VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

ANKITH S (1BM20CS017)

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 August-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 **ANKITH S (1BM20CS017),** 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 **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:
Dr. Nagarathna N
Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

16	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	**42**
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.	
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.

Write a recursive program to Solve
a) Towers-of-Hanoi problem b) To find GCD

a) TOWER OF HANOI

```
#include <stdio.h>
void toh(int n,char a,char b,char c)
{

   if(n>0)
    {
        toh(n-1,a,c,b);
        printf("move the disk %d from %c to %c\n",n,a,c);
        toh(n-1,b,a,c);
    }
}
int main()
{
    int n;
    char a,b,c;
    printf("Enter the number of disks: ");
    scanf("%d",&n);
    toh(n,'a','b','c');
    return 0;
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith S\Dropbox
Hanoi }
Enter the number of disks: 4
move the disk 1 from a to b
move the disk 2 from a to c
move the disk 1 from b to c
move the disk 3 from a to b
move the disk 1 from c to a
move the disk 2 from c to b
move the disk 1 from a to b
move the disk 4 from a to c
move the disk 1 from b to c
move the disk 2 from b to a
move the disk 1 from c to a
move the disk 3 from b to c
move the disk 1 from a to b
move the disk 2 from a to c
move the disk 1 from b to c
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

b) GREATEST COMMON DIVISOR

```
#include <stdio.h>

int gcd(int a,int b) {
     if(b!=0)
        return gcd(b,a%b);
     else
        return a;
}
void main()
{
    int a,b,c;
    printf("Enter two numbers: ");
    scanf("%d %d",&a,&b);
    c=gcd(a,b);
    printf("The gcd of two numbers is %d\n",c);
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith
Enter two numbers: 3 17
The gcd of two numbers is 1
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith
Enter two numbers: 4 16
The gcd of two numbers is 4
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith
Enter two numbers: 12 38
The gcd of two numbers is 2
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

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.

CODE:

```
#include <stdio.h>;
#include <time.h>;
#include <stdlib.h>;
int bin_srch(int[], int, int, int);
int lin_srch(int[], int, int, int);
int n, a[1000000];

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 <= 7000)
            {
                for (i = 0; i < n; i++)
                    a[i] = i;
                key = a[n - 1];
                start = clock();
                search_status = bin_srch(a, 0, n - 1, key);
                end = clock();
                if (search_status == -1)

                    printf("\nKey Not Found");
                else
                    printf("\n Key found at position %d",
search_status);
                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 <= 7000)
            {
                for (i = 0; i < n; i++)
                    a[i] = i;
                key = a[n - 1];
                start = clock();
                search_status = lin_srch(a, 0, n - 1, key);
                end = clock();
                if (search_status == -1)
                    printf("\nKey Not Found");
                else
                    printf("\n Key found at position %d",
search_status);

                printf("\nTime for n=%d is %f Secs", n,
(double)(end - start) / CLOCKS_PER_SEC);
```

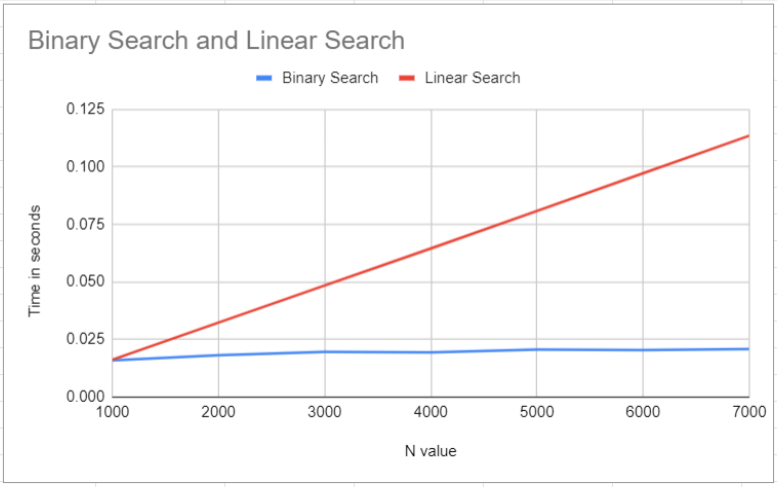
```
                n = n + 1000;
            }
            break;
        default:
            exit(0);
        }
        getchar();
    }
}
int bin_srch(int a[], int low, int high, int key)
{
    for (int j = 0; j < 1000000; j++)
        ;
    int mid;
    if (low > high)
        return -1;
    mid = (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)
{
    for (int j = 0; j < 10000; j++)
    {
        int temp = 38 / 600;
    }
    if (i > high)
        return -1;
    if (key == a[i])
        return i;
    else
        return lin_srch(a, i + 1, high, key);
}
```

OUTPUT:

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

Enter your choice:      1

 Key found at position 999
Time for n=1000 is 0.015930 Secs
 Key found at position 1999
Time for n=2000 is 0.018144 Secs
 Key found at position 2999
Time for n=3000 is 0.019664 Secs
 Key found at position 3999
Time for n=4000 is 0.019376 Secs
 Key found at position 4999
Time for n=5000 is 0.020621 Secs
 Key found at position 5999
Time for n=6000 is 0.020405 Secs
 Key found at position 6999
Time for n=7000 is 0.020836 Secs
1: Binary search         2: Linear search        3: Exit

Enter your choice:      2

 Key found at position 999
Time for n=1000 is 0.016221 Secs
 Key found at position 1999
Time for n=2000 is 0.032329 Secs
 Key found at position 2999
Time for n=3000 is 0.048496 Secs
 Key found at position 3999
Time for n=4000 is 0.064565 Secs
 Key found at position 4999
Time for n=5000 is 0.080881 Secs
 Key found at position 5999
Time for n=6000 is 0.097263 Secs
 Key found at position 6999
Time for n=7000 is 0.113580 Secs
1: Binary search         2: Linear search        3: Exit

Enter your choice:      3


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

GRAPH:

	Time Taken	
N value	Binary Search	Linear Search
1000	0.01593	0.016221
2000	0.018144	0.032329
3000	0.019664	0.048496
4000	0.019376	0.064565
5000	0.020621	0.080881
6000	0.020405	0.097263
7000	0.020836	0.11358



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.

CODE:

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

void sort(int x){
    int n=x;
    int a[n],max,i,j,k;
    for(i=0;i<n;i++)
    a[i]=i+1;
    double start,end;
    start = clock();
    for(i=0;i<(n-1);i++){
        max=a[i];
        for(j=(i+1);j<n;j++){
            if(max<a[j]){
                max=a[j];
```

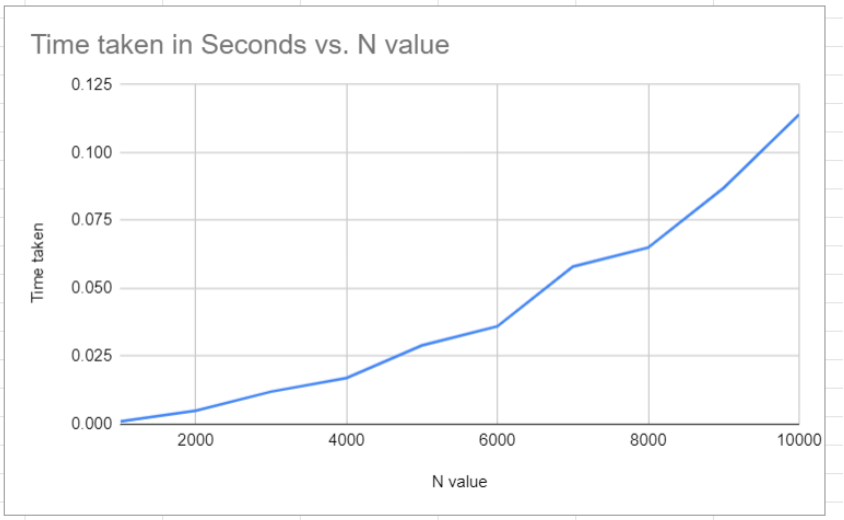
```
                k=j;
            }
        }
        if(a[i]!=max){
            int temp=a[i];
        a[i]=a[k];
        a[k]=temp;
        }
    }
    end = clock();
    printf("Time taken to sort %d numbers is %f seconds
\n",n,(end-start)/CLOCKS_PER_SEC);

    n=n+1000;
}
void main(){
    for(int x=1000;x<=10000;x+=1000){
    sort(x);}
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith S\Dro
tionsort }
Time taken to sort 1000 numbers is 0.001000 seconds
Time taken to sort 2000 numbers is 0.005000 seconds
Time taken to sort 3000 numbers is 0.012000 seconds
Time taken to sort 4000 numbers is 0.017000 seconds
Time taken to sort 5000 numbers is 0.029000 seconds
Time taken to sort 6000 numbers is 0.036000 seconds
Time taken to sort 7000 numbers is 0.058000 seconds
Time taken to sort 8000 numbers is 0.065000 seconds
Time taken to sort 9000 numbers is 0.087000 seconds
Time taken to sort 10000 numbers is 0.114000 seconds
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

GRAPH:

N value	Time taken
1000	0.001
2000	0.005
3000	0.012
4000	0.017
5000	0.029
6000	0.036
7000	0.058
8000	0.065
9000	0.087
10000	0.114



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) BREADTH FIRST SEARCH

CODE:

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

int a[15][15], n;
void bfs(int);

void main()
{
    int i, j, src;
    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++)
            scanf("%d", &a[i][j]);
    printf("\nEnter the source node:\t");
    scanf("%d", &src);
    bfs(src);
}
void bfs(int src)
{
    int q[15], f = 0, r = -1, vis[15], 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", j);
        else
            printf("\nNode %d is reachable", j);
    }
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith S\D

Enter the no of nodes:  5

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

Enter the source node:  1

Node 1 is reachable
Node 2 is reachable
Node 3 is reachable
Node 4 is not reachable
Node 5 is reachable
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

b)DEPTH FIRST SEARCH

CODE:

```
#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++)
            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:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\

Enter the no of nodes:  5

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

Enter the source node:  1

Gragh is not connected
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

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

CODE:

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

void insertionSort(int arr[], int n)
{
     int i, key, j;
     for (i = 1; i < n; i++)
     {
          key = arr[i];
          j = i - 1;
          while (j >= 0 && arr[j] > key)
          {
                    for(int k=0;k<100000;k++);
               arr[j + 1] = arr[j];
               j = j--;
          }
          arr[j + 1] = key;
     }
}
void main() {
  int i, n;
    clock_t start, end;
    printf("ENTER ARRAY SIZE =");
    scanf("%d", &n);
    int arr[150000];
    for (int j = 0; j < n; j++)
        arr[j] = rand()%10000;
    for (i = 0; i < n; i++)
        printf(" %d", arr[i]);
    printf("\n");
    start = clock();
    insertionSort(arr, n);
    end = clock();
```

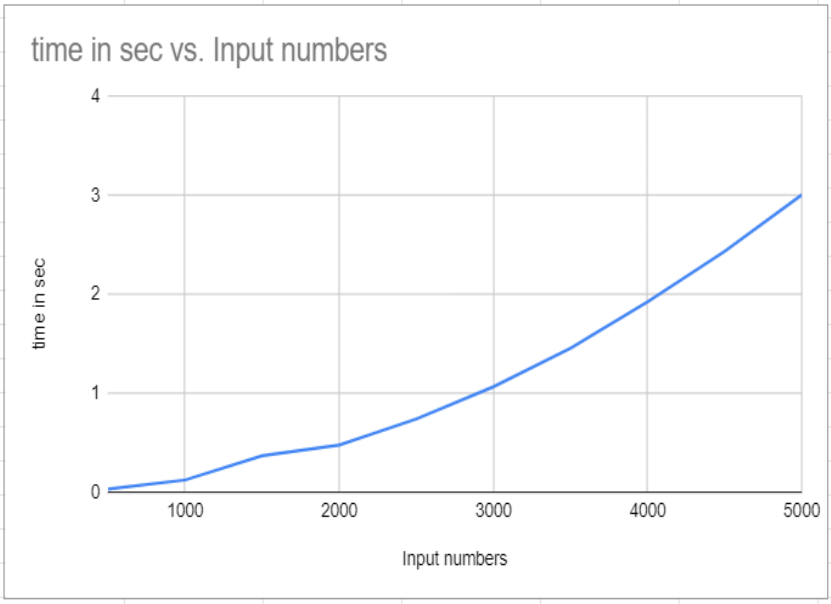
```
    printf("\nSorted elements = ");
    for (i = 0; i < n; i++)
        printf(" %d", arr[i]);
    printf("\n Time taken to sort  %d Numbers is %f secs",
n, (((double)(end - start)) / CLOCKS_PER_SEC));
}
```

OUTPUT:

```
Select "D:\ENGINEERING\4th sem\ada\lab\gcd.exe"
ENTER ARRAY SIZE =10
 41 8467 6334 6500 9169 5724 1478 9358 6962 4464

Sorted elements =  41 1478 4464 5724 6334 6500 6962 8467 9169 9358
 Time taken to sort  10 Numbers is 0.002000 secs
Process returned 49 (0x31)   execution time : 4.747 s
Press any key to continue.
```

GRAPH:

Input numbers	time in sec
500	0.031
1000	0.122
1500	0.367
2000	0.476
2500	0.74
3000	1.065
3500	1.454
4000	1.925
4500	2.434
5000	3.006



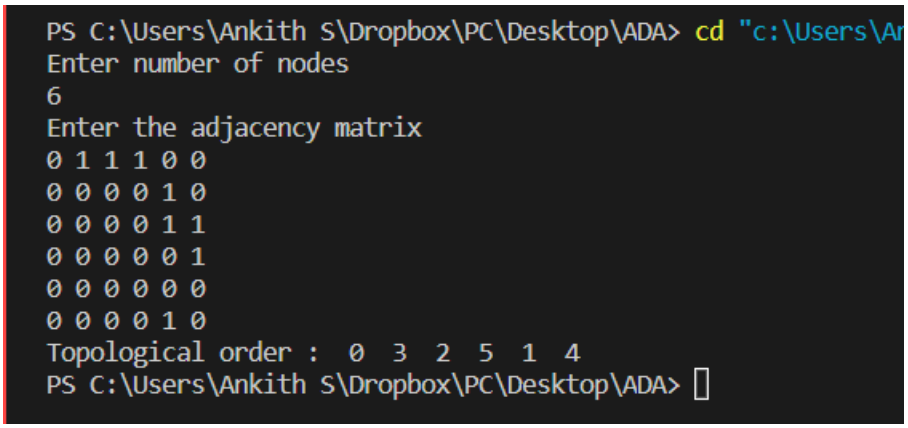
6. Write program to obtain the Topological ordering of vertices in a given digraph.

CODE:

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

void source_removal(int n, int a[10][10]) {
    int i,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;
            }
        }
    }
    printf("Topological order :");
    for(i=0;i<n;i++)
        printf("  %d", t[i]);
}

void main() {
```

```
int i,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();
}
```

OUTPUT:



7. Implement Johnson Trotter algorithm to generate permutations.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int flag = 0;

int swap(int *a,int *b) {
 int t = *a;
 *a = *b;
 *b = t;
}
int search(int arr[],int num,int mobile)
```

```
{
 int g;
 for(g=0;g<num;g++) {
 if(arr[g] == mobile)
    return g+1;
 else
    flag++;
 }
 return -1;
}

int find_Moblie(int arr[],int d[],int num)
{
 int mobile = 0;
 int mobile_p = 0;
 int i;
 for(i=0;i<num;i++)
 {
 if((d[arr[i]-1] == 0) && i != 0)
 {
 if(arr[i]>arr[i-1] && arr[i]>mobile_p)
 {
 mobile = arr[i];
 mobile_p = mobile;
 }
 else
    flag++;
 }
 else if((d[arr[i]-1] == 1) & i != num-1)
{
 if(arr[i]>arr[i+1] && arr[i]>mobile_p)
{
 mobile = arr[i];
 mobile_p = mobile;
 }
 else
    flag++;
 }
 else
    flag++;
 }
```

```
 if((mobile_p == 0) && (mobile == 0))
 return 0;
 else
 return mobile;
}
void permutations(int arr[],int d[],int num)
{
 int i;
 int mobile = find_Moblie(arr,d,num);
 int pos = search(arr,num,mobile);
 if(d[arr[pos-1]-1]==0)
 swap(&arr[pos-1],&arr[pos-2]);
 else
 swap(&arr[pos-1],&arr[pos]);
 for(int i=0;i<num;i++)
 {
 if(arr[i] > mobile)
 {
 if(d[arr[i]-1]==0)
 d[arr[i]-1] = 1;
 else
 d[arr[i]-1] = 0;
 }
 }
 for(i=0;i<num;i++)
 {
 printf(" %d ",arr[i]);
 } }

int factorial(int k)
{
 int f = 1;
 int i = 0;
 for(i=1;i<k+1;i++)
    f = f*i;
 return f;
}
int main()
{
 int num = 0;
 int i;
```

```
 int j;
 int z = 0;
 printf("Johnson trotter algorithm to find all permutations
of given numbers \n");
 printf("Enter the number\n");
 scanf("%d",&num);
 int arr[num],d[num];
 z = factorial(num);
 printf("total permutations = %d",z);
 printf("\nAll possible permutations are: \n");
 for(i=0;i<num;i++)
 {
 d[i] = 0;
 arr[i] = i+1;
 printf(" %d ",arr[i]);
 }
 printf("\n");
 for(j=1;j<z;j++) {
     permutations(arr,d,num);
     printf("\n");
    }
return 0;
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Anki
Johnson trotter algorithm to find all permutations of given nu
Enter the number
3
total permutations = 6
All possible permutations are:
 1  2  3
 1  3  2
 3  1  2
 3  2  1
 2  3  1
 2  1  3
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

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.

CODE:

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

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
clock_t start,end;
int a[30000],n=500,i;
while(n<=5000){
for(i=0;i<n;i++)
{
  a[i] = rand()%1000;
}
start = clock();
mergesort(a,0,n-1);
end = clock();
printf("\n To Sort array of %d numbers ",n);
printf("required  time is %lf secs",(double)(end-
start)/CLOCKS_PER_SEC);
printf("\n");
n+=500;
}
}

void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
```

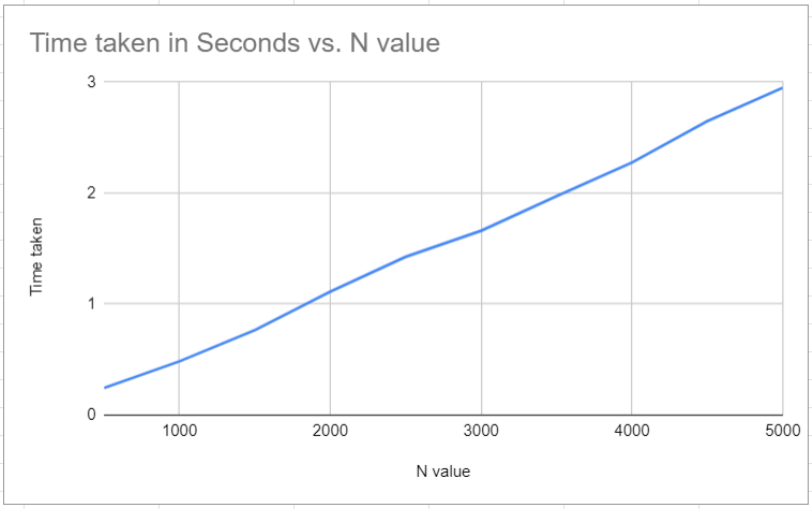
```
mid=(i+j)/2;
mergesort(a,i,mid);
mergesort(a,mid+1,j);
merge(a,i,mid,mid+1,j);
}
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
int temp[30000];
int i,j,k;
i=i1;
j=i2;
k=0;
while(i<=j1 && j<=j2)
{
    for(int j=0;j<100000;j++);
if(a[i]<a[j])
temp[k++]=a[i++];
else
temp[k++]=a[j++];
}
while(i<=j1)
temp[k++]=a[i++];
while(j<=j2)
temp[k++]=a[j++];

for(i=i1,j=0;i<=j2;i++,j++){
     a[i]=temp[j];
     }
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith S\Dropb

 To Sort array of 500 numbers required  time is 0.244000 secs
 To Sort array of 1000 numbers required  time is 0.484000 secs
 To Sort array of 1500 numbers required  time is 0.766000 secs
 To Sort array of 2000 numbers required  time is 1.112000 secs
 To Sort array of 2500 numbers required  time is 1.426000 secs
 To Sort array of 3000 numbers required  time is 1.661000 secs
 To Sort array of 3500 numbers required  time is 1.973000 secs
 To Sort array of 4000 numbers required  time is 2.275000 secs
 To Sort array of 4500 numbers required  time is 2.648000 secs
 To Sort array of 5000 numbers required  time is 2.949000 secs
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

GRAPH:

N value	Time taken
500	0.244
1000	0.484
1500	0.766
2000	1.112
2500	1.426
3000	1.661
3500	1.973
4000	2.275
4500	2.648
5000	2.949



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

CODE:

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

```
void quicksort(int number[5000],int first,int last)
{
   int i, j, pivot, temp;
   if(first<last)
 {
      pivot=first;
      i=first;
      j=last;
      while(i<j)
   {
         for(int x=0;x<10000000;x++);
              while(number[i]<=number[pivot]&&i<last)
                        i++;
         while(number[j]>number[pivot])
                    j--;
         if(i<j)
    {
            temp=number[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);
   }
}
int main()
{
   clock_t start,end;
   int i, count, number[5000];
   printf("No. of elements: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
 {
       number[i]=rand()%5000;
   }
```

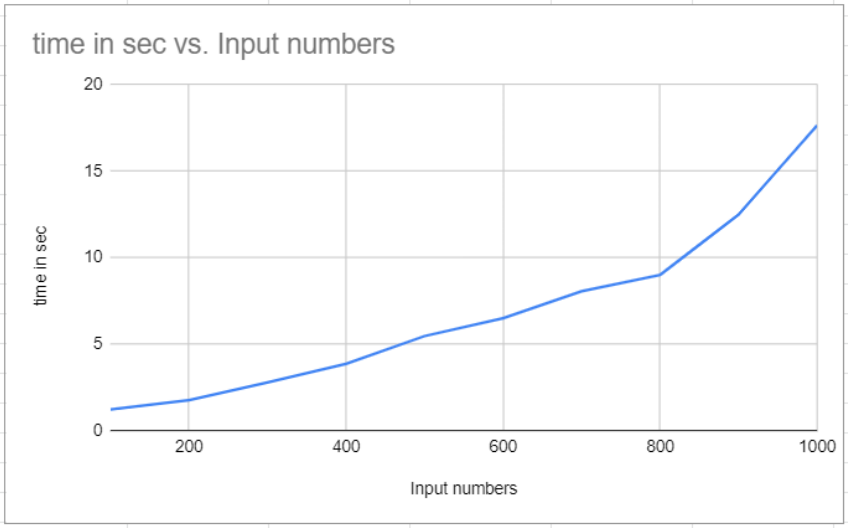
```
   start = clock();
   quicksort(number,0,count-1);
   end = clock();
   printf("Order of Sorted elements: ");
   for(i=0;i<count;i++)
 {
        printf(" %d",number[i]);
 }
   printf("\nSeconds taken %lf",(double)(end-
start)/CLOCKS_PER_SEC);
   return 0;
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA> cd "c:\Users\Ankith S\Drop
box\PC\Desktop\ADA\" ; if ($?) { gcc 09_quicksort.c -o 09_quicksort } ;
 if ($?) { .\09_quicksort }
No. of elements: 100
Enter 100 elements: Order of Sorted elements:  6 37 41 106 141 153 288
292 333 350 436 447 491 537 547 667 705 724 778 890 1101 1118 1299 1308
 1322 1323 1334 1478 1500 1538 1538 1541 1726 1729 1827 1840 1842 1868
1869 1942 1944 1962 2035 2082 2190 2316 2376 2382 2391 2421 2439 2446 2
529 2623 2644 2648 2662 2673 2711 2757 2859 2929 2995 3035 3145 3253 32
81 3467 3548 3703 3716 3723 3756 3805 3811 3902 3931 3942 4040 4084 416
9 4264 4358 4370 4393 4464 4604 4626 4629 4664 4718 4741 4771 4827 4894
 4895 4912 4954 4961 4966
Seconds taken 1.135000
PS C:\Users\Ankith S\Dropbox\PC\Desktop\ADA>
```

GRAPH:

Input numbers	time in sec
100	1.21
200	1.757
300	2.768
400	3.85
500	5.458
600	6.495
700	8.05
800	8.996
900	12.487
1000	17.647



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

CODE:

```
#include<stdio.h>
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    int temp;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        temp=arr[i];
        arr[i]=arr[largest];
        arr[largest]=temp;;
        heapify(arr, n, largest);
    }
}
```

```
void heapSort(int arr[], int n)
{
    int i,j,temp;
    for ( i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (j=n-1; j>0; j--)
    {
        temp=arr[0];
        arr[0]=arr[j];
        arr[j]=temp;
        heapify(arr, j, 0);
    }
}
void printArray(int arr[], int n)
{
    int i;
    for ( i=0; i<n; ++i)
        printf("%d  ",arr[i]);
    printf("\n");
}
int main()
{
    int arr[10],i,n;
    printf("Enter number of elements\n");
    scanf("%d",&n);
    printf("enter array elements\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    heapSort(arr, n);
    printf("Sorted array is \n");
    printArray(arr, n);
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming
.c -o hello } ; if ($?) { .\hello }
Enter number of elements
6
enter array elements
7 9 3 5 1 6
Sorted array is
1  3  5  6  7  9
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming
```

11. Implement Warshall's algorithm using dynamic programming

CODE:

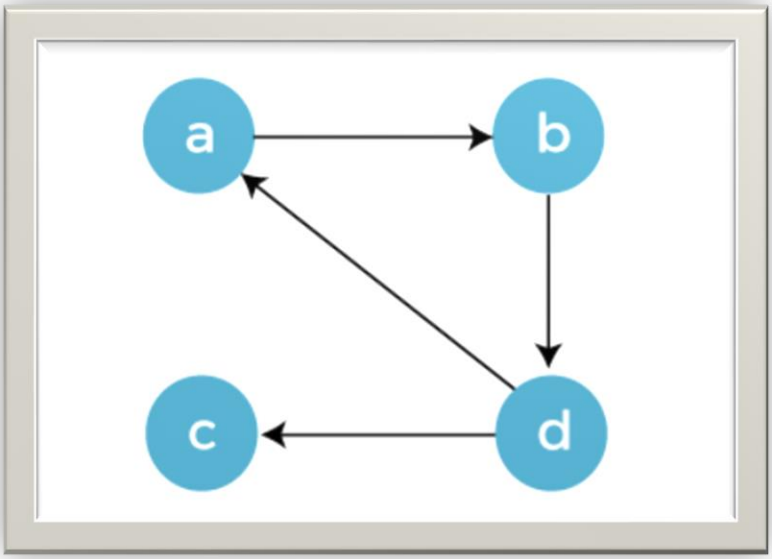
```
#include<stdio.h>
void warshall(int a[10][10], int p[10][10],int n)
{

    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");
    }

}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\Jav
.c -o hello } ; if ($?) { .\hello }
Enter number of nodes
4
Enter adjacency matrix
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
adjacency matrix is:
0  1  0  0
0  0  0  1
0  0  0  0
1  0  1  0
Path matrix is:
1  1  1  1
1  1  1  1
0  0  0  0
1  1  1  1
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\Jav
```

GRAPH:



12. Implement 0/1 Knapsack problem using dynamic programming

CODE:

```
#include<stdio.h>
int max(int a, int b)
 {
     return (a > b)? a : b;
}
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];
    for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],
K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }
     return K[n][W];
}
int main()
{
    int W,n,i;
    int val[10] , wt[10] ;
    printf("Enter number of items:\n");
    scanf("%d",&n);
    printf("Enter value for each item\n");
    for(i=0;i<n;i++)
    scanf("%d",&val[i]);
    printf("Enter weight of each item respectively\n");
    for(i=0;i<n;i++)
    scanf("%d",&wt[i]);
```

```
    printf("Enter total weight :\n");
    scanf("%d",&W);
    printf("Maximum amount : %d", knapSack(W, wt, val, n));
    return 0;
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\
.c -o hello } ; if ($?) { .\hello }
Enter number of items:
6
Enter value for each item
7 9 4 5 3 2
Enter weight of each item respectively
3 1 2 4 5 6
Enter total weight :
20
Maximum amount : 28
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\
```

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

CODE:

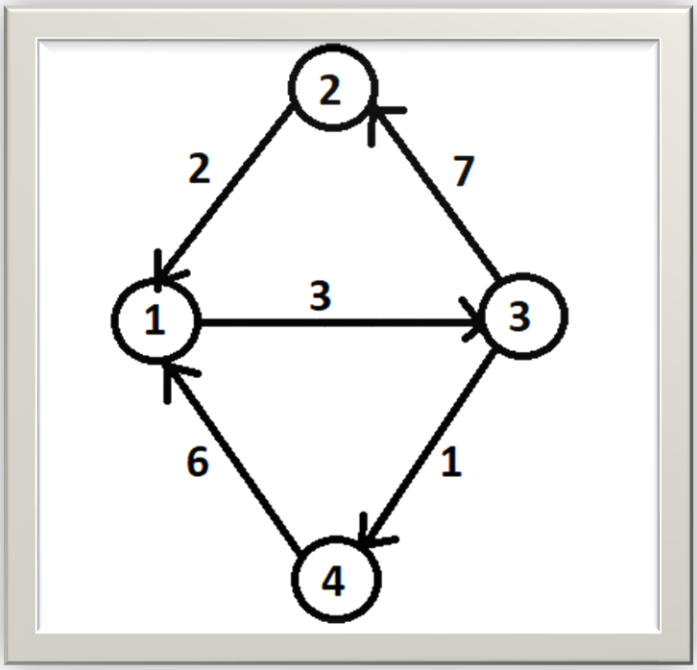
```
#include <stdio.h>
#include <stdlib.h>
int n;
int a[20][20];
int mini(int a,int b){
    if(a<b){
        return a;
    }
    else{
        return b;
    }
}
void floyd(int a[][20]){
    for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                a[i][j] = mini(a[i][j],(a[i][k]+a[k][j]));
            }
```

```
        }
    }
}
int main(){
    printf("Floyd's algorithm to find all pair shortest
paths:\n");
    printf("Enter the number of vertices: \n");
    scanf("%d",&n);
    printf("Enter the weighted matrix: \n");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&a[i][j]);
        }
    }
    floyd(a);
    printf("All pair shortest paths: \n");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

OUTPUT:

```
.c -o hello } ; if ($?) { .\hello }
Floyd's algorithm to find all pair shortest paths:
Enter the number of vertices:
4
Enter the weighted matrix:
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
All pair shortest paths:
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\
```

GRAPH:



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

CODE:

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

int a[30][30],visited[30],n,t_wt=0,c=0;
int flag = 0;

void prim(int v)
{
    int indi,indj;
    visited[v] = 1;
    int min = 10000000;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
```

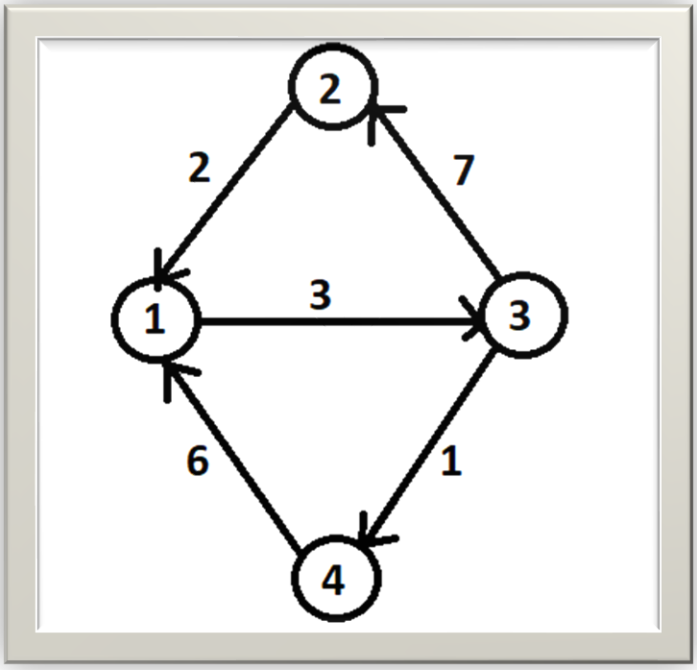
```
            if(visited[i]==1 && visited[j]==0)
            {
                if(a[i][j]<min)
                {
                    min = a[i][j];
                    indi = i;
                    indj = j;
                }
                else
                {
                    flag++;
                }
            }
            else
            {
                flag++;
            }
        }
    }
    t_wt = t_wt + a[indi][indj];
    printf("%d to %d\t%d\n",indi,indj,a[indi][indj]);
    c++;
    visited[indj] = 1;
    if(c==n-1)
        return;
    prim(indj);
}
int main()
{
    printf("Enter the no. of vertices: \n");
    scanf("%d",&n);
    printf("\nEnter the weighted matrix: \n");
    for(int i=1;i<=n;i++)
    {
         visited[i]=0;
        for(int j=1;j<=n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Minimum spanning tree is: \n");
```

```
        printf("Edges\tWeights\n");
        prim(1);
        printf("Minimum cost is: %d",t_wt);
        return 0;
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\
.c -o hello } ; if ($?) { .\hello }
Enter the no. of vertices:
4

Enter the weighted matrix:
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
Minimum spanning tree is:
Edges   Weights
1 to 3  3
3 to 4  1
3 to 2  7
Minimum cost is: 11
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\
```

GRAPH:



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

CODE:

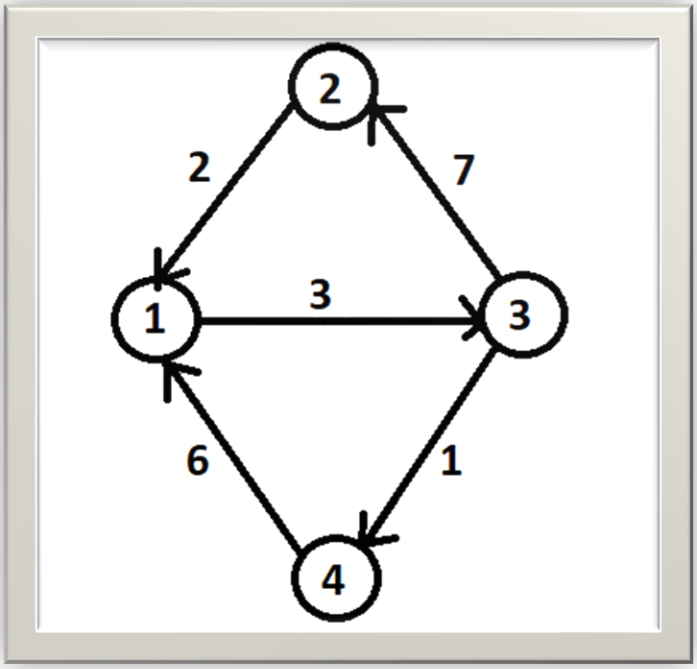
```
#include <stdio.h>
#include <stdlib.h>

int parent[30];
int find(int i)
{
    while(parent[i] != i)
        i = parent[i];
    return i;
}
int unionv(int i,int j)
{
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}
void kruskals(int a[][30],int n)
{
    int mincost = 0,c = 0,i,j;
    for(int i=0;i<n;i++)
    {
        parent[i] = i;
    }
    while(c<n-1)
    {
        int min = 10000,it=-1,jt=-1;
        for(int i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                if(find(i)!=find(j) && a[i][j]<min)
                {
```

```
                    min = a[i][j];
                    it = i;
                    jt = j;
                }
            }
        }
        unionv(it,jt);
        c++;
        printf("(%d,%d)\t %d\n",it+1,jt+1,min);
        mincost = mincost+min;
    }
    printf("\nMinimum cost = %d",mincost);
}
int main()
{
    int a[30][30],n,i,j;
    printf("Enter the no. of vertices: \n");
    scanf("%d",&n);
    printf("\nEnter the weighted matrix: \n");
    for(int i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                scanf("%d",&a[i][j]);
            }
        }
    printf("\nMinimal spanning tree is: \n");
    printf("Edges\tWeights\n");
    kruskals(a,n);
    return 0;
}
```

OUTPUT:

```
Enter the no. of vertices:
4

Enter the weighted matrix:
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0

Minimal spanning tree is:
Edges   Weights
(3,4)    1
(2,1)    2
(1,3)    3

Minimum cost = 6
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\J
```

GRAPH:



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

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <conio.h>
#include <time.h>
#include <stdbool.h>
#include <limits.h>

int V;

void dijkstra(int graph[V][V],int V)
{
    int distance[V],predefine[V],visited[V];
    int startnode,count,min_distance,nextnode,i,j;
    printf("\nEnter the start node: ");
    scanf("%d",&startnode);
    for(i=0;i<V;i++)
    {
        distance[i] = graph[startnode][i];
        predefine[i] = startnode;
        visited[i] = 0;
    }
    distance[startnode] = 0;
    visited[startnode] = 1;
    count = 1;
    while(count<V-1)
    {
        min_distance = 99;
        for(i=0;i<V;i++)
        {
            if(distance[i]<min_distance &&
visited[i]==0)
            {
                min_distance = distance[i];
                nextnode = i;
            }
        }
        visited[nextnode] = 1;
        for(i=0;i<V;i++)
```

```
        {
            if(visited[i] == 0)
            {
                if((min_distance + graph[nextnode][i]) <
distance[i])
                {
                    distance[i] = min_distance +
graph[nextnode][i];
                    predefine[i] = nextnode;
                }
            }
        }
        count = count + 1;
    }
    for(i=0;i<V;i++)
    {
        if(i!=startnode)
        {
            printf("\nDistance of node %d =
%d",i,distance[i]);
            printf("\nPath = %d",i);
            j = i;
            do
            {
                j = predefine[j];
                printf(" <- %d",j);
            }while(j!=startnode);
        }
    }
}

int main()
{
    int i,j;
    printf("Enter the number of vertices: ");
    scanf("%d",&V);
    int graph[V][V];
    printf("\nEnter the cost/weight matrix: \n");
    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
```

```
            {
                scanf("%d",&graph[i][j]);
            }
        }
        dijkstra(graph,V);
        return 0;
}
```

OUTPUT:

```
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\
.c -o hello } ; if ($?) { .\hello }
Enter the number of vertices: 4

Enter the cost/weight matrix:
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0

Enter the start node: 1

Distance of node 0 = 2
Path = 0 <- 1
Distance of node 2 = 5
Path = 2 <- 0 <- 1
Distance of node 3 = 6
Path = 3 <- 2 <- 0 <- 1
PS C:\Users\Ankith S\Dropbox\PC\Documents\Programming\
```

GRAPH: