VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

ANALYSIS AND DESIGN OF ALGORITHMS

Submitted by

Manshi Mahato (1BM20CS208)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)

BENGALURU-560019

June-2022 to Aug-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 work entitled "**ANALYSIS AND DESIGN OF ALGORITHMS**" carried out by MANSHI MAHATO **(1BM20CS208),** who is a 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 - (19CS34PCADA)** work prescribed for the said degree.

Rekha G.S   **Dr. Jyothi S Nayak**

Associate Professor   Professor and Head

Department of CSE   Department of CSE

BMSCE, Bengaluru   BMSCE, Bengaluru

Index Sheet

12	Implement 0/1 Knapsack problem using dynamic programming.	53-54
13	Implement All Pair Shortest paths problem using Floyd's algorithm.	54-56
14	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm	56-58
15	Find Minimum Cost Spanning Tree of a given undirected graph	58-59

	using Kruskals algorithm	
16	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm	59-61
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.	61-63
18	Implement "N-Queens Problem" using Backtracking	63-64

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.

LAB PROGRAM-01
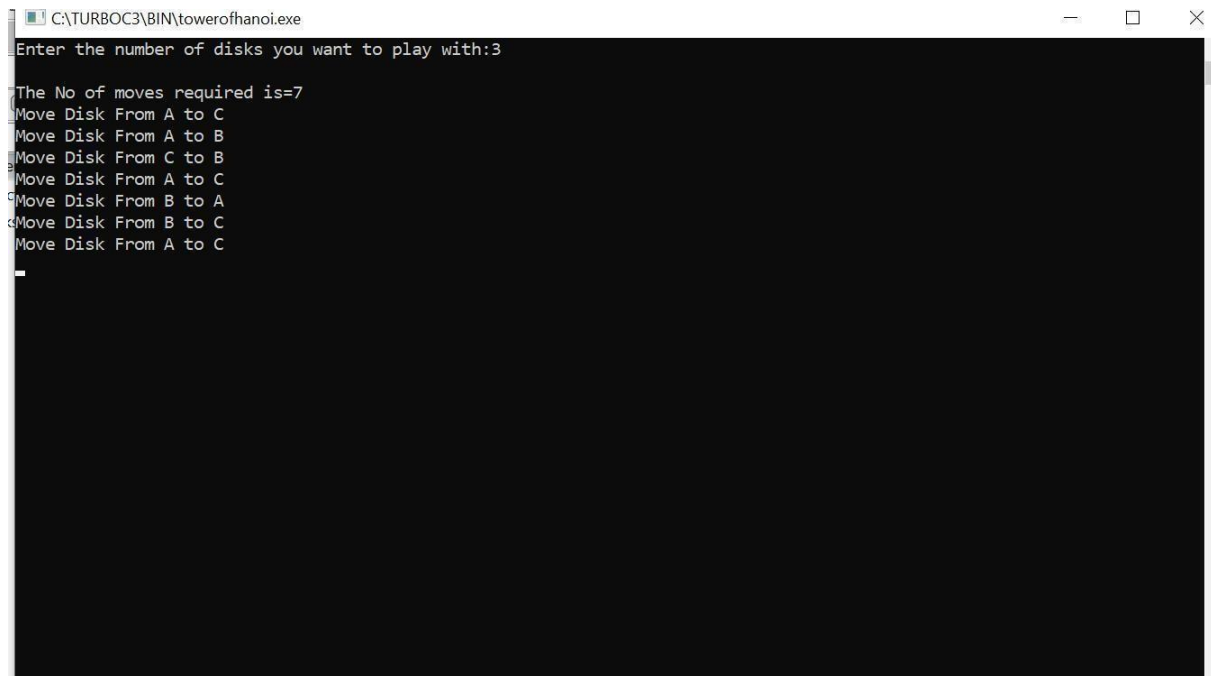
Write a recursive program to

   a. **Solve Towers-of-Hanoi problem**

   b. **To find GCD**

a.

```
#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( );
}
```

OUTPUT:

```
C:\TURBOC3\BIN\towerofhanoi.exe                                    —   □   ×
Enter the number of disks you want to play with:3

The No of moves required is=7
Move Disk From A to C
Move Disk From A to B
Move Disk From C to B
Move Disk From A to C
Move Disk From B to A
Move Disk From B to C
Move Disk From A to C
```

b.   #include <stdio.h>

int hcf(int n1, int n2);
int main()
{ int n1, n2; printf("Enter two

   positive integers:

   "); scanf("%d %d", &n1, &n2); printf("G.C.D of %d
and %d is %d.", n1, n2, hcf(n1,n2)); return 0; }

int hcf(int n1, int n2)
{ if (n2 != 0) return

 hcf(n2, n1%n2);

 else

  return n1;

}

OUTPUT:

```
 C:\TURBOC3\BIN\gcdlab.exe                                          —    □    X

Enter two positive integers: 36 60
G.C.D of 36 and 60 is 12.
Process returned 0 (0x0)   execution time : 9.854 s
Press any key to continue.
▄
```

```
 C:\TURBOC3\BIN\gcdlab.exe                                          —    □    X

Enter two positive integers: 24 12
G.C.D of 24 and 12 is 12.
Process returned 0 (0x0)   execution time : 6.907 s
Press any key to continue.
```

LAB PROGRAM-02

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.

```c
#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(1
```

```
000);
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);
```

```c
         //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-2i;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);
```

```
    }

    }
```

OUTPUT:

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

Enter your choice:      1

 Key found at position 999
Time for n=1000 is 0.000823 Secs
 Key found at position 1999
Time for n=2000 is 0.000856 Secs
 Key found at position 2999
Time for n=3000 is 0.000802 Secs
 Key found at position 3999
Time for n=4000 is 0.000791 Secs
 Key found at position 4999
Time for n=5000 is 0.000815 Secs
1: Binary search          2: Linear search        3: Exit

Enter your choice:      []
```

```
                                          input
Time for n=2000 is 0.000856 Secs
 Key found at position 2999
Time for n=3000 is 0.000802 Secs
 Key found at position 3999
Time for n=4000 is 0.000791 Secs
 Key found at position 4999
Time for n=5000 is 0.000815 Secs
1: Binary search          2: Linear search        3: Exit

Enter your choice:      2

 Key found at position 999
Time for n=1000 is 0.000882 Secs
 Key found at position 1999
Time for n=2000 is 0.000878 Secs
 Key found at position 2999
Time for n=3000 is 0.000872 Secs
 Key found at position 3999
Time for n=4000 is 0.000875 Secs
 Key found at position 4999
Time for n=5000 is 0.000882 Secs
1: Binary search          2: Linear search        3: Exit

Enter your choice:
```

n	binary	linear
1000	0.000823	0.000882
2000	0.000856	0.000878
3000	0.000802	0.000872
4000	0.000791	0.000875
5000	0.000815	0.000882

Chart Title
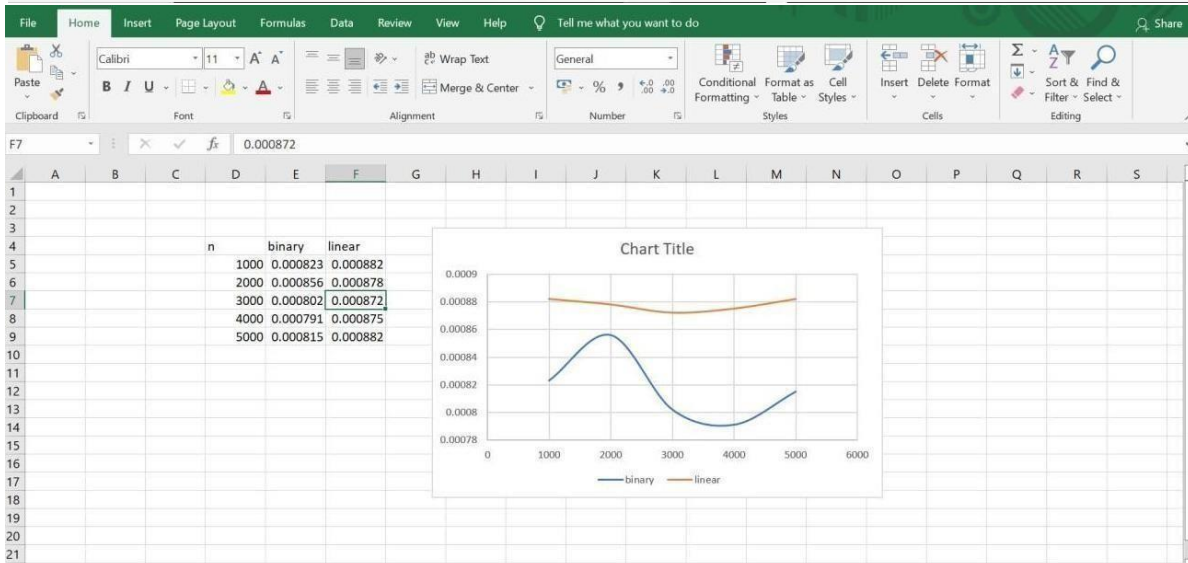
LAB PROGRAM-03

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.

```c
#include<stdio.h>

#include<time.h>

#include<stdlib.h

>void selsort(int

n,int

a[]); int 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 1000 to 10000"); 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]);
```

```c
                } 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)(endstart))/CLOCKS_PER_SEC));
                break;
        case 2:
                n=1000;
                while(n<=10000)
                {for(i=0;i<n;i++)
                    {
                      //a[i]=random(1000)
                      ; a[i]=n-i;
                    }
                start=clock();
                selsort(n,a);
                for(j=0;j<50000
                0;j++){
                temp=38/600;}
                end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n,
(((double)(endstart))/CLOCKS_PER_SEC));
```

```
                    n=n+1000;

                    }

              break;

    case 3: exit(0);

    }

    getchar();

     return 0;

     } } 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;

    }

}


OUTPUT:
```

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 1000 to 10000
3:To exit
Enter your choice:2

 Time taken to sort 1000 numbers is 0.002804 Secs
 Time taken to sort 2000 numbers is 0.007686 Secs
 Time taken to sort 3000 numbers is 0.015258 Secs
 Time taken to sort 4000 numbers is 0.027125 Secs
 Time taken to sort 5000 numbers is 0.050448 Secs
 Time taken to sort 6000 numbers is 0.057996 Secs
 Time taken to sort 7000 numbers is 0.088157 Secs
 Time taken to sort 8000 numbers is 0.113538 Secs
 Time taken to sort 9000 numbers is 0.145561 Secs
 Time taken to sort 10000 numbers is 0.173759 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 1000 to 10000
3:To exit
Enter your choice:
```

```
 Time taken to sort 6000 numbers is 0.057996 Secs
 Time taken to sort 7000 numbers is 0.088157 Secs
 Time taken to sort 8000 numbers is 0.113538 Secs
 Time taken to sort 9000 numbers is 0.145561 Secs
 Time taken to sort 10000 numbers is 0.173759 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 1000 to 10000
3:To exit
Enter your choice:1

Enter the number of elements: 10

Enter array elements: 12 156 68 45 41752 44 86 78 2 9

Sorted array is: 2        9       12      44      45      68      78      86      156     41752
 Time taken to sort 10 numbers is 0.000003 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 1000 to 10000
3:To exit
Enter your choice:3


...Program finished with exit code 0
Press ENTER to exit console.
```
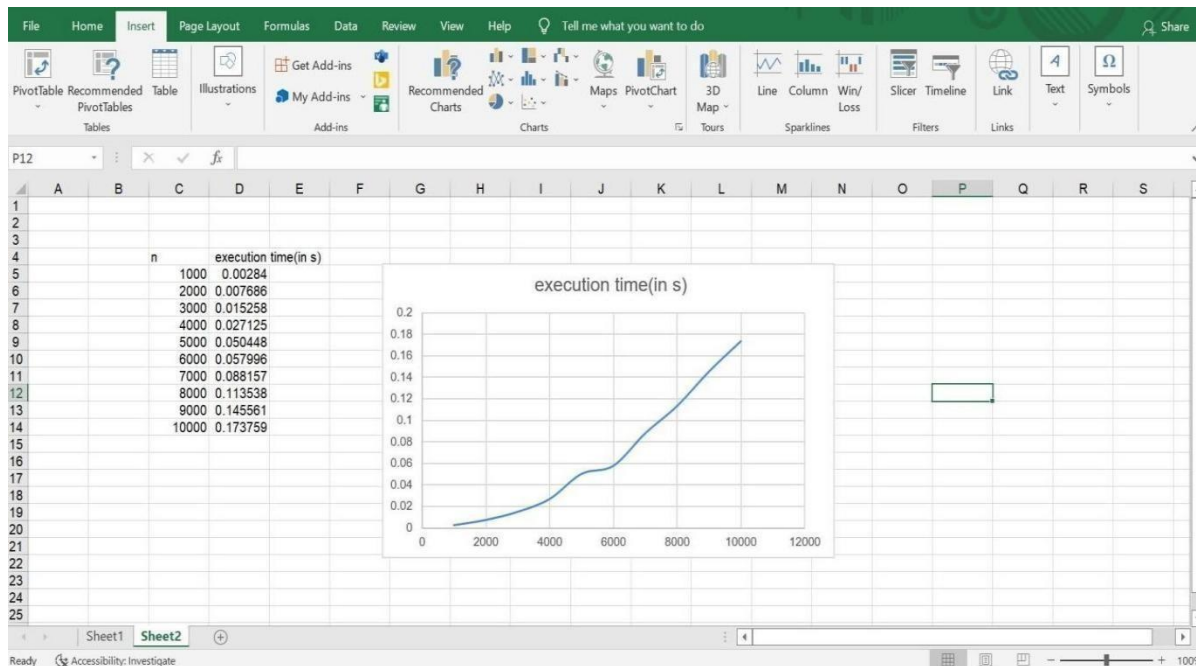
File Home Insert Page Layout Formulas Data Review View Help ☉ Tell me what you want to do   ♀ Share

PivotTable Recommended Table Illustrations   Get Add-ins   Recommended   Maps PivotChart 3D   Line Column Win/ Slicer Timeline   Link   Text Symbols
           PivotTables   My Add-ins   Charts   Map   Loss
   Tables   Add-ins   Charts   Tours   Sparklines   Filters   Links

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2																			
3																			
4			n	execution time(in s)															
5			1000	0.00284															
6			2000	0.007686															
7			3000	0.015258															
8			4000	0.027125															
9			5000	0.050448															
10			6000	0.057996															
11			7000	0.088157															
12			8000	0.113538															
13			9000	0.145561															
14			10000	0.173759															

execution time(in s)

Sheet1 Sheet2 (+)

Ready   ♿ Accessibility: Investigate

LAB PROGRAM-04

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[10][10],n;
void bfs(int); int
main()
{ int i,j,src;

    printf("\n enter the no of nodes:\t");
    scanf("%d",&n); printf("\n
    enter the adjacency
    matrix:\n"); for(i=1;i<=n;i++)
   { for(j=1;j<=n;j++)
     {
        scanf("%d",&a[i][j]);
     } }
   printf("\nenter the source
```

```
      node:\t"); scanf("%d",&src);
      bfs(src);

    return 0; }

    void bfs(int src)
   { int q[10],f=0,r=-
     1,vis[10],i,j;
     for(j=1;j<=n;j++)
     { vis[j]=0;
     } vis[src]=1;
     r=r+1;
     q[r]=src;
     while(f<=r)
   { i=q[f]; f=f+1;
 for(j=1;j<=n;j++) {
 if(a[i][j]==1&&vis[j]!=1)
 {
  vis[j]=1;
  r=r+1;
  q[r]=j;
 }
 } }
 for(j=1;j<=n;j++
 )
 {
 if(vis[j]!=1)
 { printf("\nnode %d is not
 reachable\n",j);
 }
 else
 { printf("\nnode %d is
 reachable\n",j);
 }
 }
 }
```

OUTPUT:

```
C:\TURBOC3\BIN\BFSS.exe                                    —    □    ×

enter the no of nodes:  6

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

enter the source node:  1

node 1 is reachable

node 2 is reachable

node 3 is reachable

node 4 is reachable

node 5 is reachable

node 6 is reachable

Process returned 0 (0x0)   execution time : 32.448 s
Press any key to continue.
```

b.

```c
#include<stdio.h>
#include<conio.h>
int a[10][10],n,vis[10];
int dfs(int); void
main()
{ int i,j,src,ans;
    for(j=1;j<=n;j++)
    { vis[j]=0;
    }
  printf("\nenter the no of nodes:\t");
  scanf("%d",&n); printf("\nenter the
  adjacency matrix:\n"); for(i=1;i<=n;i++)
  { for(j=1;j<=n;j++)
    {
```
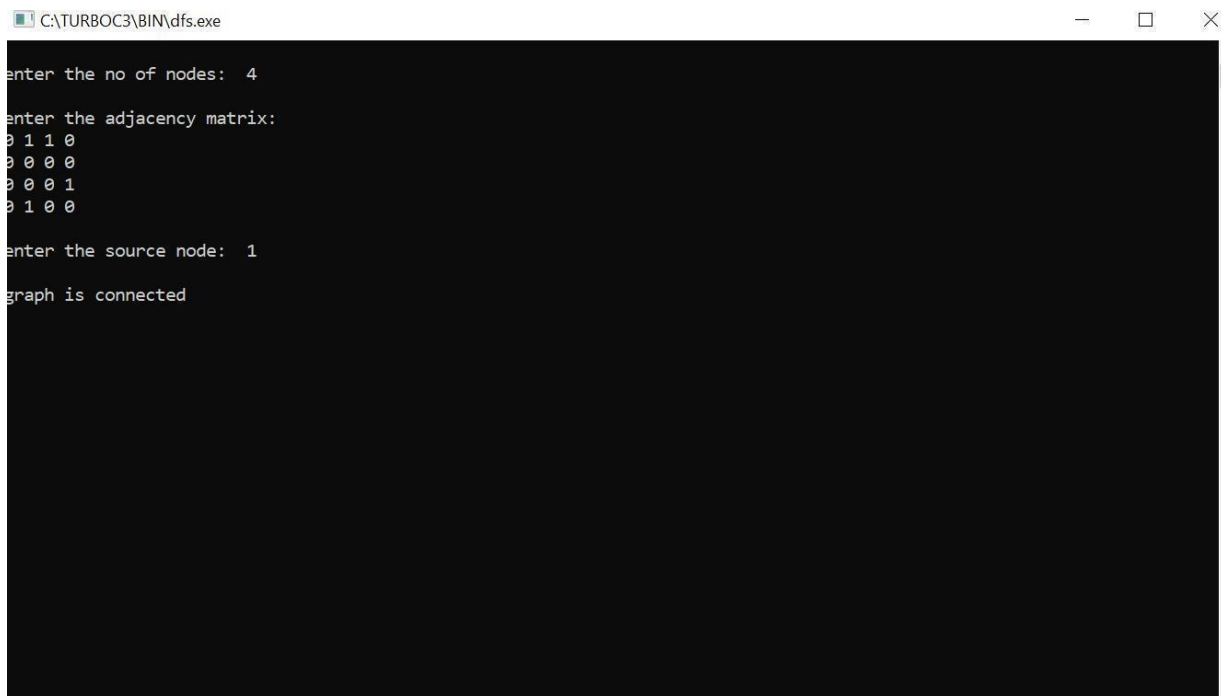
```c
        scanf("%d",&a[i][j])

        ;

      }

    }

   printf("\nenter the source

  node:\t"); scanf("%d",&src);

  ans=dfs(src); if(ans==1)

  {

     printf("\ngraph is connected\n");

  }

  else

  {

   printf("\ngragh is not connected\n");

  }

  getch();

} int dfs(int src)

{ int j; vis[src]=1; for(j=1;j<=n;j++

)

{

 if(a[src][j]==1&&vis[j]!=1)

 {

  dfs(j);

 }

}
```

```
for(j=1;j<=n;j++)

{ if(vis[j]!=1)

  { return

    0;

   }

} return

1;

}
```

OUTPUT:


```
C:\TURBOC3\BIN\dfs.exe

enter the no of nodes:  4

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

enter the source node:  1

graph is connected
```

```
enter the no of nodes:  4

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

enter the source node:  1

gragh is not connected

Process returned 13 (0xD)   execution time : 25.246 s
Press any key to continue.
```

LAB PROGRAM-05

Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator.

```c
#include<stdio.h>

#include<conio.h
>

#include<time.h> void
insertionsort(int n,int a[])
{ int i,j,val,temp;
   for(i=1; i<n;
   i++)
   { val=a[i]; j=i-1;
   while(j>=0 && a[j]>val)
   {
      temp=a[j+1];
      a[j+1]=a[j];
      a[j]=temp;
      j--;
   }
   a[j+1]=val;
   }
}
```

```c
void main()
{ clock_t start,end; int
    a[15500],i,j,temp; int
    n=100;
    while(n<1300)
    { for(i=0; i<n; i++) {
        a[i]=n-i; }
        start=clock();
        insertionsort(n,a)
        ; 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+100;
    }
}
```
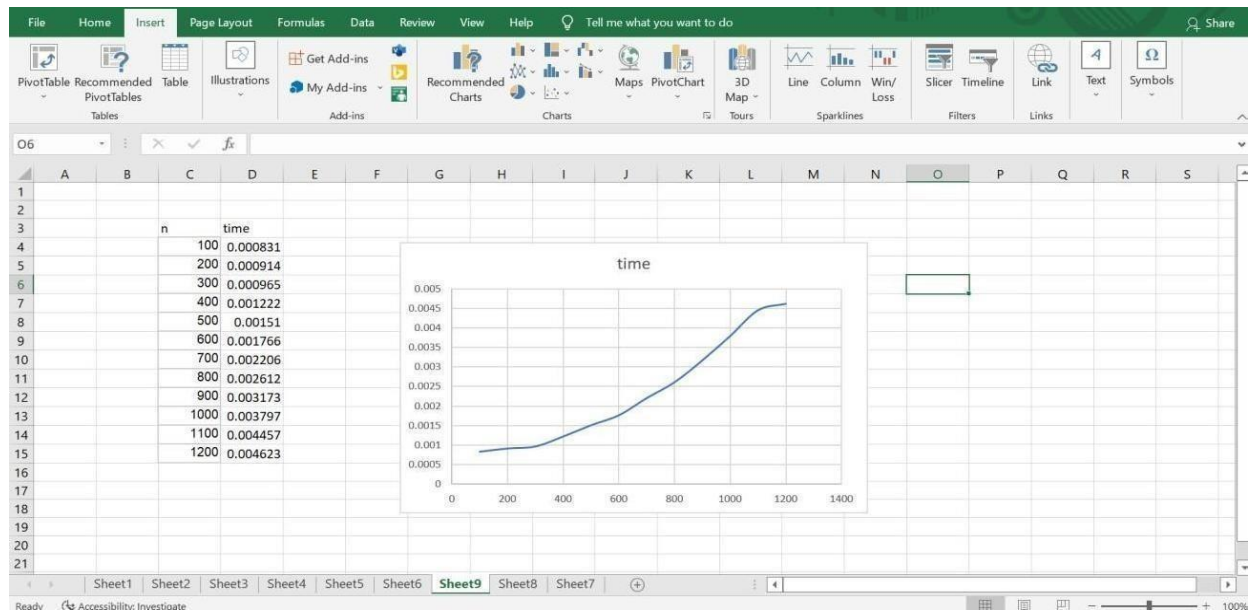
Output:

```
Time taken to sort 100 numbers is 0.000831 Secs
Time taken to sort 200 numbers is 0.000914 Secs
Time taken to sort 300 numbers is 0.000965 Secs
Time taken to sort 400 numbers is 0.001222 Secs
Time taken to sort 500 numbers is 0.001510 Secs
Time taken to sort 600 numbers is 0.001766 Secs
Time taken to sort 700 numbers is 0.002206 Secs
Time taken to sort 800 numbers is 0.002612 Secs
Time taken to sort 900 numbers is 0.003173 Secs
Time taken to sort 1000 numbers is 0.003797 Secs
Time taken to sort 1100 numbers is 0.004457 Secs
Time taken to sort 1200 numbers is 0.004623 Secs

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


n	time
100	0.000831
200	0.000914
300	0.000965
400	0.001222
500	0.00151
600	0.001766
700	0.002206
800	0.002612
900	0.003173
1000	0.003797
1100	0.004457
1200	0.004623

LAB PROGRAM-06

Write program to obtain the Topological ordering of vertices in a given digraph. #include<stdio.h>

void dfs(int);

int a[10][10],n,e[10],vis[10],j=0;

int main()

```c
{ int m, u, v, i; printf("Enter number
   of vertices :
   "); scanf("%d",&n);
   for(i=1;i<=n;i++)
   { for(j = 1; j<= n; j ++)
      { a[i][j] = 0;

      }

   }
   printf("Enter number of edges : ");
   scanf("%d",&m); for(i=1;i<=m;i++)

   {
      printf("Enter an edge : ");
      scanf("%d%d",&u,&v)
   ; a[u][v] = 1; } for(i=1;i<=n;i++)

      vis[i] = 0;
   j=0;
   for(i=1;i<=n;i++)
   { if(vis[i] == 0)

         dfs(i);

   } printf("Topological order :
   "); for(i=n-1; i>=0;i--)
      printf("%d ", e[i]);

   return 0; }
void dfs(int
```
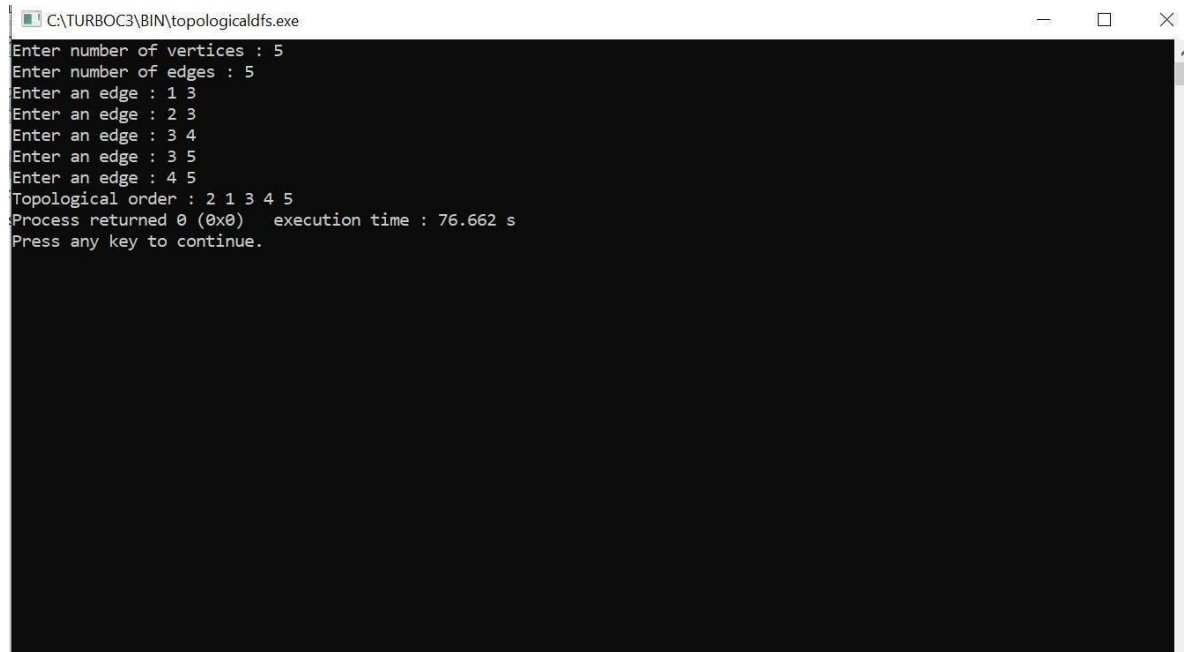
v)

{

   int i; vis[v] = 1;

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

   {

      if(a[v][i] == 1 && vis[i] ==
         0) dfs(i);
   } e[j++] = v;

}

Output:

```
C:\TURBOC3\BIN\topologicaldfs.exe                                    —    □    ✕

Enter number of vertices : 5
Enter number of edges : 5
Enter an edge : 1 3
Enter an edge : 2 3
Enter an edge : 3 4
Enter an edge : 3 5
Enter an edge : 4 5
Topological order : 2 1 3 4 5
Process returned 0 (0x0)    execution time : 76.662 s
Press any key to continue.
```

LAB PROGRAM-07

Implement Johnson Trotter algorithm to generate permutations.

```c
#include<stdio.h>

#include<conio.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;
} int getMobile(int a[], int dir[], int
 n) {int mobile_prev = 0, mobile = 0;
 for (int i = 0; i < n; i++) {


   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;
    }


   }
   if (dir[a[i]-1] == LEFT_TO_RIGHT && i!=n-1) {
if (a[i] > a[i+1] && a[i] > mobile_prev)
```

```c
            { 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);

if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT)
{ printf("\n");
  int temp;
   temp = a[pos-1] ; a[pos-1]
   = a[pos-2]; a[pos-
   2]= temp;
}
```

```c
else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
{
printf("\n");
    int temp; temp =
    a[pos] ; a[pos] = a[pos-
    1]; a[pos-
    1]= temp;
} 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]);
} int fact(int n) {
int res = 1; int i;
```

```c
    for (i = 1; i <= n;
    i++) res = res *
    i; return res;
    }
    void printPermutation(int n)
    {

    int a[n]; int
    dir[n];
    printf("\n");

    printf("\n"); for (int i
    = 0; i < n; i++)
    { a[i] = i +
    1;
    printf("%d
    \n", a[i]);
    printf("\n
    "); }
    printf("\n
    "); for (int i
    = 0; i < n;
    i++) dir[i] =
    RIGHT_TO_
    LEFT;
```

```c
for (int i = 1; i < fact(n); i++)


printOnePerm(a, dir, n);

printf("\n");

} int

main() { int n; printf("\n

Enter the value of n:N");

scanf("%d",&n);

printf("\n");

printPermutation(n);

printf("\n"); return 0;

}
```
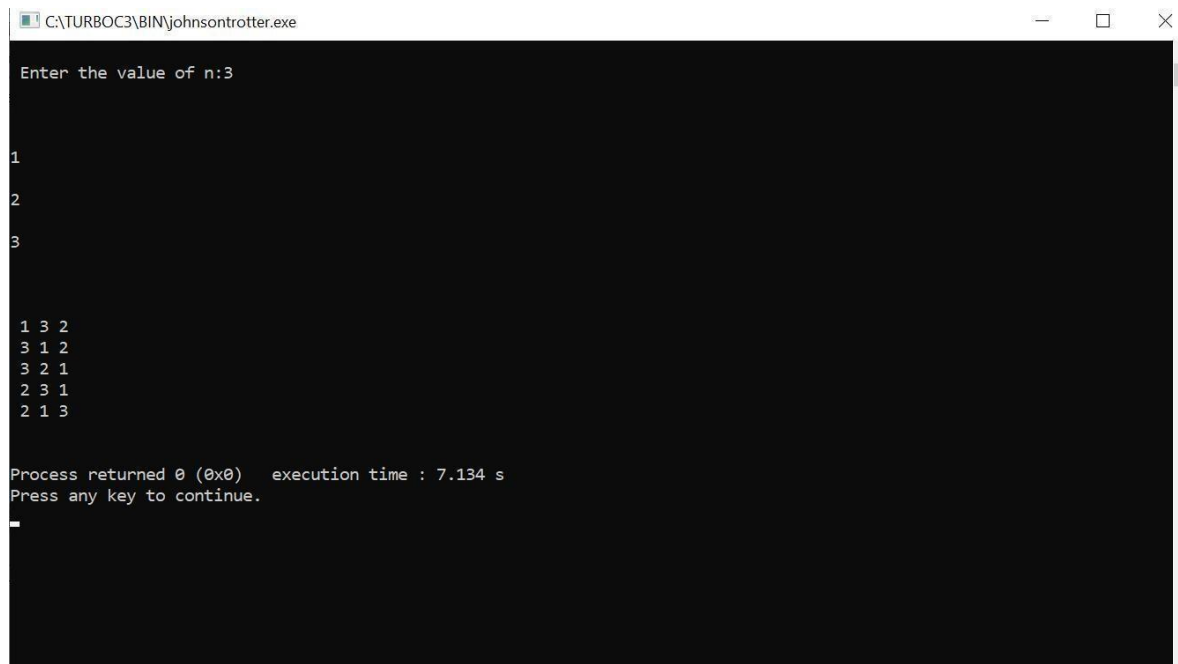
Output:

LAB PROGRAM-08

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.

```c
#include<stdio.h>

#include<time.h>

#include<stdlib.h> 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]);

}
```

```c
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)(endstart))/CLOCKS_PER_SEC));

break;

case 2: n=500;
while(n<=14500)
{
for(i=0;i<n;i++)
{

a[i]=n-i; }
start=clock()
; split(a,0,n-
1)
;
```

```c
for(j=0;j<500000;j++){ temp=38/600;} end=clock(); printf("\n Time
taken to sort %d numbers is %f Secs",n,
(((double)(endstart))/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];

}

}
```

Output:

```
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:6

Enter array elements:12 66 75 2 68 44

Sorted array is:2        12       44       66       68       75
  Time taken to sort 6 numbers is 0.000004 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:
```

```
                              input
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 0.001274 Secs
Time taken to sort 1500 numbers is 0.001298 Secs
Time taken to sort 2500 numbers is 0.001484 Secs
Time taken to sort 3500 numbers is 0.001655 Secs
Time taken to sort 4500 numbers is 0.001914 Secs
Time taken to sort 5500 numbers is 0.002116 Secs
Time taken to sort 6500 numbers is 0.002334 Secs
Time taken to sort 7500 numbers is 0.002520 Secs
Time taken to sort 8500 numbers is 0.002777 Secs
Time taken to sort 9500 numbers is 0.003020 Secs
Time taken to sort 10500 numbers is 0.003200 Secs
Time taken to sort 11500 numbers is 0.003407 Secs
Time taken to sort 12500 numbers is 0.003709 Secs
Time taken to sort 13500 numbers is 0.003902 Secs
Time taken to sort 14500 numbers is 0.003825 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:
```
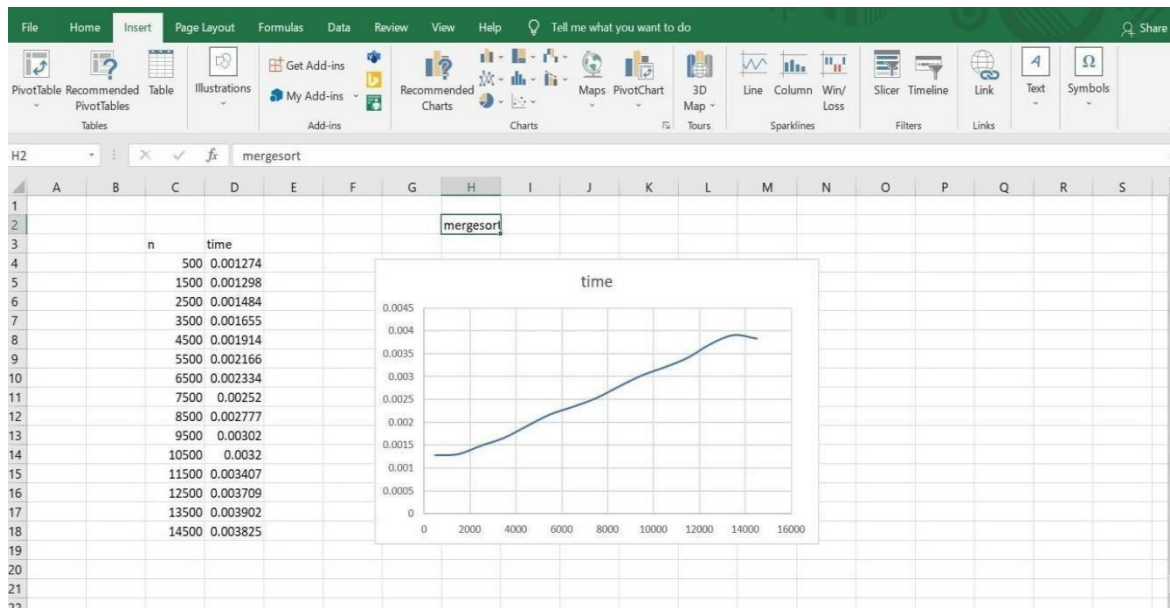
File   Home   Insert   Page Layout   Formulas   Data   Review   View   Help   Q Tell me what you want to do

PivotTable Recommended Table   Illustrations   Get Add-ins   Recommended   Maps PivotChart   3D   Line Column Win/   Slicer Timeline   Link   Text Symbols
           PivotTables   My Add-ins   Charts   Map   Loss
Tables   Add-ins   Charts   Tours   Sparklines   Filters   Links

H2   fx   mergesort

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2								mergesort											
3			n	time															
4			500	0.001274															
5			1500	0.001298															
6			2500	0.001484															
7			3500	0.001655															
8			4500	0.001914															
9			5500	0.002166															
10			6500	0.002334															
11			7500	0.00252															
12			8500	0.002777															
13			9500	0.00302															
14			10500	0.0032															
15			11500	0.003407															
16			12500	0.003709															
17			13500	0.003902															
18			14500	0.003825															
19																			
20																			
21																			

time

LAB PROGRAM-09

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

```c
#include<stdio.h>

#include<time.h>

#include<stdlib.h>

#define MAXINT

2000 void delay(int n)

{ int

i;

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

{

}

} void quickSort(int

number[],int first,int last){int

i,j,pivot,temp; if(first<last){

    pivot=first;

    i=first; j=last;

while(i<j){ while(number[i]<=number[pivot]&

    &i<last){i++;

    }
    while(number[j]>number[pivot]&&j>first)

    { j--;
    } if(i<j){

    temp=number
```

```c
        [i];
        number[i]=number[j]
        ; number[j]=temp;
        } }
temp=number[pivot];
number[pivot]=number[j];
number[j]=temp;
quickSort(number,first,j-
1)
;
quickSort(number,j+1,last)
;
} } void main() {
clock_t start,end; int
i,datasize=1; long int
n=10000; int *a;
while(datasize<=20)
{ a=(int
    *)calloc(n,sizeof(int));
    if(a==NULL){ printf("Insufficiant
    Memory");

    exit(0);
    } for(i=0;i<=n1;i++){
    a[i]=rand()%MAXI NT;
```
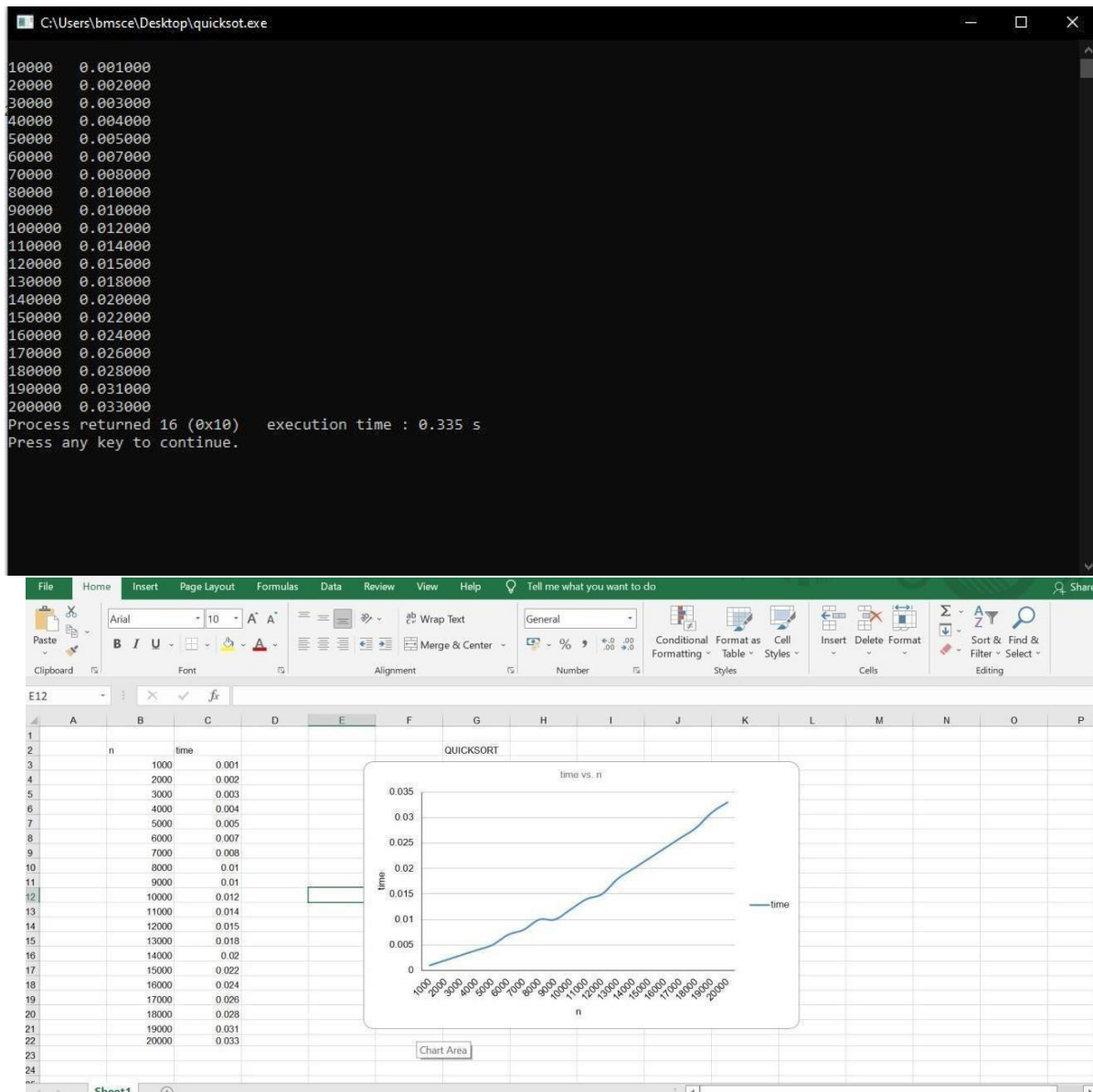
```
} start=clock(); quickSort(a,0,n-
1) ;
end=clock(); free(a); if((end-start)!=0){
printf("\n%d\t%f",n,(double)(end-start)/CLK_
 TCK); datasize++;
} n+=10000;
} return;
}
```

Output:

LAB PROGRAM-10

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

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h> #include
<math.h> void swap(int
*,int *); void heapify(int
[],int,int); void
heapSort(int[], int);

int main()
{ int
   a[15000],n,i,j,ch,temp;
   clock_t start,end;
   while(1)
   { printf("\n 1: For manual entry of N values and array elements:");
      printf("\n 2: To display time taken for sorting number of elements N
      in
the range 500 to 14500:"); printf("\n
      3: To exit"); printf("\n Enter
      your choice:");
      scanf("%d",&ch); switch(ch)
      { case 1: printf("\n Enter the number of elements:");
         scanf("%d",&n); printf("\n Enter
         array elements:");
         for(i=0;i<n;i++) {
         scanf("%d",&a[i]);
         } start=clock(); heapSort(a,
         n); end=clock(); printf("\n
         Sorted array is:"); for(i=n-
         1;i>=0;i--){
         printf("%d\t",a[i]);
         }
         printf("\n Time taken to sort %d numbers is
```

```c
        %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]=n-i;

            }
            start=clock();
            heapSort(a, n);
            for(j=0;j<50000000;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);
      }

    }
}
void swap(int *a, int *b)
{ int temp = *a;
    *a = *b;
    *b = temp;
}
void heapify(int arr[], int n, int i)
{ int largest = i; int left = 2 * i + 1; int right
    = 2 * i + 2; if (left < n && arr[left] >
    arr[largest]) largest = left; if (right < n
    && arr[right] > arr[largest])
        largest = right;
```

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

OUTPUT:

```
1: For manual entry of N values 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:5
Enter array elements:20 31 10 46 78
Sorted array is:78 46  31  20  10
Time taken to sort 5 numbers is 0.000003 secs
1: For manual entry of N values 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 0.105851 secs
Time taken to sort 1500 numbers is 0.103846 secs
Time taken to sort 2500 numbers is 0.103909 secs
Time taken to sort 3500 numbers is 0.105498 secs
Time taken to sort 4500 numbers is 0.104747 secs
Time taken to sort 5500 numbers is 0.106133 secs
Time taken to sort 6500 numbers is 0.105619 secs
Time taken to sort 7500 numbers is 0.105099 secs
Time taken to sort 8500 numbers is 0.105469 secs
Time taken to sort 9500 numbers is 0.105425 secs
Time taken to sort 10500 numbers is 0.106843 secs
```

LAB PROGRAM-11

Implement Warshall's algorithm using dynamic programming

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

int max(int,int);

void warshal(int p[10][10],int n) {
    int i,j,k;
    for (k=1;k<=n;k++)
      for (i=1;i<=n;i++)
       for (j=1;j<=n;j++)
        p[i][j]=max(p[i][j],p[i][k]&&p[k][j]);
}
int max(int a,int b) {
    ;
    if(a>b)
     return(a); else
return(b); } void
main() { int
p[10][10]= { 0
    }
    ,n,e,u,v,i,j; printf("\n Enter the number of vertices:");
    scanf("%d",&n); printf("\n Enter the number of
    edges:"); scanf("%d",&e); for (i=1;i<=e;i++) {
    printf("\n Enter the end vertices of edge %d:",i);
    scanf("%d%d",&u,&v);
        p[u][v]=1;
    } printf("\n Matrix of input
    data:
    \n"); for (i=1;i<=n;i++) { for
    (j=1;j<=n;j++) printf("%d\t",p[i][j]);
        printf("\n"); } warshal(p,n);
    printf("\n Transitive closure:
    \n"); for (i=1;i<=n;i++) { for
```

```
    (j=1;j<=n;j++)
            printf("%d\t",p[i][j]);
        printf("\n");
    }
    getch();
}
```

OUTPUT:

```
/tmp/tIy1e62gPr.o
Enter the number of vertices:5
Enter the number of edges:5
Enter the end vertices of edge 1:2
3
Enter the end vertices of edge 2:3 4
Enter the end vertices of edge 3:5 1
Enter the end vertices of edge 4:2 1
Enter the end vertices of edge 5:3 2
Matrix of input data:
0   0   0   0   0
1   0   1   0   0
0   1   0   1   0
0   0   0   0   0
1   0   0   0   0

 Transitive closure:
0   0   0   0   0
1   1   1   1   0
1   1   1   1   0
0   0   0   0   0
1   0   0   0   0
|
```

LAB PROGRAM-12

Implement 0/1 Knapsack problem using dynamic programming.

```
#include<stdio.h>
void knapsack(); int
max(int,int);
int i,j,n,m,p[10],w[10],v[10][10]; void
main()
{
```

```c
printf("\n enter the no. of items:\t");
scanf("%d",&n);
printf("\n enter the weight of the each item:\n ");
for(i=1;i<=n;i++)
{
scanf("%d",&w[i]);
}
printf("\n enter the profit of each item:\n ");
for(i=1;i<=n;i++)
{
scanf("%d",&p[i]);
}
printf("\n enter 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("\n the output is:\n");
for(i=0;i<=n;i++) { for(j=0;j<=m;j++)
{
```

```c
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 the each item:
 10 20 30 40
 enter the profit of each item:
 20 30 55 33
 enter the knapsack's capacity:  60
 the output is:
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   33

0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   33  33  33  33  33  33  33  33  33  33  33

0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   33
    33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  4
```

LAB PROGRAM-13

Implement All Pair Shortest paths problem using Floyd's algorithm

#include<stdio.h>

int a[10][10],n;
void floyds(); int
min(int,int);
void main()
{ int
i,j;

printf("\n enter the no. of vertices:\t");
scanf("%d",&n);
printf("\n enter the cost matrix:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);

```c
}
} floyds();

} 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("\n all 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:
11 999 999 44
33 44  999 3
33 55 999 555
44 3 6 9
all pair shortest path matrix is:
11   47  50  44

33  6   9   3

33  55  64  58

36  3   6   6
|
```

LAB PROGRAM-14

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

```c
#include<stdio.h> void
prims(); int
c[10][10],n; 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",&c[i][j]);
} } prims();
```

```c
} 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%d-
----
>%d=%d\n",u,
v,min);
elec[v]=1;

ne=ne+1; mincost=mincost+min;
}
c[u][v]=c[v][u]=9999;
}
printf("\nmincost=%d",mincost);
}
```
OUTPUT:

```
enter the no. of vertices:  4
enter the cost matrix:
1 0 0 1
0 2 3 4
0 3 2 4
1 2 0 3
1----->2=0

1----->3=0

1----->4=1

mincost=1
```

LAB PROGRAM-15

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm #include<stdio.h>

void kruskals();
int c[10][10],n;

void main()
{ int
i,j;

printf("\n enter the no. of vertices:\t");
scanf("%d",&n);
```

```c
printf("\n enter 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%d-----> %d=%d\n",a,b,min);
parent[v]=u; ne=ne+1; mincost=mincost+min;
} c[a][b]=c[b][a]=9999;
}
```

```
printf("\n mincost=%d",mincost);
}
```
OUTPUT:

```
/tmp/1d1WsEvHHt.o
enter the no. of vertices:  4
enter the cost matrix:
1 0 0 1
2 0 0 2
2 3 4 9
1 3 4 5
1-----> 2=0

1-----> 3=0

1-----> 4=1

 mincost=1
```

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> #define infinity 999 void
dij(int n,int v,int cost[10][10],int dist[100])
{ int
i,u,count,w,flag[10],min;
for(i=1;i<=n;i++)
flag[i]=0,dist[i]=cost[v][i];
```

```
count=2;
while(count<=n)
{ min=99; for(w=1;w<=n;w++)
if(dist[w])
min=dist[w],u=w; flag[u]=1;
count++;
for(w=1;w<=n;w++) if(dist[u]+cost[u][w])
dist[w]=dist[u]+cost[u][w];
}
}
void main()
{ int
n,v,i,j,cost[10][10],dist[10];

printf("\n Enter the number of nodes:");
scanf("%d",&n); printf("\n Enter the
cost matrix:\n"); for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{ scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=infinity;
}
printf("\n Enter the source matrix:");
scanf("%d",&v); dij(n,v,cost,dist); printf("\n
Shortest path:\n"); for(i=1;i<=n;i++)
if(i!=v) printf("%d-
>%d,cost=%d\n",v,i,dist[i]);
}
```

OUTPUT:

```
/tmp/RHJFW6kRtE.o
Enter the number of nodes:5
Enter the cost matrix:
2 3 4 999 999
999 4 5 999 999
3 45 999 2 5
999 5 6 7 999
5 6 8 999 999
Enter the source matrix:1
Shortest path:
1->2,cost=4002
1->3,cost=4004
1->4,cost=4995
1->5,cost=4995
```