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 nStudent objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.

```
import java.util.Scanner;
class Student
{
	String USN, Name, Branch, Phone;
	Scanner input = new Scanner(System.in);
	void read( )
	{
		System.out.println("Enter Student Details");
		System.out.println("Enter USN");
		USN = input.nextLine();
		System.out.println("Enter Name");
		Name = input.nextLine();
		System.out.println("Enter Branch");
		Branch = input.nextLine();
		System.out.println("Enter Phone");
		Phone = input.nextLine();
	}
	void display( )
	{
		System.out.printf("%-20s %-20s %-20s %-20s", USN, Name, Branch, Phone);
	}
}
class studentdetails
{
	public static void main(String[ ] args)
	 {
		Scanner input = new Scanner(System.in);
		System.out.println("Enter number of student details to be created");
		int number = input.nextInt( );
		Student s[ ] = new Student[number];
		// Read student details into array of student objects
		for (int i = 0; i < number; i++)
		{
			s[i] = new Student( );
			s[i].read( );
		}
```

```
                // Display student information
                System.out.printf("%-20s %-20s %-20s %-20s", "USN", "NAME", "BRANCH",
                "PHONE");
                for (int i = 0; i < number; i++)
                {
                        System.out.println( );
                        s[i].display( );
                }
        input.close( );
        }
}
```

OUTPUT :

Enter number of student details to be created
3
Enter Student Details
Enter USN
1BI15CSO31

Enter Name
RAVI

Enter Branch
CSE

Enter Phone
9845964432

Enter Student Details
Enter USN
1B115CS044

Enter Name
RAJ

Enter Branch
CSE

Enter Phone
9845964477

Enter Student Details
Enter USN
1BI15CS421

Enter Name
VASANTH

Enter Branch
CSE

Enter Phone
9731622553

USN	NAME	BRANCH	PHONE
1BI15CSO31	RAVI	CSE	9845964432
1B115CS044	RAJ	CSE	9845964477
1BI15CS421	VASANTH	CSE	9731622553

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.util.*;
class arrayStack
{
        int arr[ ];
        int top, max;
        arrayStack(int n)
        {
                max = n;
                arr = new int[max];
                top = -1;
        }
        void push(int i)
        {
                if (top == max - 1)
                        System.out.println("Stack Overflow");
                else
                        arr[++top] = i;
        }
        void pop( )
        {
                if (top == -1)
                {
                        System.out.println("Stack Underflow");
                }
                else
                {
                        int element = arr[top--];
                        System.out.println("Popped Element:  " + element);
                 }
        }
        void display( )
         {
                System.out.print("\nStack = ");
                if (top == -1)
                {
                        System.out.print("Empty\n");
                        return;
                }
                for (int i = top; i >= 0; i--)
                        System.out.print(arr[i] + " ");
                System.out.println( );
```

```
        }
}
class Stack
{
        public static void main(String[ ] args)
        {
                Scanner scan = new Scanner(System.in);
                System.out.println("Enter Size of Integer Stack ");
                int n = scan.nextInt( );
                boolean done = false;
                arrayStack stk = new arrayStack(n);
                char ch;
                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");
                        int choice = scan.nextInt( );
                        switch (choice)
                        {
                        case 1:
                                System.out.println("Enter integer element to push");
                                stk.push(scan.nextInt( ));
                                break;
                        case 2:
                                stk.pop( );
                                break;
                        case 3:
                                stk.display( );
                                break;
                        case 4:
                                done = true;
                                break;
                        default:
                                System.out.println("Wrong Entry \n ");
                                break;
                        }
                }
                while (!done);
        }
}
```

OUTPUT :

```
Enter Size of Integer Stack
5

Stack Operations
1. push
2. pop
3. display
4. Exit

1
Enter integer element to push
10

Stack Operations
1. push
2. pop
3. display
4. Exit

1
Enter integer element to push
20

Stack Operations
1. push
2. pop
3. display
4. Exit

1
Enter integer element to push
30

Stack Operations
1. push
2. pop
3. display
4. Exit

1
Enter integer element to push
40
```

Stack Operations
1. push
2. pop
3. display
4. Exit

1
Enter integer element to push
50

Stack Operations
1. push
2. pop
3. display
4. Exit

3
Stack = 50 40 30 20 10

Stack Operations
1. push
2. pop
3. display
4. Exit

1
Enter integer element to push
60
Stack Overflow

Stack Operations
1. push
2. pop
3. display
4. Exit

2
Popped Element: 50

Stack Operations
1. push
2. pop
3. display
4. Exit
2
Popped Element: 40

Stack Operations
1. push
2. pop
3. display
4. Exit

2
Popped Element: 30

Stack Operations
1. push
2. pop
3. display
4. Exit

2
Popped Element: 20

Stack Operations
1. push
2. pop
3. display
4. Exit

3
Stack = 10

Stack Operations
1. push
2. pop
3. display
4. Exit

2
Popped Element: 10

Stack Operations
1. push
2. pop
3. display
4. Exit

2
Stack Underflow

Stack Operations
1. push
2. pop
3. display
4. Exit

4

Program – 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.

```
import java.util.Scanner;
class Staff
{
	String StaffID, Name, Phone, Salary;
	Scanner input = new Scanner(System.in);
	void read( )
	{
		System.out.println("Enter StaffID");
		StaffID = input.nextLine( );
		System.out.println("Enter Name");
		Name = input.nextLine( );
		System.out.println("Enter Phone");
		Phone = input.nextLine( );
		System.out.println("Enter Salary");
		Salary = input.nextLine( );
	}
	void display( )
	{
		System.out.printf("\n%-15s", "STAFFID: ");
		System.out.printf("%-15s \n", StaffID);
		System.out.printf("%-15s", "NAME: ");
		System.out.printf("%-15s \n", Name);
		System.out.printf("%-15s", "PHONE:");
		System.out.printf("%-15s \n", Phone);
		System.out.printf("%-15s", "SALARY:");
		System.out.printf("%-15s \n", Salary);
	}
}
class Teaching extends Staff
{
	String Domain, Publication;
	void read_Teaching( )
	{
		super.read( ); // call super class read method
		System.out.println("Enter Domain");
		Domain = input.nextLine( );
		System.out.println("Enter Publication");
		Publication = input.nextLine( );
	}
```

```
        void display( )
        {
                super.display( ); // call super class display( ) method
                System.out.printf("%-15s", "DOMAIN:");
                System.out.printf("%-15s \n", Domain);
                System.out.printf("%-15s", "PUBLICATION:");
                System.out.printf("%-15s \n", Publication);
        }
}
class Technical extends Staff
{
        String Skills;
        void read_Technical( )
        {
                super.read( ); // call super class read method
                System.out.println("Enter Skills");
                Skills = input.nextLine( );
        }
        void display( )
        {
                super.display( ); // call super class display( ) method
                System.out.printf("%-15s", "SKILLS:");
                System.out.printf("%-15s \n", Skills);
        }
}
class Contract extends Staff
{
        String Period;
        void read_Contract( )
        {
                super.read( ); // call super class read method
                System.out.println("Enter Period");
                Period = input.nextLine( );
        }
        void display( )
        {
                super.display( ); // call super class display() method
                System.out.printf("%-15s", "PERIOD:");
                System.out.printf("%-15s \n", Period);
        }
}
class Staffdetails
{
        public static void main(String[ ] args)
        {
                Scanner input = new Scanner(System.in);
```

```
                System.out.println("Enter number of staff details to be created");
                int n = input.nextInt( );
                Teaching steach[] = new Teaching[n];
                Technical stech[] = new Technical[n];
                Contract scon[] = new Contract[n];
                // Read Staff information under 3 categories
                for (int i = 0; i < n; i++)
                {
                        System.out.println("Enter Teaching staff information");
                        steach[i] = new Teaching( );
                        steach[i].read_Teaching ( );
                }
                for (int i = 0; i < n; i++)
                {
                        System.out.println("Enter Technical staff information");
                        stech[i] = new Technical( );
                        stech[i].read_Technical( );
                }
                for (int i = 0; i < n; i++)
                 {
                        System.out.println("Enter Contract staff information");
                        scon[i] = new Contract( );
                        scon[i].read_Contract( );
                }
                // Display Staff Information
                System.out.println("\n STAFF DETAILS: \n");
                System.out.println("-----TEACHING STAFF DETAILS----- ");
                for (int i = 0; i < n; i++)
                {
                        steach[i].display( );
                }
                System.out.println( );
                System.out.println("-----TECHNICAL STAFF DETAILS-----");
                for (int i = 0; i < n; i++)
                {
                        stech[i].display( );
                }
                System.out.println( );
                System.out.println("-----CONTRACT STAFF DETAILS-----");
                for (int i = 0; i < n; i++)
                {
                        scon[i].display( );
                }
                        input.close();
        }
}
```

OUTPUT :

Enter number of staff details to be created
2
Enter Teaching staff information
Enter StaffID
S1

Enter Name
VINAY

Enter Phone
995463728

Enter Salary
25000

Enter Domain
DATA MINING

Enter Publication
Elsevier Publications

Enter Teaching staff information
Enter StaffID
S2

Enter Name
RAM

Enter Phone
2345618645

Enter Salary
30000

Enter Domain
IMAGE PROCESSING

Enter Publication
Elsevier Publications

Enter Technical staff information
Enter StaffID
S3

Enter Name
SHAM

Enter Phone
2854637290

Enter Salary
30000

Enter Skills
NETWORKING

Enter Technical staff information
Enter StaffID
S4

Enter Name
MANJU

Enter Phone
9864356245

Enter Salary
35000

Enter Skills
GRAPHICS

Enter Contract staff information
Enter StaffID
S5

Enter Name
RAMYA

Enter Phone
8769548675

Enter Salary
24000

Enter Period
3 YEARS

Enter Contract staff information
Enter StaffID
S6

Enter Name
HEMANTH

Enter Phone
8765432975

Enter Salary
15000

Enter Period
2 YEARS

STAFF DETAILS:

-----TEACHING STAFF DETAILS-----
STAFFID: S1
NAME: VINAY
PHONE: 995463728
SALARY: 25000
DOMAIN: DATA MINING
PUBLICATION: Elsevier Publications

STAFFID: S2
NAME: RAM
PHONE: 2345618645
SALARY: 30000
DOMAIN: IMAGE PROCESSING
PUBLICATION: Elsevier Publications

-----TECHNICAL STAFF DETAILS-----
STAFFID: S3
NAME: SHAM
PHONE: 2854637290
SALARY: 30000
SKILLS: NETWORKING

STAFFID: S4
NAME: MANJU
PHONE: 9864356245
SALARY: 35000
SKILLS: GRAPHICS

-----CONTRACT STAFF DETAILS-----

STAFFID: S5
NAME: RAMYA
PHONE: 8769548675
SALARY: 24000
PERIOD: 3 YEARS

STAFFID: S6
NAME: HEMANTH
PHONE: 8765432975
SALARY: 15000
PERIOD: 2 YEARS

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 StringTokenizer class considering the delimiter character as "/".

```
import java.util.Scanner;
import java.util.StringTokenizer;
public class Customer
{
        public static void main(String[ ] args)
        {
                String name;
                Scanner scan = new Scanner(System.in);
                System.out.println("Enter Name and Date_of_Birth in the format
                Name,DD/MM/YYYY>");
                name = scan.next( );
                // create stringTokenizer with delimiter "/"
                StringTokenizer st = new StringTokenizer(name, ",/");
                // Count the number of tokens
                int count = st.countTokens( );
                // Print one token at a time and induce new delimiter ","
                for (int i = 1; i <= count && st.hasMoreTokens( ); i++)
                {
                        System.out.print(st.nextToken( ));
                        if (i < count)
                        System.out.print(",");
                }
        }
}
```

OUTPUT 1:

```
Enter Name and Date_of_Birth in the format <Name,DD/MM/YYYY>
<PRANAV,15/05/2008>
<PRANAV,15,05,2008>
```

OUTPUT 2:

```
Enter Name and Date_of_Birth in the format <Name,DD/MM/YYYY>
<PRIYA,08/09/1981>
<PRIYA,08,09,1981>
```

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.util.Scanner;
class exception
{
	public static void main(String[ ] args)
	{
		int a, b, result;
		Scanner input = new Scanner(System.in);
		System.out.println("Input two integers");
		a = input.nextInt( );
		b = input.nextInt( );
		try
		{
			result = a / b;
			System.out.println("Result = " + result);
		}
		catch (ArithmeticException e)
		{
			System.out.println("Exception caught: Division by zero.");
		}
	}
}
```

OUTPUT 1:

```
Input two integers
100
2
Result = 50
```

OUTPUT 2:

```
Input two integers
200
0
Exception caught: Division by zero.
```

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.Random;
class SquareThread implements Runnable
{
        int x;
        SquareThread(int x)
        {
                this.x = x;
        }
        public void run( )
        {
           System.out.println("Thread Name:Square Thread and Square of " + x + " is: " + x * x);
        }
}
class CubeThread implements Runnable
{
        int x;
        CubeThread(int x)
        {
                this.x = x;
        }
        public void run( )
        {
           System.out.println("Thread Name:Cube Thread and Cube of " + x + " is: " + x * x * x);
        }
}
class RandomThread implements Runnable
{
        Random r;
        Thread t2, t3;
        public void run( )
        {
                int num;
                r = new Random( );
                try
                {
                        while (true)
                        {
                                num = r.nextInt(100);
                                System.out.println("Main Thread and Generated Number is " + num);
                                t2 = new Thread(new SquareThread(num));
```

```
                        t2.start( );
                        t3 = new Thread(new CubeThread(num));
                        t3.start( );
                        Thread.sleep(1000);
                        System.out.println("--------------------------------------");
                    }
                }
                catch (Exception ex)
                {
                    System.out.println("Interrupted Exception");
                }
        }
}
public class MainThread
{
        public static void main(String[ ] args)
        {
                RandomThread thread_obj = new RandomThread( );
                Thread t1 = new Thread(thread_obj);
                t1.start( );
        }
}
```

OUTPUT :

```
Main Thread and Generated Number is 96
Thread Name:Square Thread and Square of 96 is: 9216
Thread Name:Cube Thread and Cube of 96 is: 884736
-------------------------------------
Main Thread and Generated Number is 34
Thread Name:Cube Thread and Cube of 34 is: 39304
Thread Name:Square Thread and Square of 34 is: 1156
-------------------------------------
Main Thread and Generated Number is 71
Thread Name:Square Thread and Square of 71 is: 5041
Thread Name:Cube Thread and Cube of 71 is: 357911
-------------------------------------
Main Thread and Generated Number is 97
Thread Name:Square Thread and Square of 97 is: 9409
Thread Name:Cube Thread and Cube of 97 is: 912673
-------------------------------------
Main Thread and Generated Number is 68
Thread Name:Cube Thread and Cube of 68 is: 314432
Thread Name:Square Thread and Square of 68 is: 4624
-------------------------------------
```

```
Main Thread and Generated Number is 54
Thread Name:Square Thread and Square of 54 is: 2916
Thread Name:Cube Thread and Cube of 54 is: 157464
-------------------------------------
Main Thread and Generated Number is 53
Thread Name:Square Thread and Square of 53 is: 2809
Thread Name:Cube Thread and Cube of 53 is: 148877
-------------------------------------
Main Thread and Generated Number is 91
Thread Name:Square Thread and Square of 91 is: 8281
Thread Name:Cube Thread and Cube of 91 is: 753571
-------------------------------------
Main Thread and Generated Number is 32
Thread Name:Square Thread and Square of 32 is: 1024
Thread Name:Cube Thread and Cube of 32 is: 32768
-------------------------------------
Main Thread and Generated Number is 73
Thread Name:Square Thread and Square of 73 is: 5329
Thread Name:Cube Thread and Cube of 73 is: 389017
-------------------------------------
Main Thread and Generated Number is 55
Thread Name:Square Thread and Square of 55 is: 3025
Thread Name:Cube Thread and Cube of 55 is: 166375
-------------------------------------
Main Thread and Generated Number is 43
Thread Name:Square Thread and Square of 43 is: 1849
Thread Name:Cube Thread and Cube of 43 is: 79507
-------------------------------------
Main Thread and Generated Number is 84
Thread Name:Square Thread and Square of 84 is: 7056
Thread Name:Cube Thread and Cube of 84 is: 592704
```

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.util.Scanner;
import java.util.Arrays;
import java.util.Random;
public class QuickSortComplexity
{
        static final int MAX = 10005;
        static int[] a = new int[MAX];
        public static void main(String[ ] args)
        {
                Scanner input = new Scanner(System.in);
                System.out.print("Enter Max array size: ");
                int n = input.nextInt();
                Random random = new Random();
                // System.out.println("Enter the array elements: ");
                for (int i = 0; i < n; i++)
                        // a[i] = input.nextInt();   // for keyboard entry
                        a[i] = random.nextInt(1000); // generate
                        random numbers – uniform distribution
                        a = Arrays.copyOf(a, n); // keep only non zero elements
                       // QuickSortAlgorithm(0, n - 1);// for worst-case time complexity
                // System.out.println("Input Array:");
                // for (int i = 0; i < n; i++)
                        // System.out.print(a[i] + " ");
                       // set start time
                long startTime = System.nanoTime( );
                QuickSortAlgorithm(0, n - 1);
                long stopTime = System.nanoTime( );
                long elapsedTime = stopTime - startTime;
                /* System.out.println("\nSorted Array:");
                for (int i = 0; i < n; i++)
                        System.out.print(a[i] + " ");
                System.out.println( ); */
                System.out.println("Time Complexity in ms for
              n=" + n + " is: " + (double) elapsedTime / 1000000);
        }
        public static void QuickSortAlgorithm(int p, int r)
        {
```

```
        int i, j, temp, pivot;
        if (p < r)
        {
            i = p;
            j = r + 1;
            pivot = a[p]; // mark first element as pivot
            while (true)
            {
                i++;
                while (a[i] < pivot && i < r)
                    i++;
                    j--;
                while (a[j] > pivot)
                    j--;
                if (i < j)
                {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
                else
                break; // partition is over
            }
            a[p] = a[j];
            a[j] = pivot;
            QuickSortAlgorithm(p, j - 1);
            QuickSortAlgorithm(j + 1, r);
        }
    }
}
```

OUTPUT : Basic – QuickSort

```
Enter Max array size: 5
Enter the array elements:
4 2 6 7 3
Input Array:
4 2 6 7 3
Sorted Array:
2 3 4 6 7
Time Complexity in ms for n=5 is: 0.009682.
```

OUTPUT : Best & Average Case – QuickSort

Enter Max array size: 5000
Time Complexity in ms for n=5000 is: 4.713954

Enter Max array size: 6000
Time Complexity in ms for n=6000 is: 7.386643

Enter Max array size: 7000
Time Complexity in ms for n=7000 is: 7.838167

Enter Max array size: 8000
Time Complexity in ms for n=8000 is: 8.819339

Enter Max array size: 9000
Time Complexity in ms for n=9000 is: 8.910286

Enter Max array size: 10000
Time Complexity in ms for n=10000 is: 11.698069

OUTPUT : Worst Case – QuickSort

Enter Max array size: 5000
Time Complexity in ms for n=5000 is: 36.561776

Enter Max array size: 6000
Time Complexity in ms for n=6000 is: 41.323717

Enter Max array size: 7000
Time Complexity in ms for n=7000 is: 55.416783

Enter Max array size: 8000
Time Complexity in ms for n=8000 is: 60.235093

Enter Max array size: 9000
Time Complexity in ms for n=9000 is: 70.381734

Enter Max array size: 10000
Time Complexity in ms for n=10000 is: 80.835902

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.util.Random;
import java.util.Scanner;
public class MergeSort2
{
	static final int MAX = 10005;
	static int[] a = new int[MAX];
	public static void main(String[ ] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.print("Enter Max array size: ");
		int n = input.nextInt( );
		Random random = new Random( );
		// System.out.println("Enter the array elements: ");
		for (int i = 0; i < n; i++)
			//a[i] = input.nextInt(); // for keyboard entry
			a[i] = random.nextInt(1000); // generate random numbers –
			// uniform distribution
		// MergeSortAlgorithm(0, n - 1);
		long startTime = System.nanoTime();
		MergeSortAlgorithm(0, n - 1);
		long stopTime = System.nanoTime();
		long elapsedTime = stopTime - startTime;
		System.out.println("Time Complexity (ms) for n = " +
		n + " is : " + (double) elapsedTime / 1000000);
		// System.out.println("Sorted Array (Merge Sort):");
		// for (int i = 0; i < n; i++)
			// System.out.print(a[i] + " ");
		input.close();
	}
	public static void MergeSortAlgorithm(int low, int high)
	{
		int mid;
		if (low < high)
		 {
			mid = (low + high) / 2;
			MergeSortAlgorithm(low, mid);
			MergeSortAlgorithm(mid + 1, high);
```

```
                Merge(low, mid, high);
            }
        }

        public static void Merge(int low, int mid, int high)
        {
            int[ ] b = new int[MAX];
            int i, h, j, k;
            h = i = low;
            j = mid + 1;
            while ((h <= mid) && (j <= high))
                if (a[h] < a[j])
                    b[i++] = a[h++];
                else
                    b[i++] = a[j++];
            if (h > mid)
                for (k = j; k <= high; k++)
                    b[i++] = a[k];
            else
                for (k = h; k <= mid; k++)
                    b[i++] = a[k];
            for (k = low; k <= high; k++)
                a[k] = b[k];
        }
}
```

OUTPUT : Basic - Merge Sort

```
Enter Max array size: 8
Enter the array elements:
30 25 15 50 28 49 88 33
Time Complexity (ms) for n = 8 is : 0.353131
Sorted Array (Merge Sort):
15 25 28 30 33 49 50 88
```

OUTPUT : Best & Averege Case - Merge Sort

```
Enter Max array size: 5000
Time Complexity (ms) for n = 5000 is : 48.192483

Enter Max array size: 6000
Time Complexity (ms) for n = 6000 is : 55.245321

Enter Max array size: 7000
Time Complexity (ms) for n = 7000 is : 63.571278
```

Enter Max array size: 8000
Time Complexity (ms) for n = 8000 is : 68.213534

Enter Max array size: 9000
Time Complexity (ms) for n = 9000 is : 76.808363

Enter Max array size: 10000
Time Complexity (ms) for n = 10000 is : 80.498525

OUTPUT : Worst Case - Merge Sort

Enter Max array size: 5000
Time Complexity (ms) for n = 5000 is : 65.638676

Enter Max array size: 6000
Time Complexity (ms) for n = 6000 is : 68.691028

Enter Max array size: 7000
Time Complexity (ms) for n = 7000 is : 76.250782

Enter Max array size: 8000
Time Complexity (ms) for n = 8000 is : 85.291611

Enter Max array size: 9000
Time Complexity (ms) for n = 9000 is : 108.015192

Enter Max array size: 10000
Time Complexity (ms) for n = 10000 is : 112.477093

Program – 6

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

(a) Dynamic Programming method :

```
import java.util.Scanner;
public class KnapsackDP
{
	static final int MAX = 20; // max. no. of objects
	static int w[ ];  // weights 0 to n-1
	static int p[ ];   // profits 0 to n-1
	static int n;		// no. of objects
	static int M;		// capacity of Knapsack
	static int V[ ][ ];	// DP solution process - table
	static int Keep[ ] [ ]; // to get objects in optimal solution
	public static void main(String args[ ])
	{
		w = new int[MAX];
		p = new int[MAX];
		V = new int [MAX][MAX];
		Keep = new int[MAX][MAX];
		int optsoln;
		ReadObjects( );
		for (int i = 0; i <= M; i++)
			V[0][i] = 0;
		for (int i = 0; i <= n; i++)
			V[i][0] = 0;
		optsoln = Knapsack( );
		System.out.println("Optimal solution = " + optsoln);
	}
	static int Knapsack( )
	{
		int r; // remaining Knapsack capacity
		for (int i = 1; i <= n; i++)
			for (int j = 0; j <= M; j++)
				if ((w[i] <= j) && (p[i] + V[i - 1][j - w[i]] > V[i - 1][j]))
				{
					V[i][j] = p[i] + V[i - 1][j - w[i]];
					Keep[i][j] = 1;
				}
				else
				{
					V[i][j] = V[i - 1][j];
					Keep[i][j] = 0;
				}
```

```
                // Find the objects included in the Knapsack
                r = M;
                System.out.println("Items = ");
                for (int i = n; i > 0; i--) // start from Keep[n,M]
                        if (Keep[i][r] == 1)
                        {
                                System.out.println(i + " ");
                                r = r - w[i];
                        }
                System.out.println( );
                return V[n][M];
        }
        static void ReadObjects( )
        {
                Scanner scanner = new Scanner(System.in);
                System.out.println("Knapsack Problem - Dynamic Programming Solution: ");
                System.out.println("Enter the max capacity of knapsack: ");
                M = scanner.nextInt( );
                System.out.println("Enter number of objects:  ");
                n = scanner.nextInt( );
                System.out.println("Enter Weights: ");
                for (int i = 1; i <= n; i++)
                        w[i] = scanner.nextInt( );
                System.out.println("Enter Profits: ");
                for (int i = 1; i <= n; i++)
                        p[i] = scanner.nextInt( );
                scanner.close( );
        }
}
```

OUTPUT 1:

```
Enter the max capacity of knapsack:
10
Enter number of objects:
3
Enter Weights:
2 6 5
Enter Profits:
10 20 30
Items:
3 1

Optimal solution = 40
```

OUTPUT 2:

Enter the max capacity of knapsack:
5
Enter number of objects:
4
Enter Weights:
2 1 3 2
Enter Profits:
12 10 20 15
Items:
4 2 1

Optimal solution = 37

(b) Greedy Method :

```
import java.util.Scanner;
class KObject
{                                       // Knapsack object details
        float w;
        float p;
        float r;
}
public class KnapsackGreedy2
{
        static final int MAX = 20;      // max. no. of objects
        static int n;                   // no. of objects
        static float M;                 // capacity of Knapsack
        public static void main(String args[ ])
        {
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter number of objects:  ");
                n = scanner.nextInt( );
                KObject[ ] obj = new KObject[n];
                for(int i = 0; i<n;i++)
                        obj[i] = new KObject( );// allocate memory for members
                ReadObjects(obj);
                Knapsack(obj);
                scanner.close();
        }
        static void ReadObjects(KObject obj[ ])
        {
                KObject temp = new KObject( );
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter the max capacity of knapsack: ");
                M = scanner.nextFloat( );
                System.out.println("Enter Weights: ");
                for (int i = 0; i < n; i++)
                        obj[i].w = scanner.nextFloat( );
                System.out.println("Enter Profits: ");
                for (int i = 0; i < n; i++)
                        obj[i].p = scanner.nextFloat( );
                for (int i = 0; i < n; i++)
                        obj[i].r = obj[i].p / obj[i].w;
                // sort objects in descending order, based on p/w ratio
                for(int i = 0; i<n-1; i++)
                        for(int j=0; j<n-1-i; j++)
                                if(obj[j].r < obj[j+1].r)
                                {
                                        temp = obj[j];
```

```
                                        obj[j] = obj[j+1];
                                        obj[j+1] = temp;
                                }
                scanner.close( );
        }
        static void Knapsack(KObject kobj[ ])
        {
                float x[ ] = new float[MAX];
                float totalprofit;
                int i;
                float U;  // U place holder for M
                U = M;
                totalprofit = 0;
                for (i = 0; i < n; i++)
                        x[i] = 0;
                for (i = 0; i < n; i++)
                {
                        if (kobj[i].w > U)
                                break;
                        else
                        {
                                x[i] = 1;
                                totalprofit = totalprofit + kobj[i].p;
                                U = U - kobj[i].w;
                        }
                }
                System.out.println("i = " + i);
                if (i < n)
                        x[i] = U / kobj[i].w;
                totalprofit = totalprofit + (x[i] * kobj[i].p);
                System.out.println("The Solution vector, x[ ]: ");
                for (i = 0; i < n; i++)
                        System.out.print(x[i] + " ");
                System.out.println("\nTotal profit is = " + totalprofit);
        }
}
```

OUTPUT 1:

Enter number of objects:
3

Enter the max capacity of knapsack:
20

Enter Weights:
18 15 10

Enter Profits:
25 24 15

i = 1

The Solution vector, x[ ]:
1.0 0.5 0.0

Total profit is = 31.5

OUTPUT 2:

Enter number of objects:
3

Enter the max capacity of knapsack:
30

Enter Weights:
20 15 15

Enter Profits:
40 25 25

i = 1

The Solution vector, x[ ]:
1.0 0.6666667 0.0

Total profit is = 56.666668

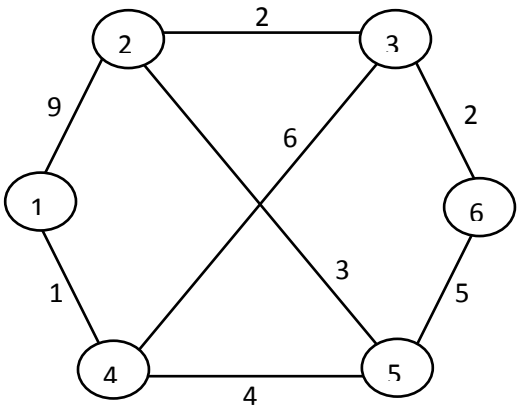
Program – 7

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

```
import java.util.*;
public class DijkstrasClass
{
        final static int MAX = 20;
        final static int infinity = 9999;
        static int n;               // No. of vertices of G
        static int a[ ][ ];         // Cost matrix
        static Scanner scan = new Scanner(System.in);
        public static void main(String[ ] args)
        {
                ReadMatrix( );
                int s = 0;                  // starting vertex
                System.out.println("Enter starting vertex: ");
                s = scan.nextInt( );
                Dijkstras(s);    // find shortest path
        }
        static void ReadMatrix( )
        {
                a = new int[MAX][MAX];
                System.out.println("Enter the number of vertices:");
                n = scan.nextInt( );
                System.out.println("Enter the cost adjacency matrix:");
                for (int i = 1; i <= n; i++)
                        for (int j = 1; j <= n; j++)
                                a[i][j] = scan.nextInt( );
        }
        static void Dijkstras(int s)
        {
                int S[ ] = new int[MAX];
                int d[ ] = new int[MAX];
                int u, v;
                int i;
                for (i = 1; i <= n; i++)
                {
                        S[i] = 0;
                        d[i] = a[s][i];
                }
                S[s] = 1;
                d[s] = 1;
                i = 2;
                while (i <= n)
```

```
		{
			u = Extract_Min(S, d);
			S[u] = 1;
			i++;
			for (v = 1; v <= n; v++)
			{
				if (((d[u] + a[u][v] < d[v]) && (S[v] == 0)))
					d[v] = d[u] + a[u][v];
			}
		}
		for (i = 1; i <= n; i++)
			if (i != s)
				System.out.println(i + ":" + d[i]);
	}
	static int Extract_Min(int S[ ], int d[ ])
	{
		int i, j = 1, min;
		min = infinity;
		for (i = 1; i <= n; i++)
		 {
			if ((d[i] < min) && (S[i] == 0))
			{
				min = d[i];
				j = i;
			}
		}
		return (j);
	}
}
```

GRAPH:



OUTPUT 1:

Enter the number of vertices:
6

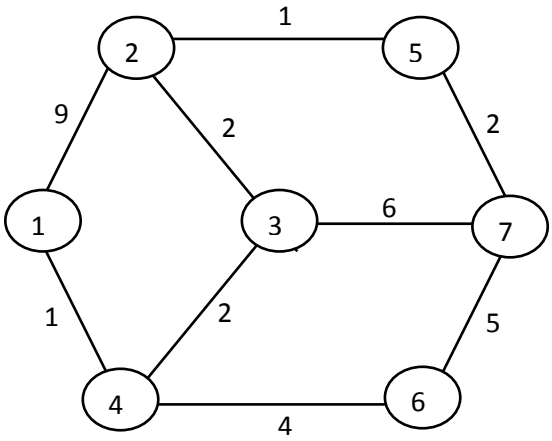
Enter the cost adjacency matrix:

0	9	9999	1	9999	9999
9	0	2	9999	3	9999
9999	2	0	6	9999	2
1	9999	6	0	4	9999
9999	3	9999	4	0	5
9999	9999	2	9999	5	0

Enter starting vertex:
6

1:9
2:4
3:2
4:8
5:5

GRAPH:



OUTPUT 2:

Enter the number of vertices:
7

Enter the cost adjacency matrix:

0	9	9999	1	9999	9999	9999
9	0	2	9999	1	9999	9999
9999	2	0	2	9999	9999	6
1	9999	2	0	9999	4	9999
9999	1	9999	9999	0	9999	2
9999	9999	9999	4	9999	0	5
9999	9999	6	9999	2	5	0

Enter starting vertex:
1

2:5
3:3
4:1
5:6
6:5
7:8

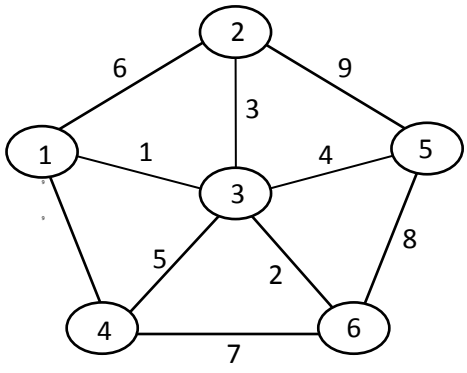
Program – 8

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

```
import java.util.Scanner;
public class KruskalsClass
{
	final static int MAX = 20;
	static int n; // No. of vertices of G
	static int cost[ ][ ]; // Cost matrix
	static Scanner scan = new Scanner(System.in);
	public static void main(String[ ] args)
	{
		ReadMatrix( );
		Kruskals( );
	}
	static void ReadMatrix( )
	{
		int i, j;
		cost = new int[MAX][MAX];
		System.out.println("Implementation of Kruskal's algorithm");
		System.out.println("Enter the no. of vertices");
		n = scan.nextInt( );
		System.out.println("Enter the cost adjacency matrix");
		for (i = 1; i <= n; i++)
		{
			for (j = 1; j <= n; j++)
			{
				cost[i][j] = scan.nextInt( );
				if (cost[i][j] == 0)
					cost[i][j] = 999;
			}
		}
	}
	static void Kruskals( )
	 {
		int a = 0, b = 0, u = 0, v = 0, i, j, ne = 1, min, mincost = 0;
		System.out.println("The edges of Minimum Cost Spanning Tree are");
		while (ne < n)
		{
			for (i = 1, min = 999; i <= n; i++)
			 {
				for (j = 1; j <= n; j++)
				{
					if (cost[i][j] < min)
```

```
                                {
                                        min = cost[i][j];
                                        a = u = i;
                                        b = v = j;
                                }
                        }
                }
                u = find(u);
                v = find(v);
                if (u != v)
                {
                        uni(u, v);
                        System.out.println(ne++ + "edge (" + a + "," + b + ") =" + min);
                        mincost += min;
                }
                cost[a][b] = cost[b][a] = 999;
        }
        System.out.println("Minimum cost :" + mincost);
    }
    static int find(int i)
    {
        int parent[ ] = new int[9];
        while (parent[i] == 1)
                i = parent[i];
        return i;
    }
    static void uni(int i, int j)
    {
        int parent[ ] = new int[9];
        parent[j] = i;
    }
}
```

OUTPUT 1:
GRAPH :



Implementation of Kruskal's algorithm

Enter the no. of vertices
6

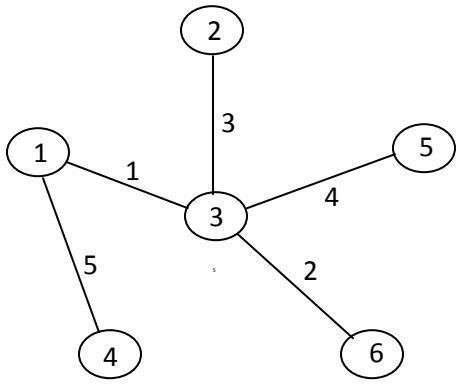
Enter the cost adjacency matrix

0	6	1	5	999	999
6	0	3	999	9	999
1	3	0	5	4	2
5	999	5	0	999	7
999	9	4	999	0	8
999	999	2	7	8	0

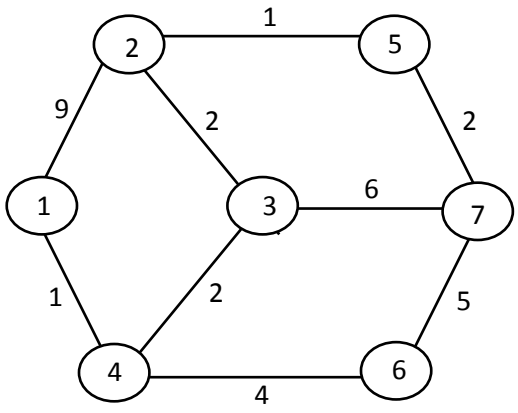
The edges of Minimum Cost Spanning Tree are
1edge (1,3) =1
2edge (3,6) =2
3edge (2,3) =3
4edge (3,5) =4
5edge (1,4) =5

Minimum cost :15

MST :



OUTPUT 2:
GRAPH :



Implementation of Kruskal's algorithm

Enter the no. of vertices
7

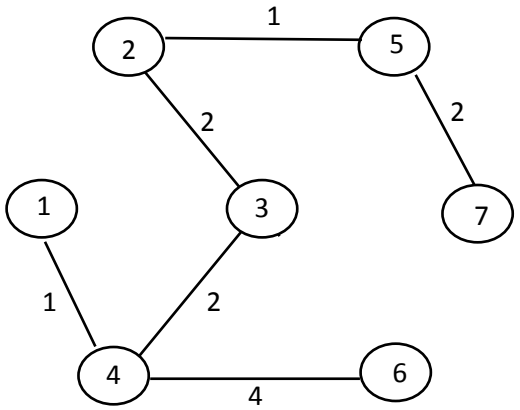
Enter the cost adjacency matrix

0	9	999	1	999	999	999
9	0	2	999	1	999	999
999	2	0	2	999	999	6
1	999	2	0	999	4	999
999	1	999	999	0	999	2
999	999	999	4	999	0	5
999	999	6	999	2	5	0

The edges of Minimum Cost Spanning Tree are
1edge (1,4) =1
2edge (2,5) =1
3edge (2,3) =2
4edge (3,4) =2
5edge (5,7) =2
6edge (4,6) =4

Minimum cost :12

MST :



Program – 9

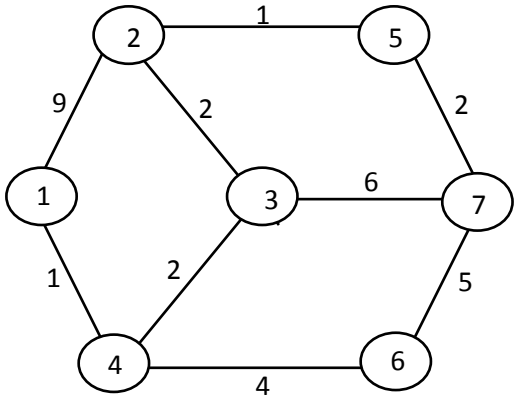
Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
import java.util.Scanner;
public class PrimsClass
{
	final static int MAX = 20;
	static int n; // No. of vertices of G
	static int cost[ ][ ]; // Cost matrix
	static Scanner scan = new Scanner(System.in);
	public static void main(String[ ] args)
	{
		ReadMatrix( );
		Prims( );
	}
	static void ReadMatrix( )
	{
		int i, j;
		cost = new int[MAX][MAX];
		System.out.println("\n Enter the number of nodes:");
		n = scan.nextInt( );
		System.out.println("\n Enter the adjacency matrix:\n");
		for (i = 1; i <= n; i++)
			for (j = 1; j <= n; j++)
			{
				cost[i][j] = scan.nextInt( );
				if (cost[i][j] == 0)
					cost[i][j] = 999;
			}
	}
	static void Prims( )
	{
		int visited[ ] = new int[10];
		int ne = 1, i, j, min, a = 0, b = 0, u = 0, v = 0;
		int mincost = 0;
		visited[1] = 1;
		while (ne < n)
		{
			for (i = 1, min = 999; i <= n; i++)
				for (j = 1; j <= n; j++)
					if (cost[i][j] < min)
						if (visited[i] != 0)
						{
							min = cost[i][j];
							a = u = i;
```

```
                                        b = v = j;
                                    }
                if (visited[u] == 0 || visited[v] == 0)
                {
                        System.out.println("Edge" + ne++ + ":(" + a + "," + b + ")" +
                        "cost:" + min);
                        mincost += min;
                        visited[b] = 1;
                }
                cost[a][b] = cost[b][a] = 999;
        }
        System.out.println("\n Minimum cost" + mincost);
    }

}
```

OUTPUT 1:
GRAPH :



Enter the number of nodes:
7

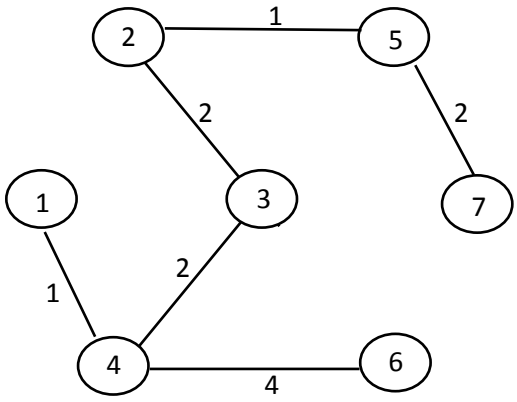
Enter the adjacency matrix:

0	9	999	1	999	999	999
9	0	2	999	1	999	999
999	2	0	2	999	999	6
1	999	2	0	999	4	999
999	1	999	999	0	9999	2
999	999	999	4	9999	0	5
999	999	6	999	2	5	0

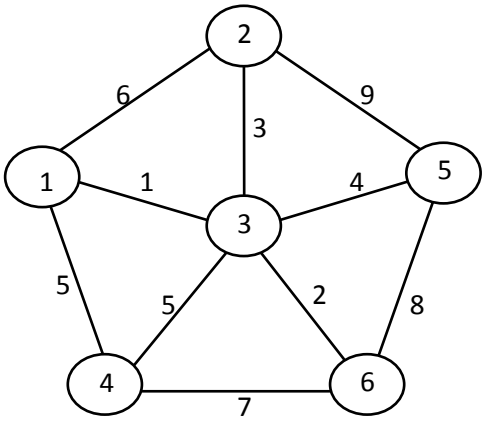
Edge1: (1,4) cost:1
Edge2: (4,3) cost:2
Edge3: (3,2) cost:2
Edge4: (2,5) cost:1
Edge5: (5,7) cost:2
Edge6: (4,6) cost:4

Minimum cost 12

MST :



OUTPUT 2:
GRAPH :



Enter the number of nodes:
6

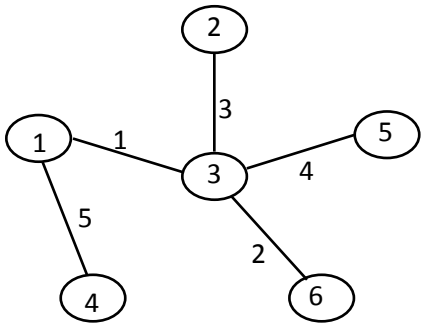
Enter the adjacency matrix:

0	6	1	5	999	999
6	0	3	999	9	999
1	3	0	5	4	2
5	999	5	0	999	7
999	9	4	999	0	8
999	999	2	7	8	0

Edge1: (1,3) cost:1
Edge2: (3,6) cost:2
Edge3: (3,2) cost:3
Edge4: (3,5) cost:4
Edge5: (1,4) cost:5

Minimum cost 15

MST :



Program – 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.

(a). Floyd's algorithm :

```
import java.util.Scanner;
public class FloydsClass
{
        static final int MAX = 20;      // max. size of cost matrix
        static int a[ ][ ];                     // cost matrix
        static int n;                           // actual matrix size

        public static void main(String args[ ])
        {
                a = new int[MAX][MAX];
                ReadMatrix( );
                Floyds( );                              // find all pairs shortest path
                PrintMatrix( );
        }
        static void ReadMatrix( )
         {
                System.out.println("Enter the number of vertices\n");
                Scanner scanner = new Scanner(System.in);
                n = scanner.nextInt( );
                System.out.println("Enter the Cost Matrix (999 for infinity) \n");
                for (int i = 1; i <= n; i++)
                {
                        for (int j = 1; j <= n; j++)
                        {
                                a[i][j] = scanner.nextInt( );
                        }
                }
                scanner.close( );
        }
        static void Floyds( )
        {
                for (int k = 1; k <= n; k++)
                {
                        for (int i = 1; i <= n; i++)
                                for (int j = 1; j <= n; j++)
                                        if ((a[i][k] + a[k][j]) < a[i][j])
                                                a[i][j] = a[i][k] + a[k][j];
                }
        }
```

```
        static void PrintMatrix( )
        {
                System.out.println("The All Pair Shortest Path Matrix is:\n");
                for(int i=1; i<=n; i++)
                {
                        for(int j=1; j<=n; j++)
                                System.out.print(a[i][j] + "\t");
                        System.out.println("\n");
                }
        }
}
```

OUTPUT 1:

Enter the number of vertices
4

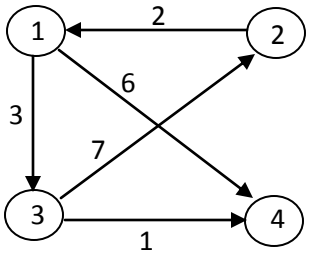
Enter the Cost Matrix (999 for infinity)

0	999	3	6
2	0	999	999
999	7	0	1
999	999	999	0

The All Pair Shortest Path Matrix is:

0	10	3	4
2	0	5	6
9	7	0	1
999	999	999	0

GRAPH :



OUTPUT 2:

Enter the number of vertices
4

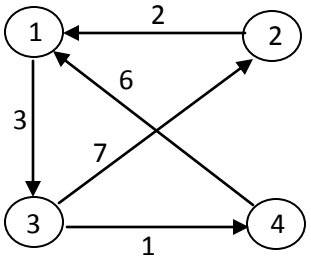
Enter the Cost Matrix (999 for infinity)

0	999	3	999
2	0	999	999
999	7	0	1
6	999	999	0

The All Pair Shortest Path Matrix is:

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

GRAPH :



(b). Travelling Sales Person :

```
import java.util.Scanner;
public class TravSalesPerson
{
	static int MAX = 100;
	static final int infinity = 999;
	public static void main(String args[ ])
	{
		int cost = infinity;
		int c[ ][ ] = new int[MAX][MAX];	// cost matrix
		int tour[ ] = new int[MAX];		// optimal tour
		int n;					// max. cities
		System.out.println("Travelling Salesman Problem using Dynamic
		Programming\n");
		System.out.println("Enter number of cities: ");
		Scanner scanner = new Scanner(System.in);
		n = scanner.nextInt( );
		System.out.println("Enter Cost matrix:\n");
		for (int i = 0; i < n; i++)
			for (int j = 0; j < n; j++)
			 {
				c[i][j] = scanner.nextInt( );
				if (c[i][j] == 0)
					c[i][j] = 999;
			}
		for (int i = 0; i < n; i++)
			tour[i] = i;
		cost = tspdp(c, tour, 0, n);
		// print tour cost and tour
		System.out.println("Minimum Tour Cost: " + cost);
		System.out.println("\nTour:");
		for (int i = 0; i < n; i++)
		{
			System.out.print(tour[i] + " -> ");
		}
		System.out.println(tour[0] + "\n");
		scanner.close( );
	}
	static int tspdp(int c[ ][ ], int tour[ ], int start, int n)
	{
		int i, j, k;
		int temp[ ] = new int[MAX];
		int mintour[ ] = new int[MAX];
		int mincost;
		int cost;
		if (start == n - 2)
```

```
                return c[tour[n - 2]][tour[n - 1]] + c[tour[n - 1]][0];
                mincost = infinity;
                for (i = start + 1; i < n; i++)
                {
                        for (j = 0; j < n; j++)
                                temp[j] = tour[j];
                        temp[start + 1] = tour[i];
                        temp[i] = tour[start + 1];
                        if (c[tour[start]][tour[i]] + (cost = tspdp(c, temp, start + 1, n)) < mincost)
                        {
                                mincost = c[tour[start]][tour[i]] + cost;
                                for (k = 0; k < n; k++)
                                        mintour[k] = temp[k];
                        }
                }
                for (i = 0; i < n; i++)
                        tour[i] = mintour[i];
                return mincost;
        }
}
```

OUTPUT 1:

Enter number of cities:
5

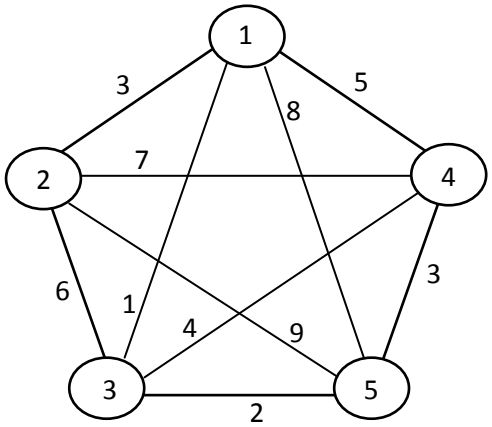
Enter Cost matrix:

0	3	1	5	8
3	0	6	7	9
1	6	0	4	2
5	7	4	0	3
8	9	2	3	0

Minimum Tour Cost: 16

Tour:
0 -> 1 -> 3 -> 4 -> 2 -> 0

GRAPH :



OUTPUT 2:

Enter number of cities:
4

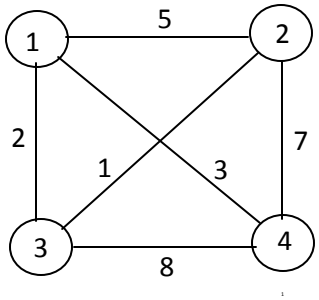
Enter Cost matrix:

0	5	2	3
5	0	1	7
2	1	0	8
3	7	8	0

Minimum Tour Cost: 13

Tour:
0 -> 2 -> 1 -> 3 -> 0

GRAPH :



Program – 11

Design and implement in Java to find a subset of a given set S = {Sl, 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.util.Scanner;
public class SumOfsubset
{
        final static int MAX = 10;
        static int n;
        static int S[ ];
        static int soln[ ];
        static int d;
        public static void main(String args[ ])
        {
                S = new int[MAX];
                soln = new int[MAX];
                int sum = 0;
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter number of elements: ");
                n = scanner.nextInt( );
                System.out.println("Enter the set in increasing order: ");
                for (int i = 1; i <= n; i++)
                        S[i] = scanner.nextInt( );
                System.out.println("Enter the max. subset value(d): ");
                d = scanner.nextInt( );
                for (int i = 1; i <= n; i++)
                        sum = sum + S[i];
                if (sum < d || S[1] > d)
                        System.out.println("No Subset possible");
                else
                        SumofSub(0, 0, sum);
                scanner.close( );
        }
        static void SumofSub(int i, int weight, int total)
        {
                if (promising(i, weight, total) == true)
                        if (weight == d)
                        {
                                for (int j = 1; j <= i; j++)
                                {
                                        if (soln[j] == 1)
                                                System.out.print(S[j] + "  ");
                                }
                                System.out.println( );
```

```
                        }
                        else
                        {
                                soln[i + 1] = 1;
                                SumofSub(i + 1, weight + S[i + 1], total - S[i + 1]);
                                soln[i + 1] = 0;
                                SumofSub(i + 1, weight, total - S[i + 1]);
                        }
        }
        static boolean promising(int i, int weight, int total)
        {
                return ((weight + total >= d) && (weight == d || weight + S[i + 1] <= d));
        }
}
```

OUTPUT 1:

```
Enter number of elements:
4

Enter the set in increasing order:
3 5 6 7

Enter the max. subset value(d):
15

3  5  7
```

OUTPUT 2:

```
Enter number of elements:
5

Enter the set in increasing order:
1 2 5 6 8

Enter the max. subset value(d):
9

1  2  6
1  8
```

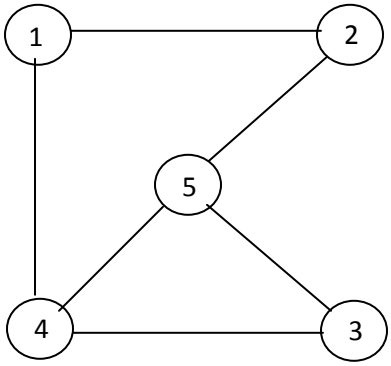
Program – 12

Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of *n* vertices using backtracking principle.

```
import java.util.Scanner;
public class Hamiltonian
{
	boolean found = false;
	int G[ ][ ];
	int x[ ];
	int n;
	public static void main(String args[ ])
	 {
		Hamiltonian hamiltonian = new Hamiltonian( );
		hamiltonian.getData( );
		System.out.println("\nSolution:");
		hamiltonian.HamiltonianMethod(2);
		hamiltonian.printNoSlnPossible( );
	}
	public void printNoSlnPossible( )
	{
		if (found == false)
			System.out.println("No Solution possible!");
	}
	public void getData( )
	{
		Scanner scanner = new Scanner(System.in);
		System.out.println("\t\t\t\tHamiltonian Cycle");
		System.out.print("\nEnter the number of the vertices: ");
		n = scanner.nextInt( );
		G = new int[n + 1][n + 1];
		x = new int[n + 1];
		System.out.print("\nIf edge between the following vertices enter 1 else 0:\n");
		for (int i = 1; i <= n; i++)
			for (int j = 1; j <= n; j++)
			 {
				if ((i != j) && (i < j))
				{
					System.out.print(i + " and " + j + ": ");
					G[j][i] = G[i][j] = scanner.nextInt( );
				}
				if (i == j)
					G[i][j] = 0;
			}
		for (int i = 1; i <= n; i++)
			x[i] = 0;
```

```
            x[1] = 1;
            scanner.close( );
        }
        void HamiltonianMethod(int k)
        {
            while (true)
            {
                NextValue(k, G, x, n);
                if (x[k] == 0)
                    return;
                if (k == n)
                {
                    for (int i = 1; i <= k; i++)
                        System.out.print(x[i] + " ");
                    System.out.println(x[1]);
                    System.out.println( );
                    found = true;
                    return;
                }
                else
                    HamiltonianMethod(k + 1);
            }
        }
        void NextValue(int k, int G[ ][ ], int x[ ], int n)
        {
            while (true)
            {
                x[k] = (x[k] + 1) % (n + 1);
                if (x[k] == 0)
                    return;
                if (G[x[k - 1]][x[k]] != 0)
                {
                    int j;
                    for (j = 1; j < k; j++)
                        if (x[k] == x[j])
                            break;
                    if (j == k)
                        if ((k < n) || ((k == n) && G[x[n]][x[1]] != 0))
                            return;
                }
            }
        }
}
```

OUTPUT 1:
GRAPH :



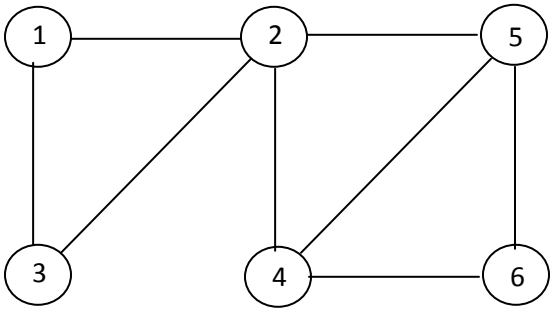
Hamiltonian Cycle
Enter the number of the vertices: 5

If edge between the following vertices enter 1 else 0:
1 and 2: 1
1 and 3: 0
1 and 4: 1
1 and 5: 0
2 and 3: 0
2 and 4: 0
2 and 5: 1
3 and 4: 1
3 and 5: 1
4 and 5: 1

Solution:
1 2 5 3 4 1

1 4 3 5 2 1

OUTPUT 2:
GRAPH :



Hamiltonian Cycle
Enter the number of the vertices: 6

If edge between the following vertices enter 1 else 0:
1 and 2: 1
1 and 3: 1
1 and 4: 0
1 and 5: 0
1 and 6: 0
2 and 3: 1
2 and 4: 1
2 and 5: 1
2 and 6: 0
3 and 4: 0
3 and 5: 0
3 and 6: 0
4 and 5: 1
4 and 6: 1
5 and 6: 1

Solution:
No Solution possible!