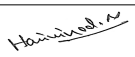
COURSE LABORATORY MANUAL

A. LABORATORY OVERVIEW

Degree:	BE	Programme:	CSE
Semester:	4	Academic Year:	2019-20
Laboratory Title:	**Design and Analysis of Algorithms Laboratory**	Laboratory Code:	**18CSL47**
L-T-P-S:	0-2-2-0	Duration of SEE:	3 Hrs
Total Contact Hours:	36 Hrs	SEE Marks:	60
Credits:	2	CIE Marks:	40
Lab Manual Author:	Mr. Nithin Kurup U G	Sign 	Dt :10/12/2019
Checked By:	Mr. Harivinod N	Sign 	Dt : 10/12/2019

B. DESCRIPTION

1. PREREQUISITES:

- Computer Programming Laboratory (18CPL16/26)
- Data Structures and Applications (18CS32)
- Creative thinking, sound mathematical insight and programming skills.

2. BASE COURSE:

- 18CS42 – Design and Analysis of algorithms

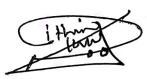
3. COURSE OUTCOMES:

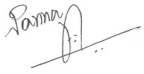
At the end of the course, the student will be able to;

1. Design algorithms using appropriate design techniques
2. Implement a variety of algorithms such as sorting, graph related problems, combinatorial, etc., in a high level language.
3. Analyze and compare the performance of algorithms using language features.
4. Apply and implement learned algorithm design techniques and data structures to solve real world problems.

4. RESOURSES REQUIRED:

- Hardware resources:
  - Personal Computer
  - Windows / Linux operating system
- Software resources:
  - Java 2 JDK
  - Netbeans IDE 8.2





Prepared by: Nithin Kurup U G Checked by: Harivinod N HOD

COURSE LABORATORY MANUAL

5. RELEVANCE OF THE COURSE:

- Data Communication (18CS46)

6. GENERAL INSTRUCTIONS:

- To execute the programs follow the steps given below.
  - Create Java program with .java extension in Netbeans IDE
  - The file name shuld be same as that of class name
  - Build and Run the program and analyze the output

7. CONTENTS:

Expt No.	Title of the Experiments	RBT	CO
1.a.	Create a Java class called **Student** with the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone Write a Java program to create n Student objects and print the USN, Name, Branch and Phone of these objects with suitable headings.	L3	CO 1,2,4
1.b.	Write a Java program to implement the **Stack** using arrays. Write Push(), Pop() and Display() methods to demonstrate its working.	L2	CO 1,2,4
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.	L3	CO 1,2,4
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 "/".	L3	CO 1,2,4
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.	L3	CO 1,2,4
3.b.	Write a Java program that implements a **Multi-thread** application that has thtree 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.	L3	CO 1,2,4
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.	L3	CO 1,2,3
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.	L3	CO 1,2,3

	Demonstrate using Java how the divideand- conquer method works along with its time complexity analysis: worst case, average case and best case.		
6.a.	Implement in Java, the **0/1 Knapsack** problem using **Dynamic Programming** method	L3	CO 1,2,4
6.b.	Implement in Java, the **0/1 Knapsack** problem using **Greedy** method.	L3	CO 1,2,4
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.	L3	CO 1,2,4
8	Find Minimum Cost Spanning Tree of a given undirected graph using **Kruskal's** algorithm. Use Union-Find algorithms in your program.	L3	CO 1,2,4
9	Find Minimum Cost Spanning Tree of a given undirected graph using **Prim's** algorithm. Implement the program in Java language.	L3	CO 1,2,4
10.a.	Write Java programs to Implement All-Pairs Shortest Paths problem using **Floyd's** algorithm.	L3	CO 1,2,4
10.b.	Write Java programs to implement **Travelling Sales Person** problem using Dynamic programming.	L3	CO 1,2,4
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.	L3	CO 1,2,4
12	Design and implement the presence of **Hamiltonian Cycle** in an undirected Graph G of n vertices using backtracking principle.	L3	CO 1,2,4
13	Open ended experiment - 1	L3	CO 1,2
14	Open ended experiment - 2	L3	CO 1,2

8. REFERENCE:

1. Anany Levitin: Introduction to The Design & Analysis of Algorithms, 2nd Edition, Pearson Education.
2. Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran: Fundamentals of Computer Algorithms, 2nd Edition, Universities Press.
3. Thomas H. Cormen, Charles E. Leiserson, Ronal L. Rivest, Clifford Stein: Introduction to Algorithms, 3rd Edition, PHI.
4. R.C.T. Lee, S.S. Tseng, R.C. Chang & Y.T.Tsai: Introduction to the Design and Analysis of Algorithms A Strategic Approach, Tata McGraw Hill.

-

C. EVALUATION SCHEME

For CBCS 2018 scheme:

1. Laboratory Components : 30 Marks
   (Record writing, Laboratory performance and Viva-voce)
2. Laboratory IA tests: 10 Marks
   (Minimum 2 IAs are mandatory. For the final IA test marks, average of the 2 IA test marks shall be considered and converted to maximum of 10)

3. Continuous Internal Evaluation (CIE) = 30 + 10 = 40 Marks
4. *SEE* : 60* Marks
   (*The SEE will be conducted for 100 marks and proportionally reduced to 60 marks)

-

D1. ARTICULATION MATRIX

Mapping of CO to PO												
	POs											
COs	1	2	3	4	5	6	7	8	9	10	11	12
1. **Design algorithms** using appropriate design techniques	3	3	3	2	-	1	-	2	3	3	1	2
2. **Implement** a variety of algorithms such assorting, graph related, combinatorial, etc., in a high level language.	3	3	3	2	3	-	-	1	2	2	2	2
3. Analyze and compare the **performance** of algorithms using language features.	3	3	3	2	1	1	2	-	3	3	1	2
4. Apply and implement learned algorithm design techniques and data structures to solve **real-world problems**.	3	3	3	3	2	3	1	2	3	3	3	3

Note: Mappings in the Tables D1 (above) and D2 (below) are done by entering in the corresponding cell the Correllation Levels in terms of numbers. For Slight (Low): 1, Moderate (Medium): 2, Substantial (High): 3 and for no correllation: " - ".

D2. ARTICULATION MATRIX CO v/s PSO

Mapping of CO to PSO			
	PSOs		
COs	1	2	3
1. Design algorithms using appropriate design techniques	3	1	1
2. Implement a variety of algorithms such assorting, graph related, combinatorial, etc., in a high level language.	3	1	3
3. Analyze and compare the performance of algorithms using language features.	2	1	1
4. Apply and implement learned algorithm design techniques and data structures to solve real-world problems.	3	2	3

-

E. EXPERIMENTS

1. EXPERIMENT NO: **1.a.**

2. TITLE: **ARRAY OF OBJECTS IN JAVA**

3. LEARNING OBJECTIVES:

- To implement class, objects and array of objects in Java

4. AIM:

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

5. THEORY / HYPOTHESIS:

A class is an expanded concept of a data structure. The object of same data structure and behaviour are grouped in to class. Instead of holding only data, it can hold both data members and member functions. Once the class has been defined we can create an array of objects. An array of objects is created just like an array of primitive type data items. For ex if following is the definition of the class.

```
class Student{
        int marks;
      }
```

then the following statement

```
Student[] studentArray = new Student[7];
```

The above statement creates the array which can hold references to seven Student objects. The studentArray contains seven memory spaces in which the address of seven Student objects may be stored.

The Student objects have to be instantiated using the constructor of the Student class and their references should be assigned to the array elements. Using a for loop we can initialize elements of an array.

6. PROCEDURE / PROGRAMME / ACTIVITY:

StudentInfo.java

```
import java.util.Scanner;

class Student {
   private String usn;
   private String name;
   private String branch;
   private String phone;

   public void read() {

      Scanner sc = new Scanner(System.in);
      usn = sc.nextLine();
      name = sc.nextLine();
      branch = sc.nextLine();
```

```
        phone = sc.nextLine();
    }

    public void display() {
        System.out.println(usn + "\t" + name + "\t" + branch + "\t" + phone);
    }
}

class StudentInfo {

    public static void main(String[] args) {

        // Read the number of students
        System.out.println("Enter the total number of students");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        // Create n objects
        Student[] st = new Student[n];
        for (int i = 0; i < st.length; i++) {
            st[i] = new Student();
        }

        // Read the student information
        for (int i = 0; i < n; i++) {
            System.out.println("Enter USN, Name, Branch & Phone no. for student " + (i + 1));
            st[i].read();
        }

        // Display the student information
        System.out.println("USN\tName\tBranch\tPhone");
        for (int i = 0; i < n; i++) {
            st[i].display();
        }
    }
}
```

7. RESULTS & CONCLUSIONS:

```
Enter the total number of students
3

Enter USN, Name, Branch & Phone no. for student 1
4vp14cs001
Karthik
cse
9876786548

Enter USN, Name, Branch & Phone no. for student 2
4vp14cs002
```

Apoorva
cse
9767654345

Enter USN, Name, Branch & Phone no. for student 3
4vp14cs034
Akshatha
cse
8797654531

USN	Name	Branch	Phone
4vp14cs001	Karthik	cse	9876786548
4vp14cs002	Apoorva	cse	9767654345
4vp14cs034	Akshatha	cse	8797654531

8. LEARNING OUTCOMES:

- Students are able to understand and implement object oriented concepts in java such as class,object, array of objects.

9. APPLICATION AREAS:

- All the software systems where object oriented programming is used.

10. REMARKS:

-

1. EXPERIMENT NO: **1.b.**

2. TITLE: **STACK IN JAVA**

3. LEARNING OBJECTIVES:

- To understand the working of stack and its implemetation in Java

4. AIM:

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

5.THEORY / HYPOTHESIS:

Stack : A stack is a linear data structure consisting a list of elements in which elements can be inserted or deleted at one end which is known as TOP of the stack. Stacks follows LIFO (last in first out) mechanism. The operations that can be performed on a stack are:

Push: Adding an element to the Top of stack. STACK Full condition is handled in push.

Pop: Deleting an element from the Top of the stack .STACK Empty condition is handled in pop.

6. PROCEDURE / PROGRAMME / ACTIVITY:

StackDemo.java

```
import java.util.Scanner;
```

```
class Stack {

   private int top;
   private int[] item;

   Stack(int size) {
      top = -1;
      item = new int[size];
   }

   public void push(int data) {
      if (top == item.length - 1) {
         System.out.println("Stack Overlow");
      } else {
         item[++top] = data;
      }
   }

  public int pop() {
      if (top < 0) {
         System.out.println("Stack Underflow");
         return 0;
      } else {
         System.out.println("Poped item is : " + item[top]);
         return item[top--];
      }
   }

    public void display() {
      if (top == -1) {
         System.out.println("Stack is empty");
      } else {
         System.out.print("Stack Items : ");
         for (int i = 0; i <= top; i++) {
            System.out.print(item[i] + "  ");
         }
         System.out.println("<-top");
      }
   }

}

class StackDemo {

   public static void main(String[] args){
      Stack stk = new Stack(3);
      boolean rerun = true;
      int choice, num;
```

```
        Scanner sc = new Scanner(System.in);

        do {
           System.out.println("\nMENU: 1)Push  2)Pop  3)Display  4)Exit ");
           System.out.println("Enter your choice: ");
           choice = sc.nextInt();

           switch (choice) {
              case 1:
                 System.out.println("Enter Item to  Pushed: ");
                 num = sc.nextInt();
                 stk.push(num);
                 break;
              case 2:
                 stk.pop();
                 break;
              case 3:
                 stk.display();
                 break;
              case 4:
                 rerun = false;
                 break;
              default:
                 System.out.println("Invalid Choice");
           }
        } while (rerun == true);
     }
}
```

7. RESULTS & CONCLUSIONS:

```
MENU: 1)Push  2)Pop  3)Display  4)Exit
Enter your choice:  1
Enter Item to  Pushed:
12

MENU: 1)Push  2)Pop  3)Display  4)Exit
Enter your choice: 1
Enter Item to  Pushed:
13

MENU: 1)Push  2)Pop  3)Display  4)Exit
Enter your choice: 1
Enter Item to  Pushed:
14

MENU: 1)Push  2)Pop  3)Display  4)Exit
Enter your choice: 3
Stack Items : 12  13  14  <-top
```

MENU: 1)Push 2)Pop 3)Display 4)Exit
Enter your choice: 2
Poped item is : 14

MENU: 1)Push 2)Pop 3)Display 4)Exit
Enter your choice: 2
Poped item is : 13

MENU: 1)Push 2)Pop 3)Display 4)Exit
Enter your choice: 2
Poped item is : 12

MENU: 1)Push 2)Pop 3)Display 4)Exit
Enter your choice: 3
Stack is empty

MENU: 1)Push 2)Pop 3)Display 4)Exit
Enter your choice: 2
Stack Underflow

MENU: 1)Push 2)Pop 3)Display 4)Exit
Enter your choice: 4

8. LEARNING OUTCOMES:

- Students are able to understand and implement operations of stack using object oriented concepts.

9. APPLICATION AREAS:

- Function calls/ Recursion
- Implementation of 'undo' control in any editor
- Balancing parenthesis in code, Expression evaluation and syntax parsing.

10. REMARKS:

-

1. EXPERIMENT NO: **2.a.**

2. TITLE: **INHERITANCE IN JAVA**

3. LEARNING OBJECTIVES:

- To understand and implement the concepts of inheritance in Java

4. AIM:

- 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.

5. THEORY / HYPOTHESIS:

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of

parent object. The idea behind inheritance in java is that you can create new classes that are built upon existing classes.
When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

6. PROCEDURE / PROGRAMME / ACTIVITY:

Staff.java

```
import java.util.Scanner;

public class Staff {

   private String staffid;
   private String name;
   private String phone;
   private double salary;

   void read() {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter StaffId, Name, Phone, salary ");
      staffid = sc.nextLine();
      name = sc.nextLine();
      phone = sc.nextLine();
      salary = sc.nextDouble();
   }

   void display() {
      System.out.print("StaffId:" + staffid + "\tName:" + name + "\tPhone:" + phone + "\tSalary:" +
salary);
   }

   public static void main(String[] args) {

      Teaching tch1 = new Teaching();
      Teaching tch2 = new Teaching();
      Teaching tch3 = new Teaching();

      System.out.println("\nEnter the details of teaching staff");
      tch1.read();
      tch2.read();
      tch3.read();
```

```
        Technical tec1 = new Technical();
        Technical tec2 = new Technical();
        Technical tec3 = new Technical();

        System.out.println("\nEnter the details of technical staff");
        tec1.read();
        tec2.read();
        tec3.read();

        Contract con1 = new Contract();
        Contract con2 = new Contract();
        Contract con3 = new Contract();

        System.out.println("\nEnter the details of Contract staff");
        con1.read();
        con2.read();
        con3.read();

        System.out.println("The details of teaching staff are ");
        tch1.display();
        tch2.display();
        tch3.display();
        System.out.println("The details of technical staff are");
        tec1.display();
        tec2.display();
        tec3.display();
        System.out.println("The details of contract staff are");
        con1.display();
        con2.display();
        con3.display();
    }
}

public class Teaching extends Staff{
    String domain;
    int publication;

    void read() {
        super.read();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter domain and total no. of publication ");
        domain = sc.nextLine();
        publication = sc.nextInt();
    }
    void display() {
        super.display();
        System.out.println("\tDomain:" + domain + "\tPublication : " + publication);
    }
```

```
}

class Technical extends Staff {

   String skills;

   void read() {
      super.read();
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the skills:  ");
      skills = sc.nextLine();
   }

   void display() {
      super.display();
      System.out.println("\tSkills:" + skills);
   }
}

class Contract extends Staff {

   double period;
      void read() {
      super.read();
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the period ");
      period = sc.nextDouble();
   }
   void display() {
      super.display();
      System.out.println("\tPeriod : " + period);
   }
}
```

7. RESULTS & CONCLUSIONS:

```
Enter the details of teaching staff
Enter StaffId, Name, Phone, salary
s123
Rahul
9878765670
35000
Enter domain and total no. of publication
network
5
…
…

Enter the details of technical staff
```

```
Enter StaffId, Name, Phone, salary
s225
Saurav
9999876679
45000
Enter the skills:
Programming
…
…

Enter the details of Contract staff
Enter StaffId, Name, Phone, salary
s454
Laxman
7876564444
23000
Enter the period
5
…
…

The details of teaching staff are
StaffId:s123   Name:Rahul   Phone:9878765670   Salary:35000.0        Domain:network
        Publication : 5
…
…
The details of technical staff are
StaffId:s225   Name:Saurav  Phone:9999876679   Salary:45000.0        Skills:Programming
…
…
The details of contract staff are
StaffId:s454   Name:LaxmanPhone:7876564444   Salary:23000.0        Period : 5.0
…
…
```

8. LEARNING OUTCOMES:

- Students are able to understand and implement the concepts of inheritance in Java

9. APPLICATION AREAS:

- For Method Overriding (so run time polymorphism can be achieved).
- For Code Reusability.

10. REMARKS:

-

1. EXPERIMENT NO: **2.b.**
2. TITLE: **DEMONSTRATION OF STRING TOKENIZER IN JAVA**
3. LEARNING OBJECTIVES:

- To learn the usage of StringTokenizer in Java

4. AIM:

- 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 "/".

5. THEORY / HYPOTHESIS:

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the *StreamTokenizer* class. The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments.

The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the *returnDelims* flag having the value true or false:

- If the flag is false, delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.
- If the flag is true, delimiter characters are themselves considered to be tokens. A token is thus either one delimiter character, or a maximal sequence of consecutive characters that are not delimiters.

A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed.

A token is returned by taking a substring of the string that was used to create the StringTokenizer object.

Construct	Description
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

private final String REGEX = "\\d"; // a single digit

In above example \d is the regular expression; the extra backslash is required for the code to compile. The test harness reads the expressions directly from the Console, however, so the extra backslash is unnecessary.

6. PROCEDURE / PROGRAMME / ACTIVITY:

Customer.java

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

public class  Customer {
```

```
    private String name;
    private String dob;

    public void Read() {
        System.out.println("Enter customer name : ");
        Scanner sc = new Scanner(System.in);
        name = sc.next();

        System.out.println("Enter customer DOB (dd/mm/yyyy):  ");
        dob = sc.next();

        String datePattern = "\\d{2}/\\d{2}/\\d{4}";

        while (!(dob.matches(datePattern))) {
            System.out.println("Please enter the  date in dd/mm/yyyy format!");
            dob = sc.next();
        }
    }

    public void Display() {

        StringTokenizer st = new StringTokenizer(dob, "/");
        String[] seq = new String[10];
        int i;

        System.out.print("Customer name and DOB is : " + name);
        for (i = 0; i < 3; i++) {
            System.out.print("," + st.nextToken());
        }
        System.out.println("\n");
    }

    public static void main(String args[]) {

        Customer c1 = new Customer();
        c1.Read();
        c1.Display();
    }
}
```

7. RESULTS & CONCLUSIONS:

```
Enter customer name :
Narendra
Enter customer DOB (dd/mm/yyyy):
3/6/82
Please enter the  date in dd/mm/yyyy format!
03/6/1982
Please enter the  date in dd/mm/yyyy format!
03/06/1982
```

Customer name and DOB is : Narendra,03,06,1982

8. LEARNING OUTCOMES:

- The student will be able to know the application of stringtokenizer in java

9. APPLICATION AREAS:

- Used to split the string in to number of tokens according to the given pattern.

10. REMARKS:

-

1. EXPERIMENT NO: **3.a.**

2. TITLE: **EXCEPTION HANDLING IN JAVA**

3. LEARNING OBJECTIVES:

- To apply the concept of exception handling in Java

4. AIM:

- 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.

5. THEORY / HYPOTHESIS:

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

- Checked exceptions − A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.
- Unchecked exceptions − An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
- Errors − These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

6. PROCEDURE / PROGRAMME / ACTIVITY:

ExceptionDemo.java

```
import java.util.Scanner;
public class ExceptionDemo {

   static int a, b, c;
   public static void main(String[] args) {
      System.out.println("Enter value for a and b :");
      Scanner sc1 = new Scanner(System.in);
      try {
         a = sc1.nextInt();
         b = sc1.nextInt();
         c = a / b;              // divide by zero is not allowed in integer arithmetic
         System.out.println("No Exception");
         System.out.println("Value of c =" + c);

      } catch (ArithmeticException ex1) {
         System.out.println(ex1);
      }
   }
}
```

7. RESULTS & CONCLUSIONS:

Output-1
Enter value for a and b :
3
2
No Exception
Value of c =1

Output-2
Enter value for a and b :
4
0
java.lang.ArithmeticException: / by zero

8. LEARNING OUTCOMES:

- On implementing this program the student will able to understand the usage of exception handling in java and will be able to apply the same in other similar problems.

9. APPLICATION AREAS:

- Separating Error-Handling Code from "Regular" Code
- Propagating Errors Up the Call Stack

10. REMARKS:

-

1. EXPERIMENT NO: **3.b.**

2. TITLE: **MULTI-THREADED APPLICATION IN JAVA**

3. LEARNING OBJECTIVES:

- To apply parallel programming using multi-threading in Java

4. AIM:

- 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.

5. THEORY / HYPOTHESIS:

Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread. Following are the stages of the life cycle −

- New − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- Runnable − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- Waiting − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- Timed Waiting − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- Terminated (Dead) − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

6. PROCEDURE / PROGRAMME / ACTIVITY:

MultithreadDemo.java

```
import java.util.*;

class Square implements Runnable {

   public int x;
```

```
    public Square(int num) {
        x = num;
    }

    public void run() {
        System.out.println("Thread 2: Square of " + x + " is: " + x * x);
    }
}

class Cube implements Runnable {

    public int x;

    public Cube(int num) {
        x = num;
    }

    public void run() {
        System.out.println("Thread 3: Cube of " + x + " is: " + x * x * x);
    }
}

class GenerateNos implements Runnable {

    public void run() {
        int num = 0;
        Random r = new Random();
        try {
            for (int i = 0; i < 5; i++) {
                num = r.nextInt(100);
                System.out.println("Thread 1: Generated random number is " + num);

                Thread t1 = new Thread(new Square(num));
                t1.start();

                Thread t2 = new Thread(new Cube(num));
                t2.start();

                Thread.sleep(1000);
                System.out.println("-------------------------------------");
            }
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
} // End of  GenerateNos

public class MultiThreadDemo {
```

```
  public static void main(String[] args) {
     Thread t = new Thread(new GenerateNos());
     t.start();
  }
}
```

7. RESULTS & CONCLUSIONS:

Sample Output

```
Thread 1: Generated random number is 80
Thread 2: Square of 80 is: 6400
Thread 3: Cube of 80 is: 512000
Thread 1: Generated random number is 0
Thread 2: Square of 0 is: 0
Thread 3: Cube of 0 is: 0
Thread 1: Generated random number is 9
Thread 2: Square of 9 is: 81
Thread 3: Cube of 9 is: 729
Thread 1: Generated random number is 46
Thread 2: Square of 46 is: 2116
Thread 3: Cube of 46 is: 97336
Thread 1: Generated random number is 91
Thread 2: Square of 91 is: 8281
Thread 3: Cube of 91 is: 753571
```

8. LEARNING OUTCOMES:

- On implementation of this program the student will be able to apply the multi-threading to problems where it is applicable.

9. APPLICATION AREAS:

- Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.
- Web Server software uses multi-threading to handle requests of multiple clients

10. REMARKS:

-

1. EXPERIMENT NO: **4.**

2. TITLE: **QUICK SORT**

3. LEARNING OBJECTIVES:

- To understand and apply the quick sort algorithm for sorting list
- To analyze the the time taken by the program for varying number of inputs

4. AIM:

- 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. THEORY / HYPOTHESIS:

Divide and Conquer is probably the best known general algorithm design technique. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

In the quick sort the given array is partitioned every time and the sub-array is sorted. Dividing is based on an element called pivot. Quick sort rearranges the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.

```
ALGORITHM: Quicksort(A[l....r])
//Sorts a subarray by quicksort
//Input: A subarray A[l..r] of A[0...n-1], defined by its left and right indices l and r
//Output: Subarray A[l...r} sorted in nondecreasing order
  if l< r
        s<- Partition(A[l...r]) //s is a split position
        Quicksort(A[l.....s-1])
        Quicksort(A[s+1.....r])

ALGORITHM: Partition(A[l..r])
//Partitions a subarray by Hoare's algorithm, using the first element as a pivot
//Input: Subarray of array A[0..n − 1], defined by its left and right indices l and r (l<r)
//Output: Partition of A[l..r], with the split position returned as this function's value
p←A[l]
i ←l; j ←r + 1
repeat
        repeat i ←i + 1 until A[i]≥ p
        repeat j ←j − 1 until A[j ]≤ p
        swap(A[i], A[j ])
until i ≥ j
swap(A[i], A[j ]) //undo last swap when i ≥ j
swap(A[l], A[j ])
return j
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

QuickSort.java

```
package quicksort;
import java.util.Random;
import java.util.Scanner;

public class QuickSort {

   private int[] a;
```

```
void input(){
   Scanner sc = new Scanner(System.in);
   Random rm = new Random();

   System.out.print("Enter the total numbers: ");
   int n = sc.nextInt();
   a = new int[n];
   for (int i = 0; i < n; i++) {
      a[i] = rm.nextInt(1000); // Generates random numbers 0-999
   }
}

void display(){
   for (int i : a) {
      System.out.print(i + " ");
   }
}

void sort(){
   quicksort(0, a.length - 1);
}

void quicksort(int left, int right) {
   if (left < right) {
      int s = partition(left, right);
      quicksort(left, s - 1);
      quicksort(s + 1, right);
   }
}

int partition(int left, int right) {
   int pivot = a[left];
   int i = left;
   int j = right + 1;

   do {

      do {
         ++i;
      } while (i < right && a[i] < pivot);

      do {
         --j;
      } while (a[j] > pivot);

      swap(i, j);

   } while (i < j);
```

```
        swap(i, j); // undo last swap
        swap(left, j);

        return j;
    }

    private void swap(int i, int j) {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }

    public static void main(String args[]) {

        QuickSort sorter = new QuickSort();
        sorter.input();
        System.out.println("Array before sorting");
        sorter.display();

        long startTime = System.nanoTime();
        sorter.sort();
        long endTime = System.nanoTime();
        double duration = (endTime - startTime) / 1000000.00;
                                        //divide by 1000000 to get milliseconds.
        System.out.println("\nArray After sorting");
        sorter.display();
        System.out.println("\nTime for sorting is " + duration + " milli seconds");
    } // end of main

} // end of class Quicksort
```

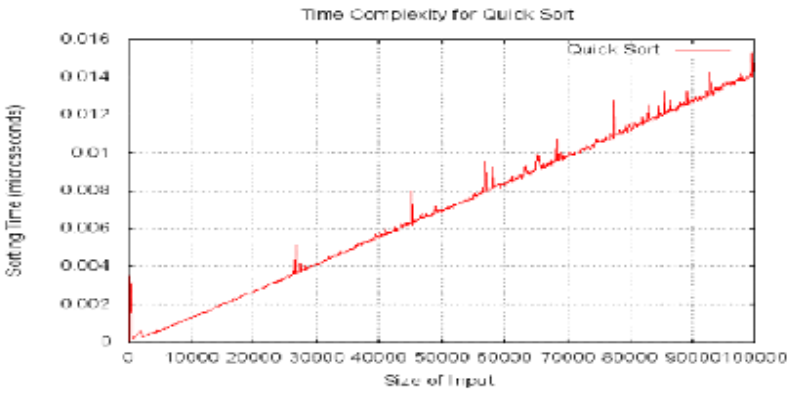
7. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:

Observation table: Run the algorithm for following input size

Total elements	5000	10000	15000	20000	25000	30000	35000	40000	45000	50000
Time to sort										

8. GRAPHS / OUTPUTS:

Plot the graph as per the data available in section 10 A sample graph is given below.



9. RESULTS & CONCLUSIONS:

Randomly generated elements were sorted successfully using quick sort.

Output

Enter the total numbers: 5
Array before sorting
619 630 34 698 760
Array After sorting
34 619 630 698 760
Time for sorting is 0.028026 milli seconds

10. LEARNING OUTCOMES:

- Students are able to implement quick sort method of sorting which is based on divide and conquer approach.

11. APPLICATION AREAS:

- Quicksort gained widespread adoption, for example, in Unix as the default library sort function.
- Useful when searching for the median, quartiles, deciles or percentiles.

12. REMARKS:

-

1. EXPERIMENT NO: **5**

2. TITLE: **MERGE SORT**

3. LEARNING OBJECTIVES:

- To understand the merge sort method of sorting which is based on divide and conquer approach.
- To study the time taken by the program for varying number of inputs.

4. AIM:

- 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.

5. THEORY / HYPOTHESIS:

Merge sort is an application of divide and conquer technique which sorts a given array by dividing into two halves and sorting each of them recursively and then merging the two smaller sorted arrays into a single sorted one.

```
ALGORITHM Mergesort(A[0..n − 1])
//Sorts array A[0..n − 1] by recursive mergesort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order

if n > 1
   copy A[0..n/2 − 1] to B[0..n/2 − 1]
   copy A[n/2..n − 1] to C[0..n/2 − 1]
   Mergesort(B[0..n/2 − 1])
   Mergesort(C[0..n/2 − 1])
   Merge(B, C, A)

ALGORITHM Merge(B[0..p − 1], C[0..q − 1], A[0..p + q − 1])
//Merges two sorted arrays into one sorted array
//Input: Arrays B[0..p − 1] and C[0..q − 1] both sorted
//Output: Sorted array A[0..p + q − 1] of the elements of B and C

i ←0; j ←0; k←0
while i <p and j <q do
        if B[i]≤ C[j ]
                A[k]←B[i]; i ←i + 1
        else
                A[k]←C[j ]; j ←j + 1
        k←k + 1
        if i = p
                copy C[j..q − 1] to A[k..p + q − 1]
        else
                copy B[i..p − 1] to A[k..p + q − 1]
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

MergeSort.java

```
package mergesort;
import java.util.Random;
import java.util.Scanner;

public class MergeSort {

   private int[] a;
```

```
void input() {
    Scanner sc = new Scanner(System.in);
    Random rm = new Random();

    System.out.print("Enter the total numbers: ");
    int n = sc.nextInt();
    a = new int[n];
    for (int i = 0; i < n; i++) {
        a[i] = rm.nextInt(1000); // Generates random numbers 0-999
    }                           \
}

void display() {
    for (int i : a) {
        System.out.print(i + " ");
    }
}

void sort() {
    mergesort(0, a.length - 1);
}

void mergesort(int left, int right) {
    int mid;
    if (left < right) {
        mid = (left + right) / 2;
        mergesort(left, mid);
        mergesort(mid + 1, right);
        merge(left, mid, right);
    }
}

void merge(int left, int mid, int right) {

    int temp[] = new int[a.length];
    for (int i = left; i <= right; i++) {
        temp[i] = a[i];
    }
    int i = left;
    int j = mid + 1;
    int k = left;
    while (i <= mid && j <= right) {
        if (temp[i] <= temp[j]) {
            a[k++] = temp[i++];
        } else {
            a[k++] = temp[j++];
        }
    }
```

```
        while (i <= mid) {
            a[k++] = temp[i++];
        }
        while (j <= right) {
            a[k++] = temp[j++];
        }
    }

    public static void main(String args[]) {

        MergeSort sorter = new MergeSort();

        sorter.input();
        System.out.println("Array before sorting");
        sorter.display();

        long startTime = System.nanoTime();
        sorter.sort();
        long endTime = System.nanoTime();
        double duration = (endTime - startTime) / 1000000.00;
                                                    //divide by 1000000 to get milliseconds.
        System.out.println("\nArray After sorting");
        sorter.display();

        System.out.println("\nTime for sorting is " + duration + " milli seconds");
    }
}
```

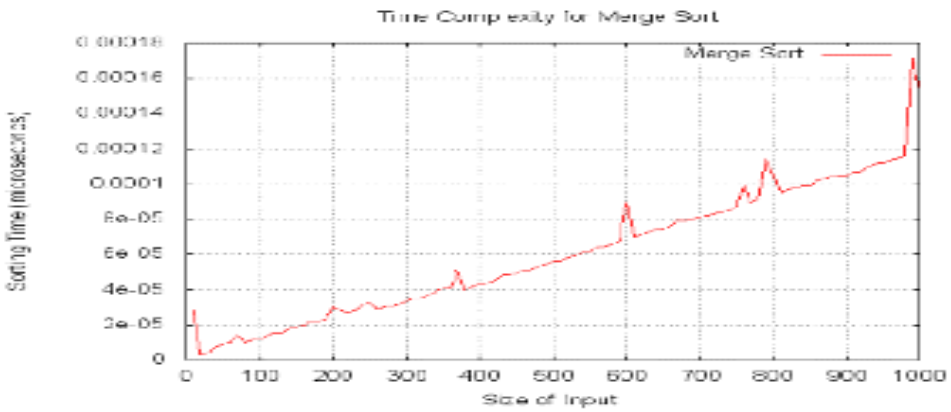
7. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:

Observation table: Run the algorithm for folowing input size

Total elements	5000	10000	15000	20000	25000	30000	35000	40000	45000	50000
Time to sort										

8. GRAPHS / OUTPUTS:

Sample graph



9. RESULTS & CONCLUSIONS:

Enter the total numbers: 7
Array before sorting
108 162 105 201 459 273 198
Array After sorting
105 108 162 198 201 273 459
Time for sorting is 0.043815 milli seconds

10. LEARNING OUTCOMES :

- Students are able to understand and implement the merge sort method of sorting which is based on divide and conquer approach.

11. APPLICATION AREAS:

- Records are stored on magnetic tape and processed on banks of magnetic tape drives. Merge sort is implemented with disk drives.

12. REMARKS:

-

1. EXPERIMENT NO: **6.a**

2. TITLE: **0/1 KNAPSACK PROBLEM USING DYNAMIC PROGRAMMING**

3. LEARNING OBJECTIVES:

- Understand the 0/1 Knapsack Problem using Dynamic programming.

4. AIM:

- Implement in Java, the 0/1 Knapsack problem using Dynamic Programming method

5. THEORY / HYPOTHESIS:

Given a knapsack with maximum capacity W,and a set S consisting of N items. Each item *i* has some weight wi and benefit value bi (all wi and W are integer values)

Problem: How to pack the knapsack to achieve maximum total value of packed items? i.e.,to find $\max \sum_{i \in T} b_i$ subject to $\sum_{i \in T} w_i \le W$

The problem is called a "0-1" problem, because each item must be entirely accepted or rejected

```
Algorithm: 0/1Knapsack(S, W)
//Input: set S of items with benefit bi and weight wi; max. weight W
//Output: benefit of best subset with weight at most W
//Sk: Set of items numbered 1 to k.
//Define B[k,w] = best selection from Sk with weight exactly equal to w
{
   for w ← 0 to n-1 do
      B[w] ← 0

   for k ← 1 to n do
   {
       for w ← W down to wk do
       {
            if B[w-wk]+bk > B[w] then
           B[w] ← B[w-wk]+bk
       }
   }
}
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

Knapsack01_DynPrg.java

```
package knapsack01dp;
import java.util.Scanner;

public class Knapsack01_DynPrg {
   static int[] p, wt;
   static int C, n;

   static void knapsack01_DP() {
      int i, j, w;
      int[ ][ ] K = new int[n + 1][C + 1];

      // Build table K[][] in bottom up manner
      for (i = 0; i <= n; i++) {
         for (w = 0; w <= C; w++) {
            if (i == 0 || w == 0) {
               K[i][w] = 0;
            } else if (wt[i] <= w) {
               K[i][w] = max(p[i] + K[i - 1][w - wt[i]], K[i - 1][w]);
            } else {
               K[i][w] = K[i - 1][w];
            }
         }
      }

      System.out.println("The selected items are :  ");
      int c = C;
```

```
    int m = n;

    while (m > 0)   //best subset with weight at most knapsack size
    {
       if (K[m][c] != K[m - 1][c]) {
          System.out.println("Item " + m+ " (weight:" + wt[m] + ", profit:" + p[m] + ")");
          c = c - wt[m];
       }
       m--;
    }

    System.out.println("Total profit of the items added to knapsack = " + K[n][C]);

    /***************** To show Intermadiate Results ****************
    System.out.print("\nIntermediate results \nCapacity:  ");
    for (j=0; j<=C;j++){
       System.out.print(j+"\t");
    }
    System.out.println();

    for (i=0;i<=n;i++){
       System.out.print("Item "+i+"  :  ");
       for (j=0; j<=C;j++){
          System.out.print(K[i][j]+"\t");
       }
       System.out.println();
    }
    ************************************************************  */
}

static int max(int a, int b) {
    return (a > b) ? a : b;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of items: ");
    n = sc.nextInt();

    System.out.println("Enter the items profit: ");
    p = new int[n + 1];
    for (int i = 1; i <= n; i++) {      // We store from index 1 to n
       p[i] = sc.nextInt();
    }

    System.out.println("Enter the items weights: ");
    wt = new int[n + 1];
    for (int i = 1; i <= n; i++) {      // We store from index 1 to n
       wt[i] = sc.nextInt();
```

```
        }

        System.out.println("Enter the maximum capacity: ");
        C = sc.nextInt();

        System.out.println("\n0/1 Knapsack using Dynamic Programming");
        knapsack01_DP();

    }
}
```

7. RESULTS & CONCLUSIONS:

Output-1

```
Enter the number of items:
4
Enter the items profit:
12 10 20 15
Enter the items weights:
2 1 3 2
Enter the maximum capacity:
5

0/1 Knapsack using Dynamic Programming
The selected items are :
Item 4 (weight:2, profit:15)
Item 2 (weight:1, profit:10)
Item 1 (weight:2, profit:12)
Total profit of the items added to knapsack = 37
```

Output-2

```
Enter the number of items:
4
Enter the items profit:
15 16 18 22
Enter the items weights:
1 3 6 7
Enter the maximum capacity:
6

0/1 Knapsack using Dynamic Programming
The selected items are :
Item 2 (weight:3, profit:16)
Item 1 (weight:1, profit:15)
Total profit of the items added to knapsack = 31
```

8. LEARNING OUTCOMES:

- Students are able to implement 0/1 knapsack problem using dynamic programming

9. APPLICATION AREAS:
- Hard-real time systems-cache memory management ,dynamic storage management.

10. REMARKS:

-

1. EXPERIMENT NO: **6.b**

2. TITLE: **0/1 KNAPSACK PROBLEM USING GREEDY METHOD.**

3. LEARNING OBJECTIVES:
- To apply greedy method
- Solve the 0/1 Knapsack Problem using Greedy Method

4. AIM:
- Implement in Java, the 0/1 Knapsack problem using Greedy method.

5. THEORY / HYPOTHESIS:

Given a knapsack with maximum capacity W,and a set S consisting of N items. Each item *i* has some weight wi and benefit value bi (all wi and W are integer values)

Problem: How to pack the knapsack to achieve maximum total value of packed items? i.e.,to find

$$\max \sum_{i \in T} b_i \text{ subject to } \sum_{i \in T} w_i \le W$$

The problem is called a "0-1" problem, because each item must be entirely accepted or rejected There are different ways to solve 0/1 Knapsack Problem using Greedy Method.

Here we use profit/weight ratio to solve using greedy method

Note: Even though 0/1 Knapsack problem using Greedy method solves the problem it does not does not guarantee optimal solution. The greedy method always give optimal solution for fractional knapsack problem.

6. PROCEDURE / PROGRAMME / ACTIVITY:

Knapsack01_Greedy

```
package knapsack01Greedy;
import java.util.Scanner;

public class Knapsack01_Greedy {
   static int[] p, wt;
   static int C, n;

   static void knapsack01_Gdy() {
      int i, j, t;
      int[] itemNo = new int[n + 1];
      double[] ratio = new double[n + 1];
      double r;
```

```
    for (i = 1; i <= n; i++) {
       itemNo[i] = i;
       ratio[i] = p[i] / wt[i];
    }

    int c = C, profit = 0;
    for (i = 1; i <= n; i++) {
       if (c >= wt[i]) {
          System.out.println("Item " + itemNo[i] +
                    "(weight:" + wt[i] + ", profit:" + p[i] + ") is added to knapsack");
          profit = profit + p[i];
          c = c - wt[i];
       }
    }

    System.out.println("Total profit of the items added to knapsack = " + profit);

  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of items: ");
    n = sc.nextInt();

    System.out.println("Enter the items profit: ");
    p = new int[n + 1];
    for (int i = 1; i <= n; i++) {      // We store from index 1 to n
       p[i] = sc.nextInt();
    }

    System.out.println("Enter the items weights: ");
    wt = new int[n + 1];
    for (int i = 1; i <= n; i++) {      // We store from index 1 to n
       wt[i] = sc.nextInt();
    }

    System.out.println("Enter the maximum capacity: ");
    C = sc.nextInt();

    System.out.println("\n0/1 Knapsack using Greedy Method");
    knapsack01_Gdy();

  }
}
```

7. RESULTS & CONCLUSIONS:

Output-1

Enter the number of items:
4
Enter the items profit:
12 10 20 15
Enter the items weights:
2 1 3 2
Enter the maximum capacity:
5

0/1 Knapsack using Greedy Method

Item 2 (weight:1, profit:10) is added to knapsack
Item 4 (weight:2, profit:15) is added to knapsack
Item 1 (weight:2, profit:12) is added to knapsack

Total profit of the items added to knapsack = 37

Output-2

Enter the number of items:
4
Enter the items profit:
15 16 18 22
Enter the items weights:
1 3 6 7
Enter the maximum capacity:
6

0/1 Knapsack using Greedy Method

Item 1 (weight:1, profit:15) is added to knapsack
Item 2 (weight:3, profit:16) is added to knapsack

Total profit of the items added to knapsack = 31

8. LEARNING OUTCOMES:

- Students are able to implement 0/1 knapsack problem using dynamic programming

9. APPLICATION AREAS:

- Hard-real time systems-cache memory management, dynamic storage management.

10. REMARKS:

-

1. EXPERIMENT NO: **7**

2. TITLE: **DIJKSTRA'S ALGORITHM**

3. LEARNING OBJECTIVES:

- To know about weighted connected graph.
- To know about finding a single shortest path from a given vertex.

4. AIM:

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

5. THEORY / HYPOTHESIS:

Dijkstra's algorithm is a graph search algorithm that solves the single source shortest path problem for a graph with non negative edge path costs,outputting a shortest path tree.
For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).
Algorithm **:** Dijkstra(G,s)
//Dijkstra's algorithm for single-source shortest paths
//Input :A weighted connected graph G=(V,E) with nonnegative weights and its vertex s
//Output : The length dv of a shortest path from s to v and its penultimate vertex pv for
//every v in V.

```
{
Initialise(Q) // Initialise vertex priority queue to empty
for every vertex v in V do
{
dv←oe; pv←null
Insert(Q,v,dv) //Initialise vertex priority queue in the priority queue
}
ds←0; Decrease(Q,s ds) //Update priority of s with ds
Vt←Ø
for i←0 to |v|-1 do
        {
          u* ← DeleteMin(Q) //delete the minimum priority element
             Vt ←Vt U
             {
                    u*
             }
                    for every vertex u in V-Vt that is adjacent to u* do
                      {
                              if du* + w(u*,u)<du
                                  {
                                      du←du* + w(u*, u): pu←u*
                                        Decrease(Q,u,du)
                                       }
```

```
                    }
            }
}
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

DijkstraSP.java

```
package dijkstrasp;
import java.util.Scanner;

public class DijkstraSP {

   static int[][] wt;
   static int[] prev,visit,dist;
   static int s, n;

   static void dijkstra() {
      int i, j, step, u;
      dist = new int [n+1];
      prev = new int [n+1];
      visit = new int [n+1];

      for (i = 1; i <= n; i++) {
         dist[i] = 0;
         prev[i] = 0;
         visit[i] = 0;
      }

      for (i = 1; i <= n; i++) {
         dist[i] = wt[s][i];
         if (dist[i] == 99) {
            prev[i] = 0;
         } else {
            prev[i] = s;
         }
      }
      visit[s] = 1;
      dist[s] = 0;
      for (step = 2; step <= n; step++) {
         u = MinVertex();    //choose the minimum distance vertex onsource
         visit[u] = 1;
         for (j = 1; j <= n; j++) //update distances of fringe vertices from source
         {
            if (((dist[u] + wt[u][j]) < dist[j]) && visit[j]==0 ) {
               dist[j] = dist[u] + wt[u][j];
               prev[j] = u;
            }
         }
```

```
    }
    printpath();
}

static int MinVertex() {
    int min = 99;
    int u=0, i;
    for (i = 1; i <= n; i++) {
        if ((dist[i] < min) && (visit[i] == 0)) {
            min = dist[i];
            u = i;
        }
    }
    return u;
}

static void printpath() {
    int i, t;
    for (i = 1; i <= n; i++) {
        if (visit[i] == 1 && i != s) {
            System.out.print(s + " to " + i +":  distance=" + dist[i] +"  path: ");
            t = prev[i];
            System.out.print(i);
            while (t != s) {
                System.out.print("<--" + t);
                t = prev[t];
            }
            System.out.println("<--" + s);

        }
    }
}

public static void main(String[] args) {
    int i, j;

    Scanner sc = new Scanner(System.in);

    System.out.println("Enter the number of vertices in a graph : ");
    n = sc.nextInt();

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

    //It is assumes that sum of all weights is less than 99
    System.out.println("Enter the weight matrix (99 for no edge):");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            wt[i][j] = sc.nextInt();
        }
```

```
        }

        System.out.print("Enter the source vertex: ");
        s = sc.nextInt();
        System.out.println("Shortest paths are");
        dijkstra();
    }
}
```

7. RESULTS & CONCLUSIONS:

```
Enter the number of vertices in a graph :
4
Enter the weight matrix (99 for no edge):
0 2 5 1
2 0 1 2
5 1 0 3
1 2 3 0
Enter the source vertex: 3
Shortest paths are
3 to 1:  distance=3  path: 1<--2<--3
3 to 2:  distance=1  path: 2<--3
3 to 4:  distance=3  path: 4<--3
```

8. LEARNING OUTCOMES:

- Students are able to implement java program to find single shortest path using dijkstra's algorithm

9. APPLICATION AREAS:

- Networks-Routing

10. REMARKS:

-

1. EXPERIMENT NO: **8.**

2. TITLE: **KRUSKAL'S ALGORITHM**

3. LEARNING OBJECTIVES:

- To understand the minimum cost spanning tree.
- To implement minimum cost spanning tree using Kruskal's algorithm.

4. AIM:

- To find minimum cost spanning tree of a given undirected graph using Kruskal's algorithm

5.THEORY / HYPOTHESIS:

Kruskal's algorithm finds the minimum spanning tree for a weighted connected graph G=(V,E) to get an acyclic subgraph with |V|-1 edges for which the sum of edge weights is the smallest. Consequently the algorithm constructs the minimum spanning tree as an expanding sequence of subgraphs, which are always acyclic but are not necessarily connected on the intermediate stages of algorithm.

The algorithm begins by sorting the graph's edges in non decreasing order of their weights. Kruskal's algorithm finds the minimum spanning tree for a weighted connected graph G=(V,E) to Then starting with the empty subgraph, it scans the sorted list adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

Algorithm **:** Kruskal(G)

```
// Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G=(V,E)
//Output: ET,the set of edges composing a minimum spanning tree of G
{
   Sort E in non decreasing order of the edge weights w(ei1)<=…….>= w(ei |E|))
   ET ← ∅ ; ecounter ← 0 //Initialize the set of tree edges and its size
   k ← 0 //initialize the number of processed edges
  while ecounter <|V|-1 do
   {
      k ← k+1
      if ET U {eik} is acyclic
      ET ← ET U {eik}; ecounter ← ecounter+1
   }
   return ET
}
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

KruskalMST.java

```
package kruskalmst;
import java.util.Scanner;

public class KruskalMST {

   static int[][] wt;
   static int n, mincost, v1 = 0, v2 = 0;

   static void kruskal() {
      int i, edgewt;
      int[] root = new int[n + 1];
      for (i = 1; i <= n; i++) {
         root[i] = i;
      }

      System.out.println("Edges of min-cost spanning tree are");
      i = 0;
      while (i != n - 1) {
         findMin(); // updates v1 & v2 with vertices of edge with minimum weight
         edgewt = wt[v1][v2];
         wt[v1][v2] = wt[v2][v1] = 0; //supresses selection of same edge on next time.
         if (root[v1] != root[v2]) {
            System.out.println("(" + v1 + "," + v2 + ")");
            doUnion(root, v1, v2);
```

```
            mincost += edgewt;
            i++;
         }
      }
      System.out.println("Cost of min-cost spanning tree = " + mincost);
   }

   static void findMin()//finding the edge having minimum weight.
   {
      int edgewt = 99, i, j;
      for (i = 1; i <= n; i++) {
         for (j = i + 1; j <= n; j++) {
            if (wt[i][j] > 0 && wt[i][j] < edgewt) {
               edgewt = wt[i][j];
               v1 = i;
               v2 = j;
            }
         }
      }
   }
   static void doUnion(int root[], int v1, int v2) {
      int temp, i;
      temp = root[v2];
      for (i = 1; i <= n; i++) {
         if (root[i] == temp) {
            root[i] = root[v1];
         }
      }
   }

  public static void main(String[] args) {
      int i, j;
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the number of vertices");
      n = sc.nextInt();

      wt = new int[n+1][n+1];
      System.out.println("Enter the undirected graph as adjacency matrix");
      for (i = 1; i <= n; i++) {
         for (j = 1; j <= n; j++) {
            wt[i][j] = sc.nextInt();
         }
      }
      kruskal();
   }
}
```

7. RESULTS & CONCLUSIONS:

```
Enter the number of vertices
4
```

Enter the undirected graph as adjacency matrix
0 2 99 2
2 0 3 1
99 3 0 5
2 1 5 0
Edges of min-cose spanning tree are
(2,4) (1,2) (2,3)
Cost of min-cost spanning tree = 6

8. LEARNING OUTCOMES:

- Students are able to implement java program to find minimum cost spanning tree using Kruskal algorithm

9. APPLICATION AREAS:

- Clustering Analysis
- Traveling Salesman Problem Approximation

10. REMARKS:

1. EXPERIMENT NO: **9**

2. TITLE: **PRIM'S ALGORITHM**

3. LEARNING OBJECTIVES:

- Learn about minimum cost spanning tree of weighted undirected graph.
- Learn about Prim's algorithm for constructing minimum cost spanning tree of weighted undirected graph.

4. AIM:

- Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm. Implement the program in Java language.

5. THEORY / HYPOTHESIS:

The program uses Prim's algorithm which is based on minimum spanning tree for a connected undirected graph. A predefined cost adjacency matrix is the input. To find the minimum spanning tree, we choose the source node at random and in every step we find the node which is closest as well as having the least cost from the previously selected node and also the cost of selected edge is being added to variable sum. Based on the value of sum, the presence of the minimum spanning tree is found.

Algorithm: Input: A non-empty connected weighted graph with vertices V and edges E
Initialize: Vnew = {x}, where x is an arbitrary node (starting point) from V, Enew = {}
Repeat until Vnew = V:
Choose an edge {u, v} with minimal weight such that u is in Vnew and v is not
Add v to Vnew, and {u, v} to Enew
OUTPUT: VNEW AND ENEW DESCRIBE A MINIMAL SPANNING TREE

6. PROCEDURE / PROGRAMME / ACTIVITY:

```
package primsmst;

import java.util.Scanner;
```

```
public class PrimsMST {

   static int[][] wt, edges;
   static int n, cost;

   static void prims() {
      int[] u,lowcost,visited;
      int min, mincost = 0, i, j, v;

      u = new int[n+1];
      lowcost = new int[n+1];
      visited = new int[n+1];
      edges = new int[n*n][3];

      //mark nodes as unvisited
      visited[1] = 1;

      //find low cost edge
      for (i = 2; i <= n; i++) {
         visited[i] = 0;
         u[i] = 1;
         lowcost[i] = wt[1][i];
      }
      for (i = 1; i <= n - 1; i++) {
         min = lowcost[2];
         v = 2;
         for (j = 3; j <= n; j++) {
            if (lowcost[j] < min) {
               min = lowcost[j];
               v = j;
            }
         }

         //save edge
         edges[i][1] = u[v];
         edges[i][2] = v;
         mincost += lowcost[v];
         visited[v] = 1;
         lowcost[v] = 99;

         for (j = 2; j <= n; j++) {
            if ( wt[v][j] < lowcost[j] && visited[j]==0 ) {
               lowcost[j] = wt[v][j];
               u[j] = v;
            }
         }

         System.out.print("\nIteration with i="+i+"\nlowcost:");
         for (int p = 1; p <= n; p++)
```

```
            System.out.print( lowcost[p]  + "  ");

         System.out.print("\n       u:");
         for (int p = 1; p <= n; p++)
            System.out.print( u[p]  + "  ");
      }
      System.out.println("\nEdges of min-cost spanning tree are");
      for (i = 1; i <= n - 1; i++) {
         System.out.println("(" + edges[i][1] + "," + edges[i][2] + ")");
      }
      System.out.println("Cost of min-cost spanning tree = " + mincost);
   }

   public static void main(String[] args) {
      int i, j;
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter the number of vertices");
      n = sc.nextInt();

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

      // It is assumed that total
      System.out.println("Enter the adjacency matrix of undirected graph as");
      for (i = 1; i <= n; i++) {
         for (j = 1; j <= n; j++) {
            wt[i][j] = sc.nextInt();
         }
      }
      prims();
   }
}
```

7. RESULTS & CONCLUSIONS:

```
Enter the number of vertices 4
 Enter the weighted matrix of undirected graph
0 2 99 2
2 0  5  3
99 5 0 4
2   3 4 0
Edges of min-cose spanning tree are
(1,2) (1,4) (4,3)
Cost of min-cost spanning tree = 8
```

8. LEARNING OUTCOMES:

- Students are to develop program based on Prim's algorithm for constructing minimum cost spanning tree of weighted undirected graph
- Program based on Prim's algorithm for constructing minimum cost spanning tree of weighted undirected graph is implemented.

9. APPLICATION AREAS:

- Prim's algorithm is an algorithm in graph theory finds a minimum spanning tree for connected weighted graph
- Minimum spanning trees have direct applications in the design of networks, including computer networks, telecommunications networks,transportation networks,water supply networks.
- Minimum spanning trees are used in case of image segmentation in image processing.
- Minimum spanning trees are also used in regionalisation of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions

10. REMARKS:

-

1. EXPERIMENT NO: **10.a**

2. TITLE: **FLOYD'S ALGORITHM**

3. LEARNING OBJECTIVES:

- Learn about All-Pairs Shortest Paths Problem to find the distances from each vertex to all other vertices in a given weighted connected graph((directed or undirected).
- Learn about Floyd's algorithm for All-Pairs Shortest Paths Problem.

4. AIM:

- Write Java programs to Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

5. THEORY / HYPOTHESIS:

Floyd's algorithm is applicable to both directed and undirected graphs provided that they do not contain a cycle. It is convenient to record the lengths of shortest path in an n- by- n matrix D called the distance matrix. The element dij in the ith row and jth column of matrix indicates the shortest path from the ith vertex to jth vertex (1<=i, j<=n). The element in the ith row and jth column of the current matrix D(k-1) is replaced by the sum of elements in the same row i and kth column and in the same column j and the kth column if and only if the latter sum is smaller than its current value.

Algorithm Floyd(W[1..n,1..n])
//Implements Floyd's algorithm for the all-pairs shortest paths problem
//Input: The weight matrix W of a graph
//Output: The distance matrix of shortest paths length

```
{
  D ← W
  for k←1 to n do
  {
      for i ← 1 to n do
      {
            for j ← 1 to n do
            {
                D[i,j] ← min (D[i, j], D[i, k]+D[k, j] )
            }
      }
  }
   return D
}
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

```
package floyds;

import java.util.Scanner;


public class Floyds {

   static int[][] dist;
   static int n;

   static void floyds() {
      int i, j, k;
      for (k = 1; k <= n; k++) // record the lengths of shortest path
      {
         for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
               dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
         }
      }
   }

   static int min(int a, int b) {
      if (a < b) {
         return a;
      } else {
         return b;
      }
   }

   public static void main(String[] args) {
      int i, j;
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter the number of vertices");
      n = sc.nextInt();
      dist = new int[n + 1][n + 1];
      System.out.println("Enter the adjacency matrix (99 for no edge)");
      for (i = 1; i <= n; i++) {
         for (j = 1; j <= n; j++) {
            dist[i][j] = sc.nextInt();
         }
      }
      floyds();

      System.out.println("All pairs shortest path matrix is");
      for (i = 1; i <= n; i++) {
```

```
            for (j = 1; j <= n; j++) {
                System.out.print(dist[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

7. RESULTS & CONCLUSIONS:

Enter the number of vertices
4
Enter the adjacency matrix (99 for no edge)
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
All pairs shortest path matrix is

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

Conclusion

- All-Pairs Shortest Paths Problem to find the distances from each vertex to all other vertices in a given weighted connected graph is studied.
- Program based on Floyd's algorithm for All-Pairs Shortest Paths Problem is implemented by using parallel execution technique

8. LEARNING OUTCOMES:

- Students are able to develop program based on Floyd's algorithm for All-Pairs Shortest Paths Problem.
-

9. APPLICATION AREAS:

- Shortest path algorithms are applied to automatically find directions between physical locations, such as driving directions on web mapping websites like google maps.
- Other applications include robotics, VLSI design

10. REMARKS:

-

1. EXPERIMENT NO: **10.b**
2. TITLE: **TSP USING DYNAMIC PROGRAMMING**
3. LEARNING OBJECTIVES:
   - To find the optimal solution for the given travelling salesman problem
4. AIM: Write Java programs to implement Travelling Sales Person problem using Dynamic programming.
5. THEORY / HYPOTHESIS:

Travelling Salesperson problem: Given n cities, a salespersTravelingon starts at a specified city

(source), visit all n-1 cities only once and return to the city from where he has started. The objective of this problem is to find a route through the cities that minimizes the cost and thereby maximizing the profit.

Algorithm: TSP(start city, current city,next city, path)
//Purpose: Tof ind the solution for TSP problem using exhaustive search
//Input: The start city, current city,next city and the path
//Output: The minimum distance covered along with the path
Step 1: Check for the disconnection between the current city and the next city
Step 2: Check whether the travelling sales person has visited all the cities
Step 3: Find the next city to be visited
Step 4: Find the solution and terminate

6. PROCEDURE / PROGRAMME / ACTIVITY:

TSP.java

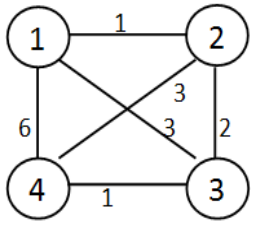
```
package TSP;
import java.util.Scanner;
public class TSP {
   int[][] d;
   int[] visited, finaltour;
   int n, cost;

   void read() {
      Scanner sc = new Scanner(System.in);
      int i, j;
      System.out.println("Enter the total cities ");
      n = sc.nextInt() + 1;
      d = new int[n][n];
      visited = new int[n];

       // It is assumed that sum of all distances is less than 99
      System.out.println("Enter the distance matrix (99 for no connectivity)");
      for (i = 1; i < n; i++) {
         for (j = 1; j < n; j++) {
            d[i][j] = sc.nextInt();
         }
         visited[i] = 0;
      }

      finaltour = new int[n];
      for (i = 1; i < n; i++) {
         finaltour[i] = i;
      }
   }

   void tsp_dyn_prg() {
      cost = tsp_dp(finaltour, 1);
      System.out.println("\nTraversal Path using Dyn. Prgg:");
      for (int i = 1; i < n; i++) {
         System.out.print(finaltour[i] + " > ");
```

```
    }
    System.out.println("1");
    System.out.println("Minimum cost : " + cost);
}

int tsp_dp(int[] tour, int start) {

    int i, j, k;        /* Loop counters. */
    int[] temp = new int[n];        /* Temporary array  during calculations. */
    int[] mintour = new int[n];        /* Minimal tour array. */
    int mincost;        /* Minimal cost. */
    int ccost;        /* Current cost. */

    /* End of recursion condition. */
    if (start == n - 2) {
       return d[tour[n - 2]][tour[n - 1]] + d[tour[n - 1]][1];
    }

    /* Compute the tour starting from the current city. */
    mincost = 99;
    for (i = start + 1; i < n; i++) {
       for (j = 1; j < n; j++) {
          temp[j] = tour[j];
       }

       /* Adjust positions. */
       temp[start + 1] = tour[i];
       temp[i] = tour[start + 1];

       /* Found a better cycle? (Recurrence derivable.) */
       if (d[tour[start]][tour[i]] + (ccost = tsp_dp(temp, start + 1)) < mincost) {
          mincost = d[tour[start]][tour[i]] + ccost;
          for (k = 1; k < n; k++) {
             mintour[k] = temp[k];
          }
       }
    }
    /* Set the minimum-tour array. */
    for (i = 1; i < n; i++) {
       tour[i] = mintour[i];
    }
    return mincost;
}

public static void main(String[] args) {

    TSP t = new TSP();
    t.read();
    t.tsp_dyn_prg();
```

```
    }

}
```

7. RESULTS & CONCLUSIONS:

Enter the total cities
4
Enter the distance matrix (99 for no connectivity)
0 1 3 6
1 0 2 3
3 2 0 1
6 3 1 0



Traversal Path using Dyn. Prgg:
1 > 2 > 4 > 3 > 1
Minimum cost : 8

8. LEARNING OUTCOMES:

- Students are able to implement traveling salesperson problem using a gynamic programming

9. APPLICATION AREAS:

- In the field of data communication to get the shortest path between two nodes

10. REMARKS:

-

1. EXPERIMENT NO: **11**

2. TITLE: **SUM OF SUBSET PROBLEM**

3. LEARNING OBJECTIVES:

- To find the optimal solution to a sum of subset problem by applying backtracking algorithm

4. AIM:

- 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.

5.THEORY / HYPOTHESIS:

Sum of Subsets: Subset-Sum Problem is to find a subset of a given set S= {s1, s2… sn} of n positive integers whose sum is equal to a given positive integer d. It is assumed that the set's elements are sorted in increasing order. The state-space tree can then be constructed as a binary tree and applying backtracking algorithm, the solutions could be obtained. Some instances of the problem may have no solutions

Algorithm: SumOfSub(s, k, r)
//Find all subsets of w[1…n] that sum to m. The values of x[j], 1<= j < k, have already been determined. s=Σk-1 w[j]*x[j] and r =Σn w[j]. The w[j]'s are in ascending order.
j=1

```
j=k
{
  x[k] ← 1 //generate left child
  if (s+w[k] = m)
       write (x[1...n]) //subset found
   else if ( s + w[k]+w[k+1] <= m)
        SumOfSub( s + w[k], k+1, r-w[k])
  //Generate right child
  if( (s + r - w[k] >= m) and (s + w[k+1] <= m) )
  {
     x[k] ← 0
     SumOfSub( s, k+1, r-w[k] )
  }
}
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

SumOfSubset.java

```
package sumofsubset;
import java.util.Scanner;

public class SumOfSubset {

   static int[] set, x;
   static int d, n;

   static void sumofsub(int s, int k) {
      int i;

      x[k] = 1;   //generate left child
      if (s + set[k] == d) {
         System.out.print("{");
         for (i = 1; i <= n; i++) {
            if (x[i] == 1) //subset found
            {
               System.out.print( set[i] + ",");
            }
         }
         System.out.println("\b}");
      } else {
         if (s + set[k] < d && k + 1 <= n) //with s[k]
         {
            sumofsub(s + set[k], k + 1);    //Generate right child
            x[k + 1] = 0;
         }
         if (k + 1 <= n && s + set[k + 1] <= d )//without s[k]
         {
            x[k] = 0;
```

```
            sumofsub(s, k + 1);
            x[k + 1] = 0;
        }
    }
}

public static void main(String[] args) {
    int sum = 0, i;
    Scanner sc = new Scanner(System.in);

    System.out.println("Enter size of the set");
    n = sc.nextInt();

    set = new int [n+1];
    x = new int [n+1];


    System.out.println("Enter set elements in increasing order");
    for (i = 1; i <= n; i++) {
        set[i] = sc.nextInt();
    }
    System.out.println("Enter maximum limit for sum");
    d = sc.nextInt();
    for (i = 1; i <= n; i++) {
        sum = sum + set[i];
    }
    if (sum < d || set[1] > d) {
        System.out.println("No subset possible");
    } else {
        System.out.println("The subsets with sum of elements = " + d + " are");
        sumofsub(0, 1);
    }
}
}
```

7. RESULTS & CONCLUSIONS:

```
Enter size of the set
5
Enter set elements in increasing order
1 2 5 6 8
Enter maximum limit for sum
9
The subsets with sum of elements = 9 are
{1,2,6}
{1,8}
```

8. LEARNING OUTCOMES:

- Students are able to implement subset sum problem using backtracking algorithm.

9. APPLICATION AREAS:
- In the field of computer network to find optimal solution

10. REMARKS:

-

1. EXPERIMENT NO: **12**

2. TITLE: **HAMILTONIAN CYCLE**

3. LEARNING OBJECTIVES:
- To understand the concept of hamiltonian cycle
- To implement an algorithm to check for the existance of hamiltonian cycle in the given undirected graph.

4. AIM:Design and implement the presence of Hamiltonian Cycle in an undirected Graph G of n vertices.

5. THEORY / HYPOTHESIS:

A Hamiltonian cycle, also called a Hamiltonian circuit, Hamilton cycle, or Hamilton circuit, is a graph cycle (i.e., closed loop) through a graph that visits each node exactly once. A graph possessing a Hamiltonian cycle is said to be a Hamiltonian graph. By convention, the singleton graph K1 is considered to be Hamiltonian even though it does not posses a Hamiltonian cycle, while the connected graph on two nodes K2 is not.The Hamiltonian cycle is named after Sir William Rowan Hamilton, who devised a puzzle in which such a path along the polyhedron edges of an dodecahedron was sought (the Icosian game). Determining whether a directed graph has a hamiltonian cycle is NP-complete.

```
Algorithm Hamiltonian(k)
{
 Loop
   Next value (k)
        If (x (k)=0) then return;
        {
                If k=n then     Print (x)
                Else            Hamiltonian (k+1);
                End if
        }
 Repeat
}

Algorithm Nextvalue (k)
{
        Repeat
        {
                X [k]=(X [k]+1) mod (n+1); //next vertex
                If (X [k]=0) then return;
                If (G [X [k-1], X [k]]  0) then
                {
                        For j=1 to k-1 do if (X [j]=X [k]) then break;
                        // Check for distinction.
                        If (j=k) then         //if true then the vertex is distinct.
                        If ((k<n) or ((k=n) and G [X [n], X [1]]   0)) then return;
```

```
                }
        } Until (false);
}
```

6. PROCEDURE / PROGRAMME / ACTIVITY:

```
package hamiltoniancycle;
import java.util.Scanner;
import java.util.Arrays;

public class HamiltonianCycle()    /** 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");
      /** all vertices selected but last vertex not linked to 0 **/
      if (pathCount == V)
         return;

      for (int v = 0; v < V; v++)
      {
         /** if connected **/
         if (graph[vertex][v] == 1 )
         {
```

```
            /** add to path **/
            path[pathCount++] = v;
            /** remove connection **/
            graph[vertex][v] = 0;
            graph[v][vertex] = 0;

            /** if vertex not already selected  solve recursively **/
            if (!isPresent(v))
               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] +" ");
      System.out.println();
   }
   public static void main (String[] args)    /** Main function **/
   {
      Scanner sc = new Scanner(System.in);

      HamiltonianCycle hc = new HamiltonianCycle();

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

      hc.findHamiltonianCycle(graph);
   }
```

```
}
```

7. RESULTS & CONCLUSIONS:

Output-1

```
Enter number of vertices
5
Enter adjacency matrix
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 1
0 1 1 1 0

Solution found
Path : 0 1 2 4 3 0
```

Output-2

```
Enter number of vertices
5
Enter adjacency matrix
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 0
0 1 1 0 0
No solution
```

8. LEARNING OUTCOMES:

- On implementation of this programme the student will know what the hamiltonian cycle is and able to implement the same in java

9. APPLICATION AREAS:

- Stripification of triangle meshes in computer graphics
- Puzzle games

10. REMARKS: