DESIGN AND ANALYSIS OF ALGORITHM LABORATORY
[As per Choice Based Credit System (CBCS) scheme]

Subject Code: 15CSL47 **Maximum marks:**20

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

1A) 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 Phoneof these objects with suitable headings.
1B) Write a Java program to implement the Stack using arrays. Write Push(), Pop(), and Display() methods to demonstrate its working.

2 A) Design a superclass 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.
2B) 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 "/".

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.
3B) 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.

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 divideand- 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'salgorithm.** 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** = {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.

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

Course Outcomes: The students should be able to:
□□Design algorithms using appropriate design techniques (brute-force, greedy, dynamic programming, etc.)
□□Implement a variety of algorithms such assorting, graph related, combinatorial, etc., in a high level language.
□□Analyze and compare the performance of algorithms using language features.
□□Apply and implement learned algorithm design techniques and data structuresto solve realworld problems.

Conduction of Practical Examination:
All laboratory experiments (Twelve problems) are to be included for practical examination. Students are allowed to pick one experiment from the lot. To generate the data set use random number generator function. Strictly follow the instructions as printed on the cover page of answer script for breakup of marks
Marks distribution: Procedure + Conduction + Viva: 20 + 50 + 10 (80). Change of experiment is allowed only once and marks allotted to the procedure

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.

1a) 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 Phoneof these objects with suitable headings.

Student.java

```
package lab1;
import java.util.Scanner;
public class Student {
	private String usn;
	private String name;
	private String branch;
	private Long phone;
	public void read()
	{
		Scanner scanner=new Scanner(System.in);
		System.out.println("Enter USN :");
		usn=scanner.nextLine();
		System.out.println("Enter Name :");
		name=scanner.nextLine();
		System.out.println("Enter branch :");
		branch=scanner.nextLine();
		System.out.println("Enter Phone Number :");
		phone=scanner.nextLong();
	}
	public void display()
	{
		System.out.println("USN : "+usn);
		System.out.println("Name : "+name);
		System.out.println("Branch : "+branch);
		System.out.println("Phone Number : "+phone);
		System.out.println("\n");
	}}
```

Main file - StudentTest.java

```
package lab1;
java import.util.Scanner;
public class StudentTest {
	public static void main(String[] args) {
		Scanner scanner=new Scanner(System.in);
		System.out.println("Entet the number of students : ");
		int n=scanner.nextInt();
		Student []s=new Student[n];
		//Creating array of Objects
		for(int i=0;i<n;i++)
			s[i]=new Student();
		System.out.println("Enter the Student Details : ");
		for(int i=0;i<n;i++)
			s[i].read();
		System.out.println("\nThe Student Details are :");
		for(int i=0;i<n;i++)
			s[i].display();
	}
}
```

/*OUTPUT:

```
Entet the number of students :
2
Enter the Student Details :
Enter USN :
111
Enter Name :
Chitra
```

Enter branch :

ISE

Enter Phone Number :

8889997776

Enter USN :

222

Enter Name :

Aparna

Enter branch :

CSE

Enter Phone Number :

7778886665

The Student Details are :

USN : 111

Name : Chitra

Branch : ISE

Phone Number : 8889997776

USN : 222

Name : Aparna

Branch : CSE

Phone Number : 7778886665

*/

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

Stack.java file

```
package lab1;

public class Stack
{
	private Object s[];
	private int top;			// stack top
	public Stack(int n)	// constructor
	{
		s = new Object[n];	// create stack array
		top = -1;				// no items in the stack
	}
	public void push(Object item) // add an item on top of stack
	{
		if(top == s.length-1)
			System.out.println("Stack is full");
		else
			s[++top] = item; // insert an item
	}
	public Object pop() // remove an item from top of stack
	{
		Object item=null;
		if(top==-1)
		{
			System.out.println("Stack is empty");
			return null;
		}
```

```
else
                item = s[top--];  // access top item
        return item;
    }
    public void display()
    {
        if(top==-1)
        {
            System.out.println("Stack is Empty \n");
            return;
        }
        System.out.println("The Content of the Stack are :");
        for(int i=top;i>=0;i--)
            System.out.println(s[i]);
    }
}
```

Main File - StackTest.java

```
package lab1;
import java.util.Scanner;
public class StackTest {
    public static void main(String[] args)
    {
        Stack s1=new Stack(3);
        Object item;
        int ch;
        for(;;)
        {
            System.out.println("\n1.Push\t 2.Pop\t 3.Display\t 4.Quit\n");
```

```
                Scanner scanner=new Scanner(System.in);
                System.out.println("Enter the choice:");
                ch=scanner.nextInt();
                switch(ch)
                {
                        case 1: System.out.println("Enter Item to push on to the stack:");
                        item=scanner.next();
                        s1.push(item);
                        break;
                        case 2: item=s1.pop();
                        if(item!=null)
                                System.out.println("Item Deleted is : "+item);
                        break;

                case 3: s1.display();
                        break;
                default:System.exit(0);
                }
            }
        }

}
```

/*OUTPUT:

```
1.Push  2.Pop   3.Display       4.Quit
Enter the choice:
1
Enter Item to push on to the stack:
11
```

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

1

Enter Item to push on to the stack:

22

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

1

Enter Item to push on to the stack:

33

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

1

Enter Item to push on to the stack:

44

Stack is full

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

3

The Content of the Stack are :

33

22

11

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

2

Item Deleted is : 33

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

2

Item Deleted is : 22

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

2

Item Deleted is : 11

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

2

Stack is empty

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

3

Stack is Empty

1.Push 2.Pop 3.Display 4.Quit

Enter the choice:

4

*/

2a) Design a superclass 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.

Staff.java

```
package lab2;

import java.util.Scanner;

public class Staff {

	protected Integer staffId;

	protected String name;

	protected Long phone;

	protected Double salary;

	public void read()

	{

		Scanner scanner=new Scanner(System.in);

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

		name=scanner.nextLine();

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

		staffId=scanner.nextInt();

		System.out.println("Enter the Phone Number");

		phone=scanner.nextLong();

		System.out.println("Enter the Salary");

		salary=scanner.nextDouble();

	}

	public void display()

	{

		System.out.println("StaffId :"+staffId);

		System.out.println("Name :"+name);

		System.out.println("PhoneNo :"+phone);
```

```
            System.out.println("Salary :"+salary);
        }
}
```

Technical.java

```
package lab2;
import java.util.Scanner;
public class Technical extends Staff
{
        private String skills;
        public void read()
        {
                super.read();
                Scanner scanner=new Scanner(System.in);
                System.out.println("Enter the Skills");
                skills=scanner.nextLine();
        }
        public void display()
        {
                super.display();
                System.out.println("Skills :"+skills);
        }
}
```

Contract.java

```
package lab2;
import java.util.Scanner;
public class Contract extends Staff
{
        private Double period;
```

```
	public void read()
	{
		super.read();
		Scanner scanner=new Scanner(System.in);
		System.out.println("Enter the contract period in years");
		period=scanner.nextDouble();
	}
	public void display()
	{
		super.display();
		System.out.println("Contract Period :"+period);
	}
}
```

Teaching.java

```
package lab2;
import java.util.Scanner;
public class Teaching extends Staff{
	private String domain;
	private Integer publications;

	public void read()
	{
		super.read();
		Scanner scanner=new Scanner(System.in);
		System.out.println("Enter the Domain");
		domain=scanner.nextLine();
		System.out.println("Enter No.of Publications");
		publications=scanner.nextInt();
```

```
    }
public void display()
    {
        super.display();
        System.out.println("Domain :"+domain);
        System.out.println("No.of Publications :"+publications);
    }
}
```

InheritenceTest.java file

Main file

```
package lab2;
public class InheritanceTest {
    public static void main(String[] args)
    {
        Staff teaching=new Teaching();
        teaching.read();
        teaching.display();
        Staff technical=new Technical();
        technical.read();
        technical.display();
        Staff contract=new Contract();
        contract.read();
        contract.display();
    }
}
```

/*OUTPUT

Enter Name

Chitra

Enter StaffId

111

Enter the Phone Number

8887779996

Enter the Salary

40000

Enter the Domain

ImageProcessing

Enter No.of Publications

2

StaffId :111

Name :Chitra

PhoneNo :8887779996

Salary :40000.0

Domain :ImageProcessing

No.of Publications :2

Enter Name

Puneeth

Enter StaffId

222

Enter the Phone Number

5556667778

Enter the Salary

35000

Enter the Skills

JAVA

StaffId :222

Name :Puneeth

PhoneNo :5556667778

Salary :35000.0

Skills :JAVA

Enter Name

Amruth

Enter StaffId

333

Enter the Phone Number

8456765456

Enter the Salary

38000

Enter the contract period in years

2

StaffId :333

Name :Amruth

PhoneNo :8456765456

Salary :38000.0

Contract Period :2.0

*/

2b) 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 "/".

Customer.java

```
package lab2;

import java.util.Scanner;

import java.util.StringTokenizer;

public class Customer

{

	private String name;

	private String date_of_birth;

	public void read()

	{

		Scanner scanner =new Scanner(System.in);

		System.out.println("Enter the Customer Name : ");

		name=scanner.next();

		System.out.println("Enter the Customer DOB in dd/mm/yyyy format : ");

		date_of_birth=scanner.next();

	}

	public void display()

	{

		System.out.println("Customer Name and DOB is :");

		StringTokenizer stk=new StringTokenizer(date_of_birth,"/");

		String dd=stk.nextToken();

		String mm=stk.nextToken();

		String yyyy=stk.nextToken();

		System.out.println(name+","+dd+","+mm+","+yyyy);

	}

}
```

TokenizerTest.java

```
package lab2;
public class TokenizerTest {
	public static void main(String[] args) {
		Customer customer=new Customer();
		customer.read();
		customer.display();
	}
}
```

/*OUTPUT:

```
Enter the Customer Name :
Chaitra
Enter the Customer DOB in dd/mm/yyyy format :
16/02/1986
Customer Name and DOB is :
Chaitra,16,02,1986
*/
```

3a) 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.

ExceptionDemo.java

```
package lab3;

import java.util.Scanner;

public class ExceptionDemo {

	public static void main(String[] args) {

		int a,b;

		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the value of a & b");

		a=sc.nextInt();

		b=sc.nextInt();

		try

		{

			if(b==0)

				throw new Exception("Denominator cannot be Zero,try with other input");

			else

				System.out.println("a/b = "+a/b);

		}

		catch(Exception e)

		{

			System.out.println(e);

		}

	}

}
```

/*OUTPUT:

Run 1 : Enter the value of a & b

25

5

a/b = 5

Run 2: Enter the value of a & b

25

0

java.lang.Exception: Denominator cannot be Zero,try with other input

*/

3b) 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.

RandomThread.java

```
package lab3;

import java.util.Random;

public class RandomThread extends Thread

{

	public void run()

	{

		Random r=new Random();

		try

		{

			for(int i=1;i<=5;i++)

			{

				System.out.println("Random Thread printing : "+r.nextInt(1500));

				Thread.sleep(1000);

			}

		}

		catch (Exception e)

		{

			System.out.println(e.getMessage());

		}

	}

}
```

SquareThread.java

```
package lab3;

public class SquareThread extends Thread

{

	public void run()

	{	try
```

```
		{
			for(int i=1;i<=5;i++)
			{
				System.out.println("Square Thread Printing Square("+i+") : "+i*i);
				Thread.sleep(1000);
			}
		}
		catch (Exception e)
		{
			System.out.println(e.getMessage());
		}
	}
}
```

CubeThread.java

```
package lab3;
public class CubeThread extends Thread
{
	public void run()
	{
		try
		{
			for(int i=1;i<=5;i++)
			{
				System.out.println("Cube Thread Printing Cube("+i+") : "+i*i*i);
				Thread.sleep(1000);
			}
		}
		catch (Exception e)
		{
			System.out.println(e.getMessage());
		}
```

```
	}
}
```

ThreadDemo.java

Main file

```
package lab3;
public class ThreadDemo {
	public static void main(String[] args) {
		RandomThread rt=new RandomThread();
		rt.start();
		SquareThread st=new SquareThread();
		st.start();
		CubeThread ct=new CubeThread();
		ct.start();
	}
}
```

/*OUTPUT:

Random Thread printing : 254

Square Thread Printing Square(1) : 1

Cube Thread Printing Cube(1) : 1

Random Thread printing : 1051

Square Thread Printing Square(2) : 4

Cube Thread Printing Cube(2) : 8

Random Thread printing : 612

Square Thread Printing Square(3) : 9

Cube Thread Printing Cube(3) : 27

Random Thread printing : 1179

Square Thread Printing Square(4) : 16

Cube Thread Printing Cube(4) : 64

Random Thread printing : 1429

Square Thread Printing Square(5) : 25

Cube Thread Printing Cube(5) : 125

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 divideand- conquer method works along with its time complexity analysis: worst case, average case and best case.

QuickSort.java

```
package lab4;

import java.util.Random;

import java.util.Scanner;

public class QuickSort {

    public static int partition(int arr[], int low, int high)

    {

        int i = low, j = high;

        int tmp;

        int pivot = arr[low];

        while (i <= j)

        {

            while (arr[i] < pivot)

                i++;

            while (arr[j] > pivot)

                j--;

            if (i <= j)

            {

                tmp = arr[i];

                arr[i] = arr[j];

                arr[j] = tmp;

                i++;

                j--;

            }

        }

        return i;

    }
```

```
    public static void quickSort(int arr[], int low, int high) {
        int index = partition(arr, low, high);
        if (low < index - 1)
            quickSort(arr, low, index - 1);
        if (index < high)
            quickSort(arr, index, high);
    }
    public static void main(String[] args)
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter No.of elements in the array,say n>5000");
            int n=sc.nextInt();
            int []arr=new int[n];
            Random r=new Random();
            for(int i=0;i<n;i++)
            {
                    arr[i]=r.nextInt(10000);
                    //System.out.println(arr[i]);
            }
            long start=System.currentTimeMillis();
            quickSort(arr,0,n-1);
            long end=System.currentTimeMillis();
            System.out.println("Time taken is :"+(end-start)+" ms");
            /*for(int i=0;i<n;i++)
            System.out.println(arr[i]);*/
    }
}
```

/*OUTPUT:

Enter No.of elements in the array,say n>5000

200000

Time taken is :33 ms

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.

MergeSort.java

```
package lab5;

import java.util.Random;

import java.util.Scanner;

public class MergeSort

{

        static int a[];

        static int b[];

        public static void merge(int a[], int low, int mid, int high)

        {

           int i, j, k;

           i=low; j=mid+1; k=low;

           while ( i<=mid && j<=high )

           {

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

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

                 else

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

           }

           while (i<=mid)   b[k++] = a[i++] ;

           while (j<=high)  b[k++] = a[j++] ;

           for(k=low; k<=high; k++)

             a[k] = b[k];

        }


        public static void mergeSort(int a[], int low, int high)

        {
```

```
    int mid;
    if(low >= high)
      return;
    mid = (low+high)/2 ;
    mergeSort(a, low, mid);
    mergeSort(a, mid+1, high);
    merge(a, low, mid, high);
  }
  public static void main(String[] args)
  {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter No.of elements in the array,say n>5000");
        int n=sc.nextInt();
        a=new int[n];
        b=new int[n];
        Random r=new Random();
        for(int i=0;i<n;i++)
        {
              a[i]=r.nextInt(100000);
              //System.out.println(a[i]);
        }
        long start=System.currentTimeMillis();
        mergeSort(a,0,n-1);
        long end=System.currentTimeMillis();
        System.out.println("Time taken is :"+(end-start)+ " ms");
        //for(int i=0;i<n;i++)
        //System.out.println(a[i]);
  }
}
```

/*OUTPUT:

Enter No. of elements in the array,say n>5000

500000

Time taken is :106 ms

*/

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

6a) Dynamic.java

```
package lab6;

import java.util.Scanner;

public class Dynamic

{

	private int val[][],wt[],p[],x[];

	private int n,m;

	public void readData()

	{

		Scanner sc = new Scanner(System.in);

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

		 n = sc.nextInt();

		System.out.println("Enter the capacity");

		m=sc.nextInt();

		val=new int[n+1][m+1];

		wt=new int[n+1];

		p=new int[n+1];

		x=new int[n+1];

		System.out.println("Enter the N weights : ");

		for(int i=1;i<=n;i++)

		wt[i]=sc.nextInt();

		System.out.println("Enter the N profits");

		for(int j=1;j<=n;j++)

		p[j]=sc.nextInt();;

	}
```

```
public void findProfit()
	{
		for(int i=0;i<=n;i++)
		for(int j=0;j<=m;j++)
		{
			if(i==0||j==0)
			val[i][j]=0;
			else if(wt[i]>j)
			val[i][j]=val[i-1][j];
			else
			val[i][j]=max(val[i-1][j],val[i-1][j-wt[i]]+p[i]);
		}
	}
	public void printData()
	{
		System.out.println("Maximum profit is:"+val[n-1][m-1]);
		for(int k=1;k<=n;k++)
		x[k]=0;
		int i=n;
		int j=m;
		while(i!=0 && j!=0)
		{
			if(val[i][j]!=val[i-1][j])
			{
				x[i]=1;
				j=j-wt[i];
			}
			i=i-1;
		}
		System.out.println("The selected objects are : ");
		for(int k=1;k<=n;k++)
```

```
                    if(x[k]==1)
                    System.out.print(k+"  ");
        }
    int max(int a, int b)
    {
       return (a > b)? a : b;
    }
/*    static int knapSack(int W, int wt[], int val[], int n)
    {
       int i, w;
       int [][]K = new int[n+1][W+1];
            // Build table K[][] in bottom up manner
       for (i = 0; i <= n; i++)
       {
          for (w = 0; w <= W; w++)
          {
             if (i==0 || w==0)
                K[i][w] = 0;
             else if (wt[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
             else
                K[i][w] = K[i-1][w];
          }
       }

       return K[n][W];
    }
    */
    public static void main(String[] args)
    {
        Dynamic d=new Dynamic();
```

```
        d.readData();
        d.findProfit();
        d.printData();
        }
}
```

/*OUTPUT:

Enter the number of items:

4

Enter the capacity

5

Enter the N weights :

1 2 3 4

Enter the N profits

30 20 50 30

Maximum profit is:80

The selected objects are :

1 3 */

//check this link for better understanding https://www.youtube.com/watch?v=ZABYAZsEJ9U

6b) Greedy.java

```
package lab6;
import java.util.Scanner;
public class GreedyKnapsack
{
        private double[] p;
        private double[] wt;
        private double[] take;
        private int n,m;
        public void readData()
        {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter the number of items: ");
                 n = sc.nextInt();
                System.out.println("Enter the capacity");
                m=sc.nextInt();
                wt=new double[n];
                p=new double[n];
                take=new double[n];
                System.out.println("Enter the N weights : ");
                for(int i=0;i<n;i++)
                wt[i]=sc.nextInt();
                System.out.println("Enter the N profits");
                for(int j=0;j<n;j++)
                p[j]=sc.nextInt();;
        }
 public void unitPriceOrder()
   {
      for (int i = 0; i < p.length; i++)
      {
         for (int j = 1; j < (p.length - i); j++)
```

```
        {
            double x=p[j - 1] / wt[j - 1];
            double y=p[j] / wt[j];
            if (x <=y)
            {
                double temp = p[j - 1];
                p[j - 1] = p[j];
                p[j] = temp;
                double temp1 = wt[j - 1];
                wt[j - 1] = wt[j];
                wt[j] = temp1;
            }
        }
    }
}
public void Knapsack()
{
    int j;
    for (j = 0; j < p.length; j++)
    {
        take[j] = 0;
    }
    double total = m;
    for (j = 0; j < p.length; j++)
    {
        if (wt[j] <= total)
        {
            take[j] = 1.00;
            total = total - wt[j];
        }
        else
```

```
        {
            break;// to exit the for-loop
        }
    }
    if (j < p.length)
    {
        take[j] = (double)(total / wt[j]);
    }
}
public void print()
{
    System.out.println("item    profit    weight   Unit Price  Take weight");
    for (int i = 0; i < p.length; i++)
    {
        if(take[i]>0.0)
        {
                System.out.printf("%d %10.1f %10.1f %10.1f %10.1f",i,p[i],wt[i],(p[i]/wt[i]),take[i]);
                System.out.println();
        }
    }
}

public static void main(String args[])
{
    GreedyKnapsack g = new GreedyKnapsack();
    g.readData();
    g.unitPriceOrder();
    g.Knapsack();
    System.out.println("Optimal soluation to knapsack is :");
    System.out.println("=======+===========+=======+===========+==========");
    g.print();
```

```
    }
}
```

/*OUTPUT:

```
 * Enter the number of items:
4
Enter the capacity
5
Enter the N weights :
1 2 3 4
Enter the N profits
30 20 50 30
Optimal solution to knapsack is :
=======+============+=======+============+========
item    profit    weight   Unit Price  Take weight
0       30.0       1.0       30.0        1.0
1       50.0       3.0       16.7        1.0
2       20.0       2.0       10.0        0.5

 */
```

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

Dijkstra.java

```
package lab7;

import java.util.Scanner;

public class Dijkstra

{

	private final int NOEDGE=999;

	private int n,v,cost[][],s[],d[];

	public void cosmatrix()

	{

		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the Number of Vertices : ");

		n=sc.nextInt();

		cost=new int[n+1][n+1];

		s=new int[n+1];

		d=new int[n+1];

		System.out.println("Enter the Cost Adjacency Matrix\n (Enter 999 if there is No edge)");

		for (int i=1;i<=n;i++)

			for (int j=1;j<=n;j++)

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

	}

	public  void print()

	{

		System.out.println("The Shortest Path with the cost is");

		for (int i=1;i<=n;i++)

			if (i!=v)

				System.out.println("<"+v+">--<"+i+"> -->"+d[i]);

	}

	public  void sspath()

	{
```

```
        int i,u,w;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Source Vertex");
        v=sc.nextInt();
        for(i=1;i<=n;i++)
        {
                s[i] =0;
                d[i] = cost[v][i];
        }
        s[v] =1;
        d[v] = 0;
        i = 2;
        while (i<=n)
        {
                u = choose();
                s[u] = 1;
                i++;
                for (w=1;w<=n;w++)
                {
                        if ((d[u]+cost[u][w] < d[w] ) && s[w]==0)
                                d[w] = d[u] + cost[u][w];
                }
        }
}

public  int choose()
{
        int w=0,j,min;
        min = NOEDGE;
        for(j=1;j<=n;j++)
        {
```

```
                if (d[j]<min && s[j]==0)
                {
                        min = d[j];
                        w = j;
                }
        }
        return w;
    }
    public static void main(String[] args)
    {
        Dijkstra d=new Dijkstra();
        d.cosmatrix();
        d.sspath();
        d.print();
    }
}
```

/*OUTPUT:

Enter the Number of Vertices :

4

Enter the Cost Adjacency Matrix

(Enter 999 if there is No edge)

0 1 4 999

1 0 2 5

4 2 0 1

999 5 1 0

Enter the Source Vertex

1

The Shortest Path with the cost is

<1>--<2> -->1

<1>--<3> -->3

<1>--<4> -->4

*/

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

Kruskal.java

```
package lab8;

import java.util.Scanner;

public class Kruskal

{

        private int c[][]; //The edges along with cost is stored

        public int n;   //Number of nodes in the graph

        public int m;   //Number of edges in the graph

        public void cost_adjacency_matrix()

        {

                Scanner sc=new Scanner(System.in);

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

                n=sc.nextInt();

                c=new int[n][n];

                System.out.println("Enter the adjacency matrix (Enter 999 if there is No edge)");

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

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

                        c[i][j]=sc.nextInt();

        }

        //function to find root of a given vertex

        int find(int v,int p[])

        {

           while(p[v]!=v)

                v=p[v];

           return v;

        }
```

```
//function to merge two trees into a single tree
        void union_ij(int i,int j,int p[])
        {
           if(i<j)
              p[j]=i;
           else
              p[i]=j;
        }
        public void minimum_spanning_tree()
        {
           int count,i,min,j,u=0,v=0,k,sum;
           int p[]=new int[n];
           int t[][]=new int[n][2];
           count=0;              //initialize the number of edges selected to zero
           k=0;                  //points to the first selected edge of MST
           sum=0;                //initial cost of minimum spanning tree
           for(i=0;i<n;i++) p[i]=i;  //create forests with n vertices
           //select n-1 edges for minimum spanning tree
           while(count<n)
           {
              min=999;
              for(i=0;i<n;i++)
              {
                 for(j=0;j<n;j++)
                 {
                    if(c[i][j]!=0&&c[i][j]<min)
                    {
                       min=c[i][j];
                       u=i;
                       v=j;
                    }
```

```
            }
        }
        if(min==999) break;   //no more root for the vertex u
        i=find(u,p);          //find the root for the vertex u
        j=find(v,p);          //find the root for the vertex v
        if(i!=j)              //if the roots of vertex u and v are different
        {                     //select the edge (u,v) as the edge of MST
            t[k][0]=u;
            t[k][1]=v;
            k++;
            count++;          //update the number of edges selected for MST
            sum+=min;         //update the cost of the MST
            union_ij(i,j,p);  //merge the two trees with the roots i and j
        }
        c[u][v]=c[v][u]=999;  //delete the edge (u,v) by storing a big value
    }
    if(count==n-1)
    {
        System.out.println("cost of spanning tree="+sum);
        System.out.println("Spanning Tree is shown below");
        for(k=0;k<n-1;k++)
        System.out.println(t[k][0]+","+t[k][1]);
        return;
    }
    System.out.println("Spanning tree do not exist");
}
public static void main(String[] args)
{
            Kruskal k=new Kruskal();
             k.cost_adjacency_matrix();
            k.minimum_spanning_tree();
```

```
        }
}
```

/*OUTPUT

```
Enter the number of nodes
4
Enter the adjacency matrix
0 1 5 2
1 0 999 999
5 999 0 3
2 999 3 999
cost of spanning tree=6
Spanning Tree is shown below
0,1
0,3
2,3
 */
```

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

Prims.java

```
package lab8;
import java.util.Scanner;
public class Prims
{
   private int n;                  //number of nodes in the graph
   private int a[][];              //cost adjacency matrix
   public void read_data()
   {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter the number of nodes");
            n=sc.nextInt();
            a=new int[n][n];
            System.out.println("Enter the cost adjacency matrix (Enter 999 if there is No edge)");
            for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                  a[i][j]=sc.nextInt();
    }
   public void minimum_spanning_tree()
   {
            int u,v,min;
            int sum;                              //holds the cost of the minimum spanning tree
            int k;                        //index for storing the edges w.r.t minimum spanning tree
            int t[][]=new int[n][2];    //holds the minimum spanning tree
            int p[]=new int[n];           //holds the vertices selected
            int d[]=new int[n];           //holds the weights for the selected edges
            int s[]=new int[n];           //has the information of what nodes are visited
                                                        //and what nodes are not visited
```

```
int source;                    //contains a vertex from which least edge starts
                                        //find the source vertex from which least edge starts
        min=9999;
        source=0;
        for(int i=0;i<n;i++)
        {
           for(int j=0;j<n;j++)
           {
              if(a[i][j]!=0 && a[i][j]<=min)
              {
                 min=a[i][j];
                 source=i;
              }
           }
        }
        //initialization
        for(int i=0;i<n;i++)
        {
           d[i]=a[source][i];
           s[i]=0;
           p[i]=source;
        }
        s[source]=1;        //add source to S
        sum=0;              //initial cost of minimum spanning tree
        k=0;              //used as index to store the edges selected
        for(int i=1;i<n;i++)
        {
           //find u and d[u] such that d[u] is minimum and u in V-S
           min=9999;
           u=-1;
           for(int j=0;j<n;j++)
```

```
    {
      if(s[j]==0)
      {
        if(d[j]<=min)
        {
          min=d[j];
          u=j;
        }
      }
    }
    //select an edge with least cost
    t[k][0]=u;
    t[k][1]=p[u];
    k++;
    //add the cost associated with edge to get total cost of MSP
    sum+=a[u][p[u]];
    //add u to s
    s[u]=1;
    //find d[v] and p[v] for every v in V-S
    for(v=0;v<n;v++)
    {
      if(s[v]==0 && a[u][v]<d[v])
      {
        d[v]=a[u][v];
        p[v]=u;
      }
    }
  }
  //check for the spanning tree
  if(sum>=9999)
    System.out.println("Spanning tree does not exist");
```

```
        else
        {
            System.out.println("Spanning tree exists and minimum spanning tree is");
          for(int i=0;i<n-1;i++)
          {
            System.out.println(t[i][0]+" "+t[i][1]);
          }
          System.out.println("The cost the spanning tree = "+sum);
        }
    }
    public static void main(String[] args)
    {
            Prims p=new Prims();
             p.read_data();
            p.minimum_spanning_tree();
    }
}
```

/*OUTPUT:

Enter the number of nodes

4

Enter the cost adjacency matrix

0 1 5 2

1 0 999 999

5 999 0 3

2 999 3 0

Spanning tree exists and minimum spanning tree is

0 1

3 0

2 3

The cost the spanning tree = 6

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) Floyds.java

```
package lab9;

import java.util.Scanner;

public class Floyds

{

	private int c[][]; // The edges along with cost is stored

	private int d[][];

	public int n; // Number of nodes in the graph

	public int m; // Number of edges in the graph

	public void cost_adjacency_matrix()

	{

		Scanner sc = new Scanner(System.in);

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

		n = sc.nextInt();

		c = new int[n][n];

		d = new int[n][n];

		System.out.println("Enter the adjacency matrix (Enter 999 if there is No edge)");

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

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

				c[i][j] = sc.nextInt();

	}

	public int min(int a, int b)

	{

		return a < b ? a : b;

	}


	public void floyd()

	{

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

```
                for (int j = 0; j < n; j++)
                        d[i][j] = c[i][j];
        for (int k = 0; k < n; k++)
                for (int i = 0; i < n; i++)
                        for (int j = 0; j < n; j++)
                                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
    }
    public void print()
    {
        System.out.println("The all pair shortest matrix is shown below");
        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {
                        System.out.printf("%5d", d[i][j]);
                }
                System.out.println();
        }
    }
    public static void main(String[] args)
    {
        Floyds f=new Floyds();
        f.cost_adjacency_matrix();
        f.floyd();
        f.print();
    }
}
```

/*OUTPUT:

Enter the number of nodes

5

Enter the adjacency matrix (Enter 999 if there is No edge)

0 1 999 5 999

1 0 15 3 1

999 15 0 999 3

5 3 999 0 6

999 1 3 6 0

The all pair shortest matrix is shown below

0 1 5 4 2

1 0 4 3 1

5 4 0 7 3

4 3 7 0 4

2 1 3 4 0

*/

10b)Traveling Salesmen Problem

```
import java.util.*;
import java.text.*;
class TSP
{
	int weight[][],n,tour[],finalCost;
	final int INF=999;
public TSP()  {
	Scanner s=new Scanner(System.in);
	System.out.println("Enter no. of nodes:=>");
	n = s.nextInt();
	weight=new int[n][n];
	tour=new int[n-1];
for(int i=0;i<n;i++)
{
	for(int j=0;j<n;j++)
	{
		if(i!=j){
			System.out.print("Enter weight of "+(i+1)+" to "+(j+1)+":=>");
			weight[i][j]=s.nextInt();
}
}
}
System.out.println();
System.out.println("Starting node assumed to be node 1.");
eval();
}
public int COST(int currentNode,int inputSet[],int setSize)
{
if(setSize==0)
```

```
return weight[currentNode][0];
int min=INF,minindex=0;
        int setToBePassedOnToNextCallOfCOST[] = new int[n-1];
        for(int i=0;i<setSize;i++)
        {
        int k=0;//initialise new set
                for(int j=0;j<setSize;j++)
                {
                if(inputSet[i]!=inputSet[j])
                setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];
}
int temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);
if((weight[currentNode][inputSet[i]]+temp) < min)
{
min=weight[currentNode][inputSet[i]]+temp;
minindex=inputSet[i];
}
}
return min;
}
public int MIN(int currentNode,int inputSet[],int setSize)
{
if(setSize==0)
return weight[currentNode][0];
int min=INF,minindex=0;
        int setToBePassedOnToNextCallOfCOST[]=new int[n-1];
        for(int i=0;i<setSize;i++){ //considers each node of inputSet
        int k=0;
                for(int j=0;j<setSize;j++){
                if(inputSet[i]!=inputSet[j])                                    }
                setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];
```

```
int temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);
if((weight[currentNode][inputSet[i]]+temp) < min)
{
min=weight[currentNode][inputSet[i]]+temp;
minindex=inputSet[i];
}
}
return minindex;
}
        public void eval()
{
        int dummySet[]=new int[n-1];
        for(int i=1;i<n;i++)
        dummySet[i-1]=i;
        finalCost = COST(0,dummySet,n-1);
        constructTour();
}


public void constructTour()
{
 int previousSet[]=new int[n-1];
int nextSet[]=new int[n-2];
for(int i=1;i<n;i++)
previousSet[i-1]=i;
int setSize=n-1;
tour[0]=MIN(0,previousSet,setSize);
for(int i=1;i<n-1;i++){
int k=0;
for(int j=0;j<setSize;j++){
```

```
        if(tour[i-1]!=previousSet[j])
        nextSet[k++]=previousSet[j];
}

--setSize;
tour[i]=MIN(tour[i-1],nextSet,setSize);
for(int j=0;j<setSize;j++)
        previousSet[j]=nextSet[j];
}
display();
}
```

```
public void display()
{
        System.out.println();
        System.out.print("The tour is 1-");
        for(int i=0;i<n-1;i++)
        System.out.print((tour[i]+1)+"-");
        System.out.print("1");
        System.out.println();
        System.out.println("The final cost is "+finalCost);
        }
        }

class TSPExp{
        public static void main(String args[]){
        TSP obj=new TSP();
}
}
```

OUTPUT:

```
Enter no. of nodes:=>
5
Enter weight of 1 to 2:=>3
Enter weight of 1 to 3:=>9
Enter weight of 1 to 4:=>5
Enter weight of 1 to 5:=>999
Enter weight of 2 to 1:=>3
Enter weight of 2 to 3:=>2
Enter weight of 2 to 4:=>6
Enter weight of 2 to 5:=>999
Enter weight of 3 to 1:=>9
Enter weight of 3 to 2:=>2
Enter weight of 3 to 4:=>4
Enter weight of 3 to 5:=>6
Enter weight of 4 to 1:=>5
Enter weight of 4 to 2:=>6
Enter weight of 4 to 3:=>4
Enter weight of 4 to 5:=>1
Enter weight of 5 to 1:=>999
Enter weight of 5 to 2:=>999
Enter weight of 5 to 3:=>6
Enter weight of 5 to 4:=>1
Starting node assumed to be node 1.
The tour is 1-2-3-5-4-1
The final cost is 17
```

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.

Subset.java

```
package lab10;

import java.util.Scanner;

public class Subset

{

        private int s[]=new int[10],x[]=new int[10],d,n;

        public void read()

        {

                Scanner sc = new Scanner(System.in);

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

                n = sc.nextInt();

                System.out.println("Enter the set in increasing order");

                for(int i=1;i<=n;i++)

                        s[i]=sc.nextInt();

                System.out.println("Enter the subset sum:");

                d=sc.nextInt();

        }

        public void check()

        {

                int sum=0,i;

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

                        sum+=s[i];

                if(sum<d || s[i]>d)

                        System.out.println("No subset possible");
```

```
        else
        {
                System.out.println("Solutions are");
                sumofsub(0,1,sum);
        }
}
void sumofsub(int m,int k,int r)
{
        x[k]=1;//Create left child(with kth element)
        if((m+s[k])==d)
        {
                for(int i=1;i<=k;i++)
                        if(x[i]==1)
                                System.out.print("  "+s[i]+"  ");
                        System.out.println();;
        }
        else
                if(m+s[k]+s[k+1]<=d)
                        sumofsub(m+s[k],k+1,r-s[k]);
                if((m+r-s[k]>=d) && (m+s[k+1]<=d))
                {
                        x[k]=0;
                        sumofsub(m,k+1,r-s[k]);
                }
}
public static void main(String[] args)
{
```

```
            Subset s1=new Subset();
            s1.read();
            s1.check();
        }
}
```

/*OUTPUT:

```
Enter the number of elements:
5
Enter the set in increasing order
1 2 5 6 8
Enter the subset sum:
9
Solutions are
 1   2   6
 1   8
 */
```

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;
import java.util.Arrays;                                    /** Class HamiltonianCycle **/
public class HamiltonianCycle
{
        private int V, pathCount;
        private int[] path;
        private int[][] graph;
                public void findHamiltonianCycle(int[][] g)                 { /** Function to find
        cycle **/
                V = g.length;
                path = new int[V];
                Arrays.fill(path, -1);
                graph = g;
                try    {
                path[0] = 0;
                pathCount = 1;
                solve(0);
        System.out.println("No solution");
}
catch (Exception e){
        System.out.println(e.getMessage());
display();
}
}
/** function to find paths recursively **/
public void solve(int vertex)  throws  Exception {                  /** solution **/
```

```
                if (graph[vertex][0] == 1 && pathCount == V)
        throw new Exception("Solution found");
if (pathCount == V)             /*all vertices selected but last vertex not linked to 0 */
return;
        for (int v = 0; v < V; v++)     {
if (graph[vertex][v] == 1 )     {                       /** if connected **/
        path[pathCount++] = v;                          /** add to path **/
        graph[vertex][v] = 0;                           /** remove connection **/
        graph[v][vertex] = 0;
if (!isPresent(v))                              /*if vertex not already selected solve recursively*/
        solve(v);
        graph[vertex][v] = 1;                           /** restore connection **/
        graph[v][vertex] = 1;
        path[--pathCount] = -1;                         /** remove path **/
}
}
}
public boolean isPresent(int v)
{                       /*function to check if path is already selected*/
        for (int i = 0; i < pathCount - 1; i++)
        if (path[i] == v)
return true;
return false;
}

public void display()
{                                                       /** display solution **/
```

```
        System.out.print("\nPath : ");
        for (int i = 0; i <= V; i++)
System.out.print((path[i % V]+1) +" ");
System.out.println();
}
public static void main (String[] args)
{                       /** Main function **/
        Scanner scan = new Scanner(System.in);
        System.out.println("Hamiltonian Cycle for an Undirected Graph\n");
        HamiltonianCycle hc = new HamiltonianCycle(); /** Make an object **/
        System.out.println("Enter number of vertices\n");/*Accept number of vertices*/
        int V = scan.nextInt();
        System.out.println("\nEnter matrix\n"); /** get graph **/
        int[][] graph = new int[V][V];
        for (int i = 0; i < V; i++)
                for (int j = 0; j < V; j++)
                graph[i][j] = scan.nextInt();
                hc.findHamiltonianCycle(graph);
        scan.close();
}
}
```

Output:

Hamiltonian Cycle for an Undirected Graph

Enter number of vertices

8

Enter matrix

0 1 0 0 1 1 0 0

1 0 1 0 0 0 0 1

0 1 0 1 1 0 0 0

0 0 1 0 1 0 0 0

1 0 1 1 0 0 0 0

1 0 0 0 0 0 1 1

0 0 0 0 0 1 0 1

0 1 0 0 0 1 1 0

Solution found

Path : 1 5 4 3 2 8 7 6 1