# PRACTICAL-1

# AIM: Implement and analyze algorithms given below

# 1. Factorial (Iterative and Recursive).

**PROGRAM(Iterative):**

#include <iostream>

using namespace std;

int main()

{

int i,fact=1,number,counter=0;

cout<<"Enter any Number: ";

cin>>number;

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

fact=fact\*i;

counter++;

}

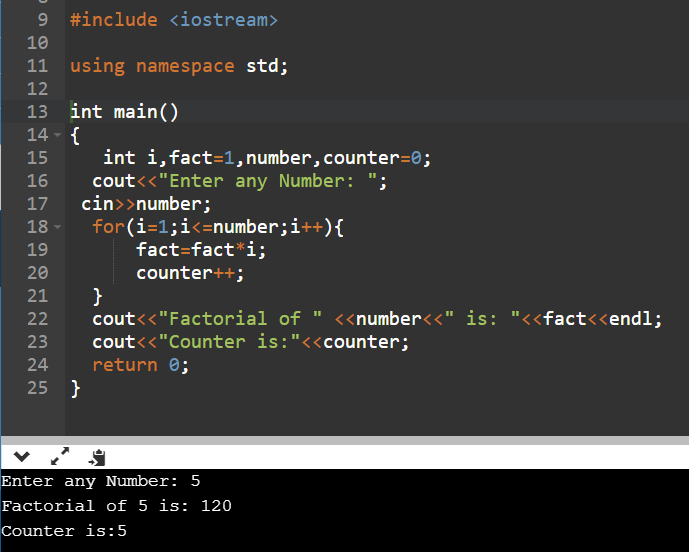
cout<<"Factorial of " <<number<<" is: "<<fact<<endl;

cout<<"Counter is:"<<counter;

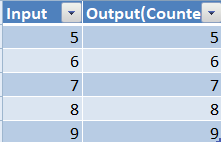
return 0;

}

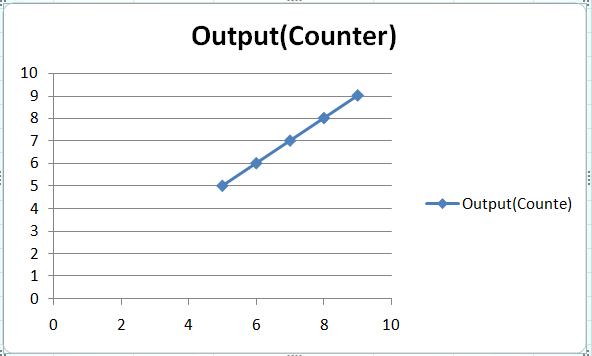
**OUTPUT:**

****

**ANALYSIS TABLE:**

****

**GRAPH:**

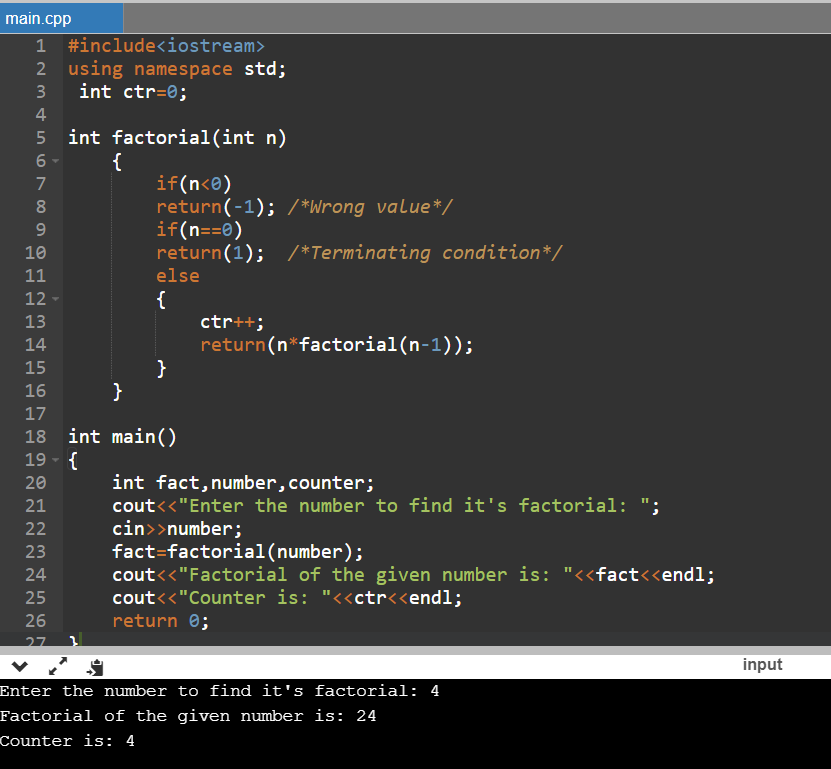
****

**CONCLUSION:** I Implemented and analyzed algorithms given below 1 Factorial using iterative method.

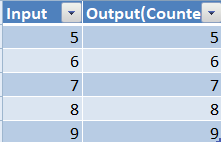
**PROGRAM(Recursive):**

#include<iostream>  
using namespace std;  
 int ctr=0;  
  
int factorial(int n)  
    {  
        if(n<0)  
        return(-1); /\*Wrong value\*/  
        if(n==0)  
        return(1);  /\*Terminating condition\*/  
        else  
        {  
            ctr++;  
            return(n\*factorial(n-1));  
        }  
    }  
  
int main()  
{  
    int fact,number,counter;  
    cout<<"Enter the number to find it's factorial: ";  
    cin>>number;  
    fact=factorial(number);  
    cout<<"Factorial of the given number is: "<<fact<<endl;  
    cout<<"Counter is: "<<ctr<<endl;  
    return 0;  
}

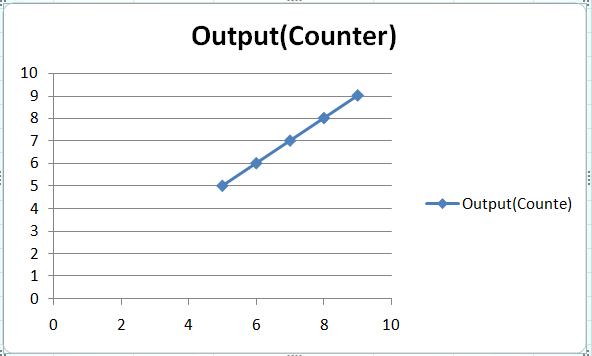
**OUTPUT:**



**ANALYSIS TABLE:**

****

**GRAPH:**



**CONCLUSION:** I Implemented and analyzed algorithms given below 1 Factorial using Recursive method.

# AIM: Implement and analyze algorithms given below

**2.** **Euclidean algorithm**

**PROGRAM CODE:**

#include <iostream>

using namespace std;

int ctr=0;

int gcd(int a, int b)

{

if (b == 0)

return a;

ctr++;

return gcd(b, a % b);

}

int main()

{

int a , b;

cout<<"Enter the values of a and b: "<<endl;

cin>>a>>b;

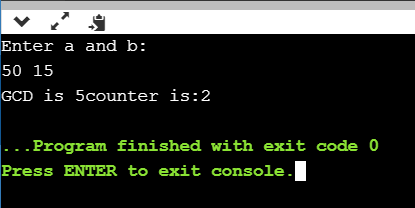
cout<<"GCD of "<< a <<" and "<< b <<" is "<< gcd(a, b);

cout<<"counter is:"<<ctr;

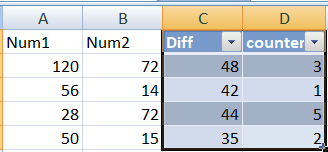
return 0;

}

**OUTPUT:**

****

**ANALYSIS TABLE:**

****

**GRAPH:**

**CONCLUSION:** I have Implemented and analyzed Euclidean algorithm.

# AIM: Implement and analyze algorithms given below

**4.** **Recursive Linear Search and Binary Search**

**PROGRAM CODE:**

#include <stdio.h>

int counter=0;

int RecursiveLS(int arr[], int value, int index, int n)

{

counter++;

int pos = 0;

if(index >= n)

{

return 0;

}

else if (arr[index] == value)

{

pos = index + 1;

return pos;

}

else

{

return RecursiveLS(arr, value, index+1, n);

}

return pos;

}

int main()

{

int n, value, pos, m = 0, arr[100];

printf("Enter the total elements in the array: ");

scanf("%d", &n);

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

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

{

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

}

printf("Enter the element to search: ");

scanf("%d", &value);

pos = RecursiveLS(arr, value, 0, n);

if (pos != 0)

{

printf("Element found at pos %d ", pos);

}

else

{

printf("Element not found");

}

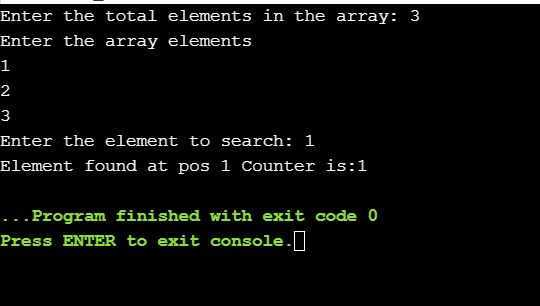
printf("Counter is:%d",counter);

return 0;

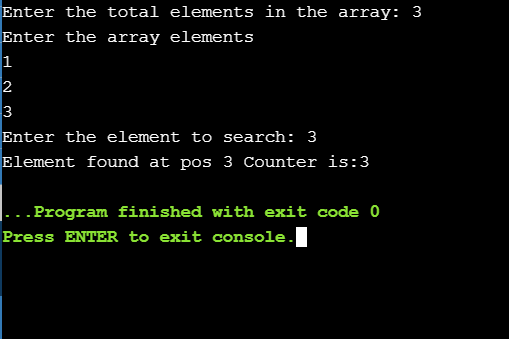
}

**OUTPUT:**

**Best Case:**

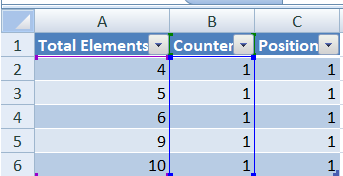
****

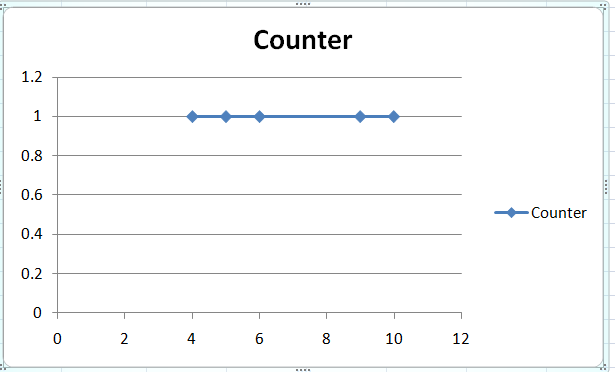
**Worst Case:**

****

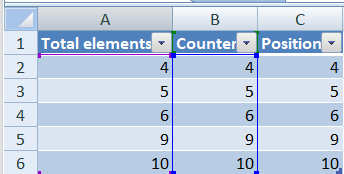
**ANALYSIS TABLE:**

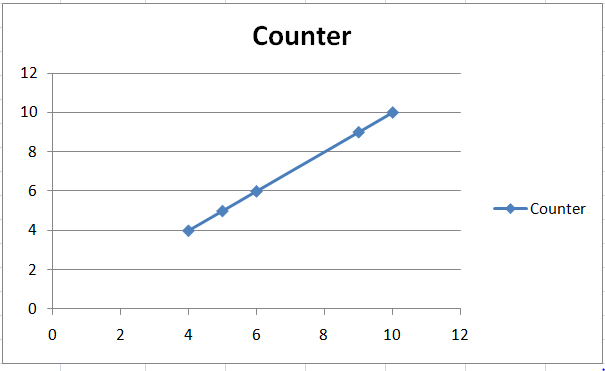
**Best Case:**

****

****

**Worst Case:**

****

****

**#Binary Search:**

**Program Code:**

#include <stdio.h>

int counter=0;

void binary\_search(int [], int, int, int);

int main()

{

int key, size, i;

int list[25];

printf("Enter size of a list: ");

scanf("%d", &size);

printf("Enter elements\n");

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

{

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

}

printf("\n");

printf("Enter key to search\n");

scanf("%d", &key);

binary\_search(list, 0, size, key);

}

void binary\_search(int list[], int lo, int hi, int key)

{

int mid;

counter++;

if (lo > hi)

{

printf("Key not found\n");

return;

}

mid = (lo + hi) / 2;

if (list[mid] == key)

{

printf("Key found\n");

printf("Counter is:%d",counter);

}

else if (list[mid] > key)

{

binary\_search(list, lo, mid - 1, key);

}

else if (list[mid] < key)

{

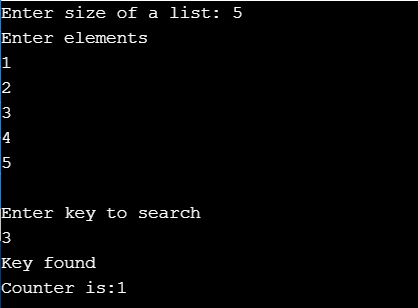
binary\_search(list, mid + 1, hi, key);

}

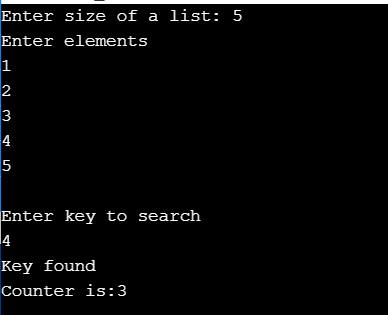
}

**OUTPUT:**

**Best Case:**

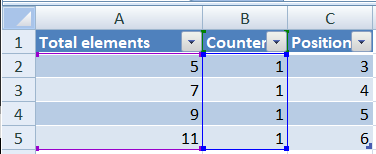
****

**Worst Case:**

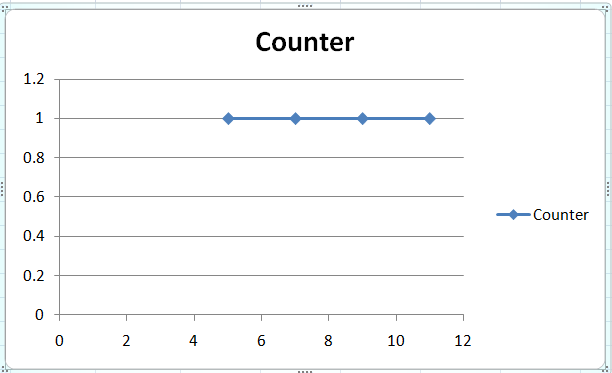
****

**ANALYSIS TABLE:**

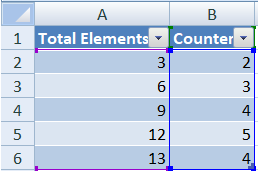
**Best Case:**

****

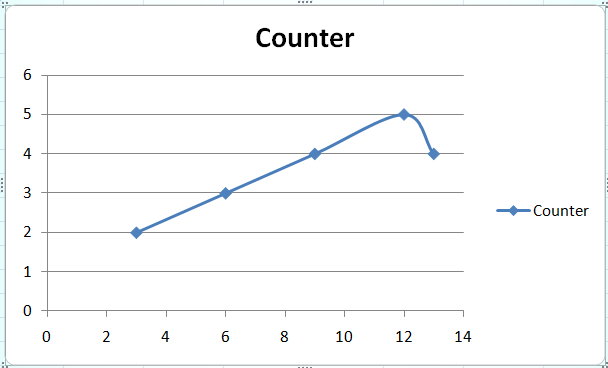
**Graph:**

****

**Worst Case:**

****

**Graph:**

****

**CONCLUSION:** Here we concluded by implementing and analyzing the program of linear and binary search the time complexity of linear search best case is O(1) and worst case is O(n) whereas in binary search the best case is O(1) and the worst case is also O(logn) by analyzing the graph of linear search in best case is constant and of worst case is linear whereas that of binary search the graph of best case is constant and of worst case is non-linear.

.

# AIM: Implement and analyze algorithms given below.

# 

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

**Matrix Addition**

**PROGRAM CODE:**

#include<iostream>

using namespace std;

int main()

{

int m, n, c, d, first[10][10], second[10][10], sum[10][10],counter=0;

cout<<"\nEnter the number of rows and columns of matrix :";

cin>>m>>n;

cout<<"\nEnter the elements of first matrix : \n";

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

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

cin>>first[c][d];

cout<<"\nEnter the elements of second matrix : \n";

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

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

cin>>second[c][d];

cout<<"\nSum of entered matrices :-\n";

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

for (d = 0 ; d < n; d++) {

counter++;

sum[c][d] = first[c][d] + second[c][d];

cout<<sum[c][d]<<" ";

}

cout<<endl;

}

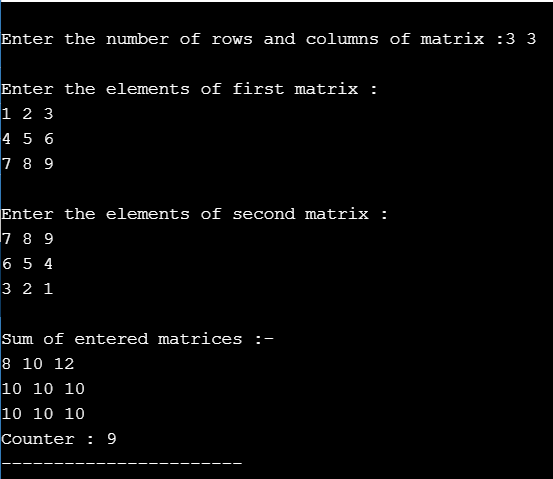
cout<<"Counter : "<<counter;

cout<<"\n-----------------------\n";

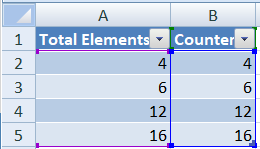
return 0;

}

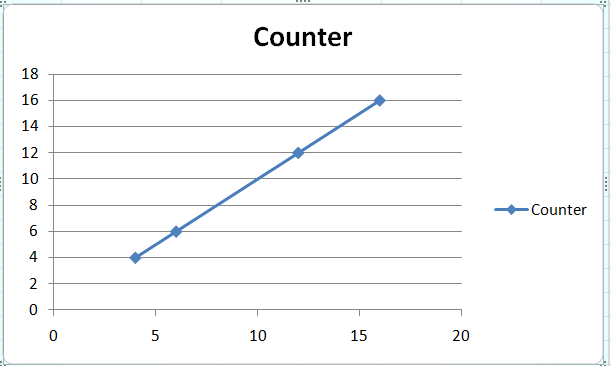
**OUTPUT:**

****

**ANALYSIS TABLE:**

****

**GRAPH:**

****

***Matrix Multiplication:***

**PROGRAM CODE:**

#include<iostream>

using namespace std;

int main(){

int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;

int counter=0;

cout << "Enter rows and columns for first matrix: ";

cin >> r1 >> c1;

cout << "Enter rows and columns for second matrix: ";

cin >> r2 >> c2;

// If column of first matrix in not equal to row of second matrix,

// ask the user to enter the size of matrix again.

while (c1!=r2)

{

cout << "Error! column of first matrix not equal to row of second.";

cout << "Enter rows and columns for first matrix: ";

cin >> r1 >> c1;

cout << "Enter rows and columns for second matrix: ";

cin >> r2 >> c2;

}

// Storing elements of first matrix.

cout << endl << "Enter elements of matrix 1:" << endl;

for(i = 0; i < r1; ++i)

for(j = 0; j < c1; ++j)

{

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

cin >> a[i][j];

}

// Storing elements of second matrix.

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

for(i = 0; i < r2; ++i)

for(j = 0; j < c2; ++j)

{

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

cin >> b[i][j];

}

// Initializing elements of matrix mult to 0.

for(i = 0; i < r1; ++i)

for(j = 0; j < c2; ++j)

{

mult[i][j]=0;

}

for(i = 0; i < r1; ++i)

for(j = 0; j < c2; ++j){

for(k = 0; k < c1; ++k)

{

counter++;

mult[i][j] += a[i][k] \* b[k][j];

}

}

cout << endl << "Output Matrix: " << endl;

for(i = 0; i < r1; ++i)

for(j = 0; j < c2; ++j)

{

cout << " " << mult[i][j];

if(j == c2-1)

cout << endl;

}

cout<<"\nCounter : "<<counter;

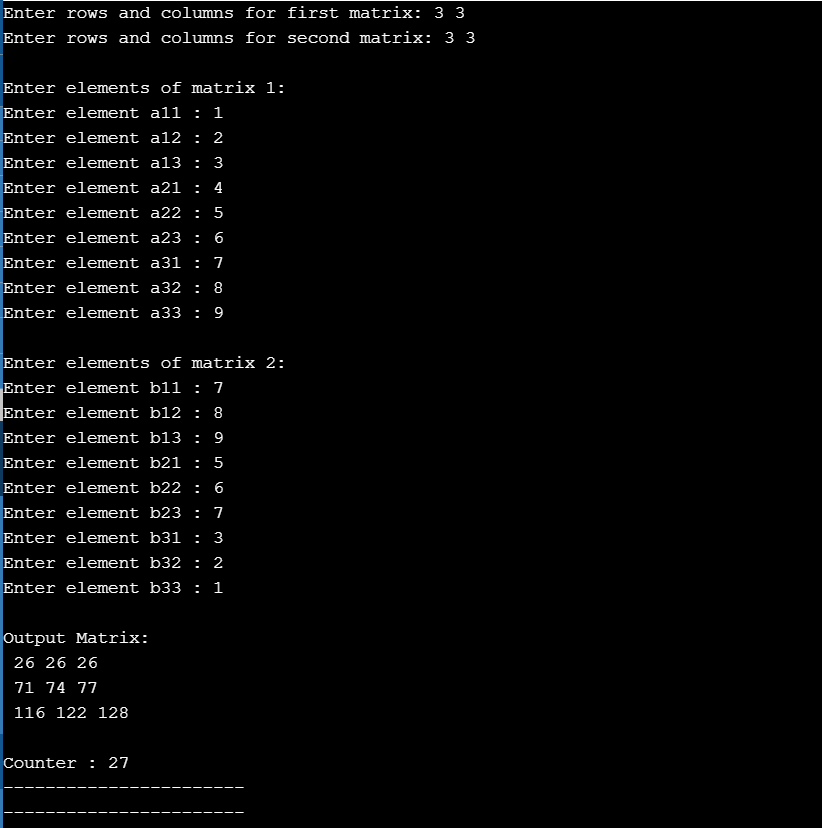
cout<<"\n-----------------------\n";

cout<<"-----------------------\n";

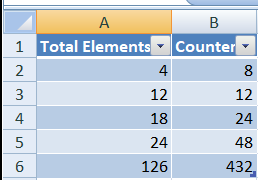
return 0;

}

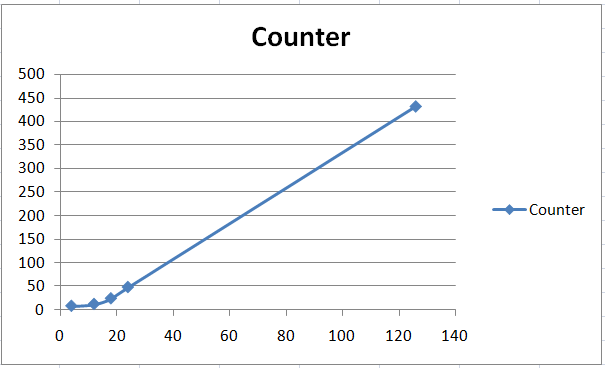
**OUPUT:**

****

**ANALYSIS TABLE:**

****

**GRAPH:**

****

**CONCLUSION:** Here we concluded by implementing and analyzing the program of matrix addition and multiplication, the time complexity of matrix addition is O(n^2) and of matrix multiplication is O(n^) by analyzing the graph of matrix addition is linear whereas of matrix multiplication is slightly curved in beginning and then linear.

**1.5** Find a subset of a given set S = {s1,s2,.....,sn} 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.

**PROGRAM CODE:**

#include<iostream>

using namespace std;

int counter=0;

boolisSubsetSum(int set[], int n, int sum)

{

counter++;

if (sum == 0)

return true;

if (n == 0 && sum != 0)

return false;

if (set[n-1] > sum)

return isSubsetSum(set, n-1, sum);

return isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum-set[n-1]);

}

int main()

{

int set[] = {1,2,7,8,3,5,9,99};

int sum = 15;

int n = sizeof(set)/sizeof(set[0]);

if (isSubsetSum(set, n, sum) == true)

cout<<"\nSubset is found for given sum\n";

else

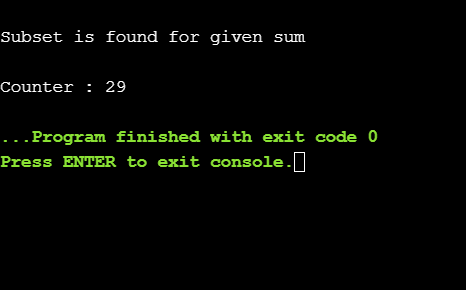
cout<<"No subset with given sum";

cout<<"\nCounter : "<<counter;

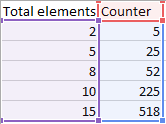
return 0;

}

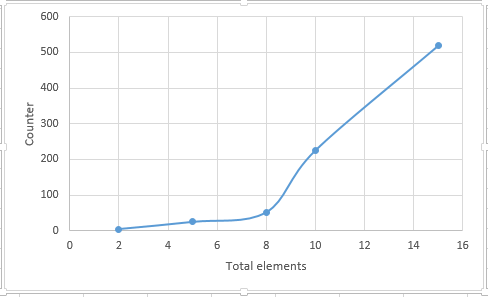
**Output:**

****

**Analysis Table:**



**Graph:**



**CONCLUSION:** Here we concluded by implementing and analyzing the program of subset sum the time complexity of subset sum is O(2^n) and by analyzing the graph is non-linear.

**PRACTICAL-2**

# Implement and analyze algorithms given below.(Compare them)

# 2.1) Bubble Sort

**PROGRAM CODE:**

#include<iostream>

using namespace std;

int counter=0;

void swapping(int &a, int &b) {

int temp;

temp = a;

a = b;

b = temp;

}

void display(int \*array, int size)

{

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

cout << array[i] << " ";

cout << endl;

}

void bubbleSort(int \*array, int size)

{

for(int i = 0; i<size; i++) {

int swaps = 0;

for(int j = 0; j<size-i-1; j++)

{

counter++;

if(array[j] > array[j+1]) {

swapping(array[j], array[j+1]);

swaps = 1; //set swap flag

}

}

if(!swaps)

break;

}

}

int main()

{

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter elements:" << endl;

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

display(arr, n);

bubbleSort(arr, n);

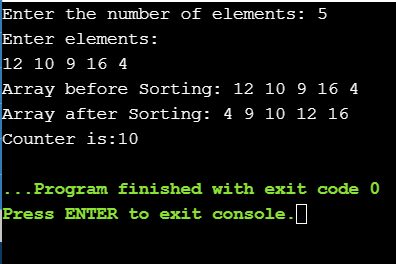
cout << "Array after Sorting: ";

display(arr, n);

cout<<"Counter is:"<<counter;

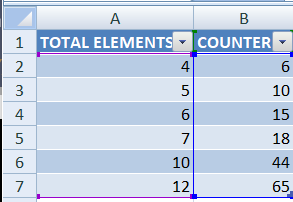
}

**OUTPUT:**

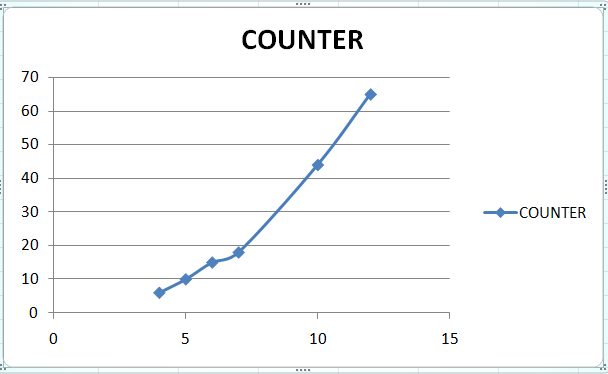
****

**ANALYSIS TABLE:**

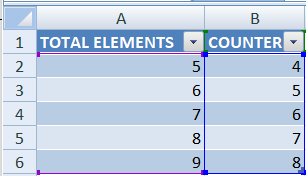
**WORST CASE:**

****

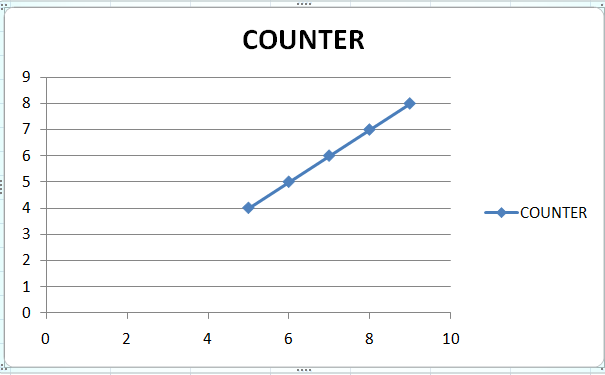
**GRAPH:**

****

**BEST CASE:**

****

**GRAPH:**

****

**CONCLUSION:** Here we concluded by implementing and analyzing the program of bubble sort,  O(n) for best case, O(n2) for average and worst case.