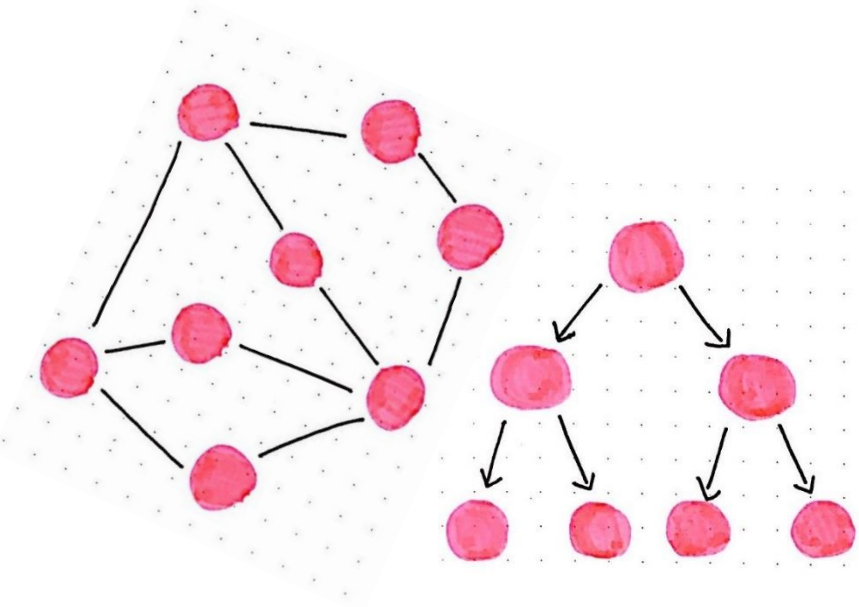
Atma Ram Sanatan Dharma College

University of Delhi



Practical File for

Discrete Structures

Paper Code - 32341202

Submitted By –

Anshul Verma

College Roll No. – 19/78065

B.Sc. (Hons) Computer Science

Submitted To –

Ms. Shalini Gupta

Department of Computer Science

Index

Practical #01

Write a Program to create a SET A and determine the cardinality of SET for an input array of elements (repetition allowed) and perform the following operations on the SET:

a) ismember (a, A): check whether an element belongs to set or not and return value as true/false.

b) powerset(A): list all the elements of power set of A.

Code :

```
#include <iostream>
#include <math.h>
using namespace std;

void arraytoSet(int *arr, int *n)
{
    if (*n == 0 || *n == 1)
        return;
    for (int i = 0; i < *n; i++)
    {
        for (int j = i + 1; j < *n; j++)
        {
            if (arr[i] == arr[j])
            {
                for (int k = j + 1; k < *n; k++)
                {
                    arr[j] = arr[k];
                    j++;
                }
                *n = *n - 1;
```

```
                j--;
            }
        }
    }
}
void print(int *arr, int n)
{
    cout << "{";
    for (int i = 0; i < n; i++)
    {
        if (i == 0)
            cout << arr[i];
        else
            cout << "," << arr[i];
    }
    cout << "} ";
}
bool ismember(int a, int *arr, int *n)
{
    for (int i = 0; i < *n; i++)
    {
        if (a == arr[i])
        {
            return true;
            break;
        }
    }
    return false;
}
/*  Helper recursive function for powerset function
    set = initial set
    n = initial set
    set_i = counter for initial set
    power_set = power set in which subsets will be stored
```

```
    powerset_i = current index for power set
*/
void helper_powerset(int *set, int n, int set_i, int *power_set, int
powerset_i)
{
    //Check for terminating recursion
    if (set_i >= n)
    {
        print(power_set, powerset_i);
        return;
    }
    //Exclude
    helper_powerset(set, n, set_i + 1, power_set, powerset_i);
    power_set[powerset_i] = set[set_i];
    //Include
    helper_powerset(set, n, set_i + 1, power_set, powerset_i + 1);
}
void powerset(int *arr, int n)
{
    int power_set[(int)pow(2, n)];
    cout << "{ ";
    helper_powerset(arr, n, 0, power_set, 0);
    cout << "}\n\n" << endl;
}
int main()
{
    int *A;
    int n;

    cout << "\nEnter the size of array : ";
    cin >> n;
    cout << "Enter the array elements : ";
    for (int i = 0; i < n; i++)
    {
```

```
        cin >> A[i];
    }

    arraytoSet(A, &n);

    cout << endl
         << "Given set A : ";
    print(A, n);
    cout << "\n\nCardinality of set A : " << n << endl;
    cout << endl
         << "Let a = 2, then" << endl;
    int a = 2;
    cout << "ismember(" << a << ", A) : ";
    if (ismember(a, A, &n) == 1)
        cout << "true" << endl;
    else
        cout << "false" << endl;

    cout << "\npowerset(A) : ";
    powerset(A, n);
    return 0;
}
```

Output :

```
Enter the size of array : 5
Enter the array elements : 2 3 4 3 4

Given set A : {2,3,4}

Cardinality of set A : 3

Let a = 2, then
ismember(2, A) : true

powerset(A) : { {} {4} {3} {3,4} {2} {2,4} {2,3} {2,3,4} }
```

Practical #02

Create a class SET and take two sets as input from user to perform following SET Operations:

a) Subset: Check whether one set is a subset of other or not.

b) Union and Intersection of two Sets.

c) Complement: Assume Universal Set as per the input elements from the user.

d) Set Difference and Symmetric Difference between two SETS

e) Cartesian Product of Sets.

Code:

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

class Set
{
public:
    int elements[100];
    int size = 0;

    Set();
    void input();
    void print();
    bool subset_of(Set);
    Set union_with(Set);
    Set intersection_with(Set);
    Set complement(Set);
    Set difference(Set);
    Set difference_sym(Set);
```

```
    void cartesian_prod(Set);

private:
    void addElement(int);
    bool has(int i);
};

Set::Set()
{
}
void Set::input()
{
    int itrCount;
    cout << "\nEnter the size of set : ";
    cin >> itrCount;
    cout << "Enter the elements of set : ";
    for (int i = 0; i < itrCount; i++)
    {
        int e;
        cin >> e;
        this->addElement(e);
    }
    cout << endl;
}
bool Set::has(int n)
{
    for (int i = 0; i < size; i++)
    {
        if (n == elements[i])
        {
            return true;
            break;
        }
    }
```

```
    return false;
}
void Set::addElement(int n)
{
    if (!has(n))
    {
        elements[size] = n;
        ++size;
    }
}
bool Set::subset_of(Set set)
{
    int count = 0;
    for (int i = 0; i < this->size; i++)
    {
        for (int j = 0; j < set.size; j++)
        {
            if (this->elements[i] == set.elements[j])
            {
                count++;
                break;
            }
        }
    }
    if (count == size)
        return true;
    else
        return false;
}
Set Set::union_with(Set set)
{
    Set temp;
    for (int i = 0; i < size; i++)
    {
```

```
        temp.addElement(elements[i]);
    }
    for (int i = 0; i < set.size; i++)
    {
        temp.addElement(set.elements[i]);
    }
    return temp;
}
Set Set::intersection_with(Set set)
{
    Set temp;
    for (int i = 0; i < size; i++)
    {
        if (set.has(elements[i]))
            temp.addElement(elements[i]);
    }
    return temp;
}
Set Set::complement(Set uni_set)
{
    Set temp;
    for (int i = 0; i < uni_set.size; i++)
    {
        if (!has(uni_set.elements[i]))
            temp.addElement(uni_set.elements[i]);
    }
    return temp;
}
Set Set::difference(Set set)
{
    Set temp;
    for (int i = 0; i < size; i++)
    {
        if (!set.has(elements[i]))
```

```
            temp.addElement(elements[i]);
    }
    return temp;
}
Set Set::difference_sym(Set set)
{
    Set unionSet = union_with(set);
    Set intrSet = intersection_with(set);
    return unionSet.difference(intrSet);
}
void Set::cartesian_prod(Set set)
{
    cout << endl
         << "{ ";
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < set.size; j++)
        {
            cout << "{" << elements[i] << "," << set.elements[j] <<
"} ";
        }
    }
    cout << "}" << endl
         << endl
         << endl;
}

void Set::print()
{
    cout << "{";
    for (int i = 0; i < size; i++)
    {
        if (i == 0)
            cout << elements[i];
```

```
        else
            cout << "," << elements[i];
    }
    cout << "} ";
}

int main()
{
    Set setA = Set();
    setA.input();
    cout << "--> Set A : ";
    setA.print();
    Set setB = Set();
    setB.input();
    cout << "--> Set B : ";
    setB.print();

    if (setA.subset_of(setB))
        cout << "\n\n--> Set A is a subset of Set B.\n";
    else
        cout << "\n\n--> Set A is not a subset of Set B.\n";

    cout << endl
         << "--> Set A union Set B : ";
    setA.union_with(setB).print();

    cout << "\n\n--> Set A intersection Set B : ";
    setA.intersection_with(setB).print();

    cout << "\n\n--> Let Universal Set be ";
    Set universalSet = setA.union_with(setB);
    universalSet.print();
    cout << endl
         << "--> Complement of Set A (A') : ";
```

```
    setA.complement(universalSet).print();
    cout << endl
         << "--> Complement of Set B (B') : ";
    setB.complement(universalSet).print();

    cout << "\n\n--> Set A difference Set B : ";
    setA.difference(setB).print();

    cout << "\n\n--> Set A symmetric difference Set B : ";
    setA.difference_sym(setB).print();

    cout << "\n\n--> Cartesian product of Set A and Set B : ";
    setA.cartesian_prod(setB);

    return 0;
}
```

Output :

```
Enter the size of set : 5
Enter the elements of set : 1 4 3 2 1

--> Set A : {1,4,3,2}

Enter the size of set : 7
Enter the elements of set : 8 3 15 5 2 1 6

--> Set B : {8,3,15,5,2,1,6}

--> Set A is not a subset of Set B.

--> Set A union Set B : {1,4,3,2,8,15,5,6}

--> Set A intersection Set B : {1,3,2}

--> Let Universal Set be {1,4,3,2,8,15,5,6}
--> Complement of Set A (A') : {8,15,5,6}
--> Complement of Set B (B') : {4}

--> Set A difference Set B : {4}

--> Set A symmetric difference Set B : {4,8,15,5,6}

--> Cartesian product of Set A and Set B :
{ {1,8} {1,3} {1,15} {1,5} {1,2} {1,1} {1,6} {4,8} {4,3} {4,15} {4,5} {4,2} {4,1} {4,6} {3,8} {3,3} {3,15} {3,5} {3,2} {3,1} {3,6} {2,8} {2,3} {2,15} {2,5}
 {2,2} {2,1} {2,6} }
```

Practical #03

Create a class RELATION, use Matrix notation to represent a relation. Include functions to check if a relation is reflexive, Symmetric, Anti-symmetric and Transitive. Write a Program to use this class.

Code :

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

class Relation
{
public:
    int matrix[100][100] = {0};
    int size, matrix_size;

    void input();
    void print_matrix();
    void isReflexive();
    void isSymmetric();
    void isAntisymmetric();
    void isTransitive();
};
void Relation::input()
{
    int max = 0;
    cout << endl
         << endl
         << "Enter the length of relation : ";
```

```
    cin >> size;
    cout << "Enter the elements of relation : " << endl;
    for (int i = 0; i < size; i++)
    {
        cout << "Element " << i + 1 << " : ";
        int a, b;
        cin >> a >> b;
        matrix[a - 1][b - 1] = 1;
        if (a > max)
            max = a;
        else if (b > max)
            max = b;
    }
    matrix_size = max;
}

void Relation::print_matrix()
{
    cout << "\n--> Matrix Representation : \n";
    for (int i = 0; i < matrix_size; i++)
    {
        cout << endl
             << "    ";
        for (int j = 0; j < matrix_size; j++)
        {
            cout << matrix[i][j] << " ";
        }
    }
}

void Relation::isReflexive()
{
```

```
    int flag = 0;
    for (int i = 0; i < matrix_size; i++)
    {
        if (matrix[i][i] == 1)
            flag = 1;
        else
        {
            flag = 0;
            break;
        }
    }

    if (flag == 0)
        cout << "--> Given relation is not reflexive\n";
    else
        cout << "--> Given relation is reflexive\n";
}
void Relation::isSymmetric()
{
    int flag = 0;
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            if (matrix[i][j] == matrix[j][i])
            {
                flag = 1;
            }
            else
            {
                flag = 0;
                break;
            }
    if (flag == 0)
```

```
        cout << "--> Given relation is not symmetric\n";
    else
        cout << "--> Given relation is symmetric\n";
}
void Relation::isAntisymmetric()
{
    int flag = 0;
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            if ((matrix[i][j] || matrix[j][i] == 0) && i != j)
            {
                flag = 1;
            }
            else
            {
                flag = 0;
                break;
            }
    if (flag == 0)
        cout << "--> Given relation is antisymmetric\n";
    else
        cout << "--> Given relation is not antisymmetric\n";
}
void Relation::isTransitive()
{
    int flag = 1;
    for (int i = 0; i < matrix_size; i++)
    {
        for (int j = 0; j < matrix_size; j++)
        {
            if (matrix[i][j] == 1)
            {
```

```
                for (int k = 0; k < size; k++)
                {
                    if (matrix[j][k] == 1 && matrix[i][k] != 1)
                    {
                        flag = 0;
                    }
                }
            }
        }
    }
    if (flag == 0)
        cout << "--> Given relation is not transitive\n";
    else
        cout << "--> Given relation is transitive\n";
}

int main()
{
    Relation rel = Relation();
    rel.input();
    rel.print_matrix();

    cout << endl
         << endl;
    rel.isReflexive();
    rel.isSymmetric();
    rel.isTransitive();
    rel.isAntisymmetric();

    return 0;
}
```

Output :

```
Enter the length of relation : 12
Enter the elements of relation :
Element 1 : 1 1
Element 2 : 2 2
Element 3 : 3 3
Element 4 : 4 4
Element 5 : 5 5
Element 6 : 6 6
Element 7 : 6 1
Element 8 : 6 4
Element 9 : 1 4
Element 10 : 6 5
Element 11 : 3 4
Element 12 : 6 2

--> Matrix Representation :

    1 0 0 1 0 0
    0 1 0 0 0 0
    0 0 1 1 0 0
    0 0 0 1 0 0
    0 0 0 0 1 0
    1 1 0 1 1 1

--> Given relation is reflexive
--> Given relation is not symmetric
--> Given relation is transitive
--> Given relation is antisymmetric
```

Practical #04

Use the functions defined in Ques 3 to find check whether the given relation is:

a) Equivalent, or

b) Partial Order relation, or

c) None

Code :

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

class Relation
{
public:
    int matrix[100][100] = {0};
    int size, matrix_size;

    void input();
    void print_matrix();
    bool isReflexive();
    bool isSymmetric();
    bool isAntisymmetric();
    bool isTransitive();
};
void Relation::input()
{
    int max = 0;
    cout << endl
         << endl
```

```
         << "Enter the length of relation : ";
    cin >> size;
    cout << "Enter the elements of relation : " << endl;
    for (int i = 0; i < size; i++)
    {
        cout << "Element " << i + 1 << " : ";
        int a, b;
        cin >> a >> b;
        matrix[a - 1][b - 1] = 1;
        if (a > max)
            max = a;
        else if (b > max)
            max = b;
    }
    matrix_size = max;
}

void Relation::print_matrix()
{
    cout << "\n--> Matrix Representation : \n";
    for (int i = 0; i < matrix_size; i++)
    {
        cout << endl
             << "    ";
        for (int j = 0; j < matrix_size; j++)
        {
            cout << matrix[i][j] << " ";
        }
    }
}

bool Relation::isReflexive()
```

```
{
    int flag = 0;
    for (int i = 0; i < matrix_size; i++)
    {
        if (matrix[i][i] == 1)
            flag = 1;
        else
        {
            flag = 0;
            break;
        }
    }

    return flag;
}
bool Relation::isSymmetric()
{
    int flag = 0;
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            if (matrix[i][j] == matrix[j][i])
            {
                flag = 1;
            }
            else
            {
                flag = 0;
                break;
            }
    return flag;
}
bool Relation::isAntisymmetric()
```

```
{
    int flag = 0;
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            if ((matrix[i][j] || matrix[j][i] == 0) && i != j)
            {
                flag = 1;
            }
            else
            {
                flag = 0;
                break;
            }
    return !flag;
}
bool Relation::isTransitive()
{
    int flag = 1;
    for (int i = 0; i < matrix_size; i++)
    {
        for (int j = 0; j < matrix_size; j++)
        {
            if (matrix[i][j] == 1)
            {
                for (int k = 0; k < size; k++)
                {
                    if (matrix[j][k] == 1 && matrix[i][k] != 1)
                    {
                        flag = 0;
                    }
                }
            }
```

```
        }
    }
    return flag;
}

int main()
{
    Relation rel = Relation();
    rel.input();
    rel.print_matrix();

    if (rel.isReflexive() && rel.isSymmetric() && rel.isTransitive()
)
        cout << "\n\n--
> The given relation is equivalence relation.\n";
    else if (rel.isReflexive() && rel.isAntisymmetric() && rel.isTra
nsitive())
        cout << "\n\n--
> The given relation is a partially ordered relation(poset).\n";
    else
        cout << "\n\n--
> The given relation is neither a equivalence relation nor a poset.\
n";

    return 0;
}
```

Output :

```
Enter the length of relation : 12
Enter the elements of relation :
Element 1 : 1 1
Element 2 : 2 2
Element 3 : 3 3
Element 4 : 4 4
Element 5 : 5 5
Element 6 : 6 6
Element 7 : 6 1
Element 8 : 6 4
Element 9 : 1 4
Element 10 : 6 5
Element 11 : 3 4
Element 12 : 6 2

--> Matrix Representation :

   1 0 0 1 0 0
   0 1 0 0 0 0
   0 0 1 1 0 0
   0 0 0 1 0 0
   0 0 0 0 1 0
   1 1 0 1 1 1

--> The given relation is a partially ordered relation(poset).
```

```
Enter the length of relation : 5
Enter the elements of relation :
Element 1 : 1 3
Element 2 : 3 1
Element 3 : 3 3
Element 4 : 1 2
Element 5 : 2 1

--> Matrix Representation :

    0 1 1
    1 0 0
    1 0 1

--> The given relation is neither a equivalence relation nor a poset.
```

Practical #05

Write a Program to generate the Fibonacci Series using recursion.

Code :

```
#include <iostream>

using namespace std;

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

int main()
{
    int n;
    cout << endl
         << endl
         << "Enter the size of Fibonacci series : ";
    cin >> n;
    cout << endl
         << "--> Fibonacci series (0-" << n << ") : ";
    for (int i = 0; i < n; i++)
    {
```

```
        cout << fib_at(i) << " ";
    }
    cout << endl
         << endl;

    return 0;
}
```

Output :

```
Enter the size of Fibonacci series : 15

--> Fibonacci series (0-15) : 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Practical #06

Write a Program to implement Tower of Hanoi using recursion.

Code :

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

// total moves
int TOH(int n)
{
    if (n == 1)
        return 1;
    return (2 * TOH(n - 1) + 1);
}
void TOH_showmoves(int n, char source_t, char dest_t, char spare_t)
{
    //Base Case
    if (n == 1)
    {
        cout << endl
             << " - Move disk 1 of tower " << source_t << " --
> tower " << dest_t;
        return;
    }
    //Step 1
    TOH_showmoves(n - 1, source_t, spare_t, dest_t);
    //Step 2
    cout << endl
         << " - Move disk " << n << " of tower " << source_t << " --
> tower " << dest_t;
```

```
    //Step 3
    TOH_showmoves(n - 1, spare_t, dest_t, source_t);
}

int main()
{
    int n;
    cout << endl
         << endl
         << "Enter number of disks : ";
    cin >> n;
    cout << endl
         << "Let the towers are A, B and C and we are required to mo
ve the disks from A to C." << endl;
    TOH_showmoves(n, 'A', 'C', 'B');
    cout << endl
         << endl
         << "--> Total number of moves : " << TOH(n) << endl
         << endl;

    return 0;
}
```

Output :

```
Enter number of disks : 3

Let the towers are A, B and C and we are required to move the disks from A to C.

 - Move disk 1 of tower A --> tower C
 - Move disk 2 of tower A --> tower B
 - Move disk 1 of tower C --> tower B
 - Move disk 3 of tower A --> tower C
 - Move disk 1 of tower B --> tower A
 - Move disk 2 of tower B --> tower C
 - Move disk 1 of tower A --> tower C

--> Total number of moves : 7
```

Practical #07

Write a Program to implement binary search using recursion.

Code :

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

void sort(int arr[], int v)
{
    for (int i = 0; i < (v - 1); i++)
    {
        for (int j = 0; j < (v - i - 1); j++)
        {
            if (arr[j] > arr[j + 1])
            {
                float temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int binarySearch(int arr[], int start, int end, int e)
{
    if (end >= 1)
    {
        int mid = (int)start + (end - 1) / 2;
```

```
        if (arr[mid] == e)
            return mid;
        if (arr[mid] > e)
            return binarySearch(arr, start, mid - 1, e);
        return binarySearch(arr, mid + 1, end, e);
    }
    return -1;
}

void print(int arr[], int n)
{
    cout << endl
         << " - [ ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << "]";
}

int main()
{
    int *arr;
    int n, size;

    cout << endl
         << endl
         << "Enter the size of array : ";
    cin >> size;

    cout << endl
         << "Enter the elements of array : ";
```

```
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    cout << endl
         << " - Sorting the array..." << endl;
    sort(arr, size);
    cout << " - Done!";
    print(arr, size);

    cout << endl
         << endl
         << "Enter the integer to be searched : ";
    cin >> n;
    cout << endl
         << " - Searching using Binary Search..." << endl;
    int result = binarySearch(arr, 0, size - 1, n);
    cout << " - Done!" << endl;
    if (result == -1)
        cout << endl
             << "--> Element is not present in the array.";
    else
        cout << endl
             << "--
> Element is present in the array at index " << result << endl
             << endl;

    return 0;
}
```

Output :

```
Enter the size of array : 7

Enter the elements of array : 2 3 8 4 9 3 1

 - Sorting the array...
 - Done!
 - [ 1 2 3 3 4 8 9 ]

Enter the integer to be searched : 4

 - Searching using Binary Search...
 - Done!

--> Element is present in the array at index 4
```

Practical #08

Write a Program to implement Bubble Sort. Find the number of comparisons during each pass and display the intermediate result. Use the observed values to plot a graph to analyse the complexity of algorithm.

Code :

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

void display(int ar[], int v)
{
    cout << " - [ ";
    for (int i = 0; i < v; i++)
    {
        cout << ar[i] << " ";
    }
    cout << "]";
}

int swap(int *ar, int a, int b)
{
    int temp = ar[a];
    ar[a] = ar[b];
    ar[b] = temp;

    return *ar;
}
```

```
int bubbleSort(int *ar, int v)
{
    int totalCount = 0, count;
    for (int i = 0; i < (v - 1); i++)
    {
        count = 0;
        for (int j = 0; j < (v - i - 1); j++)
        {
            if (ar[j] > ar[j + 1])
            {
                ++count;
                swap(ar, j, j + 1);
            }
            else
                ++count;
        }
        totalCount += count;
        display(ar, v);
        cout << " -> " << (count) << " comparisons" << endl;
    }
    return totalCount;
}

int main()
{
    int *arr;
    int size, comparisons;

    cout << endl
         << endl
         << "Enter the size of array : ";
    cin >> size;
```

```
    cout << endl
         << "Enter the elements of the array : ";
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    cout << endl;

    display(arr, size);

    cout << endl
         << endl
         << " - Sorting the array using Bubble Sort Algorithm...." <
< endl
         << endl;
    comparisons = bubbleSort(arr, size);

    cout << endl
         << "--> Sorted. Here is the resulting array";
    display(arr, size);
    cout << endl
         << endl
         << "--> Total no. of comparisons : " << comparisons;
    cout << endl
         << endl;

    return 0;
}
```

Output :

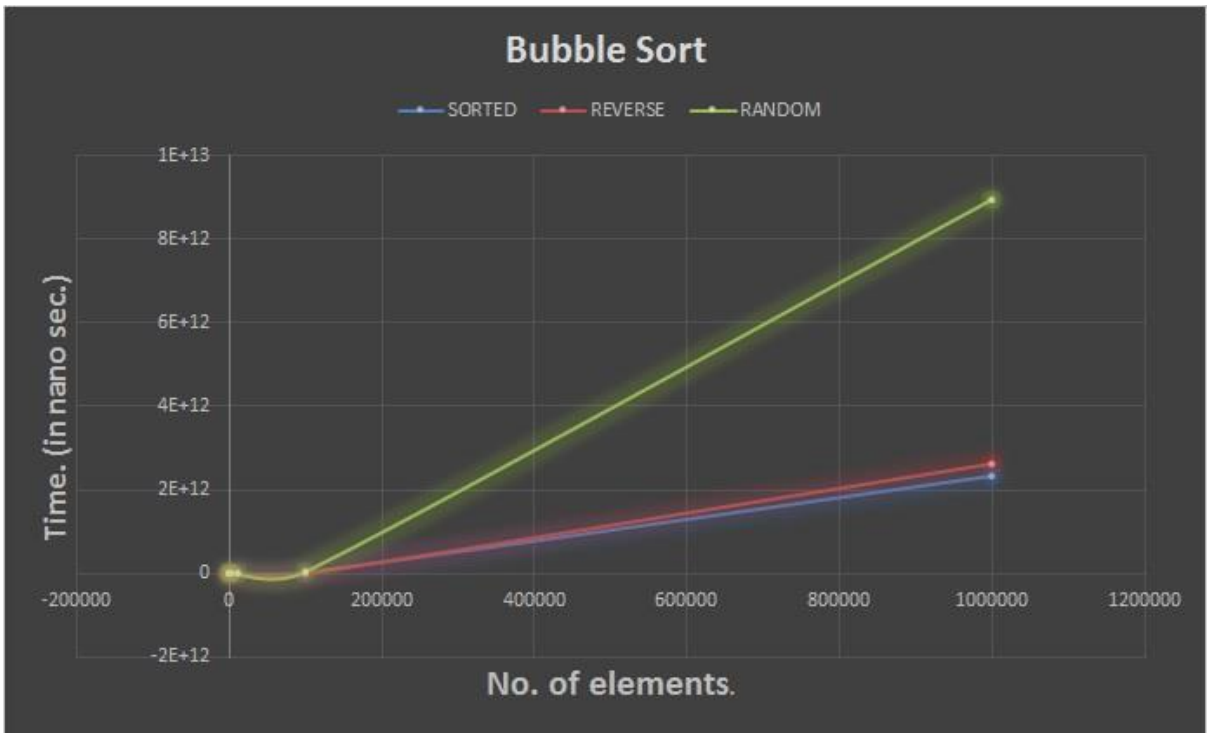
Average Case:

```
Enter the size of array : 6

Enter the elements of the array : 50 20 40 60 10 30

 - [ 50 20 40 60 10 30 ]

 - Sorting the array using Bubble Sort Algorithm....

 - [ 20 40 50 10 30 60 ] -> 5 comparisons
 - [ 20 40 10 30 50 60 ] -> 4 comparisons
 - [ 20 10 30 40 50 60 ] -> 3 comparisons
 - [ 10 20 30 40 50 60 ] -> 2 comparisons
 - [ 10 20 30 40 50 60 ] -> 1 comparisons

--> Sorted. Here is the resulting array - [ 10 20 30 40 50 60 ]

--> Total no. of comparisons : 15
```

Worst Case:

```
Enter the size of array : 6

Enter the elements of the array : 60 50 40 30 20 10

 - [ 60 50 40 30 20 10 ]

 - Sorting the array using Bubble Sort Algorithm....

 - [ 50 40 30 20 10 60 ] -> 5 comparisons
 - [ 40 30 20 10 50 60 ] -> 4 comparisons
 - [ 30 20 10 40 50 60 ] -> 3 comparisons
 - [ 20 10 30 40 50 60 ] -> 2 comparisons
 - [ 10 20 30 40 50 60 ] -> 1 comparisons

--> Sorted. Here is the resulting array - [ 10 20 30 40 50 60 ]

--> Total no. of comparisons : 15
```

Graph :


Bubble Sort
SORTED
REVERSE
RANDOM
1E+13
8E+12
6E+12
4E+12
2E+12
0
-2E+12
Time. (in nano sec.)
-200000
0
200000
400000
600000
800000
1000000
1200000
No. of elements.

Practical #09

Write a Program to implement Insertion Sort. Find the number of comparisons during each pass and display the intermediate result. Use the observed values to plot a graph to analyse the complexity of algorithm.

Code :

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

void display(int ar[], int v)
{
    cout << " - [ ";
    for (int i = 0; i < v; i++)
    {
        cout << ar[i] << " ";
    }
    cout << "]";
}

int insertionSort(int *ar, int v)
{
    int i, j, key, totalCount = 0, count = 0;
    for (i = 1; i < v; i++)
    {
        key = ar[i];
        j = i - 1;
        count = 0;
        while (j >= 0 && key < ar[j])
```

```
        {
            ++count;
            ar[j + 1] = ar[j];
            j = j - 1;
        }
        totalCount += count;
        display(ar, v);
        cout << " -> " << (count) << " comparisons" << endl;
        ar[j + 1] = key;
    }

    return totalCount;
}

int main()
{
    int *arr;
    int size, comparisons;

    cout << endl
         << endl
         << "Enter the size of array : ";
    cin >> size;

    cout << endl
         << "Enter the elements of the array : ";
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    cout << endl;
```

```
    display(arr, size);


    cout << endl
         << endl
         << " - Sorting the array using Insertion Sort Algorithm....
" << endl
         << endl;
    comparisons = insertionSort(arr, size);


    cout << endl
         << "--> Sorted. Here is the resulting array";
    display(arr, size);
    cout << endl
         << endl
         << "--> Total no. of comparisons : " << comparisons;
    cout << endl
         << endl;


    return 0;
}
```

Output : Average Case:

```
Enter the size of array : 6

Enter the elements of the array : 50 20 40 60 10 30

 - [ 50 20 40 60 10 30 ]

 - Sorting the array using Insertion Sort Algorithm....

 - [ 50 50 40 60 10 30 ] -> 1 comparisons
 - [ 20 50 50 60 10 30 ] -> 1 comparisons
 - [ 20 40 50 60 10 30 ] -> 0 comparisons
 - [ 20 20 40 50 60 30 ] -> 4 comparisons
 - [ 10 20 40 40 50 60 ] -> 3 comparisons

--> Sorted. Here is the resulting array - [ 10 20 30 40 50 60 ]

--> Total no. of comparisons : 9
```

Worst Case:

```
Enter the size of array : 6

Enter the elements of the array : 60 50 40 30 20 10

 - [ 60 50 40 30 20 10 ]

 - Sorting the array using Insertion Sort Algorithm....

 - [ 60 60 40 30 20 10 ] -> 1 comparisons
 - [ 50 50 60 30 20 10 ] -> 2 comparisons
 - [ 40 40 50 60 20 10 ] -> 3 comparisons
 - [ 30 30 40 50 60 10 ] -> 4 comparisons
 - [ 20 20 30 40 50 60 ] -> 5 comparisons

--> Sorted. Here is the resulting array - [ 10 20 30 40 50 60 ]

--> Total no. of comparisons : 15
```

Graph:

Practical #10

Write a Program that generates all the permutations of a given set of digits, with or without repetition. (For example, if the given set is {1,2}, the permutations are 12 and 21).

Code :

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

void printArray(int *ar, int v)
{
    cout << " {";
    for (int i = 0; i < v; i++)
    {
        if (i == 0)
            cout << ar[i];
        else
            cout << "," << ar[i];
        ;
    }
    cout << "}";
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
```

```
}

void permutation(int *arr, int n, int curr_i, int *permut_arr, int p
ermut_i)
{
    if (curr_i >= n)
    {
        printArray(permut_arr, permut_i);
    }
    for (int i = curr_i; i < n; i++)
    {
        swap(arr[curr_i], arr[i]);
        permut_arr[permut_i] = arr[curr_i];
        permutation(arr, n, curr_i + 1, permut_arr, permut_i + 1);
        swap(arr[curr_i], arr[i]);
    }
}

int main()
{
    int *arr, pArr[100];
    int size;

    cout << endl
         << endl
         << "Enter the no. of digits : ";
    cin >> size;

    cout << endl
         << "Enter the digits : ";
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
```

```
    }
    cout << endl
         << "--> Given set of digits :";
    printArray(arr, size);
    cout << endl
         << endl
         << "--> All possible permutations of digits :";
    permutation(arr, size, 0, pArr, 0);
    cout << endl
         << endl;

    return 0;
}
```

Output :

```
Enter the no. of digits : 4

Enter the digits : 2 5 8 4

--> Given set of digits : {2,5,8,4}

--> All possible permutations of digits : {2,5,8,4} {2,5,4,8} {2,8,5,4} {2,8,4,5} {2,4,8,5} {2,4,5,8} {5,2,8,4} {5,2,4,8} {5,8,2,4}
 {5,8,4,2} {5,4,8,2} {5,4,2,8} {8,5,2,4} {8,5,4,2} {8,2,5,4} {8,2,4,5} {8,4,2,5} {8,4,5,2} {4,5,8,2} {4,5,2,8} {4,8,5,2} {4,8,2,5}
{4,2,8,5} {4,2,5,8}
```

Practical #11

Write a Program to calculate Permutation and Combination for an input value n and r using recursive formula of n Cr and n Pr .

Code :

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

int nPr(int n, int r)
{
    if (r > n)
        return -1;
    else if (r == 0)
        return 1;
    else
        return (n * nPr(n - 1, r - 1));
}
int nCr(int n, int r)
{
    if (r > n)
        return -1;
    else if (n == 0 || r == 0 || n == r)
        return 1;
    else
        return (nCr(n - 1, r - 1) + nCr(n - 1, r));
}

int main()
```

```
{
    int n, r, p, c;
    cout << endl
         << endl
         << "Enter the value of n : ";
    cin >> n;
    cout << "Enter the value of r : ";
    cin >> r;
    p = nPr(n, r);
    c = nCr(n, r);

    cout << endl
         << "--> Given : n=" << n << ", r=" << r << endl;
    if (p == -1 || c == -1)
        cout
            << endl
            << "--> Invalid input!" << endl
            << endl;
    else
    {
        cout << endl
             << "--> Permutations (nPr) : " << p << endl;
        cout << endl
             << "--> Combinations (nCr) : " << c << endl
             << endl;
    }

    return 0;
}
```

Output :

```
Enter the value of n : 5
Enter the value of r : 3

--> Given : n=5, r=3

--> Permutations (nPr) : 60

--> Combinations (nCr) : 10
```

```
Enter the value of n : 6
Enter the value of r : 8

--> Given : n=6, r=8

--> Invalid input!
```

Practical #12

For any number n, write a program to list all the solutions of the equation x1 + x2 + x3 + ...+ xn = C, where C is a constant (C<=10) and x1, x2,x3,...,xn are nonnegative integers using brute force strategy.

Code :

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

int nCr(int n, int r)
{
    if (r > n)
        return -1;
    else if (n == 0 || r == 0 || n == r)
        return 1;
    else
        return (nCr(n - 1, r - 1) + nCr(n - 1, r));
}
void printEq(int n, int c)
{
    cout << "\nGiven equation : ";
    for (int i = 0; i < n; i++)
    {
        if (i == 0)
            cout << "x" << i + 1;
        else if (i == n - 1)
            cout << " + x" << i + 1 << " = " << c;
        else
            cout << " + x" << i + 1;
    }
```

```
    cout << endl;
}

int main()
{
    int n, c;
    cout << "\nEnter the no of variables (n) : ";
    cin >> n;
    cout << "\nEnter the value of total sum i.e. c <=10 : ";
    cin >> c;
    printEq(n, c);
    cout << "\n--
> Number of possible solutions of this eq. is " << nCr(n + c - 1, c)
 << "\n";

    return 0;
}
```

Output :

```
Enter the no of variables (n) : 5

Enter the value of total sum i.e. c <=10 : 7

Given equation : x1 + x2 + x3 + x4 + x5 = 7

--> Number of possible solutions of this eq. is 330
```

Practical #13

Write a Program to accept the truth values of variables x and y, and print the truth table of the following logical operations:

a) Conjunction
b) Disjunction
c) Exclusive OR
d) Conditional
e) Bi-conditional
f) Exclusive NOR
g) Negation
h) NAND
i) NOR

Code :

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

int main()
{
    int r;
    int x[4], y[4], con[4], dis[4], xOR[4], cond[4], bicond[4], nand
[4], nor[4], nx[4], ny[4];
    cout << "\nEnter the number of propositions i.e. rows : ";
    cin >> r;
    cout << "\nEnter the truth values of x and y (T=1,F=0) : \n";
    for (int i = 0; i < r; i++)
    {
        cout << "--> " << i + 1 << ". ";
        cin >> x[i] >> y[i];
        con[i] = x[i] & y[i];
        dis[i] = x[i] | y[i];
        xOR[i] = (x[i] & (!y[i])) | (y[i] & (!x[i]));
```

```
        cond[i] = (!x[i]) | y[i];
        bicond[i] = (x[i] & y[i]) | ((!x[i]) & (!y[i]));
        nand[i] = !con[i];
        nor[i] = !dis[i];
        nx[i] = !x[i];
        ny[i] = !y[i];
    }
    cout << "\n    (x,y) | Conjunction | Disjunction | XOR | Conditi
onal | Biconditional | NAND | NOR | Negation(x) | Negation(y)\n";
    for (int i = 0; i < r; i++)
    {
        cout << "\n--> (" << x[i] << "," << y[i] << ") |      "
             << con[i] << "       |       "
             << dis[i] << "       |   "
             << xOR[i] << "   |       "
             << cond[i] << "       |        "
             << bicond[i] << "         |   "
             << nand[i] << "    |   "
             << nor[i] << "   |       "
             << nx[i] << "       |       " << ny[i];
    }
    cout << endl;
    return 0;
}
```

Output :

```
Enter the truth values of x and y (T=1,F=0) :
--> 1. 0 1
--> 2. 0 0
--> 3. 1 0
--> 4. 1 1

    (x,y) | Conjunction | Disjunction | XOR | Conditional | Biconditional | NAND | NOR | Negation(x) | Negation(y)

--> (0,1) |     0     |     1     | 1 |     1     |     0     | 1 | 0 |     1     |     0
--> (0,0) |     0     |     0     | 0 |     1     |     1     | 1 | 1 |     1     |     1
--> (1,0) |     0     |     1     | 1 |     0     |     0     | 1 | 0 |     0     |     1
--> (1,1) |     1     |     1     | 0 |     1     |     1     | 0 | 0 |     0     |     0
```

Practical #14

Write a program to accept an input n from the user and graphically represent the values of T(n) where n varies from 0 to n for the recurrence relations.

For e.g. T(n) = T(n-1) + n, T(0) = 1, T(n) = T(n-1) + n^2, T(0) =1, T(n) = 2*T(n)/2 + n, T(1)=1.

Code :

```
#include <iostream>
#include <math.h>
using namespace std;

int Ta(int n)
{
    if (n == 0)
        return 1;
    else
        return (n + Ta(n - 1));
}
int Tb(int n)
{
    if (n == 0)
        return 1;
    else
        return (pow(n, 2) + Tb(n - 1));
}
int Tc(int n)
{

    if (n == 0)
```

```
        return 1;
    else
        return (n + (2 * Tc(n / 2)));
}

int main()
{
    int n;
    cout << "\nEnter thr value of n : ";
    cin >> n;
    cout << "\n--> for n=" << n << " : \n";
    cout << "\n--> 1. T(n) = T(n-1) + n, T(0) = 1 ==> " << Ta(n);
    cout << "\n--> 1. T(n) = T(n-1) + n^2, T(0) = 1 ==> " << Tb(n);
    cout << "\n--> 1. T(n) = 2*T(n/2) + n, T(0) = 1 ==> " << Tc(n);
    cout << endl;

    return 0;
}
```

Output :

```
Enter thr value of n : 10

--> for n=10 :

--> 1. T(n) = T(n-1) + n, T(0) = 1 ==> 56
--> 1. T(n) = T(n-1) + n^2, T(0) = 1 ==> 386
--> 1. T(n) = 2*T(n/2) + n, T(0) = 1 ==> 52
```

Practical #15

Write a Program to store a function (polynomial/exponential), and then evaluate the polynomial.

(For example, store f(x) = 4n3 + 2n + 9 in an array and for a given value of n, say n = 5, evaluate (i.e. compute the value of f(5)).

Code :

```
#include <iostream>
#include <math.h>
using namespace std;

void printPoly(int *poly, int n)
{
    cout << endl
         << "Given polynomial, f(x) = ";
    for (int i = n; i >= 0; i--)
    {
        if (!poly[i] == 0)
        {
            if (i == n)
                cout << poly[i] << "x^" << i;
            else
            {
                if (poly[i] >= 0)
                    cout << " + ";
                else
                    cout << " ";
                if (!i == 0)
                    cout << poly[i] << "x^" << i;
```

```
                else
                    cout << poly[i];
            }
        }
    }
}
float evaluatePoly(int *poly, int n, int x)
{
    float total;
    for (int i = n; i >= 0; i--)
    {
        total += poly[i] * pow(x, i);
    }
    return total;
}

int main()
{
    int *polynomial, deg, x;
    cout << endl
         << "Enter the degree of polynomial : ";
    cin >> deg;
    polynomial[deg] = {0};
    for (int i = deg; i >= 0; i--)
    {
        if (i == 0)
            cout << "Enter the constant term : ";
        else
            cout << "Enter the coefficient of x^" << i << " : ";
        cin >> polynomial[i];
    }
    printPoly(polynomial, deg);
    cout << endl
         << endl
```

```
         << "Enter the value of x for which the polynomial is to be
evaluated : ";
    cin >> x;


    cout << endl
         << "--
> f(" << x << ") : " << evaluatePoly(polynomial, deg, x);
    cout << endl
         << endl;


    return 0;
}
```

Output :

```
Enter the degree of polynomial : 4
Enter the coefficient of x^4 : 4
Enter the coefficient of x^3 : 2
Enter the coefficient of x^2 : -3
Enter the coefficient of x^1 : 0
Enter the constant term : 2

Given polynomial, f(x) = 4x^4 + 2x^3 -3x^2 + 2

Enter the value of x for which the polynomial is to be evaluated : 2

--> f(2) : 70
```

Practical #16

Write a Program to represent Graphs using the Adjacency Matrices and check if it is a complete graph.

Code :

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

#define N 50

int main()
{
    int adjMatrix[N][N] = {0};
    int v, e, x; //v= no of vertices , e= no of adjacent vertices, x
= adjacent vertex
    int count = 0;
    cout << endl
         << endl
         << "Enter the total number of vertices : ";
    cin >> v;

    for (int i = 1; i <= v; i++)
    {
        cout << "Enter the no. of vertices adjacent to " << i << " :
 ";
        cin >> e;
        cout << "Enter the adjacent vertices : ";
        for (int j = 1; j <= e; j++)
        {
```

```
            cin >> x;
            adjMatrix[i - 1][x - 1] = 1;
            count++;
        }
    }

    for (int i = 0; i < v; i++)
    {
        cout << endl
             << "\t";
        for (int j = 0; j < v; j++)
        {
            cout << adjMatrix[i][j] << " ";
        }
    }

    if (count == ((v * v) - v))
    {

        cout << endl
             << endl
             << "-> This is a Complete Graph" << endl;
    }
    else
    {
        cout << endl
             << endl
             << "-> This is not a Complete Graph" << endl;
    }

    return 0;
}
```

Output :

```
Enter the total number of vertices : 5
Enter the no. of vertices adjacent to 1 : 3
Enter the adjacent vertices : 2 3 5
Enter the no. of vertices adjacent to 2 : 3
Enter the adjacent vertices : 1 3 5
Enter the no. of vertices adjacent to 3 : 3
Enter the adjacent vertices : 1 2 4
Enter the no. of vertices adjacent to 4 : 2
Enter the adjacent vertices : 3 5
Enter the no. of vertices adjacent to 5 : 3
Enter the adjacent vertices : 1 2 4

        0 1 1 0 1
        1 0 1 0 1
        1 1 0 1 0
        0 0 1 0 1
        1 1 0 1 0

-> This is not a Complete Graph
```

Practical #17

Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex.

Code :

```
#include <iostream>
using namespace std;
#define N 25

int main()
{
    int matrix[N][N], deg[N][2] = {0};
    int v;
    char ch;

    cout << "\nEnter the number of vertices : ";
    cin >> v;
    for (int i = 0; i < v; i++)
    {
        cout << endl;
        for (int j = 0; j < v; j++)
        {
            cout << "=> Is there a edge " << i + 1 << "--
> " << j + 1 << " ? (y/n) : ";
            cin >> ch;
            if (ch == 'Y' || ch == 'y')
            {
                matrix[i][j] = 1;
                ++deg[j][0];
```

```
                ++deg[i][1];
            }
            else
                matrix[i][j] = 0;
        }
    }

    cout << "\n--> Given Adjacency Matrix : \n";
    for (int i = 0; i < v; i++)
    {
        cout << "\n\t";
        for (int j = 0; j < v; j++)
        {
            cout << matrix[i][j] << " ";
        }
    }

    cout << "\n\n--> Vertex | In-degree | Out-degree";
    for (int i = 0; i < v; i++)
    {
        cout << "\n--
>   " << i + 1 << "    |     " << deg[i][0] << "     |     " << deg[
i][1];
    }
    cout << endl;

    return 0;
}
```

Output :

```
Enter the number of vertices : 3

=> Is there a edge 1--> 1 ? (y/n) : y
=> Is there a edge 1--> 2 ? (y/n) : y
=> Is there a edge 1--> 3 ? (y/n) : y

=> Is there a edge 2--> 1 ? (y/n) : n
=> Is there a edge 2--> 2 ? (y/n) : n
=> Is there a edge 2--> 3 ? (y/n) : y

=> Is there a edge 3--> 1 ? (y/n) : n
=> Is there a edge 3--> 2 ? (y/n) : y
=> Is there a edge 3--> 3 ? (y/n) : n

--> Given Adjacency Matrix :

        1 1 1
        0 0 1
        0 1 0

--> Vertex | In-degree | Out-degree
-->   1    |     1     |     3
-->   2    |     2     |     1
-->   3    |     2     |     1
```

Practical #18

Given a graph G, Write a Program to find the number of paths of length n between the source and destination entered by the user.

Code :

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

#define N 10

// Function to multiply two matrices
void multiply(int a[][N], int b[][N], int res[][N], int v)
{
    int mul[N][N];
    for (int i = 0; i < v; i++)
    {
        for (int j = 0; j < v; j++)
        {
            mul[i][j] = 0;
            for (int k = 0; k < v; k++)
                mul[i][j] += a[i][k] * b[k][j];
        }
    }

    // Storing the multiplication result in res[][]
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++)
            res[i][j] = mul[i][j];
}
```

```
// Function to compute G raised to the power n
void power(int G[N][N], int res[N][N], int n, int v)
{

    // Base condition
    if (n == 1)
    {
        for (int i = 0; i < v; i++)
            for (int j = 0; j < v; j++)
                res[i][j] = G[i][j];
        return;
    }

    // Recursion call for first half
    power(G, res, n / 2, v);

    // Multiply two halves
    multiply(G, G, res, v);

    // If n is odd
    if (n % 2 != 0)
        multiply(res, G, res, v);
}

int main()
{
    int n, source, dest, res[N][N];
    int G[N][N];
    int v;
    char ch;

    cout << "\nEnter the number of vertices : ";
    cin >> v;
```

```
    for (int i = 0; i < v; i++)
    {
        cout << endl;
        for (int j = 0; j < v; j++)
        {
            cout << "=> Is there a edge " << i + 1 << "--
> " << j + 1 << " ? (y/n) : ";
            cin >> ch;
            if (ch == 'Y' || ch == 'y')
            {
                G[i][j] = 1;
            }
            else
                G[i][j] = 0;
        }
    }

    cout << "\n--> Given Adjacency Matrix : \n";
    for (int i = 0; i < v; i++)
    {
        cout << "\n\t";
        for (int j = 0; j < v; j++)
        {
            cout << G[i][j] << " ";
        }
    }

    cout << "\n\nEnter the path length : ";
    cin >> n;
    cout << "Enter the source : ";
    cin >> source;
    cout << "Enter the destination : ";
    cin >> dest;
```

```
    power(G, res, n, v);

    cout << "\n--> Number of paths of length " << n
         << " from <" << source << "> to <" << dest << "> : "
         << res[source - 1][dest - 1] << "\n";

    return 0;
}
```

Output :

```
Enter the number of vertices : 3

=> Is there a edge 1--> 1 ? (y/n) : y
=> Is there a edge 1--> 2 ? (y/n) : y
=> Is there a edge 1--> 3 ? (y/n) : y

=> Is there a edge 2--> 1 ? (y/n) : n
=> Is there a edge 2--> 2 ? (y/n) : n
=> Is there a edge 2--> 3 ? (y/n) : y

=> Is there a edge 3--> 1 ? (y/n) : n
=> Is there a edge 3--> 2 ? (y/n) : y
=> Is there a edge 3--> 3 ? (y/n) : n

--> Given Adjacency Matrix :

       1 1 1
       0 0 1
       0 1 0

Enter the path length : 2
Enter the source : 1
Enter the destination : 3

--> Number of paths of length 2 from <1> to <3> : 2
```

Practical #19

Given an adjacency matrix of a graph, write a program to check whether a given set of vertices {v1,v2,v3.....,vk} forms an Euler path / Euler Circuit (for circuit assume vk=v1).

Code :

```
#include <iostream>
using namespace std;
#define N 25

int main()
{
    int **matrix, *deg, *vertices;
    int v, n, sum = 0, flag = 1, count = 0;
    cout << "\nEnter the dimension of matrix : ";
    cin >> v;

    matrix = new int *[v];
    deg = new int[v];

    for (int i = 0; i < v; i++)
    {
        matrix[i] = new int[v];
        cout << "Enter the elements in row " << i + 1 << " : ";
        for (int j = 0; j < v; j++)
        {
            cin >> matrix[i][j];
        }
```

```
    }
    cout << "\n--> Adjacency Matrix \n ";
    for (int m = 0; m < v; m++)
    {
        cout << "\n    ";
        for (int n = 0; n < v; n++)
            cout << matrix[m][n] << " ";
    }

    for (int i = 0; i < v; i++)
    {
        sum = 0;
        for (int j = 0; j < v; j++)
        {
            sum += matrix[i][j];
        }
        deg[i] = sum;
    }
    cout << "\n\nEnter the no of vertices followed by the vertices t
o be checked : ";
    cin >> n;
    vertices = new int[n];

    for (int i = 0; i < n; i++)
    {
        cin >> vertices[i];
    }
    for (int i = 0; i < n; i++)
    {
        cout << "\n--
> Degree of Vertex " << vertices[i] << " : " << deg[vertices[i] - 1]
;
    }
    for (int i = 0; i < n; i++)
```

```
    {
        if ((deg[vertices[i] - 1] % 2) != 0)
        {
            flag = 0;
            count++;
        }
    }
    if (flag == 1)
        cout << "\n\n--> There is an euler circuit.";
    else
        cout << "\n\n--> There is no euler circuit.";
    if (count == 2 || flag == 1)
        cout << "\n\n--> There is an euler path.";
    else
        cout << "\n\n--> There is no euler path.";
    cout << endl;
    return 0;
}
```

Output :

```
Enter the dimension of matrix : 4
Enter the elements in row 1 : 0 1 1 1
Enter the elements in row 2 : 1 0 1 0
Enter the elements in row 3 : 1 1 0 1
Enter the elements in row 4 : 1 0 1 0

--> Adjacency Matrix

    0 1 1 1
    1 0 1 0
    1 1 0 1
    1 0 1 0

Enter the no of vertices followed by the vertices to be checked : 3 1 2 3

--> Degree of Vertex 1 : 3
--> Degree of Vertex 2 : 2
--> Degree of Vertex 3 : 3

--> There is no euler circuit.

--> There is an euler path.
```

Practical #20

Given a full m-ary tree with i internal vertices, Write a Program to find the number of leaf nodes.

Code :

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

int calcLeafNodes(int m, int i)
{
    return i * (m - 1) + 1;
}

int main()
{
    int m, i;
    cout << endl
         << endl
         << "Enter the degree of tree : ";
    cin >> m;
    cout << "Enter the number of internal nodes : ";
    cin >> i;
    cout << endl
         << "--
> Number of leaf nodes : " << calcLeafNodes(m, i) << endl
         << endl;
    return 0;
}
```

Output :

```
Enter the degree of tree : 3
Enter the number of internal nodes : 11

--> Number of leaf nodes : 23
```