DESIGN AND ANALYSIS OF ALGORITHM LABORATORY

DAA LAB MANUAL

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2017 -2018)

SEMESTER – IV

Subject Code: 17CSL47 IA Marks: 40

Exam Marks: 60

Description

Design, develop & implement the specified algorithms for the following problem using Java Language under LINUX / Windows environment. NetBeans / Eclipse IDE tool can be used for development & Demonstration

---

Experiments

---

1 A: Create a Java class called Student with the following details as variables within it.

- **USN**
- **Name**
- **Branch**
- **Phone**

Write a Java program to create n Student objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.

Program Code

```
import java.util.Scanner;

public class Room {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n,i;
        Student[] s=new Student[100];
        System.out.println("Enter the Number of
Objects");
        n=scan.nextInt();
        for(i=0;i<n;i++)
        {
            s[i]=new Student();
        }
        for(i=0;i<n;i++)
        {
            System.out.println("Enter the Details
in Order - Name , Usn , Branch , Phone No");
            String name = scan.next();
            String usn = scan.next();
            String branch = scan.next();
            String phno = scan.next();
            s[i].assignvalue(name, usn, branch,
phno);
        }
        System.out.println("Students Details are as
Follow:");
        for(i=0;i<n;i++)
```

```
        {
            s[i].display();
        }
        scan.close();
    }
}
class Student
{
    String name,usn,branch,phno;
    void assignvalue(String n,String u,String
b,String p)
    {
        name=n;
        usn=u;
        branch=b;
        phno=p;
    }
    void display()
    {
        System.out.println("Name :"+name+" USN
:"+usn+" Branch :"+branch+"Phone Number :"+phno);
    }
}
```

Expected Output

```
Enter the Number of Objects
3
Enter the Details in Order - Name , Usn , Branch , Phone No
Hemanth
1by18cs404
cse
9632580000
Enter the Details in Order - Name , Usn , Branch , Phone No
Raj
1by18cs413
cse
9632587410
Enter the Details in Order - Name , Usn , Branch , Phone No
punith
1by18cs425
cse
9632580145
Students Details are as Follow:
Name :Hemanth USN :1by18cs404 Branch :cse Phone Number :9632580000
Name :Raj USN :1by18cs413 Branch :cse Phone Number :9632587410
Name :punith USN :1by18cs425 Branch :cse Phone Number :9632580145
```

---

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

Program Code

```java
//package oneB;
import java.util.Scanner;
class arraystack
{
    int a[],top,max,i;
    arraystack(int n) //using constructor
    {
        max=n;
        a=new int[max];
        top=-1;
    }
    void push(int ele)
    {
        if(top==max-1)
```

```
        {
            System.out.println("----Stack OVER FLOW
- Elements in the array are :"+max+" ----");
        }
        else
        {
            a[++top]=ele;

        }
    }
    void pop()
    {
        if(top==-1)
        {
            System.out.println("----Stack
Underflow----");
        }
        else
        {
            System.out.println("Poped Element is
:"+a[top--]);
        }
    }
    void display()
    {
        if(top==-1)
        {
            System.out.println("----STACK UNDERFLOW
- No Element to display----");
            return;
        }
        System.out.println("Elements in the stack
are as follows :");
        int p=top;
        for(i=p;i>=0;i--)
        {
            System.out.println("ELEMENT :"+a[i]);
```

```
        }
    }
}
public class Stack {
    public static void main(String[] args) {
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter the Size of the
array");
        int n=scan.nextInt();
        boolean done=false;
        arraystack stk =new arraystack(n);
        do
        {
            System.out.println("Stack Operation ");

    System.out.println("1.Push\n2.Pop\n3.Display\n4
.Exit");
            System.out.println("Enter Your
Choice");
            int choice=scan.nextInt();
            switch(choice)
            {
            case 1:System.out.println("Enter the
Elements to be instered");
                   stk.push(scan.nextInt());
                   break;
            case 2:stk.pop();
                   break;
            case 3:stk.display();
                   break;
            case 4:done=true;
                   break;
            default : System.out.println("You Have
Entered the Wrong Choice");
                     break;
            }
        }
```

```
        while(!done);
        scan.close();
    }
}
```

Expected Output

```
Enter the Size of the array
3
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
2
----Stack Underflow----
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
3
----STACK UNDERFLOW - No Element to display----
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
1
Enter the Elements to be instered
10
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
1
Enter the Elements to be instered
20
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
1
Enter the Elements to be instered
30
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
3
Elements in the stack are as follows :
ELEMENT :30
ELEMENT :20
ELEMENT :10
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
1
Enter the Elements to be instered
40
----Stack OVER FLOW - Elements in the array are :3 ----
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
3
Elements in the stack are as follows :
ELEMENT :30
ELEMENT :20
ELEMENT :10
Stack Operation
1.Push
2.Pop
3.Display
4.Exit
Enter Your Choice
```

---

2 A: Design a superclass 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.

Program Code

```
//package twoA;
import java.util.Scanner;

class staff
{
    public static Scanner scan= new
Scanner(System.in);
    int staffid;
    String name,phno;
    double sal;
    void get_info()
    {
        System.out.println("Enter in order \nStaff
ID : Name : Phone Number : Salary :");
        staffid=scan.nextInt();
        name=scan.next();
        phno=scan.next();
        sal=scan.nextDouble();
    }
    void display()
    {
```

```
        System.out.println("Staff Details:\nStaff
Id : "+staffid+"\nName :"+name+"\nPhone Number :
"+phno+"\nSalary : "+sal);
    }
}
class Teaching extends staff
{
    public static Scanner scan= new
Scanner(System.in);
    String domain,pub;
    void get_info()
    {
        super.get_info();
        System.out.println("Enter the Domain of
Teacher "+name+" in order\nDomain : Publication");
        domain=scan.next();
        pub=scan.next();
    }
    void display()
    {
        super.display();
        System.out.println("Domain :
"+domain+"\nPublication :"+pub);
    }
}
class Technical extends Teaching
{
    public static Scanner scan= new
Scanner(System.in);
    String skills;
    void get_info()
    {
        super.get_info();
        System.out.println("Enter the Skills for
Teacher "+name);
        skills=scan.next();
    }
```

```
    void display()
    {
        super.display();
        System.out.println("Skills :"+skills);
    }
}
class Contact extends Technical
{
    public static Scanner scan= new
Scanner(System.in);
    String period;
    void get_info()
    {
        super.get_info();
        System.out.println("Enter the periods
handled by Teacher "+name);
        period=scan.next();
    }
    void display()
    {
        super.display();
        System.out.println("Period : "+period);
    }
}

public class College {
    public static Scanner scan= new
Scanner(System.in);
    public static void main(String[] args) {
        int n;
        System.out.println("Enter the Total Number
of Records");
        n=scan.nextInt();
        Contact c[] = new Contact[n];
        for(int i=0;i<n;i++)
        {
            c[i]=new Contact();
```

```
            int r=i+1;
            System.out.println("Enter the Details
of Record No "+r);
            c[i].get_info();
        }
        for(int i=0;i<n;i++)
        {
            int r=i+1;
            System.out.println("Displaying record
No :"+r);
            c[i].display();
        }
    }
}
```

Expected Output

```
Enter the Total Number of Records
3
Enter the Details of Record No 1
Enter in order
Staff ID : Name : Phone Number : Salary :
1001 Hemanth 9632587410 50000.50
Enter the Domain of Teacher Hemanth in order
Domain : Publication
cse proff
Enter the Skills for Teacher Hemanth
Programming
Enter the periods handled by Teacher Hemanth
5,8
Enter the Details of Record No 2
Enter in order
Staff ID : Name : Phone Number : Salary :
1002 Harsh 6325987410 35000.7
Enter the Domain of Teacher Harsh in order
Domain : Publication
cse ast.prof
Enter the Skills for Teacher Harsh
testing
Enter the periods handled by Teacher Harsh
2,7
Enter the Details of Record No 3
Enter in order
Staff ID : Name : Phone Number : Salary :
1003 Abdul 9635823022 45000.5
[Enter the Domain of Teacher Abdul in order
Domain : Publication
cse proff
Enter the Skills for Teacher Abdul
networking
Enter the periods handled by Teacher Abdul
1,3
Displaying record No :1
Staff Details:
Staff Id : 1001
Name :Hemanth
Phone Number : 9632587410
Salary : 50000.5
Domain : cse
Publication :proff
Skills :Programming
Period : 5,8
Displaying record No :2
Staff Details:
Staff Id : 1002
Name :Harsh
Phone Number : 6325987410
Salary : 35000.7
Domain : cse
Publication :ast.prof
Skills :testing
Period : 2,7
Displaying record No :3
Staff Details:
Staff Id : 1003
Name :Abdul
Phone Number : 9635823022
Salary : 45000.5
Domain : [cse
Publication :proff
Skills :networking
Period : 1,3
```

---

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 "/".

Program Code

```
//package twoB;
import java.util.Scanner;
import java.util.StringTokenizer;
public class Customer {
    public static void main(String[] args) {
        Scanner scan=new Scanner(System.in);
        String name;
        System.out.println("Enter the Name & DOB in
the Format <name,dd/mm/yyyy>");
        name=scan.next();
        StringTokenizer st = new
StringTokenizer(name,"/");
        int count=st.countTokens();
        for(int
i=1;i<=count&&st.hasMoreTokens();i++)
        {
            System.out.print(st.nextToken());
            if(i<count)
                System.out.println(",");
        }
        scan.close();
    }
}
```

Expected Output

```
Enter the Name & DOB in the Format <name,dd/mm/yyyy>
Hemanth,14/07/1999
Hemanth,14,
07,
1999
```

---

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.

Program Code

```
//package threeA;
import java.util.Scanner;
public class Exception {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int a,b,c;
```

```
        System.out.print("Enter the Two Interger
Values\nA :");
        a=scan.nextInt();
        System.out.print("B :");
        b=scan.nextInt();
        scan.close();
        try {
            if(b==0)
                throw new
ArithmeticException("Divide By Zero");
            c=a/b;
            System.out.println("\nThe Value of
"+a+" / "+b+" is "+c);
        }catch(ArithmeticException e)
        {
            e.printStackTrace();
        }
    }
}
```

Expected Output

Output - 3A.1:

```
Enter the Two Interger Values
A :10
B :2

The Value of 10 / 2 is 5
```

Output - 3A.2:

```
Enter the Two Interger Values
A :10
B :0
java.lang.ArithmeticException: Divide By Zero
        at threeA.Exception.main(Exception.java:14)
```

---

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.

Program Code

```
package threeB;
import java.lang.Thread;
import java.util.Random;
class first extends Thread{
    public void run()
    {
        int num=0;
        Random r = new Random();
        try
        {
            num=r.nextInt(100);
            System.out.println("First Thread : The
Number Generated is : "+num);
            Thread t2=new Thread(new second(num));
            t2.start();
            Thread t3=new Thread(new third(num));
```

```
            t3.start();
            Thread.sleep(1000);
        }catch(Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
class second implements Runnable {
    public int x;
    public second(int x)
    {
        this.x=x;
    }
    public void run()
    {
        System.out.println("Second Thread : Square
of the number is :"+x*x);
    }
}
class third implements Runnable {
    int x;
    public third(int x)
    {
        this.x=x;
    }
    public void run()
    {
        System.out.println("Third Thread : Cube of
the Number is :"+x*x*x);
    }
}
public class Multithread {
    public static void main(String[] args) {
        first a = new first();
        a.start();
    }
```

```
}
```

Expected Output

Output - 3B.1:

```
First Thread : The Number Generated is : 95
Second Thread : Square of the number is :9025
Third Thread : Cube of the Number is :857375
```

Output – 3B.2:

```
First Thread : The Number Generated is : 22
Second Thread : Square of the number is :484
Third Thread : Cube of the Number is :10648
```

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.

Program Code

```
package four;
import java.util.Random;
import java.util.Scanner;
public class QuickSort {
    public static void sort(int[] a)
    {
        quicksort(a,0,a.length-1);
    }
    public static void quicksort(int[] a,int
low,int high)
    {
        int i=low,j=high,temp,pivot=a[low];
        while(i<=j)
        {
            while(a[i]<pivot)
                i++;
            while(a[j]>pivot)
                j--;
            if(i<=j)
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
                i++;
                j--;
            }
        }
        if(j>low)
            quicksort(a,low,j);
        if(i<high)
            quicksort(a,i,high);
    }
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int i;
```

```
        Random r = new Random();
        System.out.println("Quick Sort\nEnter the
Number of times the algorithm should Run");
        int times = scan.nextInt();
        double totaldur=0;
        for(int j=0;j<times;j++)
        {
            System.out.println("Random Number
Generated are at POS  "+j+"  as follows : ");
            int[] a = new int[10];
            for(i=0;i<10;i++)
            {
                a[i]=r.nextInt(1000);
                System.out.print(a[i]+" ");
            }
            System.out.println("");
            long StartTime = System.nanoTime();
            sort(a);
            double EndTime = System.nanoTime();
            double duration = (EndTime -
StartTime);
            System.out.println("Elements after
Sorting are");
            for(i=0;i<10;i++)
                System.out.print(a[i]+" ");
            System.out.println("");
            totaldur=totaldur+duration;
        }
        System.out.println("\nTotal time taken to
Sort is :"+totaldur+" Nano Seconds");
        double miliseconds = (totaldur / 1000000);
        System.out.println("\nTotal time taken to
Sort is :"+miliseconds+" Mili Seconds");
        double avg=totaldur/times;
        System.out.println("The Average time Spend
by the System is : "+avg+" Nano Second");
        double miliavg=(avg/1000000);
```

```
        System.out.println("The Avergae time Spend
by the System is : "+miliavg+" Mili Seconds");
        scan.close();
    }
}
```

Expected Output

```
Quick Sort
Enter the Number of times the algorithm should Run
10
Random Number Generated are at POS  0  as follows :
383 522 474 352 516 612 106 248 997 926
Elements after Sorting are
106 248 352 383 474 516 522 612 926 997
Random Number Generated are at POS  1  as follows :
0 653 842 104 293 826 896 308 502 535
Elements after Sorting are
0 104 293 308 502 535 653 826 842 896
Random Number Generated are at POS  2  as follows :
899 637 654 716 632 668 357 895 124 709
Elements after Sorting are
124 357 632 637 654 668 709 716 895 899
Random Number Generated are at POS  3  as follows :
165 206 325 446 788 929 963 583 91 822
Elements after Sorting are
91 165 206 325 446 583 788 822 929 963
Random Number Generated are at POS  4  as follows :
369 668 554 8 86 64 461 881 515 812
Elements after Sorting are
8 64 86 369 461 515 554 668 812 881
Random Number Generated are at POS  5  as follows :
927 812 73 941 810 456 384 140 841 87
Elements after Sorting are
73 87 140 384 456 810 812 841 927 941
Random Number Generated are at POS  6  as follows :
22 174 650 901 122 988 666 739 916 325
Elements after Sorting are
22 122 174 325 650 666 739 901 916 988
Random Number Generated are at POS  7  as follows :
454 698 899 167 716 701 674 732 13 736
Elements after Sorting are
13 167 454 674 698 701 716 732 736 899
Random Number Generated are at POS  8  as follows :
976 209 703 509 626 66 2 725 831 993
Elements after Sorting are
2 66 209 509 626 703 725 831 976 993
Random Number Generated are at POS  9  as follows :
63 115 857 758 584 164 891 856 488 939
Elements after Sorting are
63 115 164 488 584 758 856 857 891 939

Total time taken to Sort is :54120.0 Nano Seconds

Total time taken to Sort is :0.05412 Mili Seconds
The Average time Spend by the System is : 5412.0 Nano Second
The Avergae time Spend by the System is : 0.005412 Mili Seconds
```

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.

Program Code

```
package five;
import java.util.Scanner;
import java.util.Random;
public class MergeSortExp {
    public static void mergeSort(int[] a,int
low,int high)
    {
        int N=high-low;
        if(N<=1)
            return;
        int mid=low+(N/2);
        mergeSort(a,low,mid);
        mergeSort(a,mid,high);
        int[] temp=new int[N];
        int i=low, j=mid;
        for(int k=0;k<N;k++)
        {
            if(i==mid)
                temp[k]=a[j++];
            else if(j==high)
                temp[k]=a[i++];
            else if(a[j]<a[i])
                temp[k]=a[j++];
            else
                temp[k]=a[i++];
        }
        for(int k=0;k<N;k++)
```

```
            a[low++]=temp[k];
    }
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int i;
        Random r = new Random();
        System.out.println("Enter the Number of
times the algorithm should Run");
        int times = scan.nextInt();
        double totaldur=0;
        for(int j=0;j<times;j++)
        {
            System.out.println("Random Number
Generated are at POS  "+j+"  as follows : ");
            int[] a = new int[10];
            for(i=0;i<10;i++)
            {
                a[i]=r.nextInt(1000);
                System.out.print(a[i]+" ");
            }
            System.out.println("");
            long StartTime = System.nanoTime();
            mergeSort(a,0,10);
            double EndTime = System.nanoTime();
            double duration = (EndTime -
StartTime);
            System.out.println("Elements after
Sorting are");
            for(i=0;i<10;i++)
                System.out.print(a[i]+" ");
            System.out.println("");
            totaldur=totaldur+duration;
        }
        System.out.println("\nTotal time taken to
Sort is :"+totaldur+" Nano Seconds");
        double miliseconds = (totaldur / 1000000);
```

```
        System.out.println("\nTotal time taken to
Sort is :"+miliseconds+" Mili Seconds");
        double avg=totaldur/times;
        System.out.println("The Average time Spend
by the System is : "+avg+" Nano Second");
        double miliavg=(avg/1000000);
        System.out.println("The Avergae time Spend
by the System is : "+miliavg+" Mili Seconds");
        scan.close();
    }
}
```

Expected Output

```
Merge Sort
Enter the Number of times the algorithm should Run
10
Random Number Generated are at POS  0  as follows :
416 842 314 672 7 963 27 262 947 934
Elements after Sorting are
7 27 262 314 416 672 842 934 947 963
Random Number Generated are at POS  1  as follows :
835 615 818 910 368 599 887 643 146 61
Elements after Sorting are
61 146 368 599 615 643 818 835 887 910
Random Number Generated are at POS  2  as follows :
56 181 728 163 995 305 592 68 323 909
Elements after Sorting are
56 68 163 181 305 323 592 728 909 995
Random Number Generated are at POS  3  as follows :
419 693 709 125 859 508 25 16 145 805
Elements after Sorting are
16 25 125 145 419 508 693 709 805 859
Random Number Generated are at POS  4  as follows :
680 604 871 604 585 309 135 404 295 829
Elements after Sorting are
135 295 309 404 585 604 604 680 829 871
Random Number Generated are at POS  5  as follows :
390 6 699 886 705 173 225 379 202 126
Elements after Sorting are
6 126 173 202 225 379 390 699 705 886
Random Number Generated are at POS  6  as follows :
81 268 399 294 885 160 647 875 754 29
Elements after Sorting are
29 81 160 268 294 399 647 754 875 885
Random Number Generated are at POS  7  as follows :
130 148 205 619 596 493 467 230 141 298
Elements after Sorting are
130 141 148 205 230 298 467 493 596 619
Random Number Generated are at POS  8  as follows :
69 874 888 476 942 665 118 266 698 32
Elements after Sorting are
32 69 118 266 476 665 698 874 888 942
Random Number Generated are at POS  9  as follows :
702 47 548 152 868 241 812 912 3 178
Elements after Sorting are
3 47 152 178 241 548 702 812 868 912

Total time taken to Sort is :94242.0 Nano Seconds

Total time taken to Sort is :0.094242 Mili Seconds
The Average time Spend by the System is : 9424.2 Nano Second
The Avergae time Spend by the System is : 0.0094242 Mili Seconds
```

6A: Implement in Java, the 0/1 Knapsack problem using (a) Dynamic Programming method

---

Program Code

```
package sixA;

import java.util.Scanner;

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

    Knapsack() {     //constructor automatically
called
        Scanner scan = new Scanner(System.in);

        System.out.println("Enter Number of
Items");
        n = scan.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] = scan.nextInt();
        }

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

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

```
    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();
    }
```

```
}
```

Expected Output

```
Enter Number of Items
4
Enter Weights of Items
2
1
2
3
Enter Profits of Items
12
10
15
20
Enter Capacity of Knapsack
5
Maximum Profit: 37
Items Considered: 3 2 1
```

6B: Implement in Java, the 0/1 Knapsack problem using (b) Greedy method.

Program Code

```
package sixB;
```

```
import java.util.Scanner;

class Knapsack {

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

    Knapsack() {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter number of
Items");
        int n = scan.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] = scan.nextDouble();
        }

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

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

    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();
    }
}
```

Expected Output

```
Enter number of Items
4
Enter Weights of Items
2
1
2
3
Enter Profits of Items
12
10
15
20
Enter Capacity of Knapsack
5
Items Considered: 2 3 4
Maximum Profit is: 38.333333333333336
Solution Vector: 0.0 1.0 1.0 0.6666666666666666
```

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.

Program Code

```
package seven;
import java.util.Scanner;
public class Diji {

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

            System.out.println("Enter Source
vertex");
            int src = scan.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]);
            }
            scan.close();
        }

        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;
        }
}
```

Expected Output

```
Enter Number of Vertices
5
Enter Adjacency Matrix
0 3 99 7 99
3 0 4 2 99
99 4 0 5 6
5 2 0 4 4
99 99 6 4 0
Enter Source vertex
1
Shortest Distance from 1 to 2 is 3
Shortest Distance from 1 to 3 is 5
Shortest Distance from 1 to 4 is 5
Shortest Distance from 1 to 5 is 9
```

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

Program Code

```
package eight;
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 Kruskal {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter number of
Vertices");
        int n = scan.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] = scan.nextInt();
            }
        }
        scan.close();

        // 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;
    }
}
```

Expected Output

```
Enter number of Vertices
6
Enter Adjacency Matrix
0 3 99 99 6 5

3 0 1 99 99 4
99 1 0 6 99 4
99 99 6 0 8 5
6 99 99 8 0 2
5 4 5 2 2 0
Edges:
2 <-> 3
5 <-> 6
1 <-> 2
2 <-> 6
4 <-> 6

Minimum Cost of Spanning Tree: 15
```

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

Program Code

```
package nine;

import java.util.Scanner;

class Prim {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
```

```
        System.out.println("Enter Number of
Vertices");
        int n = scan.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] = scan.nextInt();

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

        System.out.println("Enter Source Vertex");
        int srcVertex = scan.nextInt();
        scan.close();
        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);
    }
}
```

Expected Output

```
Enter Cost Adjacency Matrix
0 3 99 99 6 5
3 0 1 99 99 4
99 1 0 6 99 4
99 99 6 0 8 5
6 99 99 8 0 2
6 4 5 2 2 0
Enter Source Vertex
1
Edges: (1,2)(2,3)(2,6)(6,4)(6,5)
Minimum cost of Spanning Tree: 12
```

10A: Write Java programs to

- **Implement All-Pairs Shortest Paths problem using Floyd's algorithm.**

Program Code

```
package tenA;

import java.util.Scanner;

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

        System.out.println("Enter Number of
Vertices");
        int n = scan.nextInt();
        int[][] D = new int[10][10];

        System.out.println("Enter Distance
Matrix");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                D[i][j] = scan.nextInt();
            }
        }
        scan.close();
        for (int k = 1; k <= n; k++) {
            for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= n; j++) {
                    D[i][j] = Math.min(D[i][j],
D[i][k] + D[k][j]);
                }
            }
        }

        System.out.println("Shortest Distance
Matrix");
```

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

Expected Output

```
Enter Number of Vertices
4
Enter Distance Matrix
0 99 3 99
2 0 99 99
99 7 0 7
6 99 99 0
Shortest Distance Matrix
0 10 3 10
2 0 5 12
9 7 0 7
6 16 9 0
```

10B: Write Java programs to

- **Implement Travelling Sales Person problem using Dynamic programming.**

Program Code

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

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

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

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

        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];
        }
    }
}
```

Expected Output

```
Enter number of cities
4
Enter Adjacency Matrix
0 2 4 6
2 0 4 6
2 4 0 6
2 4 6 0
Enter Source City
1
Total Cost: 14
Path: 1 2 3 4 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.

Program Code

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

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

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

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

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

        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();
    }
}
```

Expected Output

Case 1:

```
Enter n value
5
Enter Elements of Set
1 2 5 6 8
Enter Total Sum value
9
Solution: 1 2 6
Solution: 1 8
```

Case 2:

```
Enter n value
5
Enter Elements of Set
1 2 3 4 5
Enter Total Sum value
20
No solution
```

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

Program Code

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

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

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

        System.out.println("Enter Number of
Vertices");
        n = scan.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] = scan.nextInt();
            }
        }
        scan.close();
        // 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]);
    }
}
```

Expected Output

```
Enter Number of Vertices
6
Enter Adjacency Matrix
0 1 1 1 0 0
1 0 1 0 0 1
1 1 0 1 1 0
1 0 1 0 1 0
0 0 1 1 0 1
0 1 0 0 1 0
Hamiltonian Cycle are
1 2 6 5 3 4 1
1 2 6 5 4 3 1
1 3 2 6 5 4 1
1 3 4 5 6 2 1
1 4 3 5 6 2 1
1 4 5 6 2 3 1
```

Viva Questions

END
