

# **Practical 1**

**Aim: - Implement and analyze algorithms given below.**

- 1. Factorial(Iterative and Recursive)**
- 2. Fibonacci Series(Iterative and Recursive)**
- 3. Matrix Addition and Matrix Multiplication(Iterative)**
- 4. Recursive Linear Search and Binary Search(Comparative Study)**

## **1. Factorial(Iterative and Recursive)**

**Theory: -**

Factorial is the product of all positive integers less than or equal to  $n$ . (all non-negative integers). Denoted by  $n!$ . There are two approaches for the calculation of factorial: Iterative approach and recursive approach.

The algorithm takes the input from the user about the number for which one needs the factorial for. Working of both iterative and recursive algorithm is totally different.

### **1.) Iterative Approach**

Iterative approach for obtaining the factorial of any number follows concept of iteration. Here, there's no recursion. In this algorithm, data space occupied is the variables used in the algorithm. The algorithm doesn't depend upon the number of inputs. So, the space complexity remains same, regardless of the big input.

### **2.) Recursive Approach**

The algorithm works on recursion principle. In the recursion, the function calls itself, until a previously set base condition is met. The function `fact()`, for instance, is called recursively, and so the amount of space this program uses is based on the size of the input.

**Algorithm/Pseudo code:**

Step 1. Start  
step 2. Read the number n  
step 3. [Initialize] i=1, fact=1  
step 4. Repeat step 4 through 6 until i = n  
step 5. fact = fact \* i  
step 6. i = i+1  
step 7. Print fact  
step 8. Stop

**Program:-**

1.) Iterative Approach:

**Code: -**

```
#include<iostream>

using namespace std;

main()
{
    long long n,f=1,c=0;

    c++;

    cout<<"Enter the number for which we need the factorial for:"<<endl;

    c++;

    cin>>n;

    c++;

    for(int i=1;i<=n;i++)
    {
        c++;

        f*=i;

        c++;
    }
```

```

}

cout<<"factorial of the number we entered is:"<<f<<endl;

c++;

cout<<"and the count for the code(algorithm) is "<<c;

return 0;

}

```

### Output: -

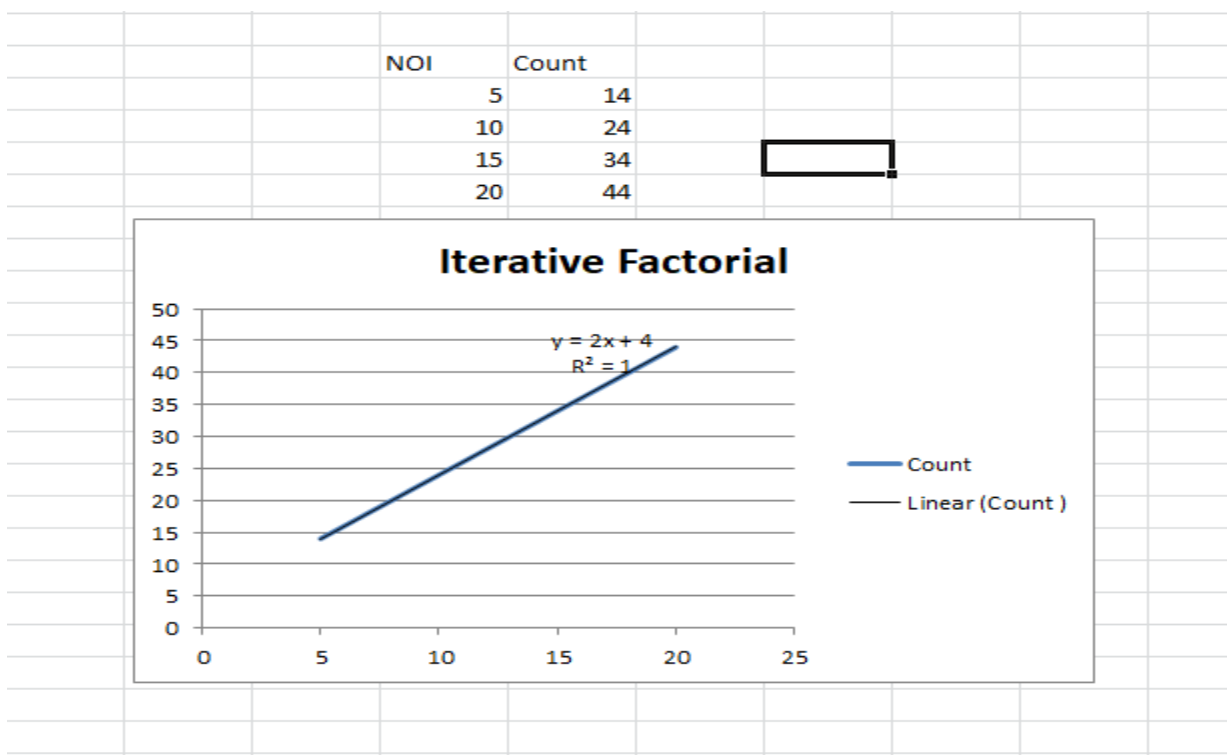
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\iterativefactorial.exe"

```

Enter the number for which we need the factorial for:
5
factorial of the number we entered is:120
and the count for the code(algorithm) is 14
Process returned 0 (0x0)   execution time : 5.553 s
Press any key to continue.

```

### Graph: -



## 2.) Recursive Approach: -

### Code:

```
#include<iostream>

using namespace std;

long long factorial(long long n);

int c=0;

int main()
{
    long long n;

    c++;

    cout << "Enter the number for which we need the factorial for: ";

    c++;

    cin >> n;

    c++;

    cout << "factorial of the number we entered is: "<<n<<" = "<<factorial(n);

    c++;

    cout<<endl;

    c++;

    cout<<"and the count for the code(algorithm) is :"<<c;

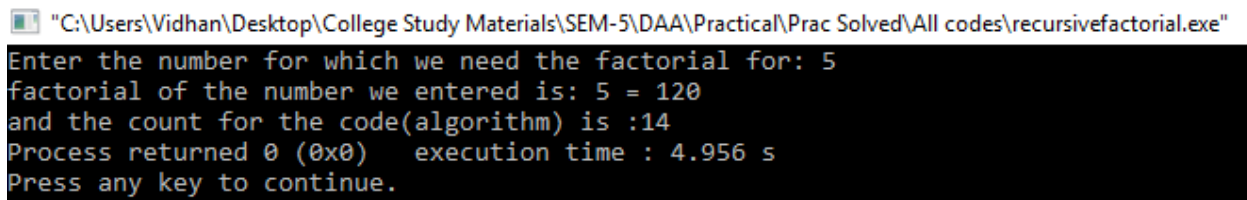
    return 0;
}

long long factorial(long long n)
{
    c++;

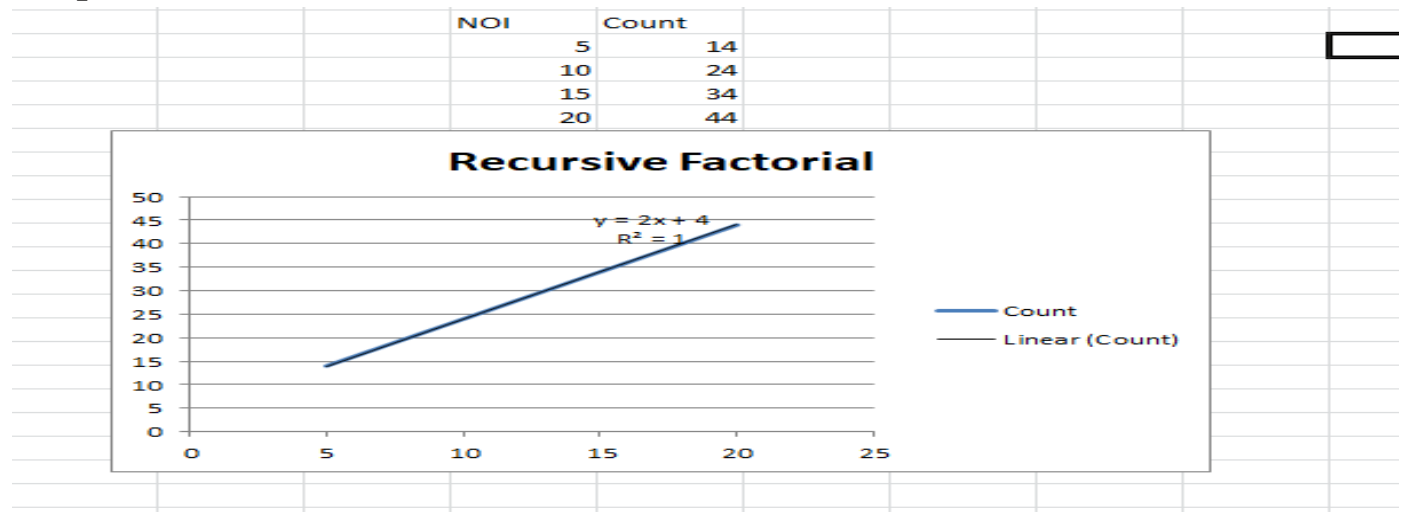
    if(n > 1)
```

```
{  
    c++;  
return n * factorial(n - 1);  
  
    c++;  
}  
  
else  
{  
    return 1;  
  
    c++;  
}  
}
```

### Output:



"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\recursivefactorial.exe"  
Enter the number for which we need the factorial for: 5  
factorial of the number we entered is: 5 = 120  
and the count for the code(algorithm) is :14  
Process returned 0 (0x0) execution time : 4.956 s  
Press any key to continue.

**Graph:****Complexity Table:-**

Algorithm	Complexity
Iterative factorial	$O(n)$
Recursive factorial	$O(n)$

**Conclusion: -**

In cases, Iterative approach and Recursive approach of the algorithm for obtaining the factorial of any number is the same.

Hence, for obtaining the factorial of any number, we can use any of the two approaches.

## 2. Fibonacci Series(Iterative and Recursive)

### Theory: -

Fibonacci series is a mathematical concept of sequences and series. Series of numbers in which each number is the sum of the two preceding numbers. Fibonacci series starts usually with 0.

Formula:  $F_n = F_{n-1} + F_{n-2}$

Example: 0,1,1,2,3,5,8,13,21,34,55,89,144...

The user enters the number up to which one wants the Fibonacci series. Working of both iterative and recursive algorithm is totally different.

### Algorithm/Pseudo code:

- step 1. Start
- step 2. Declare variables i, a, b , show
- step 3. Initialize the variables, a=0, b=1, and show = 0
- step 4. Enter the number of terms of Fibonacci series to be printed
- step 5. Print First two terms of series
- step 6. Use loop for the following steps
  - > show = a+b
  - > a=b
  - > b=show
  - > increase value of i each time by 1
  - > print the value of show
- step 7. End

**Program:-****Iterative Approach:****Code: -**

```
#include<iostream>

using namespace std;

int main()
{
    int c=0;

    int fib1 = 0, fib2 = 1, fib3 = 1,n,i=2;

    c++;

    cout<<"Enter n:";

    c++;

    cin>>n;

    c++;

    cout << "The Fibonacci Series is follows : " << endl << fib1 << " " << fib2 << " ";

    c++;

    while (i<n)
    {
        c++;

        fib3 = fib1 + fib2;

        c++;

        fib1 = fib2;

        c++;

        fib2 = fib3;

        c++;
    }
```



```

i++;

c++;

cout << fib3 << " ";

c++;

}

cout << endl<<"and the count value is ::"<<c;

return 0;

}

```

### Output: -

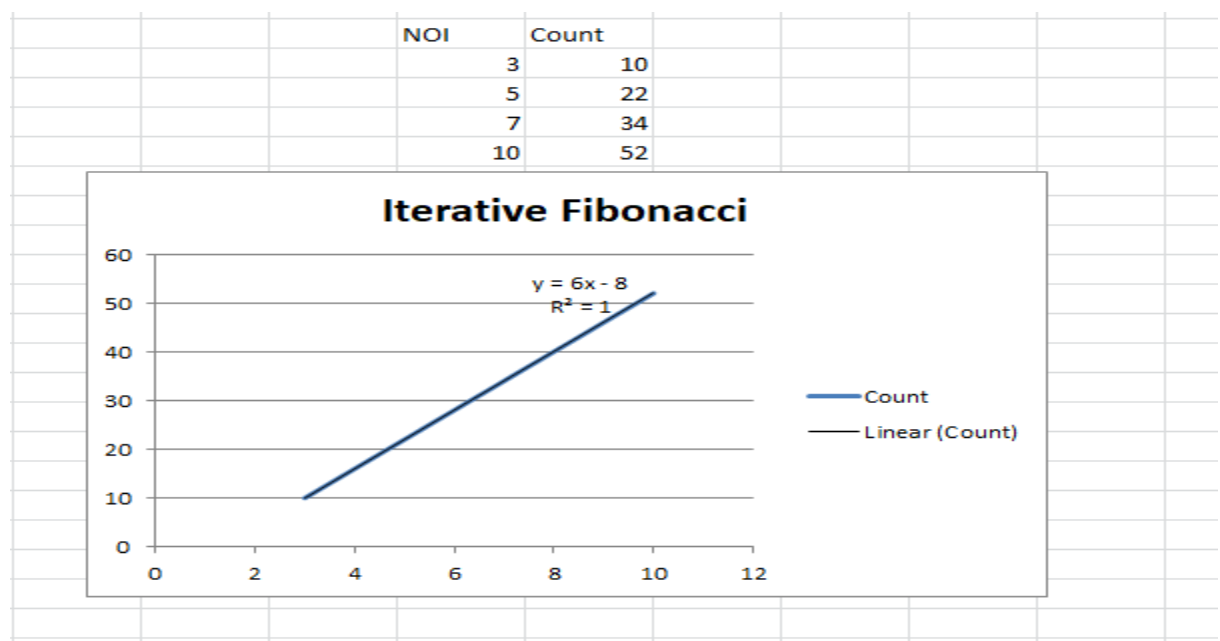
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\iterativefibonacci.exe"

```

Enter n:5
The Fibonacci Series is follows :
0 1 1 2 3
and the count value is ::22
Process returned 0 (0x0)   execution time : 1.618 s
Press any key to continue.

```

### Graph: -



**Recursive Approach: -****Code:**

```
#include<iostream>

using namespace std;

int c=0;

int fibonacci(int n)
{
    c++;
    if((n==1)||(n==0))
    {
        c++;
        return(n);
    }
    else
    {
        return(fibonacci(n-1)+fibonacci(n-2));
        c++;
    }
}

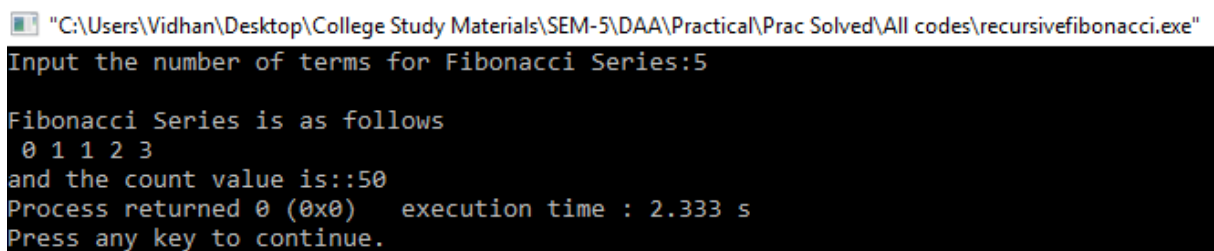
int main()
{
    int n,i=0;

    c++;

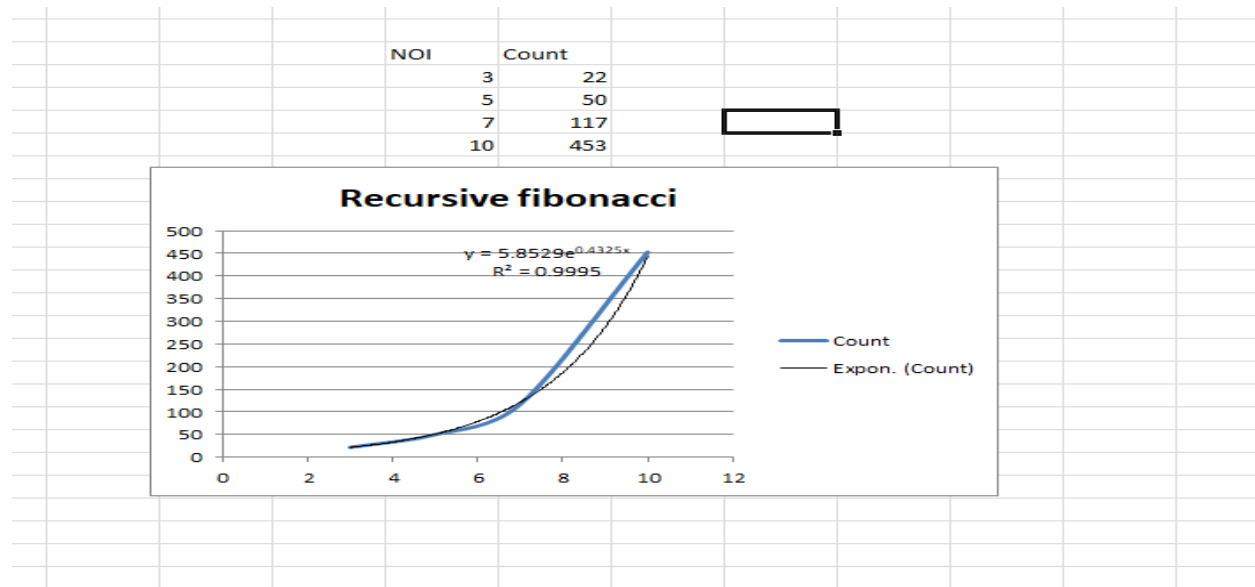
    cout<<"Input the number of terms for Fibonacci Series:";
```

```
c++;  
  
    cin>>n;  
  
    c++;  
  
cout<<"\nFibonacci Series is as follows\n";  
  
c++;  
  
    while(i<n)  
    {  
  
c++;  
  
        cout<<" "<<fibonacci(i);  
  
c++;  
  
i++;  
  
        c++;  
    }  
  
cout<<endl<<"and the count value is::"<<c;  
  
    return 0;  
  
}
```

### Output:



```
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\recursivefibonacci.exe"  
Input the number of terms for Fibonacci Series:5  
  
Fibonacci Series is as follows  
0 1 1 2 3  
and the count value is::50  
Process returned 0 (0x0)   execution time : 2.333 s  
Press any key to continue.
```

**Graph: -****Complexity Table:-**

Algorithm	Complexity
Iterative Fibonacci	$O(n)$
Recursive Fibonacci	$O(2^n)$

**Conclusion: -**

The iterative approach of algorithm for Fibonacci series has time complexity of  $O(n)$ , whereas the recursive approach for the same has time complexity  $O(2^n)$ .

Hence, for obtaining the Fibonacci series, we can use the iterative approach of algorithm for the same.

### 3. Matrix Addition and Matrix Multiplication(Iterative)

#### Matrix Addition:

##### Theory: -

Matrix addition is the operation of adding two matrices by adding the corresponding entries together.

Example:

$$\begin{array}{rcc}
 & \begin{matrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{matrix} & \begin{matrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{matrix} & \begin{matrix} 2 & 2 & 2 \\ 2 & 4 & 6 \\ 2 & 2 & 2 \end{matrix} \\
 A = & & B = & A + B =
 \end{array}$$

#### Algorithm/Pseudo code:

```

Step1: Start
Step2: Read: m and n
Step3: Read: Take inputs for Matrix A[1:m, 1:n] and Matrix B[1:m, 1:n]
Step4: Repeat for i = 1 to m by 1:
    Repeat for j = 1 to n by 1:
        C[i, j] = A[i, j] + B[i, j]
    [End of inner for loop]
    [End of outer for loop]
Step5: Print: Matrix C
Step6: Exit
  
```

**Program:-****Iterative Approach:****Code: -**

```
#include <iostream>

using namespace std;

int C=0;

int main()
{
    int r, c, a[100][100], b[100][100], sum[100][100], i, j;
    C++;
    cout << "Enter number of rows (between 1 and 100): ";
    C++;
    cin >> r;
    C++;
    cout << "Enter number of columns (between 1 and 100): ";
    C++;
    cin >> c;
    C++;
    cout << endl << "Enter elements of 1st matrix: " << endl;
    C++;
    for(i = 0; i < r; ++i){
        C++;
        for(j = 0; j < c; ++j)
        {
```

```
C++;

cout << "Enter element a" << i + 1 << j + 1 << " : ";

C++;

cin >> a[i][j];

C++;

    }

}

cout << endl << "Enter elements of 2nd matrix: " << endl;

C++;

for (i = 0; i < r; ++i){

    C++;

    for(j = 0; j < c; ++j)

    {

        C++;

        cout << "Enter element b" << i + 1 << j + 1 << " : ";

        C++;

        cin >> b[i][j];

        C++;

    }

}

for(i = 0; i < r; ++i){

    C++;

    for(j = 0; j < c; ++j){

        C++;

        sum[i][j] = a[i][j] + b[i][j];
```

```
C++;  
  
}  
  
}  
  
    cout << endl << "Sum of two matrix is: " << endl;  
  
C++;  
  
for(i = 0; i < r; ++i){  
  
    C++;  
  
    for(j = 0; j < c; ++j)  
    {  
        C++;  
  
        cout << sum[i][j] << " ";  
  
        C++;  
  
        if(j == c - 1){  
            C++;  
  
            cout << endl;  
  
            C++;  
  
        }  
    }  
  
}  
  
    cout<<"and the value of the count is :"<<C;  
  
    return 0;  
  
}
```



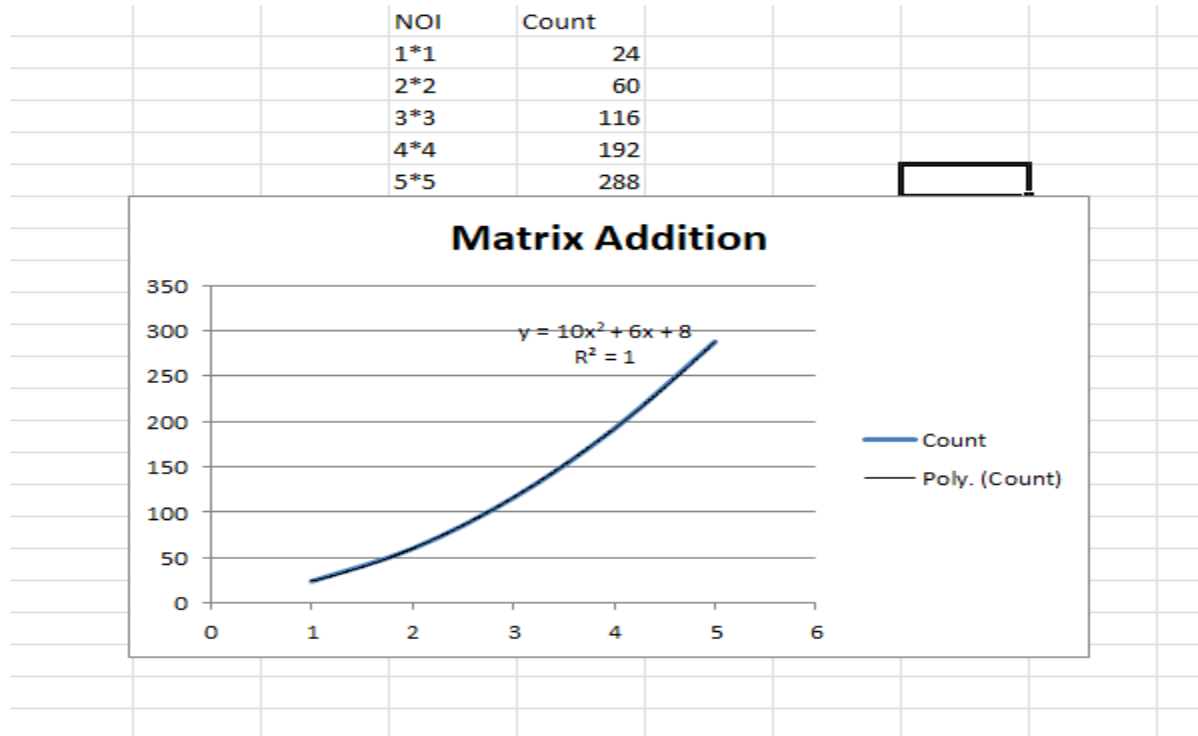
**Output: -**

```
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\matrixaddition.exe"
Enter number of rows (between 1 and 100): 4
Enter number of columns (between 1 and 100): 4

Enter elements of 1st matrix:
Enter element a11 : 1
Enter element a12 : 2
Enter element a13 : 3
Enter element a14 : 4
Enter element a21 : 1
Enter element a22 : 2
Enter element a23 : 3
Enter element a24 : 4
Enter element a31 : 2
Enter element a32 : 3
Enter element a33 : 4
Enter element a34 : 5
Enter element a41 : 3
Enter element a42 : 4
Enter element a43 : 5
Enter element a44 : 6

Enter elements of 2nd matrix:
Enter element b11 : 1
Enter element b12 : 2
Enter element b13 : 3
Enter element b14 : 4
Enter element b21 : 1
Enter element b22 : 2
Enter element b23 : 3
Enter element b24 : 4
Enter element b31 : 2
Enter element b32 : 3
Enter element b33 : 4
Enter element b34 : 5
Enter element b41 : 3
Enter element b42 : 4
Enter element b43 : 5
Enter element b44 : 6

Sum of two matrix is:
2 4 6 8
2 4 6 8
4 6 8 10
6 8 10 12
and the value of the count is :192
Process returned 0 (0x0)   execution time : 45.172 s
Press any key to continue.
```

**Graph: -****Complexity Table:-**

Algorithm	Complexity
Iterative Addition	$O(n^2)$

**Conclusion: -**

Matrix addition algorithm by iterative approach has the time complexity of  $O(n^2)$ .

## Matrix Multiplication:

### Theory: -

Matrix multiplication is a binary operation that takes a pair of matrices, and produces another matrix. Numbers such as the real or complex numbers can be multiplied according to elementary arithmetic

.Example:

$$\begin{array}{ccc}
 \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array} & 
 \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} & 
 \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array} \\
 A = & B = & A \times B =
 \end{array}$$

### Algorithm/Pseudo code:

```

Step1: Start.
Step2: Read: m, n, p and q
Step3: Read: Inputs for Matrices A[1:m, 1:n] and B[1:p, 1:q].
Step4: Repeat for i := 1 to m by 1:
    Repeat for j := 1 to q by 1:
        C[i, j] := 0 [Initializing]
        Repeat k := 1 to n by 1
            C[i, j] := C[i, j] + A[i, k] x B[k, j]
        [End of for loop]
    [End of for loop]
[End of for loop]
Step5: Print: C[1:m, 1:q]
Step6: Exit
  
```

**Program:-****Iterative Approach:****Code: -**

```
#include <iostream>

using namespace std;

int C=0;

int main()

{

int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;

C++;

cout<<"enter the number of row=";

C++;

cin>>r;

C++;

cout<<"enter the number of column=";

C++;

cin>>c;

C++;

cout<<"enter the first matrix element=\n";

C++;

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

{

C++;

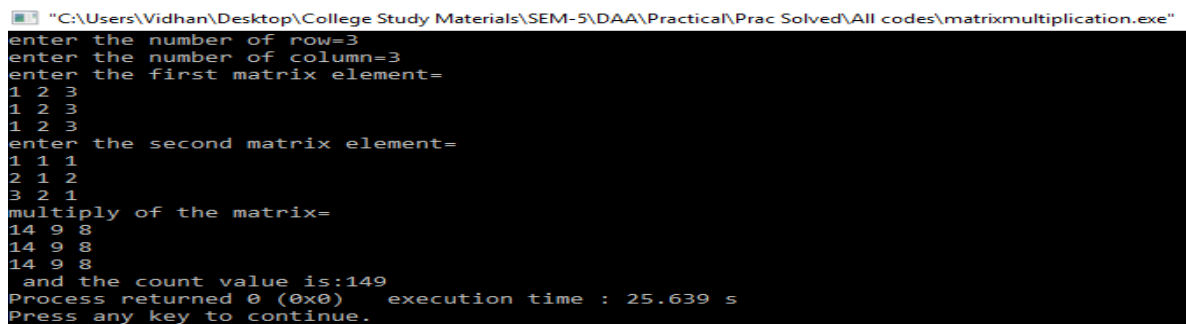
for(j=0;j<c;j++)

{
```

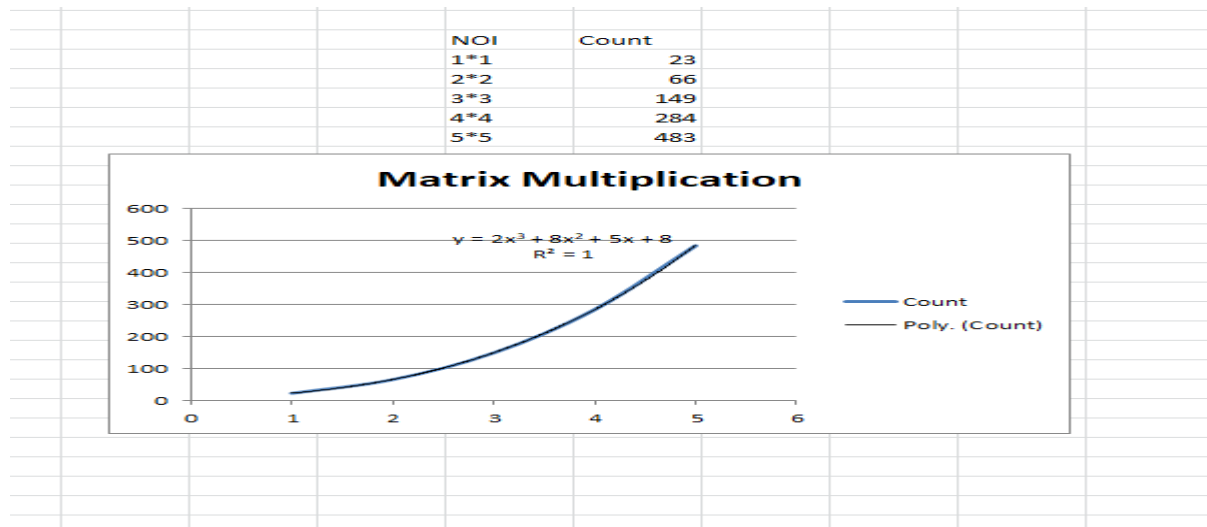
```
C++;  
cin>>a[i][j];  
C++;  
}  
}  
  
cout<<"enter the second matrix element=\n";  
  
C++;  
for(i=0;i<r;i++)  
{  
    C++;  
    for(j=0;j<c;j++)  
    {  
        C++;  
        cin>>b[i][j];  
        C++;  
    }  
}  
  
cout<<"multiply of the matrix=\n";  
  
C++;  
for(i=0;i<r;i++)  
{  
    mul[i][j]=0;  
    C++;  
    for(k=0;k<c;k++)  
    {
```

```
C++;  
mul[i][j]+=a[i][k]*b[k][j];  
C++;  
}  
}  
}  
for(i=0;i<r;i++)  
{  
    C++;  
    C++;  
    cout<<mul[i][j]<<" ";  
    C++;  
}  
cout<<"\n";  
C++;  
}  
cout<<" and the count value is:"<<C;  
return 0;  
}
```

### Output: -



```
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\matrixmultiplication.exe"  
enter the number of row=3  
enter the number of column=3  
enter the first matrix element=  
1 2 3  
1 2 3  
1 2 3  
enter the second matrix element=  
1 1 1  
2 1 2  
3 2 1  
multiply of the matrix=  
14 9 8  
14 9 8  
14 9 8  
and the count value is:149  
Process returned 0 (0x0)   execution time : 25.639 s  
Press any key to continue.
```

**Graph: -****Complexity Table:-**

Algorithm	Complexity
Iterative Multiplication	$O(n^3)$

**Conclusion: -**

Matrix multiplication algorithm by iterative approach has the time complexity of  $O(n^3)$ .

## 4. Recursive Linear Search and Binary Search(Comparative Study)

### 1. Linear Search:

#### Theory: -

Linear search is a normal and basic type of searching algorithm. Linear search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched. Linear search runs in at worst linear time and makes at most  $n$  comparisons, where  $n$  is the length of the list.

#### Algorithm/Pseudo code:

Step 1: Start  
Step 2: Set  $i$  to 1  
Step 3: if  $i > n$  then go to step 7  
Step 4: if  $A[i] = x$  then go to step 6  
Step 5: Set  $i$  to  $i+1$   
Step 6: Go to step 2  
Step 7: Print element  $x$  found at index  $i$  and go to step 8  
Step 8: Print element not found  
Step 9: Exit



**Program:-****Code: -**

```
#include<iostream>

using namespace std;

int c=0;

int recursiveLinearSearch(int array[],int key,int size) {

    c++;

    size=size-1;

    c++;

    if(size <0) {

        c++;

        return -1;

    c++;

    } else if(array[size]==key) {

        c++;

        return 1;

        c++;

    } else {

        c++;

        return recursiveLinearSearch(array,key,size);

        c++;

    }

}

int main() {

    cout<<"Enter The Size Of Array:  ";
```

```
c++;  
int size;  
c++;  
cin>>size;  
c++;  
int array[size], key,i;  
c++;  
// Taking Input In Array  
for (int j=0;j<size;j++) {  
c++;  
    cout<<"Enter "<<j<<" Element : ";  
c++;  
    cin>>array[j];  
    c++;  
}  
//Your Entered Array Is  
for (int a=0;a<size;a++) {  
c++;  
    cout<<"array[ "<<a<<" ] = ";  
c++;  
    cout<<array[a]<<endl;  
    c++;  
}  
cout<<"Enter Key To Search in Array";  
c++;
```

```
cin>>key;

c++;

int result;

c++;

result=recursiveLinearSearch(array,key,size--);

c++;

if(result==1) {

c++;

    cout<<"Key Found in Array ";

c++;

} else {

    c++;

    cout<<"Key NOT Found in Array ";

    c++;

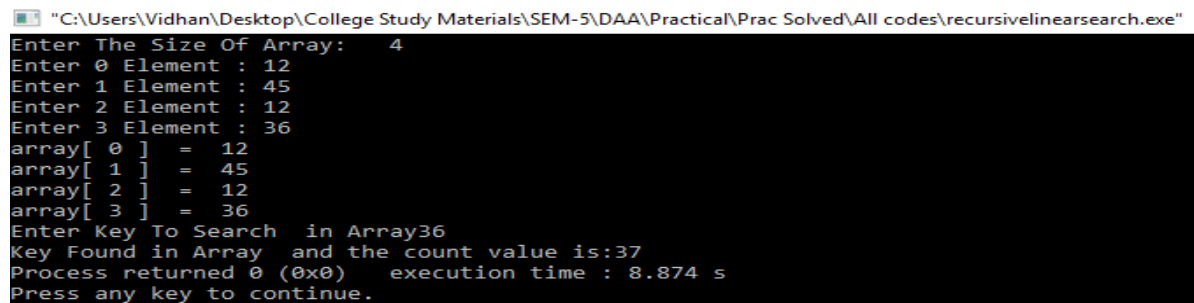
}

cout<<"and the count value is:"<<c;

return 0;

}
```

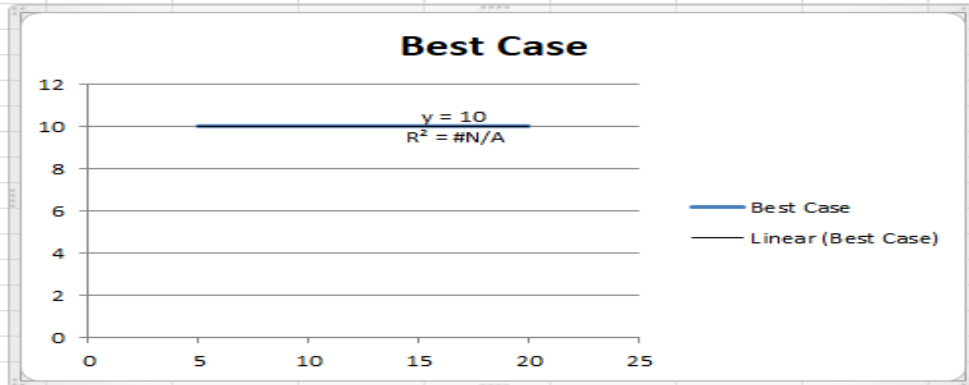
### Output: -



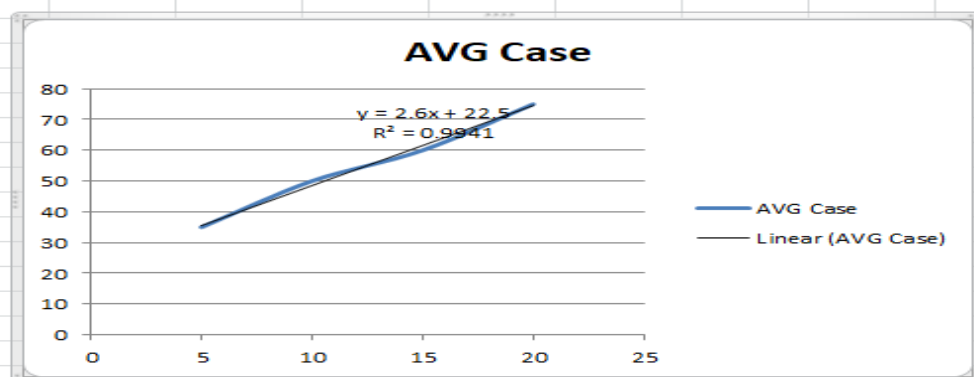
```
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\recursivelinearsearch.exe"
Enter The Size Of Array: 4
Enter 0 Element : 12
Enter 1 Element : 45
Enter 2 Element : 12
Enter 3 Element : 36
array[ 0 ] = 12
array[ 1 ] = 45
array[ 2 ] = 12
array[ 3 ] = 36
Enter Key To Search in Array36
Key Found in Array and the count value is:37
Process returned 0 (0x0) execution time : 8.874 s
Press any key to continue.
```

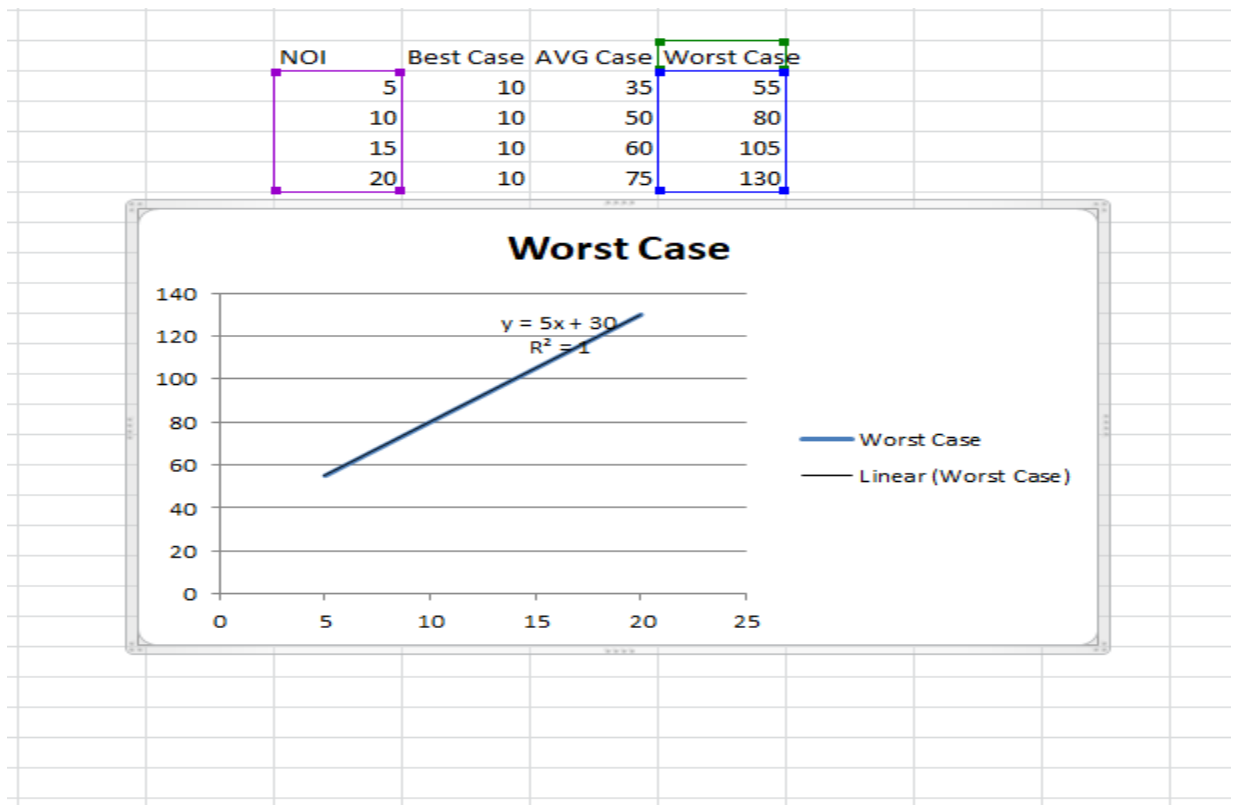
**Graph: -**

NOI	Best Case	AVG Case	Worst Case
5	10	35	55
10	10	50	80
15	10	60	105
20	10	75	130



NOI	Best Case	AVG Case	Worst Case
5	10	35	55
10	10	50	80
15	10	60	105
20	10	75	130





### Complexity Table:-

Algorithm	Best Case	Avg Case	Worst Case
Recursive Linear Search	O(1)	O(n)	O(n)

The Time Complexity of algorithm for Linear Search (recursive) is :

**Best Case - O(1)**

**Average Case - O(n)**

**Worst Case - O(n)**

## 2. Binary Search:

### Theory: -

Binary search is a search algorithm that finds the position of a target value within a sorted array. It compares the target value to the middle element of the array; if they are unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful.

### Algorithm/Pseudo code:

```
Procedure binary_search
A <- sorted array
n <- size of array
x <- value to be searched

Set lowerBound = 1
Set upperBound = n

BINARYS (lowerBound, upperBound, A, x)
    if upperBound < lowerBound
        Return -1

    Set midpoint = ( lowerBound + upperBound ) / 2

    If (A[midpoint] < x)
        Return (midpoint+1, upperBound, A, x)
    Else if (A[midpoint] > x)
        Return (lowerBound, midpoint-1, A, x)
    Else
        Return midpoint
```

**Program:-****Recursive Approach:****Code: -**

```
#include<iostream>

using namespace std;

int binary_search(int arr[],int n,int low,int high,int x);

int c=0;

int main()

{

int arr[50], i, num, n,high,low;

cout<<"Enter the array size : ";

cin>>n;

cout<<"Enter Array Elements : ";

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

{

cin>>arr[i];

}

cout<<"Enter the number to be search : ";

cin>>num;

high=n-1;

low=0;

if(binary_search(arr,n,low,high,num)==-1)

{

cout<<endl<<"not found";
```

```
}  
else  
{  
cout<<endl<<"found at position:: "<<binary_search(arr,n,low,high,num)+1;  
}  
    cout<<endl<<"Counter:: "<<c;  
}  
int binary_search(int arr[],int n,int low,int high,int x)  
{  
    c++;  
    int mid=(low+high)/2;  
    if(x==arr[mid])  
    {  
        return mid;  
    }  
    else if(x<arr[mid])  
    {  
        binary_search(arr,n,low,mid+1,x);  
    }  
    else if(x>arr[mid])  
    {  
        binary_search(arr,n,mid-1,high,x);  
    }  
}
```



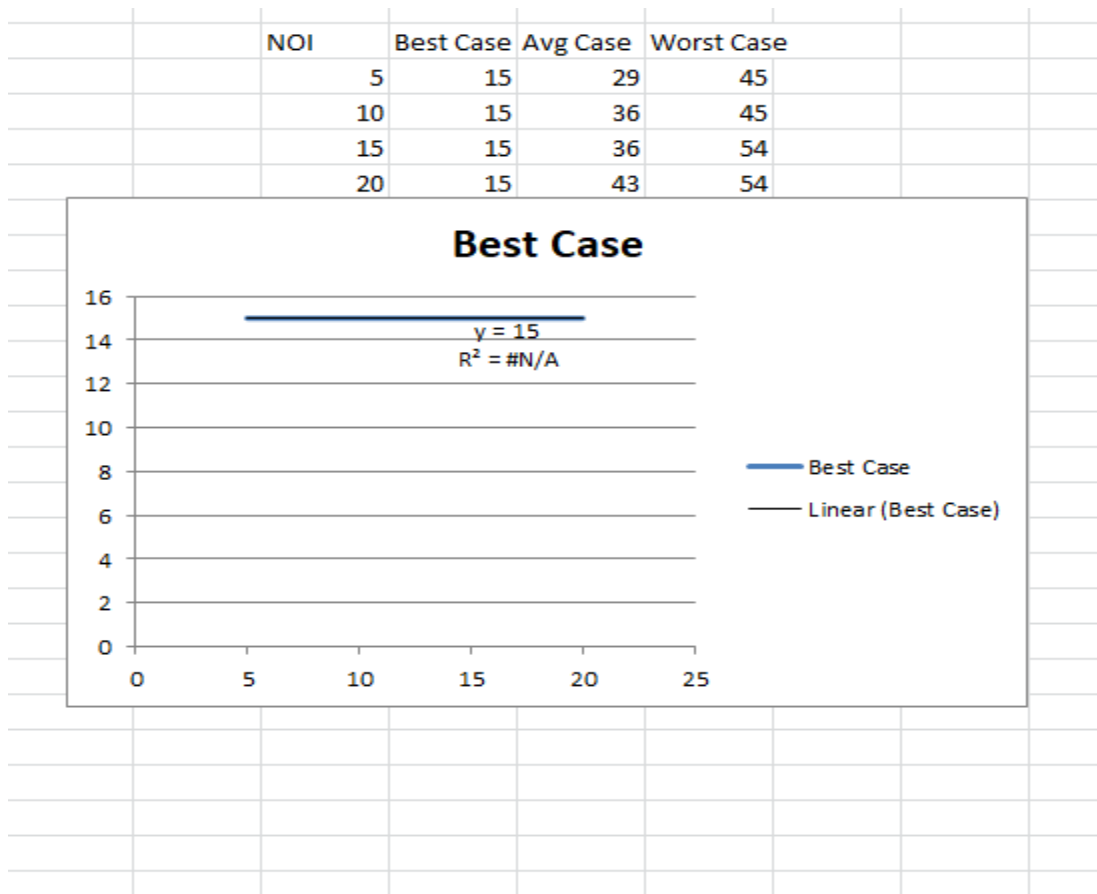
**Output: -**

```

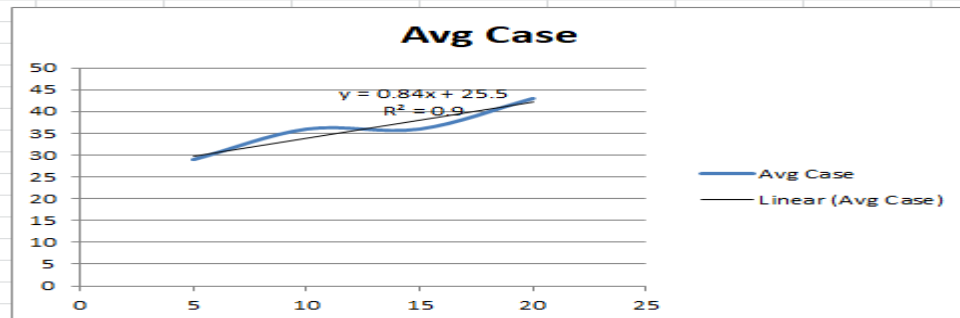
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\binarysearch.exe"
Enter the array size : 5
Enter Array Elements : 1
2
3
4
5
Enter the number to be search : 3

found at position:: 3
Counter:: 2
Process returned 0 (0x0)   execution time : 6.883 s
Press any key to continue.

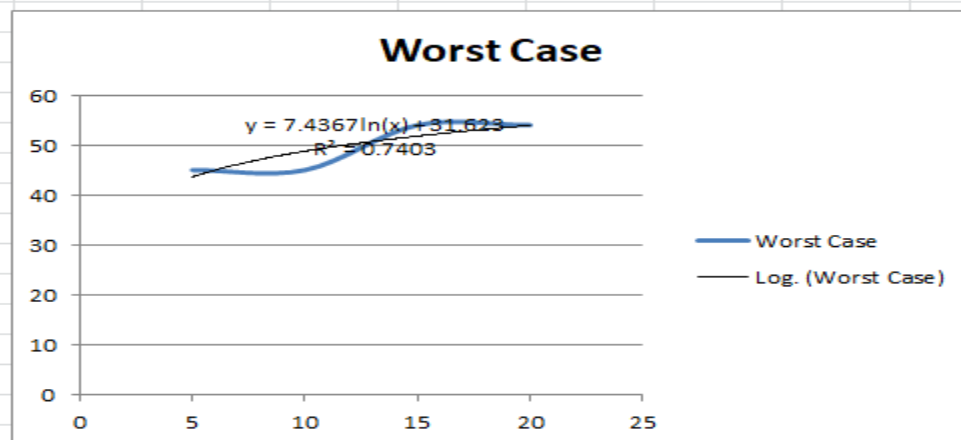
```

**Graph: -**

NOI	Best Case	Avg Case	Worst Case
5	15	29	45
10	15	36	45
15	15	36	54
20	15	43	54



NOI	Best Case	Avg Case	Worst Case
5	15	29	45
10	15	36	45
15	15	36	54
20	15	43	54



**Complexity Table:-**

Algorithm	Best Case	Avg Case	Worst Case
Recursive Linear Search	$O(1)$	$O(n)$	$O(\log n)$

- The Time Complexity of algorithm for Binary Search (recursive) is:

**Best Case -  $O(1)$**

**Average Case -  $O(\log n)$**

**Worst Case -  $O(\log n)$**

**Comparative study of recursive linear and binary search:-**

Algorithm	Best Case	Avg Case	Worst Case
Recursive Linear	$O(1)$	$O(n)$	$O(n)$
Binary Search	$O(1)$	$O(n)$	$O(\log n)$

**Conclusion:-**

Binary search algorithm is a better searching algorithm for searching of elements, having time complexity of  $O(\log n)$  in its worst case.

Hence, for searching of elements, binary search must be preferred over linear search.