VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Nagaraj Sunagar **(1BM20CS090)**

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-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 **Nagaraj Sunagar(1BM18CS111),** who is Bonafede 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 an **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:

Namratha M	**Dr. Jyothi S Nayak**
Assistant professor	Associated Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

	programming	
12	Implement 0/1 Knapsack problem using dynamic programming.	**45-47**
13	Implement All Pair Shortest paths problem using Floyd's algorithm.	**48-52**
14	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	**53-56**
15	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.	**57-60**
16	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	**61-64**
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.	**65-69**
18	Implement "N-Queens Problem" using Backtracking	**70-73**

<u>Course Outcome</u>

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.

1. Write a recursive program to Solve

a) Towers-of-Hanoi problem

```
#include<stdio.h>
void tower_hanoi(int n, char src, char temp, char dest) {
if(n == 1) {
printf("MOVE DISC %d FROM %c to %c \n",n,src,dest);
return ;
}
tower_hanoi(n - 1, src, dest, temp);
printf("MOVE DISC %d FROM %c to %c \n",n,src,dest);
tower_hanoi(n - 1, temp, src, dest);
}
int main() {
int x;
printf("Enter no of disc :");
scanf("%d",&x);
tower_hanoi(x, 'A', 'B', 'C');
return 0;
}
```

Output:

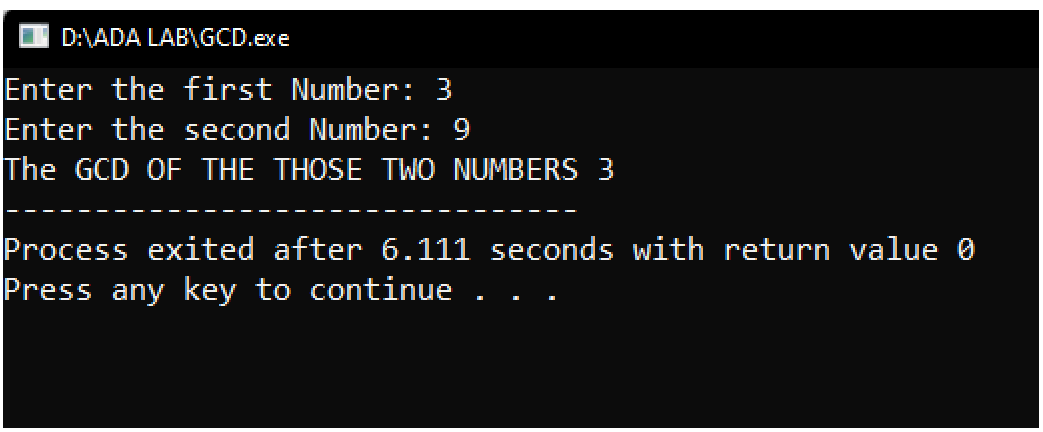
```
D:\ADA LAB\TowerOfHanoi.exe
Enter no of disc :3
MOVE DISC 1 FROM A to C
MOVE DISC 2 FROM A to B
MOVE DISC 1 FROM C to B
MOVE DISC 3 FROM A to C
MOVE DISC 1 FROM B to A
MOVE DISC 2 FROM B to C
MOVE DISC 1 FROM A to C

--------------------------------
Process exited after 45.86 seconds with return value 0
Press any key to continue . . .
```

b) To find GCD

```
#include<stdio.h>
int GCD(int,int);
int main(int argc,char **argv){
int m,n;
printf("Enter the first Number");
scanf("%d",&m);
printf("Enter the second Number");
scanf("%d",&n);
m=GCD(m,n);
printf("The GCD OF THE THOSE TWO NUMBERS %d",m);
return 0;
}
```

```
int GCD(int m,int n){
if(m==0){
return n;
}
if(n==0){
return m;
}
return GCD(n,m%n);
}
```

Output:



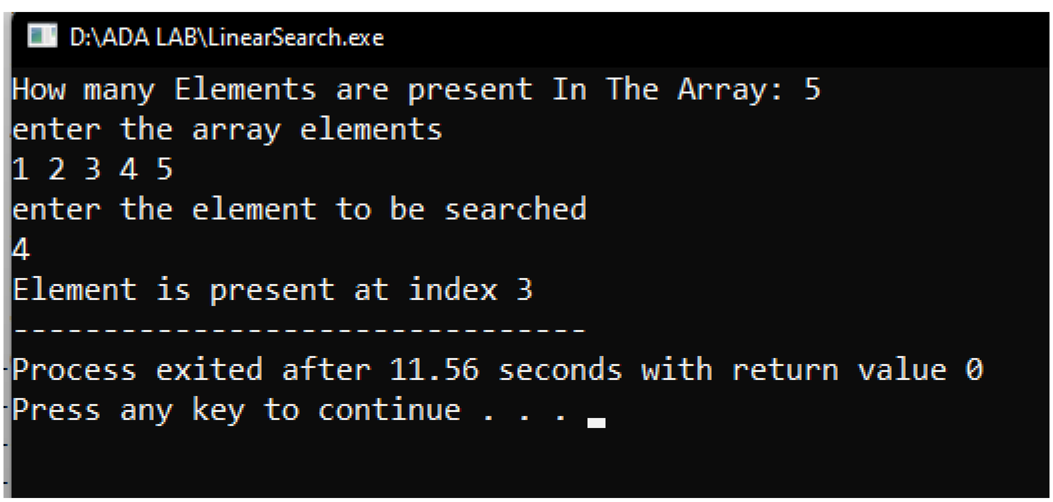
2) Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.

LINEAR SEARCH

```
#include<stdio.h>
#include<stdlib.h>
int search(int a[], int, int);
void main()
{
int ch,n, i, key, pos = 0;
printf("How many Elements are present In The Array: ");
scanf("%d",&n);
int a[n];
printf("enter the array elements\n");

for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("enter the element to be searched\n");
scanf("%d",&key);
pos = search(a, n, key);
```

```
if(pos == -1)
printf("Element is not present in array");
else
printf("Element is present at index %d", pos);
}
int search(int arr[], int n, int x)
{
int i;
for (i = 0; i < n; i++)
if (arr[i] == x)
return i;
return -1;
}
```

Output:

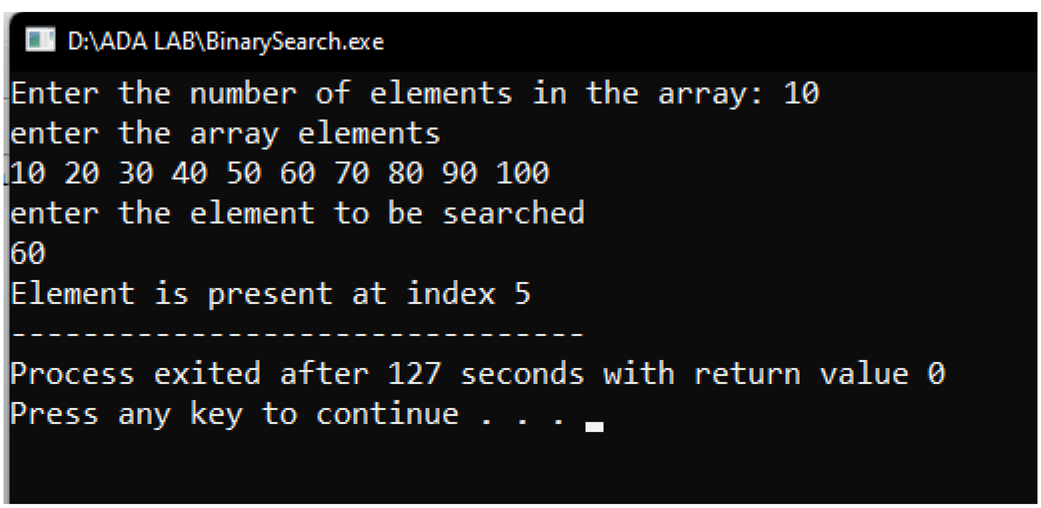


BINARY SEARCH

```
#include <stdio.h>
int binary(int a[], int low, int high, int m)
{
if (high >= low) {
int mid = low + (high - low) / 2;
if (a[mid] == m)
return mid;
if(a[mid] > m)
return binary(a, low, mid - 1, m);
return binary(a, mid + 1, high, m);
}
return -1;
}
int main()
{
int ch,n, i, key, pos = 0;
printf("Enter the number of elements in the array: ");
scanf("%d",&n);
int a[n];
printf("enter the array elements\n");
```

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

{

scanf("%d",&a[i]);

}

printf("enter the element to be searched\n");

scanf("%d",&key);

pos = binary(a, 0, n - 1,key);

if(pos == -1)

printf("Element is not present in array");

else

printf("Element is present at index %d", pos);

return 0;

}
```

Output:



```
D:\ADA LAB\BinarySearch.exe
Enter the number of elements in the array: 10
enter the array elements
10 20 30 40 50 60 70 80 90 100
enter the element to be searched
60
Element is present at index 5
--------------------------------
Process exited after 127 seconds with return value 0
Press any key to continue . . .
```

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>

#define MAXINT 2000

void delay(int n)
{
int i;
for(i=0;i<n;i++);
}
void selection(int *a,int n)
{
delay(1000);
int i,j,temp,min;
for(i=0;i<n-1;i++)
{
min=a[i];
for(j=i+1;j<n;j++)
{
```

```
if(a[j]<min)
{
min=j;
}
}
temp=a[i];a[i]=a[j];a[j]=temp;
}
}

int main()
{
clock_t  c1,c2;
int i,datasize=1;
long int n=1000;
int *a;

while(datasize<=10)
{
a=(int *)calloc(n,sizeof(int));
if(a==NULL)
{
printf("INSUFFICIENT MEMORY\n");
exit(1);
```

```
}
for(i=0;i<=n-1;i++) a[i]=rand() % MAXINT;
c1=clock();
selection(a,n);
c2=clock();
free(a);
if((c2 -c1) != 0)
{
printf("N:%d\tTIME:%f\n",n,(double)(c2-c1)/CLK_TCK);
datasize++;
}
n+=10000;
}
return 0;
}
```

Output:

```
D:\ADA LAB\SelectionSort.exe
        N:1000  TIME:0.001000
        N:11000 TIME:0.093000
        N:21000 TIME:0.337000
        N:31000 TIME:0.730000
        N:41000 TIME:1.273000
        N:51000 TIME:1.971000
        N:61000 TIME:2.823000
        N:71000 TIME:3.818000
        N:81000 TIME:4.968000
        N:91000 TIME:6.274000

--------------------------------
Process exited after 22.44 seconds with return value 0
Press any key to continue . . .
```

Graphical Representation:

4.Write program to do the following:

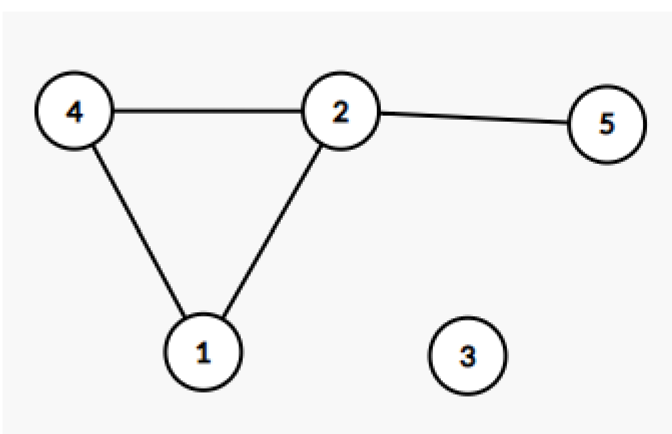
a) Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
#include<stdio.h>

void insertq(int q[],int node, int *f, int *r)
{
if((*f==-1) && (*r==-1))
{
(*f)++, (*r)++, q[*f]=node;
}
else{
(*r)++, q[*r]=node;
}
}

int deleteq(int q[],int *f,int *r)
{
int temp;
temp=q[*f];
if(*f == *r) *f=*r=-1;
else (*f)++;
return temp;
```

```
}

void bfs(int n, int adj[][10],int src, int visited[])
{
int q[20], f=-1,r=-1,v,i;
insertq(q,src,&f,&r);
while((f <=r ) && (f != -1))
{
v=deleteq(q,&f,&r);
if(visited[v]!=1)
{
visited[v]=1;
printf("%d ",v);
}
for(i=1;i<=n;i++)
if((adj[v][i]==1) && (visited[i] !=1))
insertq(q,i,&f,&r);
}
}


int main()
{
```

```
int n,i,j,adj[10][10],src,visited[10];
printf("Enter number of vertices : ");
scanf("%d",&n);
printf("Enter adjacency matrix :\n");
for(i=1;i<=n;i++)
{
visited[i]=0;
for(j=1;j<=n;j++)
scanf("%d",&adj[i][j]);
}
printf("Enter starting vertex\n");
scanf("%d",&src);
printf("The nodes reachable from src are\n");
bfs(n,adj,src,visited);
return 0;
}
```

Graph:



Output:

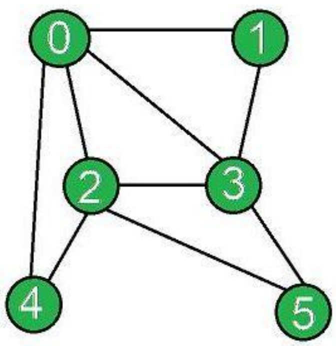
```
D:\ADA LAB\BFS.exe
Enter number of vertices : 5
Enter adjacency matrix :
0 1 0 1 0
1 0 0 1 1
0 0 0 0 0
1 1 0 0 0
0 1 0 0 0
Enter starting vertex
1
The nodes reachable from src are
1 2 4 5
--------------------------------
Process exited after 118.4 seconds with return value 0
Press any key to continue . . .
```

b) Check whether a given graph is connected or not using DFS method.

```
#include<stdio.h>


int adj[10][10];

int visited[10];

int n;

char label[] = {'A','B','C','D','E','F','G','H','I','J'};


void depthFirstSearch(int v){

printf("%c ",label[v]);

visited[v] = 1;
```

```
int i;
for(i=0;i<n;i++){
if(visited[i]==0&&adj[v][i]==1){
depthFirstSearch(i);
}
}
}
int main()
{
int i,j;
printf("Enter the number of nodes:");
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
for(i=0;i<n;i++){
for(j=0;j<n;j++){
scanf("%d",&adj[i][j]);
}
visited[i]=0;
printf("\n");
}
depthFirstSearch(0);
printf("\n");
return 0;}
```

Graph:



	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	1	1	1
3	1	1	1	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

Output:

```
D:\ADA LAB\DFS.exe
Enter the number of nodes:6
Enter the adjacency matrix:
0 1 1 1 1 0

1 0 0 1 0 0

1 0 0 1 1 1

1 1 1 0 0 1

1 0 1 0 0 0

0 0 1 1 0 0

A B D C E F

--------------------------------
Process exited after 86.47 seconds with return value 0
Press any key to continue . . .
```

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<stdlib.h>
#define MAXINT 2000
void delay(int n)
{
int i;
for(i=0;i<n;i++);
}

void insertionSort(int arr[], int n)
{
delay(1000);
long int i, key, j;
for (i = 1; i < n; i++) {
key = arr[i];
j = i - 1;
while (j >= 0 && arr[j] > key) {
arr[j + 1] = arr[j];
j = j - 1;
}
```

```
arr[j + 1] = key;
}
}

int main()
{  clock_t  c1,c2;
int i,datasize=1;
long int n=1000;
int *a;
while(datasize<=10)
{
a=(int *)calloc(n,sizeof(int));
if(a==NULL)
{
printf("INSUFFICIENT MEMORY\n");
exit(1);
}
for(i=0;i<=n-1;i++) a[i]=rand() % MAXINT;
c1=clock();
insertionSort(a,n);
c2=clock();
free(a);
if((c2 -c1) != 0)
```

```
{
printf("\tN:%d\tTIME:%f\n",n,(double)(c2-c1)/CLK_TCK);
datasize++;
}
n+=10000;
}
return 0;
}
```

Output:

```
D:\ADA LAB\InsertionSort.exe
        N:1000  TIME:0.001000
        N:11000 TIME:0.062000
        N:21000 TIME:0.226000
        N:31000 TIME:0.490000
        N:41000 TIME:0.856000
        N:51000 TIME:1.318000
        N:61000 TIME:1.898000
        N:71000 TIME:2.556000
        N:81000 TIME:3.343000
        N:91000 TIME:4.200000

--------------------------------
Process exited after 15.08 seconds with return value 0
Press any key to continue . . .
```

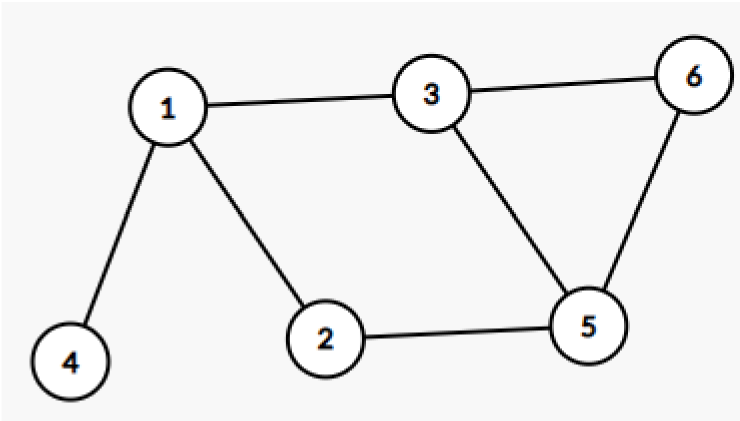
Graphical representation:

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]) {
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]);
}

int 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);
return 0;
}
```

Graph:



Output:

```
D:\ADA LAB\TopologicalOrdering.exe
Enter number 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 0
0 0 0 0 0 0
0 0 0 0 1 0
Topological order :  1  4  3  6  2  5
--------------------------------
Process exited after 29.6 seconds with return value 0
Press any key to continue . . .
```

7.Implement Johnson Trotter algorithm to generate permutations.

```
#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:

```
D:\ADA LAB\Johnson_Trotter.exe
Johnson trotter algorithm to find all permutations of given numbers
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

--------------------------------
Process exited after 2.473 seconds with return value 0
Press any key to continue . . .
```

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>

#define MAXINT 2000

void delay(int n)
{
int i;
for(i=0;i<n;i++);
}
void combine(int a[],int low,int mid,int high)
{
int c[150000],i,j,k;
i=k=low;
j=mid+1;
while(i<=mid&&j<=high)
{
if(a[i]<a[j])
{
```

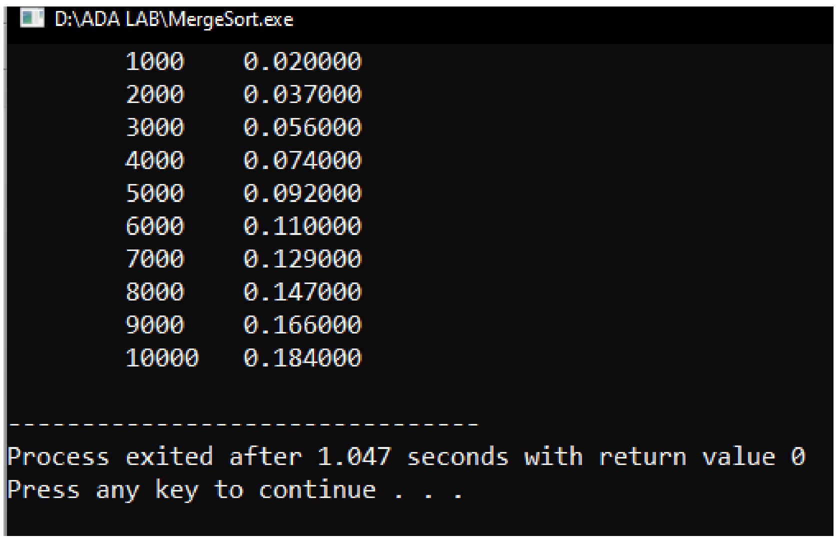
```
c[k]=a[i];
++k;
++i;
}
else
{
c[k]=a[j];
++k;
++j;
}
}
if(i>mid)
{
while(j<=high)
{
c[k]=a[j];
++k;
++j;
}
}
if(j>high)
{
while(i<=mid)
```

```
{
c[k]=a[i];
++k;
++i;
}
}
for(i=low;i<=high;i++)
{
a[i]=c[i];
}
}
void split(int a[],int low,int high)
{
delay(5000);
int mid;
if(low<high)
{
mid=(low+high)/2;
split(a,low,mid);
split(a,mid+1,high);
combine(a,low,mid,high);
}
}
```

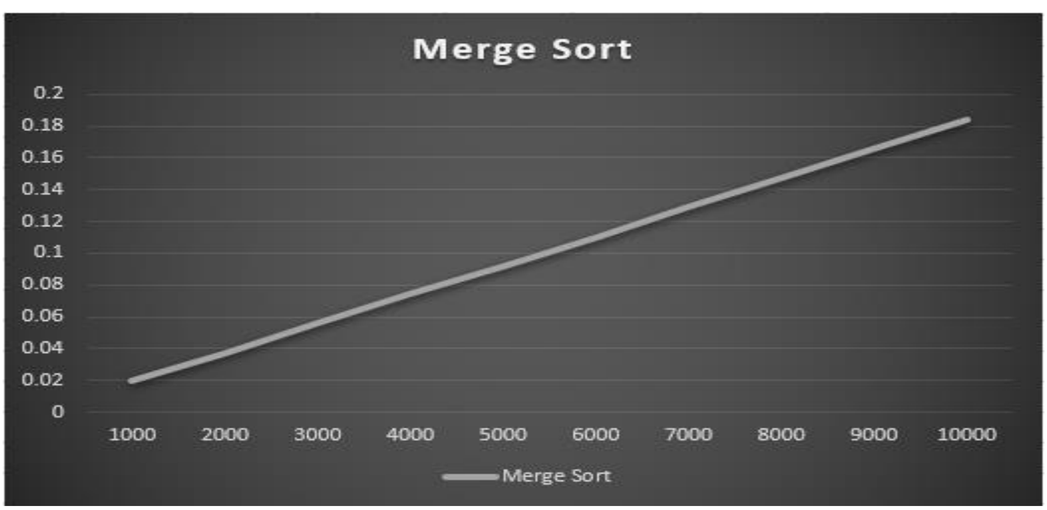
```
int main()
{
clock_t  c1,c2;
int i,datasize=1;
long int n=1000;
int *a;
while(datasize<=10)
{
a=(int *)calloc(n,sizeof(int));
if(a==NULL)
{
printf("INSUFFICIENT MEMORY\n");
exit(1);
}
for(i=0;i<=n-1;i++) a[i]=rand() % MAXINT;
c1=clock();
split(a,0,n-1);
c2=clock();
free(a);
if((c2 -c1) != 0)
{
printf("\t%d\t%f\n",n,(double)(c2-c1)/CLK_TCK);
datasize++;
```

```
}
n+=1000;
}
return 0;
}
```

Output:



Graphical representation:



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

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

#define MAXINT 2000

void delay(int n)
{
int i;
for(i=0;i<n;i++);
}
void quicksort(int number[],int first,int last){
delay(1000);
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++;
```

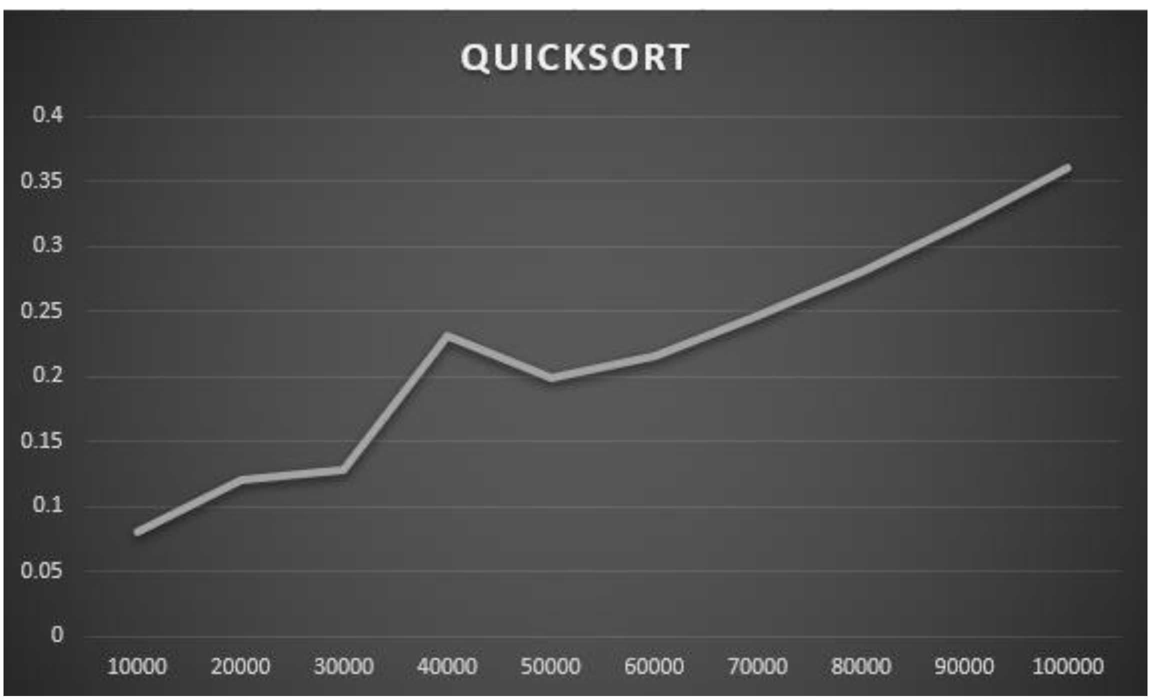
```
while(number[j]>number[pivot])
j--;
if(i<j){
temp=number[i];
number[i]=number[j];
number[j]=temp;
}
}
temp=number[pivot];
number[pivot]=number[j];
number[j]=temp;
quicksort(number,first,j-1);
quicksort(number,j+1,last);
}
}
int main()
{
clock_t  c1,c2;
int i,datasize=1;
long int n=10000;
int *a;
while(datasize<=10)
{
```

```
a=(int *)calloc(n,sizeof(int));
if(a==NULL)
{
printf("INSUFFICIENT MEMORY\n");
exit(1);
}
for(i=0;i<=n-1;i++) a[i]=rand() % MAXINT;
c1=clock();
quicksort(a,0,n-1);
c2=clock();
free(a);
if((c2 -c1) != 0)
{
printf("\tN:%d\tTIME:%f\n",n,(double)(c2-c1)/CLK_TCK);
datasize++;
}
n+=10000;
}
return 0;
}
```

Output:

```
Select D:\ADA LAB\QuickSort.exe
        N:10000 TIME:0.080000
        N:20000 TIME:0.120000
        N:30000 TIME:0.128000
        N:40000 TIME:0.231000
        N:50000 TIME:0.199000
        N:60000 TIME:0.216000
        N:70000 TIME:0.246000
        N:80000 TIME:0.281000
        N:90000 TIME:0.319000
        N:100000        TIME:0.361000

--------------------------------
Process exited after 2.446 seconds with return value 0
Press any key to continue . . .
```

Graphical representation:



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>
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) {
   int i,j;
   for (i = n / 2 - 1; i >= 0; i--)
     heapify(arr, n, i);
   for (i = n - 1; i >= 0; i--) {
    swap(&arr[0], &arr[i]);
   for(j=0;j<10000000;j++);
     heapify(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()
{
  clock_t start,end;
```

```c
  int arr[1000];
  int j,n;
  printf("Enter the number of values to be inserted : \n");
  scanf("%d",&n);
 for(j=0;j<1000;j++)
   arr[j] = rand()% 1000;
 printf("Before Heap Sort : \n");
 for(j=0;j<n;j++)
   printf("%d ",arr[j]);
 start = clock();
 heapSort(arr, n);
 end = clock();
 printf("\nSorted array is given in the following way \n");
  printArray(arr, n);
 printf("Time taken  : %lf",(double)(end-start)/CLOCKS_PER_SEC);
}
```

Output:

```
D:\sem4\ADA-LAB\REMAINING\HEAPSORT.exe
Enter the no of elements :10
Enter the elements :83 9 12 56 92 13 23 19 7 1
The sorted array is:
1 7 9 12 13 19 23 56 83 92
The time taken is 0.0000000000

--------------------------------
```

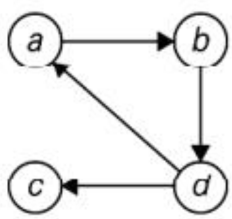
11. Implement Warshall's algorithm using dynamic programming

```
#include <stdio.h>


int a[10][10], r[10][10][10];


void warshall(int n){
    int k=0,i,j;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            r[k][i][j]=a[i][j];
    for(k=1;k<=n;k++){
        for(i=1;i<=n;i++){
            for(j=1;j<=n;j++){
                r[k][i][j]=r[k-1][i][j] || (r[k-1][i][k] && r[k-1][k][j]);
            }
        }
    }
}
```

```
void main(){
    int n,i,j;
    printf("Enter no of vertices: ");
    scanf("%d",&n);
    printf("Enter adjacency matrix: ");
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++)
            scanf("%d",&a[i][j]);
    }

    warshall(n);


    printf("Transitive Closure: \n");
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++)
            printf("%d",r[n][i][j]);
        printf("\n");
    }
}
```

Graph:

Warshall's Algorithm: Transitive Closure



(a)

$$A = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 0 & 1 & 0 \end{array}$$

(b)

$$T = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 1 & 1 \\ b & 1 & 1 & 1 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array}$$

(c)

FIGURE 8.2 (a) Digraph. (b) Its adjacency matrix. (c) Its transitive closure.

Output:

```
D:\ADA LAB\warshall.exe
Enter no of vertices: 4
Enter adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Transitive Closure:
1       1       1       1
1       1       1       1
0       0       0       0
1       1       1       1

--------------------------------
Process exited after 12.48 seconds with return value 0
Press any key to continue . . .
```

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

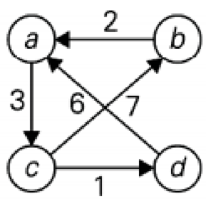
```
#include <stdio.h>
#define INF 9999
int a[10][10], d[10][10][10];
void floyd(int n){
    int k=0,i,j;
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++)
            d[k][i][j]=a[i][j];
    }
    for(k=1; k<=n; k++){
        for(i=1; i<=n; i++){
            for(j=1; j<=n; j++){
                d[k][i][j] = ((d[k-1][i][j]<(d[k-1][i][k]+d[k-1][k][j])) ? d[k-1][i][j]
:(d[k-1][i][k]+d[k-1][k][j]));
            }
        }
    }
}

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

```
printf("No of vertices: ");
scanf("%d",&n);
printf("Enter Weight matrix(-1 if there is no edge): \n");

for(i=1; i<=n; i++)
   for(j=1; j<=n; j++){
      scanf("%d",&a[i][j]);
      if(a[i][j] == -1)
         a[i][j] = INF;
   }

floyd(n);

printf("Distance matrix: \n");
for(i=1; i<=n; i++){
   for(j=1; j<=n; j++){
      if(d[n][i][j] >= INF)
         printf("-1 ");
      else
         printf("%d ",d[n][i][j]);
   }
   printf("\n");
```

```
    }
    return 0;
}
```

Graph:

Floyd's Algorithm: All pairs shortest paths



(a)

$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

(b)

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

(c)

FIGURE 8.5 (a) Digraph. (b) Its weight matrix. (c) Its distance matrix.

Output:

```
D:\ADA LAB\Floyds.exe
No of vertices: 4
Enter Weight matrix(-1 if there is no edge):
0 -1 3 -1
2 0 -1 -1
-1 7 0 1
6 -1 -1 0
Distance matrix:
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0

--------------------------------
Process exited after 38.22 seconds with return value 0
Press any key to continue . . .
```

13. 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];
int main()
{
 printf("\nenter the no. of items:\t");
 scanf("%d",&n);
 printf("\nenter the weight of the each item:\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&w[i]);
 }
 printf("\nenter the profit of each item:\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&p[i]);
 }
 printf("\nenter the knapsack's capacity:\t");
 scanf("%d",&m);
```

```
 knapsack();
 return 0;
}

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++)
{
if(x[i]!=0){
printf("%d\t",p[i]);
}
}
}

int max(int x,int y)
{
 return x>y?x:y;
}
```

Output:

```
D:\ADA LAB\knapsack.exe

enter the no. of items: 5

enter the weight of the each item:
3 4 1 5 2

enter the profit of each item:
1 5 2 6 3

enter the knapsack's capacity:  7

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

0       0       0       1       1       1       1       1

0       0       0       1       5       5       5       6

0       2       2       2       5       7       7       7

0       2       2       2       5       7       8       8

0       2       3       5       5       7       8       10


the optimal solution is 10
the solution vector is:
5       2       3
--------------------------------
Process exited after 20.38 seconds with return value 0
Press any key to continue . . .
```

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

```
#include<stdio.h>
void prims();
int c[10][10],n;
void main()
{
 int i,j,m;
 printf("\nenter the no. of vertices:\t");
 scanf("%d",&n);
 printf("\nenter the cost matrix:enter -1 for infinite distance\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&m);
   if(m==-1){
    c[i][j]=9999;
   }
   else{
    c[i][j]=m;
   }
```

```
 }
 }
 prims();
 getch();
}
void prims()
{
 int i,j,u,v,min;
 int noofVertices=0,mincost=0;
 int visited[10];
 for(i=1;i<=n;i++)
 {
  visited[i]=0;
 }
 visited[1]=1;
 while(noofVertices!=n-1)
 {
  min=9999;
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
```

```
    if(visited[i]==1)
    {
     if(c[i][j]<min)
     {
      min=c[i][j];
      u=i;
      v=j;
     }
    }
   }
  }
  if(visited[v]!=1)
  {
   printf("\n%d----->%d=%d\n",u,v,min);
   visited[v]=1;
   noofVertices+=1;
   mincost+=min;
  }
  c[u][v]=c[v][u]=9999;
 }
 printf("\nmincost=%d",mincost);
}
```

Output:

```
D:\ADA LAB\prims.exe

enter the no. of vertices:      6

enter the cost matrix:enter -1 for infinite distance
-1 3 -1 -1 6 5
3 -1 1 -1 -1 4
-1 1 -1 6 -1 4
-1 6 6 -1 8 5
6 -1 -1 8 -1 2
5 4 4 5 2 -1

1----->2=3

2----->3=1

2----->6=4

6----->5=2

6----->4=5

mincost=15
--------------------------------
Process exited after 67.08 seconds with return value 0
Press any key to continue . . .
```

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

```
#include<stdio.h>
void kruskals();
int c[10][10],n;
int main()
{
 int i,j,m;
 printf("\nenter the no. of vertices:\t");
 scanf("%d",&n);
 printf("\nenter the cost matrix: enter -1 for infinite distance\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
			scanf("%d",&m);
			if(m!=-1){
				c[i][j]=m;
   }
			else{
			c[i][j]=9999;
   }
```

```
  }
 }
 kruskals();
return 0;
}

void kruskals()
{
 int i,j,u,v,a,b,min;
 int noofVertices=0,mincost=0;
 int parent[10];
 for(i=1;i<=n;i++)
 {
  parent[i]=0;
 }
 while(noofVertices!=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;
 noofVertices=noofVertices+1;
 mincost=mincost+min;
```

```
 }
 c[a][b]=c[b][a]=9999;
}
printf("\nmincost=%d",mincost);
}
```

Output:

```
D:\ADA LAB\kruskals.exe

enter the no. of vertices:      6

enter the cost matrix: enter -1 for infinite distance
-1 3 -1 -1 6 5
3 -1 1 -1 -1 4
-1 1 -1 6 -1 4
-1 6 6 -1 8 5
6 -1 -1 8 -1 2
5 4 4 5 2 -1

2----->3=1

5----->6=2

1----->2=3

2----->6=4

4----->6=5

mincost=15
--------------------------------
Process exited after 58.98 seconds with return value 0
Press any key to continue . . .
```

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

```
#include<stdio.h>


void dijkstras();

int c[10][10],n,src;

int 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]);

  }

 }

 printf("\nenter the source node:\t");

 scanf("%d",&src);

 dijkstras();
```

```
 return 0;
}

void dijkstras()
{
  int vis[10],dist[10],u,j,count,min;
  for(j=1;j<=n;j++)
  {
   dist[j]=c[src][j];
  }
  for(j=1;j<=n;j++)
  {
   vis[j]=0;
  }
  dist[src]=0;
  vis[src]=1;
  count=1;
  while(count!=n)
  {
   min=9999;
   for(j=1;j<=n;j++)
   {
```

```
    if(dist[j]<min&&vis[j]!=1)
    {
     min=dist[j];
     u=j;
    }
   }
   vis[u]=1;
   count++;
   for(j=1;j<=n;j++)
   {
    if(min+c[u][j]<dist[j]&&vis[j]!=1)
    {
     dist[j]=min+c[u][j];
    }
   }
  }
  printf("\nthe shortest distance is:\n");
  for(j=1;j<=n;j++)
  {
   printf("\n%d----->%d=%d",src,j,dist[j]);
  }
}
```

Output:

```
D:\ADA LAB\dijkstra.exe

enter the no of vertices:       5

enter the cost matrix:
-1 3 -1 7 -1
3 -1 4 2 -1
-1 4 -1 5 6
7 2 5 -1 4
-1 -1 6 4 -1

enter the source node:  1

the shortest distance is:

1----->1=0
1----->2=3
1----->3=7
1----->4=5
1----->5=9
--------------------------------
Process exited after 37.61 seconds with return value 0
Press any key to continue . . .
```

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.

```
#include <stdio.h>

#include <stdlib.h>

static int total_nodes;


void printSubset(int A[], int size)

{

int i;

for( 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)

{

int i;

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( 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(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);

}
```

Output:

```
D:\ADA LAB\subset.exe
Enter number of elements in array
10
Enter elements of sets
1 2 4 8 3 5 7 11 13 17
Enter sum
24
1  2  3  5  13
1  2  3  7  11
1  2  4  17
1  2  8  13
1  3  4  5  11
1  3  5  7  8
1  3  7  13
1  4  8  11
1  5  7  11
2  3  4  7  8
2  3  8  11
2  4  5  13
2  4  7  11
2  5  17
3  4  17
3  8  13
4  5  7  8
4  7  13
5  8  11
7  17
11  13

--------------------------------
Process exited after 32.48 seconds with return value 1
Press any key to continue . . .
```

18. Implement “N-Queens Problem” using Backtracking

```
#define N 4
#include <stdbool.h>
#include <stdio.h>
void printSolution(int board[N][N])
{
int i,j;
for (i = 0; i < N; i++) {
for (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)
{ int i;
if (col >= N)
return true;
for (i = 0; i < N; i++)
{
if (isSafe(board, i, col))
{
board[i][col] = 1;
if (solveNQUtil(board, col + 1))
return true;
board[i][col] = 0;
}
}
return false;
}
```

```
bool solveNQ()
{
int board[N][N] = { { 0, 0, 0, 0 },
{ 0, 0, 0, 0 },
{ 0, 0, 0, 0 },
{ 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) {
printf("Solution does not exist");
return false;
}
printSolution(board);
return true;
}
int main()
{
solveNQ();
return 0;
}
```

Output:

```
D:\ADA LAB\NQueens.exe
 0  0  1  0
 1  0  0  0
 0  0  0  1
 0  1  0  0

--------------------------------
Process exited after 0.02612 seconds with return value 0
Press any key to continue . . .
```