VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi - 590 018, Karnataka

RNSInstituteofTechnology

Dr. Vishnuvardana Road, Channasandra

Bengaluru – 560098

DEPARTMENT

Of

COMPUTER SCIENCE & ENGINEERING

LABORATORY MANUAL

Design and Analysis of Algorithms Laboratory

15CSL47

IV Semester

Course objectives: This course will enable students to achieve the following:

- Design and implement various algorithms in JAVA ·
- Employ various design strategies for problem solving.
- Measure and compare the performance of different algorithms.

Course outcomes: At the end of the course, students should be able to:

1. Develop solutions for computing problems using JAVA constructs.
2. Design, implement and asymptotically analyze various algorithms on Sorting.
3. Design and implement algorithms to solve problems on Graphs.
4. Design and develop algorithms to solve combinatorial problems.
5. Choose appropriate algorithmic techniques to solve computational problems.
6. Analyze algorithms to deduce their time complexities.

CO-PO MAPPING MATRIX

SUBJECT TITLE:Design and Analysis of Algorithms Laboratory						SUBJECT CODE:15CSL47						
CO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	3	2	2	1	1	-	-	1	2	2	-	3
CO2	3	3	3	1	1	-	-	-	2	2	-	3
CO3	3	3	3	1	1	-	-	-	2	2	-	3
CO4	3	3	3	1	1	-	-	-	2	2	-	3
CO5	3	2	3	2	1	1	-	-	3	1	1	3
CO6	3	3	1	1	-	-	-	-	-	-	-	2

CO-PSO MAPPING MATRIX

CO	PSO1	PSO2	PSO3	PSO4
CO1	3	1	3	-
CO2	3	1	2	-
CO3	3	1	2	-
CO4	3	1	2	-
CO5	2	1	2	-
CO6	2	-	2	-

DESIGN & ANALYSIS OF ALGORITHMS LABORATORY

Subject Code: 15CSL47	IA Marks: 20
Hours/Week : 01I + 02P	Exam Marks: 80
Total Hours: 40	Exam Hours: 03

Lab Experiments:

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

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

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

3 A. Write a Java program to read two integers a andb. Compute a/b and print, when b is not zero. Raise an exception when b is equal to zero.
B. Write a Java program that implements a multi-thread application that 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 non 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.

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 non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and-conquer method works along with its time complexity analysis: worst case, average case and best case.

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

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

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

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

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

11 Design and implement in Java to find a subset of a given set S = {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.

Evaluation Rubrics for lab Programs(Max marks 10)

Lab write-up and execution rubrics(Max marks 3+5)

		Good	**Average**
a.	**Write up (3 marks)**	Write the program neatly demonstrating good knowledge of concepts and algorithms in java (3)	Moderate understanding of algorithms(2)
b.	**Execution (3 marks)**	Program handles all possible conditions (3)	Average conditions defined (2)
c.	**Result (2 marks)**	All results are correctly demonstrated(2)	Acceptable documentation shown(1)

VIVA Rubrics: (Max marks 2)

		Good	**Average**
	Conceptual understanding (2 marks)	Explain the complete program with the related concepts (2)	explanation provided adequately (1)

DESIGN & ANALYSIS OF ALGORITHMS LABORATORY

Execution plan

WEEK #	NAME OF THE EXPERIMENT	EXPERIMENT #	CO	REMARKS
1	Create n Student objects and print the USN, Name, Branch, and Phone of these objects & implement the Stack using arrays	1	CO1	Completed
2	Implementing super class to extend another class & String tokenizer	2	CO1	Completed
3	Implement exception and implement multithread application	3	CO1	Completed
4	Quick sort	4	CO2,CO6	Completed
5	**Lab Internal-I**			
6	Merge sort	5	CO2,CO6	Completed
7	Knapsack problem a)Greedy Approach b)Dynamic programming Approach	6	CO4	Completed
8	Implement Dijikstra's algorithm	7	CO3	Completed
9	Implement Kruskal's algorithm	8	CO3,CO6	Completed
10	Implement Prim's algorithm.	9	CO3,CO6	Completed
11	Implement All-Pairs Shortest Paths problem using (a) Floyd's algorithm. (b) Implement Travelling Sales Person problem using Dynamic programming.	10	CO5,CO6	Completed

12	1)Implementing subset of a given set S = {Sl, S2,.....,Sn}	11	CO5	Completed
	2) Implementing Hamiltonian Cycles	12	CO5	Completed
13	**Lab Internal-II**			

DESIGN & ANALYSIS OF ALGORITHMS LABORATORY

SL NO	EXPERIMENT NO	DATE OF EXECUTION	NAME OF THE EXPERIMENT	MARKS
1	1		Create n Student objects and print the USN, Name, Branch, and Phone of these objects & implement the Stack using arrays	10
2	2		Implementing super class to extend another class & String tokenizer	10
3	3		Implement exception and implement multithread application	10
4	4		Quick sort	10
5	5		Merge sort	10
6	6		Knapsack problem a)Greedy Approach b)Dynamic programming Approach	10
7	7		Implement Dijikstra's algorithm	10
8	8		Implement Kruskal's algorithm	10
9	9		Implement Prim's algorithm.	10
10	10		Implement All-Pairs Shortest Paths problem using (a) Floyd's algorithm. (b) Implement Travelling Sales Person problem using Dynamic programming.	10
11	11		Implementing subset of a given set S = {Sl, S2,.....,Sn}	10
12	12		Implementing Hamiltonian Cycles	10
				Avg:

FINAL IA MARKS

	Max marks	Marks obtained
Average of Daily execution(A)	**10**	
TEST-1(T1)	**10**	
TEST-2(T2)	**10**	
Average(A+(T1+T2)/2)	**20**	

Staff signature:

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

import java.util.Scanner;

class Student {
    private String usn, name, branch, ph;

    public void accept() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Usn\n");
        usn = scanner.next();

        System.out.println("Enter Name\n");
        name = scanner.next();

        System.out.println("Enter Branch\n");
        branch = scanner.next();

        System.out.println("Enter Phone\n");
        ph = scanner.next();
    }

    public void display() {
        System.out.println("Student details");
        System.out.println("Usn: " + usn);
        System.out.println("Name: " + name);
        System.out.println("Branch: " + branch);
        System.out.println("Phone: " + ph);
        System.out.println();
    }
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter value for n");
        int n = scanner.nextInt();
        Student[] ob1 = new Student[n];

        for (int i = 0; i < n; i++) {
            ob1[i] = new Student();
            ob1[i].accept();
        }
        for (int i = 0; i < n; i++) {
            ob1[i].display();
        }
    }
}


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

import java.util.Scanner;

class Stack {
    private int arr[], top, size;

    public Stack(int size) {
        this.size = size;
        arr = new int[size];
        top = -1;
    }

    public void push(int i) {
        if (top == size - 1) {
            System.out.println("Stack is full");
            return;
        }
        arr[++top] = i;
    }

    public void pop() {
        if (top == -1) {
            System.out.println("Stack is empty");
            return;
        }
        System.out.println("Element Popped is: " + arr[top--]);
    }

    public void display() {
        if (top == -1) {
            System.out.print("Empty\n");
            return;
        }
        System.out.print("\nStack: ");
        for (int i = top; i >= 0; i--)
            System.out.print(arr[i] + " ");
    }
}

class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Size of Integer Stack ");
        int n = scanner.nextInt();

        Stack stack = new Stack(n);

        while (true) {
            System.out.println("\nStack Operations");
            System.out.println("1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Display");
            System.out.println("4. Exit");
            int choice = scanner.nextInt();

            switch (choice) {
            case 1:
                System.out.println("Enter integer element to push");
                stack.push(scanner.nextInt());
                break;

            case 2:
                stack.pop();
                break;

            case 3:
                stack.display();
                break;

            case 4:
                System.exit(0);
            }
        }
    }
}
```

```
/*
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.
*/

import java.util.Scanner;

class Staff {
    private String staffId;
    private String name;
    private int phone;
    private float salary;

    public void accept() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Staff Id");
        staffId = scanner.next();
        System.out.println("Enter Name");
        name = scanner.next();
        System.out.println("Enter Phone");
        phone = scanner.nextInt();
        System.out.println("Enter Salary");
        salary = scanner.nextFloat();
    }

    public void display() {
        System.out.println("Staff Id: " + staffId);
        System.out.println("Name: " + name);
        System.out.println("Phone: " + phone);
        System.out.println("Salary: " + salary);
    }
}


class Teaching extends Staff {
    private String domain;
    private String[] publications;

    public void accept() {
        super.accept();
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Domain");
        domain = scanner.next();
        System.out.println("Enter Number of Publications");
        int n = scanner.nextInt();
        publications = new String[n];
        System.out.println("Enter Publications");
        for (int i = 0; i < n; i++) {
            publications[i] = scanner.next();
        }
    }

    public void display() {
        super.display();
        System.out.println("Domain: " + domain);
        System.out.println("Publications:");
        for (int i = 0; i < publications.length; i++) {
            System.out.println(publications[i]);
        }
    }
}


class Technical extends Staff {
    private String[] skills;

    public void accept() {
        super.accept();
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Number of Skills");
        int n = scanner.nextInt();
        skills = new String[n];
        System.out.println("Enter Skills");
        for (int i = 0; i < n; i++) {
            skills[i] = scanner.next();
        }
    }

    public void display() {
        super.display();
        System.out.println("Skills:");
        for (int i = 0; i < skills.length; i++) {
            System.out.println(skills[i]);
        }
    }
}


class Contract extends Staff {
    private int period;

    public void accept() {
        super.accept();
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Period");
        period = scanner.nextInt();
    }

    public void display() {
        super.display();
        System.out.println("Period: " + period);
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Teaching teaching = new Teaching();
        Technical technical = new Technical();
        Contract contract = new Contract();

        System.out.println("Enter Details for Teaching Staff Member");
        teaching.accept();
        System.out.println("Enter Details for Technical Staff
Member");
        technical.accept();
        System.out.println("Enter Details for Contract Staff Member");
        contract.accept();

        System.out.println("Details for Teaching Staff Member are");
        teaching.display();
        System.out.println("Details for Technical Staff Member are");
        technical.display();
        System.out.println("Details for Contract Staff Member are");
        contract.display();
    }
}
```

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

import java.util.NoSuchElementException;
import java.util.Scanner;
import java.util.StringJoiner;
import java.util.StringTokenizer;

class Customer {
    String name;
    String dob;

    void accept() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Name and DoB");
        String input = scanner.nextLine();
        StringTokenizer stringTokenizer = new StringTokenizer(input,
",");
        try {
            name = stringTokenizer.nextToken().trim();
            dob = stringTokenizer.nextToken().trim();
        } catch (NoSuchElementException e) {
            System.out.println("Invalid Format");
            System.exit(0);
        }
    }

    void display() {
        StringTokenizer stringTokenizer = new StringTokenizer(dob,
"/");
        try {
            String dd = stringTokenizer.nextToken().trim();
            String mm = stringTokenizer.nextToken().trim();
            String yy = stringTokenizer.nextToken().trim();

            StringJoiner stringJoiner = new StringJoiner(", ");
            stringJoiner.add(name).add(dd).add(mm).add(yy);

            System.out.println(stringJoiner.toString());
        } catch (NoSuchElementException e) {
            System.out.println("Invalid Format");
            System.exit(0);
        }
    }
}

class Main {
    static Customer customer;
    public static void main(String[] args) {
        customer = new Customer();
        customer.accept();
        customer.display();
    }
}
```

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

import java.lang.Exception;
import java.util.Scanner;

class Main {
    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter two numbers");
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        if (b != 0) {
            System.out.println("a/b = " + (((float) a) / b));
        } else {
            throw new Exception("b is zero");
        }
    }
}
```

```
/*
```

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

```
import java.util.Random;

class Generator extends Thread {
    static int number;

    @Override
    public void run() {
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            number = random.nextInt(10) + 1;
            System.out.println("\nRandom Number: " + number);

            Main.square.interrupt();
            Main.cube.interrupt();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Square extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(99999);
            } catch (InterruptedException e) {
                System.out.println("Square: " + (Generator.number *
Generator.number));
            }
        }
    }
}

class Cube extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(99999);
            } catch (InterruptedException e) {
                System.out.println("Cube: " + (Generator.number *
Generator.number * Generator.number));
            }
        }
    }
}

class Main {
    static Thread generator;
    static Thread square;
    static Thread cube;

    public static void main(String[] args) {
        generator = new Generator();
        square = new Square();
        cube = new Cube();

        square.start();
        cube.start();
        generator.start();
    }
}
```

```
/*
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 non graph sheet. The elements
can be read from a file or can be generated using the random number
generator. Demonstrate using Java how the divide-and-conquer method
works along with its time complexity analysis: worst case, average
case and best case.
*/

import java.util.Random;
import java.util.Scanner;

class QuickSort {
    static int comparisons = 0;

    static int[] arr;

    static void quickSort(int low, int high) {
        if (low < high) {
            comparisons += 1;

            int j = partition(low, high);
            quickSort(low, j - 1);
            quickSort(j + 1, high);
        }
    }

    static int partition(int low, int high) {
        int pivot = arr[low];
        int i = low, j = high;
        while (i < j) {
            comparisons += 1;

            while (i < high && arr[i] <= pivot) {
                comparisons += 2;

                i = i + 1;
            }

            while (j > low && arr[j] >= pivot) {
                comparisons += 2;

                j = j - 1;
            }
            if (i < j) {
                comparisons += 1;

                interchange(i, j);
            }
        }
        arr[low] = arr[j];
        arr[j] = pivot;
        return j;
    }

    static void interchange(int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public static void main(String[] args) {
        int n;
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter n value");
        n = scanner.nextInt();
        arr = new int[n];

        System.out.println("Quick Sort");
        System.out.println("1. Best/Average Case");
        System.out.println("2. Worst Case");

        int ch = scanner.nextInt();
        switch (ch) {
        case 1:
            Random random = new Random();
            for (int i = 0; i < n; i++) {
                arr[i] = random.nextInt();
            }
            break;
        case 2:
            for (int i = 0; i < n; i++) {
                arr[i] = i + 1;
            }
            break;
        }

        long start = System.nanoTime();
        quickSort(0, n - 1);
        long end = System.nanoTime();

        System.out.println("Sorted Array");
        for (int i = 0; i < n; i++) {
            System.out.println(arr[i]);
        }

        System.out.println("Time taken: " + (end - start));
        System.out.println("Comparisons: " + comparisons);
    }
}
```

```
/*
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 non 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.
*/

import java.util.Random;
import java.util.Scanner;

class MergeSort {
    static int comparisons = 0;

    static int[] arr;

    static void mergeSort(int low, int high) {
        if (low < high) {
            comparisons += 1;

            int mid = (low + high) / 2;
            mergeSort(low, mid);
            mergeSort(mid + 1, high);
            merge(low, mid, high);
        }
    }

    static void merge(int low, int mid, int high) {
        int n = high - low + 1;
        int[] t_arr = new int[n];
        int i = low, j = mid + 1, k = 0;
        while ((i <= mid) && (j <= high)) {
            comparisons += 2;

            if (arr[i] <= arr[j]) {
                comparisons += 1;

                t_arr[k] = arr[i];
                i++;
            } else {
                comparisons += 1;

                t_arr[k] = arr[j];
                j++;
            }
            k++;
        }
        while (i <= mid) {
            // Put remaining elements from left subpart (if any) to
temporary array.
            comparisons += 1;
            t_arr[k] = arr[i];
            i++;
            k++;
        }
        while (j <= high) {
            // Put remaining elements from right subpart (if any) to
temporary array.
            comparisons += 1;

            t_arr[k] = arr[j];
            j++;
            k++;
        }
        for (k = 0; k < n; k++) {
            // Copy Elements from temporary array to original array.
            comparisons += 1;

            arr[low + k] = t_arr[k];
        }
    }

    public static void main(String[] args) {
        int n;
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter n value");
        n = scanner.nextInt();
        arr = new int[n];
        int ch;
        System.out.println("Merge Sort");
        System.out.println("1. Best Case");
        System.out.println("2. Average Case");
        System.out.println("3. Worst Case");
        ch = scanner.nextInt();
        switch (ch) {
        case 1:
            for (int i = 0; i < n; i++) {
                arr[i] = i + 1;
            }
            break;
        case 2:
            Random random = new Random();
            for (int i = 0; i < n; i++) {
                arr[i] = random.nextInt();
            }
            break;
        case 3:
            for (int i = 0; i < n; i++) {
                arr[i] = n - i;
            }

            /*
                As per VTU textbooks descending order of elements is
worst case
                for the Merge Sort, which is not TRUE.
                For the actual worst case, uncomment the rest of the
code and delete
                the above 'for' loop.
            */

            // for (int i = 0; i < n; i++) {
            //     arr[i] = i + 1;
            // }
            // generateWorstCase(0, n - 1);

            break;
        }

        long start = System.nanoTime();
        mergeSort(0, n - 1);
        long end = System.nanoTime();
        System.out.println("Sorted Array");

        // Print the time taken to sort.
        System.out.println("Time Taken: " + (end - start));
        // Print the number of comparisons.
        System.out.println("Comparisons: " + comparisons);
    }

    // static void generateWorstCase(int low, int high) {
    //     if (low < high) {
    //         int mid = (low + high) / 2;
    //         partition(low, high);
    //         generateWorstCase(low, mid);
    //         generateWorstCase(mid + 1, high);
    //     }
    // }

    // static void partition(int low, int high) {
    //     int n = high - low + 1;
    //     int k = 0;
    //     int t_arr[] = new int[n];

    //     for (int i = low; i <= high; i += 2) {
    //         // Read elements at odd positions w.r.t. low.
    //         t_arr[k] = arr[i];
    //         k++;

    //     }

    //     for (int i = low + 1; i <= high; i += 2) {
    //         // Read elements at even positions w.r.t. low.
    //         t_arr[k] = arr[i];
    //         k++;
    //     }

    //     for (int i = 0; i < n; i++) {
    //         // Copy elements from temporary array to the original
array.
    //         arr[low + i] = t_arr[i];
```

```
       //        }
       //  }

}
```

```
/*
6.a. Implement in Java, the 0/1 Knapsack problem using Dynamic
Programming method.
*/

import java.util.Scanner;

class Knapsack {
    private int[] weight, profit;
    private int capacity, n;

    Knapsack() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Number of Items");
        n = scanner.nextInt();
        weight = new int[n];
        profit = new int[n];

        System.out.println("Enter Weights of Items");
        for (int i = 0; i < n; i++) {
            weight[i] = scanner.nextInt();
        }

        System.out.println("Enter Profits of Items");
        for (int i = 0; i < n; i++) {
            profit[i] = scanner.nextInt();
        }

        System.out.println("Enter Capacity of Knapsack");
        capacity = scanner.nextInt();
    }

    void fill() {
        int[][] K = new int[n + 1][capacity + 1];

        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= capacity; j++) {
                if (i == 0 || j == 0) {
                    K[i][j] = 0;
                } else if (j < weight[i - 1]) {
                    K[i][j] = K[i - 1][j];
                } else {
                    K[i][j] = Math.max(K[i - 1][j], profit[i - 1] +
K[i - 1][j - weight[i - 1]]);
                }
            }
        }
        System.out.println("Maximum Profit: " + (K[n][capacity]));

        System.out.print("Items Considered: ");

        int i = n, j = capacity;
```

```
            while (i > 0 && j > 0) {
                if (K[i][j] != K[i - 1][j]) {
                    System.out.print(i + " ");
                    j -= weight[i - 1];
                }
                i -= 1;
            }
            System.out.println();
        }

        public static void main(String[] args) {
            Knapsack knapsack = new Knapsack();
            knapsack.fill();
        }
    }
```

```java
/*
6.b. Implement in Java, the  Knapsack problem using Greedy method.
*/

import java.util.Scanner;

class Knapsack {

    private double[] weight, profit, ratio, solnVector;
    private double capacity;

    Knapsack() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter number of Items");
        int n = scanner.nextInt();
        weight = new double[n];
        profit = new double[n];
        ratio = new double[n];
        solnVector = new double[n];

        System.out.println("Enter Weights of Items");
        for (int i = 0; i < n; i++) {
            weight[i] = scanner.nextDouble();
        }

        System.out.println("Enter Profits of Items");
        for (int i = 0; i < n; i++) {
            profit[i] = scanner.nextDouble();
        }

        System.out.println("Enter Capacity of Knapsack");
        capacity = scanner.nextDouble();
    }

    int getNext() {
        int index = -1;
        double highest = 0;

        for (int i = 0; i < ratio.length; i++) {
            if (ratio[i] > highest) {
                highest = ratio[i];
                index = i;
            }
        }
        return index;
    }

    void fill() {
        double currentWeight = 0;
        double currentProfit = 0;

        for (int i = 0; i < ratio.length; i++) {
            ratio[i] = profit[i] / weight[i];
        }
        System.out.print("Items Considered: ");
        while (currentWeight < capacity) {
            int item = getNext();

            if (item == -1) {
                break;
            }

            System.out.print((item + 1) + " ");

            if (currentWeight + weight[item] <= capacity) {
                currentWeight += weight[item];
                currentProfit += profit[item];
                solnVector[item] = 1;
                ratio[item] = 0;
            } else {
                currentProfit += ratio[item] * (capacity -
currentWeight);
                solnVector[item] = (capacity - currentWeight) /
weight[item];
                break;
            }
        }

        System.out.println();
        System.out.println("Maximum Profit is: " + currentProfit);
        System.out.print("Solution Vector: ");
        for (int i = 0; i < solnVector.length; i++) {
            System.out.print(solnVector[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Knapsack knapsack = new Knapsack();
        knapsack.fill();
    }
}
```

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

import java.util.Scanner;

class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Number of Vertices");
        int n = scanner.nextInt();

        int adj[][] = new int[n][n];

        System.out.println("Enter Adjacency Matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                adj[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter Source vertex");
        int src = scanner.nextInt();

        int[] dist = dijkstra(adj, src);

        for (int i = 0; i < n; i++) {
            if ((src - 1) == i) {
                continue;
            }
            System.out.println("Shortest Distance from " + src + " to
" + (i + 1) + " is " + dist[i]);
        }
    }

    static int[] dijkstra(int adj[][], int src) {
        int n = adj.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        int min_dist, unvis = -1;

        for (int i = 0; i < n; i++) {
            dist[i] = adj[src - 1][i];
            visited[i] = false;
        }

        visited[src - 1] = true;

        for (int i = 1; i < n; i++) {
            min_dist = Integer.MAX_VALUE;
            for (int j = 0; j < n; j++) {
                if (!visited[j] && dist[j] < min_dist) {
                    unvis = j;
                    min_dist = dist[j];
                }
            }

            visited[unvis] = true;

            for (int v = 0; v < n; v++) {
                if (!visited[v] && dist[unvis] + adj[unvis][v] <
dist[v]) {
                    dist[v] = dist[unvis] + adj[unvis][v];
                }
            }
        }

        return dist;
    }
}
```

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

import java.util.Arrays;
import java.util.Scanner;

class Edge {
    int src;
    int dest;
    int weight;

    Edge(int src, int dest, int weight) {
        this.src = src;
        this.dest = dest;
        this.weight = weight;
    }
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter number of Vertices");
        int n = scanner.nextInt();

        int adj[][] = new int[n][n];
        System.out.println("Enter Adjacency Matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                adj[i][j] = scanner.nextInt();
            }
        }

        // Maximum Edges without any Loops can be ((n * (n - 1)) / 2).
        Edge[] edges = new Edge[(n * (n - 1)) / 2];

        int k = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                edges[k] = new Edge(i, j, adj[i][j]);
                k++;
            }
        }

        sort(edges);

        // Declare an array of size vertices to keep track of
respective leaders of each element.
        int[] parent = new int[n];
        // Assign each element of array of with value -1.
        Arrays.fill(parent, -1);

        int minCost = 0;
        System.out.println("Edges: ");
        for (int i = 0; i < k; i++) {
            // Find the super most of leader of source vertex.
            int lsrc = find(parent, edges[i].src);
            // Find the super most of leader of destination vertex.
            int ldest = find(parent, edges[i].dest);

            // If those two leaders are different then they belong to
isolated groups.
            if (lsrc != ldest) {
                System.out.println((edges[i].src + 1) + " <-> " +
(edges[i].dest + 1));
                minCost += edges[i].weight;
                union(parent, lsrc, ldest);
            }
        }

        System.out.println();
        System.out.println("Minimum Cost of Spanning Tree: " +
minCost);
    }

    static void sort(Edge[] edges) {
        // Sort Edges according to their weights using Bubble Sort.
        for (int i = 1; i < edges.length; i++) {
            for (int j = 0; j < edges.length - i; j++) {
                if (edges[j].weight > edges[j + 1].weight) {
                    Edge temp = edges[j];
                    edges[j] = edges[j + 1];
                    edges[j + 1] = temp;
                }
            }
        }
    }

    static int find(int[] parent, int i) {
        if (parent[i] == -1) {
            // Super Most Leader Element Found.
            return i;
        }
        // Find Above Leader in recurrsive manner.
        return find(parent, parent[i]);
    }

    static void union(int[] parent, int lsrc, int ldest) {
        // Make destination vertex leader of source vertex.
        parent[lsrc] = ldest;
    }
}
```

```
/*
9. Find Minimum Cost Spanning Tree of a given connected undirected
graph using Prim's algorithm.
*/
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Number of Vertices");
        int n = scanner.nextInt();
        int[][] costMatrix = new int[n][n];
        boolean[] visited = new boolean[n];

        System.out.println("Enter Cost Adjacency Matrix");
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                costMatrix[i][j] = scanner.nextInt();

        for (int i = 0; i < n; i++)
            visited[i] = false;

        System.out.println("Enter Source Vertex");
        int srcVertex = scanner.nextInt();

        visited[srcVertex - 1] = true;

        int source = 0, cost = 0, target = 0;
        System.out.print("Edges: ");

        for (int i = 1; i < n; i++) {
            int min = Integer.MAX_VALUE;

            for (int j = 0; j < n; j++) {
                if (visited[j]) {

                    for (int k = 0; k < n; k++) {
                        if (!visited[k] && min > costMatrix[j][k]) {
                            min = costMatrix[j][k];
                            source = j;
                            target = k;
                        }
                    }
                }
            }
            visited[target] = true;
            System.out.print("(" + (source + 1) + "," + (target + 1) +
")");
            cost += min;
        }
        System.out.println("\nMinimum cost of Spanning Tree: " +
cost);
    }
}
```

```
/*
10.a. Write Java programs to Implement All-Pairs Shortest Paths
problem using Floyd's algorithm.
*/

import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Number of Vertices");
        int n = scanner.nextInt();
        int[][][] D = new int[n + 1][n][n];

        System.out.println("Enter Distance Matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                D[0][i][j] = scanner.nextInt();
            }
        }

        for (int k = 1; k <= n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    D[k][i][j] = Math.min(D[k - 1][i][j], D[k -
1][i][k - 1] + D[k - 1][k - 1][j]);
                }
            }
        }

        System.out.println("Shortest Distance Matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(D[n][i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

/*
10.b. Write Java programs to Implement Travelling Sales Person problem using Dynamic programming.
***/**

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

class Main {
    static int[][] graph;
    static int n, src;

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

        graph = new int[n][n];

        System.out.println("Enter Adjacency Matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                graph[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter Source City");
        src = scanner.nextInt();

        ArrayList<Integer> set = new ArrayList<Integer>();
        for (int i = 0; i < n; i++) {
            if (i == (src - 1)) {
                continue;
            }
            set.add(i);
        }

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

        int cost = tsp(src - 1, set, path);
        System.out.println("Total Cost: " + cost);

        path[0] = src - 1;
        path[n] = src - 1;
        System.out.print("Path: ");
        for (int i = n; i >= 0; i--) {
            System.out.print((path[i] + 1) + " ");
        }
        System.out.println();
    }

    static int tsp(int v, ArrayList<Integer> set, int[] path) {
        if (set.isEmpty()) {
            return graph[v][src - 1];
        }
        int size = set.size();
        ArrayList<Integer> subSet;
        int minCost = Integer.MAX_VALUE;
        for (Integer i : set) {
            subSet = new ArrayList<Integer>(set);
            subSet.remove(i);
            int[] tempPath = new int[n+1];
            int cost = graph[v][i] + tsp(i, subSet, tempPath);

            if (cost < minCost) {
                minCost = cost;
                tempPath[size] = i;
                copyCentralArray(path, tempPath, size);
            }
        }
        return minCost;
    }

    static void copyCentralArray(int[] dest, int[] src, int size) {
        for (int i = 1; i <= size; i++) {
            dest[i] = src[i];
        }
    }
}

// ///////
// Output:
//
// Enter number of cities
// 4
// Enter Adjacency Matrix
// 0 10 15 20
// 5 0 9 10
// 6 13 0 12
// 8 8 9 0
// Enter Source City
// 1
// Total Cost: 35
// Path: 1 2 4 3 1
```

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

import java.util.Scanner;

class Main {
    static int[] arr;
    static int count;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter n value");
        int n = scanner.nextInt();
        arr = new int[n];

        System.out.println("Enter Elements of Set");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Enter Total Sum value");
        int total = scanner.nextInt();

        subSet(total, n - 1, new boolean[n]);

        if (count == 0) {
            System.out.println("No solution");
        }
    }

    static void subSet(int total, int index, boolean[] solution) {
        if (total == 0) {
            printSolution(solution);
        } else if (total < 0 || index < 0) {
            return;
        } else {
            boolean[] tempSolution = solution.clone();
            tempSolution[index] = false;
            subSet(total, index - 1, tempSolution);
            tempSolution[index] = true;
            subSet(total - arr[index], index - 1, tempSolution);
        }
    }

    static void printSolution(boolean[] solution) {
        count += 1;
        System.out.print("Solution: ");
        for (int i = 0; i < solution.length; i++) {
            if (solution[i]) {
                System.out.print(arr[i] + " ");
            }
        }
        System.out.println();
    }
}

// ///////
// Output:
//
// Enter n value
// 5
// Enter Elements of Set in Increasing Order
// 1 2 5 6 8
// Enter Total Sum value
// 9
// Solution: 1 2 6
// Solution: 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;

class Hamiltonian {
    static int[][] graph;
    static int[] soln;
    static int n, count = 0;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Number of Vertices");
        n = scanner.nextInt();

        // Read Adjacency Matrix in Graph array(1 Indexed)
        graph = new int[n + 1][n + 1];
        System.out.println("Enter Adjacency Matrix");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                graph[i][j] = scanner.nextInt();
            }
        }

        // Instatiate Solution array(1 Indexed), (Default Value is 0)
        soln = new int[n + 1];

        System.out.println("Hamiltonian Cycle are");

        // In a cycle source vertex doesn't matter
        // Assign Starting Point to prevent repetitions
        soln[1] = 1;
        // Call Hamiltonian function to start backtracking from vertex
2
        hamiltonian(2);

        if (count == 0) {
            System.out.println("No Hamiltonian Cycle");
        }
    }

    static void hamiltonian(int k) {
        while (true) {
            nextValue(k);

            // No next vertex so return
            if (soln[k] == 0) {
                return;
            }
            // if cycle is complete then print it else find next
vertex
            if (k == n) {
                printArray();
            } else {
                hamiltonian(k + 1);
            }
        }
    }

    static void nextValue(int k) {
        // Finds next feasible value
        while (true) {

            soln[k] = (soln[k] + 1) % (n + 1);

            // If no next vertex is left, then return
            if (soln[k] == 0) {
                return;
            }

            // If there exists an edge
            if (graph[soln[k - 1]][soln[k]] != 0) {
                int j;
                // Check if the vertex is not repeated
                for (j = 1; j < k; j++) {
                    if (soln[j] == soln[k]) {
                        break;
                    }
                }

                // If vertex is not repeated
                if (j == k) {
                    // If the vertex is not the last vertex or it
completes the cycle then return
                    if (k < n || (k == n && graph[soln[n]][soln[1]] !=
0)) {
                        return;
                    }
                }
            }
        }
    }

    static void printArray() {
        count += 1;
        // Print Solution Array
        for (int i = 1; i <= n; i++) {
            System.out.print(soln[i] + " ");
        }
        System.out.println(soln[1]);
    }
}
```