

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

AMULYA S A(1BM21CS020)

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-2023 to July-2023

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **AMULYA S A(1BM21CS020)**, 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 academic semester May-2023 to July-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge: Sunayana S
Designation: Assitant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Lab Program No.	Program Details	Page No.
1	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.	1-7
2	Write program to obtain the Topological ordering of vertices in a given digraph.	8-11
3	Implement Johnson Trotter algorithm to generate permutations.	11-15
4	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.	16-18
5	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	19-21
6	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	22-25
7	Implement 0/1 Knapsack problem using dynamic programming.	26-28
8	Implement All Pair Shortest paths problem using Floyd's algorithm.	29-31
9	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.	32-37
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	38-40

11	Implement “N-Queens Problem” using Backtracking.	41-43
12	implementation of heap sort	44-46

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

Program 1:

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>

inta[30][30],vis[10],n;

void bfs(int);

void main()

{

int i,j,m,p,q,start;

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

printf("enter no. of edges\n");

scanf("%d",&m);

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

{

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

a[i][j]=0;

}

}

for(i=1; i<=m; i++){

printf("enter an edge\n");

scanf("%d%d",&p,&q);

a[p][q]=1;

}

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

```

{
    vis[i]=0;
    printf("enter the starting vertex\n");
    scanf("%d",&start);
    printf("nodes reachable from starting vertex are\n"); bfs(start);
    getch();
}

Void bfs(int v)
{
    int i, q[30], f=0, r=0, u; vis[v]=1;
    q[r]=v;
    while(f<=r){
        u=q[f];
        printf("%d",u);
        for(i=1; i<=n; i++){
            if(a[u][i]==1 && vis[i]==0){
                r=r+1;
                q[r]=i;
                vis[i]=1;
            }
        }
        f=f+1;
    }
}

```

Sample Output:

```
enter no. of vertices
5
enter no. of edges
6
enter an edge
1 5
enter an edge
1 2
enter an edge
2 5
enter an edge
2 4
enter an edge
3 1
enter an edge
3 4
enter the starting vertex
1
nodes reachable from starting vertex are
1254
```

b. #include<stdio.h>

int a[10][10], n, vis[10];

int dfs(int v);

int main(){

int i, j;

printf("Enter number of vertices:\n");

scanf("%d", &n);

printf("Enter adjacency matrix:\n");

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

{

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

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

}

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

vis[i] = 0;

printf("DFS Traversal\n");

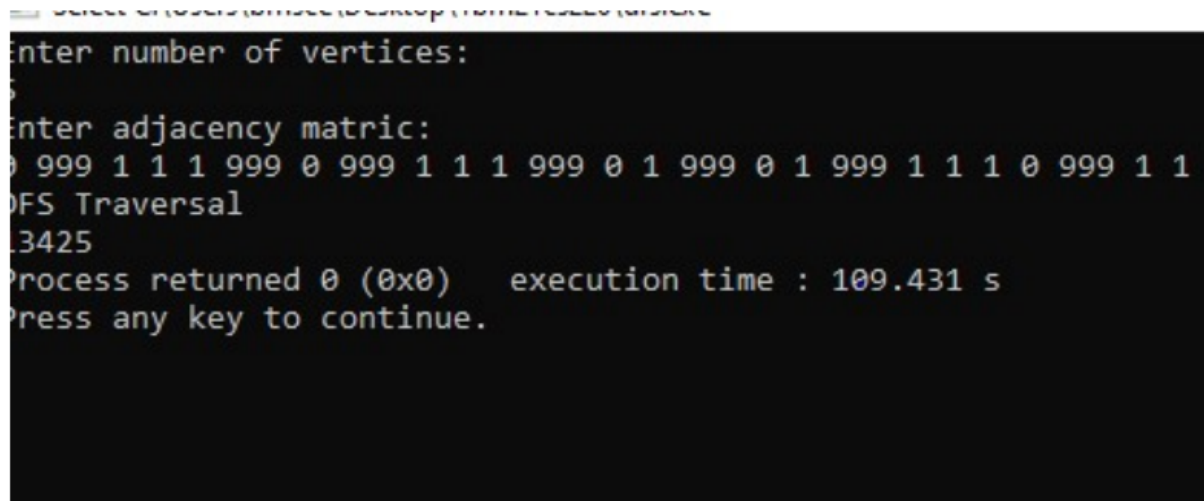
for(i = 1; i<=n; i++)

```

{
if(vis[i] == 0)
dfs(i);
}
return 0;
}
int dfs(int v){
int i;
vis[v] = 1;
printf("%d", v);
for(i=1; i<=n; i++)
{
if(a[v][i] == 1 && vis[i] == 0)
dfs(i);
}
}
}

```

Sample Output:



```

Enter number of vertices:
5
Enter adjacency matrix:
0 999 1 1 1 999 0 999 1 1 1 999 0 1 999 0 1 999 1 1 1 0 999 1 1
DFS Traversal
1 3 4 2 5
Process returned 0 (0x0)   execution time : 109.431 s
Press any key to continue.

```


Program 2:

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

```
#include<stdio.h>
#include<conio.h>
void dfs(int);
int a[10][10],vis[10],exp[10],n,i,j=0;
void main()
{
    int i,j,m,u,v;
    printf("Enter no of vertices:\n");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=1;j++)
        {
            a[i][j]=0;
        }
    }
    printf("enter the no of edges:\n");
    scanf("%d",&m);
    for(i=1;i<=m;i++)
    {
        printf("enter an edge:\n");
        scanf("%d%d",&u,&v);
        a[u][v]=1;
    }
}
```

```

for(i=1;i<=n;i++)
{
vis[i]=0;
}
for(i=1;i<=n;i++)
{
if(vis[i]==0)
{
dfs(i);
}
}
printf("\ntopological order:\n");
for(i=n-1;i>=0;i--)
{
printf("%d\t",exp[i]);
}
getch();
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);
}
}
}

```

```
}  
exp[j++]=v;  
}
```

Sample Output:

```
Enter no of vertices:  
5  
enter the no 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
```

Program 3:

Implement Johnson Trotter algorithm to generate permutations.

```
#include <stdio.h>  
  
#define RIGHT_TO_LEFT 0  
#define LEFT_TO_RIGHT 1  
  
void swap(int *a, int *b) {  
    int temp = *a;
```

```

*a = *b;

*b = temp;

}

int searchArr(int a[], int n, int mobile) {
for (int i = 0; i < n; i++) {
if (a[i] == mobile) {
return i + 1;
}
}
return -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) {
mobile = a[i];
mobile_prev = mobile;
}
}
}
}
}

```

```

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

void printOnePerm(int a[], int dir[], int n) {
int mobile = getMobile(a, dir, n);
int pos = searchArr(a, n, mobile);
if (dir[a[pos] - 1] == RIGHT_TO_LEFT) {
swap(&a[pos], &a[pos - 1]);
} else if (dir[a[pos] - 1] == LEFT_TO_RIGHT) {
swap(&a[pos], &a[pos - 1]);
}
for (int i = 0; i < n; i++) {
if (a[i] > mobile) {
if (dir[a[i] - 1] == LEFT_TO_RIGHT) {
dir[a[i] - 1] = RIGHT_TO_LEFT;
} else if (dir[a[i] - 1] == RIGHT_TO_LEFT) {
dir[a[i] - 1] = LEFT_TO_RIGHT;
}
}
}
for (int i = 0; i < n; i++) {
printf("%d ", a[i]);
}
printf("\n");
}

```

```

}

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

void printPermutation(int n) {
    int a[n];
    int dir[n];
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        printf("%d ", a[i]);
    }
    printf("\n");
    for (int i = 0; i < n; i++) {
        dir[i] = RIGHT_TO_LEFT;
    }
    for (int i = 1; i < factorial(n); i++) {
        printOnePerm(a, dir, n);
    }
}

int main() {
    int n;
    printf("Enter the value of n: ");
    scanf("%d", &n);

```

```
printPermutation(n);  
return 0;  
}
```

Sample Output:

```
Enter the value of n: 4  
1 2 3 4  
1 2 4 3  
1 4 2 3  
4 1 2 3  
4 1 3 2  
1 4 3 2  
1 3 4 2  
1 3 2 4  
3 1 2 4  
3 1 4 2  
3 4 1 2  
4 3 1 2  
4 3 2 1  
3 4 2 1  
3 2 4 1  
3 2 1 4  
2 3 1 4  
2 3 4 1  
2 4 3 1  
4 2 3 1  
4 2 1 3  
2 4 1 3  
2 1 4 3  
2 1 3 4
```

Program 4:

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<conio.h>

#include<time.h>

#include<stdlib.h>

int a[10000];

void merge(int a[],int low,int mid,int high)
{
    int i,j,k,c[10000];
    i=low;
    j=mid+1;
    k=low;
    while(i<=mid && j<=high)
    {
        if(a[i]<a[j])
            c[k++]=a[i++];
        else{
            c[k++]=a[j++];
        }
    }
    while(i<=mid)
        c[k++]=a[i++];
    while(j<=high)
        c[k++]=a[j++];
    for(i=low;i<=high;i++)
```



```

{
a[i]=c[i];
}
}

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

void main()
{
int n;
clock_t start,end;
unsigned long int i;
printf("Enter the no.of elements to be sorted:\n");
scanf("%d",&n);
printf("Enter the elements to be sorted:\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
}

```

```

start=clock();
mergesort(a,0,n-1);
end=clock();
printf("Sorted array\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
printf("\nTime taken for n=%d is %f secs",n,((double)(end-start))/(CLOCKS_PER_SEC));
getch();
}

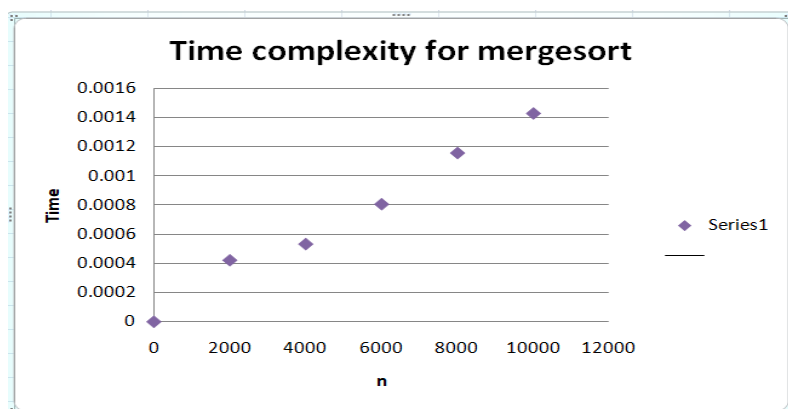
```

Sample Output:

```

Enter the no.of elements to be sorted:
7
Enter the elements to be sorted:
34 56 12 99 6 21 87
Sorted array
6      12      21      34      56      87      99
Time taken for n=7 is 0.000000 secs

```



Program 5:

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

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
void quicksort(int a[],int low,int high)
{
    int mid;
    if(low<high)
    {
        mid=partition(a,low,high);
        quicksort(a,low,mid-1);
        quicksort(a,mid+1,high);
    }
}
int partition(int a[],int low,int high)
{
    int i,j,temp,pivot;
    pivot=a[low];
    i=low+1;
    j=high;
    while(i<=j)
    {
        while(a[i]<=pivot)
        {
```

```

i++;
}
while(a[j]>pivot)
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
temp=a[low];
a[low]=a[j];
a[j]=temp;
return j;
}
void main()
{
int a[10000],n;
clock_t start,end;
unsigned long int i;
printf("Enter the no.of elements to be sorted:\n");
scanf("%d",&n);
srand(time(NULL));
for(i=0;i<n;i++)
{
a[i]=rand() %1000;

```

```

}
start=clock();
quicksort(a,0,n-1);
end=clock();
printf("Sorted array\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
printf("\nTime taken for n=%d is %f secs",n,((double)(end-start))/(CLOCKS_PER_SEC));
getch();
}

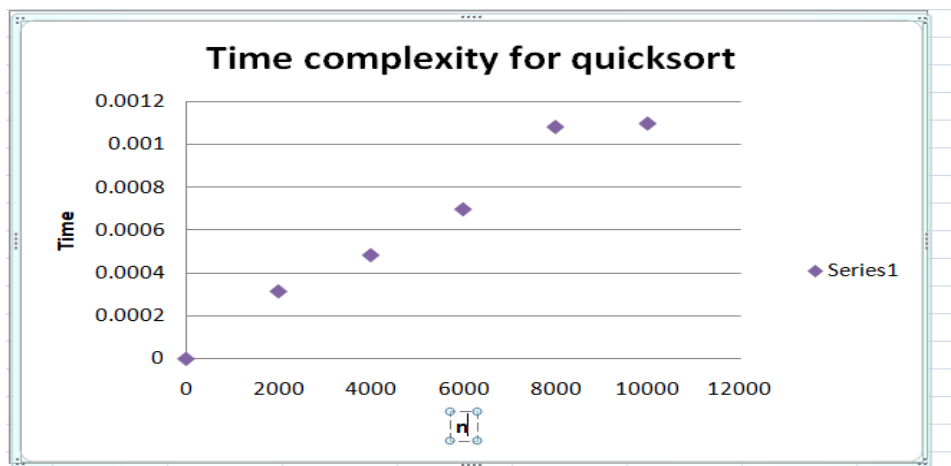
```

Sample Output:

```

Enter the no.of elements to be sorted:
10
Sorted array
112    171    389    442    485    675    690    746    852    882
Time taken for n=10 is 0.000000 secs

```



Program 6:

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

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
void heap_adj(int a[],int n)
{
    int i,j,item;
    j=0;
    item=a[j];
    i=2*j+1;
    while(i<n)
    {
        if((i+1)<=n-1)
        {
            if(a[i]<a[i+1])
                i++;
        }
        if(item<a[i])
        {
            a[j]=a[i];
            j=i;
            i=2*j+1;
        }
    }
```

```

        else
            break;
    }
    a[j]=item;
}

void heap_const(int a[],int n)
{
    int i,j,k,item;
    for(k=0;k<n;k++)
    {
        item=a[k];
        i=k;
        j=(i-1)/2;
        while(i>0 && item>a[j])
        {
            a[i]=a[j];
            i=j;
            j=(i-1)/2;
        }
        a[i]=item;
    }
}

void heapsort(int a[],int n)
{
    int i,temp;
    heap_const(a,n);
    for(i=n-1;i>0;i--)

```

```

    {
        temp=a[i];
        a[i]=a[0];
        a[0]=temp;
        heap_adj(a,i);
    }
}

void main()
{
    int a[10000],n;
    clock_t start,end;
    unsigned long int i;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    srand(time(NULL));
    for(i=0;i<n;i++)
    {
        a[i]=rand() %1000;
    }
    start=clock();
    heapsort(a,n-1);
    end=clock();
    printf("Sorted array\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",a[i]);
    }
}

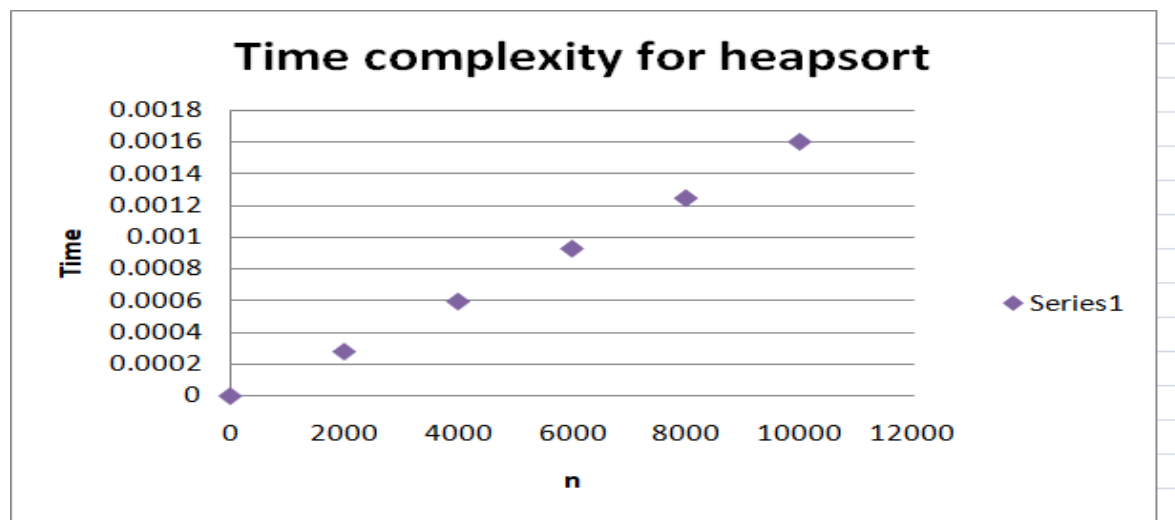
```



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

Sample Output:

```
Enter the number of elements:10
Sorted array
179    243    518    595    659    682    813    836    938    664
Time taken for n=10 is 0.000000 secs
```



Program 7:

Implement 0/1 Knapsack problem using dynamic programming.

```
#include<stdio.h>

#include<conio.h>

int n,m,w[10],p[10],v[10][10],x[10];

void knapsack()
{
    int i,j;
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=m;j++)
        {
            if(i==0 || j==0)
                v[i][j]=0;
            if(w[i]>j)
                v[i][j]=v[i-1][j];
            else
                v[i][j]=(v[i-1][j]>v[i-1][j-w[i]]+p[i])?v[i-1][j]:v[i-1][j-w[i]]+p[i];
        }
    }
    object_selected();
}

void object_selected()
{
    int i,j;
```

```

printf("Optimal solution:%d\n",v[n][m]);
for(i=1;i<=n;i++)
    x[i]=0;
    i=n;
    j=m;
while(i!=0 && j!=0)
{
    if(v[i][j]!=v[i-1][j])
    {
        x[i]=1;
        j=j-w[i];
    }
    i--;
}
for(i=1;i<=n;i++)
{
    if(x[i]==1)
    {
        printf("Object %d is selected\n",i);
    }
}
}

void main()
{
    int i;
    printf("Emter number of objects:");
    scanf("%d",&n);

```

```

printf("\nMaximum capacity:");
scanf("%d",&m);
printf("\nEnter weights of the objects:");
for(i=1;i<=n;i++)
{
    scanf("%d",&w[i]);
}
printf("\nEnter profits of the objects:");
for(i=1;i<=n;i++)
{
    scanf("%d",&p[i]);
}
knapsack();
}

```

Sample Output:

```

Enter number of objects:4
Maximum capacity:10
Enter weights of the objects:5 3 2 2
Enter profits of the objects:10 15 13 18
Optimal solution:46
Object 2 is selected
Object 3 is selected
Object 4 is selected
-----
Process exited after 54.68 seconds with return value 4
Press any key to continue . . .

```

Program 8:

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

```
#include<stdio.h>

int n;

int c[10][10],d[10][10];

void floyds();

int main()
{
    int i,j;

    printf("\nEnter the number of vertices:");

    scanf("%d",&n);

    printf("\nEnter the cost matrix:");

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

    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            d[i][j]=0;
        }
    }
}
```

```

floyds();
return 0;
}

```

```

void floyds()
{
    int i,j,k;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            d[i][j]=c[i][j];
        }
    }
    for(k=1;k<=n;k++)
    {
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                d[i][j]=d[i][j]>(d[i][k]+d[k][j])?(d[i][k]+d[k][j]):d[i][j];
            }
        }
    }
    printf("\nThe distance matrix is:\n");
    for(i=1;i<=n;i++)
    {

```

```

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

```

Sample Output:

```

Enter the number of vertices:4

Enter the cost matrix:0 1 10 7
999 0 999 3
999 999 0 999
999 999 2 0

The distance matrix is:
0      1      6      4
999    0      5      3
999    999    0      999
999    999    2      0

-----
Process exited after 46.87 seconds with return value 0
Press any key to continue . . .

```

Program 9:

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

Prims:

```
#include<stdio.h>
#include<conio.h>
int vis[10],vt[10],et[10][2],e=0,n;
float cost[10][10],sum=0;
void prims()
{
    int x=1,min,i,j,m,k,u,v;
    vt[x]=1;
    vis[x]=1;
    for(i=1;i<n;i++)
    {
        j=x;
        min=999;
        while(j>0)
        {
            k=vt[j];
            for(m=2;m<=n;m++)
            {
                if(cost[k][m]<min && vis[m]==0)
                {
                    min=cost[k][m];
                    u=k;

```



```

v=m;
}
}
j--;
}
vt[++x]=v;
et[i][1]=u;
et[i][2]=v;
e++;
vis[v]=1;
sum=sum+cost[u][v];
}
}
void main()
{
int i,j;
printf("Enter the number of vertices:");
scanf("%d",&n);
printf("\nEnter the cost matrix:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%f",&cost[i][j]);
}
}
prims();

```

```

for(i=1;i<=e;i++)
{
printf("%d-->%d\n",et[i][1],et[i][2]);
}
printf("Total cost=%f",sum);

```

Sample Output

```

Enter the number of vertices:5

Enter the cost matrix:
0 1 5 2 999
1 0 999 999 999
3 999 0 3 999
2 999 3 0 1.5
999 999 999 1.5 0
1-->2
1-->4
4-->5
4-->3
Total cost=7.500000

```

Kruskal:

```

#include<stdio.h>

int cost[10][10],t[10][10],parent[10],n;

void kruskal()
{
int i,j,u,v;

int count=0;

int k=1;

int sum=0;

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

```

```

{
parent[i]=i;
}
while(count!=n-1)
{
int min=999;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(cost[i][j]<min&&cost[i][j]!=0)
{
min=cost[i][j];
u=i;
v=j;
}
}
}
i=find(u);
j=find(v);
if(i!=j)
{
t[k][0]=u;
t[k][1]=v;
k++;
count++;
sum=sum+cost[u][v];

```

```

union_ij(i,j);
}
cost[u][v]=cost[v][u]=999;
}
printf("Spanning Tree:\n");
for(i=1;i<=count;i++)
printf("%d->%d\t",t[i][0],t[i][1]);
printf("\nTotal Cost=%d",sum);
getch();
}
void union_ij(int i,int j)
{
if(i<j)
{
parent[j]=i;
}
else
{
parent[i]=j;
}
}
int find(int v)
{
while(parent[v]!=v)
{
v=parent[v];
}
}

```

```

    return v;
}

int main()
{
    int i,j;
    printf("\nEnter the number of vertices:");
    scanf("%d",&n);
    printf("\nEnter the cost matrix:");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
        }
    }
    kruskal();
    return 0;
}

```

Sample Output:

```

Enter the number of vertices:6
Enter the cost matrix:0 3 999 999 6 2 3 0 1 999 999 4 999 1 0 6 999 4 999 999 6 0 8 5 6 999 999
Spanning Tree:
2->3    1->6    5->6    1->2    4->6
Total Cost=13

```

Program 10:

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

```
#include<stdio.h>

#include<conio.h>

int vis[10],cost[10][10],dest[10];

int n,src;

void dijktras()
{
    int i,count,min,u;
    for(i=1;i<=n;i++)
    {
        dest[i]=cost[src][i];
    }
    vis[src]=1;
    count=1;
    while(count<=n)
    {
        min=999;
        for(i=1;i<=n;i++)
        {
            if(dest[i]<min && vis[i]==0)
            {
                min=dest[i];
                u=i;
            }
        }
    }
}
```

```

    }
}
vis[u]=1;
for(i=1;i<=n;i++)
{
    if(dest[u]+cost[u][i]<dest[i] && vis[i]==0)
    {
        dest[i]=dest[u]+cost[u][i];
    }
    count++;
    printf("%d -> %d=%d\n",src,i,dest[i]);
}
}
}
void main()
{
    int i,j;
    printf("enter the no of vetices:");
    scanf("%d",&n);
    printf("\nEnter the cost matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
        }
    }
}

```

```
printf("\nenter the source vertex:");  
scanf("%d",&src);  
dijktras();  
getch();  
}
```

Sample Output:

```
enter the no of vetices:4  
  
Enter the cost matrix:  
0 3 999 7  
3 0 4 2  
999 4 0 5  
7 2 5 0  
  
enter the source vertex:1  
1 -> 1=0  
1 -> 2=3  
1 -> 3=7  
1 -> 4=5
```


Program 11:

Implement “N-Queens Problem” using Backtracking.

```
#include<stdio.h>

#include<stdlib.h>

int board[20],count;

int main()
{
    int n,i,j;
    void queen(int row,int n);
    printf(" - N Queens Problem Using Backtracking -");
    printf("\n\nEnter number of Queens:");
    scanf("%d",&n);
    queen(1,n);
    return 0;
}

void print(int n)
{
    int i,j;
    printf("\n\nSolution %d:\n\n",++count);
    for(i=1;i<=n;++i)
        printf("\t%d",i);
    for(i=1;i<=n;++i)
    {
        printf("\n\n%d",i);
```

```

        for(j=1;j<=n;++j)
        {
            if(board[i]==j)
                printf("\tQ");
            else
                printf("\t-");
        }
    }
}

int place(int row,int column)
{
    int i;
    for(i=1;i<=row-1;++i)
    {
        if(board[i]==column)
            return 0;
        else if(abs(board[i]-column)==abs(i-row))
            return 0;
    }
    return 1;
}

void queen(int row,int n)
{
    int column;
    for(column=1;column<=n;++column)
    {

```

```

    if(place(row,column))
    {
        board[row]=column;
        if(row==n)
            print(n);
        else
            queen(row+1,n);
    }
}
}

```

Sample Output:

```

Enter number of Queens:4

Solution 1:

    1      2      3      4
1      -      Q      -      -
2      -      -      -      Q
3      Q      -      -      -
4      -      -      Q      -

Solution 2:

    1      2      3      4
1      -      -      Q      -
2      Q      -      -      -
3      -      -      -      Q
4      -      Q      -      -
Process returned 0 (0x0)   execution time : 10.551 s
Press any key to continue.

```

PROGRAM-12

IMPLEMENTATION OF HEAP SORT

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

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

```
void swap (int *x, int *y){
```

```
    int temp = *x;
```

```
    *x = *y;
```

```
    *y = temp;
```

```
}
```

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

```
    int largest = i, left = 2*i+1, 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);
    }
}

```

```

int main (){
    int n;

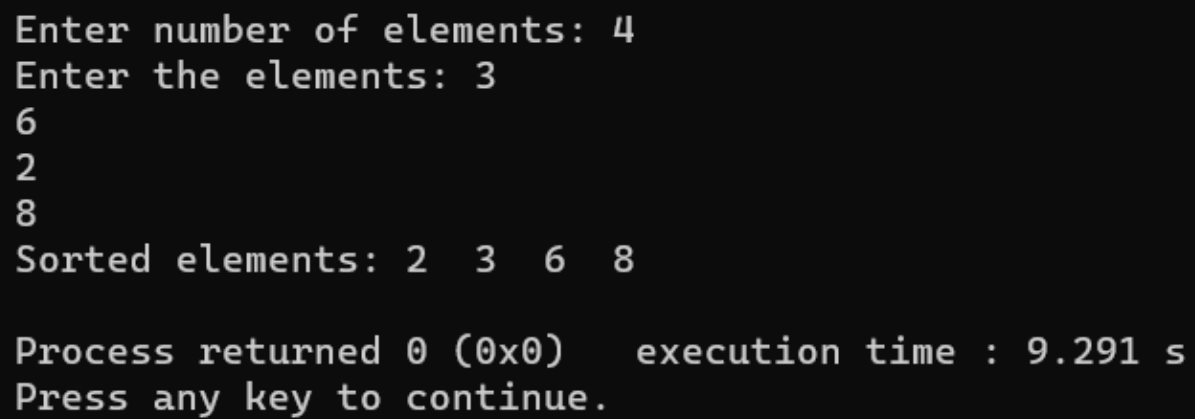
    printf ("Enter number of elements: ");
    scanf ("%d", &n);
    int arr[n];

    printf ("Enter the elements: ");
    for (int i = 0; i < n; i++){
        scanf ("%d", &arr[i]);
    }
    heapsort (arr, n);
    printf ("Sorted elements: ");
}

```

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

SAMPLE OUTPUT:



```
Enter number of elements: 4  
Enter the elements: 3  
6  
2  
8  
Sorted elements: 2 3 6 8  
  
Process returned 0 (0x0)   execution time : 9.291 s  
Press any key to continue.
```