ACHARYA INSTITUTE OF TECHNOLOGY

Soladevanahalli, Bangalore - 560107

(Affiliated to Visvesvaraya Technological University, Belgaum)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY MANUAL

18CSL47

As per VTU Syllabus 2018-19 (CBCS)

For

IV Semester B.E.

Compiled By

DEPT. OF COMPUTER SCIENCE & ENGINEERING
ACHARYA INSTITUTE OF TECHNOLOGY
BANGALORE-560107

DESIGN AND ANALYSIS OF ALGORITHM LABORATORY [As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2018 -2019) SEMESTER – IV

Subject Code 18CSL47	IA Marks 40
Number of Lecture Hours/Week 02 I + 02 P	Exam Marks 60
Total Number of Lecture Hours 36	Exam Hours 03

CREDITS – 02

Description

Design, develop, and implement the specified algorithms for the following problems using Java language under LINUX /Windows environment. NetBeans/Eclipse IDE tool can be used for development and demonstration.

Experiments

1 (a) Create a Java class called ***Student*** with the following details as variables within it.
(i) USN (ii) Name (iii) Branch (iv) Phone
Write a Java program to create *n Student* objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.
(b) Write a Java program to implement the Stack using arrays. Write Push (), Pop (), and Display () methods to demonstrate its working.

2 (a) Design a super class called ***Staff*** with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely ***Teaching*** (domain, publications), ***Technical*** (skills), and ***Contract*** (period). Write a Java program to read and display at least 3 *staff* objects of all three categories.
(b) Write a Java class called ***Customer*** to store their name and date_of_birth. The date_of_birth format should be dd/mm/yyyy. Write methods to read customer data as <name, dd/mm/yyyy> and display as <name, dd, mm, yyyy> using String Tokenizer class considering the delimiter character as "/".

3 (a) Write a Java program to read two integers *a* and *b*. Compute a/b and print, when *b* is not zero. Raise an exception when *b* is equal to zero.
(b) Write a Java program that implements a multi-thread application that hashtree threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.

4 Sort a given set of ***n*** integer elements using **Quick Sort** method and compute its time complexity. Run the program for varied values of $n > 5000$ and record the time taken to sort. Plot a graph of the time taken versus ***n*** on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and conquer method works along with its time complexity analysis: worst case, average case and best case.

5 Sort a given set of ***n*** integer elements using **Merge Sort** method and compute its time complexity. Run the program for varied values of $n > 5000$, and record the time taken to sort. Plot a graph of the time taken versus ***n*** on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and conquer method works along with its time complexity analysis: worst case, average case and best case.

6 Implement in Java, the **0/1 Knapsack** problem using
(a) Dynamic Programming method (b) Greedy method.

7 From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra's algorithm**. Write the program in Java.

8 Find Minimum Cost Spanning Tree of a given connected undirected graph using **Kruskal's algorithm**. Use Union-Find algorithms in your program.

9 Find Minimum Cost Spanning Tree of a given connected undirected graph using **Prim's algorithm**.

10 Write Java programs to
(a) Implement All-Pairs Shortest Paths problem using **Floyd's algorithm**.
(b) Implement **Travelling Sales Person problem** using Dynamic programming.

11 Design and implement in Java to find a **subset** of a given set **S** = {S1, S2, ... Sn} of ***n*** positive integers whose SUM is equal to a given positive integer ***d***. For example, if S = {1, 2, 5, 6, 8} and ***d***= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

12 Design and implement the presence of **Hamiltonian Cycle** in an undirected Graph **G** of ***n*** vertices.

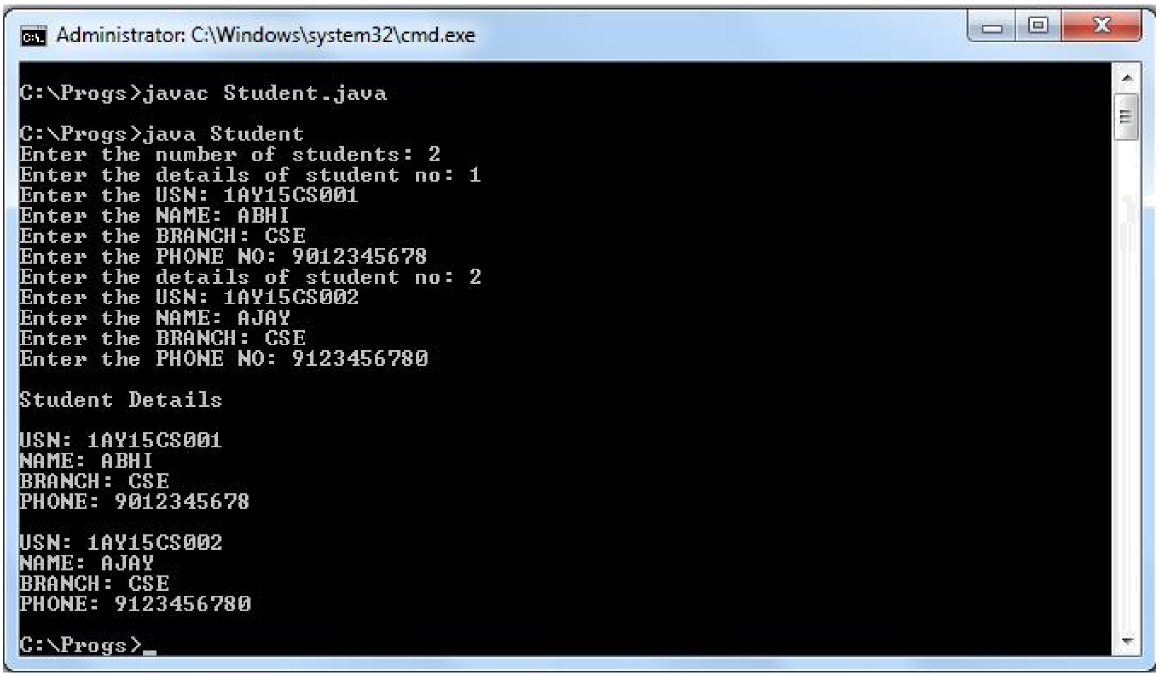
Conduction of Practical Examination:
1. All laboratory experiments (TEN problems) are to be included for practical examination.
2. Students are allowed to pick one experiment from the lot.
3. To generate the data set, use random number generator function.
4. Strictly follow the instructions as printed on the cover page of answer script for breakup of marks.

Conduct of Practical Examination:
• Experiment distribution
o For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
o For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
• Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
• Marks Distribution *(Course to change in accordance with university regulations)*
e) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
f) For laboratories having PART A and PART B
i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Program - 1. a) Create a Java class called *Student* with the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone
Write a Java program to create *n Student* objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.

```
import java.io.*;
import java.util.Scanner;

class Student
{
        String usn, name, branch;
        long phone;
        Student (String u, String n, String b, long p)
        {
                usn=u;
                name=n;
                branch=b;
                phone=p;
        }
        void display ( )
        {
                System.out.println( );
                System.out.println("USN: "+usn);
                System.out.println("NAME: "+name);
                System.out.println("BRANCH: "+branch);
                System.out.println("PHONE: "+phone);
        }

        public static void main (String args[ ])throws IOException
        {
                Scanner sc = new Scanner (System.in); //Keyboard input object
                System.out.println("Enter the number of students: "); int n =
                sc.nextInt ( );
                int i;

                Student s[ ] = new Student [n]; // N student objects
                for (i=0; i<n; i++)
                {
                        System.out.println ("Enter the details of student no: "+(i+1));
                        System.out.println ("Enter the USN: ");
                        String us = sc.next ( );
                        System.out.println ("Enter the NAME: ");
                        String na = sc.next ( );
                        System.out.println ("Enter the BRANCH: ");
                        String br = sc.next ( );
                        System.out.println ("Enter the PHONE NO: ");
                        long ph = sc.nextLong ( );

                        s[i] = new Student (us, na, br, ph);
                }
                System.out.println( );
                System.out.println("Student Details");
                for (i=0; i<n; i++)
                        s[i].display ( );
        }
}
```

Output:



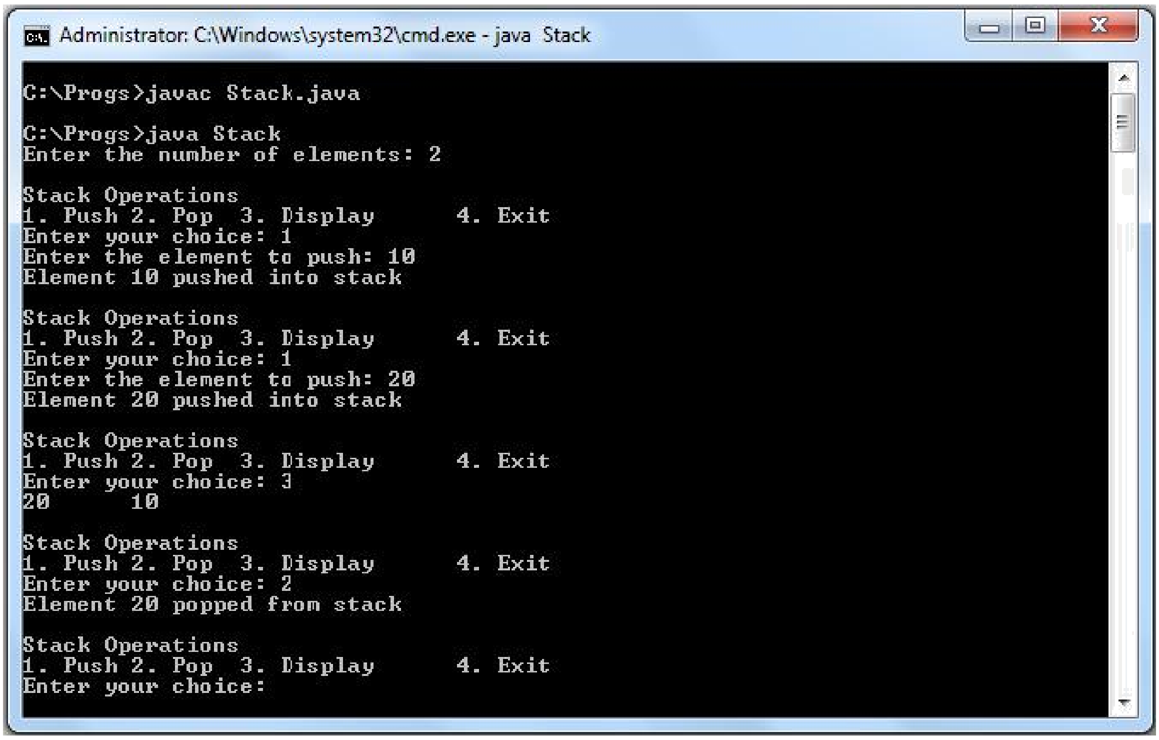
```
C:\Progs>javac Student.java

C:\Progs>java Student
Enter the number of students: 2
Enter the details of student no: 1
Enter the USN: 1AY15CS001
Enter the NAME: ABHI
Enter the BRANCH: CSE
Enter the PHONE NO: 9012345678
Enter the details of student no: 2
Enter the USN: 1AY15CS002
Enter the NAME: AJAY
Enter the BRANCH: CSE
Enter the PHONE NO: 9123456780

Student Details

USN: 1AY15CS001
NAME: ABHI
BRANCH: CSE
PHONE: 9012345678

USN: 1AY15CS002
NAME: AJAY
BRANCH: CSE
PHONE: 9123456780

C:\Progs>_
```

Program – 1 b) Write a Java program to implement the Stack using arrays. Write Push (), Pop (), and Display () methods to demonstrate its working.

```
import java.io.*;
import java.util.Scanner;

class Stack
{
        private int maxsize, top;
        private int stack[ ];

        public Stack(int s) //Constructor
        {
                maxsize = s;
                stack = new int[maxsize];
                top = -1;
        }

        public void push(int pu)
        {
                if (top==maxsize-1)
                {
                        System.out.println("Stack is Full");
                }
                else
                {
                        stack[++top] = pu;
                        System.out.println("Element "+pu+" pushed into stack");
                }
        }
        public void pop()
        {
                if(top==-1)
                        System.out.println("Stack is Empty");
                else
                        System.out.println("Element "+(stack[top--])+" popped from stack");
        }
        public void display()
        {
                if(top==-1)
                        System.out.println("Stack is Empty");
                else
                {
                        for (inti = top; i>= 0; i--)
                                System.out.print(stack[i]+"\t");
                        System.out.println();
                }
        }
        public static void main(String args[])
        {
        Scanner sc = new Scanner (System.in);
```

```
                System.out.println("Enter the number of elements:
                "); int choice;
                int n = sc.nextInt ( );
                Stack s = new Stack(n);
                do{
                        System.out.println("\nStack Operations");
                        System.out.println("1. Push");
                        System.out.println("2. Pop");
                        System.out.println("3. Display");
                        System.out.println("4. Exit");
                        System.out.println("Enter your choice: ");
                        choice = sc.nextInt();
                        switch(choice)
                        {
                                case 1: System.out.println("Enter the element to push:
                                        "); int item=sc.nextInt ( );
                                        s.push(item);
                                        break;
                                case 2: s.pop();
                                        break;
                                case 3: s.display();
                                        break;
                                case 4: break;
                        }
                }while (choice!=4);
                System.out.println("");
        }
}
```

Output:

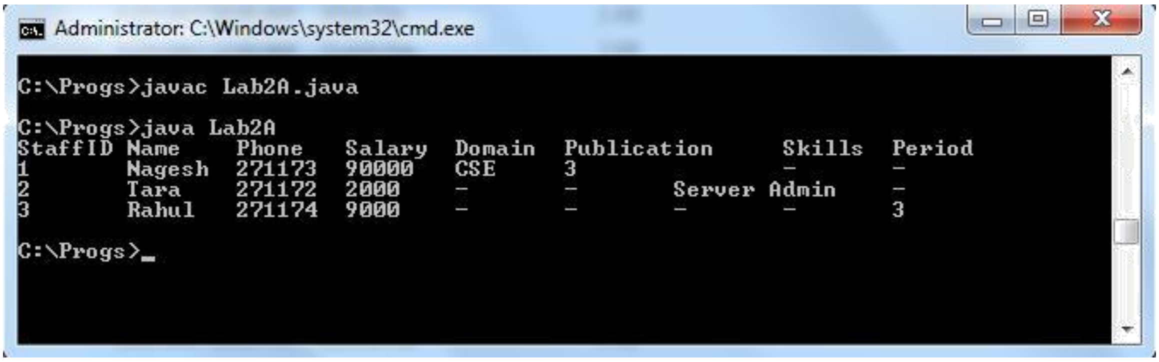


```
C:\Progs>javac Stack.java

C:\Progs>java Stack
Enter the number of elements: 2

Stack Operations
1. Push 2. Pop  3. Display      4. Exit
Enter your choice: 1
Enter the element to push: 10
Element 10 pushed into stack

Stack Operations
1. Push 2. Pop  3. Display      4. Exit
Enter your choice: 1
Enter the element to push: 20
Element 20 pushed into stack

Stack Operations
1. Push 2. Pop  3. Display      4. Exit
Enter your choice: 3
20      10

Stack Operations
1. Push 2. Pop  3. Display      4. Exit
Enter your choice: 2
Element 20 popped from stack

Stack Operations
1. Push 2. Pop  3. Display      4. Exit
Enter your choice:
```

Program – 2 a) Design a super class called *Staff* with details as StaffId, Name, Phone, and Salary. Extend this class by writing three subclasses namely *Teaching* (domain, publications), *Technical* (skills), and *Contract* (period). Write a Java program to read and display at least 3 *staff* objects of all three categories.

```
import java.io.*;

class Staff
{
        private int StaffId;
        private String Name;
        private String Phone;
        private long Salary;
        public Staff(int staffId,String name,String phone,long salary)
{
                StaffId = staffId;
                Name = name;
                Phone = phone;
                Salary = salary;
        }
        public void Display()
{
                System.out.print(StaffId+"\t"+Name+"\t"+Phone+"\t"+Salary);
        }
}
class Teaching extends Staff
{
        private String Domain;
        private int Publications;
        public Teaching(int staffId, String name, String phone, long salary, String domain,
int publications)
{
                super(staffId, name, phone, salary);
                Domain = domain;
                Publications = publications;
        }
        public void Display()
        {
                super.Display();
                System.out.print("\t"+Domain+"\t"+Publications+"\t\t-\t-");
        }
}
class Technical extends Staff
{
        private String Skills;
        public Technical(int staffId, String name, String phone, long salary, String skills)
{
                super(staffId, name, phone, salary);
                Skills = skills;
        }
        public void Display()
        {
                super.Display();
                System.out.print("\t-\t-\t"+Skills+"\t-");
        }
}
class Contract extends Staff
{
        private int Period;
        public Contract(int staffId, String name, String phone, long salary, int period)
        {
                super(staffId, name, phone, salary);
                this.Period = period;
        }
        public void Display()
        {
                super.Display();
                System.out.print("\t-\t-\t-\t-\t"+Period);
        }
}
public class Lab2A
{
        public static void main(String[] args)
        {
                Staff staff[]=new Staff[3];
                staff[0]=new Teaching(1,"Nagesh","271173",90000,"CSE",3);
                staff[1]=new Technical(2,"Tara","271172",2000,"Server
                Admin"); staff[2]=new Contract(3,"Rahul","271174",9000,3);
        System.out.println("StaffID\tName\tPhone\tSalary\tDomain\tPublication\tSkills\t
Period");
                for(int i=0;i<3;i++)
                {
                        staff[i].Display();
                        System.out.println();
                }
        }
}
```

Output:

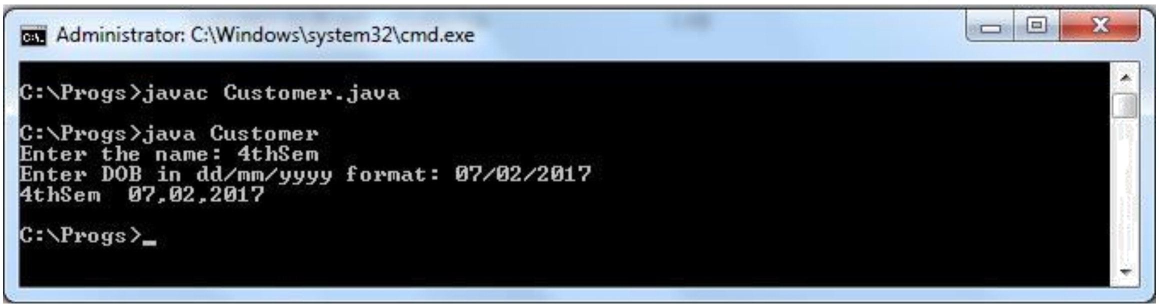


```
C:\Progs>javac Lab2A.java

C:\Progs>java Lab2A
StaffID Name    Phone   Salary  Domain  Publication     Skills  Period
1       Nagesh  271173  90000   CSE     3               -       -
2       Tara    271172  2000    -       -       Server Admin    -
3       Rahul   271174  9000    -       -       -       -       3

C:\Progs>_
```

Program – 2 b) Write a Java class called *Customer* to store their name and date_of_birth. The date_of_birth format should be dd/mm/yyyy. Write methods to read customer data as <name, dd/mm/yyyy> and display as <name, dd, mm, yyyy> using String Tokenizer class considering the delimiter character as "/".

```
import java.io.*;
import java.util.Scanner;
import java.util.StringTokenizer;

class Customer
{
        String temp;
        String dd, mm, yyyy;
        public void rd(String n, String d)
        {
                StringTokenizer token = new StringTokenizer (d, "/");
                dd = token.nextToken ( );
                mm = token.nextToken ( );
                yyyy = token.nextToken ( );
                System.out.println (n +"\t"+dd+","+mm+","+yyyy);
        }

        public static void main (String args[ ])
        {
                Scanner s = new Scanner (System.in);
                System.out.print ("Enter the name: ");
                String name = s.next ( );
                System.out.print ("Enter DOB in dd/mm/yyyy format: ");
                String date = s.next ( );
                Customer c=new Customer();
                c.rd(name, date);
        }
}
```

Output:

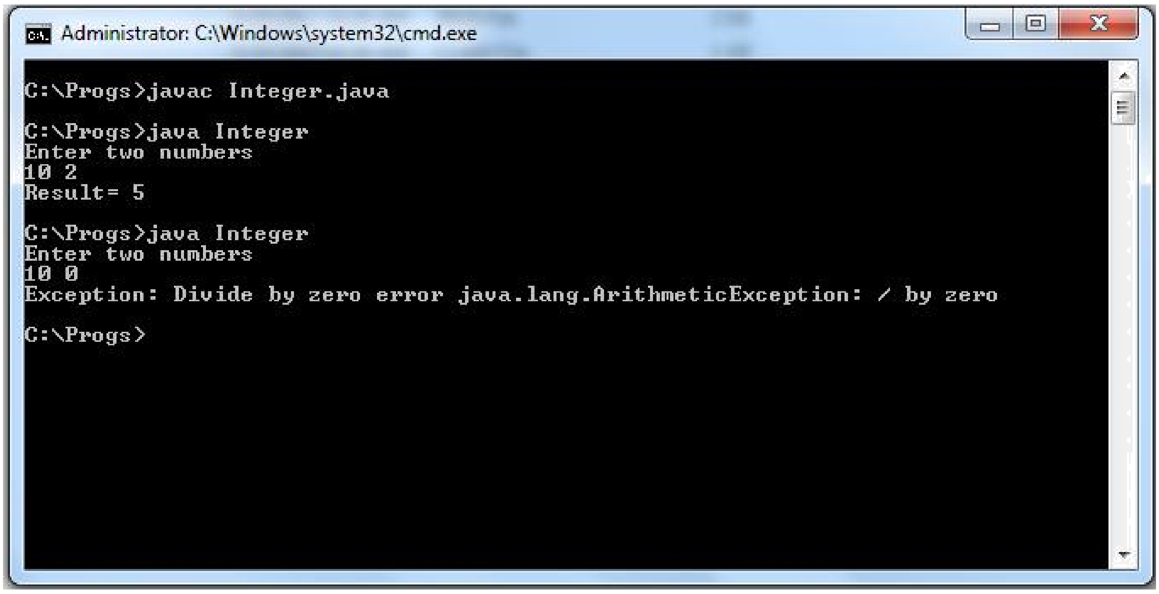


```
C:\Progs>javac Customer.java

C:\Progs>java Customer
Enter the name: 4thSem
Enter DOB in dd/mm/yyyy format: 07/02/2017
4thSem  07,02,2017

C:\Progs>_
```

Program – 3 a) Write a Java program to read two integers *a* and *b*. Compute *a*/*b* and print, when *b* is not zero. Raise an exception when *b* is equal to zero.

```
import java.io.*;
import java.util.Scanner;
class Integer
{
        public static void main(String args[])
        {
                int a, b, res;
                Scanner in = new Scanner(System.in);
                System.out.println("Enter two numbers ");
                a = in.nextInt ( );
                b = in.nextInt ( );

                try
                {
                        res=a/b;
                        System.out.println("Result= "+res);
                }
                catch(ArithmeticException e)
                {
                        System.out.println("Exception: Divide by zero error "+e);
                }
        }
}
```

Output:

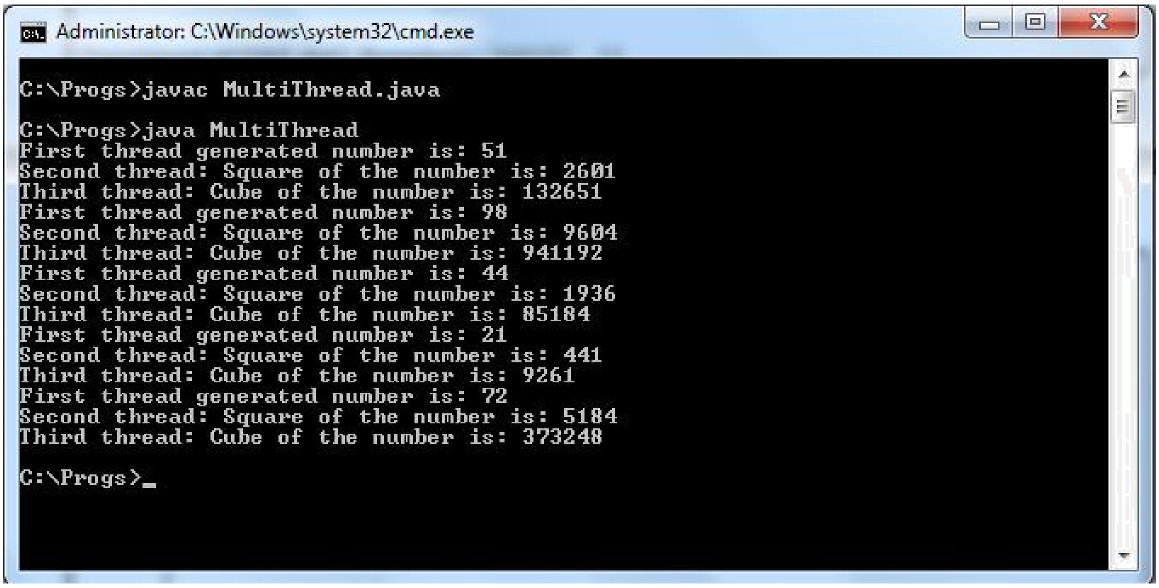


```
C:\Progs>javac Integer.java

C:\Progs>java Integer
Enter two numbers
10 2
Result= 5

C:\Progs>java Integer
Enter two numbers
10 0
Exception: Divide by zero error java.lang.ArithmeticException: / by zero

C:\Progs>
```

Program – 3 b) Write a Java program that implements a multi-thread application that has three threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.

```
import java.util.*;
class Num extends Thread
{
        public void run()
        {
                int n=0;
                Random r=new Random();
                try
                {
                for(int i=0;i<5;i++)
                {
                        n=r.nextInt(100);
                        System.out.println("First thread generated number is: +n);
                        Thread t2=new Thread (new SNum(n));
                        t2.start();

                        Thread t3=new Thread(new CNum(n));
                        t3.start();
                        Thread.sleep(1000);
                }
                }
                catch(Exception e)
                {
                        System.out.println(e.getMessage());
                }
        }
}
class SNum implements Runnable
{
        public int x;
        public SNum (int x)
        {
                this.x=x;
        }
        public void run()
        {
                System.out.println("Second thread: Square of the number is: "+x*x);
        }
}
class CNum implements Runnable
{
        public int x;
        public CNum(int x)
        {
                this.x=x;
        }
        public void run()
        {
                System.out.println("Third thread: Cube of the number is: "+x*x*x);
        }
}

public class MultiThread
{
        public static void main (String args[])
        {
                Num n=new Num();
                n.start();
        }
}
```

Output:

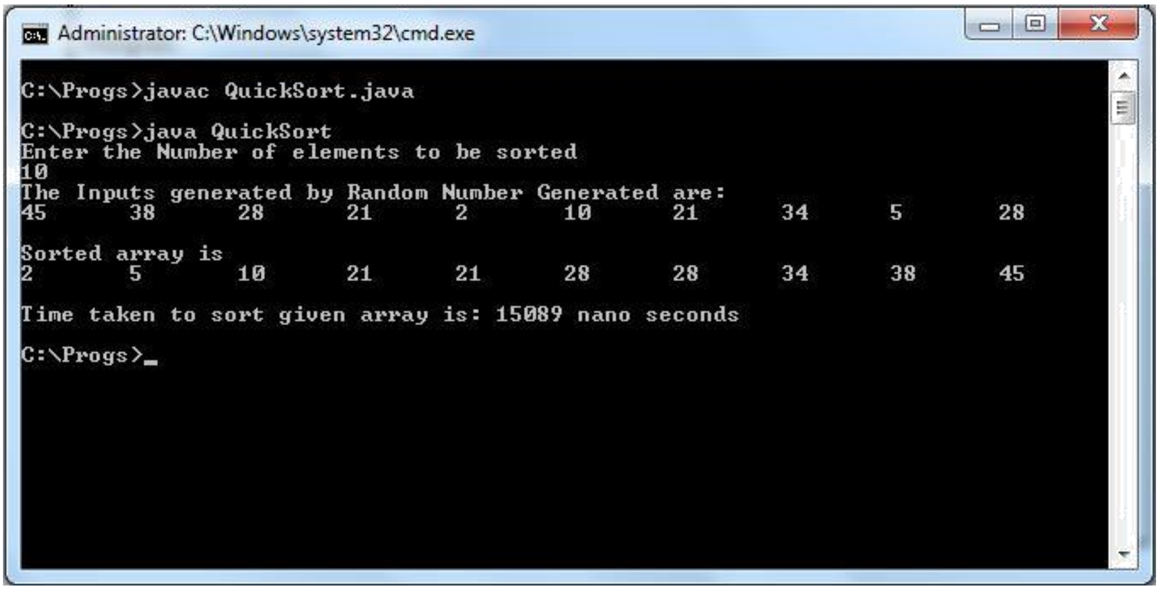


```
C:\Progs>javac MultiThread.java

C:\Progs>java MultiThread
First thread generated number is: 51
Second thread: Square of the number is: 2601
Third thread: Cube of the number is: 132651
First thread generated number is: 98
Second thread: Square of the number is: 9604
Third thread: Cube of the number is: 941192
First thread generated number is: 44
Second thread: Square of the number is: 1936
Third thread: Cube of the number is: 85184
First thread generated number is: 21
Second thread: Square of the number is: 441
Third thread: Cube of the number is: 9261
First thread generated number is: 72
Second thread: Square of the number is: 5184
Third thread: Cube of the number is: 373248

C:\Progs>_
```

Program – 4 Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n > 5000 and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.io.*;
import java.util.Random;
import java.util.Scanner;

class QuickSort
{
    static int max=5000;
    void quicksort(int a[], int low, int high)
    {
        int s;
        if(low<high) //To check for boundary condition
        {
            s=partition(a,low,high);
            quicksort(a,low,s-1);
            quicksort(a,s+1,high);
        }
    }
    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;
            }
        } //End While
        return j;
    } //End Partition
    public static void main(String args[])
    {
        int a[], i, n;
        Scanner sc =new Scanner(System.in);
        System.out.println("Enter the Number of elements to be sorted");
        n=sc.nextInt();
        a= new int[max]; //initialize array with max size
        Random generator=new Random();

        for(i=0; i<n; i++)
        {
            a[i]=generator.nextInt(50);
        }
        System.out.println("The Inputs generated by Random Number Generated
are: ");
        for(i=0; i<n; i++)
        System.out.print(a[i]+"\t");
        long startTime=System.nanoTime(); //start
        time QuickSortqs = new QuickSort(); //Object
        qs.quicksort(a,0,n-1);
        long stopTime=System.nanoTime();
        long elapseTime=(stopTime-startTime);
            System.out.println();
        System.out.println("Sorted array is");
        for(i=0;i<n;i++)
            System.out.print(a[i]+"\t");
        System.out.println();
          System.out.println("Time taken to sort given array is: "+elapseTime+" nano
seconds");
    }
}
```

Output:

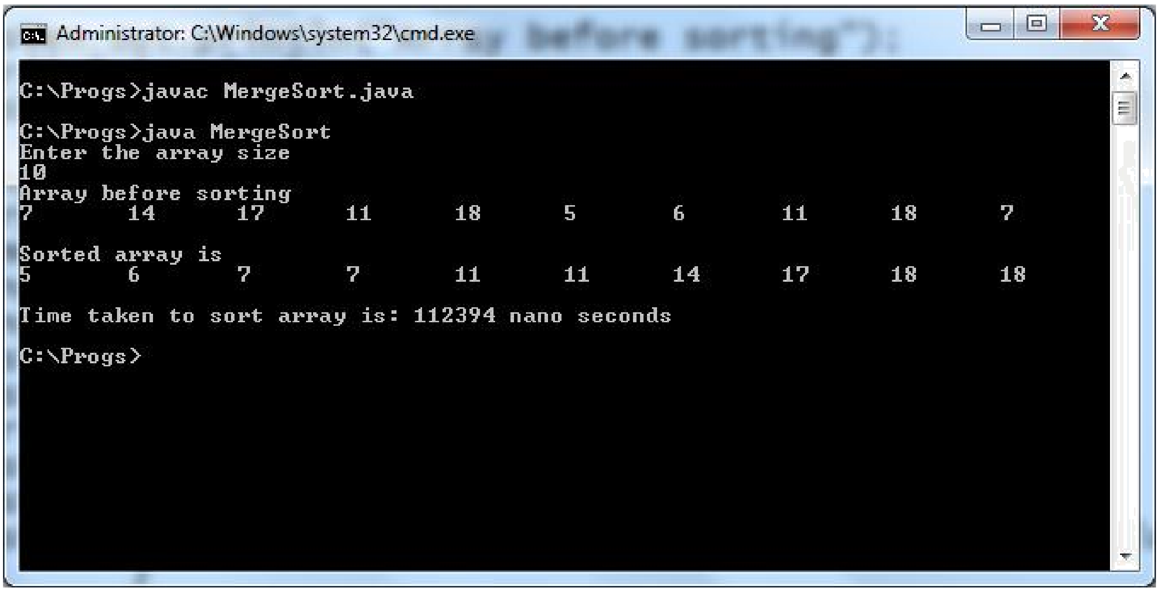


```
C:\Progs>javac QuickSort.java

C:\Progs>java QuickSort
Enter the Number of elements to be sorted
10
The Inputs generated by Random Number Generated are:
45      38      28      21      2       10      21      34      5       28

Sorted array is
2       5       10      21      21      28      28      34      38      45

Time taken to sort given array is: 15089 nano seconds

C:\Progs>_
```

Program – 5 Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n > 5000, and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.io.*;
import java.util.Scanner;
import java.util.Random;

class MergeSort
{
        static int max=5000;
        void mergesort (int a[], int low, int high)
        {
                int mid;
                if (low<high)
                {
                        mid = (low+high)/2;
                        mergesort (a, low, mid);
                        mergesort (a, mid+1, high);
                        merge (a, low, mid, high);
                }
        }
        void merge (int a[], int low, int mid, int high)
        {
                int i, j, k, t[] = new int[max];
                i=low; j=mid+1; k=low; t= new int[max]; while ((i<=mid)
                && (j<=high)) {

                        if (a[i] <= a[j])
                                t[k++] = a[i++];
                        else
                                t[k++] = a[j++];
                }
                while (i<= mid)
                        t[k++] = a[i++];
                while (j <= high)
                        t[k++] = a[j++];
                for (i=low; i<=high; i++)
                        a[i] = t[i];
        }

        public static void main(String args[])
        {
                int i, n, a[];
                System.out.println("Enter the array size");
                Scanner sc =new Scanner(System.in);
                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.print(a[i]+"\t");

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

        }
}
```

Output:



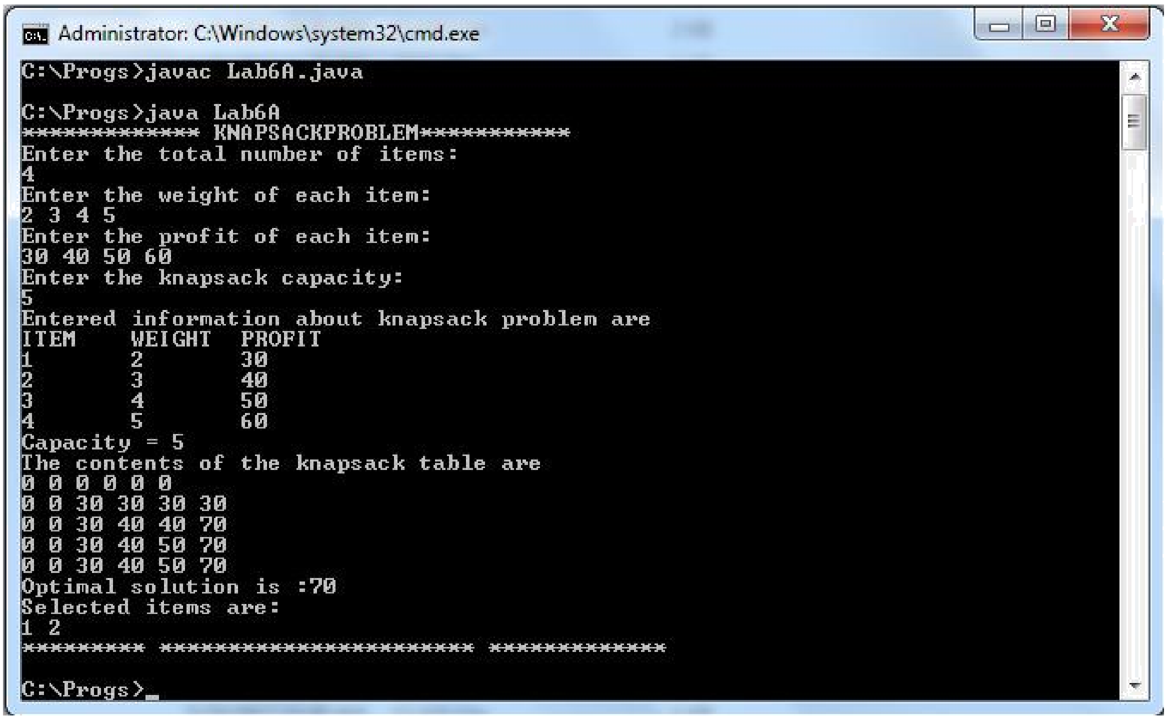
Administrator: C:\Windows\system32\cmd.exe

```
C:\Progs>javac MergeSort.java

C:\Progs>java MergeSort
Enter the array size
10
Array before sorting
7       14      17      11      18      5       6       11      18      7

Sorted array is
5       6       7       7       11      11      14      17      18      18

Time taken to sort array is: 112394 nano seconds

C:\Progs>
```

Program – 6 Implement in Java, the 0/1 Knapsack problem using a) Dynamic Programming method.

```
import java.util.Scanner;

public class Lab6A
{
        public static void main(String args[ ])
        {
                int v[][]=new int[10][10], w[]=new int[10], p[]=new int[10], i, j;
                Scanner in = new Scanner(System.in);
                System.out.println("************
                KNAPSACKPROBLEM***********"); System.out.println("Enter the
                total number of items: "); int n = in.nextInt();
                System.out.println("Enter the weight of each item: ");
                for(i=1;i<=n;i++)
                w[i] = in.nextInt();
                System.out.println("Enter the profit of each item:
                "); for(i=1;i<=n;i++)
                        p[i] = in.nextInt();
                System.out.println("Enter the knapsack capacity: ");
                int m= in.nextInt();
                displayinfo(m,n,w,p);
                knapsack(m,n,w,p,v);
                System.out.println("The contents of the knapsack table are");
                for(i=0; i<=n; i++)
                {
                        for(j=0; j<=m; j++)
                        {
                                System.out.print(v[i][j]+" " );
                        }
                        System.out.println();
                }
                optimal(m,n,w,v); //call optimal function
        }
        static void displayinfo(int m,int n,int w[],int p[])
        {
                System.out.println("Entered information about knapsack problem
                are"); System.out.println("ITEM\tWEIGHT\tPROFIT"); for(int i=1; i<=n;
                i++)
                        System.out.println(i+"\t"+w[i]+"\t"+p[i]);
                System.out.println("Capacity = "+m);
        }
        static void knapsack(int m,int n,int w[],int p[],int v[][])
        {
                for(int i=0; i<=n; i++)
                {
                        for(int j=0; j<=m; j++)
                        {
                                if(i==0 ||j==0)
                                        v[i][j]=0;
                                else if(j < w[i])
                                        v[i][j]=v[i-1][j];
                                else
                                        v[i][j]=max(v[i-1][j], v[i-1][j-w[i]]+p[i]);
                        }
                }
        }
        private static int max(int i, int j)
        {
                if(i>j)  return i;
                else      return j;
        }
        static void optimal(int m,int n,int w[],int v[][])
        {
                int i = n, j = m, item=0, x[]=new int[10];
                while( i != 0 && j != 0)
                {
                        if(v[i][j] != v[i-1][j])
                        {
                                x[i] = 1;
                                j = j-w[i];
                        }
                        i = i-1;
                }
                System.out.println("Optimal solution is :"+
                v[n][m]); System.out.println("Selected items are: ");
                for(i=1; i<= n;i++)
                        if(x[i] == 1)
                        {
                                System.out.print(i+" ");
                                item=1;
                        }
                if(item == 0)
                        System.out.println("NIL\t Sorry ! No item can be placed in Knapsack");
                System.out.println("\n********* *********************** *************");
        }
}
```

Output:

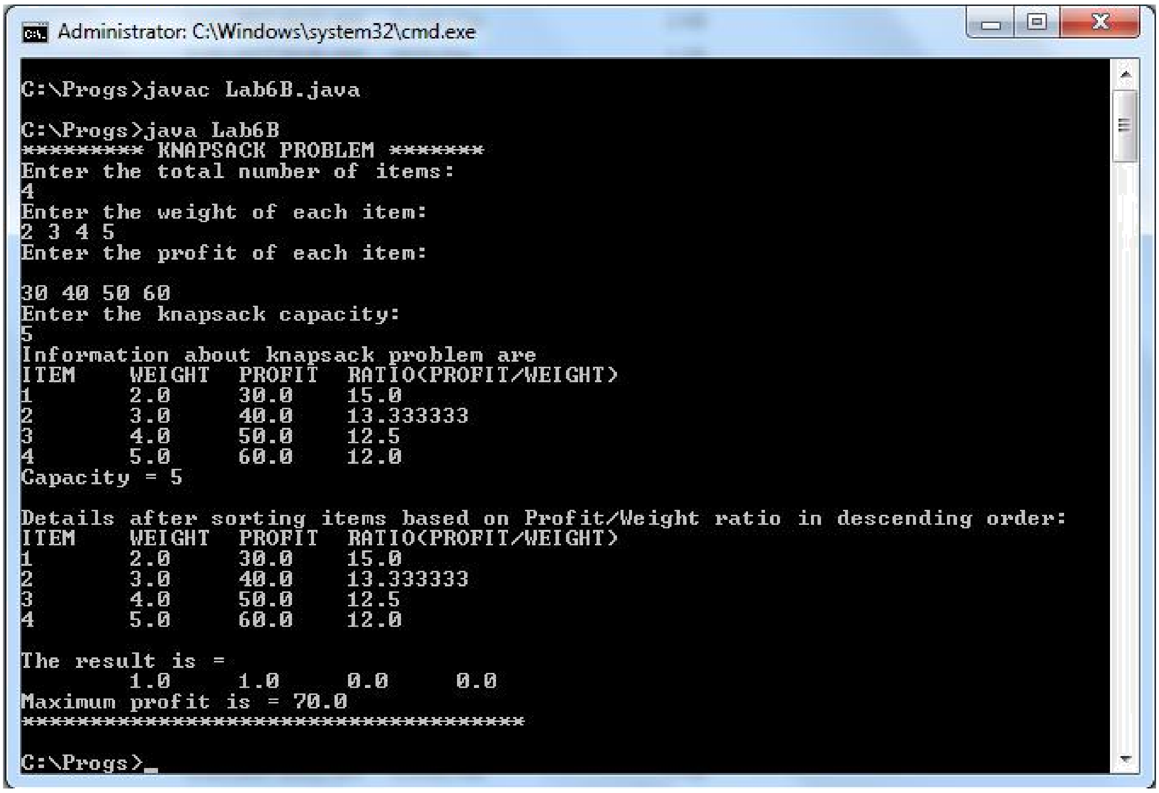


```
C:\Progs>javac Lab6A.java

C:\Progs>java Lab6A
************* KNAPSACKPROBLEM***********
Enter the total number of items:
4
Enter the weight of each item:
2 3 4 5
Enter the profit of each item:
30 40 50 60
Enter the knapsack capacity:
5
Entered information about knapsack problem are
ITEM    WEIGHT  PROFIT
1       2       30
2       3       40
3       4       50
4       5       60
Capacity = 5
The contents of the knapsack table are
0 0 0 0 0 0
0 0 30 30 30 30
0 0 30 40 40 70
0 0 30 40 50 70
0 0 30 40 50 70
Optimal solution is :70
Selected items are:
1 2
********* *********************** *************

C:\Progs>_
```

b) Greedy method.

```
import java.util.Scanner;

public class Lab6B
{
        public static void main(String args[ ])
        {
                float w[]=new float[10],p[]=new float[10];
                float ratio[]=new float[10];
                Scanner in = new Scanner(System.in);
                int i;

                System.out.println("********* KNAPSACK PROBLEM *******");
                System.out.println("Enter the total number of items:
                "); int n = in.nextInt();
                System.out.println("Enter the weight of each item:
                "); for(i=1;i<=n;i++)
                        w[i] = in.nextFloat();
                System.out.println("Enter the profit of each item: ");
                for(i=1;i<=n;i++)
                        p[i] = in.nextFloat();
                System.out.println("Enter the knapsack capacity: ");
                int m= in.nextInt();

                for(i=1;i<=n;i++)
                        ratio[i]=p[i]/w[i];
                System.out.println("Information about knapsack problem are");
                displayinfo(n,w,p,ratio);
                System.out.println("Capacity = "+m);
                sortArray(n,ratio,w,p);

                System.out.println("\nDetails after sorting items based on
                Profit/Weight ratio in descending order: ");
                displayinfo(n,w,p,ratio);
                knapsack(m,n,w,p);
                System.out.println("**************************************");
        }

        static void sortArray(int n,float ratio[],float w[],float p[])
        {
                int i,j;
                for(i=1; i<=n; i++)
                        for(j=1; j<=n-i; j++)
                        {
                                if(ratio[j]<ratio[j+1])
                                {
                                        float temp=ratio[j];
                                        ratio[j]=ratio[j+1];
                                        ratio[j+1]=temp;
                                        temp=w[j];
                                        w[j]=w[j+1];
                                        w[j+1]=temp;
                                        temp=p[j];
                                        p[j]=p[j+1];
                                        p[j+1]=temp;
                                }
                        }
        }

        static void displayinfo(int n,float w[],float p[],float ratio[])
        {
                System.out.println("ITEM\tWEIGHT\tPROFIT\tRATIO(PROFIT/WEIGHT)");
                for(int i=1; i<=n; i++)
                        System.out.println(i+"\t"+w[i]+"\t"+p[i]+"\t"+ratio[i]);
        }

        static void knapsack(int u,int n,float w[],float p[])
        {
                float x[]=new float[10],tp=0;
                int i;
                for(i=1; i<=n; i++)
                x[i]=0;
                for(i=1; i<=n; i++)
                {
                        if(w[i]>u)
                                break;
                        else
                        {
                                x[i]=1;
                                tp=tp+p[i];
                                u=(int) (u-w[i]);
                        }
                }
                if(i<n)
                        x[i]=u/w[i];
                tp=tp+(x[i]*p[i]);

                System.out.println("\nThe result is = ");
                for(i=1; i<=n; i++)
                        System.out.print("\t"+x[i]);

                System.out.println("\nMaximum profit is = "+tp);
        }
```

Output:



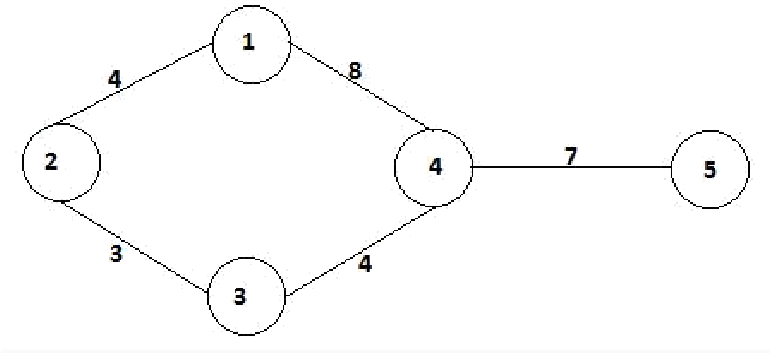
```
C:\Progs>javac Lab6B.java

C:\Progs>java Lab6B
********* KNAPSACK PROBLEM *******
Enter the total number of items:
4
Enter the weight of each item:
2 3 4 5
Enter the profit of each item:

30 40 50 60
Enter the knapsack capacity:
5
Information about knapsack problem are
ITEM    WEIGHT  PROFIT  RATIO(PROFIT/WEIGHT)
1       2.0     30.0    15.0
2       3.0     40.0    13.333333
3       4.0     50.0    12.5
4       5.0     60.0    12.0
Capacity = 5

Details after sorting items based on Profit/Weight ratio in descending order:
ITEM    WEIGHT  PROFIT  RATIO(PROFIT/WEIGHT)
1       2.0     30.0    15.0
2       3.0     40.0    13.333333
3       4.0     50.0    12.5
4       5.0     60.0    12.0

The result is =
        1.0     1.0     0.0     0.0
Maximum profit is = 70.0
*************************************

C:\Progs>_
```

Program – 7 For a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in Java.

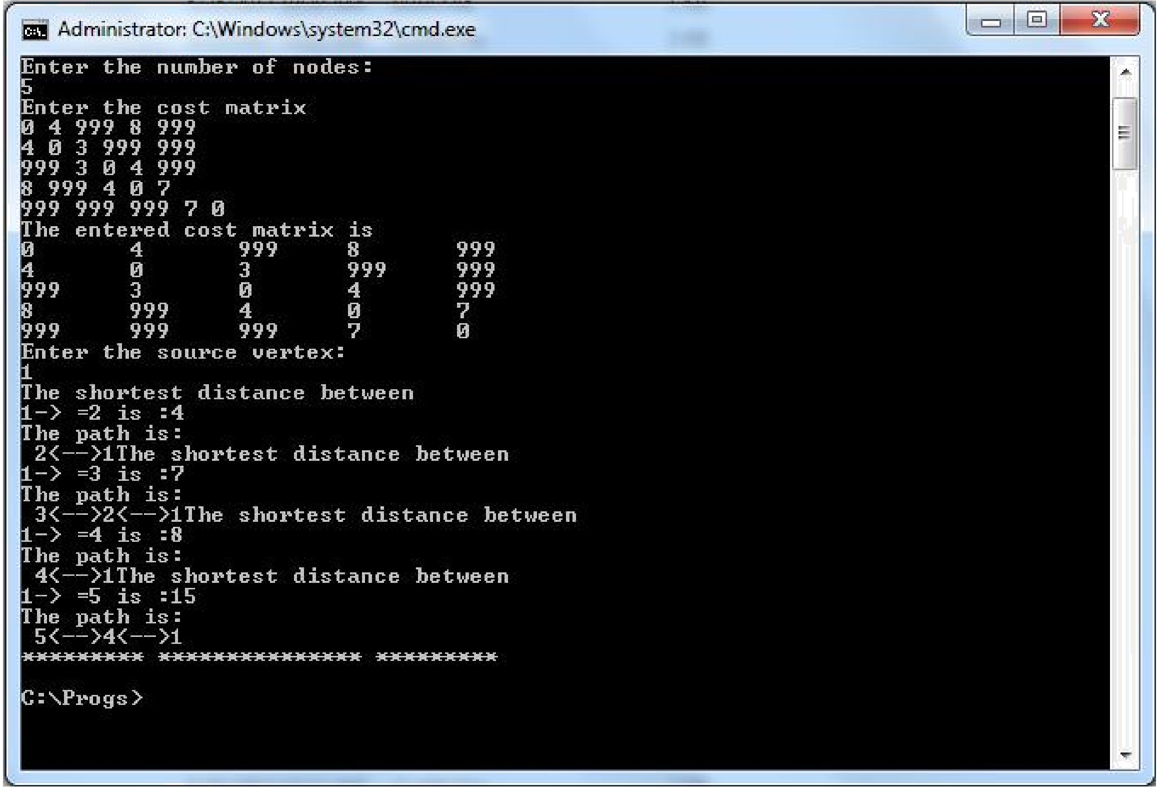
```
import java.io.*;
import java.util.Scanner;
public class Dijkstra
{
        public static void main(String[] args)
        {
                int i, j;
                int dist[]=new int[10], visited[]=new int[10]; int
                cost[][]=new int[10][10], path[]=new int[10]; Scanner in =
                new Scanner(System.in); System.out.println("****
                DIJKSTRA'S ALGORITHM ******");
                System.out.println("Enter the number of nodes: ");
                int n = in.nextInt();
                System.out.println("Enter the cost matrix");
                for(i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
                                cost[i][j] = in.nextInt();
                System.out.println("The entered cost matrix is");
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                System.out.print(cost[i][j]+"\t");
                        }
                        System.out.println();
                }
                System.out.println("Enter the source vertex: ");
                int sv = in.nextInt();
                dij(cost,dist,sv,n,path,visited);
                printpath(sv,n,dist,path,visited );
                System.out.println("\n********* ************** *********");
        }
        static void dij(int cost[][],int dist[],int sv,int n,int path[],int visited[])
        {
                int count = 2,min,v=0;
                for(int i=1; i<=n; i++)
                {
                        visited[i]=0;
                        dist[i] = cost[sv][i];
                        if(cost[sv][i] == 999)
                                path[i] = 0;
                        else
                                path[i] = sv;
                }
                visited[sv]=1;
                while(count<=n)
                {
                        min = 999;
```

```
            for(int w=1; w<=n; w++)
                if((dist[w]< min) && (visited[w]==0))
                {
                    min = dist[w];
                    v = w;
                }
            visited[v] = 1;
            count++;
            for(int w=1; w<=n; w++)
            {
                if((dist[w]) >(dist[v] + cost[v][w]))
                {
                    dist[w] = dist[v] + cost[v][w];
                    path[w] = v;
                }
            }
        }
    }

    static void printpath(int sv,int n,int dist[],int path[], int visited[])
    {
        for(int w=1; w<=n; w++)
        {
            if(visited[w] == 1 && w != sv)
            {
                System.out.println("The shortest distance between ");
                System.out.println(sv+"-> ="+w+" is :"+ dist[w]);
                int t=path[w];
                System.out.println("The path is:");
                System.out.print(" "+w);
                while(t != sv)
                {
                    System.out.print("<-->"+t);
                    t=path[t];
                }
                System.out.print("<-->"+sv);
            }
        }
    }
}
```

Input Graph



Output:

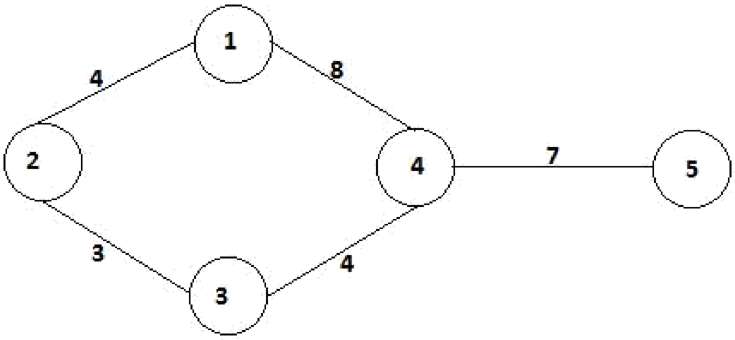


```
Enter the number of nodes:
5
Enter the cost matrix
0 4 999 8 999
4 0 3 999 999
999 3 0 4 999
8 999 4 0 7
999 999 999 7 0
The entered cost matrix is
0       4       999     8       999
4       0       3       999     999
999     3       0       4       999
8       999     4       0       7
999     999     999     7       0
Enter the source vertex:
1
The shortest distance between
1-> =2 is :4
The path is:
 2<-->1The shortest distance between
1-> =3 is :7
The path is:
 3<-->2<-->1The shortest distance between
1-> =4 is :8
The path is:
 4<-->1The shortest distance between
1-> =5 is :15
The path is:
 5<-->4<-->1
********* *************** *********

C:\Progs>
```

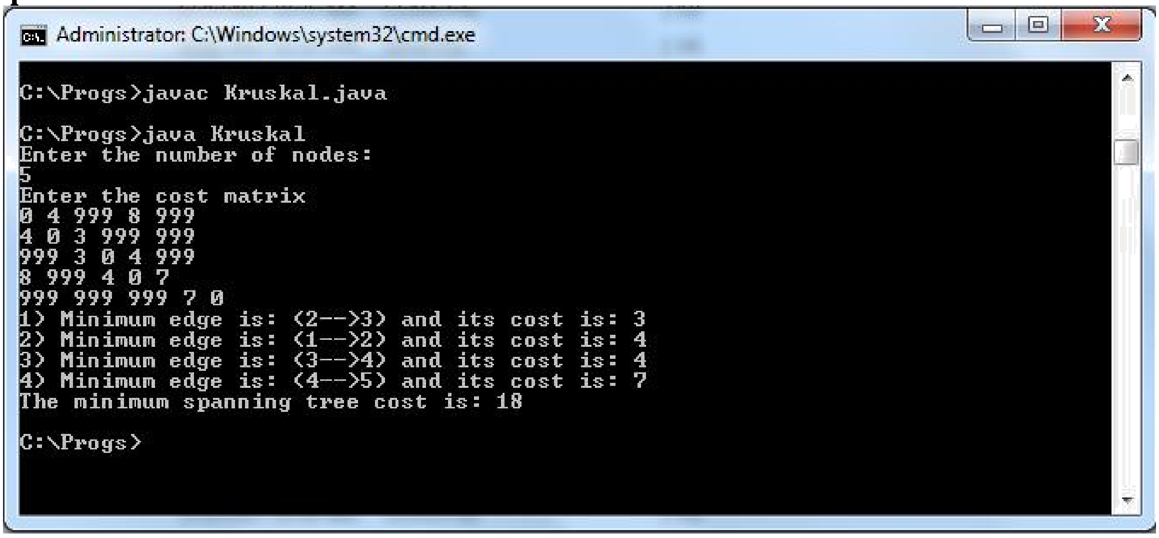
Program – 8 Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.

```
import java.io.*;
import java.util.Scanner;

public class Kruskal
{
        public static void main(String args[])
        {
                int cost[][]=new int[10][10];
                int i,j,mincost=0;
                Scanner in = new Scanner(System.in);

                System.out.println("Enter the number of nodes: ");
                int n = in.nextInt();
                System.out.println("Enter the cost matrix");
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                cost[i][j] = in.nextInt();
                        }
                }
                mincost=kruskals(n,mincost,cost);
                System.out.println("The minimum spanning tree cost is: "+mincost);
        }

        static int kruskals(int n,int mincost,int cost[][])
        {
                int ne = 1, a=0, u=0, b=0, v=0, min;
                int parent[]=new int[10];
                while(ne < n)
                {
                        min=999;
                        for(int i=1; i<=n; i++)
                        {
                                for(int j=1; j<=n; j++)
                                {
                                        if(cost[i][j] < min)
                                        {
                                                min = cost[i][j];
                                                a=u=i;
                                                b=v=j;
                                        }
                                }
                        }
                        while(parent[u]>0)
                                u = parent[u];
                        while(parent[v]>0)
                                v = parent[v];
                        if(u != v)
                        {
                                System.out.print((ne++)+") Minimum edge is: ");
                                System.out.println("("+a+"-->"+b+") and its cost is: "+min);
                                mincost += min;
                                parent[v] = u;
                        }
                        cost[a][b] = cost[b][a] = 999;
                }
                return mincost;
        }
}
```

Input Graph



Output



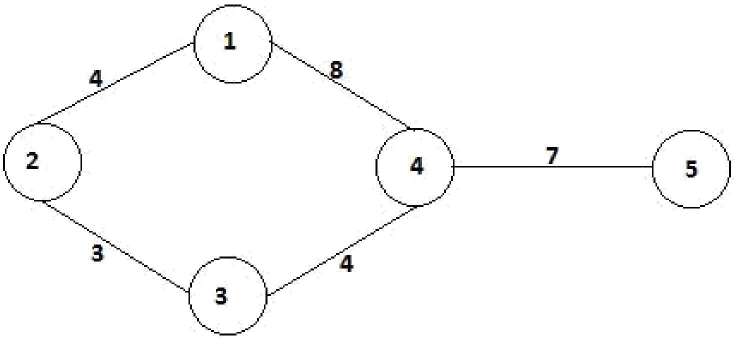
```
C:\Progs>javac Kruskal.java

C:\Progs>java Kruskal
Enter the number of nodes:
5
Enter the cost matrix
0 4 999 8 999
4 0 3 999 999
999 3 0 4 999
8 999 4 0 7
999 999 999 7 0
1> Minimun edge is: <2-->3> and its cost is: 3
2> Minimun edge is: <1-->2> and its cost is: 4
3> Minimun edge is: <3-->4> and its cost is: 4
4> Minimun edge is: <4-->5> and its cost is: 7
The minimum spanning tree cost is: 18

C:\Progs>
```

Program – 9 Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

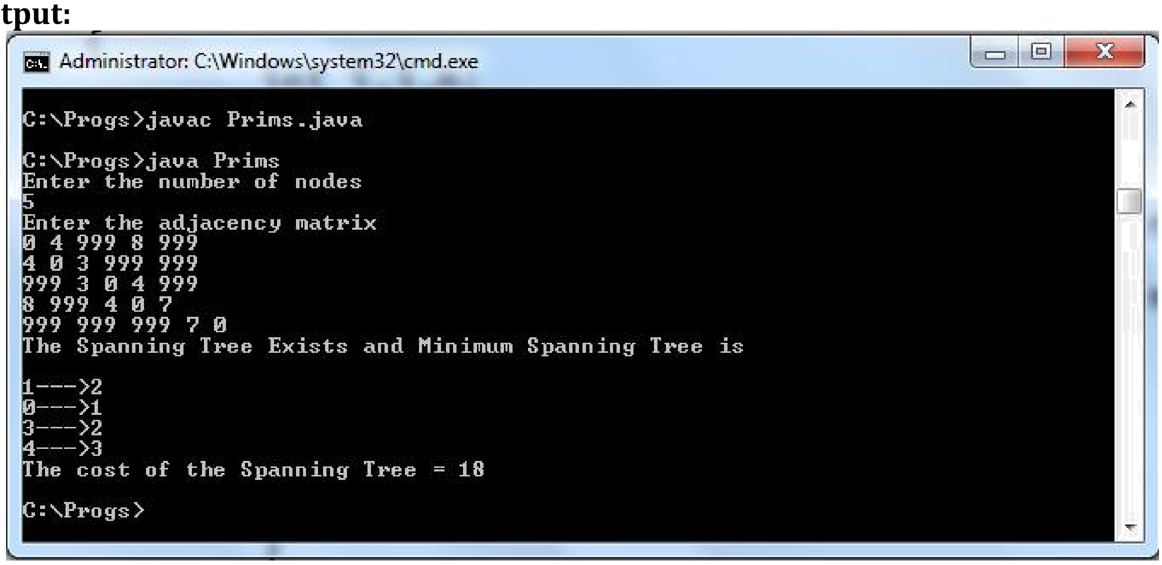
```
import java.io.*;
import java.util.Scanner;

class Prims
{
	void prim(int n, int a[][])
	{
		int i, j, k, u, v;
		int sum, min, source;
		int p[]=new int[10];
		int d[]=new int[10];
		int s[]=new int[10];
		int t[][]=new int[10][2];
		min=9999;
		source=0;
		for(i=0;i<n;i++)
		{
			for(j=0;j<n;j++)
			{
				if(a[i][j]!=0&&a[i][j]<=min)
				{
					min=a[i][j];
					source=i;
				}
			}
		}
		for(i=0;i<n;i++)
		{
			d[i]=a[source][i];
			p[i]=source;
			s[i]=0;
		}
		s[source]=1;
		sum=0;		k=0;
		for(i=1;i<n;i++)
		{
			min=9999;
			u=-1;
			for(j=0;j<n;j++)
			{
				if(s[j]==0)
				{
					if(d[j]<min)
					{
						min=d[j];
						u=j;
					}
				}
```

```
            }
            if(u==-1)
                return;
            t[k][0]=u;
            t[k][1]=p[u];
            k++;
            sum=sum+a[u][p[u]];
            s[u]=1;
            for(v=0;v<n;v++)
            {
                if(s[v]==0 && a[u][v]<d[v])
                {
                    d[v]=a[u][v];
                    p[v]=u;
                }
            }
        }
        if(sum>=9999)
        {
            System.out.println("spanning tree does not exists\n");
        }
        else
        {
            System.out.println("The Spanning Tree Exists and Minimum Spanning Tree is\n");
            for(i=0;i<n-1;i++)
            {
                System.out.println(t[i][0]+"--->"+t[i][1]);
            }
            System.out.println("The cost of the Spanning Tree = "+sum);
        }
    }
    public static void main(String args[])
    {
        int i,j,n;
        Prims p = new Prims();
        int cost[][] = new int[10][10];
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter the number of nodes");
        n = sc.nextInt();
        System.out.println("Enter the adjacency matrix");
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                cost[i][j]=sc.nextInt();
            }
        }
        p.prim(n,cost);
    }
}
```

Input Graph



Output:

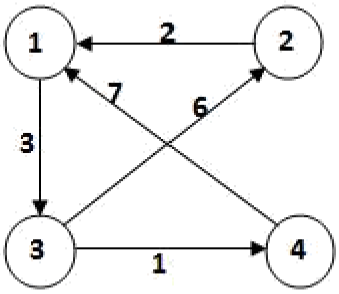


```
C:\Progs>javac Prims.java

C:\Progs>java Prims
Enter the number of nodes
5
Enter the adjacency matrix
0 4 999 8 999
4 0 3 999 999
999 3 0 4 999
8 999 4 0 7
999 999 999 7 0
The Spanning Tree Exists and Minimum Spanning Tree is

1--->2
0--->1
3--->2
4--->3
The cost of the Spanning Tree = 18

C:\Progs>
```

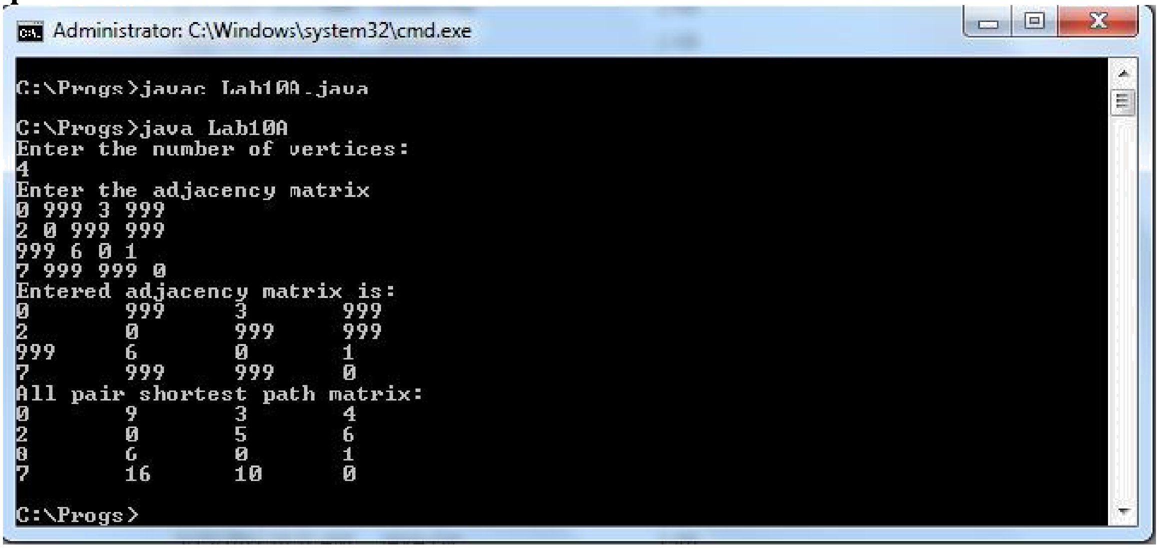
Program – 10 a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

```
import java.io.*;
import java.util.Scanner;

public class Lab10A
{
	public static void main(String args[])
	{
		int a[][]=new int[10][10], i, j;
		Scanner in = new Scanner(System.in);
		System.out.println("Enter the number of vertices: ")
		int n = in.nextInt();
		System.out.println("Enter the adjacency matrix");
		for (i=1;i<=n;i++)
			for (j=1;j<=n;j++)
				a[i][j] = in.nextInt();
		System.out.println("Entered adjacency matrix is: ");
		for(i=1;i<=n;i++)
		{
			for(j=1; j<=n; j++)
			{
				System.out.print(a[i][j]+"\t");
			}
			System.out.println();
		}
		floyd(a,n);
		System.out.println("All pair shortest path matrix:");
		for (i=1; i<=n; i++)
		{
			for (j=1; j<=n; j++)
				System.out.print(a[i][j]+"\t");
			System.out.println();
		}
	}
	static void floyd(int a[][], int n)
	{
		for (int k=1; k<=n; k++)
		{
			for (int i=1; i<=n; i++)
				for (int j=1; j<=n; j++)
					a[i][j] = min(a[i][j], a[i][k] + a[k][j]);
		}
	}
	static int min(int a, int b)
	{
		if(a>b)		return b;
		else		return a;
	}
}
```

Input Graph



Output:



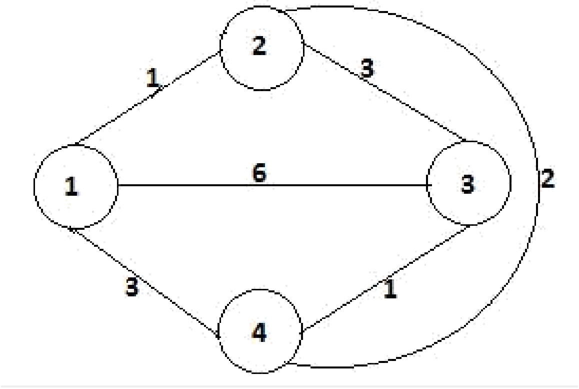
```
C:\Progs>javac Lab10A.java

C:\Progs>java Lab10A
Enter the number of vertices:
4
Enter the adjacency matrix
0 999 3 999
2 0 999 999
999 6 0 1
7 999 999 0
Entered adjacency matrix is:
0       999     3       999
2       0       999     999
999     6       0       1
7       999     999     0
All pair shortest path matrix:
0       9       3       4
2       0       5       6
8       6       0       1
7       16      10      0

C:\Progs>
```

Program – 10 b) Implement Travelling Sales Person problem using Dynamic programming.

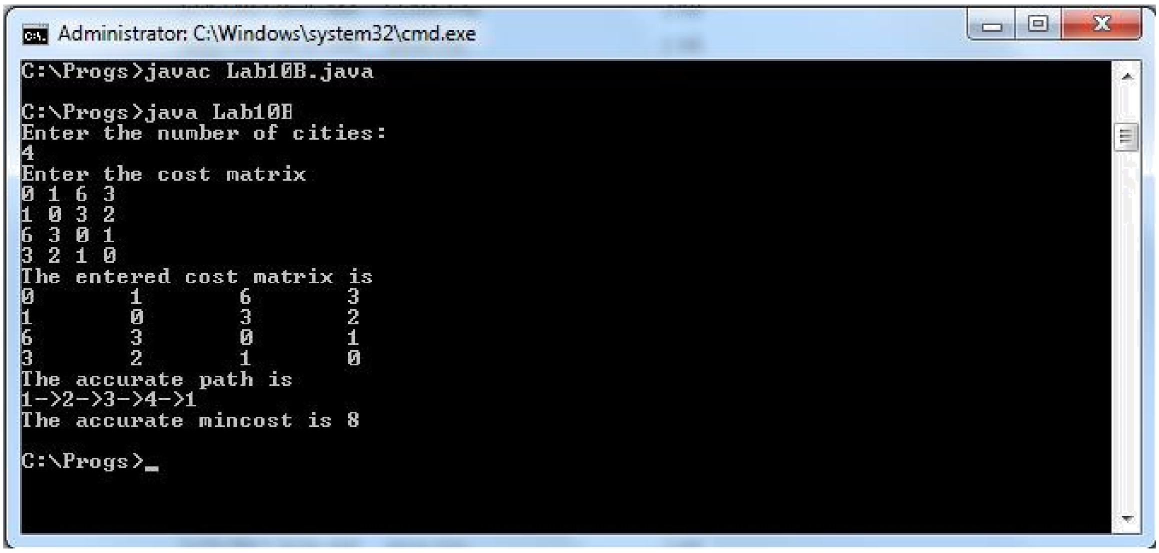
```
import java.io.*;
import java.util.Scanner;

public class Lab10B
{
	public static void main(String args[])
	{
		int c[][]=new int[10][10], tour[]=new
		int[10]; Scanner in = new
		Scanner(System.in); int i, j, cost;
		System.out.println("Enter the number of cities: ");
		int n = in.nextInt();
		if(n==1)
		{
			System.out.println("Path is not possible");
			System.exit(0);
		}
		System.out.println("Enter the cost matrix");
		for(i=1;i<=n;i++)
			for(j=1;j<=n;j++)
				c[i][j] = in.nextInt();
		System.out.println("The entered cost matrix is");
		for(i=1;i<=n;i++)
		{
			for(j=1;j<=n;j++)
			{
				System.out.print(c[i][j]+"\t");
			}
			System.out.println();
		}
		for(i=1;i<=n;i++)
			tour[i]=i;
		cost = tspdp(c, tour, 1, n);
		System.out.println("The accurate path is");
		for(i=1;i<=n;i++)
			System.out.print(tour[i]+"->");
		System.out.println("1");
		System.out.println("The accurate mincost is "+cost);
	}
	static int tspdp(int c[][], int tour[], int start, int n)
	{
		int mintour[]=new int[10], temp[]=new int[10], mincost=999, ccost, i, j, k;
		if(start == n-1)
		{
			return (c[tour[n-1]][tour[n]] + c[tour[n]][1]);
		}
		for(i=start+1; i<=n; i++)
		{
```

```
                    for(j=1; j<=n; j++)
                        temp[j] = tour[j];
                    temp[start+1] = tour[i];
                    temp[i] = tour[start+1];
                    if((c[tour[start]][tour[i]]+(ccost=tspdp(c,temp,start+1,n)))<mincost)
                    {
                        mincost = c[tour[start]][tour[i]] + ccost;
                        for(k=1; k<=n; k++)
                            mintour[k] = temp[k];
                    }
                }
                for(i=1; i<=n; i++)
                    tour[i] = mintour[i];
                return mincost;
        }
}
```

Input Graph



Output:

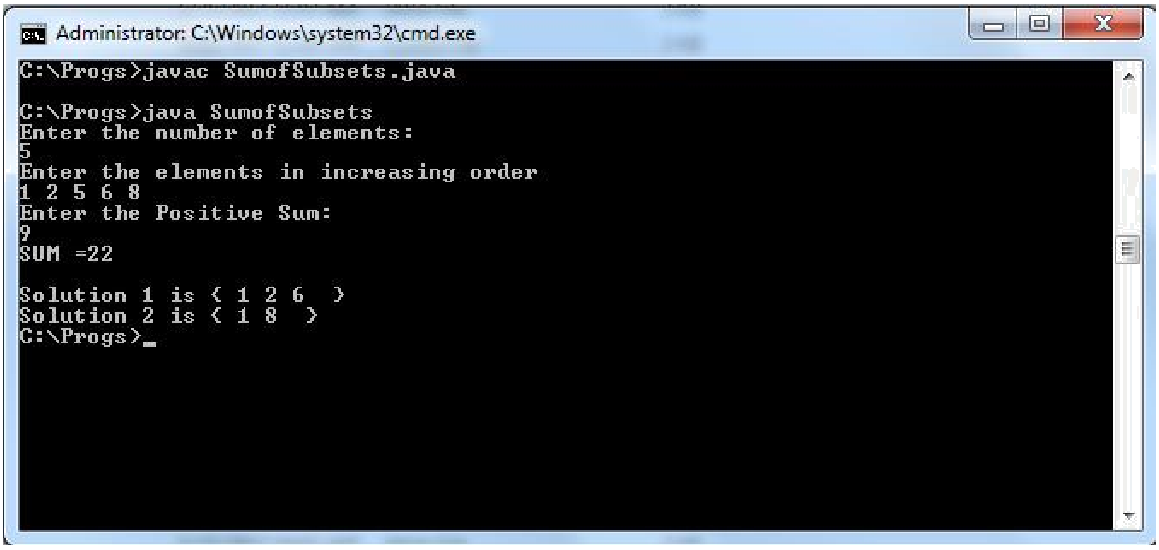


```
C:\Progs>javac Lab10B.java

C:\Progs>java Lab10B
Enter the number of cities:
4
Enter the cost matrix
0 1 6 3
1 0 3 2
6 3 0 1
3 2 1 0
The entered cost matrix is
0       1       6       3
1       0       3       2
6       3       0       1
3       2       1       0
The accurate path is
1->2->3->4->1
The accurate mincost is 8

C:\Progs>_
```

Program – 11 Design and implement in Java to find a subset of a given set S = {S1, S2, ….. Sn} of n positive integers whose SUM is equal to a given positive integer d. For example, if S = {1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

```
import java.io.*;
import java.util.Scanner;

public class SumofSubsets
{
        static int c=0;
        public static void main(String args[])
        {
                int w[]=new int[10], x[]=new int[10], n, d, i, sum=0;
                Scanner in=new Scanner(System.in);

                System.out.println("Enter the number of elements:
                "); n=in.nextInt();
                System.out.println("Enter the elements in increasing
                order"); for(i=0;i<n;i++)
                        w[i]=in.nextInt();
                System.out.println("Enter the Positive Sum: ");
                d=in.nextInt();
                for(i=0;i<n;i++)
                        sum=sum+w[i];
                System.out.println("SUM ="+sum);
                if(sum < d || w[0] > d)
                {
                        System.out.println("Subset is not possible !
                        "); System.exit(0);
                }
                subset(0,0,sum,x,w,d);
                if(c==0)
                        System.out.println("Subset is not possible ! ");
        }
        static void subset(int cs, int k, int r,int x[],int w[],int d)
        {
                x[k] = 1;
                if(cs+w[k] == d)
                {
                        c++;
                        System.out.print("\nSolution "+c+" is { ");
                        for(int i=0;i<=k;i++)
                        {
                                if(x[i] == 1)
                                {
                                        System.out.print(w[i]+" ");
                                }
                        }
                        System.out.print(" }");
                }
                else if((cs + w[k] + w[k+1]) <= d)
                        subset(cs + w[k], k+1, r-w[k],x,w,d);
                if((cs + r - w[k]) >= d && (cs + w[k+1]) <= d)
                {
                        x[k] = 0;
                        subset(cs, k+1, r-w[k],x,w,d);
                }
        }
}
```

Output:

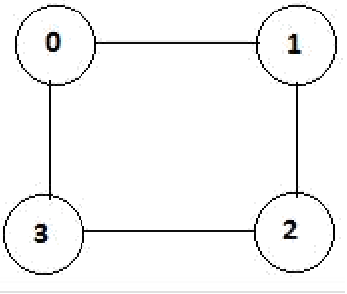


```
C:\Progs>javac SumofSubsets.java

C:\Progs>java SumofSubsets
Enter the number of elements:
5
Enter the elements in increasing order
1 2 5 6 8
Enter the Positive Sum:
9
SUM =22

Solution 1 is { 1 2 6  }
Solution 2 is { 1 8  }
C:\Progs>_
```

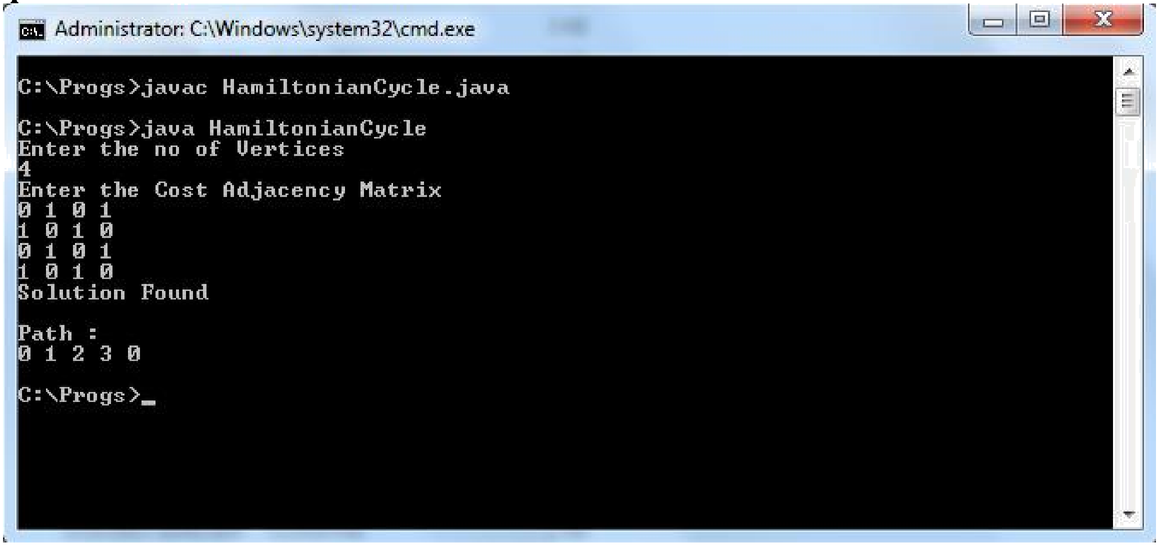
Program – 12 Design and implement the presence of Hamiltonian Cycle in an undirected Graph G of *n* vertices.

```
import java.io.*;
import java.util.Scanner;
import java.util.Arrays;

public class HamiltonianCycle
{
        private int V, pathcount;
        private int path[];
        private int graph[][];
        public void findHamiltonianCycle(int g[][])
        {
                V=g.length;
                path=new int[V];
                Arrays.fill(path, -1);
                graph = g;
                try
                {
                        path[0] = 0;
                        pathcount = 1;
                        solve (0);      //Function call
                        System.out.println("No Solution");
                }
                catch(Exception e)
                {
                        System.out.println(e.getMessage());
                        display();
                }
        }
        public void solve(int vertex)throws Exception
        {
                if(graph[vertex][0] == 1 && pathcount == V)
                        throw new Exception ("Solution Found");
                if(pathcount == V)
                        return;
                for(int v=0; v<V; v++)
                {
                        if(graph[vertex][v] == 1)
                        {
                                path[pathcount++] = v;
                                graph[vertex][v] = graph[v][vertex]= 0;
                                if(!isPresent(v))
                                        solve (v);
                                graph[vertex][v] = graph[v][vertex] = 1;
                                path[--pathcount] = -1;
                        }
                }
        }
        public boolean isPresent(int v)
        {
                for(int i=0; i<pathcount-1; i++)
                        if(path[i] == v)
                                return true;
                        return false;
        }
        public void display()
        {
                System.out.println("\nPath : ");
                for(int i=0; i<=V; i++)
                        System.out.print(path[i%V] + " ");
                System.out.println();
        }
        public static void main(String args[])
        {
                Scanner sc = new Scanner(System.in);
                HamiltonianCycle hc = new
                HamiltonianCycle(); System.out.println("Enter
                the no of Vertices"); int V = sc.nextInt();
                System.out.println("Enter the Cost Adjacency Matrix");
                int graph[][] = new int [V][V]; for (int i=0; i<V; i++)

                        for (int j=0; j<V; j++)
                                graph[i][j] = sc.nextInt();
                hc.findHamiltonianCycle(graph);
        }
}
```

Input Graph



Output:



```
C:\Progs>javac HamiltonianCycle.java

C:\Progs>java HamiltonianCycle
Enter the no of Vertices
4
Enter the Cost Adjacency Matrix
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
Solution Found

Path :
0 1 2 3 0

C:\Progs>_
```

Viva Questions

1. What is Algorithm?
2. Name the design techniques.
3. Which is efficient Sorting technique?
4. Which sorting technique has the lowest worst case efficiency?
5. Which sorting technique is space efficient?
6. Which sorting technique is Time efficient?
7. Define order of growth.
8. How binary search is advantageous over linear search?
9. What is divide & conquer technique?
10. Give few problems which can be solved using divide & conquer technique.
11. What is dynamic programming?
12. Give few problems which can be solved using dynamic programming.
13. What is Back Tracking?
14. Define N-Queens problem.
15. What is Brute force Methodology?
16. What is Greedy Technique?
17. Give few problems which can be solved using Greedy Technique.
18. Which is the efficient method for finding MST?
19. What is MST?
20. What is DFS & BFS?
21. What is a heap?
22. What are P NP problems?
23. Which are the String Matching algorithms?
24. What is Transitive closure?
25. Define Knapsack problem.
26. Define topological sorting?
27. Which algorithm used for checking graph is connected or not?
28. What is Java?
29. List applications of Java?
30. What is the role of static keyword in main function?
31. Why Arrays.fill ( ) method is used?
32. What is constructor? Mention its types.
33. Why Java won’t support copy constructor?
34. Explain Travelling Salesperson problem?
35. What is Hamiltonian cycle?
36. Define Exceptions.
37. Define a Thread. Explain the role of a Thread.
38. What is a Class?
39. What is an Object?
40. What is Reference variable?