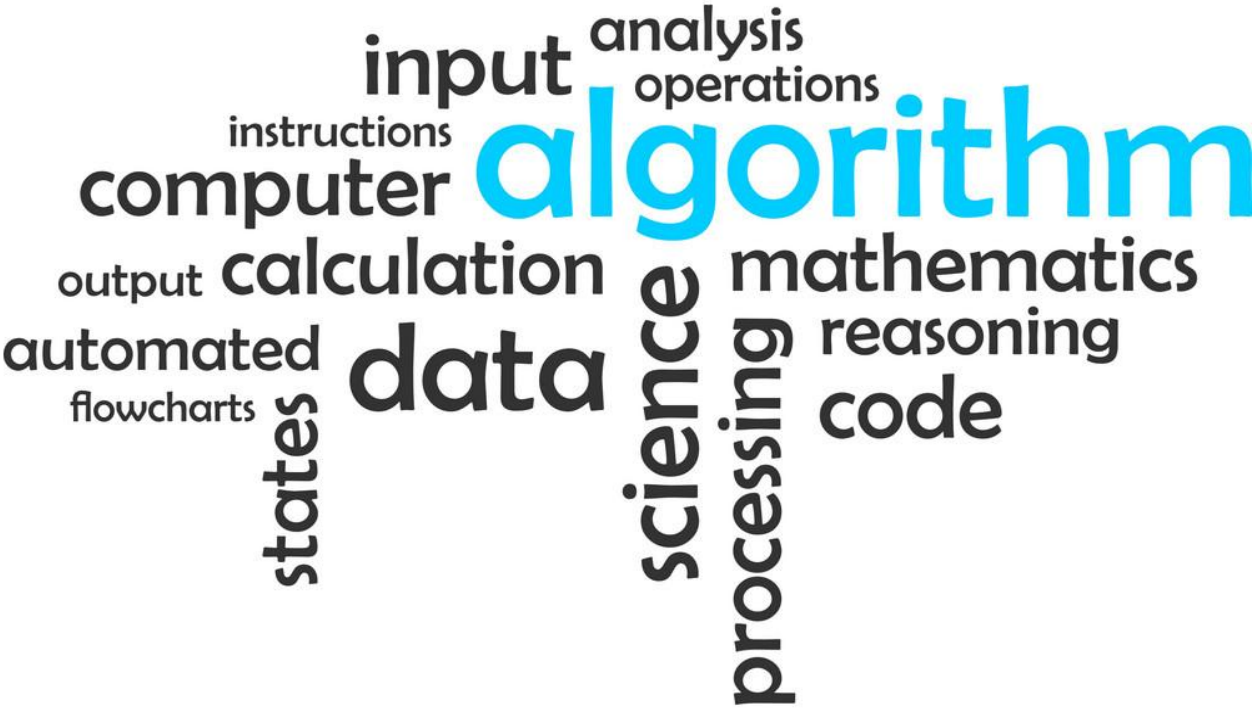
DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY (18CSL47)



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

(Effective from the academic year 2019 -2020)

Bapuji Educational Association(R.)
Bapuji Institute of Engineering and Technology (BIET)
Davangere, Karnataka - 577004

Department of Computer Science and Engineering

Experiments

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 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.

   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 "/".

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.

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

5. Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus 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.

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.

```
import java.util.Scanner;
public class Student
{
   private String usn, name, branch, phone;

   public Student(String usn, String name, String branch, String phone)
   {
            super();
            this.usn = usn;
            this.name = name;
            this.branch = branch;
            this.phone = phone;
   }
   @Override
   public String toString()
   {
            return "Student [USN = " + usn + ", NAME = " + name + ", BRANCH = " + branch
                            + ", PHONE NUMBER = " + phone + "]";
   }

   public static void main(String args[])
   {
            int i;
            String usn, branch, name, phone;

            Scanner s = new Scanner(System.in);

            System.out.println("Enter number of Students: ");
            int n = s.nextInt();

            Student[] students = new Student[n + 1];

            for(i = 1; i<= n; i ++)
            {
                        System.out.println("Enter student "+ i +" details\n");
            System.out.println("Give Student Details USN, Name, Branch, Phone Number");
            usn = s.next();
            name = s.next();
            branch = s.next();
            phone = s.next();

            students[i] = new Student(usn, name, branch, phone);
        }
        System.out.println("DATABASE:");

        for(i = 1; i <= n; i ++)
        {
            System.out.println(students[i]);
        }
    }
}
```

OUTPUT :

```
Enter number of Students:
2
Enter student 1 details

Give Student Details USN, Name, Branch, Phone Number
4BD13CV001
ARJUN
CIVIL
9264921640
Enter student 2 details

Give Student Details USN, Name, Branch, Phone Number
4BD15IS010
CHARAN
IS
7592783640
DATABASE:
Student [USN = 4BD13CV001, NAME = ARJUN, BRANCH = CIVIL, PHONE NUMBER = 9264921640]
Student [USN = 4BD15IS010, NAME = CHARAN, BRANCH = IS, PHONE NUMBER = 7592783640]
```

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;
public class StackDemo
{
  public static void main(String[] args)
  {
          int top = -1;
          int n,element,i;
          int[] a;
          Scanner s = new Scanner(System.in);
          System.out.println("Enter Stack Size");
          n = s.nextInt();
          a = new int[n+1];
          System.out.println("Stack operations are" +"\t"+ "1.Push"+"\t"+ "2.POP"+"\t"+
"3.Display"+"\t"+ "4.Exit");

          for(;;)
          {
                    System.out.println("Enter your choice");
                    int choice = s.nextInt();

                    switch(choice)
                    {
                              case 1: if(top == n-1)
                                                  {
                                                            System.out.println("Stack overflow");
                                                  }
                                                  else
                                                  {
                                                            System.out.println("Enter element to be pushed");
                                                            element = s.nextInt();
                                                            a[top++] = element;
                                                  }
                                                  break;

                              case 2: if(top == -1)
                                                  {
                                                            System.out.println("Stack Underflow");
                                                  }
                                                  else
                    {
                        System.out.println("Popped element " + a[top--]);
                    }
                    break;

            case 3: if(top== -1)
                    {
                        System.out.println("Stack Empty");
                    }
                    else
                    {
                        System.out.println("Elements in stack :");
                        for ( i = top; i >= 0; i--)
                        {
                            System.out.println(a[i]);
                        }
                    }
                    break;

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

OUTPUT:

```
Enter Stack Size
3
Stack operations are   1.Push   2.POP   3.Display   4.Exit
Enter your choice
1
Enter element to be pushed
53
Enter your choice
1
Enter element to be pushed
68
Enter your choice
1
Enter element to be pushed
20
Enter your choice
1
```

```
Stack overflow
Enter your choice
3
Elements in stack :
20
68
53
Enter your choice
2
Popped element 20
Enter your choice
2
Popped element 68
Enter your choice
2
Popped element 53
Enter your choice
2
Stack Underflow
Enter your choice
3
Stack Empty
Enter your choice
4
```

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
{
   protected String staffId, name, ph;
   protected float salary;

   public Staff(String staffId, String name, float salary2, String ph)
   {
             super();
             this.staffId = staffId;
             this.name = name;
             this.salary = salary2;
             this.ph = ph;
   }

   @Override
   public String toString()
   {
             return "Staff [staffId=" + staffId + ", name=" + name + ", salary="
                                     + salary + ", ph=" + ph + "]";
   }
}

class Teaching extends Staff
{
   private String domian, publication;

   public Teaching(String staffId, String name, float salary, String ph,
                      String domian, String publication)
   {
             super(staffId, name, salary, ph);
             this.domian = domian;
             this.publication = publication;
   }
```

```
  @Override
  public String toString()
  {
        return "Teaching [domian=" + domian + ", publication=" + publication
                        + ", staffId=" + staffId + ", name=" + name + ", ph=" + ph
                        + ", salary=" + salary + "]";
  }
}

class Technical extends Staff
{
  private String skills;

  public Technical(String staffId, String name, float salary, String ph,
              String skills)
  {
        super(staffId, name, salary, ph);
        this.skills = skills;
  }

  @Override
  public String toString()
  {
        return "Technical [skills=" + skills + ", staffId=" + staffId
                        + ", name=" + name + ", ph=" + ph + ", salary=" + salary + "]";
  }
}

class Contract extends Staff
{
  private String period;

  public Contract(String staffId, String name, float salary, String ph,
              String period)
  {
        super(staffId, name, salary, ph);
        this.period = period;
  }

  @Override
  public String toString()
  {
        return "Contract [period=" + period + ", staffId=" + staffId
                        + ", name=" + name + ", ph=" + ph + ", salary=" + salary + "]";
  }
}
```

```
public class StaffDemo
{

   public static void main(String[] args)
   {
            int i,choice;
            String staffId, name, ph, domain, publication, skills, period;
            float salary;
            int teachingCount=0, technicalCount=0, contractCount=0;

            Teaching[] teachingStaff = new Teaching[10];
            Contract[] contractStaff = new Contract[10];
            Technical[] technicalStaff = new Technical[10];

            Scanner s = new Scanner(System.in);

            System.out.println("1 Teaching Staff Entry");
            System.out.println("2 Technical Staff Entry");
            System.out.println("3 Contract Staff Entry");
            System.out.println("4 Teaching Staff Details");
            System.out.println("5 Technical Staff Details");
            System.out.println("6 Contract Staff Details");
            System.out.println("7.Exit");

            for(;;)
            {
                        System.out.println("enter your choice");
                        choice = s.nextInt();
                        switch(choice)
                        {
                        case 1: System.out.println("Enter Teaching
Details(StaffId,Name,Salary,PhoneNumber,Domain,Publication)");
                                                staffId = s.next();
                                                name = s.next();
                                                salary = s.nextFloat();
                                                ph = s.next();
                                                domain = s.next();
                                                publication = s.next();
                                                teachingStaff[teachingCount]= new
Teaching(staffId,name,salary,ph,domain,publication);
                                                teachingCount++;
                                                break;

                        case 2: System.out.println("Enter Technical
staffDetails(StaffId,Name,Salary,PhoneNumber,Skills)");
```

```
                staffId = s.next();
                name = s.next();
                salary = s.nextFloat();
                ph = s.next();
                skills = s.next();
                technicalStaff[technicalCount] = new
Technical(staffId,name,salary,ph,skills);
                technicalCount++;
                break;

        case 3: System.out.println("enter Contract staff
details(StaffId,Name,Salary,PhoneNumber,Period)");
                staffId = s.next();
                name = s.next();
                salary = s.nextFloat();
                ph = s.next();
                period = s.next();
                contractStaff[contractCount] = new Contract(staffId,name,salary,
ph ,period);
                contractCount++;
                break;

        case 4: System.out.println("Teaching Staff Details");

                if(teachingCount==0)
                {
                        System.out.println("No teaching staff details
available");
                }
                else
                {
                    for(i=0;i<teachingCount;i++)
                    {
                    System.out.println(teachingStaff[i]);
                    }
                }
                break;

        case 5: System.out.println("Technical Staff Details:");
                if(technicalCount==0)
                {
                    System.out.println("No technical staff details available");
                }
                else
                {
                    for(i=0;i<technicalCount;i++)
                    {
                    System.out.println(technicalStaff[i]);
                    }
                }
                break;

        case 6: System.out.println("contract Staff Details:");

                if(contractCount==0)
                {
                    System.out.println("No contract staff details available");
                }
                else
                {
                    for(i=0;i<contractCount;i++)
                    {
                        System.out.println(contractStaff[i]);
                    }
                }
                break;

        case 7: System.exit(0);
        }
    }

 }

}
```

OUTPUT :

```
1 Teaching Staff Entry
2 Technical Staff Entry
3 Contract Staff Entry
4 Teaching Staff Details
5 Technical Staff Details
6 Contract Staff Details
7.Exit
enter your choice
1
Enter Teaching Details(StaffId,Name,Salary,PhoneNumber,Domain,Publication)
100
Arjun
10000
9988776655
```

```
CSE
COMPUTER_GRAPHICS
enter your choice
1
Enter Teaching Details(StaffId,Name,Salary,PhoneNumber,Domain,Publication)
201
Amith
10001
9898767655
CSE
MOBILE_COMPUTING
enter your choice
4
Teaching Staff Details
Teaching [domian=CSE, publication=COMPUTER_GRAPHICS, staffId=100, name=Arjun,
ph=9988776655, salary=10000.0]
Teaching [domian=CSE, publication=MOBILE_COMPUTING, staffId=201, name=Amith,
ph=9898767655, salary=10001.0]
enter your choice
2
Enter Technical staffDetails(StaffId,Name,Salary,PhoneNumber,Skills)
105
Namratha
25000
9988666444
Python
enter your choice
2
Enter Technical staffDetails(StaffId,Name,Salary,PhoneNumber,Skills)
110
Sanjay
20500
9911226640
Siebel
enter your choice
5
Technical Staff Details:
Technical [skills=Python, staffId=105, name=Namratha, ph=9988666444, salary=25000.0]
Technical [skills=Siebel, staffId=110, name=Sanjay, ph=9911226640, salary=20500.0]
enter your choice
3
enter Contract staff details(StaffId,Name,Salary,PhoneNumber,Period)
550
Kiran
25600
9882648109
```

```
7
enter your choice
6
contract Staff Details:
Contract [period=7, staffId=550, name=Kiran, ph=9882648109, salary=25600.0]
enter your choice
7
```

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

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

class Customer
{
   private String customerName,date;
   public Customer(String customerName, String date)
   {
             super();
            this.customerName = customerName;
            this.date = date;
   }

            @Override
   public String toString()
   {
             String returnValue = customerName+",";
             StringTokenizer tokenizer = new StringTokenizer(date,"/");
             System.out.println("The Customer details are ");
             while(tokenizer.hasMoreTokens())
             {
                         returnValue = returnValue+tokenizer.nextToken();
                         if(tokenizer.hasMoreElements())
                         {
                                    returnValue = returnValue+",";
                         }
             }
             return returnValue;
   }
}
public class CustomerDetails
{
   public static void main(String[] args)
   {
             String customerName;
             String date;
             Scanner scanner = new Scanner(System.in);
             System.out.println("Enter customer name");
             customerName = scanner.next();
             System.out.println("Enter Date (dd/mm/yyy)");
             date = scanner.next();
             Customer customer = new Customer(customerName,date);
             System.out.println(customer.toString());
   }
}
```

OUTPUT :

```
Enter customer name
Thomas
Enter Date (dd/mm/yyy)
10/10/1916
The Customer details are
Thomas,10,10,1916
```

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

```
import java.util.*;
public class Division
{
   public static void main(String[] args)
   {
           int a,b,quotient;
           Scanner s = new Scanner(System.in);
           System.out.println("Enter Numerator:" );
           a = s.nextInt();
           System.out.println("Enter Denominator:" );
           b = s.nextInt();
           try
           {
                   quotient=a/b;
                   System.out.println("Quotient=" + quotient);
           }

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

OUTPUT :

```
(1) Enter Numerator:
        12
        Enter Denominator:
        6
        Quotient=2
(2) Enter Numerator:
        20
        Enter Denominator:
        0
        java.lang.ArithmeticException: / by 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.

```
import java.util.*;

class Cube implements Runnable
{
   public int x;
   public Cube (int x)
   {
            this.x=x;
   }
   public void run()
   {
            System.out.println("From third thread-Cube of" + x + "is:" +x*x*x);
   }
}

class Square implements Runnable
{
   public int x;
   public Square (int x)
   {
            this.x=x;
   }
   public void run()
   {
            System.out.println("From second thread-Square of" + x + "is:" +x*x);
   }
}

class FirstThreadIsRandom extends Thread
{
   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("Main Thread Started and Generated Number is"+num);
                        Thread t2 = new Thread(new Square (num));
                        t2.start();
                        Thread t3 = new Thread (new Cube(num));
                        t3.start();
                        Thread.sleep(1000);
                        System.out.println("------------------------------------------------");
                }
        }
        catch (Exception ex)
        {
                System.out.println(ex.getMessage());
        }
  }
}

public class MultiThreaded
{

  public static void main(String[] args)
  {
        FirstThreadIsRandom firstThread = new FirstThreadIsRandom();
        Thread t1 = new Thread (firstThread);
        t1.start();
  }
}
```

OUTPUT:

```
Main Thread Started and Generated Number is 41
From second thread-Square of 41 is:1681
From third thread-Cube of 41 is:68921
------------------------------------------------
Main Thread Started and Generated Number is68
From second thread-Square of 68 is:4624
From third thread-Cube of 68 is:314432
------------------------------------------------
Main Thread Started and Generated Number is 34
From second thread-Square of 34 is:1156
From third thread-Cube of 34 is:39304
```

```
-------------------------------------------------
Main Thread Started and Generated Number is 9
From second thread-Square of 9 is:81
From third thread-Cube of 9is:729
-------------------------------------------------
Main Thread Started and Generated Number is 76
From second thread-Square of 76 is:5776
From third thread-Cube of 76 is:438976
-------------------------------------------------
```

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
{
   private int a[];

   public QuickSort(int[] a)
   {
           this.a = a;
   }

   public int partition ( int a[], int m, int p )
   {
           int v = a[m];
           int i = m;
           int j = p;
           do
           {
          while ( a[++ i] < v );
          while ( a[-- j] > v );
                  if ( i < j )
                          interchange ( a, i, j );
           } while ( i <= j );
           a[m] = a[j]; a[j] = v;
           return j;
   }

   public void qSort ( int p, int q )
   {
           int j;
```

```
        if ( p < q )
        {
                j = partition ( a, p, q + 1 );
                qSort ( p, j - 1 );
                qSort ( j + 1, q );
        }
  }

  public void interchange ( int a[], int i, int j )
  {
        int t;
        t = a[i];
        a[i] = a[j];
        a[j] = t;
  }
}

public class QuickSortDemo
{
  public static void main(String[] args)
  {
        int n, a[], i;
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the Size of an Array:  ");
        n = input.nextInt();
        a = new int[n + 1];
        Random rn = new Random();
        System.out.println("System automatically generates numbers ");
        for ( i = 0; i < n; ++ i )
        {
                a[i] = rn.nextInt(n);
        }
        a[i] = 100000; //Sentinel value
        QuickSort qSort = new QuickSort(a);

        System.out.println("Before Sort: ");
        for ( i = 0; i < n; ++ i )
        {
                System.out.print(a[i] + "\t");
        }

        int p = 0;
        int q = n - 1;

        qSort.qSort(p, q);
        System.out.println("\n\nAfter Sort: ");
        for ( i = 0; i < n; ++ i )
        {
            System.out.print(a[i] + "\t");
        }

        int step = 2000;
        double duration;

        /* times for n = 0, 10, ..., 100, 200, ..., 5000 */
        System.out.println ( "\n\nN\tRepetitions\tTime\n" );
        for ( n = 5000; n < 50000; n += step )
        {
            a = new int[n + 1];
            qSort = new QuickSort(a);

            /*get time for size n */
            long repetitions = 0;
            long start = System.nanoTime();
            do
            {
                repetitions ++;
                for ( i = 0; i < n; ++ i )
                    a[i] = rn.nextInt(n);
                a[i] = 100000; //Sentinel value
                qSort.qSort(0, n - 1);
            } while ( System.nanoTime() - start < 1000000000 );
                    /* repeat until enough time has elapsed */
            duration = ( ( double ) ( System.nanoTime() - start ) ) / 1000000000;
            duration /= repetitions;
            System.out.println ( n + "\t" + repetitions + "\t\t" + duration );
        }
    }
}
```

OUTPUT:

```
Enter the Size of an Array:
5
System automatically generates numbers
Before Sort:
2   4   0   4   2

After Sort:
0   2   2   4   4
```

```
N       Repetitions     Time

5000   2604            3.840779374039939E-4
7000   1826            5.47683173603505E-4
9000   1384            7.22663938583815E-4
11000   1116          8.963977114695341E-4
13000   933            0.0010729254876741692
15000   803            0.0012468262278953922
17000   694            0.0014428503530259367
19000   623            0.0016070116115569826
21000   559            0.0017905278372093024
23000   506            0.001978315013833992
25000   465            0.0021531490322580643
27000   428            0.0023395274672897196
29000   396            0.0025256930378787876
31000   369            0.0027141917425474254
33000   345            0.0028995773043478264
35000   325            0.0030829968984615384
37000   305            0.00328287162295082
39000   289            0.003461591204152249
41000   274            0.0036523042846715323
43000   243            0.004119721567901235
45000   248            0.004037317338709678
47000   233            0.004306232450643777
49000   227            0.004423571559471365
```

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.

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

class MergeSort
{
  private int a[];

  public MergeSort(int[] a)
  {
        this.a = a;
  }

  void merge ( int low, int mid, int high )
  {
       int b[] = new int[high + 1];
        int h = low;
        int i = low;
        int j = mid + 1;
        int k;

        while ( ( h <= mid ) && ( j <= high ) )
        {
                if ( a[h] <= a[j] ) b[i ++] = a[h ++];
                else b[i ++] = a[j ++];
        }
        if ( h > mid )
        {
                for ( k = j; k <= high; ++ k ) b[i ++] = a[k];
        }
        else
        {
```

```
                       for ( k = h; k <= mid; ++ k ) b[i ++] = a[k];
           }
           for ( k = low; k <= high; ++ k ) a[k] = b[k];
   }

   void mergeSort ( int low, int high )
   {
           int mid;

           if ( low < high )
           {
                       mid = ( low + high ) / 2;

                       mergeSort ( low, mid );
                       mergeSort ( mid + 1, high );

                       merge ( low, mid, high );
           }
   }
}

public class MergeSortDemo
{
   public static void main(String[] args)
   {
           int n, a[], i;
           Scanner input = new Scanner(System.in);
           System.out.println("Enter the Size of an Array:  ");

           n = input.nextInt();
           a = new int[n + 1];

           Random rn = new Random();
           System.out.println("System automatically generates numbers ");
           for ( i = 0; i < n; ++ i )
           {
                       a[i] = rn.nextInt(n);//a[i] = input.nextInt();
           }
           a[i] = 100000; //Sentinel value
           MergeSort mSort = new MergeSort(a);

           System.out.println("Before Sort: ");
           for ( i = 0; i < n; ++ i )
           {
                       System.out.print(a[i] + "\t");
           }
```

```
    int low = 0;
    int high = n - 1;

    mSort.mergeSort(low, high);

    System.out.println("\n\nAfter Sort: ");
    for ( i = 0; i < n; ++ i )
    {
        System.out.print(a[i] + "\t");
    }

    int step = 2000;
    double duration;

    /* times for n = 0, 10, ..., 100, 200, ..., 5000 */
    System.out.println ( "\n\nN\tRepetitions\tTime\n" );
    for ( n = 5000; n < 50000; n += step )
    {
        a = new int[n + 1];
        mSort = new MergeSort(a);

        /*get time for size n */
        long repetitions = 0;
        long start = System.nanoTime();
        do
        {
            repetitions ++;
            for ( i = 0; i < n; ++ i )
                a[i] = rn.nextInt(n);
            a[i] = 100000; //Sentinel value
            mSort.mergeSort(0, n - 1);
        } while ( System.nanoTime() - start < 1000000000 );
                /* repeat until enough time has elapsed */
        duration = ( ( double ) ( System.nanoTime() - start ) ) / 1000000000;
        duration /= repetitions;
        System.out.println ( n + "\t" + repetitions + "\t\t" + duration );

    }
 }
}
```

OUTPUT:

```
Enter the Size of an Array:
5
System automatically generates numbers
```

```
Before Sort:
4   2   1   2   3

After Sort:
1   2   2   3   4

N        Repetitions       Time

5000    199                      0.005027964120603015
7000    153                      0.006545848712418300 5
9000    97                       0.010360987432989691
11000   59                       0.017194349559322034
13000   54                       0.018756191537037035
15000   42                       0.024344312833333333
17000   33                       0.030582966272727274
19000   27                       0.037587088074074 07
21000   22                       0.046705298409090906
23000   18                       0.05575357561111111
25000   16                       0.0653416245625
27000   13                       0.07881347792307693
29000   10                       0.1020311572
31000   11                       0.09932865818181819
33000   10                       0.110072756
35000   9                        0.12348744877777779
37000   8                        0.139554033875
39000   7                        0.15578334585714287
41000   7                        0.16581026885714284
43000   6                        0.19381527966666667
45000   5                        0.215364133
47000   5                        0.22233623480000003
49000   4                        0.25112471825
```

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

```
import java.util.Scanner;
class DKnapsack
{
   int n;
   int c;
   int p[];
   int w[];
   int v[][];

   public DKnapsack(int n, int c, int[] p, int[] w)
   {
         super();
         this.n = n;
         this.c = c;
         this.p = p;
         this.w = w;
         this.v = new int[n + 1][c + 1];
   }

   void compute()
   {
         for ( int i = 0; i <= n; ++ i)
         {
                 for ( int j = 0; j <= c; ++ j)
                 {
                         if ( i == 0 || j == 0 )
                         {
                                 v[i][j] = 0;
                         }
                         else if ( j - w[i] >= 0 )
                         {
                                 v[i][j] = max ( v[i - 1][j], p[i] + v[i - 1][j - w[i]]);
                         }
                         else if ( j - w[i] < 0 )
                         {
                                 v[i][j] = v[i - 1][j];
                         }
                 }
         }
```

```
        System.out.println("Optimal Solution: " + v[n][c]);
        traceback();
  }

  void traceback()
  {
        System.out.println("The objects picked up into knapsack are:");

        int i = n;
        int j = c;

        while( i > 0)
        {
                if(v[i][j] != v[i-1][j])
                {
                        System.out.print(i + " ");
                        j = j - w[i];
                        i--;
                }
                else
                {
                        i--;
                }
        }
  }

  private int max(int i, int j)
  {
        if ( i > j ) return i;
        else return j;
  }
}
public class KpDynamic
{
  public static void main(String[] args)
  {
        int n;
        int c;

        Scanner input = new Scanner(System.in);
        System.out.println("Enter number of objects");
        n = input.nextInt();

        int[] p = new int[n+1];
        int[] w = new int[n+1];
        int i;
```

```
            System.out.println("Enter capacity of Knapsack");
            c = input.nextInt();

            System.out.println("Enter profit for each " + n + " objects");

            for ( i = 1; i <= n; i ++)
                        p[i] = input.nextInt();

            System.out.println("Enter weight for each " + n + " objects");

            for ( i = 1; i <= n; i ++)
                        w[i] = input.nextInt();

            DKnapsack dk = new DKnapsack(n, c, p, w);
            dk.compute();
   }
}
```

OUTPUT:

```
Enter number of objects
5
Enter capacity of Knapsack
20
Enter profit for each 5 objects
3
4
5
8
10
Enter weight for each 5 objects
2
3
4
5
9
Optimal Solution: 26
The objects picked up into knapsack are:
5 4 3 1
```

(b) Greedy method.

```
import java.util.Scanner;

class GKnapsack
{
  int n;
  double c;
  double p[];
  double w[];

  public GKnapsack(int n, double c, double[] p, double[] w)
  {
        super();
        this.n = n;
        this.c = c;
        this.p = p;
        this.w = w;
  }

  void compute()
  {
        int i;
        double[] x= new double[n+1];

        for (i=0; i<n; i++)
        {
                x[i] = 0.0;
        }

        double rc = c;

        for(i=0; i<n; i++)
        {
                if(w[i] > rc) break;
                x[i] = 1;
                rc = rc - w[i];
        }

        if(i<=n)
        {
                x[i] = rc/w[i];
        }

        double netProfit = 0.0;
        for ( i = 0; i < n; ++ i)
        {
                if ( x[i] > 0.0)
                {
                        netProfit = netProfit + x[i] * p[i];

                }

        }

        System.out.println("Net Profit: " + netProfit);
        System.out.println("The objects picked up into knapsack are:");

        for ( i = 0; i < n; ++ i)
        {
                System.out.println(x[i] + " ");

        }
  }
}

public class KpGreedy
{
  public static void main(String[] args)
  {
        int n;
        double c;

        Scanner input = new Scanner(System.in);
        System.out.println("Enter number of objects");
        n = input.nextInt();

        double[] p = new double[n+1];
        double[] w = new double[n+1];
        int i;

        System.out.println("Enter capacity of Knapsack");
        c = input.nextDouble();

        System.out.println("Enter profit for each " + n + " objects");

        for ( i = 0; i < n; i ++)
                p[i] = input.nextDouble();

        System.out.println("Enter weight for each " + n + " objects");
        for ( i = 0; i < n; i ++)
                w[i] = input.nextDouble();

        GKnapsack gk = new GKnapsack(n, c, p, w);
        gk.compute();
  }
}
```

OUTPUT:

```
Enter number of objects
7
Enter capacity of Knapsack
15
Enter profit for each 7 objects
6
10
18
15
3
5
7
Enter weight for each 7 objects
1
2
4
5
1
3
7
Net Profit: 55.333333333333336
The objects picked up into knapsack are:
1.0
1.0
1.0
1.0
1.0
0.6666666666666666
0.0
```

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.Arrays;
import java.util.Scanner;

public class Dijkstra
{

  static int n,cost[][],i,j,u,dist[],src;
  void dij(int src,int cost[][],int dist[],int n)
  {
       int visited[],min;
       visited=new int[n];

       for(i=0;i<n;i++)
       {
               visited[i]=0;
               dist[i]=cost[src][i];
       }

       visited[src]=1;
       dist[src]=0;

       for(i=0;i<n;i++)
       {
               if(i==src) continue;
               min=999;

               for(j=0;j<n;j++)
                      if((visited[j]==0)&&(min>dist[j]))
                      {
                              min=dist[j];
                              u=j;
                      }
                      visited[u]=1;

               for(j=0;j<n;j++)
                      if(visited[j]==0)
                      {
                              if(dist[j]>dist[u]+cost[u][j])
                                      dist[j]=dist[u]+cost[u][j];
                      }
       }
  }
```

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

           System.out.println("Enter the matrix");
           cost=new int[n][n];
           dist=new int[n];

           Arrays.fill(dist,0);

           for(i=0;i<n;i++)
                   for(j=0;j<n;j++)
                           cost[i][j]=sc.nextInt();

           System.out.println("Enter the source vertex");
           src=sc.nextInt();

           new Dijkstra().dij(src, cost, dist, n);

           System.out.println("Shortest path from "+src+" to all other vertices");
           for(i=0;i<n;i++)
                   System.out.println("To " +i+" is "+dist[i]);
   }
}
```

OUTPUT:

```
Enter the number of vertices
4
Enter the matrix
0       15      10      9999
9999    0       15      9999
20      9999    0       20
9999    10      9999    0
Enter the source vertex
2
Shortest path from 2 to all other vertices
To 0 is 20
To 1 is 30
To 2 is 0
To 3 is 20
```

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

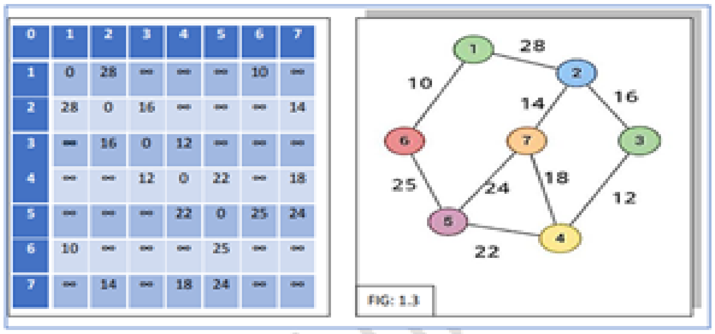
```
import java.util.Scanner;

public class Kruskals
{
  static int parent[],cost[][], mincost,n,i,j,ne,a,b,min,u,v;


  public void kruskal(int n,int[][] cost)
  {
        ne=1;
        while(ne<n)
        {
                min=999;
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                                if(cost[i][j]<min)
                                {
                                        min=cost[i][j];
                                        a=u=i;
                                        b=v=j;
                                }
                }

                u=find(u);
                v=find(v);

                if(v!=u)
                {
                        System.out.println( ne+"edge("+a+","+b+")="+min);
                        ne=ne+1;
                        mincost=mincost+min;
                        uni(u,v);
                }
                cost[a][b]=cost[b][a]=999;
        }
        System.out.println("The minimum cost of spanning tree is "+mincost);
  }

  public int find (int i)
  {
```

```
            while (parent[i] != 0)
                        i=parent[i];
            return i;
    }

            public void uni(int i,int j)
            {
    parent[j]=i;
            }


            public static void main(String[] args)
    {

            Scanner sc=new Scanner(System.in);

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

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

            parent=new int[n+1];

            System.out.println("Enter the cost matrix\n");
            for(i=1;i<=n;i++)
            {
                        for(j=1;j<=n;j++)
                        {
                                    cost[i][j]=sc.nextInt();
                                    if(cost[i][j]==0)
                                                cost[i][j]=999;
                        }
            }

            Kruskals k = new Kruskals();
            k.kruskal(n,cost);


    }

}
```

OUTPUT:

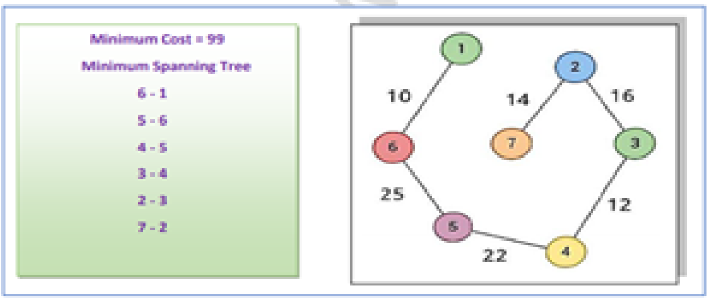
```
Enter the number of vertices
```

```
7
Enter the cost matrix

0       28      999     999     999     10      999

28      0       16      999     999     999     14

999     16      0       12      999     999     999

999     999     12      0       22      999     18

999     999     999     22      0       25      24

10      999     999     999     25      999     999

999     14      999     18      24      999     999

1edge(1,6)=10
2edge(6,5)=25
3edge(5,4)=22
4edge(4,3)=12
5edge(3,2)=16
6edge(2,7)=14

The minimum cost of spanning tree is 99
```



0	1	2	3	4	5	6	7
1	0	28	∞	∞	∞	10	∞
2	28	0	16	∞	∞	∞	14
3	∞	16	0	12	∞	∞	∞
4	∞	∞	12	0	22	∞	18
5	∞	∞	∞	22	0	25	24
6	10	∞	∞	∞	25	∞	∞
7	∞	14	∞	18	24	∞	∞

FIG: 1.3



Minimum Cost = 99

Minimum Spanning Tree

6 - 1

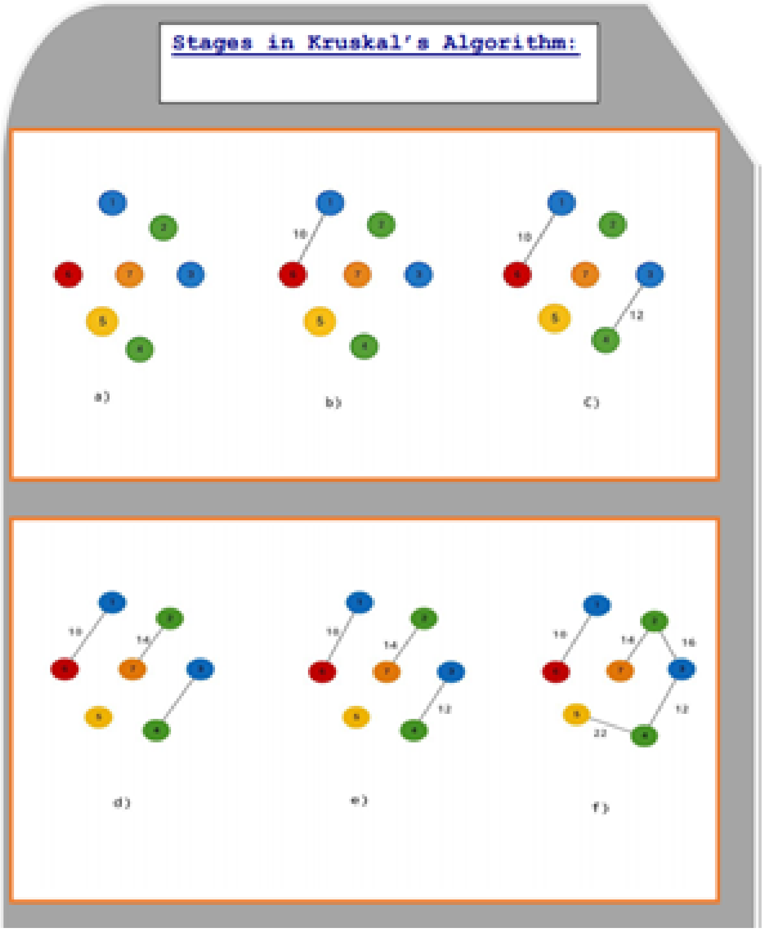
5 - 6

4 - 5

3 - 4

2 - 3

7 - 2



Stages in Kruskal's Algorithm:

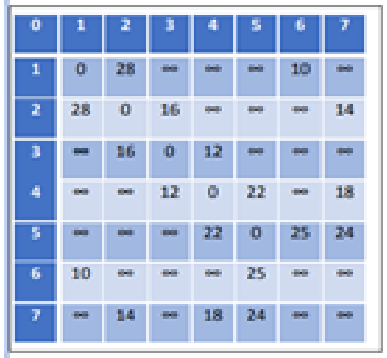
9. Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
import java.util.Scanner;

public class Prims
{
  static int mincost=0,n,i,j,ne,a=0,b=0,min,u = 0,v=0;

  public void prim(int n,int[][] cost)
  {

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

        for(i=2;i<=n;i++)
                visited[i]=0;

        visited[1]=1;
        ne=1;
        while(ne<n)
        {
               min=999;
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                if(cost[i][j]<min)
                                {
                                        if(visited[i]==0)
                                                continue;
                                        else
                                        {
                                                min=cost[i][j];
                                                a=u=i;
                                                b=v=j;
                                        }
                                }
                        }
                }
                if(visited[u]==0||visited[v]==0)
                {

                        System.out.println((ne)+"edge("+a+","+b+")="+min);
                        ne=ne+1;
                        mincost=mincost+min;
                        visited[v]=1;

                }
                cost[a][b]=cost[b][a]=999;
        }
        System.out.println("The minimum cost of spanning tree is "+mincost);

}

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

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

  int cost[][]= new int [n+1][n+1];
  System.out.println("Enter the cost matrix\n");
  for(i=1;i<=n;i++)
  {
                for(j=1;j<=n;j++)
                {
                        cost[i][j]=sc.nextInt();
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }
  }

  Prims p = new Prims();
  p.prim(n,cost);
}

  }
```

OUTPUT:

```
Enter the number of vertices
7
Enter the cost matrix

0       28      999     999     999     10      999

28      0       16      999     999     999     14
```

```
999    16     0      12     999    999    999

999    999    12     0      22     999    18

999    999    999    22     0      25     24

10     999    999    999    25     999    999

999    14     999    18     24     999    999

1edge(1,6)=10
2edge(6,5)=25
3edge(5,4)=22
4edge(4,3)=12
5edge(3,2)=16
6edge(2,7)=14

The minimum cost of spanning tree is 99
```



0	1	2	3	4	5	6	7
1	0	28	∞	∞	∞	10	∞
2	28	0	16	∞	∞	∞	14
3	∞	16	0	12	∞	∞	∞
4	∞	∞	12	0	22	∞	18
5	∞	∞	∞	22	0	25	24
6	10	∞	∞	∞	25	∞	∞
7	∞	14	∞	18	24	∞	∞

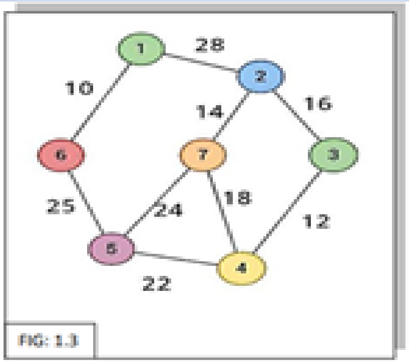
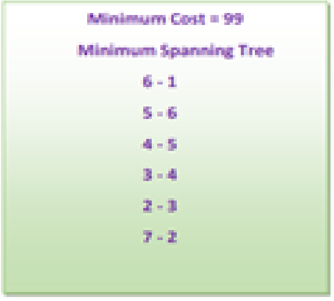


FIG: 1.3



Minimum Cost = 99

Minimum Spanning Tree

6 - 1

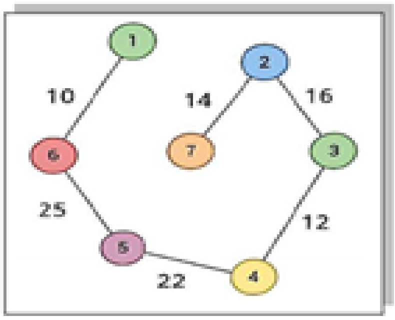
5 - 6

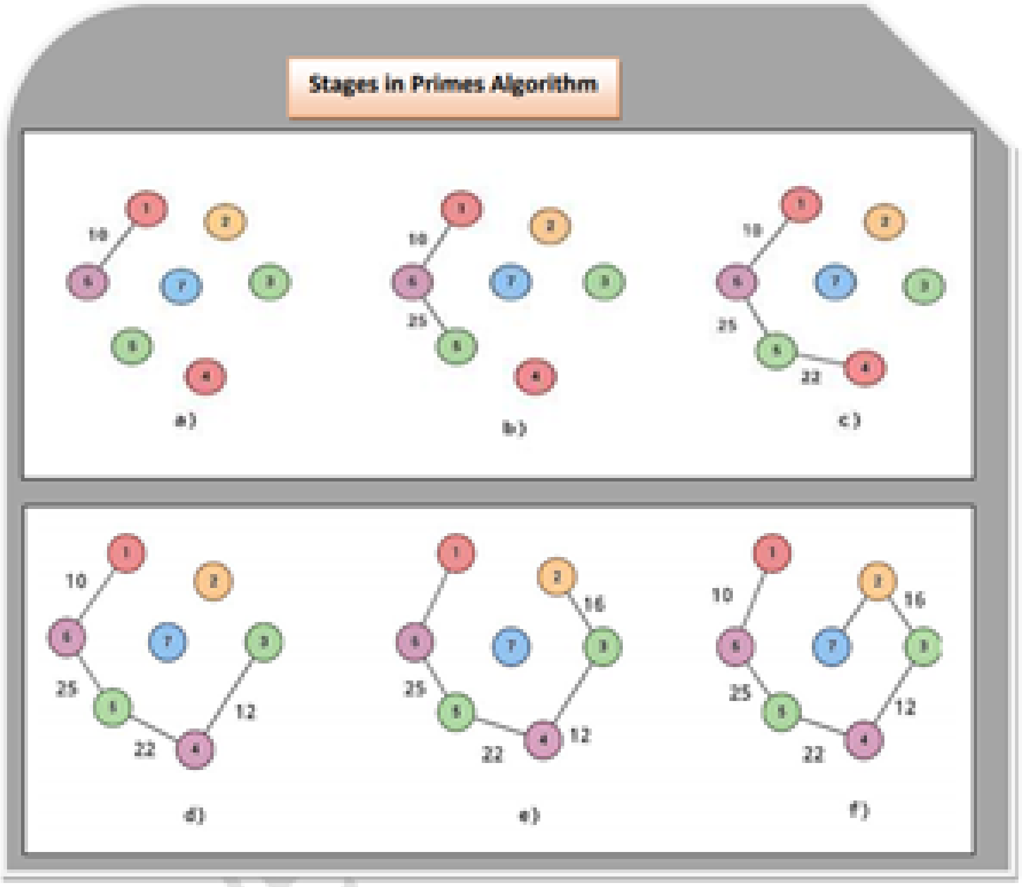
4 - 5

3 - 4

2 - 3

7 - 2




Stages in Primes Algorithm
10
a)
10
25
b)
10
25
22
c)
10
25
22
12
d)
16
25
22
12
e)
10
16
25
12
22
f)

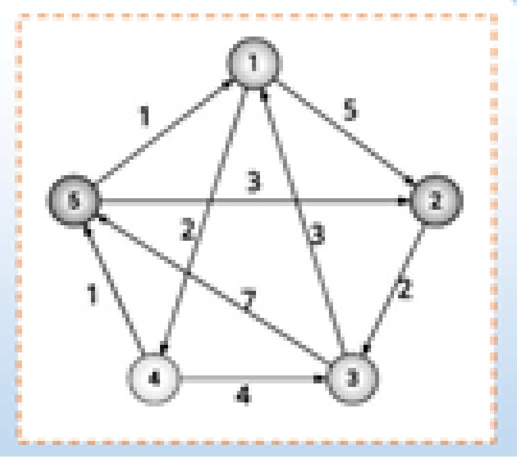
10. Write Java programs to (a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

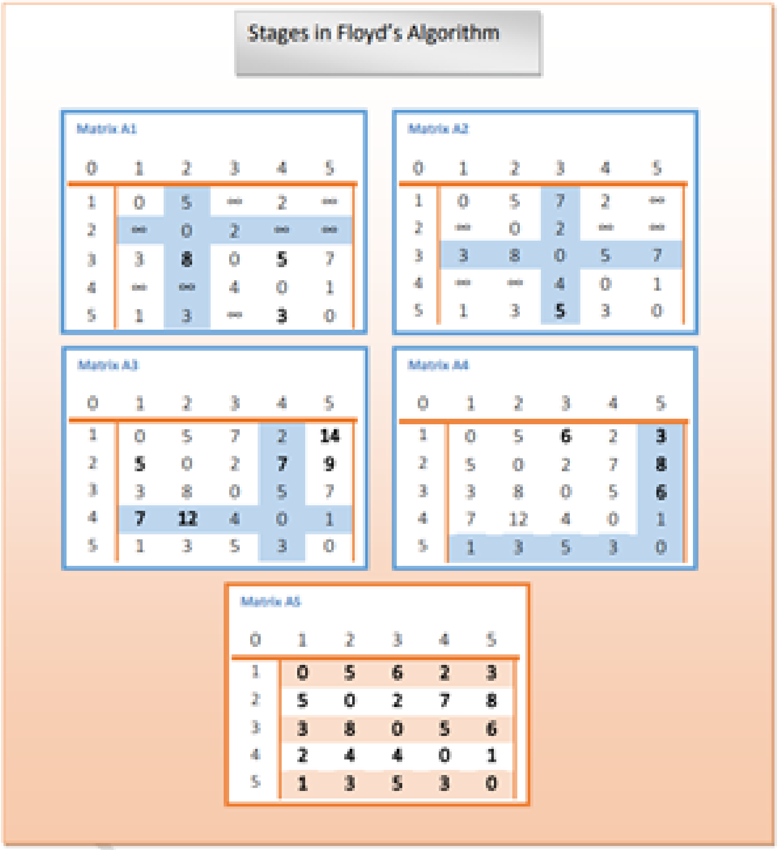
```
import java.util.*;

public class Floyds
{
   static int n,i,j,k;

   public void floyd(int n , int[][] cost)
   {

            for(k=1;k<=n;k++)
            {
                        for(i=1;i<=n;i++)
                        {
                                    for(j=1;j<=n;j++)
                                    {
                                    cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j]);

                                    }
                        }
            }


System.out.println("all pair shortest paths matrix \n");

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

   }

   public int min(int i,int j)
           {
            if(i<j)
                        return i;
            else
                         return j;
```

```
        }

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

  System.out.println("Eneter the no of vertices\n");
  n=sc.nextInt();

  int cost[][]=new int[n+1][n+1];
  System.out.println("Enter the cost matrix:");
  for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
         cost[i][j]=sc.nextInt();

  Floyds f = new Floyds();
  f.floyd(n,cost);

}

}
```

OUTPUT:

```
Enter the number of vertices
5

Enter the cost matrix:
0       5       999     2       999
999     0       2       999     999
3       999     0       999     7
999     999     4        0      1
1       3       999     999     0
all pair shortest paths matrix

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





Stages in Floyd's Algorithm

Matrix A1

0	1	2	3	4	5
1	0	5	∞	2	∞
2	∞	0	2	∞	∞
3	3	**8**	0	**5**	7
4	∞	∞	4	0	1
5	1	3	∞	**3**	0

Matrix A2

0	1	2	3	4	5
1	0	5	7	2	∞
2	∞	0	2	∞	∞
3	3	8	0	5	7
4	∞	∞	4	0	1
5	1	3	**5**	3	0

Matrix A3

0	1	2	3	4	5
1	0	5	7	2	**14**
2	**5**	0	2	**7**	**9**
3	3	8	0	5	7
4	**7**	**12**	4	0	1
5	1	3	5	3	0

Matrix A4

0	1	2	3	4	5
1	0	5	**6**	2	**3**
2	5	0	2	7	**8**
3	3	8	0	5	**6**
4	7	12	4	0	1
5	1	3	5	3	0

Matrix A5

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

(b) Implement Travelling Sales Person problem using Dynamic programming.

```
import java.util.Scanner;

public class Tsp
{
  static int cost[][];

  public int tsp(int[] path,int start,int n)
  {
        int i,j,k,ccost;

        int[] mintour=new int[n+1];
        int[] temp=new int[n+1];


        if(start==n-1)
         return cost[path[n-1]][path[n]]+cost[path[n]][1];

        int mincost=999;

        for(i=start+1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                 temp[j]=path[j];

                        temp[start+1]=path[i];
                         temp[i]=path[start+1];

                         if(cost[path[start]][path[i]]+(ccost=tsp(temp,start+1,n))<mincost)
                         {
                                 mincost=cost[path[start]][path[i]]+ccost;

                                 for(k=1;k<=n;k++)
                                         mintour[k]=temp[k];

                         }
        }

        for(i=1;i<=n;i++)
                path[i]=mintour[i];

        return mincost;
  }



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

  System.out.println("enter the no of cities");
  n=s.nextInt();

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


  System.out.println("Enter the cost matrix");
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
          cost[i][j]=s.nextInt();
          for(i=1;i<=n;i++)
                      path[i]=i;

  Tsp obj = new Tsp();

  mincost=obj.tsp(path,1,n);

  System.out.println("tsp tour");
  for(i=1;i<=n;i++)
          System.out.print(path[i] + "--->");

  System.out.println("1");
  System.out.println("Tourcost=" + mincost);
 }



}
```

OUTPUT:

```
Enter the no of cities
4
```

```
Enter the cost matrix

999   1   3   6
1   999   2   3
3   2   999   1
6   3   1   999

tsp tour
1--->2--->4--->3--->1
Tourcost = 8
```

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

```
import java.util.Scanner;
public class Subset
{
  static int w[],x[],flag,sum,n,total,i,s,k,r;
   public void sumOfSubset(int s,int k,int r)
   {
    x[k]=1;
    if(s+w[k]==sum)
    {
            System.out.println("The subset: ");
            for(i=1;i<=k;i++)
            {
              flag=1;
              if(x[i]==1)
              {
                        System.out.println(w[i]);
              }
            }
    }

    else if(s+w[k]+w[k+1]<=sum)
    {
            sumOfSubset(s+w[k],k+1,r-w[k]);
    }
    if(s+r-w[k]>=sum && s+w[k+1]<=sum)
    {
            x[k]=0;
            sumOfSubset(s,k+1,r-w[k]);
    }
}
public static void main(String args[])
{
    Scanner s=new Scanner(System.in);
    System.out.println("Enter the number of elements");
    n=s.nextInt();
   w=new int[n+1];
   x=new int[n+1];
   System.out.println("Enter the elements");
  for(int i=1;i<=n;i++)
  {
          w[i]=s.nextInt();
          total=total+w[i];
  }
  System.out.println("Enter the sum");
  sum=s.nextInt();
  if(total<sum)
  {
          System.out.println("subset is not possible");
          System.exit(0);
  }
  Subset ss = new Subset();
  ss.sumOfSubset(0,1,total);
  if(flag==0)
  {
          System.out.println("Subset not possible");
  }
}
}
```

OUTPUT:

```
Enter the number of elements

7

Enter the elements

1      2      3      4      5      6      7

Enter the sum

8

The subset:

1

2
```

```
5
The subset:
1
3
4
The subset:
1
7
The subset:
2
6
The subset:
3
5
```

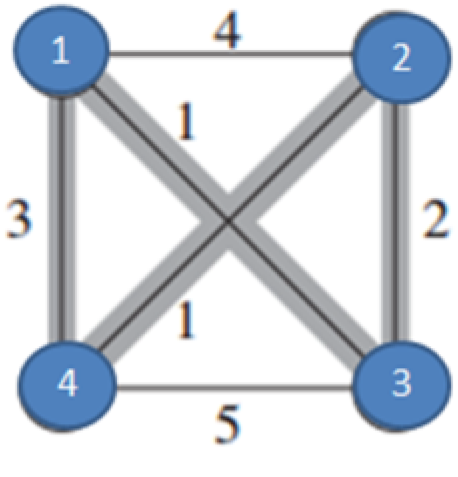
12. Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.

```
import java.util.Scanner;

class HamiltonianCycles
{
  int n,g[][],x[],i,j,k;

  public HamiltonianCycles(int n,int[][] g)
  {
        this.n=n;
        this.g=g;
        this.x = new int[n+1];
        x[1]=1;

  }


  public void hamiltonian(int k)
  {

        while(true)
        {
                nextValue(k);

                if(x[k] == 0)
                {
                        return;

                }

                if(k==n)
                {
                        System.out.println("Solution :");
                        for(int i=1;i<=n;i++)
                        {
                                System.out.print(x[i] + "\t");

                        }

                        System.out.println(1);
                }
        else
        {
            hamiltonian(k+1);
        }

    }
}


public void nextValue(int k)
{

    while(true)
    {
        x[k] = (x[k]+1)%(n+1);

        if(x[k]==0)
        {
            return;
        }

        if(g[x[k-1]][x[k]] != 0)
        {
            for(j=1;j<=k-1;j++)
            {
                if(x[j] == x[k])
                {
                    break;
                }
            }

            if(j==k)
            {
                if((k<n) || ((k==n) && (g[x[n]][x[1]] != 0 )))
                {
                    return;
                }
            }

        }


    }
}
```

```
   public static void main(String[] args)
   {
           int n;
           Scanner s = new Scanner(System.in);

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

           int[][] g = new int[n+1][n+1];
           System.out.println("Enter the matrix :");
           for(int i=1;i<=n;i++)
                   for(int j=1;j<=n;j++)
                           g[i][j]=s.nextInt();

           HamiltonianCycles ham = new HamiltonianCycles(n,g);

           ham.hamiltonian(2);


   }
}
```

OUTPUT:

```
Enter the number of vertices :
4
Enter the matrix :
0    4    1    3
4    0    2    1
1    2    0    5
3    1    5    0
Solution :
1    2    3    4    1
Solution :
1    2    4    3    1
Solution :
1    3    2    4    1
Solution :
1    3    4    2    1
Solution :
1    4    2    3    1
Solution :
1    4    3    2    1
```



$$\begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \left\{ \begin{array}{cccc} 0 & 4 & 1 & 3 \\ 4 & 0 & 2 & 1 \\ 1 & 2 & 0 & 5 \\ 3 & 1 & 5 & 0 \end{array} \right\} \end{array}$$