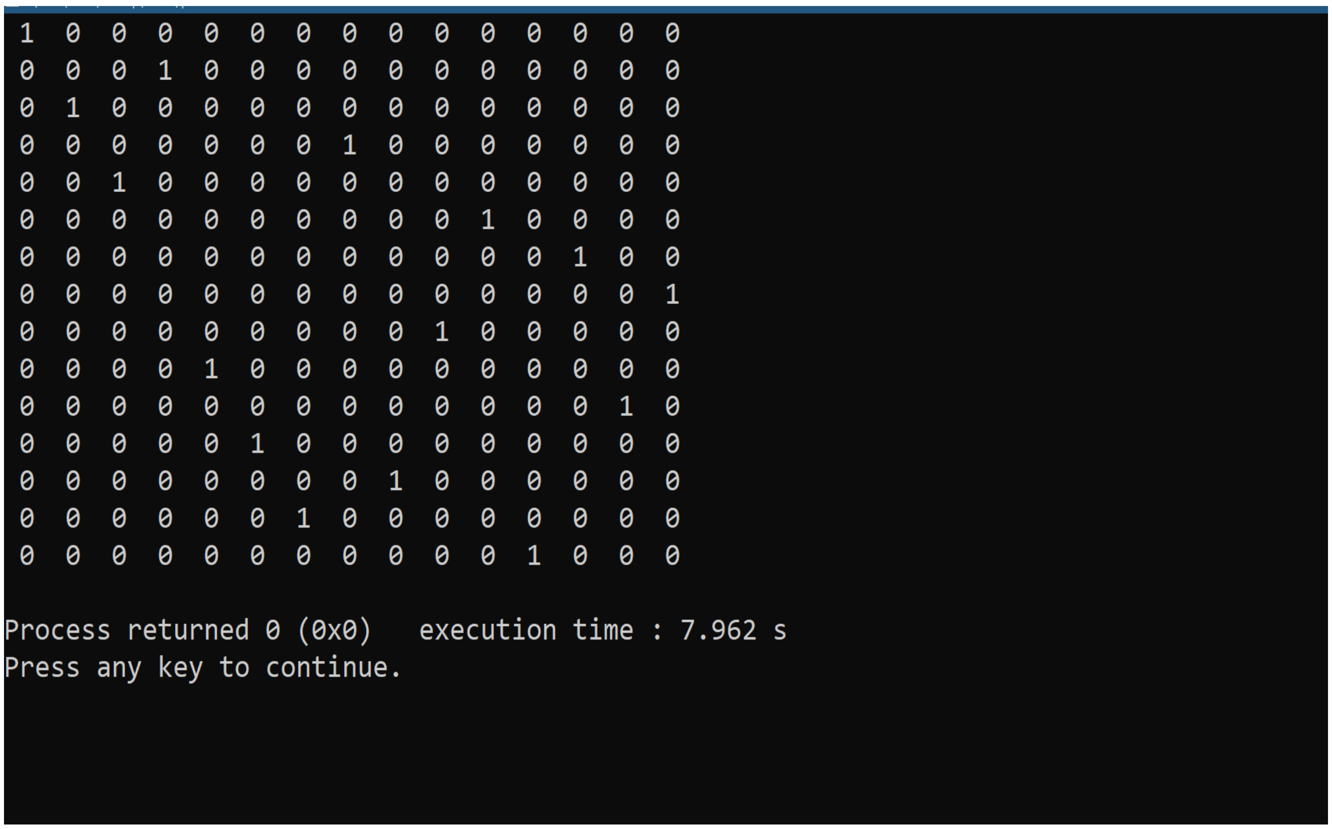


```
Enter number of elements in array
10
Enter elements of sets
1 2 3 4 33 54 2 433 65 10
Enter sum
45
{ 1  2  2  3  4  33  }
{ 2  10  33  }
{ 2  10  33  }

Process returned 0 (0x0)   execution time : 34.419 s
Press any key to continue.
```

LAB PROGRAM 18: Implement “N-Queens Problem” using Backtracking.

```
#define N 15
#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;
    }
```



```
 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0

Process returned 0 (0x0)   execution time : 7.962 s
Press any key to continue.
```