

Practical 1

Aim: - Implement and analyze algorithms given below:

1.1 Factorial (Iterative and recursive)

Program: -

```
#include
<iostream> using
namespace std;
int counter=1;
int fact(int
    no){
    if(no==1)
    {
        return 1;
        counter++;
    }
    else{

        counter++;
        return no*fact(no-1);
    }
}

int main()

{ int n;

    cout<<"Factorial using Recursive function" <<endl;
    cout<<"Enter the number :- ";
    cin>>n;

    cout<<"factorial of given number is
    "<<fact(n); cout<<"\nThe value of counter is
    :- "<< counter; return 0;
}
```

Output:-

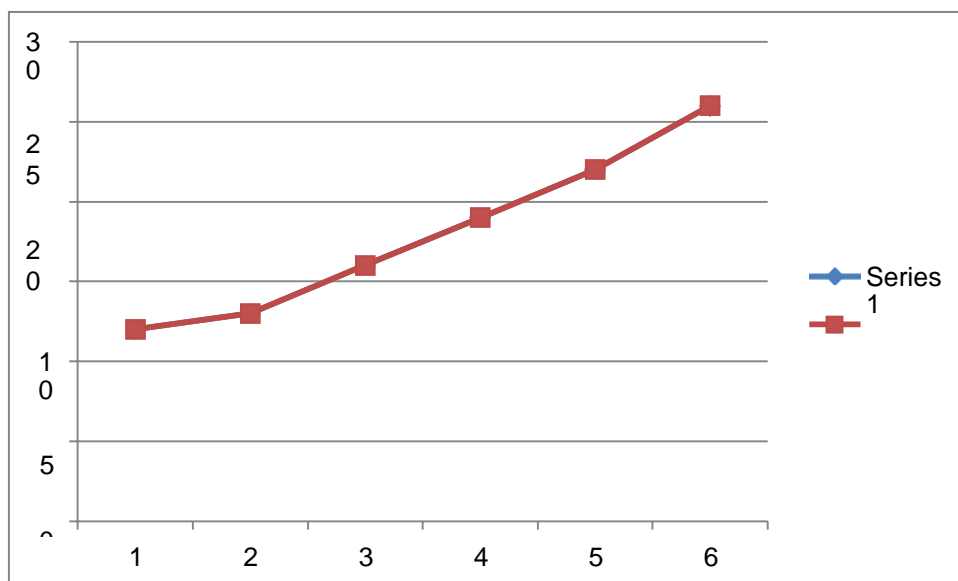
```

Factorial using Recursive function
Enter the number :- 5
factorial of given number is 120
The value of counter is :- 5

```

Analysis Table:-

input	output
12	12
15	15
26	26
17	17
20	20
25	25

Graph:

Aim: - Implement and analyze algorithms given below Factorial (**Iterative**).

Program: -

```
#include
<iostream> using
namespace std;
int counter=0;
int main()

{ int n,no=1;

    cout<<"Factorial using Iterative method" <<endl;
    cout<<"Enter the number :- ";
    cin>>n;

    for(int i=1;i<=n;++i){
        no*=i;
        counter++;
    }

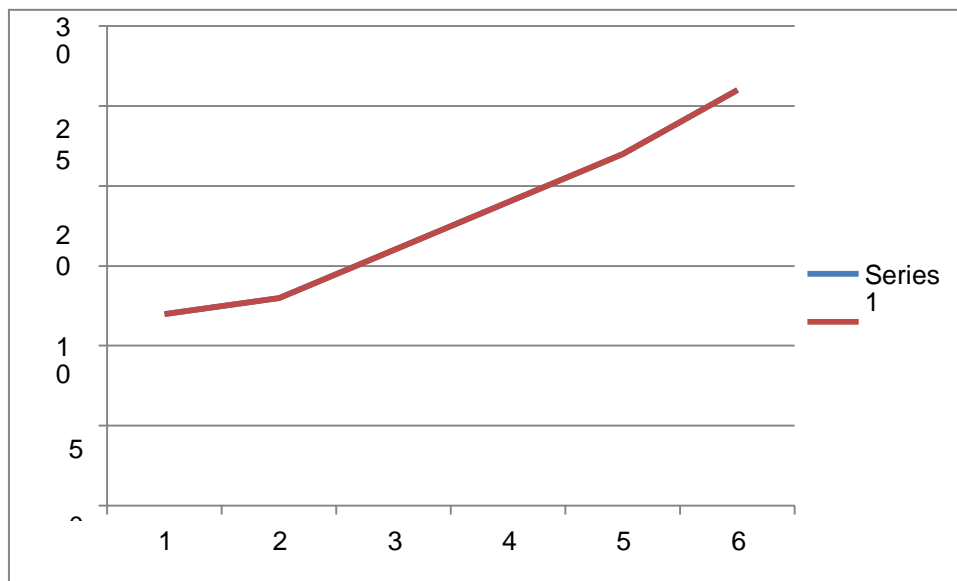
    cout<<"factorial of given number is "<<no;
    cout<<"\nThe value of counter is :- "<<
    counter; return 0;
}
```

Output:-

```
Factorial using Iterative method
Enter the number :- 6
factorial of given number is 720
The value of counter is :- 6
```

Analysis Table:-

input	output
12	12
13	13
16	16
19	19
22	22
26	26

Graph:**Conclusion:-**

From this practical, I have learnt how to analyze the algorithms using graphs.

1.2 Euclidean Algorithm

PROGRAM:-

```
#include<stdio.h>

int counter = 0;
int gcd(int
x,int y)
{
    while(y!=0)
    {
        counter
        ++; int temp
        = x%y;
        x
        = y; y
        =
        temp;
    }
    return x;
}

int main() {
    int n1,n2,result;
    printf("-----Euclidean Algorithm ----- \n");
    printf("Enter First
Integer: ");
    scanf("%d",&n1);
    printf("\nEnter Second Integer: ");
    scanf("%d",&n2);
    result = gcd(n1,n2);
    printf("\nGCD of %d and %d is
%d",n1,n2,result); printf("\nThe counter is
%d",counter);
}
```

OUTPUT:-

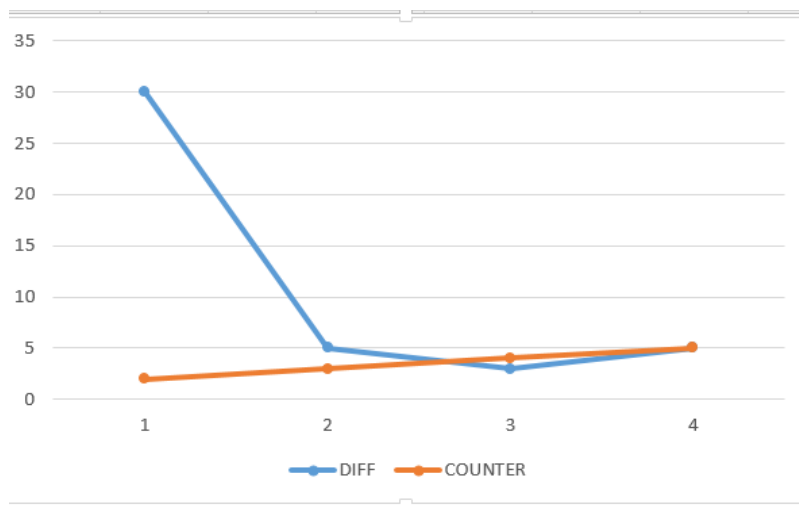
```
-----Euclidean Algorithm-----
Enter First Integer: 6

Enter Second Integer: 9

GCD of 6 and 9 is 3
The counter is 3
Process returned 0 (0x0)   execution time : 4.216 s
Press any key to continue.
```

ANALYSIS TABLE:-

Number 1	Number 2	Difference	Counter
3	33	30	2
4	9	5	3
4	7	3	4
7	12	5	5

GRAPH:-**CONCLUSION:-**

From this practical, I learnt how to code GCD for Euclidean algorithm and find its count it requires to find the solution. I was also able to do analysis and develop a graph for the same.

1.3 (1) Matrix Addition

PROGRAM:-

```
#include <iostream>
using namespace std;

int main()
{
    int r, c, a[100][100], b[100][100], sum[100][100], i, j, counter=0;

    cout << "Enter number of rows (between 1 and 100): ";
    cin >> r;

    cout << "Enter number of columns (between 1 and 100): ";
    cin >> c;

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

    // Storing elements of first matrix entered by user.
    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
        {
            cout << "Enter element a" << i + 1 << j + 1 << " : ";
            cin >> a[i][j];
        }

    // Storing elements of second matrix entered by user.
    cout << endl << "Enter elements of 2nd matrix: " << endl;
    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
        {
            cout << "Enter element b" << i + 1 << j + 1 << " : ";
            cin >> b[i][j];
        }

    // Adding Two matrices

    for(i = 0; i < r; ++i){
        for(j = 0; j < c; ++j){
            sum[i][j] = a[i][j] + b[i][j];
            counter++;
        }
    }
    cout << "\nCounter: " << counter;
```

```
// Displaying the resultant sum matrix.
cout << endl << "Sum of two matrix is: " << endl;
for(i = 0; i < r; ++i)
    for(j = 0; j < c; ++j)
    {
        cout << sum[i][j] << " ";
        if(j == c - 1)
            cout << endl;
    }

    return 0;
}
```

OUTPUT:-

```
Enter number of rows (between 1 and 100): 2
Enter number of columns (between 1 and 100): 2

Enter elements of 1st matrix:
Enter element a11 : 1
Enter element a12 : 2
Enter element a21 : 3
Enter element a22 : 4

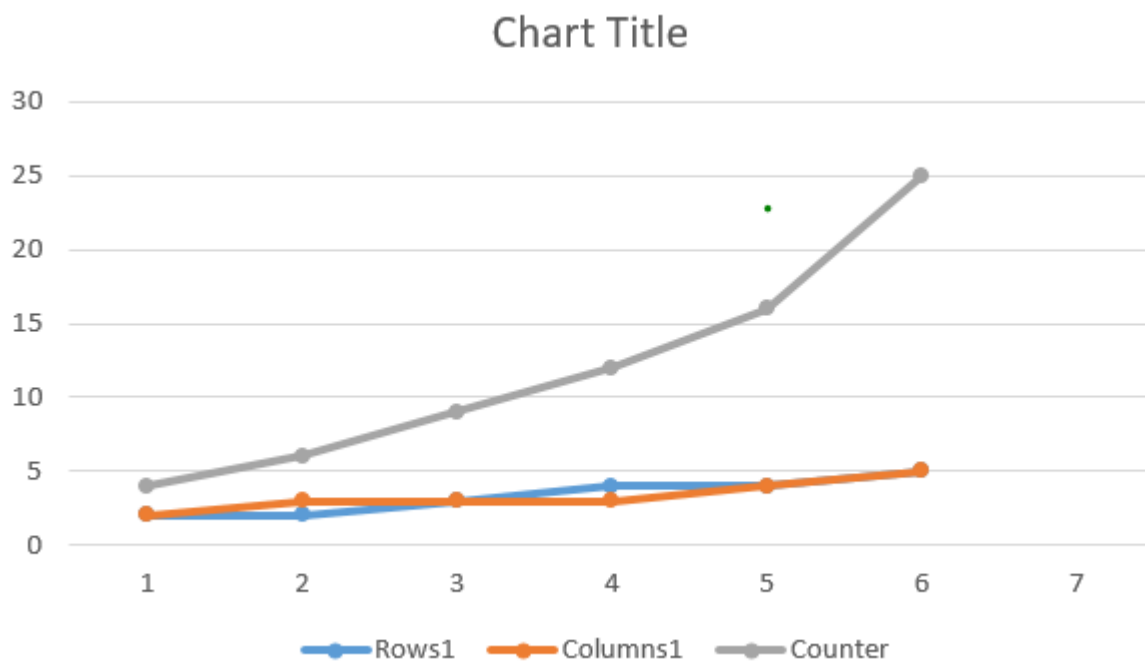
Enter elements of 2nd matrix:
Enter element b11 : 1
Enter element b12 : 2
Enter element b21 : 3
Enter element b22 : 4

Counter: 4
Sum of two matrix is:
2  4
6  8

Process returned 0 (0x0)   execution time : 4.538 s
Press any key to continue.
```


ANALYSIS TABLE:-

Rows1	Columns1	Rows2	Columns2	Counter
2	2	2	2	4
2	3	2	3	6
3	3	3	3	9
4	3	4	3	12
4	4	4	4	16
5	5	5	5	25

GRAPH:-**CONCLUSION:-**

From this practical, I learnt how to code matrix addition algorithm and find its count it requires to find the solution. I was also able to do analysis and develop a graph for the same.

(2) Matrix Multiplication

PROGRAM:-

```
#include <iostream>
using namespace std;

int main()
{
    int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k, 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 is 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;
    }

// Multiplying matrix a and b and storing in array mult.
for(i = 0; i < r1; ++i){
    for(j = 0; j < c2; ++j){
        for(k = 0; k < c1; ++k){
            mult[i][j] += a[i][k] * b[k][j];
            counter++;
        }
    }
}
cout<<"\nCounter:"<<counter;

// Displaying the multiplication of two matrix.
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;
}
return 0;
}
```

OUTPUT:-

```

Enter rows and columns for first matrix: 2
2
Enter rows and columns for second matrix: 2
3

Enter elements of matrix 1:
Enter element a11 : 1
Enter element a12 : 2
Enter element a21 : 3
Enter element a22 : 4

Enter elements of matrix 2:
Enter element b11 : 1
Enter element b12 : 2
Enter element b13 : 3
Enter element b21 : 4
Enter element b22 : 5
Enter element b23 : 6

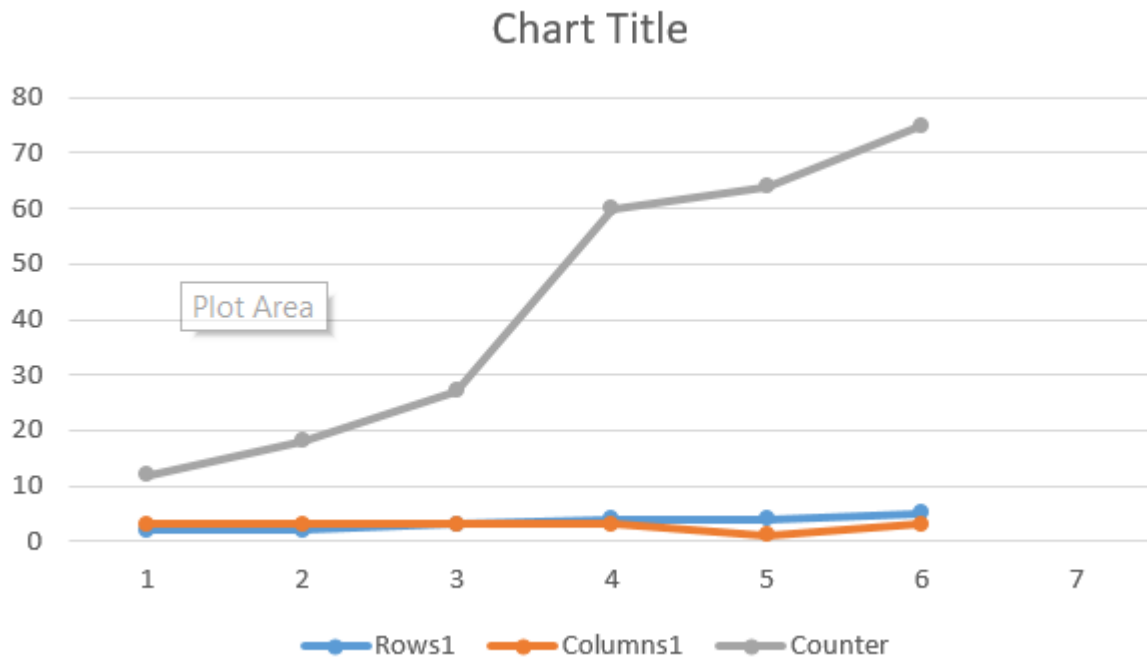
Counter:12
Output Matrix:
 9 12 15
19 26 33

Process returned 0 (0x0)   execution time : 6.340 s
Press any key to continue.

```

ANALYSIS TABLE:-

Rows1	Columns1	Rows2	Columns2	Counter
2	3	3	2	12
2	3	3	3	18
3	3	3	3	27
4	3	3	5	60
4	1	1	16	64
5	3	3	5	75

GRAPH:-**CONCLUSION:-**

From this practical, I learnt how to code matrix multiplication algorithm and find its count it requires to find the solution. I was also able to do analysis and develop a graph for the same.

1.4 (1) Recursive Linear Search

PROGRAM:-

```
#include<iostream>
using namespace std;
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];
    cout<<"\nEnter the total elements in the array : ";
    cin>>n;

    cout<<"\nEnter the array elements : \n";
    for (int i = 0; i < n; i++)
        cin>>arr[i];

    cout<<"\nEnter the element to search : ";
    cin>>value;

    pos = RecursiveLS(arr, value, 0, n);
    if (pos != 0)
        cout<<"\nElement found at pos "<< pos;
    else
        cout<<"\nElement not found";

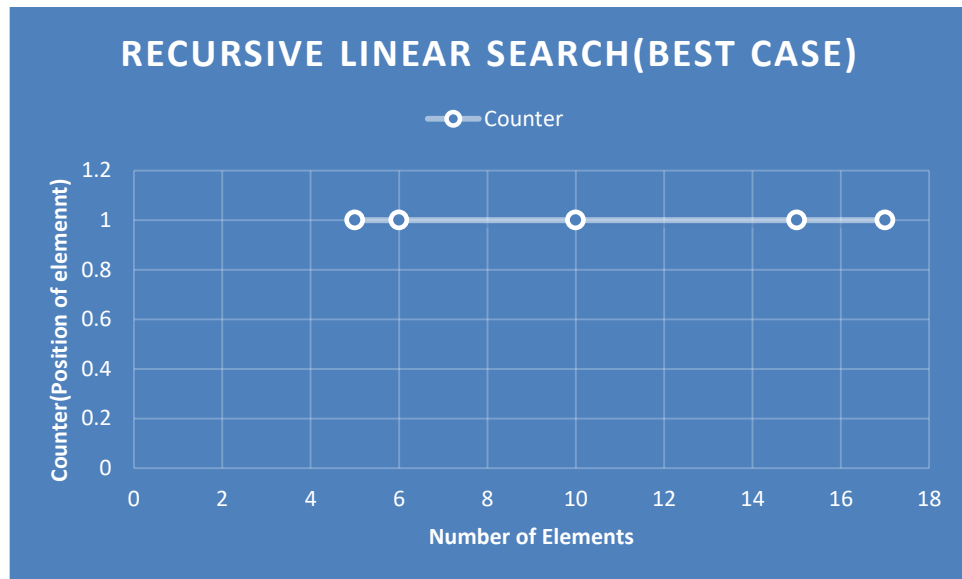
    cout<<"\nCounter : "<<counter;
    return 0;
}
```

OUTPUT:-

```
Enter the total elements in the array : 5
Enter the array elements :
12
14
3
16
9
Enter the element to search :3
Element found at pos 3
Counter : 3
Process returned 0 (0x0)   execution time : 36.710 s
Press any key to continue.
```

(Best Case)**ANALYSIS TABLE:**

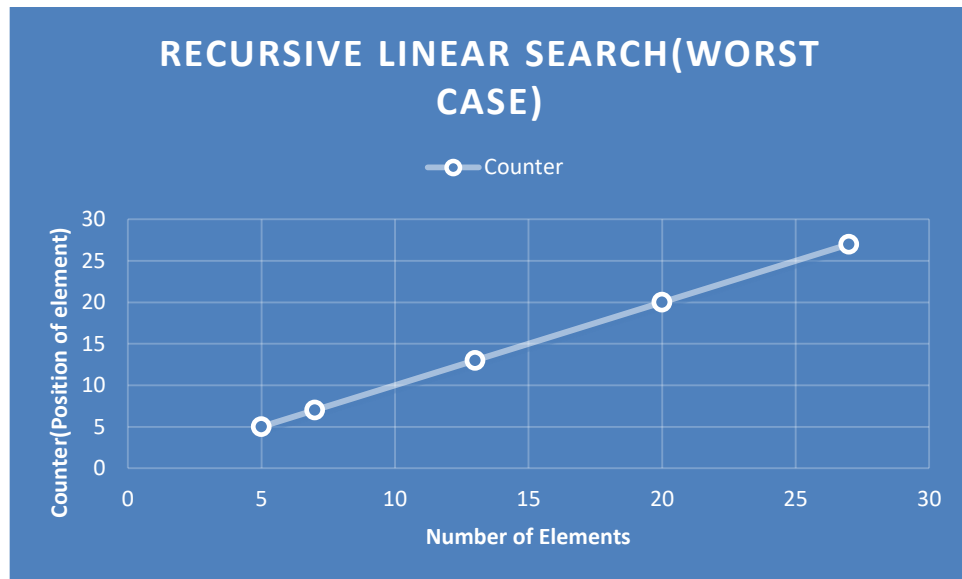
Total Elements	Pos of element to be searched	Counter
5	1	1
6	1	1
10	1	1
15	1	1
17	1	1

GRAPH:

(Worst case)

ANALYSIS TABLE:

Total Elements	Pos of element to be searched	Counter
5	5	5
7	7	7
13	13	13
20	20	20
27	27	27

GRAPH:

1.4 (2) Binary Search

PROGRAM:-

```
#include<iostream>
using namespace std;
int counter=0;
int binarySearch(int A[], int low, int high, int x)
{
    counter++;
    if (low > high)
        return -1;

    int mid = (low + high)/2;
    // int mid = low + (high - low)/2;
    if (x == A[mid])
        return mid;
    else if (x < A[mid])
        return binarySearch(A, low, mid - 1, x);
    else
        return binarySearch(A, mid + 1, high, x);
}

int main()
{
    int nums,target;
    cout<<"\nEnter the total elements in the array : ";
    cin>>nums;

    int arr[nums];
    cout<<"\nEnter the array elements : \n";
    for (int i = 0; i < nums; i++)
        cin>>arr[i];

    cout<<"\nEnter the element to search ";
    cin>>target;

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

    int low = 0, high = n - 1;
    int index = binarySearch(arr, low, high, target);

    if (index != -1)
        cout<<"\nElement found at index "<<index;
    else
        cout<<"\nElement not found in the array";
}
```

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

    return 0;
}
```

OUTPUT:-

```
Enter the total elements in the array : 5

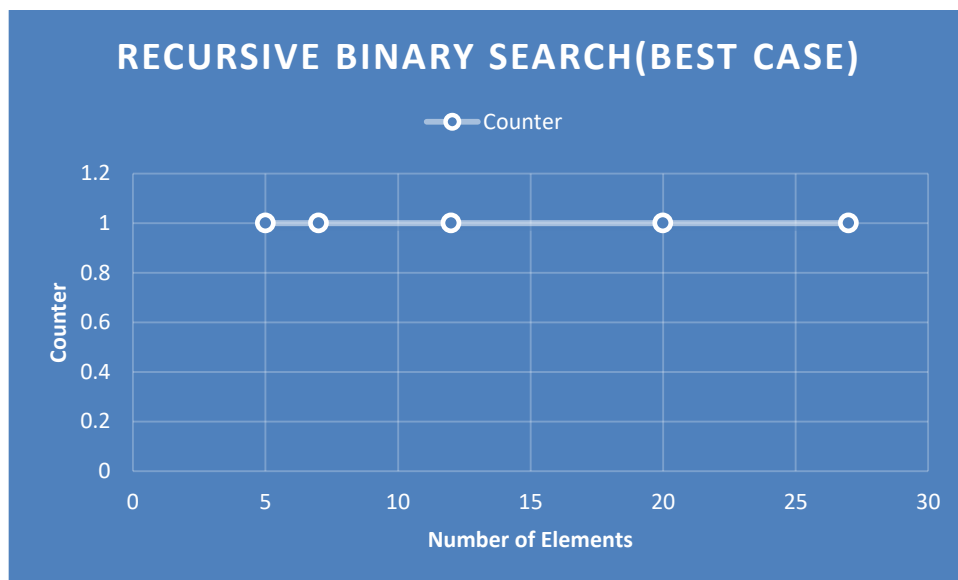
Enter the array elements :
35
14
11
6
21

Enter the element to search  21

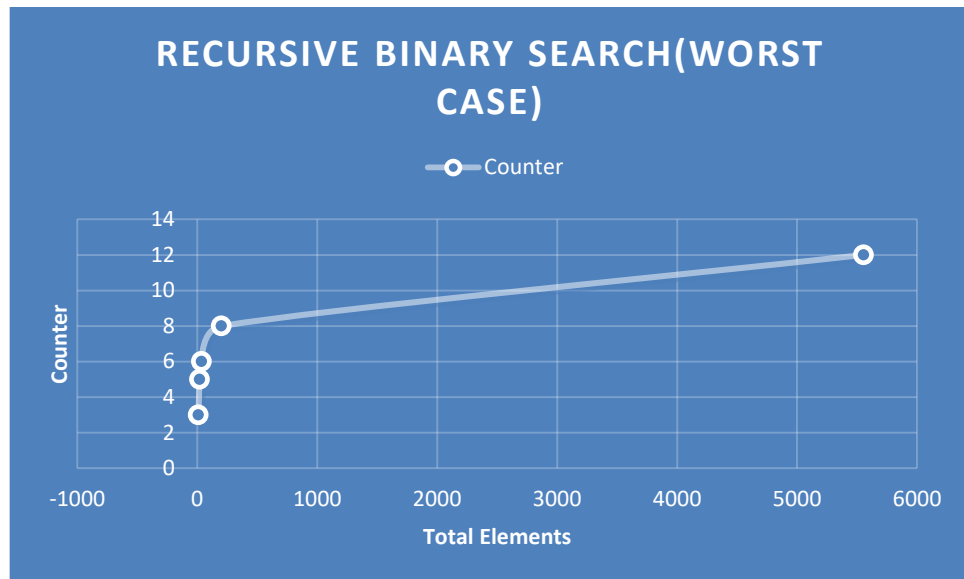
Element found at index 4
Counter : 3
Process returned 0 (0x0)   execution time : 18.418 s
Press any key to continue.
```

(Best Case)**ANALYSIS TABLE:**

Total Elements	Pos of element to be searched	Counter
5	3	1
7	4	1
12	6	1
20	10	1
27	13	1

GRAPH:**(Worst Case)****ANALYSIS TABLE:**

Total Elements	Pos of element to be searched	Counter
10	1	3
20	20	5
35	35	6
200	200	8
5555	1	12

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 $O(\log n)$ 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.

1.5

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.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int s[10],d,n,set[10],count=0;
void Display(int);
int flag=0, Counter=0;
void main()
{
    int subset(int,int);

    int i;

    printf("Enter the number of elements in set : \n");
    scanf("%d",&n);

    printf("Enter the set values : \n");
    for(i=0; i<n; ++i)
        scanf("%d",&s[i]);

    printf("Enter the sum : \n");
    scanf("%d",&d);

    printf("The program output is : \n");
    subset(0,0);

    if(flag==0)
        printf("there is no solution");

    printf("\nCounter : %d",Counter);
    getch();
}
int subset(int sum,int i)
{
    if(sum==d)
    {
        flag=1;
        Display(count);
        return;
    }
}
```

```
    }
    else if(sum>d||i>=n)
    {
        return;
    }
    else
    {
        set[count]=s[i];
        count++;
        subset(sum+s[i],i+1);
        count--;
        subset(sum,i+1);
    }
    Counter++;
}

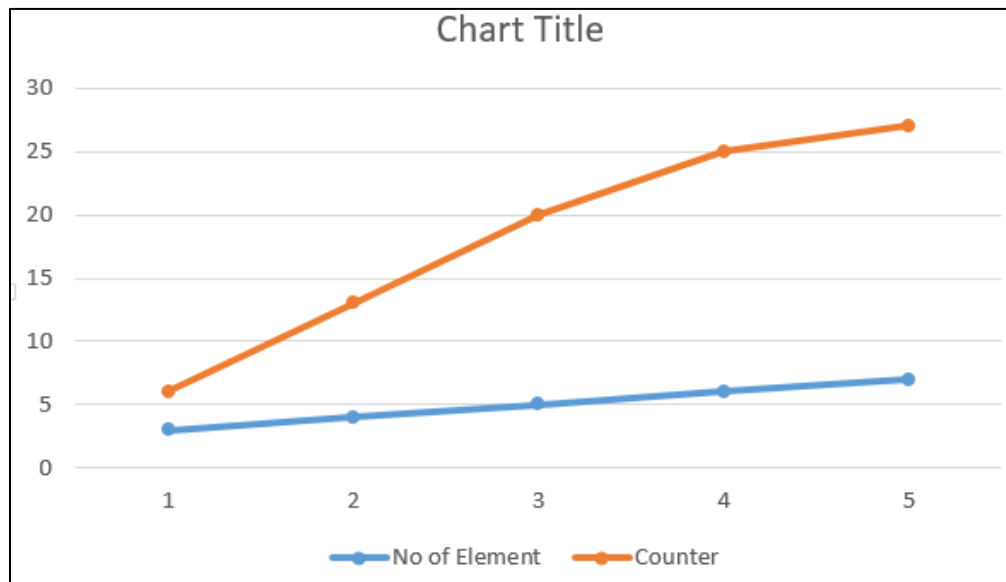
void Display(int count)
{
    int i;
    printf("{");
    for(i=0; i<count; i++)
        printf(" %d ",set[i]);
    printf("}");
}
```

OUTPUT:

```
Enter the number of elements in set :
5
Enter the set values :
1
3
4
7
10
Enter the sum :
8
The program output is :
{ 1 3 4 }{ 1 7 }
Counter : 22
Process returned 96 (0x60)   execution time : 26.000 s
Press any key to continue.
```

ANALYSIS TABLE:

No of Element	Counter
3	6
4	13
5	20
6	25
7	27

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

AIM: Implement and analyse algorithms given below:

2.1) Bubble Sort

PROGRAM:

```
#include<iostream>
using namespace std;
int main()
{
    int siz,counter=0;
    cout<<"Enter the size of array:- ";
    cin>>siz;
    int arr[siz];
    cout<<"\n Enter the elements of array :- ";
    for(int i=0; i<siz; ++i)
    {
        cin>>arr[i];
    }

    for(int i=0; i<siz-1; ++i)
    {
        int temp=0;
        for(int j=0; j<siz-i-1; ++j)
        {
            if(arr[j]>arr[j+1])
            {
                counter++;
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }

    cout<<"\n Sorted array is : - ";
    for(int i=0; i<siz; ++i)
    {
        cout<<arr[i]<<" ";
    }

    cout<<"\n The value of counter is :- "<<counter;
    return 0;
}
```

OUTPUT:

```

Enter the size of array:- 5

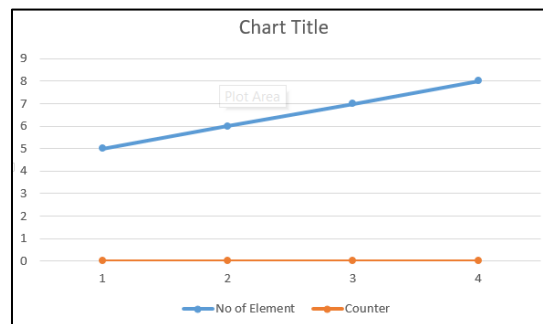
Enter the elements of array :- 3
2
4
5
1

Sorted array is : - 1 2 3 4 5
The value of counter is :- 5
Process returned 0 (0x0)   execution time : 5.105 s
Press any key to continue.

```

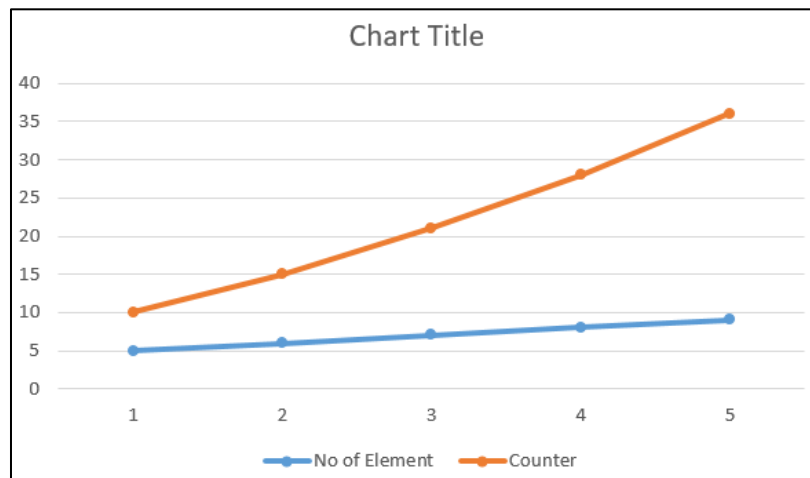
1) Best Case:**ANALYSIS TABLE:**

No of Element	Counter
5	0
6	0
7	0
8	0

GRAPH:

2) Worst Case:**ANALYSIS TABLE:**

No of Element	Counter
5	10
6	15
7	21
8	28
9	36

GRAPH:**CONCLUSION:**

Here we concluded by implementing and analyzing the program of bubble sort, the time complexity of algorithm by analyzing the graph is constant for best case i.e. 0 but for worst case it increases parabolically.

2.2

PROGRAM:

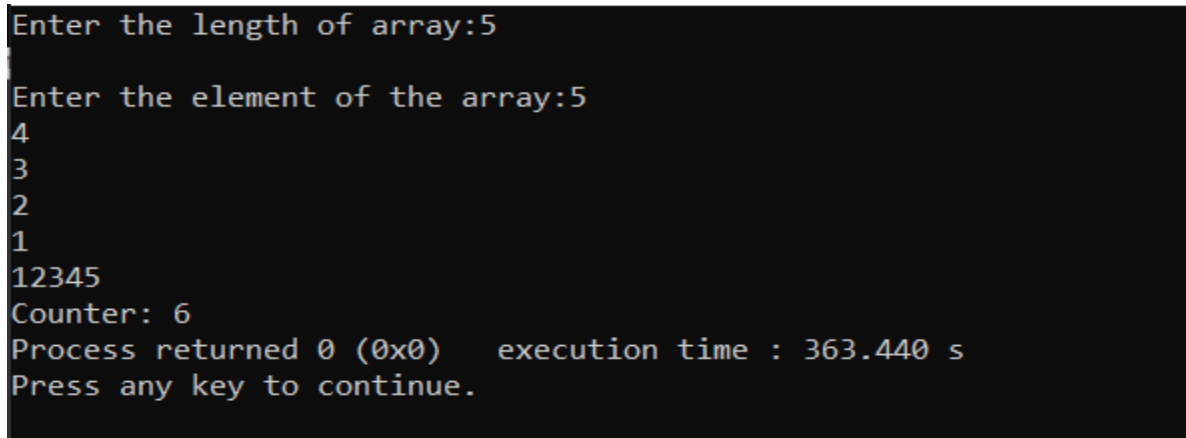
```
#include<iostream>
using namespace std;
int main(){
    int n,i,minimum,j,temp,c = 0;
    cout<<"Enter the length of array:";
    cin>>n;

    int arr[n];
    cout<<"\nEnter the element of the array:";
    for(i=0; i<n; i++){
        cin>>arr[i];
    }

    for(i=0;i<=n-1;i++){
        minimum = i;
        for(j = i+1;j<=n;j++){
            if(arr[j]<arr[minimum]){
                minimum = j;
                c++;
            }
        }
        temp = arr[minimum];
        arr[minimum] = arr[i];
        arr[i] = temp;
        cout<<arr[i];

    }
    cout<<"\nCounter: "<< c;
}
```

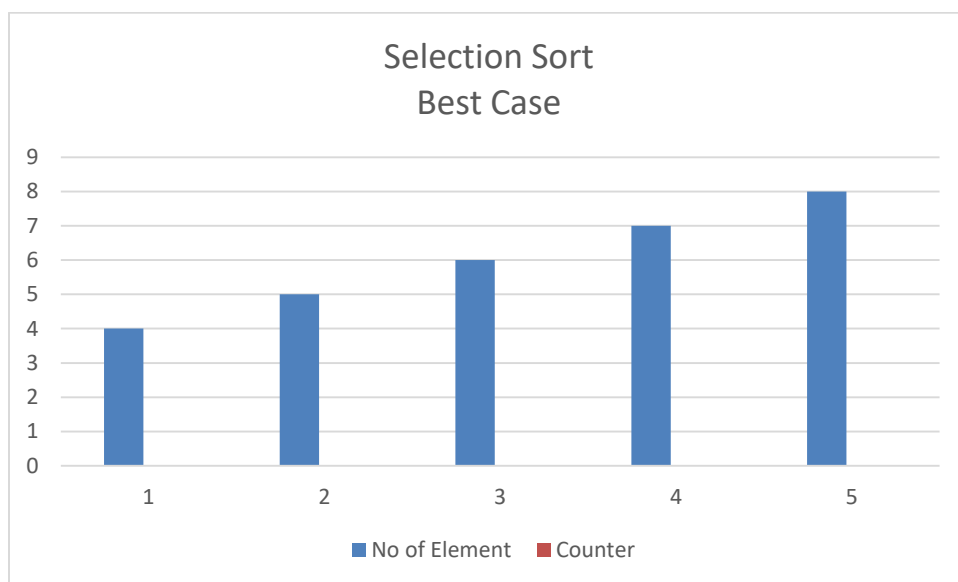
OUTPUT:



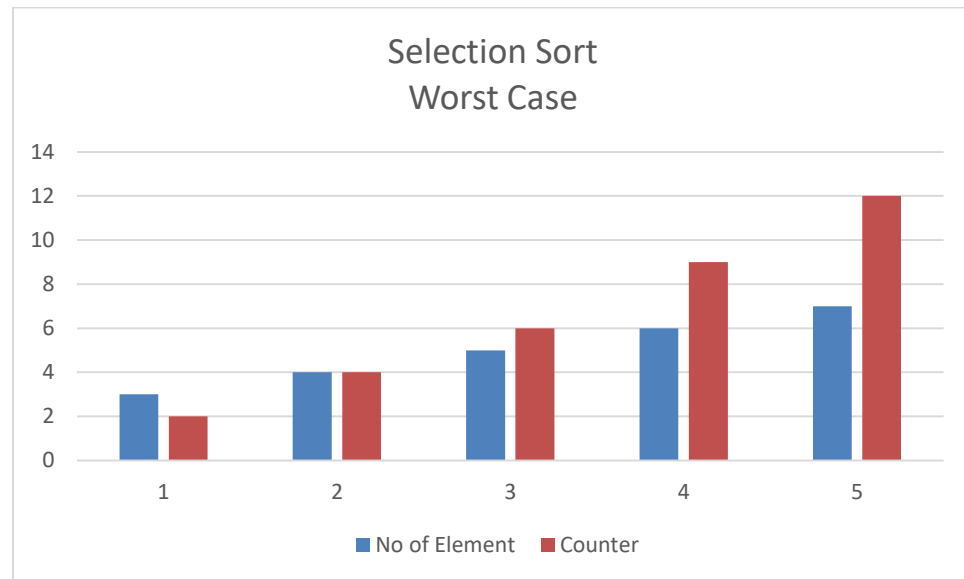
```
Enter the length of array:5
Enter the element of the array:5
4
3
2
1
12345
Counter: 6
Process returned 0 (0x0)    execution time : 363.440 s
Press any key to continue.
```

1) Best Case:**ANALYSIS TABLE:**

No of Element	Counter
4	0
5	0
6	0
7	0
8	0

GRAPH:**2) Worst Case:****ANALYSIS TABLE:**

No of Element	Counter
3	2
4	4
5	6
6	9
7	12

GRAPH:**CONCLUSION:**

Here I have learnt about analysis of selection Sort for the best case and worst case.

2.3 Insertion Sort

PROGRAM:

```
#include<iostream>
using namespace std;
int main(){
    int n,i,key,j,c = 0;
    cout<<"Enter the length of array:";
    cin>>n;

    int arr[n];
    cout<<"\nEnter the element of the array:";
    for(i=0; i<n; i++){
        cin>>arr[i];
    }

    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;
            c++;
        }
        arr[j + 1] = key;
    }
    for (i = 0; i < n; i++){
        cout << arr[i] << " ";
    }
    cout<<"\nCounter: "<<c;
}
```

OUTPUT:

```

Enter the length of array:5

Enter the element of the array:5
4
3
2
1
1 2 3 4 5
Counter: 10
Process returned 0 (0x0)   execution time : 8.615 s
Press any key to continue.

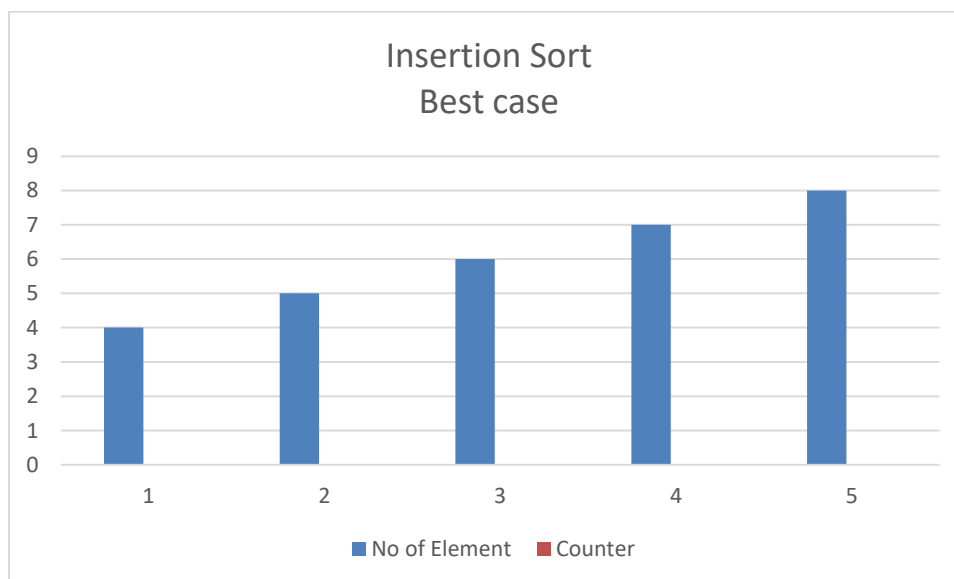
```

1) Best Case:

ANALYSIS TABLE:

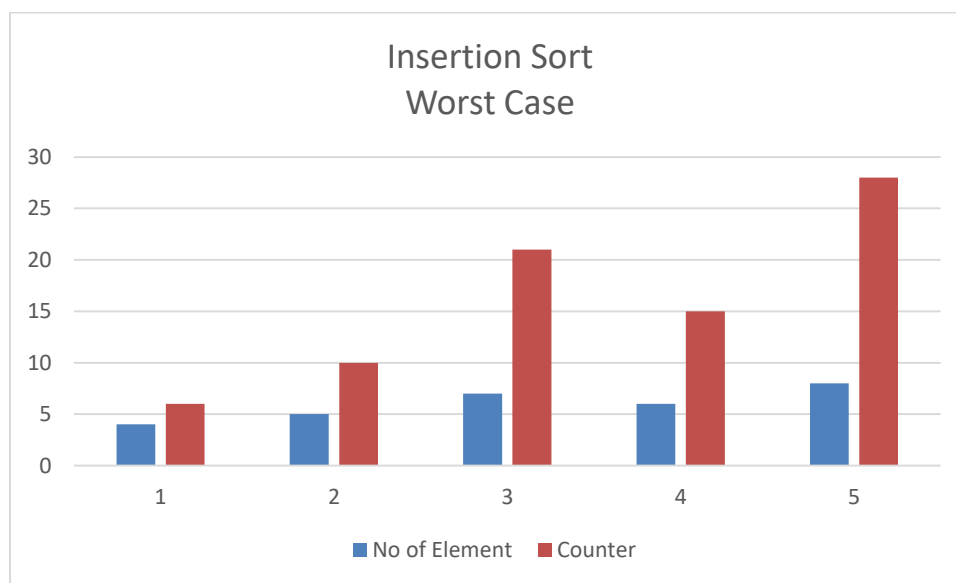
No of Element	Counter
4	0
5	0
6	0
7	0
8	0

GRAPH:



2) Worst Case:**ANALYSIS TABLE:**

No of Element	Counter
4	6
5	10
7	21
6	15
8	28

GRAPH:**CONCLUSION:**

Here I have learnt about analysis of Insertion Sort for the best case and worst case.

Practical 3.1

AIM: Implement and perform analysis of worst case of Merge Sort and Quick sort. Compare both algorithms.

1) MERGE SORT:

PROGRAM:

```
import java.util.*;
class Mergesort
{
    static int counter=0;
    void merge(int arr[],int l,int r,int m)
    {
        int i=0,j=0,k=l,l1=m-l+1,l2=r-m;
        int left[]=new int[l1];
        int right[]=new int[l2];
        for(int c=0;c<l1;c++){ left[c]=arr[l+c];}
        for(int c=0;c<l2;c++){ right[c]=arr[m+1+c];}
        while(i<l1 && j<l2)
        {
            counter++;
            if(left[i]<=right[j])
            {
                arr[k]=left[i];
                i++;
                k++;
            }
            else
            {
                arr[k]=right[j];
                j++;
                k++;
            }
        }
        while (i < l1)
        {
            counter++;
            arr[k] = left[i];
            i++;
        }
    }
}
```

```

        k++;
    }
    while (j < l2)
    {
        counter++;
        arr[k] = right[j];
        j++;
        k++;
    }
}

void mergesrt(int arr[],int l,int r)
{
    int m;
    if(r==l)
        return;
    else
    {
        m=(l+r)/2;
        mergesrt(arr,l,m);
        mergesrt(arr,m+1,r);
        merge(arr,l,r,m);
    }
}

}

public static void main(String args[])
{
    Mergesort m1=new Mergesort ();
    int n,A[];
    Scanner s=new Scanner(System.in);
    System.out.print("Enter no of elements: ");
    n=s.nextInt();
    A=new int[n];
    System.out.print("Enter the elements: ");
    for(int i=0;i<n;i++){
        A[i]=s.nextInt();
    }
    m1.mergesrt(A,0,n-1);
    System.out.println("\nSorted Array is:");
    for(int i=0;i<n;i++)
    {
        System.out.print(A[i]+" ");
    }
    System.out.println("\nValue of counter = "+ Mergesort.counter);
}
}

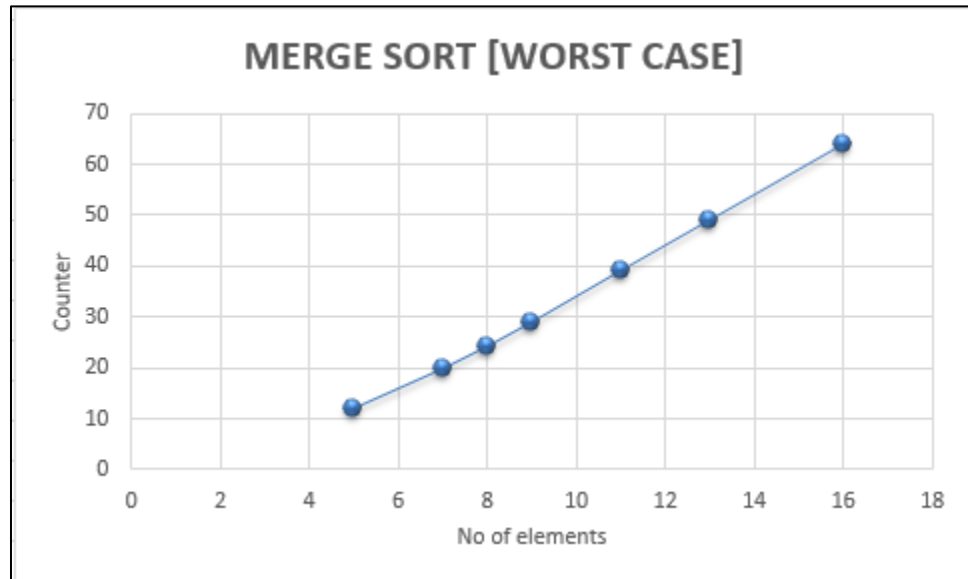
```

OUTPUT:

```
C:\Users\Dp\Desktop>javac mergesort.java
C:\Users\Dp\Desktop>java Mergesort
Enter no of elements: 5
Enter the elements: 5
4
3
2
1
Sorted Array is:
1 2 3 4 5
Value of counter = 12
```

ANALYSIS TABLE:

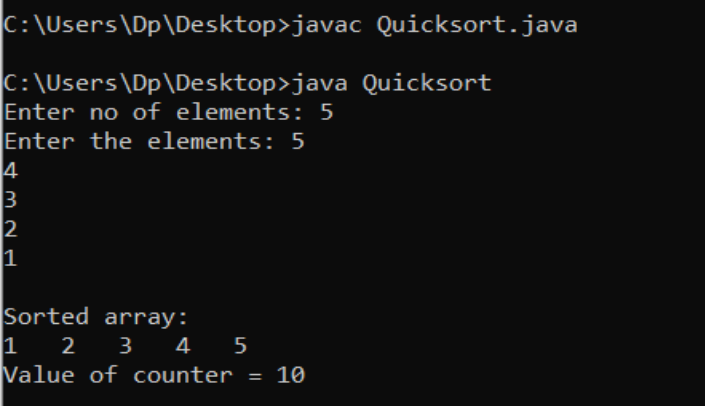
WORST CASE	
No of elements	Counter
5	12
7	20
8	24
9	29
11	39
13	49
16	64

GRAPH:**2) QUICK SORT:****PROGRAM:**

```
import java.util.*;
class Quicksort
{
    static int counter=0;
    int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1);
        for (int j=low; j<high; j++)
        {
            counter++;
            if (arr[j] < pivot)
            {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;
        return i+1;
    }
}
```

```
void sort(int arr[], int low, int high)
{
    if (low < high)
    {
        int p = partition(arr, low, high);
        sort(arr, low, p-1);
        sort(arr, p+1, high);
    }
}

public static void main(String ae[])
{
    int n,A[];
    Scanner s=new Scanner(System.in);
    System.out.print("Enter no of elements: ");
    n=s.nextInt();
    A=new int[n];
    System.out.print("Enter the elements: ");
    for(int i=0;i<n;i++){ A[i]=s.nextInt();}
    Quicksort a1=new Quicksort ();
    a1.sort(A,0,n-1);
    System.out.println("\nSorted array: ");
    for(int i=0;i<n;i++)
    {System.out.print(A[i]+" "); }
    System.out.println("\nValue of counter = "+Quicksort.counter);
}
}
```

OUTPUT:

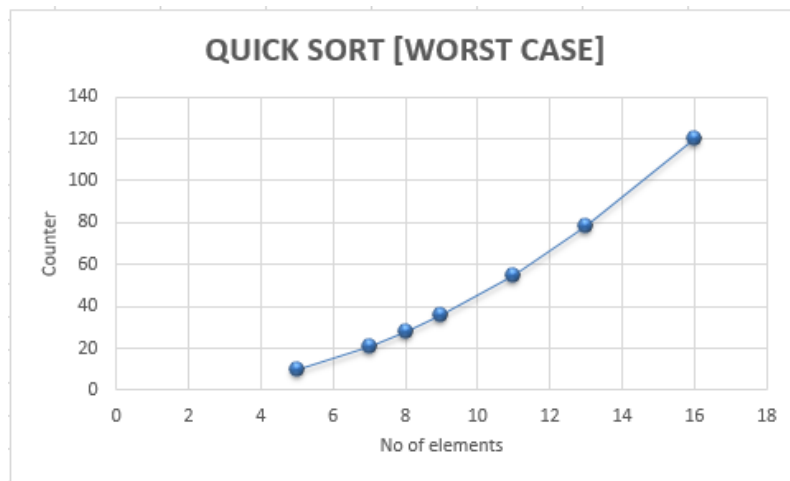
```
C:\Users\Dp\Desktop>javac Quicksort.java

C:\Users\Dp\Desktop>java Quicksort
Enter no of elements: 5
Enter the elements: 5
4
3
2
1

Sorted array:
1 2 3 4 5
Value of counter = 10
```

ANALYSIS TABLE:

WORST CASE	
No of elements	Counter
5	10
7	21
8	28
9	36
11	55
13	78
16	120

GRAPH:**CONCLUSION:**

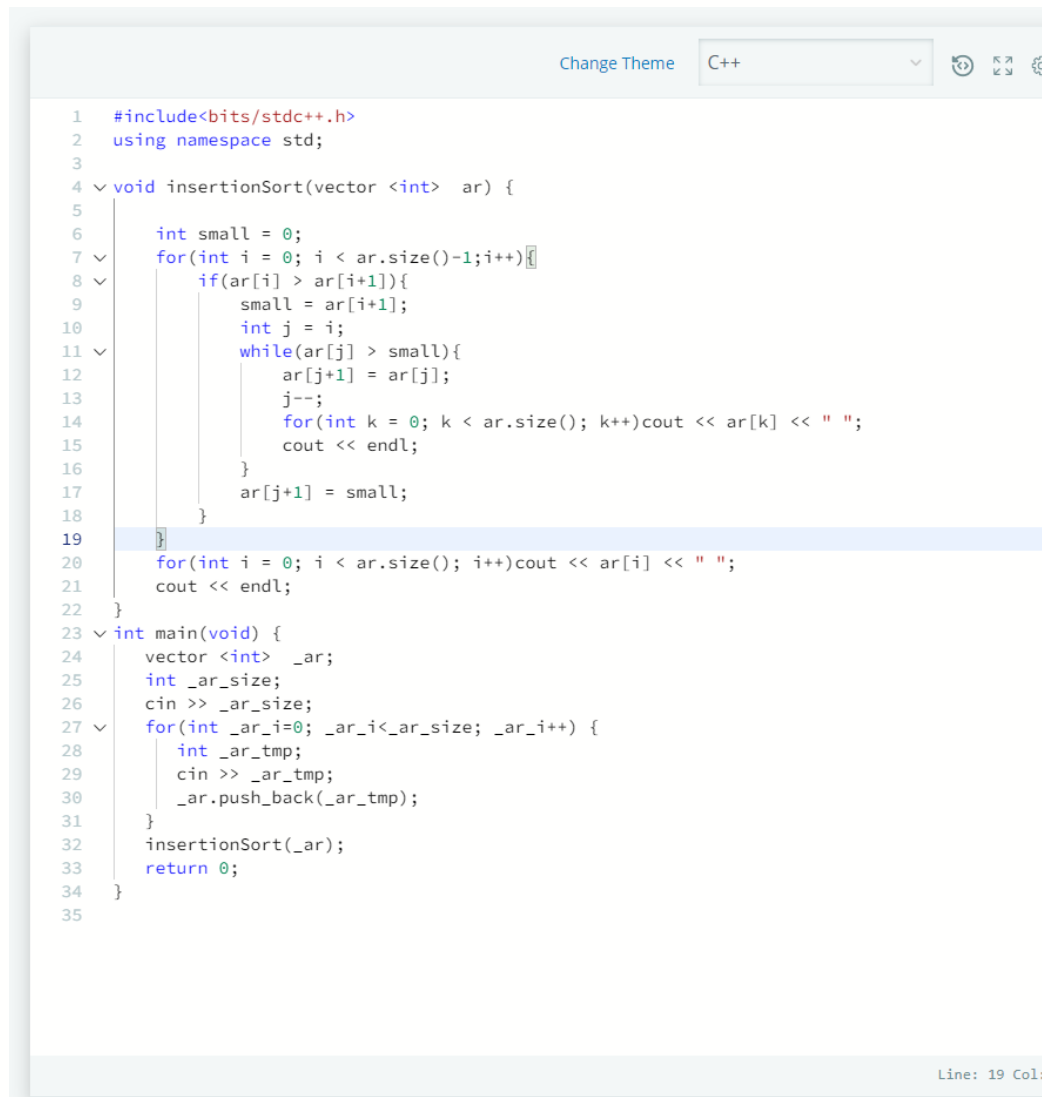
From this practical we analyzed the worst case of **Merge sort** and **Quick sort** algorithm. The graph was non-linear and counter is increasing with the number of elements for both the algorithms.

Hacker Rank : INSERTION SORT

PART –I

AIM: Given a sorted list with an unsorted number in the rightmost cell, can you write some simple code to insert into the array so that it remains sorted.

PROGRAM:



```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  void insertionSort(vector<int> ar) {
5
6      int small = 0;
7      for(int i = 0; i < ar.size()-1; i++){
8          if(ar[i] > ar[i+1]){
9              small = ar[i+1];
10             int j = i;
11             while(ar[j] > small){
12                 ar[j+1] = ar[j];
13                 j--;
14                 for(int k = 0; k < ar.size(); k++)cout << ar[k] << " ";
15                 cout << endl;
16             }
17             ar[j+1] = small;
18         }
19     }
20     for(int i = 0; i < ar.size(); i++)cout << ar[i] << " ";
21     cout << endl;
22 }
23 int main(void) {
24     vector<int> _ar;
25     int _ar_size;
26     cin >> _ar_size;
27     for(int _ar_i=0; _ar_i<_ar_size; _ar_i++) {
28         int _ar_tmp;
29         cin >> _ar_tmp;
30         _ar.push_back(_ar_tmp);
31     }
32     insertionSort(_ar);
33     return 0;
34 }
35
```

Line: 19 Col:

OUTPUT:

The screenshot shows a web browser window with the URL `hackerrank.com/challenges/insertionsort1/problem`. The page features a 'Congratulations!' message and a 'Sample Test case 0' window. The window displays the input, the user's output, and the expected output, all of which match.

Input (stdin)

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

Your Output (stdout)

```
1 2 4 6 8 8
2 2 4 6 6 8
3 2 4 4 6 8
4 2 3 4 6 8
```

Expected Output

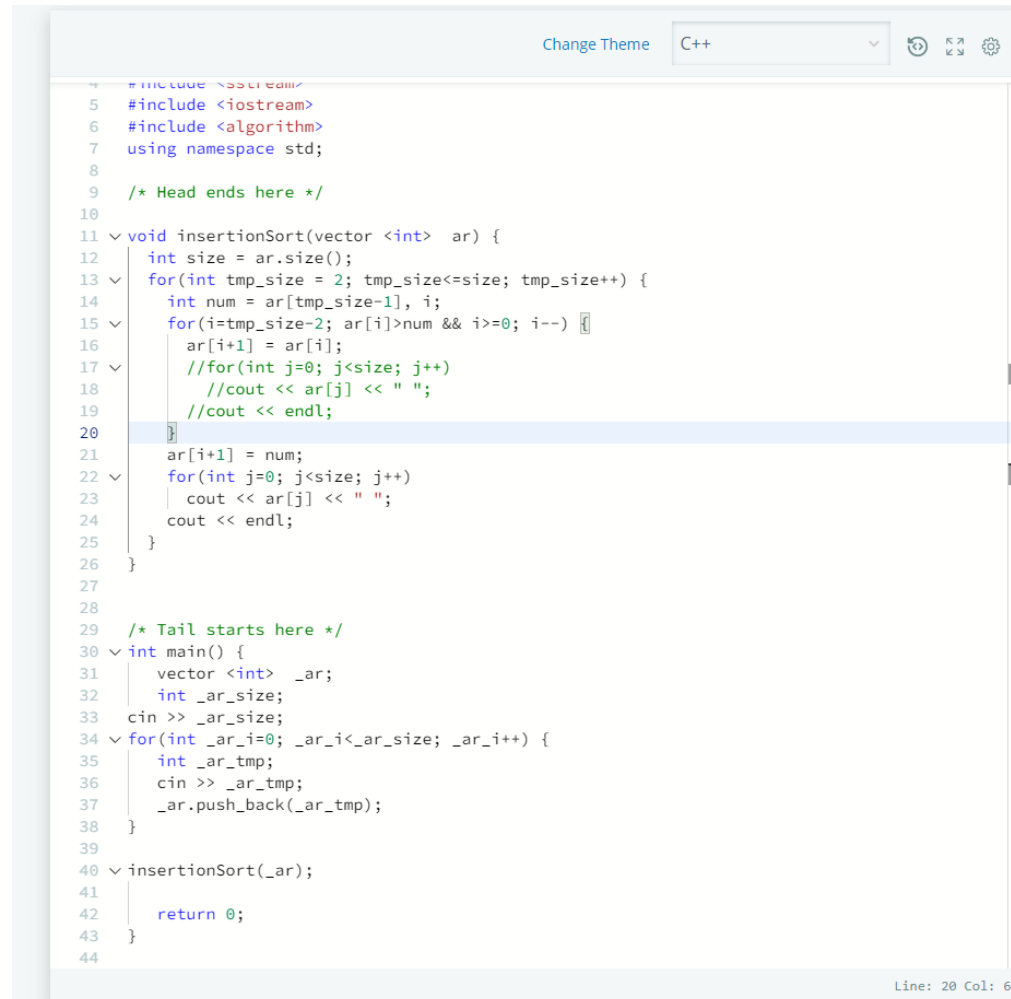
```
1 2 4 6 8 8
2 2 4 6 6 8
```

Hacker Rank : INSERTION SORT

PART –II

AIM: In Insertion Sort Part 1, you inserted one element into an array at its correct sorted position. Using the same approach repeatedly, can you sort an entire array?

PROGRAM:

A screenshot of a C++ code editor window. The editor has a light blue header bar with 'Change Theme' and 'C++' dropdown menus. The code is written in C++ and implements an Insertion Sort algorithm. It includes headers for <sstream>, <iostream>, and <algorithm>, and uses the std namespace. The insertionSort function takes a vector of integers and sorts it by iterating through each element and inserting it into its correct position in a temporary sorted subarray. The main function reads the size of the array, the array elements, and calls the insertionSort function. The status bar at the bottom right shows 'Line: 20 Col: 6'.

```
1  #include <sstream>
2  #include <iostream>
3  #include <algorithm>
4  using namespace std;
5
6  /* Head ends here */
7
8  void insertionSort(vector<int> ar) {
9      int size = ar.size();
10     for(int tmp_size = 2; tmp_size<=size; tmp_size++) {
11         int num = ar[tmp_size-1], i;
12         for(i=tmp_size-2; ar[i]>num && i>=0; i--) {
13             ar[i+1] = ar[i];
14             //for(int j=0; j<size; j++)
15             //cout << ar[j] << " ";
16             //cout << endl;
17         }
18         ar[i+1] = num;
19         for(int j=0; j<size; j++)
20             cout << ar[j] << " ";
21         cout << endl;
22     }
23 }
24
25 /* Tail starts here */
26 int main() {
27     vector<int> _ar;
28     int _ar_size;
29     cin >> _ar_size;
30     for(int _ar_i=0; _ar_i<_ar_size; _ar_i++) {
31         int _ar_tmp;
32         cin >> _ar_tmp;
33         _ar.push_back(_ar_tmp);
34     }
35     insertionSort(_ar);
36     return 0;
37 }
```

OUTPUT:

Practical 2.2 CE2_21/8/2020 AIM: x Insertion Sort - Part 2 | HackerRank x

hackerrank.com/challenges/insertionsort2/problem

Upload Code as File ☐ Test against custom input [Run Code](#) [Submit Code](#)

Congratulations!
You have passed the sample test cases. Click the submit button to run your code against all the test cases.

✓ **Sample Test case 0**

✓ **Sample Test case 1**

Input (stdin) [Download](#)

1	6
2	1 4 3 5 6 2

Your Output (stdout)

1	1 4 3 5 6 2
2	1 3 4 5 6 2
3	1 3 4 5 6 2
4	1 3 4 5 6 2
5	1 2 3 4 5 6

Expected Output [Download](#)

1	1 4 3 5 6 2
---	-------------

[Contest Calendar](#) | [Blog](#) | [Scoring](#) | [Environment](#) | [FAQ](#) | [About Us](#) | [Support](#) | [Careers](#) | [Terms Of Service](#) | [Privacy Policy](#) | [Request a Feature](#)

Practical 3.2

AIM: Implement the program to find X^Y using divide and conquer strategy and print the total number of multiplications required to find X^Y . Test the program for following test cases:

Test	X2	Y2
1	2	6
2	7	25
3	5	34

PROGRAM:

```
#include<iostream>
using namespace std;

int counter=0;

long int power(unsigned long long int x,unsigned long long int y)
{
    counter++;
    if(y==0)
        return 1;
    else if(y%2==0)
        return power(x,y/2)*power(x,y/2);
    else
        return x*power(x,y/2)*power(x,y/2);
}

int main()
{
    unsigned long long int x,y,result;
    cout<<"X to the power Y....!!\n";
    cout<<"\nEnter the value of x : ";
    cin>>x;
    cout<<"Enter the value of y : ";
    cin>>y;

    result=power(x,y);
    cout<<"\n\nResult : "<<result<<endl;
```

```
cout<<"\n\nCounter : "<<counter<<endl;

return 0;
}
```

OUTPUT:

```
X to the power Y....!!

Enter the value of x : 5
Enter the value of y : 34

Result : 8046632842880574361

Counter : 127
```

```
X to the power Y....!!

Enter the value of x : 2
Enter the value of y : 6

Result : 64

Counter : 15
```

```
X to the power Y....!!

Enter the value of x : 7
Enter the value of y : 25

Result : 12903046356877184455

Counter : 63
```

CONCLUSION:

In this practical we have learnt the basics of divide and conquer strategy. We have also tried to optimize the power of element problem using divide and conquer strategy.

Practical: 4

Greedy Approach

4.1

AIM: A cashier at any mall needs to give change of an amount to customers many times in a day. Cashier has multiple number of coins available with different denominations which is described by a set C. Implement the program for a cashier to find the minimum number of coins required to find a change of a particular amount A. Output should be the total number of coins required of given denominations.

Check the program for following test cases:

Test Case	Coin denominations C	Amount A
1	₹1, ₹2, ₹3	₹ 5
2	₹18, ₹17, ₹5, ₹1	₹ 22
3	₹100, ₹25, ₹10, ₹5, ₹1	₹ 289

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
// All denominations of Indian Currency
int deno[] = { 1, 5, 10, 25, 100};
int n = sizeof(deno)/sizeof(deno[0]);
// Driver program
void findMin(int V)
{
// Initialize result
vector<int> ans;
// Traverse through all denomination
for (int i=n-1; i>=0; i--)
{ // Find denominations
while (V >= deno[i])
{
V -= deno[i];
```

```
        ans.push_back(deno[i]);
    }
}
// Print result
for (int i = 0; i < ans.size(); i++)
{
    cout << ans[i] << " ";
}
}
// Driver program
int main()
{

    int m ;
    cout<<"enter the amount:";
    cin>>m;
    cout << "Following is minimal number of change for " << m << " is ";
    findMin(m);
    return 0;
}
```

OUTPUT:

```
enter the amount:5
Following is minimal number of change for 5 is 5
```

```
enter the amount:22
Following is minimal number of change for 22 is 10 10 1 1
```

```
enter the amount:289
Following is minimal number of change for 289 is 100 100 25 25 25 10 1 1 1 1
```

OBSERVATION:

From the observation we can say that the output no 2 is not optimal because the output for change of 22rs is one 18rs coin and four 1rs coin. Rather the one 17rs coin and one 5rs is more optimal because the total number of coins are reduced to one.

CONCLUSION:

This problem is a variation of 'Coin Change Problem'. Worst case: When the only coin present is Rs. 1 Coin. So in that case, time complexity becomes $O(A)$ where A is the amount to be paid.

4.2

AIM: Let S be a collection of objects with profit-weight values. Implement the fractional Knap sack problem for S assuming we have a sack that can hold objects with total weight W. Check the program for following test cases:

Test Case	S	profit-weight values	W
1	{A,B,C}	Profit:(1,2,5) Weight: (2,3,4)	5
2	{A,B,C,D,E,F,G}	Profit:(10,5,15,7,6,18,3) Weight: (2,3,5,7,1,4,1)	15
3	{A,B,C,D,E,F,G}	A:(12,4),B:(10,6), C:(8,5),D:(11,7), E:(14,3),F:(7,1), G:(9,6)	18

PROGRAM:

```
#include<iostream>
#include<math.h>
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n, weight;
    cout << "enter the number of element : ";
    cin >> n ;
    float p[n],w[n];
    float pw[n];
    cout << "enter the Profit for all " << n << " elements." <<endl;
    for(int i=0;i<n;i++){
        cout << "enter the profit of "<< i+1 <<"th element : " ;
        cin >> p[i];
        cout << "enter the weight of "<< i+1 <<"th element : " ;
        cin >> w[i];
        pw[i] = p[i]/w[i];
    }
    cout<<endl << "Enter the maximum weight : ";
    cin >> weight;
```



```

for(int i=0;i<n;i++){
    for(int j=0;j<n-1;j++){
        if(pw[j] < pw[j+1]){
            float temp = pw[j];
            pw[j] = pw[j+1];
            pw[j+1] = temp;

            temp = p[j];
            p[j] = p[j+1];
            p[j+1] = temp;

            temp = w[j];
            w[j] = w[j+1];
            w[j+1] = temp;
        }
    }
}
cout <<endl<< "Considering weight of : " <<endl;
float ans_p =0;
int ans = 0, i = 0;
while((ans+w[i]) <= weight){
    ans = ans + w[i];
    ans_p = ans_p + p[i];
    cout << w[i]<< " " << p[i] << " " << pw[i] << " " <<endl;
    i++;
}
if(ans <= weight){
    float temp1 = weight-ans;
    float temp2 = temp1/w[i];
    ans_p = ans_p + (p[i]*temp2);
    cout <<"Weight : " << temp1 << " profit : " << (p[i]*temp2) << " this is the " <<temp1<< "
part of having weight " <<w[i]<< " and profit " << p[i]<<endl;

}
cout<<endl<< "Final max profit is : " <<ans_p<<endl;
}

```

OUTPUT:

```
enter the number of element : 3
enter the Profit for all 3 elements.
enter the profit of 1th element : 1
enter the weight of 1th element : 2
enter the profit of 2th element : 2
enter the weight of 2th element : 3
enter the profit of 3th element : 5
enter the weight of 3th element : 4

Enter the maximum weight : 5

Considering weight of :
4 5 1.25
Weight : 1 profit : 0.666667 this is the 1 part of having weight 3 and profit 2

Final max profit is : 5.66667
```

```
enter the number of element : 7
enter the Profit for all 7 elements.
enter the profit of 1th element : 10
enter the weight of 1th element : 2
enter the profit of 2th element : 5
enter the weight of 2th element : 3
enter the profit of 3th element : 15
enter the weight of 3th element : 5
enter the profit of 4th element : 7
enter the weight of 4th element : 7
enter the profit of 5th element : 6
enter the weight of 5th element : 1
enter the profit of 6th element : 18
enter the weight of 6th element : 4
enter the profit of 7th element : 3
enter the weight of 7th element : 1

Enter the maximum weight : 15

Considering weight of :
1  6  6
2  10 5
4  18 4.5
5  15 3
1  3  3
Weight : 2 profit : 3.33333 this is the 2 part of having weight 3 and profit 5

Final max profit is : 55.3333
```

```
enter the number of element : 7
enter the Profit for all 7 elements.
enter the profit of 1th element : 12
enter the weight of 1th element : 4
enter the profit of 2th element : 10
enter the weight of 2th element : 6
enter the profit of 3th element : 8
enter the weight of 3th element : 5
enter the profit of 4th element : 11
enter the weight of 4th element : 7
enter the profit of 5th element : 14
enter the weight of 5th element : 3
enter the profit of 6th element : 7
enter the weight of 6th element : 1
enter the profit of 7th element : 9
enter the weight of 7th element : 6

Enter the maximum weight : 18

Considering weight of :
1  7  7
3  14  4.66667
4  12  3
6  10  1.66667
Weight : 4 profit : 6.4 this is the 4 part of having weight 5 and profit 8

Final max profit is : 49.4
```

CONCLUSION:

In this practical we implemented the concept of knapsack problem.

4.3

AIM: Suppose you want to schedule N activities in a Seminar Hall. Start time and Finish time of activities are given by pair of (si,fi) for ith activity. Implement the program to maximize the utilization of Seminar Hall. (Maximum activities should be selected.)

Test Case	Number of activities (N)	(si,fi)
1	9	(1,2), (1,3), (1,4), (2,5), (3,7), (4,9), (5,6), (6,8), (7,9)
2	11	(1,4), (3,5), (0,6), (3,8), (5,7), (5,9), (6,10), (8,12), (8,11), (12,14), (2,13)

PROGRAM:

```
#include <iostream>
#include<math.h>
#include <bits/stdc++.h>
using namespace std;

struct Activity
{
    int start, finish;
};
// A utility function that is used for sorting
bool activityCompare(Activity s1, Activity s2)
{
    return (s1.finish < s2.finish);
}
void printMaxActivities(Activity arr[], int n)
{
    sort(arr, arr+n, activityCompare);
    cout << "Following activities are selected: ";
    int i = 0;
    cout << "(" << arr[i].start << ", " << arr[i].finish << ") ";
    for (int j = 1; j < n; j++)
    {
        if (arr[j].start >= arr[i].finish)
```

```

    {
        cout << "(" << arr[j].start << ", "
            << arr[j].finish << ") ";
        i = j;
    }
}
}
int main()
{
    Activity arr[] = {{1,2}, {1,3}, {1,4}, {2,5}, {3,7}, {4,9}, {5,6}, {6,8}, {7,9}};
    // Activity arr[] = {{1,4},{3,5},{0,6},{3,8},{5,7},{5,9}, {6,10}, {8,12}, {8,11}, {12,14},
    {2,13}};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "\nTotal number of activities: " << n << "\n";
    cout << "Values of activities are:";
    for (int j = 0; j < n; j++)
    {
        cout << "(" << arr[j].start << ", "
            << arr[j].finish << ") ";
    }
    cout << "\n";
    printMaxActivities(arr, n);
    return 0;
}

```

OUTPUT:

```

Total number of activities: 9
Values of activities are:(1, 2) (1, 3) (1, 4) (2, 5) (3, 7) (4, 9) (5, 6) (6, 8) (7, 9)
Following activities are selected: (1, 2) (2, 5) (5, 6) (6, 8)

Total number of activities: 11
Values of activities are:(1, 4) (3, 5) (0, 6) (3, 8) (5, 7) (5, 9) (6, 10) (8, 12) (8, 11) (12, 14) (2, 13)
Following activities are selected: (1, 4) (5, 7) (8, 11) (12, 14)

```

CONCLUSION:

In this practical I have learnt and implemented the concept of Activity Selection Problem.

Practical 5

Dynamic Programming

5.1

AIM: Implement a program which has BNMCOEF() function that takes two parameters n and k and returns the value of Binomial Coefficient $C(n, k)$. Compare the dynamic programming implementation with recursive implementation of BNMCOEF(). (In output, entire table should be displayed.)

Test Case	n	k
1	5	2
2	11	6
3	12	5

PROGRAM:

```
#include<iostream>
using namespace std;

int binomialCoeff(int n, int k)
{
    // Base Cases
    if (k == 0 || k == n)
        return 1;

    // Recur
    return binomialCoeff(n - 1, k - 1) +
           binomialCoeff(n - 1, k);
}

int main()
{
    int n,k;
    cout<<"...BINOMIAL CO-EFFICIENT...\n";
    cout<<"Enter the value of n : ";
    cin>>n;
    cout<<"Enter the value of k : ";
    cin>>k;
```

```
    cout<<"\nValue is : "<<binomialCoeff(n,k)<<endl;  
    return 0;  
}
```

OUTPUT:

```
...BINOMIAL CO-EFFICIENT...  
Enter the value of n : 5  
Enter the value of k : 2  
  
Value is : 10
```

```
...BINOMIAL CO-EFFICIENT...  
Enter the value of n : 12  
Enter the value of k : 5  
  
Value is : 792
```

```
...BINOMIAL CO-EFFICIENT...  
Enter the value of n : 11  
Enter the value of k : 6  
  
Value is : 462
```

CONCLUSION: In this practical we implemented Binomial Coefficient using Dynamic Programming.

5.2

AIM: Implement the program 4.2 using Dynamic Programming. Compare Greedy and Dynamic approach.

PROGRAM:

```
#include<iostream>
using namespace std;

int maximum(int x,int y)
{
    if(x>y)
        return x;
    else
        return y;
}

int knapsack(int bag_capacity,int weight[],int profit[],int number)
{
    int matrix[number+1][bag_capacity+1];

    for(int i=0;i<number+1;i++)
        for(int j=0;j<bag_capacity+1;j++)
        {
            if(i==0 || j==0)
                matrix[i][j]=0;
            else if (j>=weight[i-1])
                matrix[i][j]=maximum(matrix[i-1][j],profit[i-1]+matrix[i-1][j-weight[i-1]]);
            else
                matrix[i][j]=matrix[i-1][j];
        }

    return matrix[number][bag_capacity];
}

int main()
{
    int number,bag_capacity;
    cout<<".....BINARY KNAPSACK PROBLEM.....";
    cout<<"\nEnter the size of arrays : ";
    cin>>number;

    int weight[number],profit[number];
```

```

cout<<"\nEnter the weights :";
for(int i=0;i<number;i++)
    cin>>weight[i];

cout<<"Enter the profits :";
for(int i=0;i<number;i++)
    cin>>profit[i];

cout<<"Enter bag capacity : ";
cin>>bag_capacity;

cout<<"\nMaximum possible profit is: " << knapsack(bag_capacity,weight,profit,number)
    <<endl;

return 0;
}

```

OUTPUT:

<pre>BINARY KNAPSACK PROBLEM..... Enter the size of arrays : 3 Enter the weights :2 3 4 Enter the profits :1 2 5 Enter bag capacity : 5 Maximum possible profit is : 5 </pre>	<pre>BINARY KNAPSACK PROBLEM..... Enter the size of arrays : 7 Enter the weights :2 3 4 7 1 4 1 Enter the profits :10 5 15 7 6 18 3 Enter bag capacity : 15 Maximum possible profit is : 57 </pre>
<pre>BINARY KNAPSACK PROBLEM..... Enter the size of arrays : 7 Enter the weights :4 6 5 7 3 1 6 Enter the profits :12 10 8 11 14 7 9 Enter bag capacity : 18 Maximum possible profit is : 44 </pre>	

CONCLUSION:

In this practical I have learnt how to implement the 4.2 program using Dynamic Programing.

5.3

AIM: Given a chain $\langle A_1, A_2, \dots, A_n \rangle$ of n matrices, where for $i=1, 2, \dots, n$ matrix A_i with dimensions. Implement the program to fully parenthesize the product A_1, A_2, \dots, A_n in a way that minimizes the number of scalar multiplications. Also calculate the number of scalar multiplications for all possible combinations of matrices.

Test Case	n	Matrices with dimensions
1	3	$A_1: 3 \times 5, A_2: 5 \times 6, A_3: 6 \times 4$
2	6	$A_1: 30 \times 35, A_2: 35 \times 15, A_3: 15 \times 5, A_4: 5 \times 10, A_5: 10 \times 20, A_6: 20 \times 25$

PROGRAM:

```
#include<bits/stdc++.h>
using namespace std;

int MatrixMultiplication(int p[], int n)
{
    int m[n][n];

    int i, j, k, L, q;

    for (i = 1; i < n; i++)
        m[i][i] = 0;

    // L is chain length.
    for (L = 2; L < n; L++)
    {
        for (i = 1; i < n - L + 1; i++)
        {
            j = i + L - 1;
            m[i][j] = INT_MAX;
            for (k = i; k <= j - 1; k++)
            {
                // q = cost/scalar multiplications
                q = m[i][k] + m[k + 1][j] +
                    p[i - 1] * p[k] * p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }
}
```

```

        return m[1][n - 1];
    }

int main()
{
    int n;
    cout<<"...MATRIX CHAIN MULTIPLICATION...\n\n";
    cout<<"Enter total number of dimension values : ";
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)
    {
        cout<<"Enter P"<<i<<" : ";
        cin>>arr[i];
    }
    int length=sizeof(arr)/sizeof(arr[0]);

    cout    <<    "Minimum    number    of    multiplications    is    :
"<<MatrixMultiplication(arr,length)<<endl;

    return 0;
}

```

OUTPUT:

```

...MATRIX CHAIN MULTIPLICATION...

Enter total number of dimension values : 4
Enter P0 : 3
Enter P1 : 5
Enter P2 : 6
Enter P3 : 4
Minimum number of multiplications is : 162

```

```

...MATRIX CHAIN MULTIPLICATION...

Enter total number of dimension values : 7
Enter P0 : 30
Enter P1 : 35
Enter P2 : 15
Enter P3 : 5
Enter P4 : 10
Enter P5 : 20
Enter P6 : 25
Minimum number of multiplications is : 15125

```

CONCLUSION:

In this practical I have learnt how to implement scalar multiplications.

5.4

AIM: Implement a program to print the longest common subsequence for the following strings:

Check the program for following test cases:

Test Case	String1	String2
1	ABCDAB	BDCABA
2	EXPONENTIAL	POLYNOMIAL
3	LOGARITHM	ALGORITHM

PROGRAM:

```
#include<iostream>
#include<string.h>
using namespace std;
int max(int a, int b);
int lcs( char *X, char *Y, int m, int n )
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m-1] == Y[n-1])
        return 1 + lcs(X, Y, m-1, n-1);
    else
        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}
int max(int a, int b)
{
    return (a > b)? a : b;
}
int main()
{
    cout<<"...LONGEST COMMON SUBSEQUENCE..."<<endl;
    char X[100],Y[100];
    cout<<"Enter 1st string sequence : ";cin>>X;
    cout<<"Enter 2nd string sequence : ";cin>>Y;
    int m = strlen(X);int n = strlen(Y);
    cout<<"Length of LCS is : "<<lcs(X,Y,m,n);
    return 0;
}
```

OUTPUT:

```
...LONGEST COMMON SUBSEQUENCE...  
Enter 1st string sequence : ABCDAB  
Enter 2nd string sequence : BDCABA  
Length of LCS is : 4
```

```
...LONGEST COMMON SUBSEQUENCE...  
Enter 1st string sequence : EXPONENTIAL  
Enter 2nd string sequence : POLYNOMIAL  
Length of LCS is : 6
```

```
...LONGEST COMMON SUBSEQUENCE...  
Enter 1st string sequence : LOGARITHM  
Enter 2nd string sequence : ALGORITHM  
Length of LCS is : 7
```

CONCLUSION:

- Time Complexity of the above implementation is $O(mn)$ where m and n is length of string1 and string2.
- And, it is observed to be much better than the worst-case time complexity of Naive Recursive implementation.

Practical 6: Graph

6.1

AIM: Write a program to detect cycles in a directed graph.

PROGRAM:

```
#include<iostream>
using namespace std;

void DFS(int);
int G[20][20],visited[20],n;
bool flag=false;

main()
{
    int i,j;
    cout<<"Enter number of vertices : ";
    cin>>n;
    cout<<"\nEnter adjacency matrix of the graph \n";

    for(i=0;i<n;i++)
    {
        cout<<"For "<<i<<" : ";
        for(j=0;j<n;j++)
            cin>>G[i][j];
    }
    for(i=0;i<n;i++)
        visited[i]=0;

    DFS(0);
    if(flag)
        cout<<"\n\nLoop is present in the graph.";
    else
        cout<<"\n\nLoop is not present in the graph.";
}

void DFS(int i)
{
    int j;
    if(visited[i]==1)
    {
        flag=true;
        return;
    }
    visited[i]=1;
```

```
        for(j=0;j<n;j++)  
        if(G[i][j]==1)  
            DFS(j);  
    }
```

OUTPUT:

```
Enter number of vertices : 3  
:  
Enter adjacency matrix of the graph  
For 0 : 0 1 1  
For 1 : 1 0 1  
For 2 : 1 0 0  
  
Loop is present in the graph.
```

```
Enter number of vertices : 4  
:  
Enter adjacency matrix of the graph  
For 0 : 0 1 0 0  
For 1 : 0 0 1 0  
For 2 : 0 0 0 0  
For 3 : 1 0 0 0  
  
Loop is not present in the graph.
```

CONCLUSION:

- Depth First Traversal can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree.
- Time Complexity: $O(V + E)$

6.2

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

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
    int G[MAX][MAX],i,j,n,u;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);
    printf("\nEnter the starting node:");
    scanf("%d",&u);
    dijkstra(G,n,u);
    return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
    int cost[MAX][MAX],distance[MAX],pred[MAX];
    int visited[MAX],count,mindistance,nextnode,i,j;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(G[i][j]==0)
                cost[i][j]=INFINITY;
            else
                cost[i][j]=G[i][j];
    for(i=0;i<n;i++)
        { distance[i]=cost[startnode][i];
          pred[i]=startnode;
          visited[i]=0; }
    distance[startnode]=0;
    visited[startnode]=1;
    count=1;
    while(count<n-1)
    {
        mindistance=INFINITY;
        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
```

```

{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
    distance[i]=mindistance+cost[nextnode][i];
    pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
    printf("\nDistance of node%d=%d",i,distance[i]);
    printf("\nPath=%d",i);
    j=i;
do
{
    j=pred[j];
    printf("<-%d",j);
} while(j!=startnode);
}
}

```

OUTPUT:

1.

```

Enter no. of vertices:8
Enter the adjacency matrix:
0 0 0 2 0 0 0 0
0 0 0 0 0 0 7 0
0 0 0 0 3 0 0 0
2 0 0 0 0 0 0 0
0 0 3 0 0 0 1 7
0 0 0 0 0 0 9 0
0 7 0 0 1 9 0 0
0 0 0 0 7 0 0 0
Enter the starting node:1
Distance of node0=9999
Path=0<-1
Distance of node2=11
Path=2<-4<-6<-1
Distance of node3=9999
Path=3<-1
Distance of node4=8
Path=4<-6<-1
Distance of node5=16
Path=5<-6<-1
Distance of node6=7
Path=6<-1
Distance of node7=15
Path=7<-4<-6<-1
Process returned 0 (0x0)   execution time : 123.798 s
Press any key to continue.

```

2.

```

Enter no. of vertices:8

Enter the adjacency matrix:
0 0 2 0 0 0 0 0
6 0 0 0 0 0 0 0
3 8 0 0 5 0 0 0
0 9 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 7 0 0 0 0
0 0 9 0 4 0 0 3
0 0 0 0 0 1 6 0

Enter the starting node:3

Distance of node0=15
Path=0<-1<-3
Distance of node1=9
Path=1<-3
Distance of node2=17
Path=2<-0<-1<-3
Distance of node4=22
Path=4<-2<-0<-1<-3
Distance of node5=27
Path=5<-7<-6<-4<-2<-0<-1<-3
Distance of node6=23
Path=6<-4<-2<-0<-1<-3
Distance of node7=26
Path=7<-6<-4<-2<-0<-1<-3
Process returned 0 (0x0)    execution time : 94.857 s
Press any key to continue.

```

CONCLUSION:

- Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree.
- Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.
- Time Complexity: $O(E \log V)$

6.3

AIM: Find Minimum Cost spanning tree of a given undirected graph using Prim's algorithm.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX],spanning[MAX][MAX],n;
int prims();
int main()
{
    int i,j,total_cost;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);
    total_cost=prims();
    printf("\nspanning tree matrix:\n");
    for(i=0;i<n;i++)
    {
        printf("\n");
        for(j=0;j<n;j++)
            printf("%d\t",spanning[i][j]);    }
    printf("\n\nTotal cost of spanning tree=%d",total_cost);
    return 0;    }

int prims()
{
    int cost[MAX][MAX];
    int u,v,min_distance,distance[MAX],from[MAX];
    int visited[MAX],no_of_edges,i,min_cost,j;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            if(G[i][j]==0)
                cost[i][j]=infinity;
            else
                cost[i][j]=G[i][j];
            spanning[i][j]=0;
        }
    distance[0]=0;
    visited[0]=1;
    for(i=1;i<n;i++)
    {
        distance[i]=cost[0][i];
        from[i]=0;
    }
}
```

```

        visited[i]=0; }
    min_cost=0;
    no_of_edges=n-1;
    while(no_of_edges>0)
    {
        min_distance=infinity;
        for(i=1;i<n;i++)
            if(visited[i]==0&&distance[i]<min_distance)
            {
                v=i;
                min_distance=distance[i];
            }
        u=from[v];
        spanning[u][v]=distance[v];
        spanning[v][u]=distance[v];
        no_of_edges--;
        visited[v]=1;
        for(i=1;i<n;i++)
            if(visited[i]==0&&cost[i][v]<distance[i])
            {
                distance[i]=cost[i][v];
                from[i]=v;
            }
        min_cost=min_cost+cost[u][v];
    }
    return(min_cost);
}

```

OUTPUT:

```

Enter no. of vertices:5
Enter the adjacency matrix:
0 1 1 0 1
1 0 1 0 1
1 0 1 0 0
0 0 1 0 1
1 1 0 1 0
spanning tree matrix:
0      1      1      0      1
1      0      0      0      0
1      0      0      1      0
0      0      1      0      0
1      0      0      0      0
Total cost of spanning tree=4

```

```

Enter no. of vertices:
8
Enter the adjacency matrix:
0 0 0 2 0 0 0 0
0 0 0 0 0 0 7 0
0 0 0 0 3 0 0 0
2 0 0 0 0 0 0 0
0 0 3 0 0 0 1 7
0 0 0 0 0 0 9 0
0 7 0 0 1 9 0 0
0 0 0 0 7 0 0 0
spanning tree matrix:
0      0      0      2      0      0      0      0
0      0      0      0      0      0      0      0
0      0      0      0      0      0      0      0
0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0
0      0      0      0      0      0      0      0
0      0      0      0      0      0      0      0
0      0      0      0      0      0      0      0
Total cost of spanning tree=14

```

CONCLUSION:

- Prim's algorithm is very similar to Kruskal's: whereas Kruskal's "grows" a forest of trees, Prim's algorithm grows a single tree until it becomes the minimum spanning tree.
- Time Complexity: $O(E \log V)$

Practical 7: Backtracking

7.1

AIM: Implement a program to print all permutations of a given string.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
void swap(char *x, char *y)
{
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
void permute(char *a, int l, int r)
{
    int i;
    if (l == r)
        printf("%s\n", a);
    else
    {
        for (i = l; i <= r; i++)
        {
            swap((a+l), (a+i));
            permute(a, l+1, r);
            swap((a+l), (a+i)); //backtrack
        }
    }
}
int main()
{
    char str[] = "NOTE";
    int n = strlen(str);
    permute(str, 0, n-1);
    return 0;
}
```

OUTPUT:

```
ACT
ATC
CAT
CTA
TCA
TAC

Process returned 0 (0x0)   execution time : 0.013 s
Press any key to continue.
```

```
NOTE
NOET
NTOE
NTEO
NETO
NEOT
ONTE
ONET
OTNE
OTEN
OETN
OENT
TONE
TOEN
TNOE
TNEO
TEN0
TEON
EOTN
EONT
ETON
ETNO
ENTO
ENOT

Process returned 0 (0x0)   execution time : 0.013 s
Press any key to continue.
```

CONCLUSION:

- Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.
- A permutation, also called an “arrangement number” or “order,” is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself.
- **Time Complexity:** $O(n*n!)$.

Practical 8: String Matching Algorithm

8.1

AIM: Suppose you are given a source string $S[0..n-1]$ of length n , consisting of symbols a and b . Suppose that you are given a pattern string $P[0..m-1]$ of length $m < n$, consisting of symbols a , b , and $*$, representing a pattern to be found in string S . The symbol $*$ is a “wild card” symbol, which matches a single symbol, either a or b . The other symbols must match exactly. The problem is to output a sorted list M of valid “match positions”, which are positions j in S such that pattern P matches the substring $S[j..j+|P|-1]$. For example, if $S = ababbab$ and $P = ab*$, then the output M should be $[0, 2]$. Implement a straightforward, naive algorithm to solve the problem.

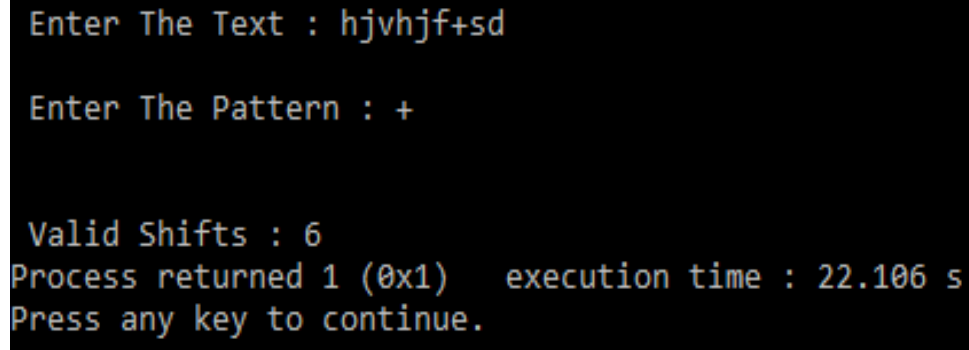
PROGRAM:

```
#include<stdio.h>
void main()
{
    char t[100],p[100];
    int tn,pn,shift[20]={0},s=0,i,j=0,count=0,m=0;
    printf("\n Enter The Text : ");
    scanf("%s",t);
    fflush(stdin);
    printf("\n Enter The Pattern : ");
    scanf("%s",p);
    tn = strlen(t);
    pn = strlen(p);
    while(s!=(tn-pn+1))
    {
        j=0;
        for(i=s;i<pn+s;i++)
        {
            if(p[j]==t[i])
            {
                count++;
                if(count==pn)
                {
                    count=0;
                    shift[m]=s;
                    m++;
                }
            }
            else
            {
                count=0;
                break;
            }
            j++;
        }
        s++;
    }

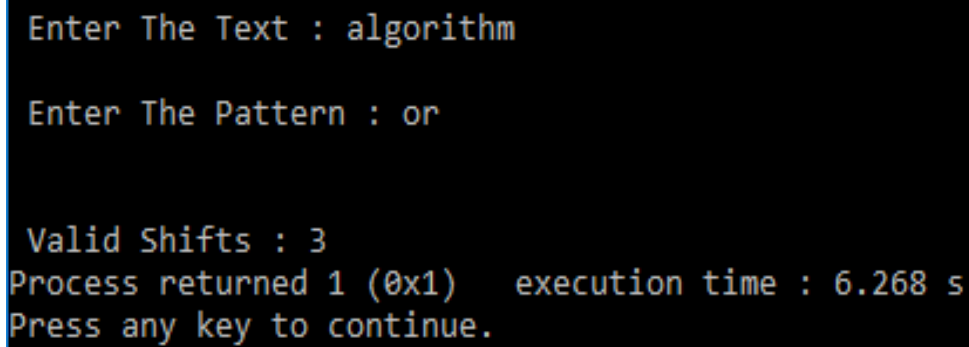
    if(m>0)
```



```
{    printf("\n\n Valid Shifts : ");  
    for(i=0;i<m;i++) printf("%d ",shift[i]);  
}  
else  
{    printf("\n\n No Valid Shifts."); }  
}
```

OUTPUT:

```
Enter The Text : hjevbjf+sd  
  
Enter The Pattern : +  
  
Valid Shifts : 6  
Process returned 1 (0x1)   execution time : 22.106 s  
Press any key to continue.
```



```
Enter The Text : algorithm  
  
Enter The Pattern : or  
  
Valid Shifts : 3  
Process returned 1 (0x1)   execution time : 6.268 s  
Press any key to continue.
```

CONCLUSION:

- Naive pattern searching is the simplest method among other pattern searching algorithms.
- It checks for all character of the main string to the pattern.
- This algorithm is helpful for smaller texts. It does not need any pre-processing phases.
- The time complexity is $O(m*n)$. The m is the size of pattern and n is the size of the main string.

8.2

AIM: Implement Rabin karp algorithm and test it on the following test cases:

Test Case	String	Pattern
1	2359023141526739921	31415 q=13
2	ABAAABCDBBABCDDDEBCABC	ABC q=101

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define d 256
void search(char pat[], char txt[], int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0; // hash value for pattern
    int t = 0; // hash value for txt
    int h = 1;
    for (i = 0; i < M-1; i++)
        h = (h*d)%q;
    for (i = 0; i < M; i++)
    {
        p = (d*p + pat[i])%q;
        t = (d*t + txt[i])%q;
    }
    for (i = 0; i <= N - M; i++)
    {
        if ( p == t )
        {
            for (j = 0; j < M; j++)
            {
                if (txt[i+j] != pat[j])
                    break; }

            if (j == M)
                printf("Pattern found at index %d \n", i);
        }
        if ( i < N-M )
        {
            t = (d*(t - txt[i]*h) + txt[i+M])%q;
            if (t < 0)
                t = (t + q); } } }

int main()
{
    char txt[] = "ABAAABCDBBABCDDDEBCABC";
    char pat[] = "ABC";
    int q = 101; // A prime number
    search(pat, txt, q);
}
```

```
    return 0;  
}
```

OUTPUT:

```
Pattern found at index 6  
  
Process returned 0 (0x0)   execution time : 0.013 s  
Press any key to continue.
```

```
Pattern found at index 4  
Pattern found at index 10  
Pattern found at index 18  
  
Process returned 0 (0x0)   execution time : 0.011 s  
Press any key to continue.
```

CONCLUSION:

- The Rabin–Karp algorithm is a form of rolling hash used in string searching ^[1] to find any one of a set of pattern strings in a text.
- For text of length n and p patterns of combined length m , its average and best case time complexity is $\underline{O}(n+m)$, but its worst-case time complexity is $O(nm)$.