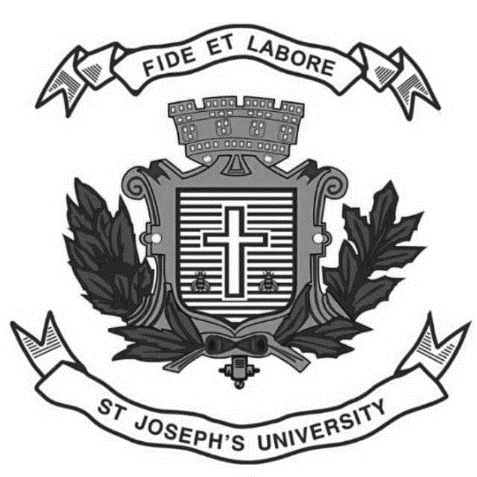
**St. Joseph’s UNIVERSITY**

**Bangalore-560027**



**Estd-1882**

**COMPUTER SCIENCE**

**PRACTICAL RECORD**

**(DESIGN AND ANALYSIS OF ALGORITHM)**

**Under the guidance of Submitted by:**

**1. Prof. Sandhya Student Name: Shashank**

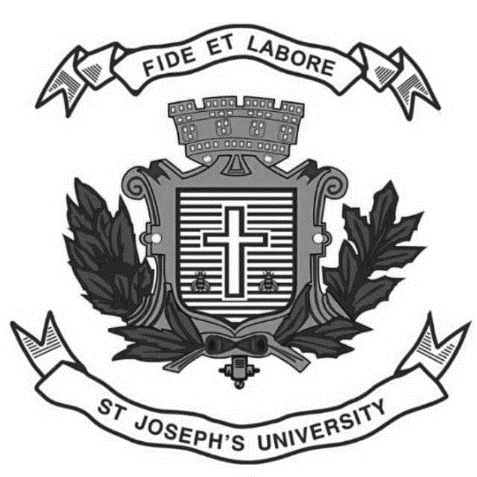
**2. Prof. Bojamma Roll No: 222CSC33**

**3. Prof. Sarakutty**

**4. Prof. Mrinmoyee**

**ST. JOSEPH’S UNIVERSITY**

**BANGALORE**



LABORATORY CERTIFICATE

This is to certify that, Sri. Shashank has satisfactorily completed the course of laboratory assignments in Design and Analysis of Algorithms prescribed by St. Joseph’s University for the First Semester Master’s degree course in Computer Science for the year 2022-23.

Name: Shashank Teacher In-Charge

Reg. No.: 222CSC33

Date of Exam: 18/11/2022 Head of the Department/PG Coordinator

***INDEX***

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Program Names** | **Page No.** |
| 1. | Write a program to implement linear search for a set of given elements. | 01 |
| 2. | Write a program to implement binary search using divide and conquer technique. | 03 |
| 3. | Write a program to implement bubble sort for a set of given elements. | 05 |
| 4. | Write a program using selection sort to arrange n numbers. | 07 |
| 5. | Write a program to perform insertion sort to reorder n numbers in ascending order. | 09 |
| 6. | Write a program to perform quick sort using divide and conquer technique to arrange n numbers. | 11 |
| 7. | Write a program to perform merge sort using divide and conquer  technique to arrange n numbers. | 13 |
| 8. | Write a program to implement push and pop operations in a stack. | 15 |
| 9 | Write a program to implement insert and delete operations in a queue. | 16 |
| 10. | Write a program to find Maximum and Minimum elements using  Divide and Conquer techniques. | 18 |
| 11. | Write a program to implement 0/1 Knapsack Problem using Dynamic Programming. | 20 |
| 12. | Write a program to display the Degree of Vertices of a Edge Matrix for a particular graph G. | 23 |
| 13. | Write a program to find the shortest path for a given vertex in a  weighted connected graph using Dijkstra's Algorithm. | 25 |
| 14. | Write a program to find the minimum cost spanning tree of a  given undirected graph using Kruskal’s Algorithm. | 27 |
| 15. | Write a program to print all reachable nodes from a given starting  node in a digraph using BFS Algorithm. | 30 |
| 16. | Write a program to check whether a given graph is connected or is not  connected using DFS Algorithm. | 32 |
| 17. | Write a program to compute the transitive closure of a given directed  graph using Warshall’s Algorithm. | 34 |
| 18. | Write a program to implement all pair shortest path’s problem using Floyd’s Algorithm. | 37 |

1. **Write a program to implement linear search for a set of given elements**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to perform linear search on a set of elements

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.Scanner;

public class LinearSearch {

    public static void linearSearch(int element, int[] array){

        boolean flag=false;

        for(int i=0;i<array.length;i++){

            if(element==array[i]){

                flag=true;

                System.out.println("Element found in the index "+i);

                break;

            }

        }

        if(flag==false) System.out.println("element not found");

    }

    //method overload for multi-linear search

    public static void linearSearch(int element, int[] array, boolean multiple){

        if(multiple){

            ArrayList<Integer> newArr = new ArrayList<>();

            for(int i=0;i<array.length;i++){

                if(element==array[i]){

                    newArr.add(i);

                }

            }

            if(newArr.isEmpty())

                System.out.println("element not found");

            else

                System.out.println("element found in");

                System.out.println(newArr.toString());

        }

        else{

            linearSearch(element, array);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for(int i =0;i<n;i++){

            array[i] = sc.nextInt();

        }

        System.out.println("Enter the element to be searched in the array");

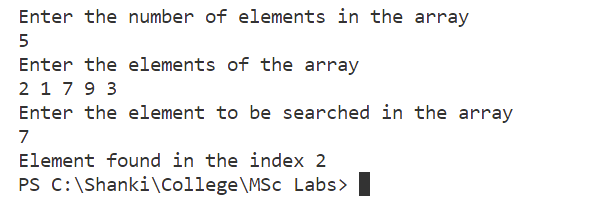
        int element=sc.nextInt();

        linearSearch(element, array);

    }

}

Output:



1. **Write a program to implement binary search using divide and conquer technique.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to search for a given element in the array

 using binary search which uses divide and conquer method

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class BinarySearch {

    private static int binarySearchInternal(int element, int[] array, int min, int max) {

        int mid = (min + max) / 2;

        // when the problem is not small

        if (max >= min) {

            if (array[mid] == element) {

                return mid;

            }

            // when the element is in the 1st half

            if (array[mid] > element) {

                max = mid - 1;

                return binarySearchInternal(element, array, min, max);

            }

            // when the element is in the second half

            else {

                min = mid + 1;

                return binarySearchInternal(element, array, min, max);

            }

        }

        // when element is not found

        return -1;

    }

    public static void binarySearch(int element, int[] array) {

        int min = 0;

        int max = array.length - 1;

        // binary search only works on sorted arrays so..

        Arrays.sort(array);

        int pos = binarySearchInternal(element, array, min, max);

        if (pos == -1) {

            System.out.println("Element not found");

        } else {

            System.out.println("Element found in position " + pos);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

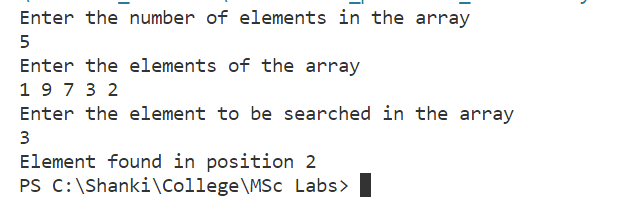
        System.out.println("Enter the element to be searched in the array");

        int element = sc.nextInt();

        binarySearch(element, array);

    }

}



1. **Write a program to implement bubble sort for a set of given elements.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to sort the given array in ascending order

 using bubble sort

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class BubbleSort {

    public static int[] bubbleSort(int[] unsorted) {

        System.out.println("Using bubble sort");

        for (int i = 0; i < unsorted.length; i++) {

            boolean swapped = false;

            for (int j = 0; j < unsorted.length - 1 - i; j++) {

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

                    int t = unsorted[j];

                    unsorted[j] = unsorted[j + 1];

                    unsorted[j + 1] = t;

                    swapped = true;

                }

            }

            // if the swap did not take place the array is already sorted

            // hence we can break out of the loop to not do unnecessary computation

            if (!swapped) {

                break;

            }

        }

        return (unsorted);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // input the array

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

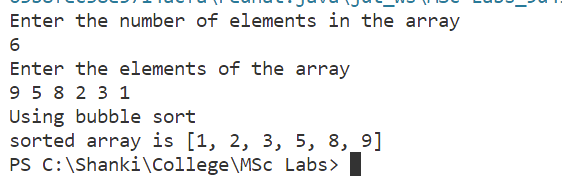
        // printing the sorted array

        System.out.println("sorted array is " + Arrays.toString(bubbleSort(array)));

    }

}

Output:



1. **Write a program using selection sort to arrange n numbers.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to perform selection sort on a given unsorted array

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class SelectionSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // input of array

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        // sorting and printing the array

        System.out.println("sorted array is " + Arrays.toString(selectionSort(array)));

    }

    public static int[] selectionSort(int[] unsorted) {

        System.out.println("Using selection sort");

        for (int i = 0; i < unsorted.length; i++) {

            int min\_index = i;

            for (int j = i + 1; j < unsorted.length; j++) {

                if (unsorted[min\_index] > unsorted[j]) {

                    min\_index = j;

                }

            }

            int t = unsorted[i];

            unsorted[i] = unsorted[min\_index];

            unsorted[min\_index] = t;

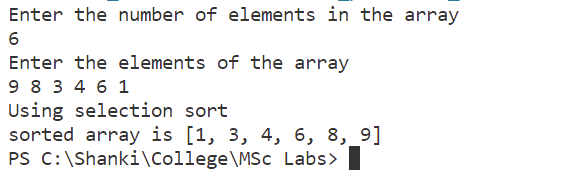
        }

        return (unsorted);

    }

}

Output:



1. **Write a program to perform insertion sort to reorder n numbers in ascending order.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to sort a given array in ascending order using

 insertion sort algorithm

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class InsertionSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // input of the array

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        // function call and printing the array

        System.out.println("sorted array is " + Arrays.toString(insertionSort(array)));

    }

    public static int[] insertionSort(int[] array) {

        System.out.println("Using insertion sort to sort the array .. ");

        // starting from 2nd element as we assume 1st is sorted

        for (int i = 1; i < array.length; i++) {

            int key = array[i];

            int j = i - 1;

            while (j >= 0 && key < array[j]) {

                array[j + 1] = array[j];

                j--;

            }

            array[j + 1] = key;

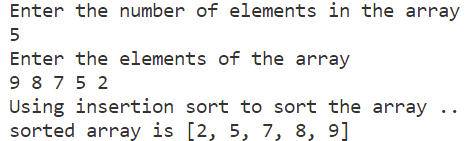
        }

        return (array);

    }

}

Output:



1. **Write a program to perform quick sort using divide and conquer technique to arrange n numbers.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to perform quick sort and arrange a given array in

 ascending order

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class QuickSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("sorted array is " + Arrays.toString(quickSort2(array, 0, array.length - 1)));

    }

    public static int[] quickSort2(int[] unsorted, int low, int high) {

        if (low < high) {

            int p = partition2(unsorted, low, high);

            quickSort2(unsorted, low, p - 1);

            quickSort2(unsorted, p + 1, high);

        }

        return unsorted;

    }

    // outputs the position on which array is split

    private static int partition2(int[] unsorted, int low, int high) {

        int i = low, j = high;

        int index = unsorted[low];

        while (i < j) {

            while (unsorted[i] < index && i < high) {

                i++;

            }

            while (unsorted[j] > index && j >= low) {

                j--;

            }

            if (i < j) {

int temp = unsorted[i];

                unsorted[i] = unsorted[j];

                unsorted[j] = temp;

            }

        }

        int temp2 = unsorted[j];

        unsorted[j] = index;

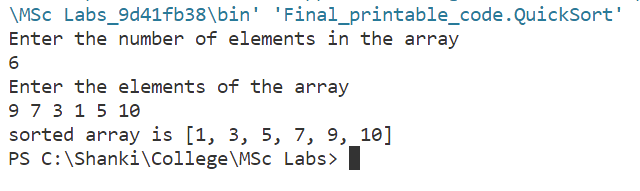
        index = temp2;

        return j;

    }

}

Output:



1. **Write a program to perform merge sort on n number**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to perform merge sort on a given unsorted array

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class MergeSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("sorted array is " + Arrays.toString(mergeSort(array, 0, array.length - 1)));

    }

    public static int[] mergeSort(int[] unsorted, int beg, int end) {

        if (beg < end) {

            int mid = (beg + end) / 2;

            // Divide the entire array into 2 halves and call merge sort on them

            mergeSort(unsorted, beg, mid);

            mergeSort(unsorted, mid + 1, end);

            merge(unsorted, beg, end, mid);

        }

        return unsorted;

    }

    private static void merge(int[] unsorted, int beg, int end, int mid) {

        int n1 = mid - beg + 1;

        int n2 = end - mid;

        int i, j, k = beg;

        int[] leftArray = new int[n1];

        int[] rightArray = new int[n2];

        // Copying the half of the arrays to 2 new temporary arrays

        for (i = 0; i < n1; i++)

            leftArray[i] = unsorted[beg + i];

        for (j = 0; j < n2; j++)

            rightArray[j] = unsorted[mid + j + 1];

        i = 0;

        j = 0;

        // comparing the left and right array and adding in order

        while (i < n1 && j < n2) {

            // count++;//to keep track of each comparisions

            if (leftArray[i] <= rightArray[j]) {

                unsorted[k] = leftArray[i];

                i++;

            } else {

                unsorted[k] = rightArray[j];

                j++;

            }

            k++;

        }

        // happens after either left or the right array runs

        // out and the remaining of the 2 are added

        while (i < n1) {

            unsorted[k] = leftArray[i];

            i++;

            k++;

        }

        while (j < n2) {

            unsorted[k] = rightArray[j];

            j++;

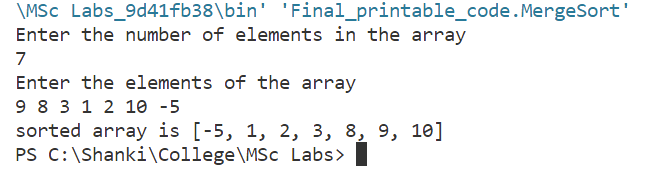
            k++;

        }

    }

}

Output:



1. **Write a program to implement push and pop operations in a stack.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to demonstrate and implement the stack data structure

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.ArrayList;

public class StackDemo {

    public ArrayList<Integer> array = new ArrayList<>();

    // delete the element at top

    public int pop() throws Exception {

        if (array.isEmpty()) {

            System.out.println("Array empty");

            throw new Exception("Array empty");

        }

        int ele = array.get(array.size() - 1);

        array.remove(array.size() - 1);

        return ele;

    }

    // insert element at the top

    public void push(int ele) {

        array.add(ele);

    }

    public void display() {

        System.out.println(“Stack elements:”+array.toString());

    }

    public static void main(String[] args) throws Exception {

        StackDemo newStack = new StackDemo();

        newStack.push(6);

        newStack.push(5);

        newStack.display();

        newStack.pop();

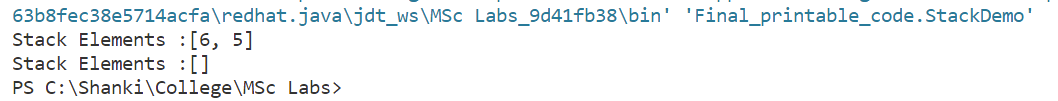
        newStack.pop();

        newStack.display();

    }

}

Output:



1. **Write a program to implement insert and delete operations in a queue.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to demonstrate implementation of a queue

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

public class QueueDemo {

    int front = -1;

    int rear = -1;

    int[] array;

    QueueDemo(int size) {

        this.array = new int[size];

    }

    QueueDemo() {

        this.array = new int[10];

    }

    public int deQueue() {

        if (isQempty()) {

            System.out.println("Queue is empty and cant delete");

            return -1;

        }

        // creating a backup of the deleted element just in case

        int t = array[front];

        if (front == rear) {

            front = rear = -1;

            return t;

        }

        front++;

        return t;

    }

    public void enQueue(int ele) {

        if (isQfull()) {

            System.out.println("Queue is full");

            return;

        }

        if (front == -1) {

            front = 0;

        }

        rear++;

        // adding the element to therea

        array[rear] = ele;

    }

    public boolean isQempty() {

        if (front == -1)

            return true;

        return false;

    }

    public boolean isQfull() {

        if (front == 0 && rear == array.length - 1)

            return true;

        return false;

    }

    public int peek() {

        return array[front];

    }

    public void display() {

        int i;

        for (i = front; i != rear; i = (++i) % array.length) {

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

        }

        System.out.print(array[i]);

        System.out.println("");

        return;

    }

    public static void main(String[] args) {

        QueueDemo newQueue = new QueueDemo(5);

        newQueue.enQueue(5);

        newQueue.enQueue(9);

        newQueue.enQueue(4);

        newQueue.display();

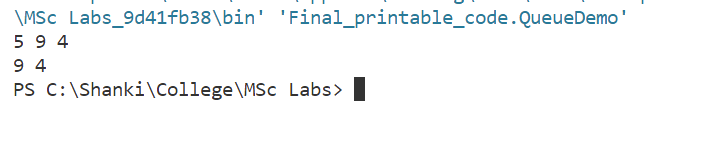
        newQueue.deQueue();

        newQueue.display();

    }

}

Output:



1. **Write a program to find maximum and minimum of an array using divide and conquer algorithm.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find the maximum and the minimum of a given array using divide

 and conquor technique for a given matrix

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Scanner;

public class MaxMinDandC {

    public static int[] minMaxSearch(int[] array, int min, int max) {

        int i = min, j = max;

        // The array contains local minimum at position 0 and local maximum at position

        // 1

        int[] minmax\_array = new int[2];

        // if its a small problem of length 1

        if (i == j) {

            minmax\_array[0] = array[i];

            minmax\_array[1] = array[i];

            return minmax\_array;

        }

        // if its a small problem of length 2

        if (i == j - 1) {

            if (array[i] < array[j]) {

                minmax\_array[0] = array[i];

                minmax\_array[1] = array[j];

            } else {

                minmax\_array[1] = array[i];

                minmax\_array[0] = array[j];

            }

            return minmax\_array;

        }

        // if its not a small problem then use the divide and conquor method

        int mid = (min + max) / 2;

        int[] left\_minmax = minMaxSearch(array, min, mid);

        int[] right\_minmax = minMaxSearch(array, mid + 1, max);

        if (left\_minmax[0] < right\_minmax[0])

            minmax\_array[0] = left\_minmax[0];

        else

            minmax\_array[0] = right\_minmax[0];

        if (left\_minmax[1] > right\_minmax[1])

            minmax\_array[1] = left\_minmax[1];

        else

            minmax\_array[1] = right\_minmax[1];

        return minmax\_array;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter all the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        int[] mmarr = new int[2];

        mmarr = minMaxSearch(array, 0, array.length - 1);

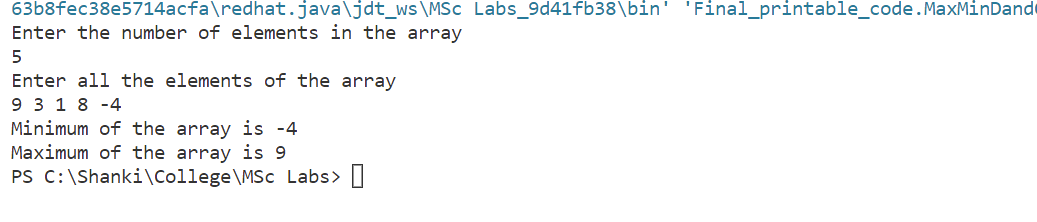
        System.out.println("Minimum of the array is " + mmarr[0]);

        System.out.println("Maximum of the array is " + mmarr[1]);

    }

}

Output:



1. **Write a program to implement 0/1 Knapsack Problem using Dynamic Programming.**

package Final\_printable\_code;

import java.util.\*;

class NewKnapSack {

    int totpro, capacity;

    void knapsck(int w[], int ob[], int p[], int Case, float piwi[], int tempcap, int x[], int n) {

        int i, k, temppro = 0, j;

        capacity = tempcap;

        if (Case <= 3) {

            for (i = 0; i < n; i++)

                x[i] = 0;

            sort(w, ob, p, n, Case, piwi);

            for (i = 0; i < n; i++) {

                if (tempcap >= w[i]) {

                    tempcap = tempcap - w[i];

                    k = ob[i];

                    x[k - 1] = 1;

                }

            }

            for (i = 0; i < n; i++)

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

                    if (i + 1 == ob[j])

                        temppro = temppro + (x[i] \* p[j]);

            System.out.print("\nSolution Set For Case " + Case + ":X= ");

            for (i = 0; i < n; i++)

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

            System.out.println("\n Case " + Case + " Profit is " + temppro);

            Case++;

            if (totpro < temppro)

                totpro = temppro;

            knapsck(w, ob, p, Case, piwi, capacity, x, n);

        } else

            System.out.println("\n\n Optimal Profit is " + totpro);

    }

    void swap(int a[], int i, int j) {

        int temp;

        temp = a[i];

        a[i] = a[j];

        a[j] = temp;

    }

    void swap1(float a[], int i, int j) {

        float temp;

        temp = a[i];

        a[i] = a[j];

        a[j] = temp;

    }

    void sort(int w[], int ob[], int p[], int n, int cas, float piwi[]) {

        int i, j;

        for (i = 0; i < n; i++)

            for (j = i + 1; j < n; j++) {

                if (w[i] >= w[j] && cas == 1) {

                    swap(w, i, j);

                    swap(ob, i, j);

                    swap(p, i, j);

                    swap1(piwi, i, j);

                }

                if (p[i] <= p[j] && cas == 2) {

                    swap(p, j, i);

                    swap(ob, i, j);

                    swap(w, i, j);

                    swap1(piwi, i, j);

                }

                if (piwi[i] <= piwi[j] && cas == 3) {

                    swap1(piwi, j, i);

                    swap(p, i, j);

                    swap(ob, i, j);

                    swap(w, i, j);

                }

            }

    }

    public static void main(String args[]) {

        int n, m, w[], p[], i, ob[], x[];

        float piwi[], a, b;

        w = new int[50];

        p = new int[50];

        ob = new int[50];

        x = new int[50];

        piwi = new float[50];

        Scanner sc = new Scanner(System.in);

        System.out.println("\n Enter the Value of N: ");

        n = sc.nextInt();

        System.out.println("\n Enter the Capacity: ");

        m = sc.nextInt();

        System.out.println("\n Enter " + n + " Weigths: ");

        for (i = 0; i < n; i++) {

            w[i] = sc.nextInt();

            ob[i] = i + 1;

        }

        System.out.println("\n Enter " + n + " Profits: ");

        for (i = 0; i < n; i++)

            p[i] = sc.nextInt();

        for (i = 0; i < n; i++) {

            a = p[i];

            b = w[i];

            piwi[i] = a / b;

        }

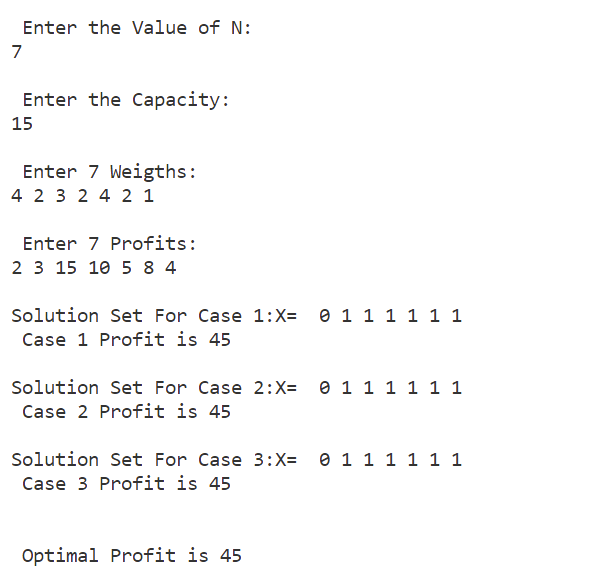
        NewKnapSack ks = new NewKnapSack();

        ks.knapsck(w, ob, p, 1, piwi, m, x, n);

    }

}

**Output:**

****

1. **Write a program to display the degree of vertices for an edge matrix for a particular graph G.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find the degree of a node for a given graph

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

/\*Works on both directed and non directed graphs \*/

import java.util.Scanner;

public class NodeDegree {

    // private method to input edge\_matrix and output the degree array of nodes

    private static int[] getDegreeArray(int[][] edge\_matrix) {

        int n = edge\_matrix.length;

        int[] degree\_array = new int[n];

        for (int i = 0; i < n; i++) {

            int local\_count = 0;

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

                if (edge\_matrix[i][j] != 999 && edge\_matrix[i][j] >= 0) {

                    local\_count++;

                }

            }

            degree\_array[i] = local\_count;

        }

        return degree\_array;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n = sc.nextInt();

        int[][] edge\_matrix = new int[n][n];

        System.out.println("Enter the edge adjacency matrix");

        for (int i = 0; i < n; i++) {

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

                edge\_matrix[i][j] = sc.nextInt();

                // idiot proofing to allow 0 as input in vertex to itself

                if (i == j) {

                    edge\_matrix[i][j] = 999;

                }

            }

        }

        int[] cost\_array = getDegreeArray(edge\_matrix);

        for (int i = 0; i < n; i++) {

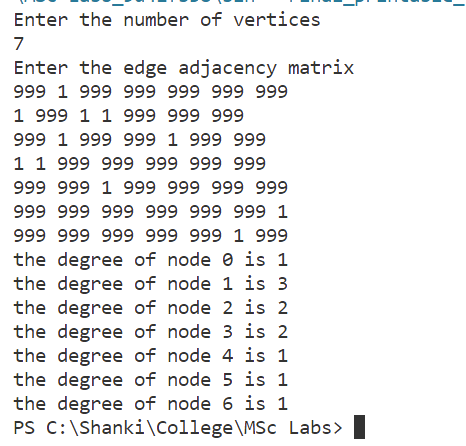
            System.out.println("the degree of node " + i + " is " + cost\_array[i]);

        }

    }

}

Output:



1. **Write a program to find the shortest path for a given vertex in a weighted connected graph using Dijkstra’s algorithm.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find the minimum cost to visit all the nodes from

 a single source using dijikstras algorithm

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class DijikstrasAlgo {

  public static void dijkstra(int[][] graph, int source) {

    int count = graph.length;

    boolean[] visitedVertex = new boolean[count];

    int[] distance = new int[count];

    Arrays.fill(visitedVertex, false);

    Arrays.fill(distance, 999);

    // Distance of self loop is zero

    distance[source] = 0;

    for (int i = 0; i < count; i++) {

      // Update the distance between neighbouring vertex and source vertex

      int u = findMinDistance(distance, visitedVertex);

      visitedVertex[u] = true;

      // Update all the neighbouring vertex distances

      for (int v = 0; v < count; v++) {

        if (!visitedVertex[v] && graph[u][v] != 0 && (distance[u] + graph[u][v] < distance[v])) {

          distance[v] = distance[u] + graph[u][v];

        }

      }

    }

    for (int i = 0; i < distance.length; i++) {

      System.out.println(String.format("Distance from %s to %s is %s", source, i, distance[i]));

    }

  }

  // Finding the minimum distance

  private static int findMinDistance(int[] distance, boolean[] visitedVertex) {

    int minDistance = 999;

    int minDistanceVertex = -1;

    for (int i = 0; i < distance.length; i++) {

      if (!visitedVertex[i] && distance[i] < minDistance) {

        minDistance = distance[i];

        minDistanceVertex = i;

      }

    }

    return minDistanceVertex;

  }

  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);

    System.out.println("Enter the number of vertices");

    int n = sc.nextInt();

    int[][] cost\_matrix = new int[n][n];

    System.out.println("Enter the cost adjacency matrix");

    for (int i = 0; i < n; i++) {

      for (int j = 0; j < n; j++) {

        cost\_matrix[i][j] = sc.nextInt();

        // idiot proofing to allow 0 as input in vertex to itself

        if (i == j) {

          cost\_matrix[i][j] = 999;

        }

      }

    }

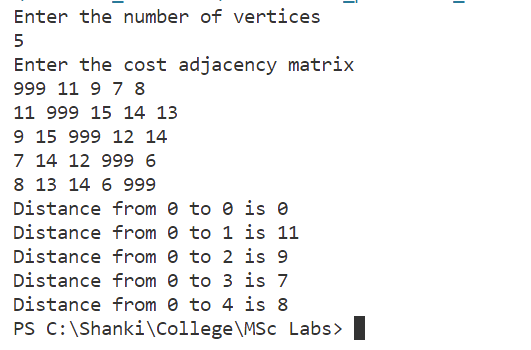
    DijikstrasAlgo T = new DijikstrasAlgo();

    T.dijkstra(cost\_matrix, 0);

  }

}

**Output:**

****

1. **Write a program to find the minimum cost spanning tree of a given undirected graph using Kruskal’s algorithm**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find the minimum spanning tree of a given graph

 using the kruskal's algorithm

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.\*;

public class Kruskals {

    static class Edge {

        int source;

        int destination;

        int weight;

        public Edge(int source, int destination, int weight) {

            this.source = source;

            this.destination = destination;

            this.weight = weight;

        }

    }

    static class Graph {

        int vertices;

        ArrayList<Edge> allEdges = new ArrayList<>();

        Graph(int vertices) {

            this.vertices = vertices;

        }

        public void addEgde(int source, int destination, int weight) {

            Edge edge = new Edge(source, destination, weight);

            allEdges.add(edge);

        }

        public void kruskalMST() {

            PriorityQueue<Edge> pq = new PriorityQueue<>(allEdges.size(),

                    Comparator.comparingInt(o -> o.weight));

            // adding all edges to a priority queue which sorts according to the edge

            // weights

            pq.addAll(allEdges);

            int[] parent = new int[vertices];

            // makes an array of all vertices

            makeSet(parent);

            ArrayList<Edge> mst = new ArrayList<>();

            int index = 0;

            // start adding edges to the mst according to lowest weights

            while (index < vertices - 1) {

                Edge edge = pq.remove();

                int x\_set = find(parent, edge.source);

                int y\_set = find(parent, edge.destination);

                // if forms a loop do nothing

                if (x\_set == y\_set) {

                }

                // else add them to mst

                else {

                    mst.add(edge);

                    index++;

                    union(parent, x\_set, y\_set);

                }

            }

            System.out.println("Minimum Spanning Tree: ");

            printGraph(mst);

        }

        public void makeSet(int[] parent) {

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

                parent[i] = i;

            }

        }

        public int find(int[] parent, int vertex) {

            if (parent[vertex] != vertex)

                return find(parent, parent[vertex]);

            return vertex;

        }

        public void union(int[] parent, int x, int y) {

            int x\_set\_parent = find(parent, x);

            int y\_set\_parent = find(parent, y);

            parent[y\_set\_parent] = x\_set\_parent;

        }

        public void printGraph(ArrayList<Edge> edgeList) {

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

                Edge edge = edgeList.get(i);

                System.out.println("Edge-" + i + " source: " + edge.source + " destination: " + edge.destination

                        + " weight: " + edge.weight);

            }

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Input the number vertices and edges

        System.out.println("Enter the number of vertices initially");

        int n = sc.nextInt();

        Graph graph = new Graph(n);

        System.out.println("Enter the number of edges present");

        int edges = sc.nextInt();

        // input the edges

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

            System.out.println("Enter the source, destination and weight of edge " + (i + 1));

            int s = sc.nextInt();

            int d = sc.nextInt();

            int w = sc.nextInt();

            graph.addEgde(s, d, w);

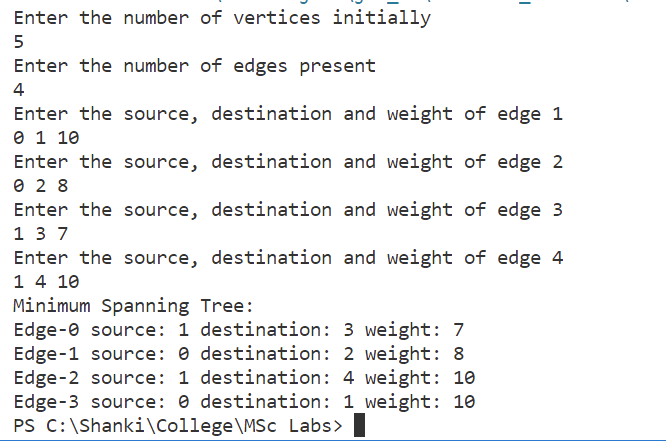
        }

        graph.kruskalMST();

    }

}

**Output:**

****

1. **Write a program to print all reachable nodes given a starting node in a digraph using BFS algorithm.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find all reachable nodes from a source node using BFS

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.Scanner;

public class TraversalBFS {

    int n;

    boolean[] visited;

    TraversalBFS(int n) {

        visited = new boolean[n];

    }

    ArrayList<Integer> traversal\_arr = new ArrayList<>();

    int front = 0;

    void rBFS(int[][] cost\_matrix, int start\_node) {

        n = cost\_matrix.length;

        this.visited[start\_node] = true;

        if (!traversal\_arr.contains(start\_node))

            traversal\_arr.add(start\_node);

        for (int i = 0; i < n; i++) {

            if (cost\_matrix[start\_node][i] != 999 && !visited[i]) {

                // add all unvisted neighbours to the traversal array

                if (!traversal\_arr.contains(i))

                    traversal\_arr.add(i);

            }

        }

        // if there are elements in the traversal array who have not been searched

        // through yet then ..

        if (traversal\_arr.size() > front + 1)

            rBFS(cost\_matrix, traversal\_arr.get(++front));

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n = sc.nextInt();

        int[][] cost\_matrix = new int[n][n];

        System.out.println("Enter the edge matrix");

        for (int i = 0; i < n; i++) {

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

                cost\_matrix[i][j] = sc.nextInt();

                if (i == j) {

                    cost\_matrix[i][j] = 999;

                }

            }

        }

        System.out.println("Enter starting node");

        int s\_node = sc.nextInt();

        TraversalBFS gs1 = new TraversalBFS(cost\_matrix.length);

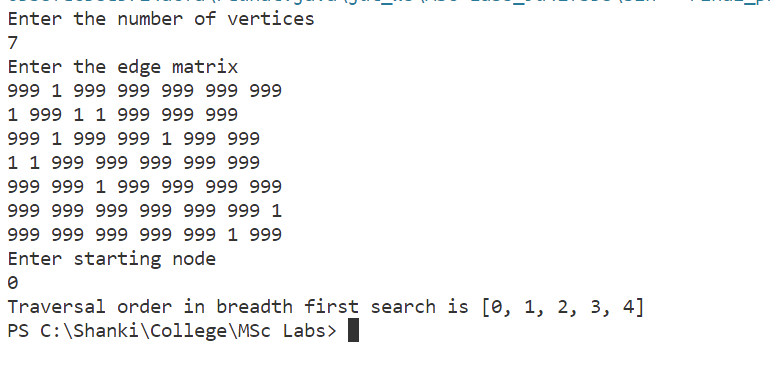
        gs1.rBFS(cost\_matrix, s\_node);

        System.out.println("Traversal order in breadth first search is " + (gs1.traversal\_arr));

    }

}

Output:



1. **Write a program to check whether the given graph is connected or not using DFS algorithm.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find the degree of a node for a given graph

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.Scanner;

public class TraversalDFS {

    int n;

    boolean[] visited;

    TraversalDFS(int n) {

        visited = new boolean[n];

    }

    ArrayList<Integer> traversal\_arr = new ArrayList<>();

    int front = 0;

    void DFS(int[][] cost\_matrix, int start\_node) {

        n = cost\_matrix.length;

        this.visited[start\_node] = true;

        traversal\_arr.add(start\_node);

        if (isUnvisted(visited)) {

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

                if (cost\_matrix[start\_node][i] != 999 && !visited[i]) {

                    // if the neighbour is not visited then call dfs on it

                    DFS(cost\_matrix, i);

                }

            }

        }

    }

    // checks if there are unvisited vertices remaining

    private static boolean isUnvisted(boolean[] arr) {

        for (boolean x : arr) {

            if (x == false) {

                return true;

            }

        }

        return false;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n = sc.nextInt();

        int[][] cost\_matrix = new int[n][n];

        System.out.println("Enter the edge matrix");

        for (int i = 0; i < n; i++) {

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

                cost\_matrix[i][j] = sc.nextInt();

                if (i == j) {

                    cost\_matrix[i][j] = 999;

                }

            }

        }

        System.out.println("Enter starting node");

        int s\_node = sc.nextInt();

        TraversalDFS gs1 = new TraversalDFS(cost\_matrix.length);

        gs1.DFS(cost\_matrix, s\_node);

        System.out.println("Traversal order in breadth first search is " + (gs1.traversal\_arr));

        // The visited array contains booleans for each vertices

        if (isUnvisted(gs1.visited)) {

            System.out.println("The graph is disjoint");

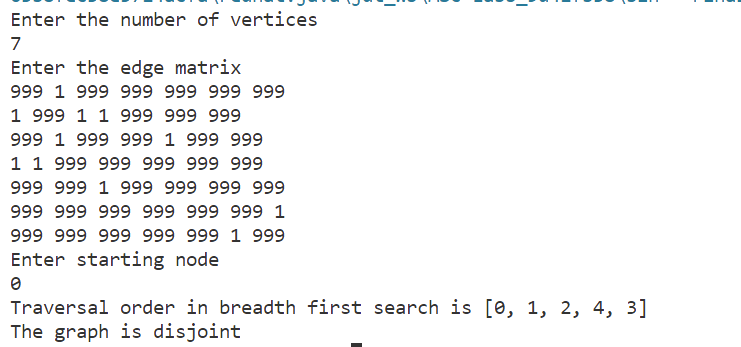
        } else

            System.out.println("All nodes are reachable");

    }

}

Output:



1. **Write a program to compute the transitive closure of a given directed graph using Warshall’s algorithm.**

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find the transitive closure edge matrix

 for a given matrix

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

//Edge class describes an edge with source and destination properties

class Edge {

    int source, dest;

    public Edge(int source, int dest) {

        this.source = source;

        this.dest = dest;

    }

}

class Graph {

    // contains a matrix which represents the graph

    List<List<Integer>> adjList = null;

    // constructor to initialize the graph

    Graph(List<Edge> edges, int N) {

        adjList = new ArrayList<>(N);

        for (int i = 0; i < N; i++) {

            adjList.add(i, new ArrayList<>());

        }

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

            int src = edges.get(i).source;

            int dest = edges.get(i).dest;

            adjList.get(src).add(dest);

        }

    }

}

public class TransClosure {

    public static void DFS(Graph graph, byte[][] C, int root, int descendant) {

        for (int child : graph.adjList.get(descendant)) {

            if (C[root][child] == 0) {

                C[root][child] = 1;

                DFS(graph, C, root, child);

            }

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of edges initially");

        int n = sc.nextInt();

        // taking all the edges present as input

        List<Edge> edges = new ArrayList<>();

        for (int i = 0; i < n; i++) {

            System.out.println("Edge " + (i + 1));

            System.out.println("Enter source");

            int source = sc.nextInt();

            System.out.println("Enter destination");

            int destination = sc.nextInt();

            edges.add(new Edge(source, destination));

        }

        final int N = n + 1;

        // initalizing the graph

        Graph graph = new Graph(edges, N);

        // creating the byte array representing the output graph

        byte[][] C = new byte[N][N];

        System.out.println("TRANSITIVE CLOSURE:-\n");

        for (int v = 0; v < N; v++) {

            C[v][v] = 1;

            // using depth first search on each vertex to visit all children possible

            DFS(graph, C, v, v);

            // printing each line

            for (int u = 0; u < N; u++)

                System.out.print(C[v][u] + " ");

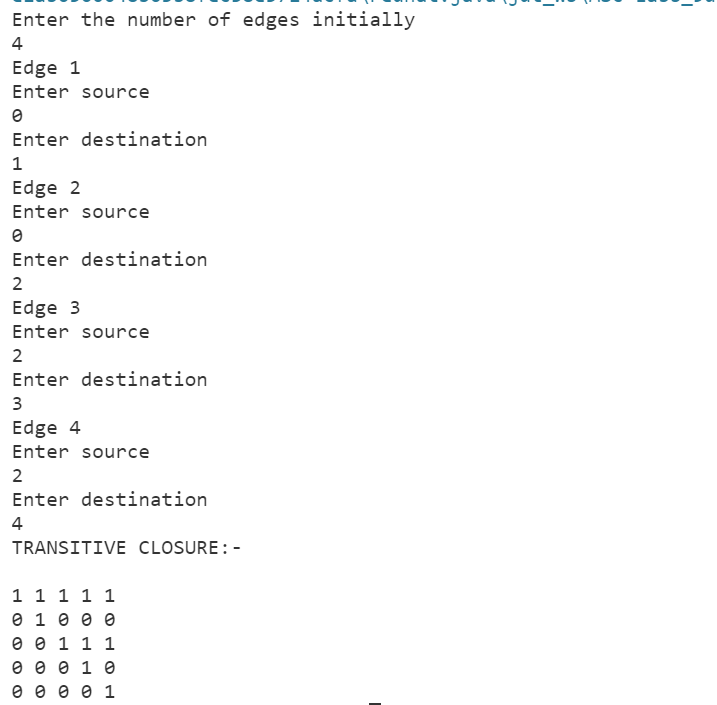
            System.out.println();

        }

    }

}

**Output:**

****

1. **Write a program to implement all pair shortest path’s problem using Floyd’s Algorithm**.

/\*----------------------------------------------------------------

 @Author: Shashank

 Program to find the all pair shortest path using the

 Floyd-Warshall’s algorithm (dynamic programming)

 ------------------------------------------------------------------\*/

package Final\_printable\_code;

import java.util.Scanner;

class FloydWarshall {

    void floydWarshall(int graph[][], int n) {

        int dist[][] = new int[n][n];

        int i, j, k;

        for (i = 0; i < n; i++)

            for (j = 0; j < n; j++)

                dist[i][j] = graph[i][j];

        // k counts the cycles

        for (k = 0; k < n; k++) {

            // traversing the graph with i and j

            for (i = 0; i < n; i++) {

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

                    if (dist[i][k] + dist[k][j] < dist[i][j])

                        dist[i][j] = dist[i][k] + dist[k][j];

                }

            }

        }

        printSolution(dist, n);

    }

    void printSolution(int dist[][], int n) {

        System.out.println("SHORTEST PATH MATRIX:-");

        for (int i = 0; i < n; ++i) {

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

                if (dist[i][j] == 999)

                    System.out.print("I ");

                else

                    System.out.print(dist[i][j] + " ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n = sc.nextInt();

        int[][] cost\_matrix = new int[n][n];

        // taking cost matrix of graph as input

        System.out.println("Enter the cost matrix");

        for (int i = 0; i < n; i++) {

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

                cost\_matrix[i][j] = sc.nextInt();

                if (i == j) {

                    cost\_matrix[i][j] = 0;

                }

            }

        }

        FloydWarshall a = new FloydWarshall();

        a.floydWarshall(cost\_matrix, n);

    }

}

Output:

