1. Binary Search

```
"// Online Java Compiler
// Use this editor to write, compile and run your Java code online


import java.time.Instant;
import java.util.concurrent.TimeUnit;
import java.util.Scanner;
class BinarySearchExample{
  public static void binarySearch(int arr[], int first, int last, int key){
      int mid = (first + last)/2;
      while( first <= last ){
            if ( arr[mid] < key ){
                first = mid + 1;
            }else if ( arr[mid] == key ){
                System.out.println("Element is found at index: " + mid);
                break;
            }else{
                  last = mid - 1;
            }
            mid = (first + last)/2;
      }
      if ( first > last ){
            System.out.println("Element is not found!");
      }
  }
  public static void main(String args[]){


                    //int arr[] = {10,20,30,40,50};
```

```
            //int key = 50 ;
            Scanner keyboard = new Scanner(System.in);
System.out.println("enter     key you want to search");
int key = keyboard.nextInt();


          int n;
          Scanner sc=new Scanner(System.in);
System.out.print("Enter the number of elements you want to store: ");
//reading the number of elements from the that we want to enter
n=sc1.nextInt();
//creates an array in the memory of length 10
int[] arr = new int[10];
int last=arr.length-1;
System.out.println("Enter the elements of the array: ");
for(int i=0; i<n; i++)
{
//reading array elements from the user
arr[i]=sc.nextInt();
}
long startTime = System.nanoTime();
            binarySearch(arr,0,last,key);


long timeElapsed = System.nanoTime() - startTime;


          System.out.println("Elapsed time in in nanosecond: " + timeElapsed);
}
}
```

2. Quicksort

```
import java.util.Random;

import java.util.Scanner;

public class quicksort {

static int max=2000;

int partition (int[] a, int low,int high)

{

int p,i,j,temp;

p=a[low];

i=low+1;

j=high;

while(low<high)

{

while(a[i]<=p&&i<high)

i++;

while(a[j]>p)

j--;

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

else

{

temp=a[low];

a[low]=a[j];

a[j]=temp;
```

```
return j;
}
}
return j;
}
void sort(int[] a,int low,int high)
{
if(low<high)
{
int s=partition(a,low,high);
sort(a,low,s-1);
sort(a,s+1,high);
}
}
public static void main(String[] args) {
// TODO Auto-generated method stub

int[] a;
int i;
System.out.println("Enter the array size");
Scanner sc =new Scanner(System.in);
int n=sc.nextInt();
a= new int[max];
Random generator=new Random();
for( i=0;i<n;i++)
a[i]=generator.nextInt(20);
System.out.println("Array before sorting");
for( i=0;i<n;i++)
System.out.println(a[i]+" ");
```

```
long startTime=System.nanoTime();
quicksort m=new quicksort();
m.sort(a,0,n-1);
long stopTime=System.nanoTime();
long elapseTime=(stopTime-startTime);
System.out.println("Time taken to sort array is:"+elapseTime+"nano seconds");
System.out.println("Sorted array is\n");
for(i=0;i<n;i++)
System.out.println(a[i]);
}
}
```

3. Mergesort

```
import java.util.Scanner;

public class MSort

{



  public static void merge(int a[],int l,int m,int h)

     {

            int i, j,c=l;

            int b[]=new int[h+1];


     for(i = l,j = m+1; i<=m && j<=h; c++)

                    {


                           if(a[i] <= a[j])

                           b[c] = a[i++];

                          else

                          b[c] = a[j++];

               }

       while(i <= m )

                          b[c++] = a[i++];


                       while(j<=h)

                          b[c++] = a[j++];


       for(i = l ; i <= h; i++)

                                        a[i] = b[i];

     }
```

```
public static void Sort(int a[],int l,int h)
{
        if(l<h)
       {
           int m=(l+h)/2;
           Sort(a,l,m);
           Sort(a,m+1,h);
           merge(a,l,m,h);

        }



}

public static void printarray(int a[])
{
        for(int i=0; i < a.length; i++)
         {

         System.out.print(a[i]+"     ");
         }


}
public static void main(String[] args)
{
       int n, res,i;
       Scanner s = new Scanner(System.in);
       System.out.print("Enter number of elements in the array:");
```

```
            n = s.nextInt();
            int a[] = new int[n];
            System.out.println("Enter "+n+" elements ");
            for( i=0; i < n; i++)
            {
                a[i] = s.nextInt();
            }

            System.out.println("elements in array ");
            printarray(a);
             Sort(a,0,n-1);
            System.out.println( "\nelements after sorting");
             printarray(a);
             }
        }
```

<u>program output:</u>

```
Enter number of elements in the array:5
Enter 5 elements
12
4
0
5
3
elements in array
12    4    0    5    3
elements after sorting
0    3    4    5    12
```

4a. Knapsack

```
import java.util.Scanner;
class DKnapsack
{
int n;
int c;
int p[];
int w[];
int v[][];
public DKnapsack(int n, int c, int[] p, int[] w)
{
super();
this.n = n;
this.c = c;
this.p = p;
this.w = w;
this.v = new int[n + 1][c + 1];
}
void compute()
{
for ( int i = 0; i <= n; ++ i)
{
for ( int j = 0; j <= c; ++ j)
{
if ( i == 0 || j == 0 )
{
v[i][j] = 0;
}
else if ( j - w[i] >= 0 )
```

```
{
v[i][j] = max ( v[i - 1][j], p[i] + v[i - 1][j - w[i]]);
}
else if ( j - w[i] < 0 )
{
v[i][j] = v[i - 1][j];
}
}
}

System.out.println("Optimal Solution: " + v[n][c]);
traceback();
}
void traceback()
{
System.out.println("The objects picked up into knapsack are:");
int i = n;
int j = c;
while( i > 0)
{
if(v[i][j] != v[i-1][j])
{
System.out.print(i + " ");
j = j - w[i];
i--;
}
else
{
i--;
}
```

```
}
}
private int max(int i, int j)
{
if ( i > j ) return i;
else return j;
}
}
public class KpDynamic
{
public static void main(String[] args)
{
int n;
int c;
Scanner input = new Scanner(System.in);
System.out.println("Enter number of objects");
n = input.nextInt();
int[] p = new int[n+1];
int[] w = new int[n+1];
int i;

System.out.println("Enter capacity of Knapsack");
c = input.nextInt();
System.out.println("Enter profit for each " + n + " objects");
for ( i = 1; i <= n; i ++)
p[i] = input.nextInt();
System.out.println("Enter weight for each " + n + " objects");
for ( i = 1; i <= n; i ++)
w[i] = input.nextInt();
DKnapsack dk = new DKnapsack(n, c, p, w);
```

dk.compute();

}

}

OUTPUT:

Enter number of objects

5

Enter capacity of Knapsack

20

Enter profit for each 5 objects

3458 10

Enter weight for each 5 objects

23459

Optimal Solution: 26

The objects picked up into knapsack are:

- **4 3 1**

4b. Greedy method.

```
import java.util.Scanner;
class GKnapsack
{
int n;
double c;
double p[];
double w[];
public GKnapsack(int n, double c, double[] p, double[] w)
{
super();
this.n = n;
this.c = c;this.p = p;
this.w = w;
}
void compute()
{
int i;
double[] x= new double[n+1];
for (i=0; i<n; i++)
{
x[i] = 0.0;
}
double rc = c;
for(i=0; i<n; i++)
{
if(w[i] > rc) break;
x[i] = 1;
rc = rc - w[i];
```

```
        }
        if(i<=n)
        {
        x[i] = rc/w[i];
        }
        double netProfit = 0.0;
for ( i = 0; i < n; ++ i)
{
if ( x[i] > 0.0)
{
netProfit = netProfit + x[i] * p[i];
}
}
System.out.println("Net Profit: " + netProfit);
System.out.println("The objects picked up into knapsack are:");
for ( i = 0; i < n; ++ i)
{
System.out.println(x[i] + " ");
}
}
}
public class KpGreedy
{
public static void main(String[] args)
{
int n;
double c;
Scanner input = new Scanner(System.in);
System.out.println("Enter number of objects");
```

```
n = input.nextInt();
double[] p = new double[n+1];
double[] w = new double[n+1];
int i;
System.out.println("Enter capacity of Knapsack");
c = input.nextDouble();
System.out.println("Enter profit for each " + n + " objects");
for ( i = 0; i < n; i ++)
p[i] = input.nextDouble();
System.out.println("Enter weight for each " + n + " objects");
for ( i = 0; i < n; i ++)
w[i] = input.nextDouble();
GKnapsack gk = new GKnapsack(n, c, p, w);
gk.compute();
}
}
```

5.Design a program to print all the node reachable from a given starting node in a given digraph using DFS method.

```
// DFS algorithm in Java
import java.util.*;
public class Graph {
   private LinkedList<Integer> adjLists[];
   private boolean visited[];
   // Graph creation
   Graph(int vertices) {
```

```
    adjLists = new LinkedList[vertices];
    visited = new boolean[vertices];
    for (int i = 0; i < vertices; i++)
        adjLists[i] = new LinkedList<Integer>();
}
// Add edges
void addEdge(int src, int dest) {
    adjLists[src].add(dest);
}
// DFS algorithm
void DFS(int vertex) {
    visited[vertex] = true;
    System.out.print(vertex + " ");
    Iterator<Integer> ite = adjLists[vertex].listIterator();
    while (ite.hasNext()) {
        int adj = ite.next();
        if (!visited[adj])
            DFS(adj);
    }
}
public static void main(String args[]) {
    Graph g = new Graph(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 3);
    g.addEdge(3, 0);
    long startTime=System.nanoTime();
    System.out.println("Following is Depth First Traversal");
```

```
        g.DFS(1);
        long stopTime=System.nanoTime();
        long elapseTime=(stopTime-startTime);
        System.out.println("Time taken to sort array is:"+elapseTime+"nano seconds");
    }
}
```

Output:

- **For starting vertex2**

Following is Depth First Traversal

2 3 0 1

Time taken to sort array is:26333916nano seconds

- **For starting vertex1**

Following is Depth First Traversal

1 3 0 2 Time taken to sort array is:79836064nano second

PART-B

1b.Write a Program find shortest paths to other vertices using Dijkstra's algorithm.

```
import java.util.Arrays;
import java.util.Scanner;
public class Dijkstra
{
static int n,cost[][],i,j,u,dist[],src;
void dij(int src,int cost[][],int dist[],int n)
{
int visited[],min;
visited=new int[n];
for(i=0;i<n;i++)
{
visited[i]=0;
dist[i]=cost[src][i];
}
visited[src]=1;
dist[src]=0;
for(i=0;i<n;i++)
{
if(i==src) continue;
min=999;
for(j=0;j<n;j++)
if((visited[j]==0)&&(min>dist[j]))
{
min=dist[j];
u=j;
}
visited[u]=1;
for(j=0;j<n;j++)
if(visited[j]==0)
```

```
{
if(dist[j]>dist[u]+cost[u][j])
dist[j]=dist[u]+cost[u][j];
}
}
}

{
Scanner sc=new Scanner(System.in);
System.out.println("Enter the number of vertices");
n=sc.nextInt();
System.out.println("Enter the matrix");
cost=new int[n][n];
dist=new int[n];
Arrays.fill(dist,0);
for(i=0;i<n;i++)
for(j=0;j<n;j++)
cost[i][j]=sc.nextInt();
System.out.println("Enter the source vertex");
src=sc.nextInt();
new Dijkstra().dij(src, cost, dist, n);
System.out.println("Shortest path from "+src+" to all other vertices");
for(i=0;i<n;i++)
System.out.println("To " +i+" is "+dist[i]);
}
}
```

OUTPUT:

```
Enter the number of vertices
4
Enter the matrix
0 15 10 9999
9999 0 15 9999
```

20 9999 0 20

9999 10 9999 0

Enter the source vertex

2

Shortest path from 2 to all other vertices

To 0 is 20

To 1 is 30

To 2 is 0

To 3 is 20

PART B

2a. A Java program for Prim's Minimum Spanning Tree (MST) algorithm.

```
// The program is for adjacency matrix representation of the graph


import java.util.*;

import java.lang.*;

import java.io.*;


class MST {

	// Number of vertices in the graph

	private static final int V = 5;


	// A utility function to find the vertex with minimum key

	// value, from the set of vertices not yet included in MST

	int minKey(int key[], Boolean mstSet[])

	{

		// Initialize min value

		int min = Integer.MAX_VALUE, min_index = -1;


		for (int v = 0; v < V; v++)

			if (mstSet[v] == false && key[v] < min) {

				min = key[v];

				min_index = v;

			}
```

```
        return min_index;
}

// A utility function to print the constructed MST stored in
// parent[]
void printMST(int parent[], int graph[][])
{
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++)
                System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
}

// Function to construct and print MST for a graph represented
// using adjacency matrix representation
void primMST(int graph[][])
{
        // Array to store constructed MST
        int parent[] = new int[V];

        // Key values used to pick minimum weight edge in cut
        int key[] = new int[V];

        // To represent set of vertices included in MST
        Boolean mstSet[] = new Boolean[V];
```

```
// Initialize all keys as INFINITE

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

        key[i] = Integer.MAX_VALUE;

        mstSet[i] = false;

}


// Always include first 1st vertex in MST.

key[0] = 0; // Make key 0 so that this vertex is

// picked as first vertex

parent[0] = -1; // First node is always root of MST


// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

        // Pick thd minimum key vertex from the set of vertices

        // not yet included in MST

        int u = minKey(key, mstSet);


        // Add the picked vertex to the MST Set

        mstSet[u] = true;


        // Update key value and parent index of the adjacent

        // vertices of the picked vertex. Consider only those

        // vertices which are not yet included in MST

        for (int v = 0; v < V; v++)
```

```
                    // graph[u][v] is non zero only for adjacent vertices of m
                    // mstSet[v] is false for vertices not yet included in MST
                    // Update the key only if graph[u][v] is smaller than key[v]
                    if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
                        parent[v] = u;
                        key[v] = graph[u][v];
                    }
        }

        // print the constructed MST
        printMST(parent, graph);
}

public static void main(String[] args)
{
        /* Let us create the following graph
        2 3
        (0)--(1)--(2)
        | / \ |
        6| 8/ \5 |7
        | /     \ |
        (3)-------(4)
            9           */
        MST t = new MST();
        int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },
```

```
                                        { 2, 0, 3, 8, 5 },
                                        { 0, 3, 0, 0, 7 },
                                        { 6, 8, 0, 0, 9 },
                                        { 0, 5, 7, 9, 0 } };

        // Print the solution
        t.primMST(graph);
    }
}
//output
java -cp /tmp/t0b7IsPHRm MST
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

2b.Java program for Kruskal's algorithm to

```java
// find Minimum Spanning Tree of a given
//connected, undirected and weighted graph
import java.util.*;
import java.lang.*;
import java.io.*;

class Graph {
	// A class to represent a graph edge
	class Edge implements Comparable<Edge>
	{
		int src, dest, weight;

		// Comparator function used for
		// sorting edgesbased on their weight
		public int compareTo(Edge compareEdge)
		{
			return this.weight - compareEdge.weight;
		}
	};

	// A class to represent a subset for
	// union-find
	class subset
	{
```

```
	int parent, rank;
};

int V, E; // V-> no. of vertices & E->no.of edges
Edge edge[]; // collection of all edges

// Creates a graph with V vertices and E edges
Graph(int v, int e)
{
	V = v;
	E = e;
	edge = new Edge[E];
	for (int i = 0; i < e; ++i)
		edge[i] = new Edge();
}

// A utility function to find set of an
// element i (uses path compression technique)
int find(subset subsets[], int i)
{
	// find root and make root as parent of i
	// (path compression)
	if (subsets[i].parent != i)
		subsets[i].parent
			= find(subsets, subsets[i].parent);
```

```
        return subsets[i].parent;
}

// A function that does union of two sets
// of x and y (uses union by rank)
void Union(subset subsets[], int x, int y)
{
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        // Attach smaller rank tree under root
        // of high rank tree (Union by Rank)
        if (subsets[xroot].rank
                < subsets[yroot].rank)
                subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank
                        > subsets[yroot].rank)
                subsets[yroot].parent = xroot;

        // If ranks are same, then make one as
        // root and increment its rank by one
        else {
                subsets[yroot].parent = xroot;
                subsets[xroot].rank++;
```

```
        }
    }

    // The main function to construct MST using Kruskal's
    // algorithm
    void KruskalMST()
    {
        // This will store the resultant MST
        Edge result[] = new Edge[V];

        // An index variable, used for result[]
        int e = 0;

        // An index variable, used for sorted edges
        int i = 0;
        for (i = 0; i < V; ++i)
            result[i] = new Edge();

        // Step 1: Sort all the edges in non-decreasing
        // order of their weight. If we are not allowed to
        // change the given graph, we can create a copy of
        // array of edges
        Arrays.sort(edge);

        // Allocate memory for creating V subsets
```

```
subset subsets[] = new subset[V];
for (i = 0; i < V; ++i)
	subsets[i] = new subset();

// Create V subsets with single elements
for (int v = 0; v < V; ++v)
{
	subsets[v].parent = v;
	subsets[v].rank = 0;
}

i = 0; // Index used to pick next edge

// Number of edges to be taken is equal to V-1
while (e < V - 1)
{
	// Step 2: Pick the smallest edge. And increment
	// the index for next iteration
	Edge next_edge = edge[i++];

	int x = find(subsets, next_edge.src);
	int y = find(subsets, next_edge.dest);

	// If including this edge doesn't cause cycle,
	// include it in result and increment the index
```

```java
                // of result for next edge
                if (x != y) {
                        result[e++] = next_edge;
                        Union(subsets, x, y);
                }
                // Else discard the next_edge
        }

        // print the contents of result[] to display
        // the built MST
        System.out.println("Following are the edges in "
                                + "the constructed MST");
        int minimumCost = 0;
        for (i = 0; i < e; ++i)
        {
                System.out.println(result[i].src + " -- "
                                        + result[i].dest
                                        + " == " + result[i].weight);
                minimumCost += result[i].weight;
        }
        System.out.println("Minimum Cost Spanning Tree "
                                + minimumCost);
}

// Driver Code
```

```
public static void main(String[] args)

{

        /* Let us create following weighted graph

                    10

            0--------1

            | \        |

        6| 5\ |15

            |         \ |

            2--------3

                    4          */

        int V = 4; // Number of vertices in graph

        int E = 5; // Number of edges in graph

        Graph graph = new Graph(V, E);

        // add edge 0-1

        graph.edge[0].src = 0;

        graph.edge[0].dest = 1;

        graph.edge[0].weight = 10;

        // add edge 0-2

        graph.edge[1].src = 0;

        graph.edge[1].dest = 2;

        graph.edge[1].weight = 6;
```

```
            // add edge 0-3
            graph.edge[2].src = 0;
            graph.edge[2].dest = 3;
            graph.edge[2].weight = 5;

            // add edge 1-3
            graph.edge[3].src = 1;
            graph.edge[3].dest = 3;
            graph.edge[3].weight = 15;

            // add edge 2-3
            graph.edge[4].src = 2;
            graph.edge[4].dest = 3;
            graph.edge[4].weight = 4;

            // Function call
            graph.KruskalMST();
        }
}
//output
java -cp /tmp/t0b7IsPHRm Graph
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
```

Minimum Cost Spanning Tree 19

PART B

3. Write a program to

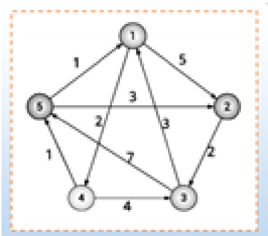
(a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

```
import java.util.*;
public class Floyds
{
static int n,i,j,k;
public void floyd(int n , int[][] cost)
{
for(k=1;k<=n;k++)
{
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j]);
}
}
}
System.out.println("all pair shortest paths matrix \n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
System.out.print(cost[i][j]+" ");
}
System.out.println();
}
}
public int min(int i,int j)
{
```

```
if(i<j)
return i;
else
	return j;

public static void main(String[] args)
{
Scanner sc=new Scanner(System.in);
System.out.println("Eneter the no of vertices\n");
n=sc.nextInt();
int cost[][]=new int[n+1][n+1];
System.out.println("Enter the cost matrix:");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
cost[i][j]=sc.nextInt();
Floyds f = new Floyds();
f.floyd(n,cost);
}
}
```

OUTPUT:



Enter the number of vertices

5

Enter the cost matrix:

0 5 999 2 999

999 0 2 999 999

3 999 0 999 7

999 999 4 0 1

1 3 999 999 0

all pair shortest paths matrix

0 5 6 2 3

5 0 2 7 8

3 8 0 5 6

2 4 4 0 1

1 3 5 3 0

(b) Implement transitive closure using warshall Algorithm.

```
/**
 **  Java Program to Implement Warshall Algorithm
 **/

import java.util.Scanner;
```

```
/** Class Warshall **/
public class Warshall
{
    private int V;
    private boolean[][] tc;
    /** Function to make the transitive closure **/
    public void getTC(int[][] graph)
    {
        this.V = graph.length;
        tc = new boolean[V][V];
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
                if (graph[i][j] != 0)
                    tc[i][j] = true;
```

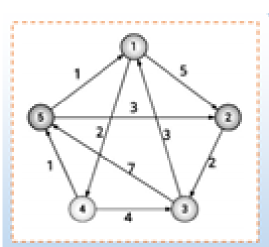
3. Write a program to

(c) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

```
import java.util.*;
public class Floyds
{
static int n,i,j,k;
public void floyd(int n , int[][] cost)
{
for(k=1;k<=n;k++)
{
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j]);
}
}
}
System.out.println("all pair shortest paths matrix \n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
```

```
System.out.print(cost[i][j]+" ");
}
System.out.println();
}
}
public int min(int i,int j)
{
if(i<j)
return i;
else
        return j;
public static void main(String[] args)
{
Scanner sc=new Scanner(System.in);
System.out.println("Eneter the no of vertices\n");
n=sc.nextInt();
int cost[][]=new int[n+1][n+1];
System.out.println("Enter the cost matrix:");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
cost[i][j]=sc.nextInt();
Floyds f = new Floyds();
f.floyd(n,cost);
}
}
```

OUTPUT:



Enter the number of vertices

5

Enter the cost matrix:

0 5 999 2 999

999 0 2 999 999

3 999 0 999 7

999 999 4 0 1

1 3 999 999 0

all pair shortest paths matrix

0 5 6 2 3

5 0 2 7 8

3 8 0 5 6

2 4 4 0 1

1 3 5 3 0

(d) **Implement transitive closure using warshall Algorithm.**

```
/**
 **  Java Program to Implement Warshall Algorithm
 **/

import java.util.Scanner;

/** Class Warshall **/
public class Warshall
{
    private int V;
    private boolean[][] tc;
    /** Function to make the transitive closure **/
    public void getTC(int[][] graph)
    {
        this.V = graph.length;
        tc = new boolean[V][V];
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
                if (graph[i][j] != 0)
                    tc[i][j] = true;
            tc[i][i] = true;
        }
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
            {
                if (tc[j][i])
                    for (int k = 0; k < V; k++)
                       if (tc[j][i] && tc[i][k])
                            tc[j][k] = true;
            }
        }
    }
    /** Funtion to display the trasitive closure **/
    public void displayTC()
    {
        System.out.println("\nTransitive closure :\n");
        System.out.print(" ");
        for (int v = 0; v < V; v++)
           System.out.print("   " + v );
        System.out.println();
        for (int v = 0; v < V; v++)
        {
            System.out.print(v +" ");
            for (int w = 0; w < V; w++)
            {
                if (tc[v][w])
                    System.out.print("  * ");
                else
                    System.out.print("    ");
            }
            System.out.println();
```

```
        }
    }

    /** Main function **/
    public static void main (String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Warshall Algorithm Test\n");
        /** Make an object of Warshall class **/
        Warshall w = new Warshall();

        /** Accept number of vertices **/
        System.out.println("Enter number of vertices\n");
        int V = scan.nextInt();

        /** get graph **/
        System.out.println("\nEnter matrix\n");
        int[][] graph = new int[V][V];
        for (int i = 0; i < V; i++)
            for (int j = 0; j < V; j++)
                graph[i][j] = scan.nextInt();
         w.getTC(graph);
        w.displayTC();
    }
}
```

```
Warshall Algorithm Test

Enter number of vertices

6

Enter matrix

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

Transitive closure :

    0   1   2   3   4   5
0   *   *       *   *   *
1       *
2   *   *   *   *   *   *
3               *
4               *   *
5               *   *   *
```

```
            tc[i][i] = true;
        }
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
```

```
            {
                if (tc[j][i])
                    for (int k = 0; k < V; k++)
                       if (tc[j][i] && tc[i][k])
                            tc[j][k] = true;
            }
        }
    }
    /** Funtion to display the trasitive closure **/
    public void displayTC()
    {
        System.out.println("\nTransitive closure :\n");
        System.out.print(" ");
        for (int v = 0; v < V; v++)
           System.out.print("   " + v );
        System.out.println();
        for (int v = 0; v < V; v++)
        {
            System.out.print(v +" ");
            for (int w = 0; w < V; w++)
            {
                if (tc[v][w])
                    System.out.print("  * ");
                else
                    System.out.print("    ");
            }
            System.out.println();
        }
    }

    /** Main function **/
    public static void main (String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Warshall Algorithm Test\n");
        /** Make an object of Warshall class **/
        Warshall w = new Warshall();

        /** Accept number of vertices **/
        System.out.println("Enter number of vertices\n");
        int V = scan.nextInt();

        /** get graph **/
        System.out.println("\nEnter matrix\n");
        int[][] graph = new int[V][V];
        for (int i = 0; i < V; i++)
            for (int j = 0; j < V; j++)
                graph[i][j] = scan.nextInt();
         w.getTC(graph);
        w.displayTC();
    }
}
```

```
Warshall Algorithm Test

Enter number of vertices
```

```
6

Enter matrix

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

Transitive closure :

    0   1   2   3   4   5
0   *   *       *   *   *
1       *
2   *   *   *   *   *   *
3               *
4               *   *
5               *   *   *
```