

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

KEERTHANA N P(1BM20CS071)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

**B.M.S. COLLEGE OF
560019 May-2022 to July-2022**



ENGINEERING BENGALURU-

(Autonomous Institution under VTU)

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 **KEERTHANA N P (1BM20CS071)**, 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.
1	Write a recursive program to Solve a) Towers-of-Hanoi problem b) To find GCD	4-7
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.	8-14
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.	15-17
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.	18-21
5	Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.	22-24
6	Write program to obtain the Topological ordering of vertices in a given digraph.	25-26
7	Implement Johnson Trotter algorithm to generate permutations.	27-30
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.	31-34
9	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	35-37
10	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	38-40
11	Implement Warshall's algorithm using dynamic programming.	41-43
12	Implement 0/1 Knapsack problem using dynamic programming.	44-46
13	Implement All Pair Shortest paths problem using Floyd's algorithm.	47-49
14	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	50-52
15	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.	53-56

16	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	57-59
17	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.	60-62
18	Implement "N-Queens Problem" using Backtracking.	63-65

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.

#Program 1:

Write a recursive program to Solve

a. Towers-of-Hanoi problem

```
#include<stdio.h>
void tower(int n,char from,char to,char aux)
{
    if(n==1)
    {
        printf("Move disc 1 from %c to %c\n",from,to);
        return;
    }
    tower(n-1,from,aux,to);
    printf("Move disc %d from %c to %c\n",n,from,to);
    tower(n-1,aux,to,from);
}
int
main() {
    int n;
    printf("Enter the number of discs:\n");
    scanf("%d",&n);    tower(n,'A','C','B');
    return 0;
}
```

Output:

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL1.exe"

Enter the number of discs:

3

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 3 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

Move disc 1 from A to C

Process returned 0 (0x0) execution time : 4.134 s

Press any key to continue.

b. To find GCD

```
#include<stdio.h> int
gcd(int n1,int n2)
{
if(n2!=0)
{
return gcd(n2,n1%n2);
} else
return n1;
} int main()
{ int
n1,n2;
printf("Enter the value of n1 and n2:\n");
scanf("%d%d",&n1,&n2);
printf("The gcd of %d and %d is %d",n1,n2,gcd(n1,n2));
return 0;
}
```

Output:

 "C:\Users\Keerthana Natali P\OneDrive\Desktop\programs\ADAL2.exe"

Enter the value of n1 and n2:

36

48

The gcd of 36 and 48 is 12

Process returned 0 (0x0) execution time : 5.787 s

Press any key to continue.

#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.

• Binary Search

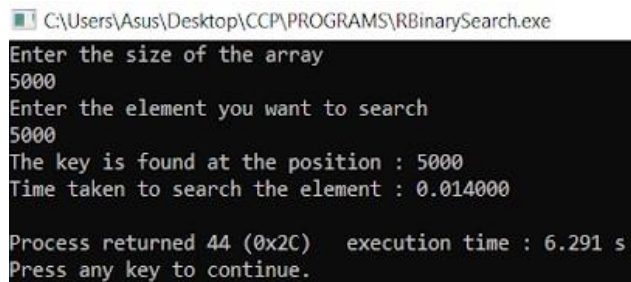
```
#include<stdio.h> #include<time.h>
int RBinarySearch(int arr[],int low,int high,int key)
{   int mid;
    delay();
    if(low>high)
        return -1;   mid =
    (low+high)/2;
    if(arr[mid] == key)
        return mid;   else if(key
    > arr[mid])
        return RBinarySearch(arr,mid+1,high,key);
    else
        return RBinarySearch(arr,low,mid-1,key);
}

void delay()
{   int temp;   for(int
i=0;i<80000;i++)
    temp = 387/2382;
}

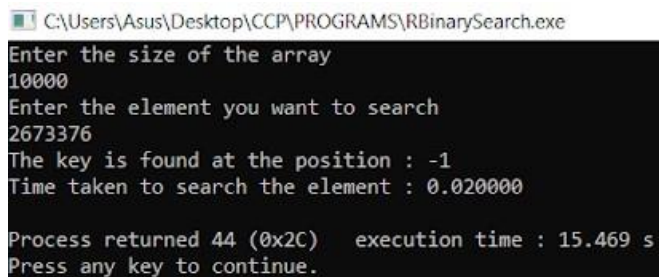
void main()
{   int n,key;
    clock_t start,end;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    int nums[n];   for(int
i=0;i<n;i++)
        nums[i]=i+1;
    printf("Enter the element you want to search\n");
    scanf("%d",&key);
    start = clock();
    int ans = RBinarySearch(nums,0,n-1,key);
    end = clock();
```

```
printf("The key is found at the position : %d\n",ans+1);  
printf("Time taken to search the element : %lf\n",(double)(end-start)/CLOCKS_PER_SEC); }
```

Output:



```
C:\Users\Asus\Desktop\CCP\PROGRAMS\RBinarySearch.exe  
Enter the size of the array  
5000  
Enter the element you want to search  
5000  
The key is found at the position : 5000  
Time taken to search the element : 0.014000  
  
Process returned 44 (0x2C) execution time : 6.291 s  
Press any key to continue.  
_
```



```
C:\Users\Asus\Desktop\CCP\PROGRAMS\RBinarySearch.exe  
Enter the size of the array  
10000  
Enter the element you want to search  
2673376  
The key is found at the position : -1  
Time taken to search the element : 0.020000  
  
Process returned 44 (0x2C) execution time : 15.469 s  
Press any key to continue.  
_
```

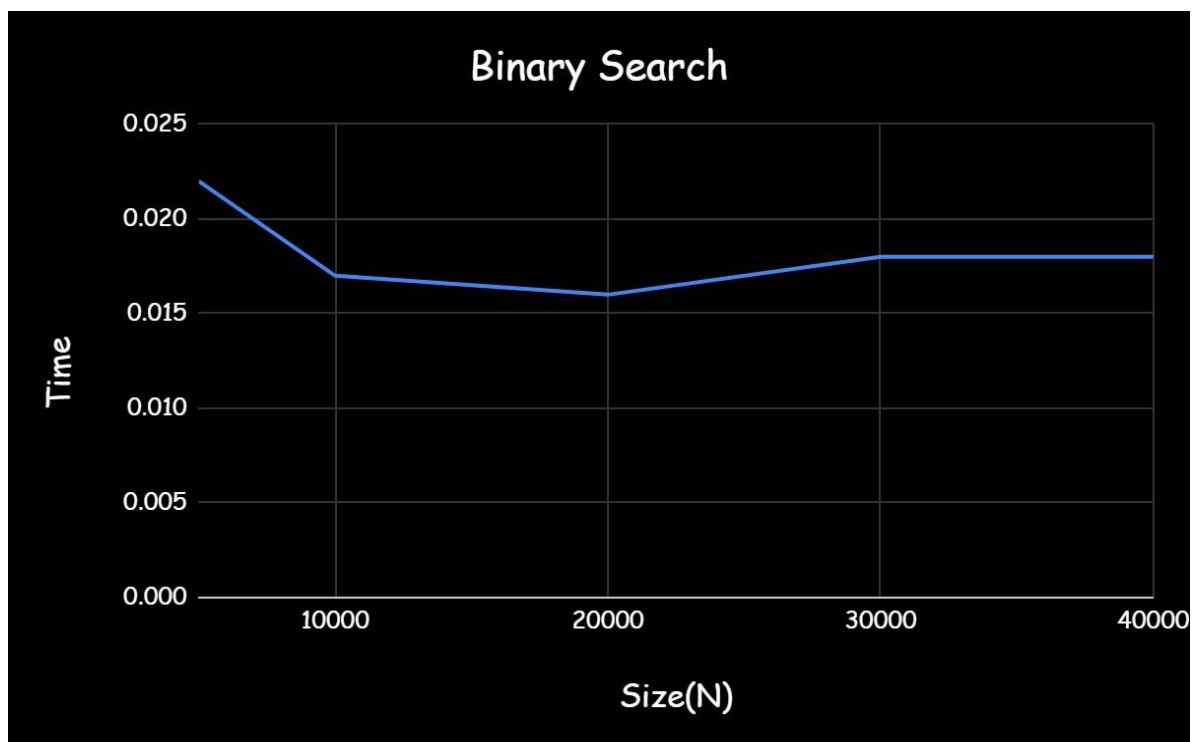


```
C:\Users\Asus\Desktop\CCP\PROGRAMS\RBinarySearch.exe
Enter the size of the array
20000
Enter the element you want to search
20000
The key is found at the position : 20000
Time taken to search the element : 0.019000

Process returned 44 (0x2C)   execution time : 10.076 s
Press any key to continue.
```

Graph:

Time Complexity: $O(\log n)$



• Linear Search

```
#include<stdio.h> #include<time.h>
int RLinearSearch(int arr[],int key,int index,int length)
{   delay();
    if(index<length)
    {
        if(arr[index] == key)
        {
            return index;
        }
    }
    else
        return RLinearSearch(arr,key,index+1,length);
}
if(index>=length)
{
    return -1;
}
}

void delay()
{   int temp;   for(int
i=0;i<8000;i++)
temp = 387/2382;
}

void main()
{   int n,key;
    clock_t start,end;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    int nums[n];   for(int
i=0;i<n;i++)
    nums[i]=i+1;
    printf("Enter the element you want to search\n");
    scanf("%d",&key);
    start = clock();
    int ans = RLinearSearch(nums,key,0,n);
    end = clock();
    printf("The key is found at the position : %d\n",ans+1);
    printf("Time taken to search the element : %lf\n",(double)(end-start)/CLOCKS_PER_SEC); }
```

Output:

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

```
Enter the size of the array
5000
Enter the element you want to search
5000
The key is found at the position : 5000
Time taken to search the element : 0.065000

Process returned 44 (0x2C)   execution time : 5.413 s
Press any key to continue.
```

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

```
Enter the size of the array
1000
Enter the element you want to search
1000
The key is found at the position : 1000
Time taken to search the element : 0.015000

Process returned 44 (0x2C)   execution time : 13.459 s
Press any key to continue.
```

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

```
Enter the size of the array
3000
Enter the element you want to search
3000
The key is found at the position : 3000
Time taken to search the element : 0.047000

Process returned 44 (0x2C)   execution time : 10.042 s
Press any key to continue.
```

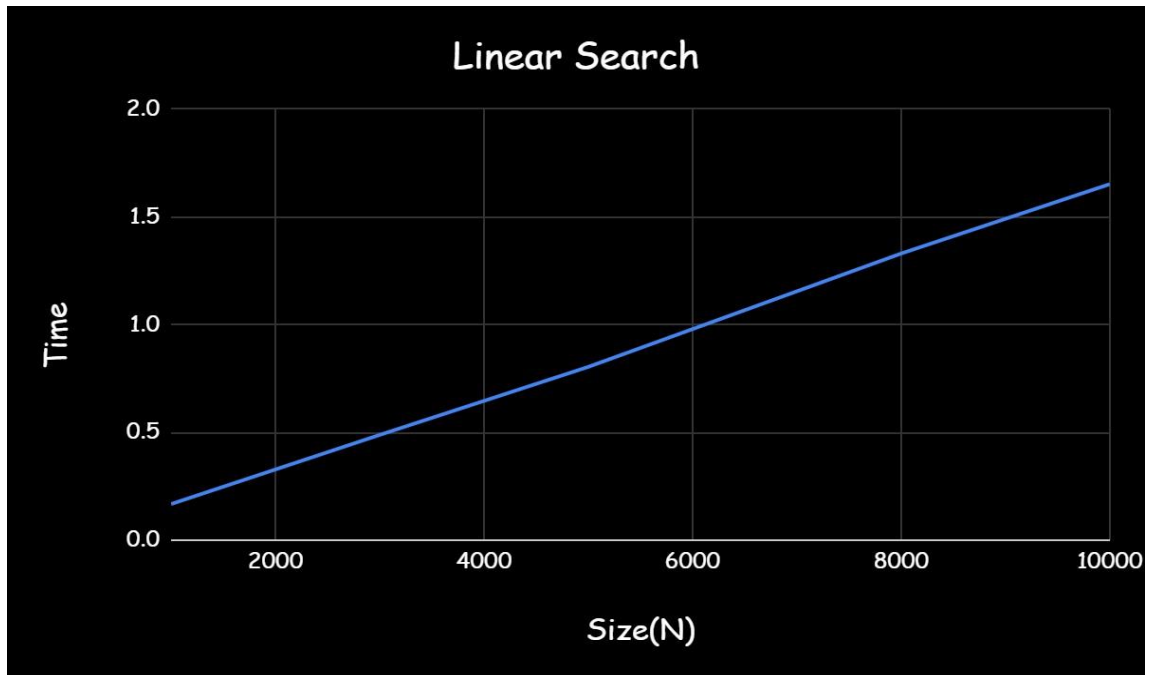
"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

```
Enter the size of the array
10000
Enter the element you want to search
10000
The key is found at the position : 10000
Time taken to search the element : 0.137000

Process returned 44 (0x2C)   execution time : 6.081 s
Press any key to continue.
```

Graph:

Time Complexity: $O(n)$



#Program 3:

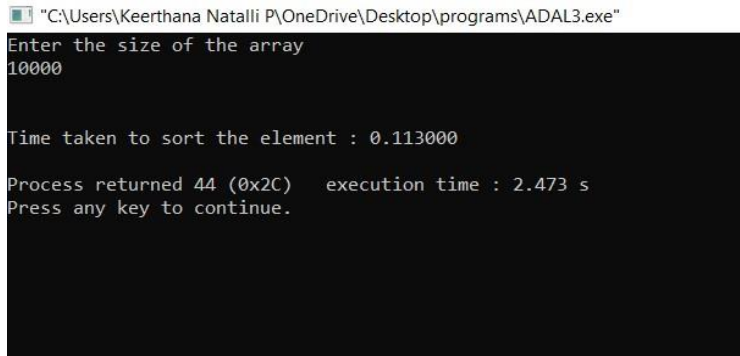
Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include<stdio.h>
#include<time.h> void
selSort(int a[],int n)
{
    int s,temp,k,m;
    for(int i=0;i < n-1 ;i++)
    {
        s =
a[i];
        for(k = i+1 ;k < n;k++)
        {
            if(a[k] < s)
            {
s = a[k];
m = k;
            }
        }
        temp = s;
a[m] = a[i];
a[i] = temp;
    }
}

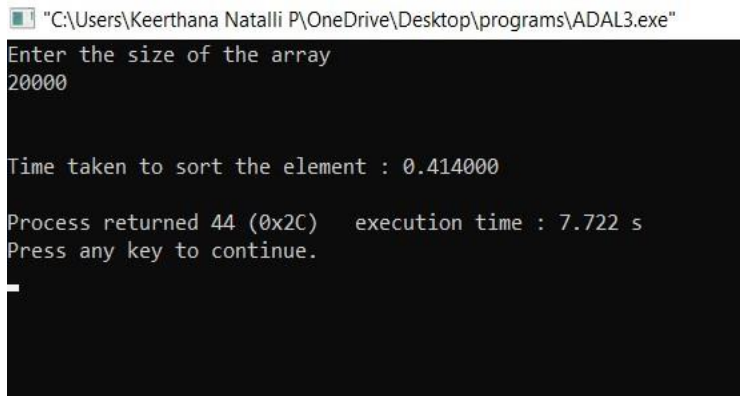
void main()
{
    int n,key;
    clock_t start,end;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    int nums[n];   for(int
i=0;i<n;i++)
    nums[i]=rand();
    printf("Array before sorting\n");
    for(int i=0;i<n;i++)
    printf("%d\t",nums[i]);   start =
clock();   selSort(nums,n);   end
= clock();
```

```
    printf("\n\nArray after sorting\n");  
for(int i=0;i<n;i++)  
    printf("%d\t",nums[i]);  
    printf("\n\nTime taken to sort the element : %f\n", (double)(end-start)/CLOCKS_PER_SEC);  
}
```

Output:



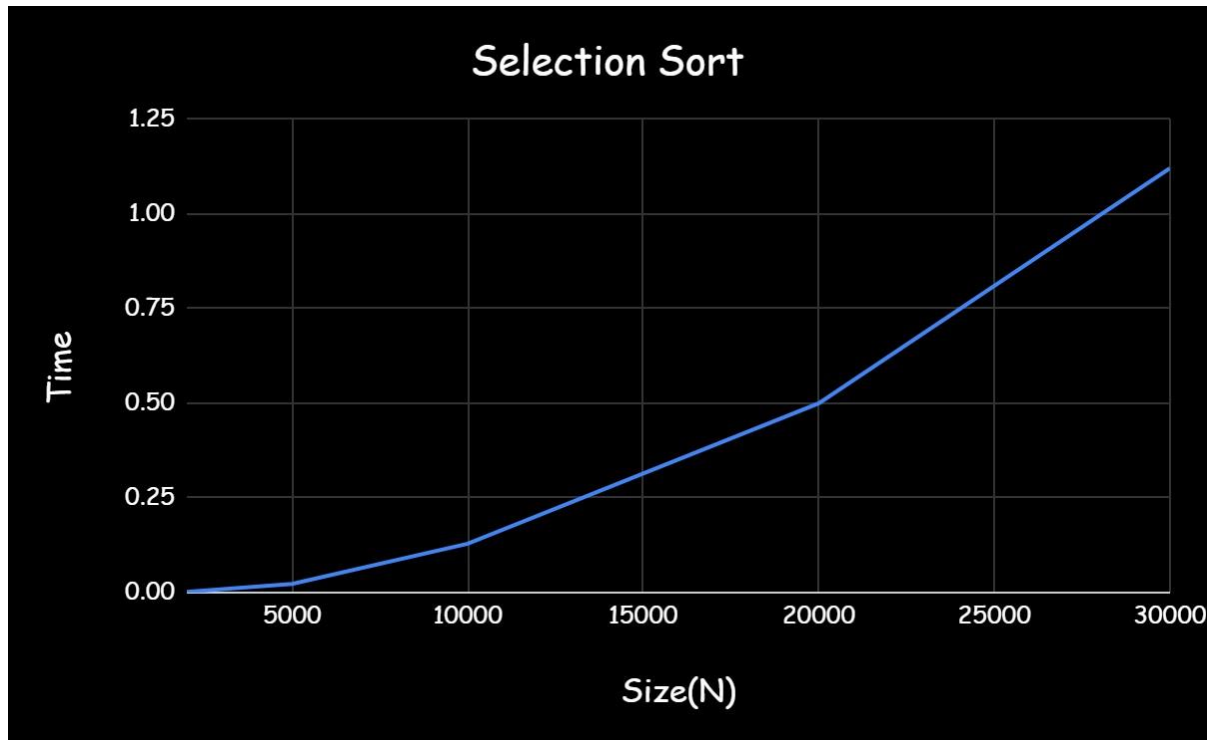
```
"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"  
Enter the size of the array  
10000  
  
Time taken to sort the element : 0.113000  
  
Process returned 44 (0x2C)   execution time : 2.473 s  
Press any key to continue.
```



```
"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"  
Enter the size of the array  
20000  
  
Time taken to sort the element : 0.414000  
  
Process returned 44 (0x2C)   execution time : 7.722 s  
Press any key to continue.  
_
```

Graph:

Time Complexity: $O(n^2)$



#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.**

```
#include<stdio.h> #include<conio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;
void bfs(int v) { for(i = 1; i <= n; i++) if(a[v][i]
&& !visited[i])
    q[++r] = i;

    if(f <= r) {
        visited[q[f]] = 1;
        bfs(q[f++]);
    }
}

void main() {
    int
    v;
    printf("Enter the number of vertices: ");
    scanf("%d",&n);
    for(i=1; i <= n; i++) {
        q[i] = 0;
        visited[i] = 0;
    }
    printf("\nEnter graph data in matrix form:\n");
    for(i=1; i<=n; i++) { for(j=1;j<=n;j++) {
        scanf("%d", &a[i][j]);
    }
    }
    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    bfs(v);
    printf("\nThe node which are reachable are:");
    for(i=1; i <= n; i++) {
        if(visited[i])
            printf(" %d", i);
        else {
            printf("\nBFS is not possible. All nodes are not reachable!");
            break;
        }
    }
}
```

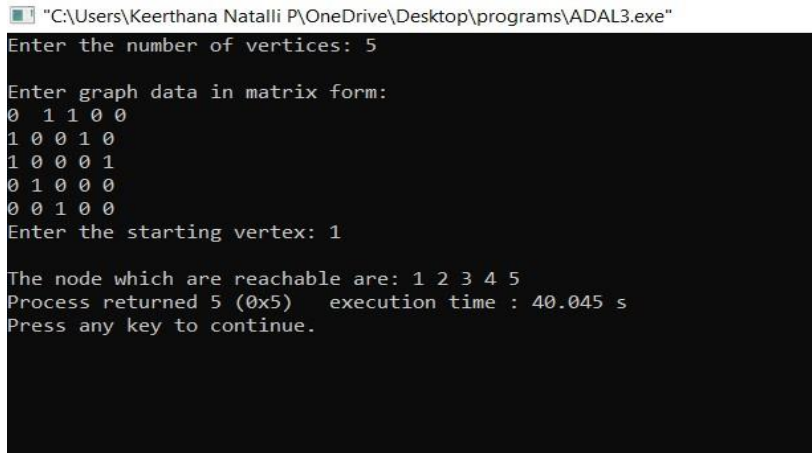


```

    }
}
}

```

Output:



```

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"
Enter the number of vertices: 5
Enter graph data in matrix form:
0 1 1 0 0
1 0 0 1 0
1 0 0 0 1
0 1 0 0 0
0 0 1 0 0
Enter the starting vertex: 1
The node which are reachable are: 1 2 3 4 5
Process returned 5 (0x5)   execution time : 40.045 s
Press any key to continue.

```

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

```

#include <stdio.h>
#include<time.h> int
a[20][20],reach[20],n;
void dfs(int v) {
    int i;
    reach[v]=1;    for
    (i=1;i<=n;i++)
        if(a[v][i] && !reach[i]) {
            printf("\n %d->%d",v,i);
            dfs(i);
        }
} void main()
{
    clock_t start,end;
    int i,j,count=0;
    printf("\n Enter number of
vertices:");    scanf("%d",&n);    for
    (i=1;i<=n;i++) {
        reach[i]=0;
    }
}

```

```

        for (j=1;j<=n;j++)
            a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i=1;i<=n;i++)
    for (j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");    for
(i=1;i<=n;i++) {
        if(reach[i])
            count++;
    }
    if(count==n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
}

```

Output:

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

```

Enter number of vertices:5

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

1->2
2->4
1->3
3->5

Graph is connected
Process returned 20 (0x14)   execution time : 30.167 s
Press any key to continue.

```

#Program 5:

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

```
#include<stdio.h>
#include<time.h>
void insertionSort(int a[],int n)
{   int
j,key;
    for(int i=1;i<=n-1;i++)
    {
        key = a[i];    for(j=i-
1;j>=0;j--)
        {
            if(a[j] > key)
            {
                a[j+1] = a[j];
            }
        }
        else
        break;
        a[j+1] = key;
    }
}

void main()
{   int n,key;
clock_t start,end;
    printf("Enter the size of the array\n");
    scanf("%d",&n);   int
nums[n];   for(int
i=0;i<n;i++)
    nums[i]=rand();   start
= clock();
insertionSort(nums,n);
end = clock();
    printf("\nTime taken to sort the element : %f\n",(double)(end-start)/CLOCKS_PER_SEC); }
```

Output:

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

Enter the size of the array

5000

Time taken to sort the element : 0.016000

Process returned 43 (0x2B) execution time : 77.740 s

Press any key to continue.

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

Enter the size of the array

20000

Time taken to sort the element : 0.204000

Process returned 43 (0x2B) execution time : 5.721 s

Press any key to continue.

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

Enter the size of the array

30000

Time taken to sort the element : 0.465000

Process returned 43 (0x2B) execution time : 6.446 s

Press any key to continue.

Graph:

Time Complexity: $O(n^2)$



#Program 6:

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

```
#include<stdio.h> int
main() {
    int i,j,k,n,a[10][10], indeg[10],flag[10],count=0;
    printf("Enter the no of vertices:\n"); scanf("%d",&n);
    printf("Enter the adjacency matrix: \n");
    for(i=0;i<n;i++){ printf("Enter
row%d\n", i+1); for(j=0;j<n;j++)
scanf("%d", &a[i][j]);
    }
    for(i=0;i<n;i++){
indeg[i]=0; flag[i]=0;
    }
    for(i=0;i<n;i++){
for(j=0;j<n;j++){
indeg[i]=indeg[i]+a[j][i];
    }
    }
    printf("\nThe Topological order is: ");
    while(count<n) {
for(k=0;k<n;k++){
    if((indeg[k]==0) && (flag[k]==0)){
        printf("%d\t",(k+1));
        flag[k]=1;
    }
for(i=0;i<n;i++) {
    if(a[i][k]==1) indeg[k]--;
    }
    }
    count++;
    } return
0;
}
```

Output:

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\ADAL3.exe"

```
Enter the no of vertices:
5
Enter the adjacency matrix:
Enter row1
0 0 1 0 0
Enter row2
0 0 1 0 0
Enter row3
1 1 0 1 1
Enter row4
0 0 1 0 1
Enter row5
0 0 1 1 0

The Topological order is: 1    2    3    4    5
Process returned 0 (0x0)   execution time : 39.469 s
Press any key to continue.
```

#Program 7:

Implement Johnson Trotter algorithm to generate permutations

```
#include <stdio.h>
#include<stdlib.h> int
LEFT_TO_RIGHT =1; int
RIGHT_TO_LEFT =0;
int searchArr(int a[], int n, int mobile)
{
    for (int i = 0; i < n; i++)
        if (a[i] == mobile)
            return i + 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 swap(int *m,int *n)
{   int temp;
temp = *m;
*m = *n;
    *n = temp;
}

```

```

int 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-1], &a[pos-2]);
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 fact(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");

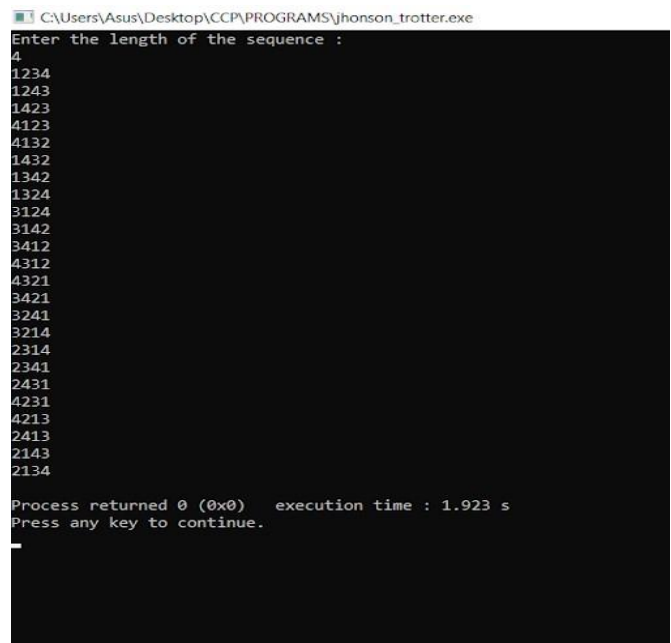
        for (int i = 0; i < n; i++)
            dir[i] = RIGHT_TO_LEFT;

        for (int i = 1; i < fact(n); i++)
            printOnePerm(a, dir, n);
    }

int main()
{
    int n;
    printf("Enter the length of the sequence : \n");
    scanf("%d",&n);
printPermutation(n);
    return 0;
}

```

Output:



```

C:\Users\Asus\Desktop\CCP\PROGRAMS\jhonson_trotter.exe
Enter the length of the sequence :
4
1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134

Process returned 0 (0x0)   execution time : 1.923 s
Press any key to continue.

```

#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>
void merge(int a[],int s,int m,int e)
{   delay();   int
merged[e-s+1];   int
i=s,j=m+1,k=0;
    while(i<=m && j<=e)
    {
        if(a[i]<a[j])
        {
            merged[k]=a[i];
            i++;
k++;        }
    else
    {
        merged[k]=a[j];
        j++;
k++;
    }
    }
    while(i<=m)
    {
        merged[k]=a[i];
        i++;
k++;
    }
    while(j<=e)
    {
        merged[k]=a[j];
        j++;
k++;
    }
    for(int p=0;p<e-s+1;p++)
a[s+p]=merged[p];
} void
delay() {
```

```

int temp ;
for(int i=0
;i<50000;i+
+)
temp=46/47
48;
}

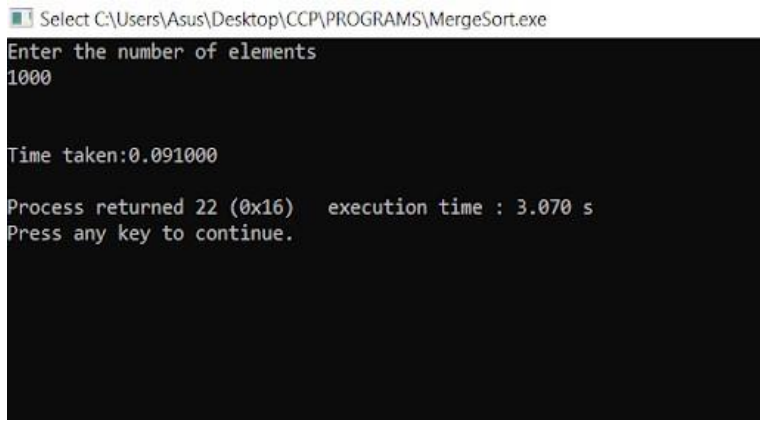
void mergeSort(int a[],int s,int e)
{   if(e-s+1 ==
1)
    return;

    int mid=(s+e)/2;
    mergeSort(a,s,mid);
    mergeSort(a,mid+1,e);
    merge(a,s,mid,e);
}

void main() {
    int n;   clock_t
start,end;
    printf("Enter the number of elements\n");
    scanf("%d",&n);   int nums[n];   for(int
i=0;i<n;i++)       nums[i]=rand();
    start=clock();   mergeSort(nums,0,n-1);
    end=clock();
    printf("\n\nTime taken:%lf\n",(double)(end-start)/CLOCKS_PER_SEC); }

```

Output:



```

Select C:\Users\Asus\Desktop\CCP\PROGRAMS\MergeSort.exe
Enter the number of elements
1000

Time taken:0.091000

Process returned 22 (0x16)  execution time : 3.070 s
Press any key to continue.

```

```
C:\Users\Asus\Desktop\CCP\PROGRAMS\MergeSort.exe
Enter the number of elements
5000

Time taken:0.528000

Process returned 22 (0x16)   execution time : 7.991 s
Press any key to continue.
```

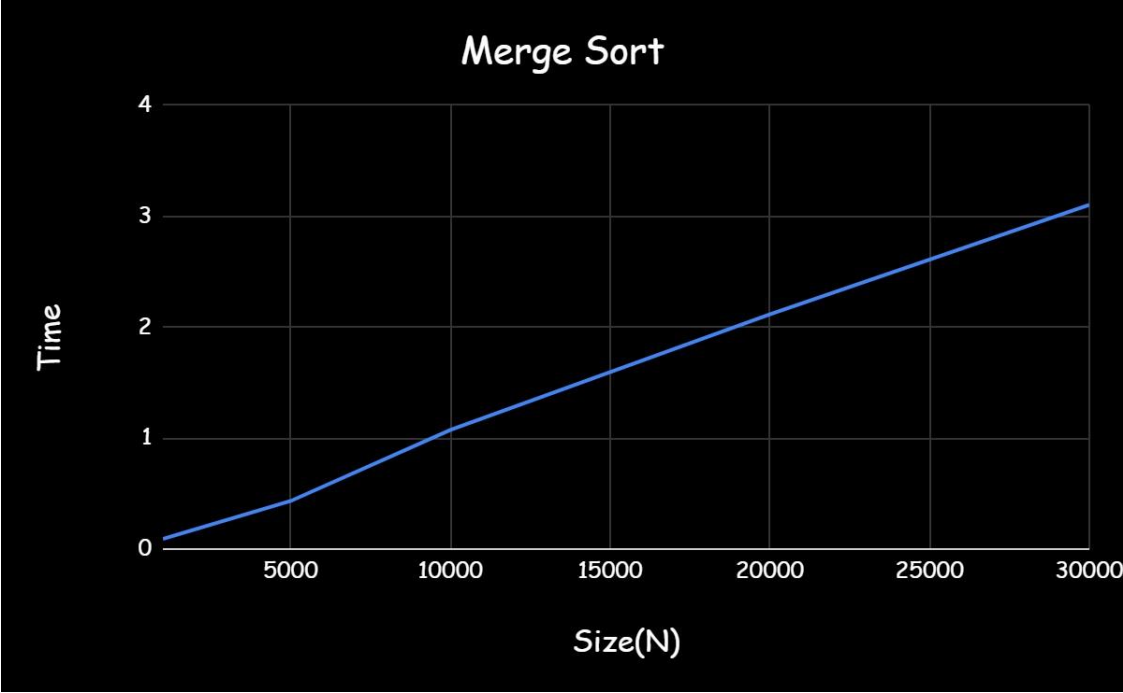
```
C:\Users\Asus\Desktop\CCP\PROGRAMS\MergeSort.exe
Enter the number of elements
10000

Time taken:1.087000

Process returned 22 (0x16)   execution time : 2.595 s
Press any key to continue.
```

Graph:

Time Complexity : $O(n \log n)$



#Program 9:

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

```
#include<stdio.h>
#include<time.h>
void quicksort(int number[],int first,int last){
    int i, j, pivot, temp;
    delay();
    if(first<last){
        pivot=first;
        i=first;    j=last;
        while(i<j){
            while(number[i]<=number[pivot]&& i<last)
                i++;
            while(number[j]>number[pivot])
                j--;
            if(i<j){
                temp=number[i];
                number[i]=number[j];
                number[j]=temp;
            }
        }
        temp=number[pivot];
        number[pivot]=number[j];    number[j]=temp;
        quicksort(number,first,j-1);
        quicksort(number,j+1,last);
    } }
void delay()
{    int temp ;    for(int i=0
;i<50000;i++)
temp=46/4748;
}
void main()
{    int n;
    clock_t start,end;
    printf("Enter the number of elements\n");
    scanf("%d",&n);    int nums[n];
    for(int i=0;i<n;i++)
        nums[i]=rand();    start=clock();
    quicksort(nums,0,n-1);
    end=clock();
```

```
printf("\n\nTime taken to sort:%lf\n", (double)(end-start)/CLOCKS_PER_SEC); }
```

Output:

C:\Users\Asus\Desktop\CCP\PROGRAMS\QuickSort.exe

```
Enter the number of elements
5000
```

```
Time taken to sort:0.733000
```

```
Process returned 30 (0x1E)   execution time : 3.707 s
Press any key to continue.
```

C:\Users\Asus\Desktop\CCP\PROGRAMS\QuickSort.exe

```
Enter the number of elements
10000
```

```
Time taken to sort:1.478000
```

```
Process returned 30 (0x1E)   execution time : 4.064 s
Press any key to continue.
```



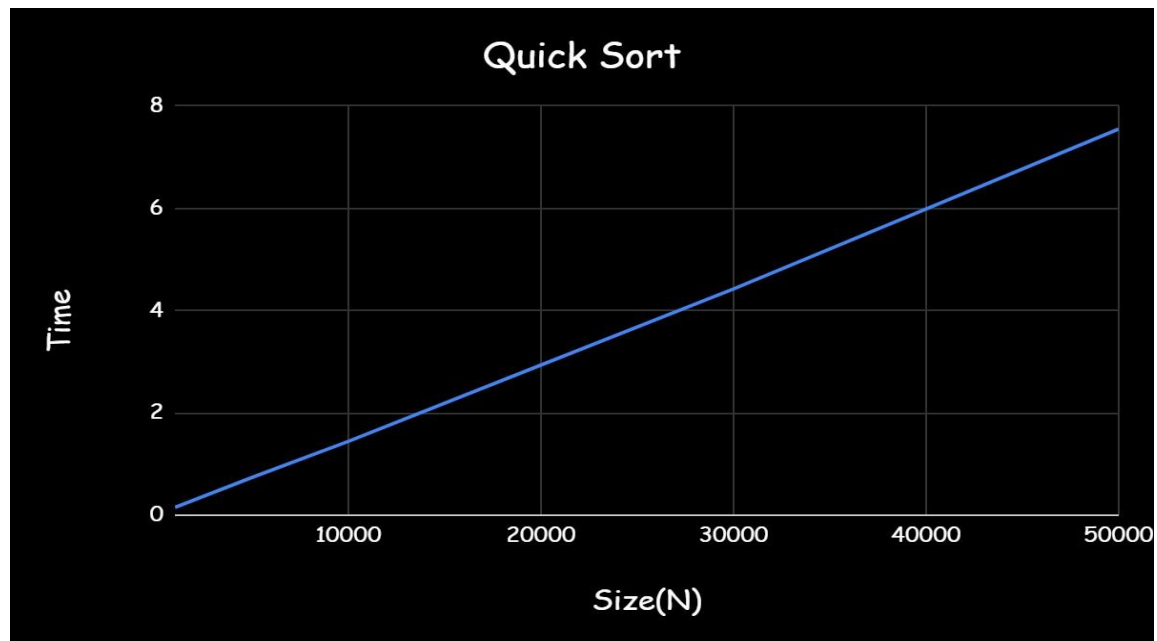
```
C:\Users\Asus\Desktop\CCP\PROGRAMS\QuickSort.exe
Enter the number of elements
20000

Time taken to sort:2.929000

Process returned 30 (0x1E)   execution time : 8.815 s
Press any key to continue.
```

GRAPH :

Time Complexity : $O(n \log n)$



#Program 10:

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

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

void heapify(int a[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left child
    int right = 2 * i + 2; // right child
    // If left child is larger than root
    if (left < n && a[left] > a[largest])
        largest = left;
    // If right child is larger than root
    if (right < n && a[right] > a[largest])
        largest = right;
    // If root is not largest
    if (largest != i) {
        // swap a[i] with a[largest]
        int temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;

        heapify(a, n, largest);
    }
}

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

    for (i = n - 1; i >= 0; i--) {

        temp = a[0];
        a[0] = a[i];
        a[i] = temp;

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

```

    }
}

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

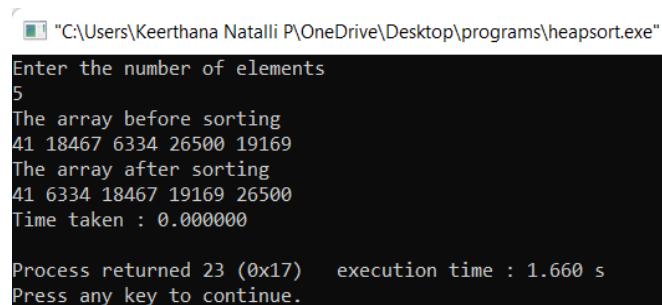
}

int main()
{
    int i,n;
    int a[100];
    clrscr();
    printf("enter the no.of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        a[i]=rand()%10;
    }

    printf("Before sorting array elements are - \n");
    printArr(a, n);
    heapSort(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    getch();
    return 0;
}

```

Output:



```

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"
Enter the number of elements
5
The array before sorting
41 18467 6334 26500 19169
The array after sorting
41 6334 18467 19169 26500
Time taken : 0.000000

Process returned 23 (0x17)   execution time : 1.660 s
Press any key to continue.

```

#Program 11:

Implement Warshall's algorithm using dynamic programming.

```
#include<stdio.h>
#define nodes 4
void floyds(int p[nodes+1][nodes+1])
{
    for(int k=1;k<=nodes;k++)
    {
        for(int i=1;i<=nodes;i++)
        {
            for(int j=0;j<=nodes;j++)
            {
                p[i][j]=(p[i][j] || (p[i][k]&& p[k][j]));
            }
        }
    }
}

int max(int a,int b)
{
    if(a > b)
        return a;
    else
        return b;
}

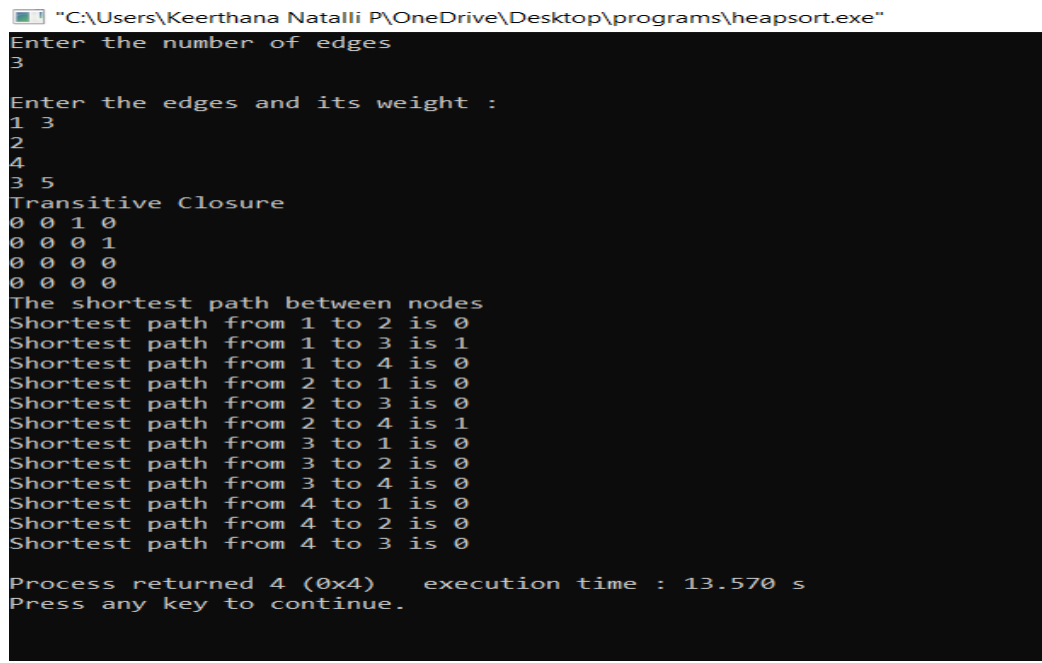
void main()
{
    int e,u,v,w,p[nodes+1][nodes+1];
    printf("\nEnter the number of edges\n");
    scanf("%d",&e);
    for(int i=1;i<=nodes;i++)
        for(int j=1;j<=nodes;j++)
            p[i][j] = 0;
    printf("\nEnter the edges and its weight :\n");
    for(int i=1;i<=e;i++)
    {
        scanf("%d%d",&u,&v);
        p[u][v]=1;
    }
}
```

```

floyds(p);
printf("Transitive Closure\n");
for(int i=1;i<=nodes;i++)
{
    for(int j=1;j<=nodes;j++)
        printf("%d ",p[i][j]);
    printf("\n");
}
printf("The shortest path between nodes\n");
for(int i=1;i<=nodes;i++)
{
    for(int j=1;j<=nodes;j++)
    {
        if(i!=j)
            printf("Shortest path from %d to %d is %d\n",i,j,p[i][j]);
    }
}
}

```

Output:



```

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"
Enter the number of edges
3
Enter the edges and its weight :
1 3
2
4
3 5
Transitive Closure
0 0 1 0
0 0 0 1
0 0 0 0
0 0 0 0
The shortest path between nodes
Shortest path from 1 to 2 is 0
Shortest path from 1 to 3 is 1
Shortest path from 1 to 4 is 0
Shortest path from 2 to 1 is 0
Shortest path from 2 to 3 is 0
Shortest path from 2 to 4 is 1
Shortest path from 3 to 1 is 0
Shortest path from 3 to 2 is 0
Shortest path from 3 to 4 is 0
Shortest path from 4 to 1 is 0
Shortest path from 4 to 2 is 0
Shortest path from 4 to 3 is 0

Process returned 4 (0x4)    execution time : 13.570 s
Press any key to continue.

```

#Program 12:

Implement 0/1 Knapsack problem using dynamic programming.

```
#include<stdio.h>
void main()
{
    int n,cap;
    printf("Enter the number of items\n");
    scanf("%d",&n);
    int w[n+1],p[n+1],included[n+1],count=0;
    printf("Enter the weight and profit obtained for each item\n");
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d",&w[i],&p[i]);
    }
    printf("Enter the capacity of the knapsack\n");
    scanf("%d",&cap);
    int mat[n+1][cap+1];
    for(int i=0;i<=n;i++)
        mat[i][0]=0;
    for(int i=0;i<=cap;i++)
        mat[0][i]=0;

    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=cap;j++)
        {
            int prowithoutCurr = mat[i-1][j];
            int prowithCurr = 0;


            int wtOfCurr = w[i];
            if(wtOfCurr <= j)
            {
                prowithCurr = p[i];
                int rem = j - wtOfCurr;
                prowithCurr += mat[i-1][rem];
            }
            mat[i][j] = (prowithCurr > prowithoutCurr) ? prowithCurr : prowithoutCurr;
        }
    }
    int i=n,j=cap;
```

```

while(i!=0 && j!=0)
{
    if(mat[i][j] != mat[i-1][j])
    {
        included[i]=1;
        j = j-w[i];
        i--;
        count++;
    }
    else i--;
}
printf("Number of items included : %d\n",count);
printf("Sl.No\tWeight\tProfit\n");
for(int i=1;i<=n;i++)
{
    if(included[i])
    {
        printf("%d\t%d\t%d\n",i,w[i],p[i]);
    }
}
printf("Maximum profit obtained is %d\n",mat[n][cap]);
}

```

Output:

 "C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"

```

Enter the number of items
3
Enter the weight and profit obtained for each item
12 5
14 6
13 7
Enter the capacity of the knapsack
20
Number of items included : 1
Sl.No  Weight  Profit
2      14      6
3      13      7
Maximum profit obtained is 7

Process returned 29 (0x1D)   execution time : 14.204 s
Press any key to continue.

```

#Program 13:

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

```
#include<stdio.h>
#define nodes 4
void floyds(int p[nodes+1][nodes+1])
{
    for(int k=1;k<=nodes;k++)
    {
        for(int i=1;i<=nodes;i++)
        {
            for(int j=0;j<=nodes;j++)
            {
                p[i][j]=max(p[i][j] , p[i][k]&& p[k][j]);
            }
        }
    }
}

int max(int a,int b)
{
    if(a > b)
        return a;
    else
        return b;
}

void main()
{
    int e,u,v,w,p[nodes+1][nodes+1];
    printf("\nEnter the number of edges\n");
    scanf("%d",&e);
    for(int i=1;i<=nodes;i++)
        for(int j=1;j<=nodes;j++)
            p[i][j] = 0;
    printf("\nEnter the edges and its weight :\n");
    for(int i=1;i<=e;i++)
    {
        scanf("%d%d",&u,&v);
        p[u][v]=1;
    }
}
```

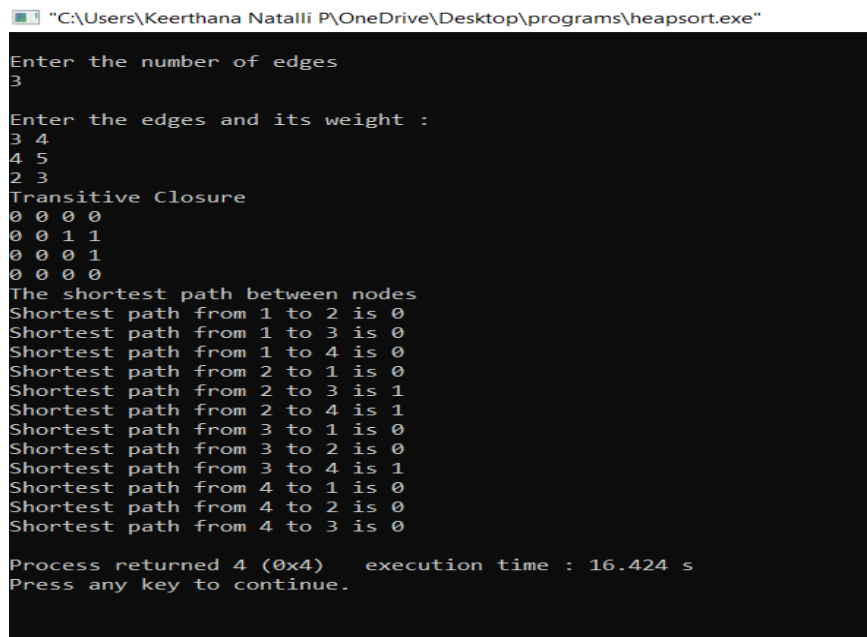


```

floyds(p);
printf("Transitive Closure\n");
for(int i=1;i<=nodes;i++)
{
    for(int j=1;j<=nodes;j++)
        printf("%d ",p[i][j]);
    printf("\n");
}
printf("The shortest path between nodes\n");
for(int i=1;i<=nodes;i++)
{
    for(int j=1;j<=nodes;j++)
    {
        if(i!=j)
            printf("Shortest path from %d to %d is %d\n",i,j,p[i][j]);
    }
}
}

```

Output:



```

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"
Enter the number of edges
3
Enter the edges and its weight :
3 4
4 5
2 3
Transitive Closure
0 0 0 0
0 0 1 1
0 0 0 1
0 0 0 0
The shortest path between nodes
Shortest path from 1 to 2 is 0
Shortest path from 1 to 3 is 0
Shortest path from 1 to 4 is 0
Shortest path from 2 to 1 is 0
Shortest path from 2 to 3 is 1
Shortest path from 2 to 4 is 1
Shortest path from 3 to 1 is 0
Shortest path from 3 to 2 is 0
Shortest path from 3 to 4 is 1
Shortest path from 4 to 1 is 0
Shortest path from 4 to 2 is 0
Shortest path from 4 to 3 is 0
Process returned 4 (0x4)   execution time : 16.424 s
Press any key to continue.

```


#Program 14:

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

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
printf("\nEnter the number of nodes:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
visited[1]=1;
printf("\n");
while(ne < n)
{
for(i=1,min=999;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]< min)
if(visited[i]!=0)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
if(visited[u]==0 || visited[v]==0)
{
printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
mincost+=min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
```

```
}  
printf("\n Minimun cost=%d",mincost);  
}
```

Output:

 "C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"

```
Enter the number of nodes:3  
  
Enter the adjacency matrix:  
0 1 1  
1 0 1  
0 1 1  
  
Edge 1:(1 2) cost:1  
Edge 2:(1 3) cost:1  
Minimun cost=2  
Process returned 16 (0x10)   execution time : 25.969 s  
Press any key to continue.
```

#Program 15:

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


```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
clrscr();
printf("\n\tImplementation of Kruskal's algorithm\n");
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j <= n;j++)
{
if(cost[i][j] < min)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
```

```

    }
    }
    u=find(u);
    v=find(v);
    if(uni(u,v))
    {
        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
        mincost +=min;
    }
    cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
getch();
}
int find(int i)
{
    while(parent[i])
    i=parent[i];
    return i;
}
int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
        return 1;
    }
    return 0;
}

```

Output:

 "C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"

```

Implementation of Kruskal's algorithm
Enter the no. of vertices:3
Enter the cost adjacency matrix:
0 1 1
1 0 0
1 0 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) =1
2 edge (1,3) =1
Minimum cost = 2

```

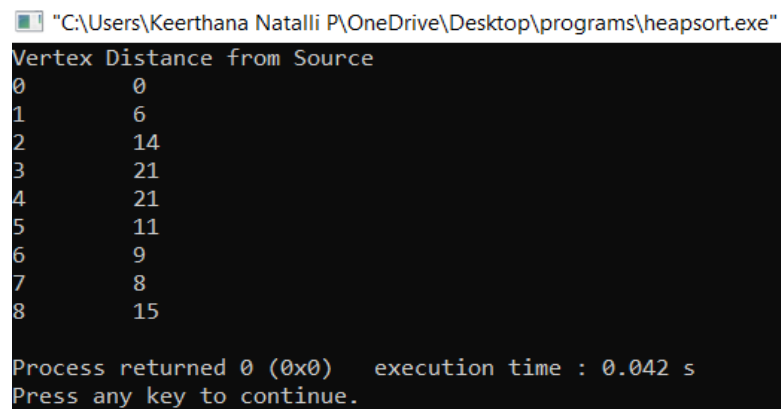
#Program 16:

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

```
#include <stdio.h>
#define V 9
int minDistance(int dist[], int sptSet[]) {
    int min = 999, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == 0 && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}
int printSolution(int dist[], int n) {
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src) {
    int dist[V];
    int sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = 999, sptSet[i] = 0;
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = 1;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != 999 && dist[u] + graph[u][v] < dist[v]) dist[v] =
dist[u] + graph[u][v];
    }
    printSolution(dist, V);
}
int main() {
    int graph[V][V] = { { 0, 6, 0, 0, 0, 0, 0, 8, 0 },
        { 6, 0, 8, 0, 0, 0, 0, 13, 0 },
        { 0, 8, 0, 7, 0, 6, 0, 0, 2 },
        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
        { 0, 0, 6, 14, 10, 0, 2, 0, 0 },
        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
        { 8, 13, 2, 0, 0, 0, 1, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0 }
    };
```

```
    { 8, 13, 0, 0, 0, 0, 1, 0, 7 },  
    { 0, 0, 2, 0, 0, 0, 6, 7, 0 }  
};  
dijkstra(graph, 0);  
return 0;  
}
```

Output:



```
"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"  
Vertex Distance from Source  
0          0  
1          6  
2         14  
3         21  
4         21  
5         11  
6          9  
7          8  
8         15  
  
Process returned 0 (0x0)   execution time : 0.042 s  
Press any key to continue.
```

#Program 17:


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.

```
#include<stdio.h>
#include <stdlib.h>
int s[10] , x[10],d ;
void sumofsub ( int , int , int ) ;
void main ()
{
    int n , sum = 0 ;
    int i ;
    printf ( " \n Enter the size of the set : " ) ;
    scanf ( "%d" , &n ) ;
    printf ( " \n Enter the set in increasing order:\n" ) ;
    for ( i = 1 ; i <= n ; i++ )
        scanf ( "%d" , &s[i] ) ;
    printf ( " \n Enter the value of d : \n " ) ;
    scanf ( "%d" , &d ) ;
    for ( i = 1 ; i <= n ; i++ )
        sum = sum + s[i] ;
    if ( sum < d || s[1] > d )
        printf ( " \n No subset possible : " ) ;
    else
        sumofsub ( 0 , 1 , sum ) ;
    getch () ;
}
void sumofsub ( int m , int k , int r )
{
    int i=1 ;
    x[k] = 1 ;
    if ( ( m + s[k] ) == d )
    {
        printf("Subset:");
        for ( i = 1 ; i <= k ; i++ )
```



```
if ( x[i] == 1 )
printf ( "\\t%d" , s[i] ) ;
printf ( "\\n" ) ;
}
else
if ( m + s[k] + s[k+1] <= d )
sumofsub ( m + s[k] , k + 1 , r - s[k] ) ;
if ( ( m + r - s[k] >= d ) && ( m + s[k+1] <=d ) )
{
x[k] = 0;
sumofsub ( m , k + 1 , r - s[k] ) ;
}
}
```

Output:

 "C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"

```
Enter the size of the set : 3

Enter the set in increasing order:
1 2 3

Enter the value of d :
2
Subset: 2

Process returned 13 (0xD)   execution time : 9.822 s
Press any key to continue.
```

#Program 18:

Implement “N-Queens Problem” using Backtracking.

```
#include<stdio.h>
#include<math.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);
        }
    }
}
}

```

Output:

"C:\Users\Keerthana Natalli P\OneDrive\Desktop\programs\heapsort.exe"

```

- N Queens Problem Using Backtracking -

Enter number of Queens:1

Solution 1:

    1

1      Q
Process returned 0 (0x0)   execution time : 1.445 s
Press any key to continue.

```