

# **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



## **LAB REPORT on**

# **Analysis and Design of Algorithms**

*Submitted by*

**IMADH AJAZ BANDAY (1BM20CS059)**

*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 **IMADH AJAZ BANDAY (1BM20CS059)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:

Dr. Nagarathna N  
Professor  
Department of CSE  
BMSCE, Bengaluru

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

## Index Sheet

Sl. No.	Experiment Title	Page No.
<b>1</b>	Write a recursive program to Solve <b>a)</b> Towers-of-Hanoi problem <b>b)</b> To find GCD	
<b>2</b>	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.	
<b>3</b>	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.	
<b>4</b>	Write program to do the following: <b>a)</b> Print all the nodes reachable from a given starting node in a digraph using BFS method. <b>b)</b> Check whether a given graph is connected or not using DFS method.	
<b>5</b>	Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.	
<b>6</b>	Write program to obtain the Topological ordering of vertices in a given digraph.	
<b>7</b>	Implement Johnson Trotter algorithm to generate permutations.	
<b>8</b>	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.	
<b>9</b>	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	
<b>10</b>	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	
<b>11</b>	Implement Warshall's algorithm using dynamic programming	
<b>12</b>	Implement 0/1 Knapsack problem using dynamic programming.	
<b>13</b>	Implement All Pair Shortest paths problem using Floyd's algorithm.	
<b>14</b>	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	
<b>15</b>	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.	
<b>16</b>	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	
<b>17</b>	Implement "Sum of Subsets" using Backtracking. "Sum of Subsets" problem: Find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ 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.	
<b>18</b>	Implement "N-Queens Problem" using Backtracking.	

## Course Outcome

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

**Program Title**

**Complete executed program**

**Result Screen shot**

**Graph (only for Searching and Sorting programs)**

**Program-1:**

*Write a recursive program to Solve*

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

**a. Tower Of Hanoi**

```
#include <stdio.h>

#include <conio.h>

#include <math.h>

void hanoi(int x, char from, char to, char aux)
{
    if(x==1)
        printf("Move disk from %c to %c\n",from,to);
    else
    {
        hanoi(x-1,from,aux,to);
        printf("Move disk from %c to %c\n",from,to);
        hanoi(x-1,aux,to,from);
    }
}

void main()
{
    int disk;
    int moves;


    printf("Enter the number of disks you want to play with:");
    scanf("%d",&disk);
```

```

moves=pow(2,disk)-1;
printf("The number of moves required is:%d\n",moves);
hanoi(disk,'A','C','B');
}

```

**OUTPUT:**



```

input
Enter the the number of disks you want to play with:
3
The number of moves required are: 7
Move disk from A to B
Move disk from A to B
Move disk from C to A
Move disk from A to B
Move disk from C to A
Move disk from C to A
Move disk from B to C

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

```

## **b. GCD**


```

#include <stdio.h>
int gcd(int n1,int n2);
int main()
{
    int n1,n2;
    printf("Enter two positive integers:");
    scanf("%d %d",&n1, &n2);
    printf("GCD of %d and %d is %d",n1,n2,gcd(n1,n2));
    return 0;
}

```

```
}  
int gcd(int n1, int n2)  
{  
    if(n2!=0)  
        return gcd(n2,n1%n2);  
    else  
    {  
        return n1;  
    }  
  
}
```

**OUTPUT:**

A screenshot of a console window titled 'input'. The window has a black background with white text. The text shows the program's execution: it prompts for two numbers, receives '36 48', calculates the GCD as 12, and then displays a green message indicating the program finished successfully with exit code 0. The prompt 'Press ENTER to exit console.' is also visible.

```
input  
Enter the two numbers:36 48  
GCD of given numbers is: 12  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```



**Program-2:**

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

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

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

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

```
int bin_srch(int [], int ,int, int);
```

```
int lin_srch(int [], int ,int, int);
```

```
int n,a[10000];
```

```
int main()
```

```
{
```

```
    int ch,key,search_status,temp;
```

```
    clock_t end,start;
```

```
    unsigned long int i,j;
```

```
    while(1)
```

```
    {
```

```
        printf("1.Binary Search  2.Linear Search  3. Exit\n ");
```

```
        printf("Enter your choice: \t");
```

```
        scanf("%d",&ch);
```

```
        switch(ch)
```

```
        {
```

```
            case 1:
```

```

n = 1000;
while(n<=5000)
{
    for(i=0; i<n;i++)
    {
        //a[i] = random(1000);
        a[i] = i;
    }
    key = a[n-1];
    start = clock();
    search_status = bin_srch(a,0,n-1,key);
    if(search_status == -1)
        printf("key not found \n");
    else
        printf("Key found at position %d",search_status);

    for(j=0;j<5000000;j++){temp = 38/600;}
    end= clock();
    printf("\n time for n = %d is %f
Secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
    n = n+1000;
}
break;

case 2:
    n = 1000;

```

```

while(n<=5000)
{
    for(i=0; i<n;i++)
    {
        //a[i] = random(1000);
        a[i] = i;
    }
    key = a[n-1];
    start = clock();
    search_status = lin_srch(a,0,n-1,key);
    if(search_status == -1)
        printf("key not found \n");
    else
        printf("Key found at position %d",search_status);

    for(j=0;j<5000000;j++){temp = 38/600;}
    end= clock();
    printf("\n time 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)  
{  
    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 low,int high,int key)
{
    if(low>high)
    {
        return -1;
    }
    if(key == a[low])
    {
        return low;
    }
    else
    {
        return lin_srch(a,low+1,high,key);
    }
}
```

**OUTPUT:**

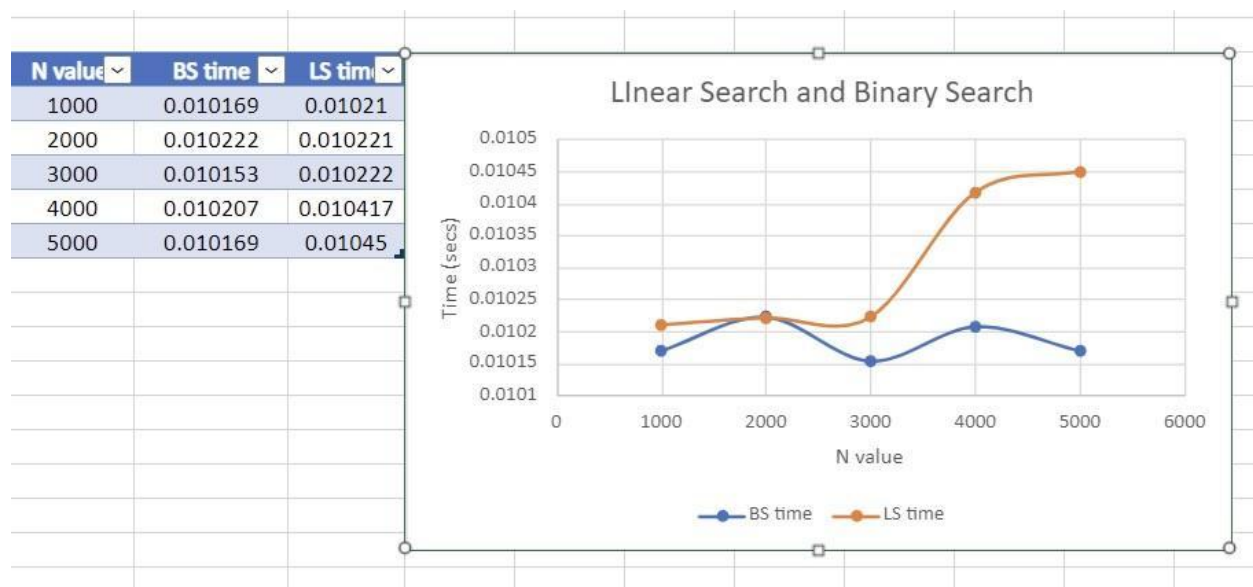
```

input
1.Binary Search  2.Linear Search  3. Exit
Enter your choice: 1
Key found at position 999
time for n = 1000 is 0.010169 SecsKey found at position 1999
time for n = 2000 is 0.010222 SecsKey found at position 2999
time for n = 3000 is 0.010153 SecsKey found at position 3999
time for n = 4000 is 0.010207 SecsKey found at position 4999
time for n = 5000 is 0.010169 Secs1.Binary Search  2.Linear Search  3. Exit
Enter your choice: 2
Key found at position 999
time for n = 1000 is 0.010210 SecsKey found at position 1999
time for n = 2000 is 0.010221 SecsKey found at position 2999
time for n = 3000 is 0.010222 SecsKey found at position 3999
time for n = 4000 is 0.010417 SecsKey found at position 4999
time for n = 5000 is 0.010450 Secs1.Binary Search  2.Linear Search  3. Exit
Enter your choice: 3

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

```

**TABLE VALUES AND GRAPH:**



**Program-3:**

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

```
#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void selsort(int n, int a[]);

void main()

{

    int a[15000],n,i,j,ch,temp;

    clock_t start,end;

    while(1)

    {

        printf("\n 1: For manual entry of N values and array elements:");

        printf("\n 2: To display time taken for sorting number of elements N in the range 500 to 14500:");

        printf("\n 3: To exit");

        printf("\n Enter your choice:");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1: printf("\n Enter the number of elements:");

                    scanf("%d",&n);

                    printf("\n Enter array elements:");

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

                    {

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

```

    }
    start=clock();
    selsort(n,a);
    end=clock();
    printf("\n Sorted array is:");
    for(i=0;i<n;i++){
        printf("%d\t",a[i]);
    }

    printf("\n Time taken to sort %d numbers is %f
secs",n,((double)(end-start)/CLOCKS_PER_SEC));
    break;
case 2:
n=500;
while(n<=14500){
    for(i=0;i<n;i++){
        a[i]=n-i;

    }
    start=clock();
    selsort(n,a);
    for(j=0;j<500000;j++){
        temp=38/600;

    }
    end=clock();

```



```
printf("\n Time taken to sort %d numbers is %f  
secs",n,((double)(end-start)/CLOCKS_PER_SEC));
```

```
    n=n+1000;
```

```
    }
```

```
    break;
```

```
    case 3: exit(0);
```

```
    }
```

```
    }
```

```
}
```

```
void selsort(int n, int a[])
```

```
{
```

```
    int i,j,t,small,pos;
```

```
    for(i=0;i<n-1;i++){
```

```
        pos=i;
```

```
        small=a[i];
```

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

```
            if(a[j]<small)
```

```
            {
```

```
                small=a[j];
```

```
                pos=j;
```

```
            }
```

```
    }
```

```

t=a[i];
a[i]=a[pos];
a[pos]=t;
}
}

```

### OUTPUT:

```

Input

Enter the number of elements:5

Enter array elements:5
4
3
2
1

Sorted array is:1      2      3      4      5
Time taken to sort 5 numbers is 0.000001 secs
1: For manual entry of N values and array elements:
2: To display time taken for sorting number of elements N in the range 500 to 14500:
3: To exit
Enter your choice:2

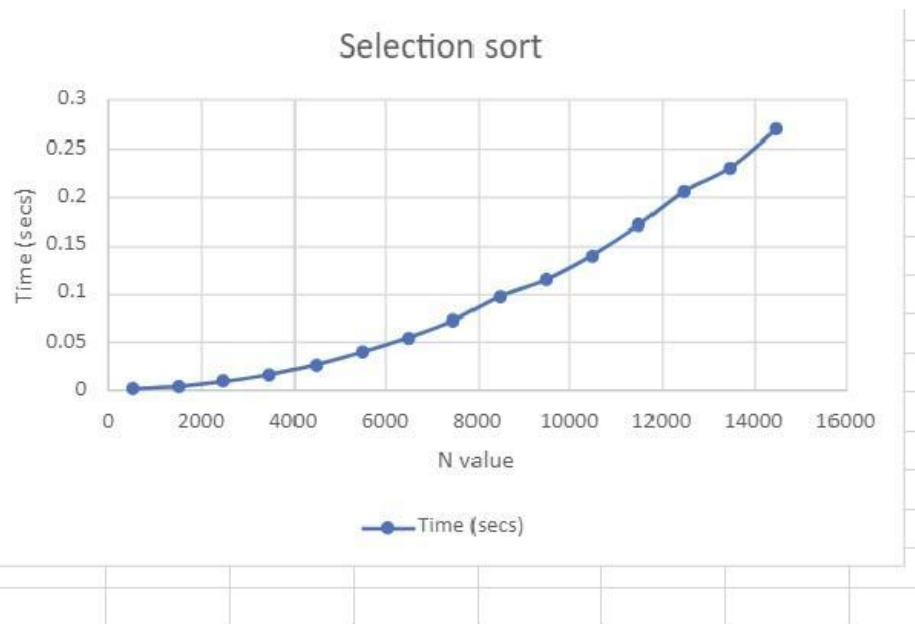
Time taken to sort 500 numbers is 0.001184 secs
Time taken to sort 1500 numbers is 0.003722 secs
Time taken to sort 2500 numbers is 0.008693 secs
Time taken to sort 3500 numbers is 0.016209 secs
Time taken to sort 4500 numbers is 0.026643 secs
Time taken to sort 5500 numbers is 0.039635 secs
Time taken to sort 6500 numbers is 0.054651 secs
Time taken to sort 7500 numbers is 0.072600 secs
Time taken to sort 8500 numbers is 0.097000 secs
Time taken to sort 9500 numbers is 0.114253 secs
Time taken to sort 10500 numbers is 0.139544 secs
Time taken to sort 11500 numbers is 0.170116 secs
Time taken to sort 12500 numbers is 0.205752 secs
Time taken to sort 13500 numbers is 0.230030 secs
Time taken to sort 14500 numbers is 0.269154 secs
1: For manual entry of N values and array elements:
2: To display time taken for sorting number of elements N in the range 500 to 14500:
3: To exit
Enter your choice:3

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

```

**TABLE VALUES AND GRAPH:**

N value	Time (secs)
500	0.001184
1500	0.003722
2500	0.008693
3500	0.016209
4500	0.026643
5500	0.039635
6500	0.054651
7500	0.0726
8500	0.097
9500	0.114253
10500	0.139544
11500	0.170116
12500	0.205752
13500	0.23003
14500	0.269154



**Program-4:**

*Write program to do the following:*

- a. Print all the nodes reachable from a given starting node in a digraph using BFS method.*
- b. Check whether a given graph is connected or not using DFS method.*

**a. BFS**

```
#include<stdio.h>
#include<conio.h>
int a[10][10],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);
getch();
}

```

```

void bfs(int src)
{
    int q[10],f=0,r=-1,vis[10],i,j;
    for(j=1;j<=n;j++)
    {
        vis[j]=0;
    }
    vis[src]=1;
    r=r+1;
    q[r]=src;
    while(f<=r)
    {
        i=q[f];
        f=f+1;
        for(j=1;j<=n;j++)
        {
            if(a[i][j]==1&&vis[j]!=1)
            {
                vis[j]=1;

```

```
    r=r+1;
    q[r]=j;
}
}
}
for(j=1;j<=n;j++)
{
    if(vis[j]!=1)
    {
        printf("\nnode %d is not reachable\n",j);
    }
    else
    {
        printf("\nnode %d is reachable\n",j);
    }
}
}
```

### OUTPUT:

```
input
enter the no of nodes: 6
enter the adjacency matrix:
0 1 1 1 0 0
0 0 0 0 1 0
0 0 0 0 1 1
0 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 1 0

enter the source node: 1

node 1 is reachable
node 2 is reachable
node 3 is reachable
node 4 is reachable
node 5 is reachable
node 6 is reachable

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

### b. DFS

```
#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("\ngraph 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:

```
input
enter the no of nodes: 4
enter the adjacency matrix:
0 1 1 0
0 0 0 0
0 0 0 1
0 1 0 0
enter the source node: 1
graph is connected
...Program finished with exit code 0
Press ENTER to exit console.
```

```
input
enter the no of nodes: 4
enter the adjacency matrix:
0 1 1 0
0 0 0 0
0 1 0 0
0 0 0 0
enter the source node: 1
graph is not connected
...Program finished with exit code 0
Press ENTER to exit console.
```

**Program-5:**

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

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

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

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

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

```
/*void printarray(int a[], int size) {  
    for (int i = 0; i < size; i++) {  
        printf("%d ", a[i]);  
    }  
    printf("\n");  
}*/
```

```
void insertionsort(int a[], int n) {  
    for (int i = 1; i < n; i++) {  
        int key = a[i];  
        int j = i - 1;  
        while (key < a[j] && j >= 0) {  
            a[j + 1] = a[j];  
            --j;  
        }  
        a[j + 1] = key;  
    }  
}
```

```
}
```

```
int main()
```

```
{
```

```
    int i,j,k,a[15000];
```

```
    int ch,n;
```

```
    clock_t start,end;
```

```
    while(1)
```

```
{
```

```
    printf("\n1:For manual entry of n value and array elements");
```

```
        printf("\n2:To display time taken for sorting number of elements");
```

```
    printf("\n3:Exit");
```

```
    printf("\nEnter your choice:");
```

```
    scanf("%d",&ch);
```

```
switch(ch)
{

    case 1:printf("\nEnter the number of elements: ");

    scanf("%d",&n);

    printf("\nEnter the array elements: ");

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

    {

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

    }

    start=clock();

    insertionsort(a,n);

    end=clock();

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

```
}
```

```
//printarray(a,n);
```

```
printf("\ntime taken to sort %d number of elements is %f  
secs",n,(((double)(end-start)/CLOCKS_PER_SEC)));
```

```
break;
```

```
case 2:
```

```
printf("\nrRunning values from 500 to 14500");
```

```
n=500;
```

```
while(n<=14500)
```

```
{
```

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

```
{
```

```
a[i]=rand();
```

```
}
```

```
start=clock();
```

```
insertionsort(a,n);
```

```
for(j=0;j<5000;j++)
```

```
{
```

```
int temp=38/600;
```

```
}
```

```
end=clock();
```

```
printf("\ntime taken to sort %d numbers is %f  
secs",n,(((double)(end-start)/CLOCKS_PER_SEC)));
```

```
n=n+1000;
```

```
}
```

```
break;
```

```
case 3:exit(0);
```

```
}
```

```
}
```

```
return 0;
```

```
}
```



## OUTPUT:

```
input
1:For manual entry of n value and array elements
2:To display time taken for sorting number of elements
3:Exit
Enter your choice:1

Enter the number of elements: 5

Enter the array elements: 5
4
3
2
1

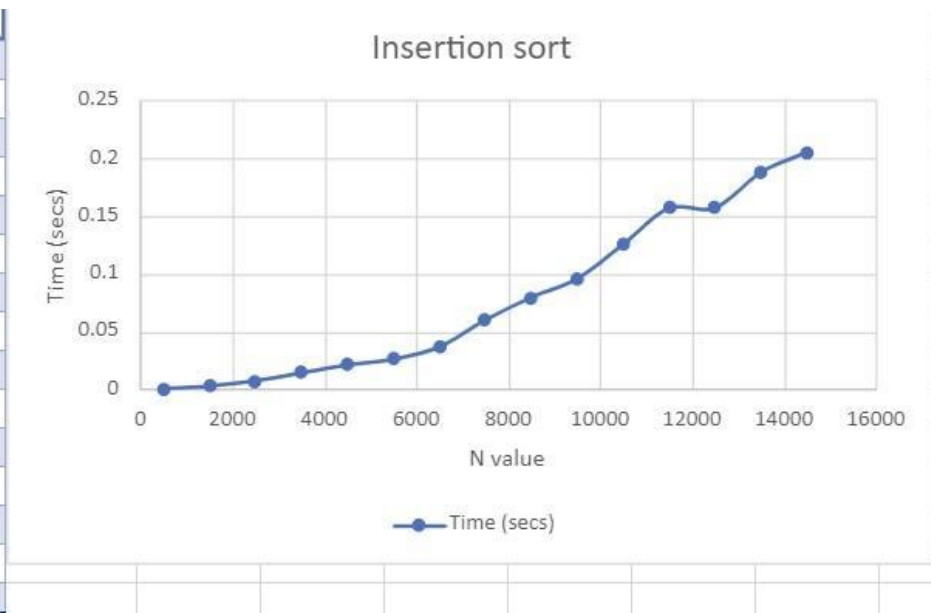
sorted array1 2 3 4 5
time taken to sort 5 number of elements is 0.000003 secs
1:For manual entry of n value and array elements
2:To display time taken for sorting number of elements
3:Exit
Enter your choice:2

Running values from 500 to 14500
time taken to sort 500 numbers is 0.000288 secs
time taken to sort 1500 numbers is 0.002515 secs
time taken to sort 2500 numbers is 0.007142 secs
time taken to sort 3500 numbers is 0.013971 secs
time taken to sort 4500 numbers is 0.020911 secs
time taken to sort 5500 numbers is 0.025815 secs
time taken to sort 6500 numbers is 0.036466 secs
time taken to sort 7500 numbers is 0.059718 secs
time taken to sort 8500 numbers is 0.079427 secs
time taken to sort 9500 numbers is 0.095532 secs
time taken to sort 10500 numbers is 0.125182 secs
time taken to sort 11500 numbers is 0.156899 secs
time taken to sort 12500 numbers is 0.156783 secs
time taken to sort 13500 numbers is 0.187800 secs
time taken to sort 14500 numbers is 0.205259 secs
1:For manual entry of n value and array elements
2:To display time taken for sorting number of elements
3:Exit
Enter your choice:3

...Program finished with exit code 0
```

**TABLE VALUES AND GRAPH:**

N value	Time (secs)
500	0.000288
1500	0.002515
2500	0.007142
3500	0.013971
4500	0.020911
5500	0.025815
6500	0.036466
7500	0.059718
8500	0.079427
9500	0.095532
10500	0.125182
11500	0.156899
12500	0.156783
13500	0.1878
14500	0.205259



**Program-6:**

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

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

void source_removal(int n, int a[10][10]) {
    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;
    }
}
}

```

```

for(i = 0; i < n; i++) {
    printf("%d\n", 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:

```
input
Enter number of nodes
6
Enter the adjacency matrix
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0
1
4
0
2
3
5

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

**Program-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("%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:**



```
input  
Johnson trotter algorithm to find all permutations of given numbers  
Enter the number  
3  
6  
All possible permutations are:  
1 2 3  
1 3 2  
3 1 2  
3 2 1  
2 3 1  
2 1 3  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

**Program-8:**

*Sort a given set of  $N$  integer elements using Merge Sort technique and compute its time taken. Run the program for different values of  $N$  and record the time taken to sort.*

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
void split(int[],int,int);
void combine(int[],int,int,int);
void main()
{
int a[15000],n,i,j,ch,temp;
clock_t start,end;
while(1)
{
printf("\n1.For manual entry of N value and array elements");
printf("\n2.To display time taken for sorting number elements N in the range 500 to 14500");
printf("\n3.To exit");
printf("\nEnter your choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the number of elements:");
```

```
scanf("%d",&n);
printf("\nEnter array elements:");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
start=clock();
split(a,0,n-1);
end=clock();
printf("\nSorted array is:");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n,
(((double)(end-start))/CLOCKS_PER_SEC));
break;
```

```
case 2:
n=500;
while(n<=15000)
{
for(i=0;i<n;i++)
{
a[i]=n-i;
}
start=clock();
split(a,0,n-1);
```

```
for(j=0;j<50000000;j++)
{
temp=38/600;
}
end=clock();
printf("\n Time taken to sort %d numbers is %f
Secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
n=n+1000;
}
break;
```

```
case 3:exit(0);
```

```
}
```

```
getchar();
```

```
}
```

```
}
```

```
void split(int a[],int low, int high){
```

```
    int mid;
```

```
    if(low<high){
```

```
        mid=(low+high)/2;
```

```
        split(a,low,mid);
```

```
        split(a,mid+1,high);
```

```
        combine(a,low,mid,high);
```

```
    }
```

```
}
```

```

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

```

```

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

```

### OUTPUT:

Input

```

1.For manual entry of N value and array elements
2.To display time taken for sorting number elements N in the range 500 to 14500
3.To exit
Enter your choice1

Enter the number of elements:5

Enter array elements:5
4
3
2
1

Sorted array is:1      2      3      4      5
Time taken to sort 5 numbers is 0.000031 Secs
1.For manual entry of N value and array elements
2.To display time taken for sorting number elements N in the range 500 to 14500
3.To exit
Enter your choice2

Time taken to sort 500 numbers is 0.081458 Secs
Time taken to sort 1500 numbers is 0.081562 Secs
Time taken to sort 2500 numbers is 0.081957 Secs
Time taken to sort 3500 numbers is 0.082062 Secs
Time taken to sort 4500 numbers is 0.082271 Secs
Time taken to sort 5500 numbers is 0.081892 Secs
Time taken to sort 6500 numbers is 0.082415 Secs
Time taken to sort 7500 numbers is 0.082636 Secs
Time taken to sort 8500 numbers is 0.082641 Secs
Time taken to sort 9500 numbers is 0.082886 Secs
Time taken to sort 10500 numbers is 0.083184 Secs
Time taken to sort 11500 numbers is 0.083216 Secs
Time taken to sort 12500 numbers is 0.083265 Secs
Time taken to sort 13500 numbers is 0.083121 Secs
Time taken to sort 14500 numbers is 0.083892 Secs
1.For manual entry of N value and array elements
2.To display time taken for sorting number elements N in the range 500 to 14500
3.To exit
Enter your choice3

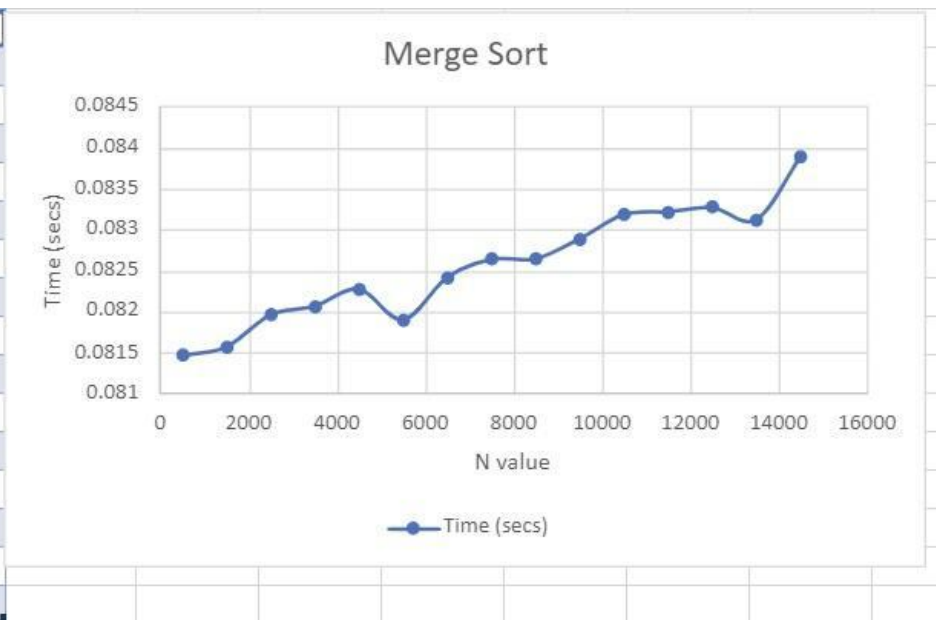
...Program finished with exit code 0
Press ENTER to exit console.

```



**TABLE VALUES AND GRAPH:**

N value	Time (secs)
500	0.081458
1500	0.081562
2500	0.081957
3500	0.082062
4500	0.082271
5500	0.081892
6500	0.082415
7500	0.082636
8500	0.082641
9500	0.082886
10500	0.083184
11500	0.083216
12500	0.083265
13500	0.083121
14500	0.083892



**Program-9:**

*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>
int partition(int a[],int low,int high)
{
    int start=low;
    int end=high;
    int pivot=a[low];
    int i,j,temp;

    while(start<end)
    {
        while(a[start]<=pivot)
        {
            start++;
        }

        while(a[end]>pivot)
        {
            end--;
        }
    }
}
```

```
    if(start<end)
    {

        temp=a[start];
        a[start]=a[end];
        a[end]=temp;
    }
}
temp=a[low];
a[low]=a[end];
a[end]=temp;
return end;
}

void quicksort(int a[],int low,int high)
{
    int loc;
    if(low<high)
    {
        loc=partition(a,low,high);
        quicksort(a,low,loc-1);
        quicksort(a,loc+1,high);
    }
}
```

```

void main()
{
int a[15000],n,i,j,ch,temp;
clock_t start,end;
while(1)
{
printf("\n1.For manual entry of N value and array elements");
printf("\n2.To display time taken for sorting number elements N in the range 500 to 14500");
printf("\n3.To exit");
printf("\nEnter your choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the number of elements:");
scanf("%d",&n);
printf("\nEnter array elements:");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
start=clock();
quicksort(a,0,n-1);
end=clock();
printf("\nSorted array is:");

```

```
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n,
(((double)(end-start))/CLOCKS_PER_SEC));
break;
```

case 2:

```
n=500;
while(n<=15000)
{
for(i=0;i<n;i++)
{
a[i]=n-i;
}
start=clock();
quicksort(a,0,n-1);
```

```
for(j=0;j<50000000;j++)
{
temp=38/600;
}
end=clock();
printf("\n Time taken to sort %d numbers is %f
Secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
n=n+1000;
}
```

```
break;
```

```
case 3:exit(0);
```

```
}
```

```
getchar();
```

```
}
```

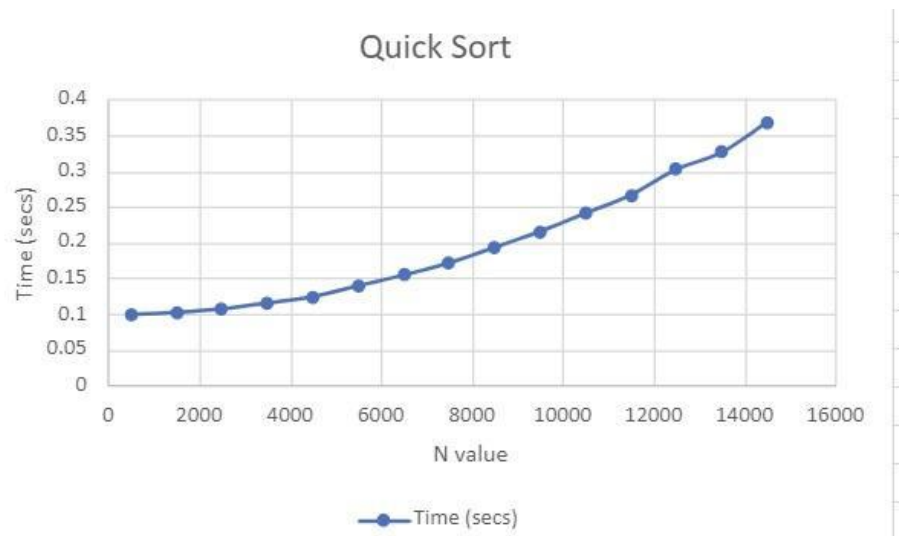
```
}
```

### OUTPUT:

```
Input
1.For manual entry of N value and array elements
2.To display time taken for sorting number elements N in the range 500 to 14500
3.To exit
Enter your choice1
Enter the number of elements:5
Enter array elements:5
4
3
2
1
Sorted array is:1      2      3      4      5
Time taken to sort 5 numbers is 0.000038 Secs
1.For manual entry of N value and array elements
2.To display time taken for sorting number elements N in the range 500 to 14500
3.To exit
Enter your choice2
Time taken to sort 500 numbers is 0.099207 Secs
Time taken to sort 1500 numbers is 0.102361 Secs
Time taken to sort 2500 numbers is 0.107224 Secs
Time taken to sort 3500 numbers is 0.115543 Secs
Time taken to sort 4500 numbers is 0.124213 Secs
Time taken to sort 5500 numbers is 0.139525 Secs
Time taken to sort 6500 numbers is 0.154606 Secs
Time taken to sort 7500 numbers is 0.171595 Secs
Time taken to sort 8500 numbers is 0.193204 Secs
Time taken to sort 9500 numbers is 0.215557 Secs
Time taken to sort 10500 numbers is 0.241029 Secs
Time taken to sort 11500 numbers is 0.266699 Secs
Time taken to sort 12500 numbers is 0.302377 Secs
Time taken to sort 13500 numbers is 0.326285 Secs
Time taken to sort 14500 numbers is 0.368597 Secs
1.For manual entry of N value and array elements
2.To display time taken for sorting number elements N in the range 500 to 14500
3.To exit
Enter your choice3
...Program finished with exit code 0
Press ENTER to exit console.
```

**TABLE VALUES AND GRAPH:**

N value	Time (secs)
500	0.099207
1500	0.102361
2500	0.107224
3500	0.115543
4500	0.124213
5500	0.139525
6500	0.154606
7500	0.171595
8500	0.193204
9500	0.215557
10500	0.241029
11500	0.266699
12500	0.302377
13500	0.326285
14500	0.368597



**Program-10:**

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

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
void swap(int *,int *);
void heapify(int [],int,int);
void heapSort(int[], int);
void main()
{
    int a[15000],n,i,j,ch,temp;
    clock_t start,end;
    while(1)
    {
        printf("\n 1: For manual entry of N values and array elements:");
        printf("\n 2: To display time taken for sorting number of elements N in the
range 500 to 14500:");
        printf("\n 3: To exit");
        printf("\n Enter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: printf("\n Enter the number of elements:");
                scanf("%d",&n);
```



```

printf("\n Enter array elements:");
for(i=0;i<n;i++)
{
    scanf("%d",&a[i]);
}
start=clock();
heapSort(a, n);
end=clock();
printf("\n Sorted array is:");
for(i=n-1;i>=0;i--){
printf("%d\t",a[i]);
}

printf("\n Time taken to sort %d numbers is %f
secs",n,((double)(end-start)/CLOCKS_PER_SEC));
break;
case 2:
n=500;
while(n<=14500){
    for(i=0;i<n;i++){
        a[i]=n-i;
    }
    start=clock();
    heapSort(a, n);
    for(j=0;j<50000000;j++){

```

```

        temp=38/600;

    }

    end=clock();

    printf("\n Time taken to sort %d numbers is %f
secs",n,((double)(end-start)/CLOCKS_PER_SEC));

    n=n+1000;

}

break;


case 3: exit(0);
}

}

}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

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

```

```

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

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

```

## OUTPUT:

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

Enter the number of elements:5

Enter array elements:1
2
3
4
5

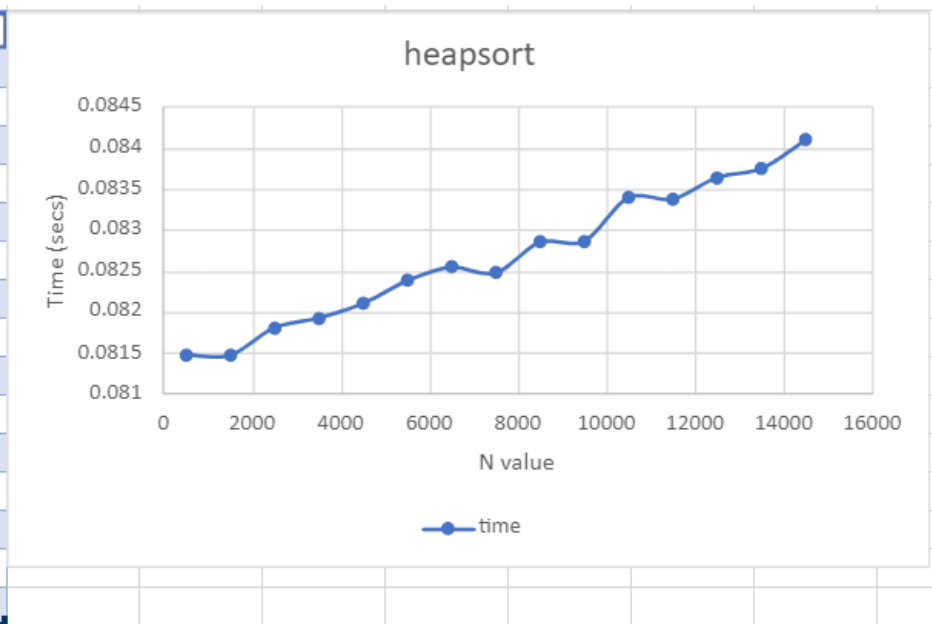
Sorted array is:5      4      3      2      1
Time taken to sort 5 numbers is 0.000002 secs
1: For manual entry of N values and array elements:
2: To display time taken for sorting number of elements N in the range 500 to 14500:
3: To exit
Enter your choice:
2

Time taken to sort 500 numbers is 0.081471 secs
Time taken to sort 1500 numbers is 0.081463 secs
Time taken to sort 2500 numbers is 0.081800 secs
Time taken to sort 3500 numbers is 0.081919 secs
Time taken to sort 4500 numbers is 0.082095 secs
Time taken to sort 5500 numbers is 0.082375 secs
Time taken to sort 6500 numbers is 0.082540 secs
Time taken to sort 7500 numbers is 0.082467 secs
Time taken to sort 8500 numbers is 0.082848 secs
Time taken to sort 9500 numbers is 0.082855 secs
Time taken to sort 10500 numbers is 0.083393 secs
Time taken to sort 11500 numbers is 0.083372 secs
Time taken to sort 12500 numbers is 0.083636 secs
Time taken to sort 13500 numbers is 0.083745 secs
Time taken to sort 14500 numbers is 0.084091 secs
1: For manual entry of N values and array elements:
2: To display time taken for sorting number of elements N in the range 500 to 14500:
3: To exit
Enter your choice:Killed

...Program finished with exit code 9
```

**TABLE VALUES AND GRAPH:**

n value	time
500	0.081471
1500	0.081463
2500	0.0818
3500	0.081919
4500	0.082095
5500	0.082375
6500	0.08254
7500	0.082467
8500	0.082848
9500	0.082855
10500	0.083393
11500	0.083372
12500	0.083636
13500	0.083745
14500	0.084091



**Program-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;
```

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

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

```
            r[k][i][j]=a[i][j];
```

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

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

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

```
                r[k][i][j]=r[k-1][i][j] || (r[k-1][i][k] && r[k-1][k][j]);
```

```
    }
```

```
int main(){
```

```
    int n;
```

```
    printf("Enter no of vertices: \n");
```

```
    scanf("%d",&n);
```

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

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

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

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

```
    warshall(n);
```

```

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

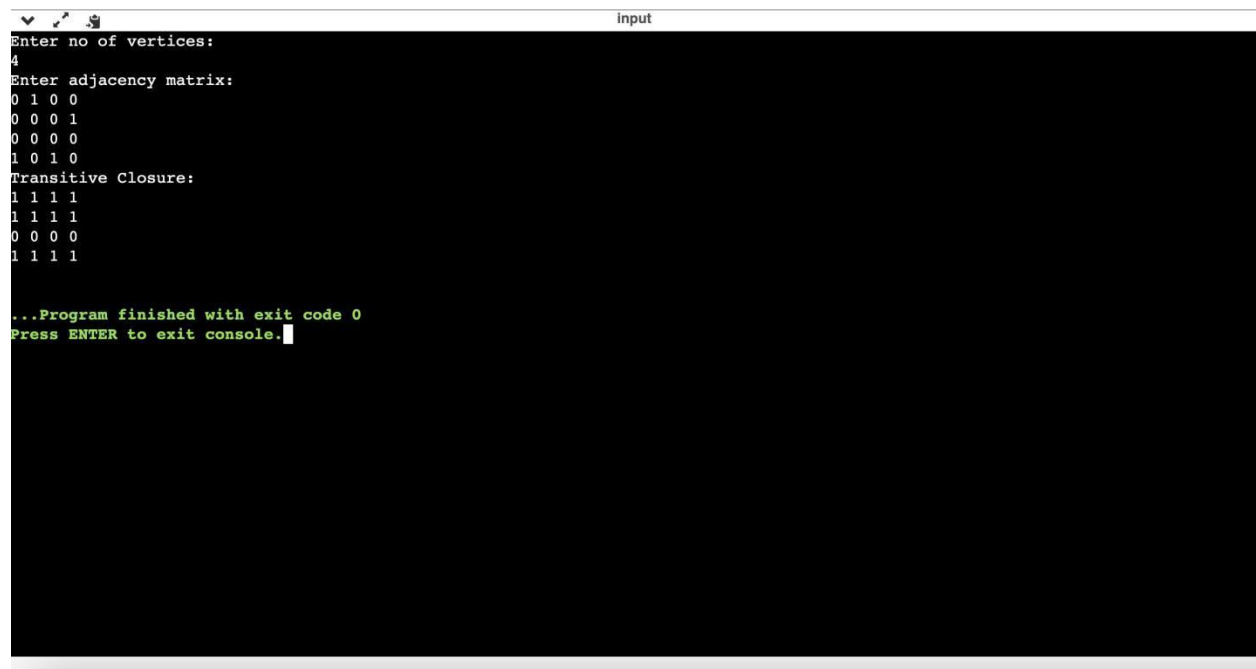
    }

    printf("\n");

}
}

```

#### **OUTPUT:**



The screenshot shows a terminal window titled "input" with the following output:

```

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

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

```

**Program-12:**

*Implement 0/1 Knapsack problem using dynamic programming.*

```
#include <stdio.h>
#include<conio.h>
int max(int,int);
void knapsack();
int w[10],p[10],n,m,i,j,k,v[10][10];
int main()
{
    printf("\nenter the number of items\n");
    scanf("%d",&n);
    printf("\nenter the weight of 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 capacity \n");
    scanf("%d",&m);

    knapsack();
    return 0;
```



```

}
void knapsack()
{
    int x[10];
    k=0;
    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("\nOutput matrix is\n");
for(i=0;i<=n;i++)
{

```

```

    for(j=0;j<=m;j++)
    {
        printf(" %d ",v[i][j]);
    }
    printf("\n\n");
}
printf("\nOptimal solution is %d \n",v[n][m]);
printf("\nSolution vector is \n");
for(i=n;i>=1;i--)
{
    if(v[i][m]!=v[i-1][m])
    {
        x[i]=1;
        m=m-w[i];
    }
    else
    {
        x[i]=0;
    }
}
for(i=1;i<=n;i++)
{
    printf(" %d ",x[i]);
}
}

```

```
int max(int x,int y)
```

```
{
```

```
    if(x>y)
```

```
    {
```

```
        return x;
```

```
    }
```

```
    else
```

```
    {
```

```
        return y;
```

```
    }
```

```
}
```

**OUTPUT:**

```
enter the number of items
4

enter the weight of each item
2 1 3 2

enter the profit of each item
12 10 20 15

enter the knapsack capacity
5

output matrix is
0 0 0 0 0 0

0 0 12 12 12 12

0 10 12 22 22 22

0 10 12 22 30 32

0 10 15 25 30 37

optimal solution is 37

solution vector is
1 1 0 1
```

**Program-13 :**

*Implement All Pair Shortest Paths problem using Floyd's algorithm.*

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

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

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

```
void floyds();
```

```
int min(int,int);
```

```
void main()
```

```
{
```

```
int i,j;
```

```
printf("\nenter the no. of vertices:\t");
```

```
scanf("%d",&n);
```

```
printf("\nenter the cost matrix:\n");
```

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

```
{
```

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

```
{
```

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

```
}
```

```
}
```

```
floyds();
```

```
getch();
```

```
}
```

```
void floyds()
```

```
{
```

```
int i,j,k;

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

{

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

    {

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

        {

            a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

        }

    }

}

printf("\nall pair shortest path matrix is:\n");

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

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

```
int min(int x,int y)
```

```
{  
  
if(x<y)  
  
{  
  
return x;
```



```
}
```

```
else
```

```
{
```

```
    return y;
```

```
}
```

```
}
```

**OUTPUT:**

```
enter the no. of vertices:      4
```

```
enter the cost matrix:
```

```
9999 9999      3 9999
      2 9999 9999 9999
9999      7 9999      1
      6 9999 9999 9999
```

```
all pair shortest path matrix is:
```

```
10      10      3      4
2       12      5      6
7       7       10     1
6       16      9     10
```

**Program-14 :**

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

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

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

```
void prims();
```

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

```
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]);  
  
}  
  
}  
  
prims();  
return 0;  
}
```

```
void prims()
```

```
{
```

```
    int i,j,u,v,min;
```

```
    int ne=0,mincost=0;
```

```
    int elec[10];
```

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

```
{
```

```
    elec[i]=0;
```

```
}
```

```
elec[1]=1;
```

```
while(ne!=n-1)
```

```
{
```

```
    min=9999;
```

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

```
    {
```

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

```
        {
```

```
            if(elec[i]==1)
```

```
            {
```

```
if(c[i][j]<min)

{

    min=c[i][j];

    u=i;

    v=j;

}

}

}

}

if(elec[v]!=1)

{

    printf("\n%d----->%d=%d\n",u,v,min);

    elec[v]=1;
```

```
ne=ne+1;
```

```
mincost=mincost+min;
```

```
}
```

```
c[u][v]=c[v][u]=9999;
```

```
}
```

```
printf("\nmincost=%d",mincost);
```

```
}
```

output:

```
enter the no. of vertices:      6
```

```
enter the cost matrix:
```

```
9999      3 9999 9999      6      5
      3 9999      1 9999 9999      4
9999      1 9999      6 9999      4
9999      6      6 9999      8      5
      6 9999 9999      8 9999      2
      5      4      4      5      2 9999
```

```
1----->2=3
```

```
2----->3=1
```

```
2----->6=4
```

```
6----->5=2
```

```
6----->4=5
```

```
mincost=15
```



**Program-15:**

*Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's Algorithm.*

```
#include<stdio.h>

void kruskals();

int c[10][10],n;

void main()
{
    int i,j;
    printf("\nEnter the no. of vertices:\t");
    scanf("%d",&n);
    printf("\nEnter the cost matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&c[i][j]);
        }
    }
    kruskals();
}

void kruskals()
{
    int i,j,u,v,a,b,min;
    int ne=0,mincost=0;
```

```
int parent[10];
for(i=1;i<=n;i++)
{
    parent[i]=0;
}
while(ne!=n-1)
{
    min=9999;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            if(c[i][j]<min)
            {
                min=c[i][j];
                u=a=i;
                v=b=j;
            }
        }
    }
    while(parent[u]!=0)
    {
        u=parent[u];
    }
    while(parent[v]!=0)
    {
```

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

**OUTPUT:**

```
enter the no. of vertices:      6

enter the cost matrix:
9999      3 9999 9999      6      5
      3 9999      1 9999 9999      4
9999      1 9999      6 9999      4
9999      6      6 9999      8      5
      6 9999 9999      8 9999      2
      5      4      4      5      2 9999

2----->3=1

5----->6=2

1----->2=3

2----->6=4

4----->6=5

mincost=15
```

**Program-16 :**

*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>
void dijkstras();
int c[10][10],n,src;
void main()
{
    int i,j;

    printf("\nEnter the no of vertices:\t");
    scanf("%d",&n);
    printf("\nEnter the cost matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&c[i][j]);
        }
    }
    printf("\nEnter the source node:\t");
    scanf("%d",&src);
    dijkstras();
    getch();
}
```

```

}
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:

```
enter the number of vertices
5
```

```
enter the cost matrix
```

```
9999      3 9999      7 9999
      3 9999      4      5 9999
9999      4 9999      5      6
      7      2      5 9999      4
9999 9999      6      4 9999
```

```
enter the source vertex
```

```
1
```

```
shortest distance is
```

```
1 -----> 1 = 0
```

```
1 -----> 2 = 3
```

```
1 -----> 3 = 7
```

```
1 -----> 4 = 7
```

```
1 -----> 5 = 11
```