VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

M GAGAN (1BM20CS222)

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

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by
M GAGAN (1BM20CS222), 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.

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

Index Sheet

	example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.	
18	Implement "N-Queens Problem" using Backtracking.	

Course Outcome

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

1) Write a recursive program to Solve

a) Towers-of-Hanoi problem

b) To find GCD

a) PROGRAM:-

```c
#include <stdio.h>
void towers(int, char, char, char);
int main()
{
    int num;
    printf("Enter the number of disks : ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers(num, 'A', 'C', 'B');
    return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
    if (num == 1)
    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```

OUTPUT:-

```
Enter the number of disks : 3
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from peg A to peg C
 Move disk 2 from peg A to peg B
 Move disk 1 from peg C to peg B
 Move disk 3 from peg A to peg C
 Move disk 1 from peg B to peg A
 Move disk 2 from peg B to peg C
 Move disk 1 from peg A to peg C

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

b) PROGRAM:-

```c
#include<stdio.h>
int gcd(int,int);
void main()
{
    int m, n;
    scanf("%d %d",&m,&n);
    int ans = gcd(m,n);
    printf("%d",ans);
}
int gcd(int m,int n)
{
    if(n!=0)
    {
        return gcd(n,m%n);
    }
    else
    {
        return m;
    }
}
```

OUTPUT:-

```
ENTER 2 POSITIVE INTEGERS
6 4
G.C.D of 6 and 4 is 2.

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

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.

PROGRAM:-

```c
#include<stdio.h>

#include<time.h>

#include<math.h>

#include<stdlib.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;

 long int i, j;


 while(1)

 {

 printf("\nenter the choice: 1.linear search 2.binary search");

 scanf("%d",&ch);

 switch(ch)

 {

  case 2:

    n=1000;

         while(n<=5000)

         {

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

         {
```

```c
            a[i]=i;
            }
            key=a[n-1];
            start=clock();


            search_status=bin_srch(a,0,n-1,key);
            if(search_status==-1)
        printf("\nKey Not Found");
      else
        printf("\n seach element is found %d",search_status);


            for(j=0;j<500000;j++){ temp=38/600;}
            end=clock();
            printf("\nTIME FOR n=%d IS %f SECS",n,(((double)(end-start))/CLOCKS_PER_SEC));
            n=n+1000;
            }
            break;
  case   1:
        n=1000;
            while(n<=5000)
            {
        for(i=0;i<n;i++)
            {


             a[i]=i;
            }
            key=a[n-1];
            start=clock();
            search_status=lin_srch(a,0,n-1,key);
            if(search_status==-1)
```

```c
            printf("\nKey Not Found");
        else
            printf("\n seach element is found");


            for(j=0;j<500000;j++){ temp=38/600;}
            end=clock();
            printf("\ntime for n=%d is %f secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
            n=n+1000;
            }
            break;
    default:
        exit(0);
 }
 getchar();
 }
}


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])
```

```c
    {
    return bin_srch(a,low,mid-1,key);
    }
    else
    {
    return bin_srch(a,mid+1,high,key);
    }
    }

int lin_srch(int a[],int i,int high,int key)
{
 if(i>high)
 {
 return -1;
 }
 if(key==a[i])
 {
 return i;
 }
 else
 {
 return lin_srch(a,i+1,high,key);
 }
}
```

OUTPUT:-

```
enter the choice: 1.linear search 2.binary search1

search element is found
time for n=1000 is 0.804261 secs
search element is found
time for n=2000 is 1.609531 secs
search element is found
time for n=3000 is 2.393621 secs
search element is found
time for n=4000 is 3.213841 secs
search element is found
time for n=5000 is 3.972152 secs

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

```
enter the choice: 1.linear search 2.binary search2

search element is found
time for n=1000 is 0.010167 secs
search element is found
time for n=2000 is 0.011159 secs
search element is found
time for n=3000 is 0.012174 secs
search element is found
time for n=4000 is 0.012470 secs
search element is found
time for n=5000 is 0.013195 secs

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

Time(LS) and Time(BS)
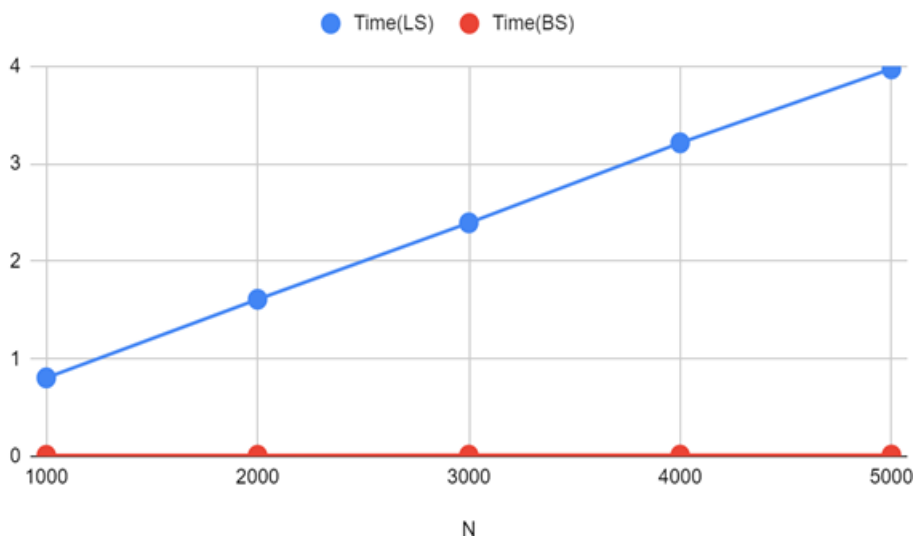
3)Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort

PROGRAM:-

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include<math.h>
void selsort(int n, int a[]);
int main()
{
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;
    while (1)
    {
        printf("\n1:FOR MANUAL ENTRY");
        printf("\n2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000");
        printf("\n3:EXIT");
        printf("\nENTER YOUR CHOICE:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nENTER NUMBER OF ELEMENTS: ");
            scanf("%d", &n);
            printf("\nENTER ARRAY ELEMENTS: ");
            for (i = 0; i < n; i++)
            {
                scanf("%d", &a[i]);
            }
```

```c
        start = clock();

        selsort(n, a);

        end = clock();

        printf("\nSORTED ELEMENTS 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]=rand()%n;

                //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:
```

```c
            exit(0);
        }
        getchar();
    }
}


void selsort(int n, int a[])
{
    int i, j, t, small, pos;
    for (i = 0; i < n - 1; i++)
    {
        pos = i;
        small = a[i];
        for (j = i + 1; j < n; j++)
        {
            if (a[j] < small)
            {
                small = a[j];
                pos = j;
            }
        }
        t = a[i];
        a[i] = a[pos];
        a[pos] = t;
    }
}
```

OUTPUT:-

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:1

ENTER NUMBER OF ELEMENTS: 5

ENTER ARRAY ELEMENTS: 5 4 3 2 1

SORTED ELEMENTS IS: 1    2        3        4        5
 TIME TAKEN TO SORT 5 NUMBERS IS 0.000003 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


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

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:2

 TIME TAKEN TO SORT 500 NUMBERS IS 0.001418 SECS
 TIME TAKEN TO SORT 1500 NUMBERS IS 0.004573 SECS
 TIME TAKEN TO SORT 2500 NUMBERS IS 0.010871 SECS
 TIME TAKEN TO SORT 3500 NUMBERS IS 0.020120 SECS
 TIME TAKEN TO SORT 4500 NUMBERS IS 0.032745 SECS
 TIME TAKEN TO SORT 5500 NUMBERS IS 0.047732 SECS
 TIME TAKEN TO SORT 6500 NUMBERS IS 0.066943 SECS
 TIME TAKEN TO SORT 7500 NUMBERS IS 0.093525 SECS
 TIME TAKEN TO SORT 8500 NUMBERS IS 0.111119 SECS
 TIME TAKEN TO SORT 9500 NUMBERS IS 0.139058 SECS
 TIME TAKEN TO SORT 10500 NUMBERS IS 0.170597 SECS
 TIME TAKEN TO SORT 11500 NUMBERS IS 0.204679 SECS
 TIME TAKEN TO SORT 12500 NUMBERS IS 0.242964 SECS
 TIME TAKEN TO SORT 13500 NUMBERS IS 0.286544 SECS
 TIME TAKEN TO SORT 14500 NUMBERS IS 0.325380 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


...Program finished with exit code 0
Press ENTER to exit console.
```
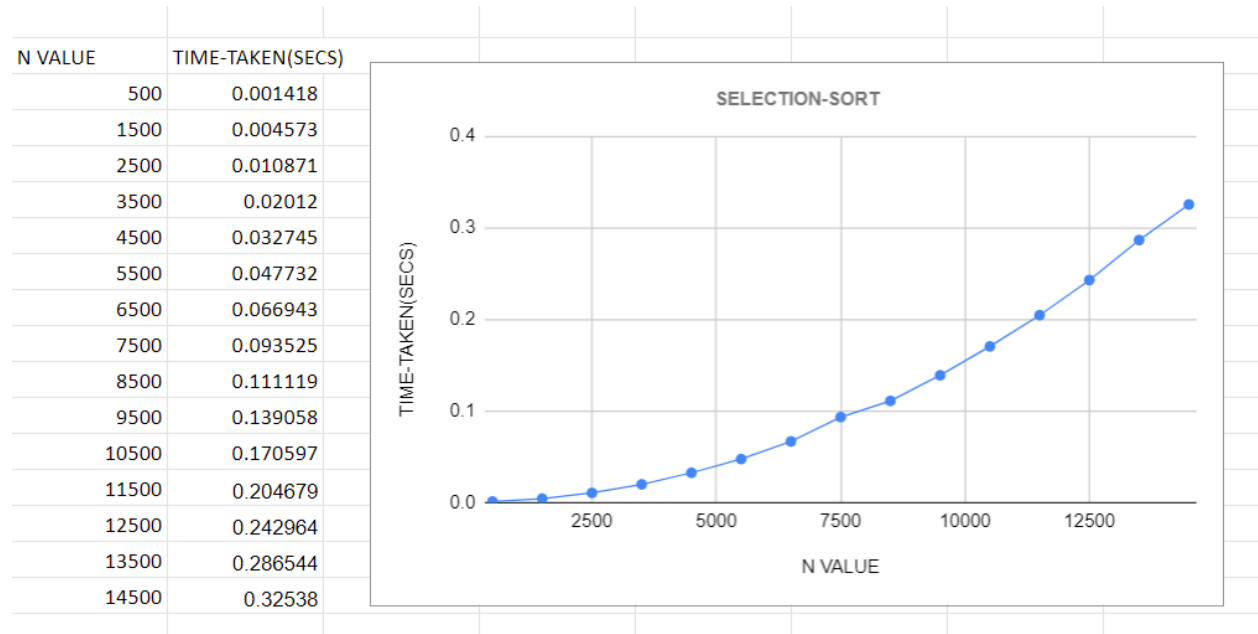
GRAPH: Generated graph for N value vs Time taken for Selection sort for different values of N

N VALUE	TIME-TAKEN(SECS)
500	0.001418
1500	0.004573
2500	0.010871
3500	0.02012
4500	0.032745
5500	0.047732
6500	0.066943
7500	0.093525
8500	0.111119
9500	0.139058
10500	0.170597
11500	0.204679
12500	0.242964
13500	0.286544
14500	0.32538

SELECTION-SORT

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)PROGRAM:-

```c
#include <stdio.h>
#include <conio.h>
int a[10][10], n;
void bfs(int);
int main()
{
    int i, j, src;
    printf("\nENTER NUMBER OF NODES:\t");
    scanf("%d", &n);
    printf("\nENTER ADJACENCY MATRIX:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf("\nENTER SOURCE NODE:\t");
    scanf("%d", &src);
    bfs(src);
}

void bfs(int src)
{
    int q[10], f = 0, r = -1, vis[10], i, j;
    for (j = 1; j <= n; j++)
```

```c
{
    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:-

```
ENTER NUMBER OF NODES:   6

ENTER 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 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) PROGRAM:-

```c
#include<stdio.h>
#include<conio.h>
int a[10][10],n,vis[10];
int dfs(int);
int main()
{
int i,j,src,ans;
for(j=1;j<=n;j++)
{
vis[j]=0;
}
printf("\nENTER NO OF NODES:\t");
scanf("%d",&n);
printf("\nENTER ADJACENCY MATRIX:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nENTER SOURCE NODE:");
scanf("%d",&src);
ans=dfs(src);
if(ans==1)
{
printf("\nGRAPH IS CONNECTED\n");
}
else
```

```c
   {
    printf("\nGRAPH IS NOT CONNECTED\n");
   }
  }

  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:-

```
ENTER NO OF NODES: 4

ENTER ADJACENCY MATRIX:
0 1 1 0
0 0 0 0
0 0 0 1
0 1 0 0

ENTER SOURCE NODE:1

GRAPH IS CONNECTED


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

```
ENTER NO OF NODES: 4

ENTER ADJACENCY MATRIX:
0 1 1 0
0 0 0 0
0 1 0 0
0 0 0 0

ENTER SOURCE NODE:1

GRAPH IS NOT CONNECTED


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

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

PROGRAM:-

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

void insertionSort(int arr[], int n)
{
        int i, key, j;
        for (i = 1; i < n; i++)
        {
                key = arr[i];
                j = i - 1;

                while (j >= 0 && arr[j] > key)
                {
                        arr[j + 1] = arr[j];
                        j = j - 1;
                }
                arr[j + 1] = key;
        }
}

int main()
{
        int i, n;
   clock_t start, end;
   printf("ENTER ARRAY SIZE =");
```

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

    int arr[150000];

    printf("ENTER ARRAY ELEMENTS");

    for (int j = 0; j < n; j++)

    {

        arr[j] = rand() % 10000;

    }

    printf("\n");

    start = clock();

    insertionSort(arr, n);

    end = clock();


    printf("\nSORTED ELEMNETS = ");

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

        printf(" %d", arr[i]);

    printf("\n TIME TAKEN TO SORT %d NUMBERS IS %f SECS", n, (((double)(end - start)) /
CLOCKS_PER_SEC));

    return 0;

}
```

OUTPUT:-

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:1

ENTER NUMBER OF ARRAY 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
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


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

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:2

 TIME TAKEN TO SORT 500 NUMBERS IS 0.001006 SECS
 TIME TAKEN TO SORT 1500 NUMBERS IS 0.002282 SECS
 TIME TAKEN TO SORT 2500 NUMBERS IS 0.005011 SECS
 TIME TAKEN TO SORT 3500 NUMBERS IS 0.009002 SECS
 TIME TAKEN TO SORT 4500 NUMBERS IS 0.014338 SECS
 TIME TAKEN TO SORT 5500 NUMBERS IS 0.020480 SECS
 TIME TAKEN TO SORT 6500 NUMBERS IS 0.028249 SECS
 TIME TAKEN TO SORT 7500 NUMBERS IS 0.039292 SECS
 TIME TAKEN TO SORT 8500 NUMBERS IS 0.048375 SECS
 TIME TAKEN TO SORT 9500 NUMBERS IS 0.062193 SECS
 TIME TAKEN TO SORT 10500 NUMBERS IS 0.076936 SECS
 TIME TAKEN TO SORT 11500 NUMBERS IS 0.094794 SECS
 TIME TAKEN TO SORT 12500 NUMBERS IS 0.122660 SECS
 TIME TAKEN TO SORT 13500 NUMBERS IS 0.140906 SECS
 TIME TAKEN TO SORT 14500 NUMBERS IS 0.158705 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


...Program finished with exit code 0
Press ENTER to exit console.
```
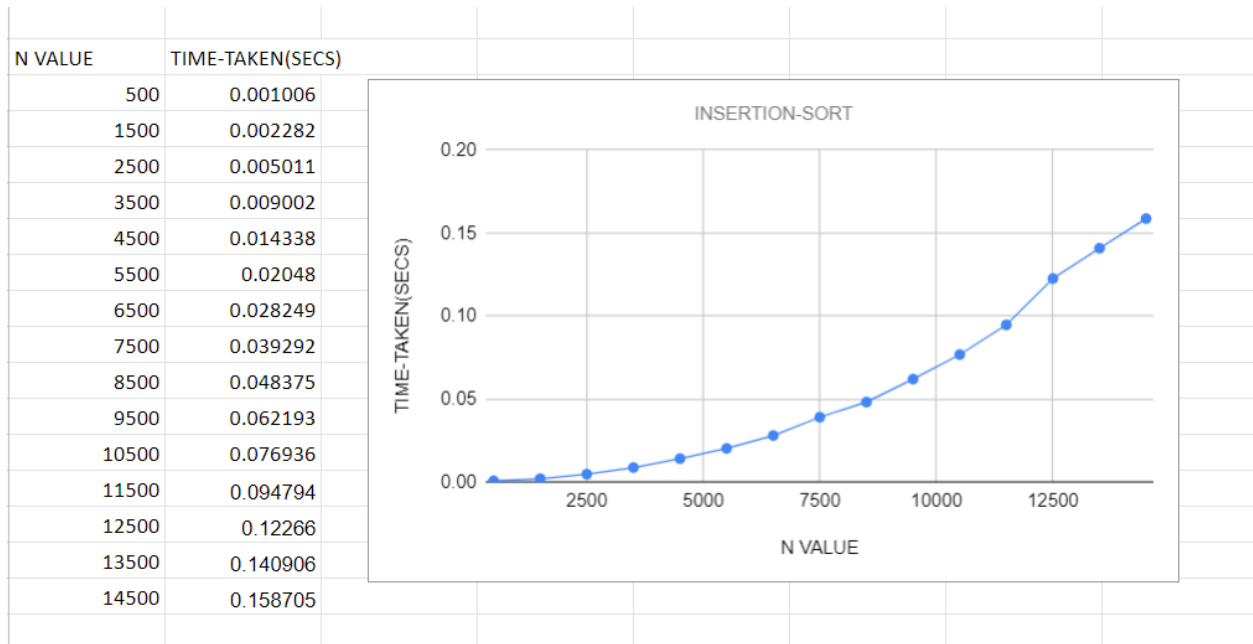
GRAPH: Generated graph for N value vs Time taken for Selection sort for different values of N

N VALUE	TIME-TAKEN(SECS)
500	0.001006
1500	0.002282
2500	0.005011
3500	0.009002
4500	0.014338
5500	0.02048
6500	0.028249
7500	0.039292
8500	0.048375
9500	0.062193
10500	0.076936
11500	0.094794
12500	0.12266
13500	0.140906
14500	0.158705

INSERTION-SORT

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

PROGRAM:-

```c
#include<stdio.h>
#include<conio.h>
void source_removal(int n, int a[10][10])
{
        int i,j,k,u,v,top;
   int 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--];
```

```c
                        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]);
        }
}
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);
```

}

```
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:-

```c
#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)
```

```c
{
    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
```

```c
        {
            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++)
    {
```

```c
        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("The total permutations are %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:-

```
Johnson trotter algorithm to find all permutations of given numbers
Enter the number
3
The total permutations are 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.
```

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.

PROGRAM:-

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
void split(int[],int,int);
void combine(int[],int,int,int);
int main()
{
   int a[15000], n, i, j, ch, temp;
   clock_t start, end;
   while (1)
   {
      printf("\n1:FOR MANUAL ENTRY");
      printf("\n2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000");
      printf("\n3:EXIT");
      printf("\nENTER YOUR CHOICE:");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         printf("\nENTER NUMBER OF ARRAY ELEMENTS: ");
         scanf("%d", &n);
         printf("\nENTER ARRAY ELEMENTS: ");
         for (i = 0; i < n; i++)
         {
            scanf("%d", &a[i]);
         }
```

```c
        start = clock();

        split(a, 0, n - 1);

        end = clock();

        printf("\nSORTED ARRAY IS: ");

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

           printf("%d\t", a[i]);

        printf("\n TIME TAKEN TO SORT %d NUMBERS IS %f SECS", n, (((double)(end - start)) /
CLOCKS_PER_SEC));

        break;

     case 2:

        n = 500;

        while (n <= 14500)

        {

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

           {

              a[i]=rand()%n;

              //a[i] = n - i;

           }

           start = clock();

           split(a, 0, n - 1);

           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:
```

```c
            exit(0);
        }
        getchar();
    }
}
void split(int a[], int low, int high)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        split(a, low, mid);
        split(a, mid + 1, high);
        combine(a, low, mid, high);
    }
}
void combine(int a[], int low, int mid, int high)
{
    int c[15000], i, j, k;
    i = k = low;
    j = mid + 1;
    while (i <= mid && j <= high)
    {
        if (a[i] < a[j])
        {
            c[k] = a[i];
            ++k;
            ++i;
        }
        else
```

```
    {
       c[k] = a[j];

       ++k;

       ++j;

    }

  }

  if (i > mid)

  {

     while (j <= high)

     {

        c[k] = a[j];

        ++k;

        ++j;

     }

  }

  if (j > high)

  {

     while (i <= mid)

     {

        c[k] = a[i];

        ++k;

        ++i;

     }

  }

  for (i = low; i <= high; i++)

  {

     a[i] = c[i];

  }

}
```

OUTPUT:-

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:1

ENTER NUMBER OF ARRAY ELEMENTS: 10

ENTER ARRAY ELEMENTS: 9 8 7 6 5 4 3 2 1 0

SORTED ARRAY IS: 0      1      2      3      4      5      6      7      8      9
 TIME TAKEN TO SORT 10 NUMBERS IS 0.000047 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


...Program finished with exit code 0
Press ENTER to exit console.
```
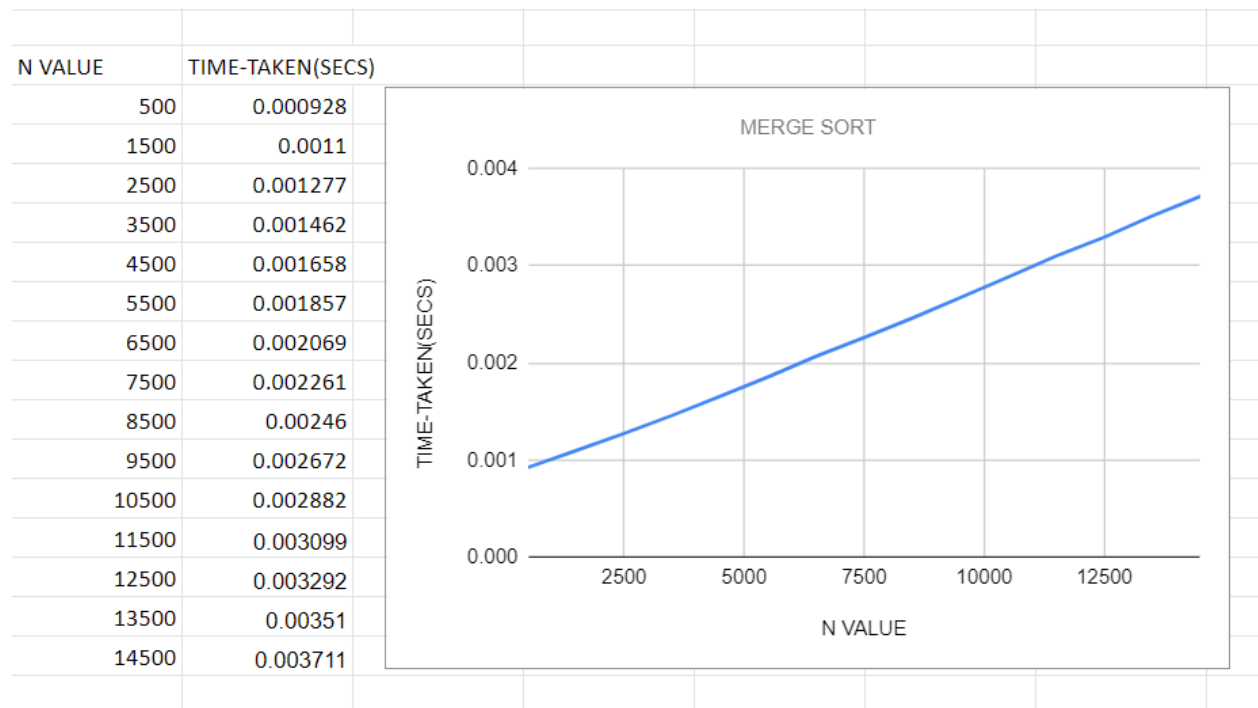
```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:2

 TIME TAKEN TO SORT 500 NUMBERS IS 0.000928 SECS
 TIME TAKEN TO SORT 1500 NUMBERS IS 0.001100 SECS
 TIME TAKEN TO SORT 2500 NUMBERS IS 0.001277 SECS
 TIME TAKEN TO SORT 3500 NUMBERS IS 0.001462 SECS
 TIME TAKEN TO SORT 4500 NUMBERS IS 0.001658 SECS
 TIME TAKEN TO SORT 5500 NUMBERS IS 0.001857 SECS
 TIME TAKEN TO SORT 6500 NUMBERS IS 0.002069 SECS
 TIME TAKEN TO SORT 7500 NUMBERS IS 0.002261 SECS
 TIME TAKEN TO SORT 8500 NUMBERS IS 0.002460 SECS
 TIME TAKEN TO SORT 9500 NUMBERS IS 0.002672 SECS
 TIME TAKEN TO SORT 10500 NUMBERS IS 0.002882 SECS
 TIME TAKEN TO SORT 11500 NUMBERS IS 0.003099 SECS
 TIME TAKEN TO SORT 12500 NUMBERS IS 0.003292 SECS
 TIME TAKEN TO SORT 13500 NUMBERS IS 0.003510 SECS
 TIME TAKEN TO SORT 14500 NUMBERS IS 0.003711 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


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

GRAPH: Generated graph for N value vs Time taken for Merge sort for different values of N

N VALUE	TIME-TAKEN(SECS)
500	0.000928
1500	0.0011
2500	0.001277
3500	0.001462
4500	0.001658
5500	0.001857
6500	0.002069
7500	0.002261
8500	0.00246
9500	0.002672
10500	0.002882
11500	0.003099
12500	0.003292
13500	0.00351
14500	0.003711

MERGE SORT

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

PROGRAM:

```c
#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <math.h>

void quicksort(int arr[25], int first, int last)

{


    int i, j, pivot, temp;

    if (first < last)

    {

        pivot = first;

        i = first;

        j = last;

        while (i < j)

        {

            while (arr[i] <= arr[pivot] && i < last)

                i++;

            while (arr[j] > arr[pivot])

                j--;

            if (i < j)

            {

                temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }
```

```c
        }
        temp = arr[pivot];
        arr[pivot] = arr[j];
        arr[j] = temp;
        for (int p = 0; p < 1000000; p++)
            ;
        quicksort(arr, first, j - 1);
        quicksort(arr, j + 1, last);
    }
}
int main()
{
    int i, n;
    clock_t start, end;
    printf("ENTER ARRAY SIZE =");
    scanf("%d", &n);
    int arr[150000];
    printf("ENTER ARRAY ELEMENTS");
    for (int j = 0; j < n; j++)
    {
        arr[j] = rand() % 10000;
    }
    printf("\n");
    start = clock();
    quicksort(arr, 0, n - 1);
    end = clock();

    printf("\nSORTED ELEMNETS = ");
    for (i = 0; i < n; i++)
        printf(" %d", arr[i]);
```
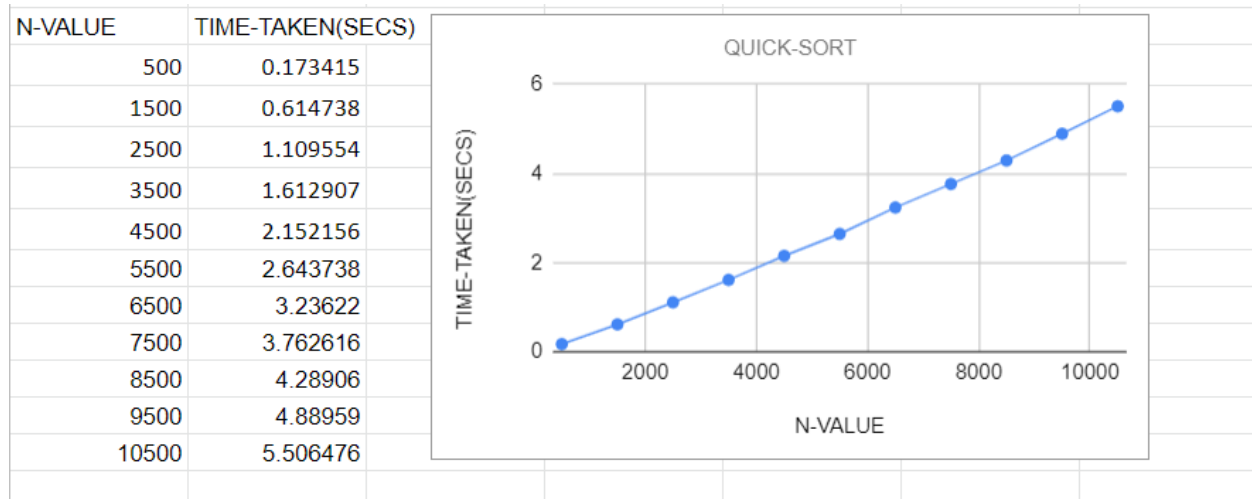
```
    printf("\n TIME TAKEN TO SORT %d NUMBERS IS %f SECS", n, (((double)(end - start)) /
CLOCKS_PER_SEC));

    return 0;

}
```

OUTPUT:-

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:1

ENTER NUMBER OF ARRAY 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 0.000657 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:2

 TIME TAKEN TO SORT 500 NUMBERS IS 0.173415 SECS
 TIME TAKEN TO SORT 1500 NUMBERS IS 0.614738 SECS
 TIME TAKEN TO SORT 2500 NUMBERS IS 1.109554 SECS
 TIME TAKEN TO SORT 3500 NUMBERS IS 1.612907 SECS
 TIME TAKEN TO SORT 4500 NUMBERS IS 2.152156 SECS
 TIME TAKEN TO SORT 5500 NUMBERS IS 2.643738 SECS
 TIME TAKEN TO SORT 6500 NUMBERS IS 3.236220 SECS
 TIME TAKEN TO SORT 7500 NUMBERS IS 3.762616 SECS
 TIME TAKEN TO SORT 8500 NUMBERS IS 4.289086 SECS
 TIME TAKEN TO SORT 9500 NUMBERS IS 4.889590 SECS
 TIME TAKEN TO SORT 10500 NUMBERS IS 5.506476 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


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

GRAPH: Generated graph for N value vs Time taken for Quick sort for different values of N

N-VALUE	TIME-TAKEN(SECS)
500	0.173415
1500	0.614738
2500	1.109554
3500	1.612907
4500	2.152156
5500	2.643738
6500	3.23622
7500	3.762616
8500	4.28906
9500	4.88959
10500	5.506476

QUICK-SORT — TIME-TAKEN(SECS) vs N-VALUE

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

PROGRAM:

```c
#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <math.h>

void swap(int *,int *);

void heapify(int [],int,int);

void heapSort(int[], int);

int main()

{

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

   clock_t start, end;

   while (1)

   {

      printf("\n1:FOR MANUAL ENTRY");

      printf("\n2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000");

      printf("\n3:EXIT");

      printf("\nENTER YOUR CHOICE:");

      scanf("%d", &ch);

      switch (ch)

      {

      case 1:

         printf("\nENTER NUMBER OF ARRAY ELEMENTS: ");

         scanf("%d", &n);

         printf("\nENTER ARRAY ELEMENTS: ");

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

         {
```

```c
            scanf("%d", &a[i]);
        }
        start = clock();
        heapSort(a, n);
        end = clock();
        printf("\nSORTED 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]=rand()%n;
                a[i] = n - i;
            }
            start = clock();
            heapSort(a, n);
            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;
```

```c
            }
            break;
        case 3:
            exit(0);
        }
        getchar();
    }
}


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

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

    for (int i = n - 1; i >= 0; i--)
    {
        swap(&arr[0], &arr[i]);

        heapify(arr, i, 0);
    }
}
```

OUTPUT:-

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:1

ENTER NUMBER OF ARRAY 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 0.000003 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:2

 TIME TAKEN TO SORT 500 NUMBERS IS 0.000904 SECS
 TIME TAKEN TO SORT 1500 NUMBERS IS 0.001096 SECS
 TIME TAKEN TO SORT 2500 NUMBERS IS 0.001301 SECS
 TIME TAKEN TO SORT 3500 NUMBERS IS 0.001544 SECS
 TIME TAKEN TO SORT 4500 NUMBERS IS 0.001772 SECS
 TIME TAKEN TO SORT 5500 NUMBERS IS 0.002010 SECS
 TIME TAKEN TO SORT 6500 NUMBERS IS 0.002236 SECS
 TIME TAKEN TO SORT 7500 NUMBERS IS 0.002510 SECS
 TIME TAKEN TO SORT 8500 NUMBERS IS 0.002771 SECS
 TIME TAKEN TO SORT 9500 NUMBERS IS 0.002984 SECS
 TIME TAKEN TO SORT 10500 NUMBERS IS 0.003273 SECS
 TIME TAKEN TO SORT 11500 NUMBERS IS 0.003504 SECS
 TIME TAKEN TO SORT 12500 NUMBERS IS 0.003755 SECS
 TIME TAKEN TO SORT 13500 NUMBERS IS 0.004015 SECS
 TIME TAKEN TO SORT 14500 NUMBERS IS 0.004275 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000
3:EXIT
ENTER YOUR CHOICE:3


...Program finished with exit code 0
Press ENTER to exit console.
```
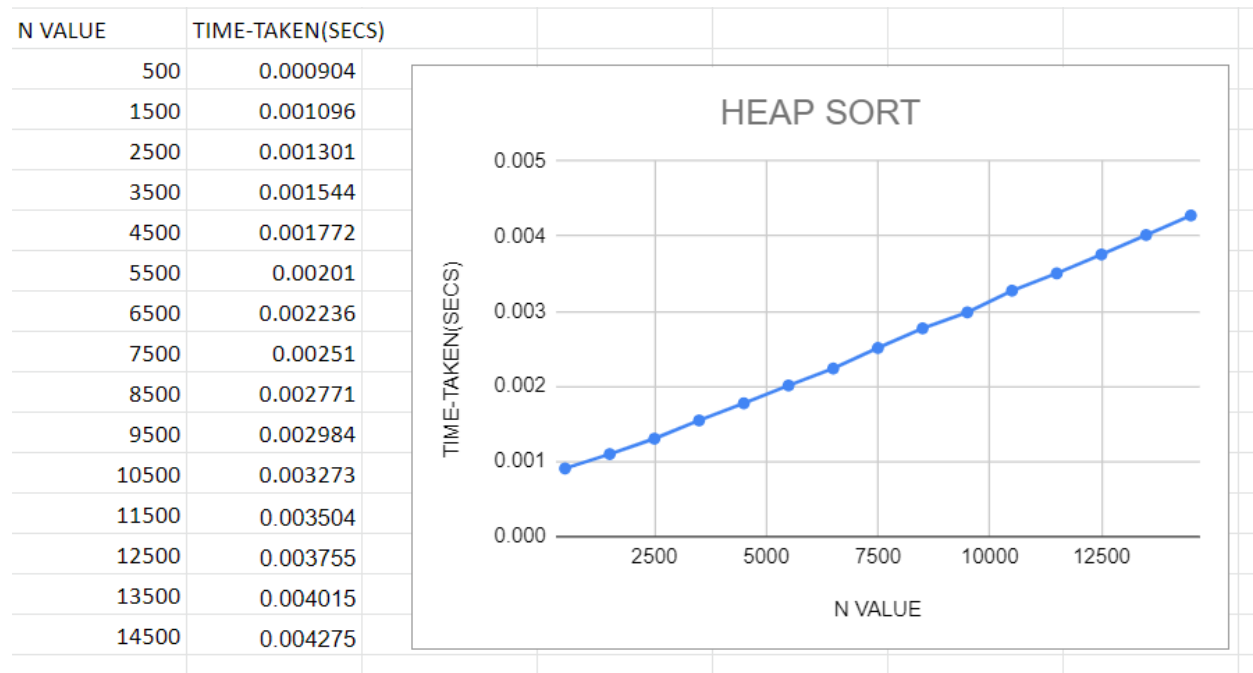
GRAPH: Generated graph for N value vs Time taken for Quick sort for different values of N

N VALUE	TIME-TAKEN(SECS)
500	0.000904
1500	0.001096
2500	0.001301
3500	0.001544
4500	0.001772
5500	0.00201
6500	0.002236
7500	0.00251
8500	0.002771
9500	0.002984
10500	0.003273
11500	0.003504
12500	0.003755
13500	0.004015
14500	0.004275

11) Implement Warshall's algorithm using dynamic programming.

PROGRAM:-

```c
#include<stdio.h>

int a[30][30];

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

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.
```

12) Implement 0/1 Knapsack problem using dynamic programming.

PROGRAM:-

```c
#include<stdio.h>

#include<conio.h>

void knapsack();

int max(int,int);

int i,j,n,m,p[10],w[10],v[10][10];

void 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();

 getch();

}


void knapsack()

{
```

```c
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]);
```

```c
printf("\nthe solution vector is:\n");
for(i=n;i>=1;i--)
{
if(v[i][m]!=v[i-1][m])
{
x[i]=1;
m=m-w[i];
}
else
{
x[i]=0;
}
}
for(i=1;i<=n;i++)
{
printf("%d\t",x[i]);
}
}
int max(int x,int y)
{
if(x>y)
{
return x;
}
else
{
return y;
}
}
```

```
enter the no. of items: 4

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

enter the profit of each item:
12 10 20 15

enter the knapsack's capacity:  5

the output 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


the optimal solution is 37
the solution vector is:
1         1         0         1

...Program finished with exit code 255
Press ENTER to exit console.
```

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

PROGRAM:-

```c
#include<stdio.h>

int n;

void display(int dist[][n]);

void floyd (int graph[][n])
{
        int dist[n][n], i, j, k;

        for (i = 0; i < n; i++)
                for (j = 0; j < n; j++)
                        dist[i][j] = graph[i][j];

        for (k = 0; k < n; k++)
        {
                for (i = 0; i < n; i++)
                {
                        for (j = 0; j < n; j++)
                        {
                                if (dist[i][k] + dist[k][j] < dist[i][j])
                                        dist[i][j] = dist[i][k] + dist[k][j];
                        }
                }
        }

        display(dist);
}

void display(int dist[][n])
```

```c
{
        printf ("DISTANCE MATRIX \n");

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

        {

                for (int j = 0; j < n; j++)

                {

                        if (dist[i][j] == 99)

                                printf("99 ");

                        else

                                printf ("%d ", dist[i][j]);

                }

                printf("\n");

        }

}


int main()

{

   printf("ENTER ORDER OF MATRIX \n");

   scanf("%d",&n);

   int graph[n][n];

   printf("ENTER ELEMENTS OF MATRIX and 99 FOR INFINITY\n");

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

   {

      for(int j = 0;j < n; j++)

      {

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

      }

   }

         floyd(graph);
```

```
    return 0;

}
```

OUTPUT:-

```
ENTER ORDER OF MATRIX
4
ENTER ELEMENTS OF MATRIX and 99 FOR INFINITY
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
DISTANCE MATRIX
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0


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

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

PROGRAM:-

```c
#include<stdio.h>
void prims();
int c[10][10],n;
void main()
{
 int i,j;
 printf("\nenter the no. of vertices: ");
 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();
}
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;
 }
```

```c
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:
99 3 99 99 6 5
3 99 1 99 99 4
99 1 99 6 99 4
99 6 6 99 8 5
6 99 99 8 99 2
5 4 4 5 2 99

1----->2=3

2----->3=1

2----->6=4

6----->5=2

6----->4=5

mincost=15

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

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

PROGRAM:-

```c
#include<stdio.h>
void kruskals();
int c[10][10],n;
void main()
{
 int i,j;
 printf("\nenter the no. of vertices: ");
 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;
 }
```

```c
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:
99 3 99 99 6 5
3 99 1 99 99 4
99 1 99 6 99 4
99 6 6 99 8 5
6 99 99 8 99 2
5 4 4 5 2 99

2----->3=1

5----->6=2

1----->2=3

2----->6=4

4----->6=5

mincost=15

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

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

PROGRAM:-

```
#include<stdio.h>

void dijkstras();

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

void main()

{

 int i,j;

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

 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: ");

 scanf("%d",&src);

 dijkstras();

}


void dijkstras()

{

 int vis[10],dist[10],u,j,count,min;

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

 {
```

```c
 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 no of vertices: 5

enter the cost matrix:
99 3 99 7 99
3 99 4 2 99
99 4 99 5 6
7 2 5 99 4
99 99 6 4 99

enter the source node: 1

the shortest distance is:

1----->1=0
1----->2=3
1----->3=7
1----->4=5
1----->5=9

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