

**THE VISION OF THE INSTITUTE**

PACE is envisaged as a centre of evolution for excellence in Technological, Management and Research Education. The institution aspires to spread universal knowledge through villages and cities enabling expansion of human resources.

**THE MISSION OF THE INSTITUTE**

- To provide career-oriented professional education to produce technically competent engineers and managers with moral and ethical values.
- To foster and promote an effective learning environment in the campus to be recognized as place that encourages excellence and diversity in thoughts and endeavour.
- To provide research and intellectual resources to address problems facing the industry and the society while advancing the scopes of multidisciplinary applications.

**THE VISION OF THE DEPARTMENT**

To be a center of excellence in providing quality education of global standards and be a pioneer in spreading the knowledge and innovative ideas of Computer Science and Engineering for catering the industrial demands and societal needs.

**THE MISSION OF THE DEPARTMENT**

- To develop competent innovative professionals having effective communication skills with proficiency in real world problem solving with leadership traits.
- To impart quality and value based education to reach the expectations of all the stakeholders through an ever evolving, effective and creative teaching-learning process.
- To undertake salient research works for the establishment of center of excellence in the field of Computer Science and Engineering and multi-disciplinary areas.

**PROGRAM EDUCATIONAL OBJECTIVES (PEOS)**

- PEO1. Graduates of the program will analyze, design and solve problems related to Computer Science and Engineering and possess adaptability to changes in technology by self-learning.
- PEO2. Graduates are provided with an educational foundation that prepares them for leadership roles in their diverse career paths and to pursue higher education.
- PEO3. Graduates of the program are capable to deliver the software product for various real life problems within the scheduled time.
- PEO4. Graduates must develop professional and communication skills for their successful professional career.

**PROGRAM OUTCOMES (PO)**

Engineering Graduates will be able to:

PO1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to solve complex problems in Computer Science & Engineering.

PO2. Problem Analysis: Identify, formulate, review research literature and analyze complex problems in Computer Science and reach to a substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/Development of Solutions: Use Computer Technology to design and develop solutions for complex problems faced in the industry, research and society and develop system components or processes that meet the specified needs with appropriate consideration for the health, safety and security of the public.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including computer based design of experiments, analysis and interpretation of data and synthesis of the information using computing resources to arrive at a valid conclusion.

PO5. Modern tool usage: Develop, select, and/or apply appropriate computing tools, techniques, resources such as programming languages and open/closed source software's including prediction and modeling based tools to solve complex engineering problems with the understanding of its limitations.

PO6. The engineer and society: Apply reasoning based on the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of and need for sustainable development.

PO8. Ethics: Apply ethical principles, commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, as a member or as a leader in diverse teams in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles via apply these to one's own work as a member and leader of a team and to manage projects in multidisciplinary environments. Also extract the skills, experience and expertise of the team members in an efficient and cost effective manner.

PO12. Life-long learning: Recognize the need for and have the preparation and ability to engage in independent life-long learning in the broadest context of technological development in computing

### PROGRAM SPECIFIC OUTCOMES (PSO)

- PSO1. Analyze, design, develop and optimize solutions in Java, C++, .NET Technology and Android based applications.
- PSO2. Apply concepts in core areas of Computer Science – Algorithms and Data Structures, Operating Systems, Database Management Systems, Computer Networks, Computer Architecture and Software Engineering to solve technical issues.

**DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY**  
(Effective from the academic year 2018 -2019)  
**SEMESTER – IV**

Course Code 18CSL47	CIE Marks 40
Number of Contact Hours/Week 0:2:2	SEE Marks 60
Total Number of Lab Contact Hours:36	Exam Hours 03 Credits – 2

**Course Learning Objectives:**

This course (18CSL47) will enable students to:

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

Descriptions (if any):

- Design, develop, and implement the specified algorithms for the following problems using Java language under LINUX /Windows environment. Netbeans / Eclipse or IntelliJIdea Community Edition IDE tool can be used for development and demonstration.
- Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

**Programs List:**

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

- (i) USN
- (ii) Name
- (iii) Programme
- (iv) Phone

Write a Java program to create nStudent objects and print the USN, Name, Programme, 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  $S = \{S_1, S_2, \dots, S_n\}$  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.
- Laboratory Outcomes: The student should be able to:
- Design algorithms using appropriate design techniques (brute-force, greedy, dynamic programming, etc.)
  - Implement a variety of algorithms such as sorting, graph related, combinatorial, etc., in a high level language.
  - Analyze and compare the performance of algorithms using language features.
  - Apply and implement learned algorithm design techniques and data structures to solve real-world problems.
- Conduct of Practical Examination:
- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.

- For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
  - Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
  - Marks Distribution (Courseed to change in accordance with university regulations)
  - e) For laboratories having only one part – Procedure + Execution + Viva-Voce:  $15+70+15 = 100$  Marks
  - f) For laboratories having PART A and PART B
    - Part A – Procedure + Execution + Viva =  $6 + 28 + 6 = 40$  Marks
    - Part B – Procedure + Execution + Viva =  $9 + 42 + 9 = 60$  Marks

## How to Install NetBeans 8.2

### 1.1 How to Install NetBeans on Windows

Step 0: Install JDK

To use NetBeans for Java programming, you need to first install Java Development Kit (JDK). See "JDK - How to Install".

Step 1: Download

Download "NetBeans IDE" installer from <http://netbeans.org/downloads/index.html>. There are many "bundles" available. For beginners, choose the 1st entry "Java SE" (e.g., "netbeans-8.2-javase-windows.exe" 95MB).

Step 2: Run the Installer

Run the downloaded installer.

### 1.2 How to Install NetBeans on Mac OS X

To use NetBeans for Java programming, you need to first install JDK. Read "How to install JDK on Mac".

To install NetBeans:

1. Download NetBeans from <http://netbeans.org/downloads/>. Set "Platform" to "Mac OS X". There are many "bundles" available. For beginners, choose "Java SE" (e.g., "netbeans-8.2-javase-macosx.dmg" 116MB).
2. Double-click the download Disk Image (DMG) file.
3. Double-click the "NetBeans 8.x.mpkg", and follow the instructions to install NetBeans. NetBeans will be installed under "/Applications/NetBeans".

4. Eject the Disk Image (".dmg").

You can launch NetBeans from the "Applications".

Notes: To uninstall NetBeans, drag the "/Applications/NetBeans" folder to trash.

### 1.3 How to Install NetBeans on Ubuntu Linux

To use NetBeans for Java programming, you need to first install JDK. Read "How to install JDK on Ubuntu".

To install NetBeans:

1. Download NetBeans from <http://netbeans.org/downloads/>. Choose platform "Linux (x86/x64)" ⇒ "Java SE". You shall receive a sh file (e.g., "netbeans-7.x-ml-javase-linux.sh") in "~/Downloads".

2. Set the downloaded sh file to executable and run the sh file. Open a Terminal:

3. \$ cd ~/Downloads

4. \$ chmod a+x netbeans-7.x-ml-javase-linux.sh // Set to executable for all (a+x)

\$ ./netbeans-7.x-ml-javase-linux.sh // Run

Follow the instructions to install NetBeans.

To start NetBeans, run the script "netbeans" in the NetBeans' bin directory:

\$ cd netbeans-bin-directory

\$ ./netbeans

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.

```
package student;
```

```
import java.util.Scanner;    /* @author PROF SAKEENA *
```

```
class Stu {
```

```
    private String USN;
```

```
    private String Name;
```

```
    private String Branch;
```

```
    private String Phone;
```

```
    public String getUSN()
```

```
{
```



```
        return USN;
    }

    public String getName()
    {
        return Name;
    }

    public String getBranch() {
        return Branch;
    }

    public String getPhone()
    {
        return Phone;
    }

    public Stu(String usn,String name,String branch,String phone) {
        super();
        USN=usn;
        Name=name;
        Branch= branch;
        Phone=phone;
    }
}

public class Student {
    public static void main(String[] args) {
        Scanner in =new Scanner(System.in);
        Scanner s =new Scanner(System.in);
        System.out.println("enter no of students");
        int n=in.nextInt();
```

```
Stu[] st=new Stu[n];

String usn,name,branch,phone;

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

    System.out.println("enter student details"+(i+1));

    System.out.print("enter student USN");

    usn=s.nextLine();

    System.out.print("enter student NAME");

    name=s.nextLine();

    System.out.print("enter student BRANCH");

    branch=s.nextLine();

    System.out.print("enter student PHONE");

    phone=s.nextLine();

    st[i]=new Stu(usn,name,branch,phone);

}

System.out.println("the student details");

System.out.println("USN\tNAME\tBRANCH \tPHONE");

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

{

System.out.println(st[i].getUSN()+"\t"+st[i].getName()+"\t"+st[i].getBranch()+"\t"+st[i].getPhone());

}

}

}
```

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

```
package stack;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/*@author PROF SAKKEENA */

public class Stack {
    private int top;
    private final int s[];
    Stack(int size) {
        top=-1;
        s =new int[size];
    }
    void pushItem(int item) {
        if(top==s.length-1){
            System.out.println("STACKFULL");
        }
        else
        {
            s[++top]=item;
            System.out.println("pushed item :-"+s[top]);
        }
    }
    int popitem()
    {
        if(top<0)
        {
```

```
        System.out.println("STACK UNDERFLOW");
        return 0;
    }
    else
    {
        System.out.println("popped item :- "+s[top]);
        return s[top--];
    }
}

public void display()
{
    System.out.println("the stack is\n");
    for(int i=0;i<=top;i++)
    {
        System.out.println(s[i]);
    }
}

public static void main(String[] args) throws IOException {
    boolean yes=true;    int choice;
    Stack stk= new Stack(4);
    BufferedReader n = new BufferedReader(new InputStreamReader(System.in));
do{
    System.out.println("press 1 for push,2 for pop,3 for display,4 for exit\nEnter your choice");
    choice=Integer.parseInt(n.readLine());
    switch (choice)
    {
        case 1:System.out.println("enter number to push:-");
```

```
        stk.pushItem(Integer.parseInt(n.readLine()));  
        break;  
    case 2:stk.popitem();  
        break;  
    case 3:stk.display();  
        break;  
    case 4:System.out.println("DONE");  
        yes=false;  
        break;  
    default:System.out.println("invalid choice");  
    }  
}  
while(yes == true);  
}  
}
```

2 A) Design a super class called **Staff** with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely **Teaching** (domain, publications), **Technical** (skills), and **Contract** (period). Write a Java program to read and display at least 3 *staff* objects of all three categories.

```
package staff; /* @author PROF SAKEENA */

class Stf                      // class name Stf
{
    private int StafId;        // declaration of fields
    private String Name;
    private String Phone;
    private long Salary;
    public Stf(int stafid,String name,String phone,long salary)
    {
        StafId=stafid;
        Name=name;
        Phone=phone;
        Salary=salary;
    }
    public void Display()
    {
        System.out.print(StafId+"\t"+Name+"\t"+Phone+"\t"+Salary);
    }
}

class Teaching extends Stf    //inheritance
{
    private String Domain;
    private int Publication;
    public Teaching(int stafid,String name,String phone,long salary,String domain,int publication)
```

```
{
    super(stafid,name,phone,salary);
    Domain=domain;
    Publication=publication;
}

public void Display()
{
    super.Display();
    System.out.print("\t"+Domain+"\t\t"+Publication+"\t\t"+"---"+" \t\t"+"---");
}
}

class Technical extends Stf
{
    private String Skills;
    public Technical(int stafid,String name,String phone,long salary,String skills){
        super(stafid,name,phone,salary);
        Skills=skills;
    }
    public void Display()
    {
        super.Display();
        System.out.print("\t"+"---"+" \t\t---"+" \t\t"+Skills+"\t"+"---");
    }
}

class Contract extends Stf
{
    private int Period;
```

```
public Contract(int stafid,String name,String phone,long salary,int period){
    super(stafid,name,phone,salary);
    this.Period=period;
}

public void Display()
{
    super.Display();
    System.out.print("\t---"+"\t\t---"+"\t\t---"+"\t\t"+Period);
}
}

public class STAFF{
public static void main(String[] args)
{
    Stf stf[]=new Stf[3];
    stf[0]= new Teaching(1,"RAHIL","944955001",25000,"CSE",10);
    stf[1]= new Technical(2,"REEHA","944955002",10000,"system admin");
    stf[2]= new Contract(3,"RIFA","944955003",20000,3);

    System.out.println("STAFFID\tNAME\tPHONE\t\tSALARY\tDOMAIN\t\tPUBLICATION\tSK
ILLS\t\tPERIOD");

    for(int i=0;i<3;i++)
    {
        stf[i].Display();
        System.out.println();
    }
}
}
```



STAFFID	NAME	PHONE	SALARY	DOMAIN	PUBLICATION
	SKILLS	PERIOD			
1	RAHIL	944955001	25000 CSE	10	---
2	REEHA	944955002	10000 ---	---	system admin ---
3	RIFA	944955003	20000 ---	---	---

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 “/”.

```
package customer1;

import java.util.Scanner;

import java.util.StringTokenizer; /* @author PROF.SAKEENA */

public class Customer1 {

    private String custName;

    private String dob;

    public Customer1(String custName,String dob)

    {

        this.custName=custName;

        this.dob=dob;

    }

    public static void main(String[] args) {

        Scanner scanner =new Scanner(System.in);

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

        String custName=scanner.nextLine();

        System.out.println("enter date(dd/mm/yyyy)");

        String dob=scanner.next();

        Customer1 customer=new Customer1(custName,dob);

        System.out.println("Customer name "+customer.custName);

        StringTokenizer date=new StringTokenizer (customer.dob,"/");

        System.out.print("customer DOB: " + date.nextToken()+ "," + date.nextToken() + ","

+date.nextToken());

    }

}
```

Run:

Enter customer name

sakeena

enter date(dd/mm/yyyy)

22/3/2017

Customer name sakeena

customer DOB: 22,3,2017BUILD SUCCESSFUL (total time: 10 seconds)

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.

package divide;

import java.util.Scanner; /\*author PROF SAKREENA \*/

public class Divide {

public static void main(String[] args) {

Scanner s=new Scanner(System.in);

System.out.println("Enter the integers a and b");

int a=s.nextInt();

int b=s.nextInt();

try //try block

{

System.out.println("the result of division of a by b is "+(a/b));

}

catch(ArithmeticException e) //catch block to catch any exceptions

{

System.out.println("division by zero error");

}

}

}

Enter the integers a and b

10 5

the result of division of a by b is 2

BUILD SUCCESSFUL (total time: 8 seconds)

Enter the integers a and b

10 0

division by zero error

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

```
package multithread;
import java.util.Scanner;
/* @author PROF SAKEENA */

public class Multithread {
    public static int num;
    public static int state=0;

    public static void main(String[] args) {
        new Thread(new Runnable(){
            @Override
            public void run(){
                int i=0;
                System.out.println("Enter n");
                Scanner s=new Scanner(System.in);
                int n=s.nextInt();

                while(i<n)
                {
                    num =(int) (Math.random() * 100);
                    System.out.println("\nNumber :"+num);
                    i++;
                    state = 1;
                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException e){
                        System.err.println("error"+e);
                    }
                }
            }
        }).start();

        new Thread(new Runnable(){
            @Override
            public void run(){
                while(true)
                {
                    while(state!= 1);
                    System.out.println("square :"+(num* num));
                    state=2;
                }
            }
        }).start();
    }
}
```

```
    }  
    }  
    ).start();  
  
    new Thread(new Runnable(){  
        @Override  
        public void run(){  
            while(true)  
            {  
                while(state!= 2);  
                System.out.println("cube :"+(num* num*num));  
                state=0;  
            }  
        }  
    }).start();  
    }  
}
```

run:

Enter n

5

Number :27

square :729

cube :19683

Number :82

square :6724

cube :551368

Number :6

square :36

cube :216

Number :33

square :1089

cube :35937

Number :60

square :3600

cube :216000

BUILD STOPPED (total time: 11 seconds)

4) Sort a given set of  $n$  integer elements using **Quick Sort** method and compute its time complexity. Run the program for varied values of  $n > 5000$  and record the time taken to sort.

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

```
import java.util.Random;

import java.util.Scanner; /* @author PROF SAKEENA */

public class Quick {

    private int[] numbers;

    private int size;

    public void sort(int[] values) {

        // check for empty or null array

        /* if (values ==null || values.length==0){

            return;

        }*/

        this.numbers = values;

        size = values.length;

        long startTime =System.currentTimeMillis();

        quicksort(0, size - 1);

        long stopTime =System.currentTimeMillis();

        long elapsedTime= stopTime-startTime;

        System.out.println("Time taken");

        System.out.println(elapsedTime);

    }

    private void quicksort(int low, int high) {

        int j;

        if(low<=high)

        {
```

```
        j=part(low,high);
        quicksort(low,j-1);
        quicksort(j+1,high);
    }
}

private int part(int low,int high)
{
    int i,j,pivot;
    pivot=numbers[low];
    i=low+1;j=high;
    while(true)
    {
        while(i<high && pivot >numbers[i])
            i++;
        while(numbers[j]>pivot)
            j--;
        if (i<j)
        {
            int temp=numbers[i];
            numbers[i]=numbers[j];
            numbers[j]=temp;
        }
        else
        {
            int temp = numbers[low];
            numbers[low]=numbers[j];
            numbers[j]=temp;
        }
    }
}
```



```
        return j;
    }
}
}

public static void main(String[] args) {
    Quick sorter = new Quick();
    int[] numbers;int size;
    Scanner scan =new Scanner(System.in);
    System.out.println("enter size of the array");
    size =scan.nextInt();
    numbers=new int[size];
    System.out.println("the elements of the array");
    Random random = new Random();
    for(int i=0;i<size;i++)
    {
        numbers[i]=Math.abs(random.nextInt(100));

    }
    for(int i=0;i<size;i++)
    {
        System.out.print("\t" + numbers[i]);

    }
    long startTime =System.currentTimeMillis();
    sorter.sort(numbers);
    long stopTime =System.currentTimeMillis();
}
```

```
long elapsedTime= stopTime-startTime;  
System.out.println();  
for(int i:numbers){  
    System.out.print(i);  
    System.out.print(" ");  
}  
System.out.println();  
System.out.println("Time taken");  
System.out.println(elapsedTime);  
}
```

```
}
```

enter size of the array

10

the elements of the array

54    63    43    72    84    63    73    83    37    12

Time taken

0

12    37    43    54    63    63    72    73    83    84

Time taken

1

5) Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n > 5000, and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

```
package merge;

import java.util.Scanner;

import java.util.Random;

/* @author PROF SAKEENA */

public class Merge {

    private int[] numbers;

    private int[] c;

    private int number;

    public void sort(int[] values) {          // check for empty or null array

        if (values == null || values.length == 0) {

            return;

        }

        this.numbers = values;

        number = values.length;

        this.c = new int[number];

        mergesort(0, number - 1);

    }

    private void mergesort(int low, int high) {

        // int i = low, j = high;

        // Get the pivot element from the middle of the list

        if (low < high)

        {

            int mid = (low + (high)) / 2;

            // Recursion
```

```

        mergesort(low, mid);
        mergesort(mid+1, high);
        mergearray( low, mid, high);
    }
}

private void mergearray(int low,int mid,int high)    {
    int i=low; int j=mid+1; int k =low;
    while(i<=mid && j<=high)
    {
        if( numbers[i] < numbers[j])
            c[k++]= numbers[i++];
        else
            c[k++]=numbers[j++];
    }
    while(i<=mid)
        c[k++]=numbers[i++];
    while (j<=high)
        c[k++]=numbers[j++];
    for(i=low;i<=high;i++)
        numbers[i]= c[i];

}

public static void main(String[] args) {
    Merge sorter = new Merge();

    int[] numbers; int number;    int[] c;

    Scanner scan =new Scanner(System.in);

    System.out.println("enter size of the array");

    number =scan.nextInt();

```

```
numbers=new int[number];

System.out.println("The elements of the array");

Random random = new Random();

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

    numbers[i]=Math.abs(random.nextInt(5000));

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

    System.out.print( "\t" + numbers[i]);        System.out.println();

long startTime =System.currentTimeMillis();

    sorter.sort(numbers);

long stopTime =System.currentTimeMillis();

long elapsedTime= stopTime-startTime;

    for(int i:numbers){

        System.out.print(i);    System.out.print(" ");

    }

System.out.println();

System.out.println("Time taken");

System.out.println(elapsedTime);

}

}
```

6 A) Implement in Java, the **0/1 Knapsack** problem using (a) Dynamic Programming method  
package knapsack;

```
import java.util.Scanner;    /* @author PROF. SAKEENA */

public class Knapsack {
    static int max(int a,int b)
    {
        return (a>b)?a:b;
    }
    public static void main(String[] args) {
        int n,i,j,cap;
        int[] p =new int[20];
        int[] w= new int [20];
        int[][] v =new int[10][10];
        System.out.println("enter no of items\n");
        Scanner s=new Scanner(System.in);
        n =s.nextInt();
        for(i=1;i<=n;i++)
        {
            System.out.println("\n enter weights and profit of each object");
            w[i]=s.nextInt();
            p[i]= s.nextInt();

        }
        System.out.println("\ncapacity of knapsack");
        cap=s.nextInt();

        for(i=0;i<=n;i++)
            v[i][0]=0;
        for(i=1;i<=n;i++) // to find the maximum values of item to be placed
            for(j=1;j<=cap;j++)
            {
                if(w[i] > j)
                    v[i][j]=v[i-1][j];
                else
                    v[i][j] = max(v[i-1][j],v[i-1][j-w[i]]+p[i]);
            }
        System.out.println("\n the table is\n");

        for(i=0;i<=n;i++)
        {
            for(j=0;j<=cap;j++)
            {
                System.out.print(v[i][j]+" ");
            }
            System.out.println("\n");
        }
    }
}
```

```

    }
    System.out.println("the maximum profit is\n"+v[n][cap]);
    System.out.println("optimal subset {");          // to find the items placed in the knapsack
    j=cap;
    for(i=n; i>=1; i--)
        if( v[i][j] != v[i-1][j]) // if values not of v[i][j] and v[i-1][j] are not same then pick that
item i
        {
            System.out.println("item\t"+i);
            j=j-w[i];          // reduce the capacity of the knapsack after picking item i
        }
    System.out.println("\t }");

```

```

    }
}
run:
enter no of items
4
enter weights and profit of each object
2 12
enter weights and profit of each object
1 10
enter weights and profit of each object
3 20
enter weights and profit of each object
2 15
capacity of knapsack
5
the table is

0 0 0 0 0 0

0 0 12 12 12 12

0 10 12 22 22 22

0 10 12 22 30 32

0 10 15 25 30 37
the maximum profit is
37
optimal subset {
item    4
item    2
item    1      }

```

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

```
package greedyknapsack; /* @author PROF. SAKEENA */
import java.util.Scanner;
public class Greedyknapsack{
    static void knapsack (int n,float weight[],float profit[],float capacity)
    {
        float x[]= new float [20],tp=0;
        int i,j,u;
        c= (int) capacity;
        for (i=0;i<n;i++)
            x[i]= (float) 0.0;
        for(i=0;i<n;i++)
        {
            if(weight[i] >c )
                break;
            else
            {
                x[i]=(float) 1.0;
                tp = tp + profit[i];
                c=(int)(c-weight[i]);
            }
        }
        if(i<n)
            x[i]= c/weight[i];
        tp=tp+(x[i] * profit[i]);
        System.out.println("the result vector");
        for (i=0;i<n;i++)
            System.out.println( + x[i]);
        System.out.println("the total profit is");
        System.out.println( + tp);
    }
    public static void main(String[] args)
    {
        float weight[]= new float[20];
        float profit[]= new float[20];
        float capacity;
        int num,i,j;
        float ratio[] = new float[20],temp;
        System.out.println("\nEnter the no of objects");
        Scanner s = new Scanner(System.in);
        num =s.nextInt();
        System.out.println("\nEnter weights and profit of each object");
        for(i=0;i<num;i++)
        {
            weight[i]=s.nextInt();
            profit[i]= s.nextInt();
        }
    }
}
```



```

    }
    System.out.println("\ncapacity of knapsack");
    capacity=s.nextInt();
    for(i=0;i<num;i++)
    {
        ratio[i]= profit[i]/weight[i];
    }
    for(i=0;i<num;i++)
    {
        for(j=i+1;j<num;j++)

        {
            if (ratio[i]< ratio[j])
            {
                temp= ratio[j];
                ratio[j]=ratio[i];
                ratio[i]=temp;
                temp= weight[j];
                weight[j]=weight[i];
                weight[i]=temp;
                temp= profit[j];
                profit[j]=profit[i];
                profit[i]=temp;
            }
        }
    }
    knapsack(num,weight,profit,capacity);
}
}
enter the no of objects
4
enter weights and profit of each object
2 12
1 10
3 20
2 15
capacity of knapsack
5
the result vector
1.0
1.0
0.6666667
0.0
the total profit is
38.333336

```

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.*;
public class Dijkstra
{
    public int distance[] = new int[10];
    public int cost[][] = new int[10][10];

    public void calc(int n,int s)
    {
        int flag[] = new int[n+1];
        int i,minpos=1,k,c,minimum;

        for(i=1; i<=n; i++)
        {
            flag[i]=0;
            this.distance[i] = this.cost[s][i];
        }
        c = 2;
        while(c<=n)
        {
            minimum=99;
            for(k=1; k<=n; k++)
            {
                if(this.distance[k] < minimum && flag[k]!=1)
                {
                    minimum = this.distance[i];
                    minpos = k;
                }
            }
            flag[minpos] = 1;
            c++;
            for(k=1;k<=n;k++)
            {
                if(this.distance[minpos]+this.cost[minpos][k] < this.distance[k] && flag[k]!=1 )
                this.distance[k]=this.distance[minpos]+this.cost[minpos][k];
            }
        }
    }

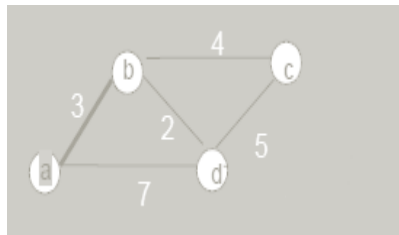
    public static void main(String args[])
    {
        int nodes,source,i,j;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the Number of Nodes \n");
        nodes = in.nextInt();
```

```

Dijkstra d = new Dijkstra();
System.out.println("Enter the Cost Matrix Weights: \n");
for(i=1;i<=nodes;i++)
    for(j=1;j<=nodes;j++)
    {
        d.cost[i][j]=in.nextInt();
        if(d.cost[i][j]==0)
            d.cost[i][j]=999;
    }
System.out.println("Enter the Source Vertex :\n");
source=in.nextInt();
d.calc(nodes,source);
System.out.println("The Shortest Path from Source \t"+source+"\t to all other vertices are : \n");
for(i=1;i<=nodes;i++)
    if(i!=source)
        System.out.println("source :"+source+"\t destination :"+i+"\t MinCost is :"+d.distance[i)+"\t");
}
}

```

Enter the Number of Nodes :4



Enter the Cost Matrix Weights:

```

0   3  999  7
3   0   4   2
999  4   0   5
7   2   5   0

```

Enter the Source Vertex :

1

The Shortest Path from Source Vertex 1 to all other vertices are :

```

source :1    destination :2    MinCost is :3
source :1    destination :3    MinCost is :7
source :1    destination :4    MinCost is :5

```

8) Find Minimum Cost Spanning Tree of a given connected undirected graph using **Kruskal's algorithm**

```
package kruskals;
import java.util.*; /*author PROF. SAKEENA */

public class Kruskals {

    public int parent[] = new int[15];
    public int cost[][] = new int[10][10];
    public int mincost;

    public void calc(int n)
    {
        int flag[] = new int[n+1];
        int i,j,min=999,num_edges=1,a=1,b=1,minpos_i=1,minpos_j=1;
        parent[minpos_i]=0;parent[minpos_j]=0;

        while(num_edges < n)
        {

            for(i=1 , min=999; i<=n; i++)
            for(j=1 ; j<=n; j++)
            if(this.cost[i][j] < min)

                {
                    min = this. cost[i][j];
                    a = minpos_i = i;
                    b = minpos_j = j;
                }

            while(parent[minpos_i]!=0)
                minpos_i=parent[minpos_i];

            while(parent[minpos_j]!=0)
                minpos_j=parent[minpos_j];

            if(minpos_i!= minpos_j)
            {
                System.out.println("\t from Vertex \t"+a+"\t to Vertex \t"+b+"-mincost:"+min+" \n");
                this.mincost=this.mincost+min;
                num_edges=num_edges+1;
                this.parent[minpos_j]=minpos_i;
            }
            this.cost[a][b]=this.cost[b][a]=999;
        }
    }
}
```

```

System.out .println("MINIMUM COST SPANNING TREE (MCST)="+mincost);
    }
    public static void main(String[] args) {
        int nodes,i,j;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the Number of Nodes \n");
        nodes = in.nextInt ();
        Kruskals k = new Kruskals();
        System.out.println ("Enter the Cost Matrix Weights : \n");
        for(i=1; i<=nodes; i++)
            for(j=1; j<=nodes; j++)
            {
                k.cost[i][j] = in.nextInt();
                if(k.cost[i][j] == 0)
                    k.cost[i][j] = 999;
            }

        k.calc(nodes);

    }
}

```

run:

Enter the Number of Nodes

6

Enter the Cost Matrix Weights :

0 3 0 0 6 5

3 0 1 0 0 4

0 1 0 6 0 4

0 0 6 0 8 5

6 0 0 8 0 2

5 4 4 5 2 0

from Vertex 2 to Vertex 3-mincost:1

from Vertex 5 to Vertex 6-mincost:2

from Vertex 1 to Vertex 2-mincost:3

from Vertex 2 to Vertex 6-mincost:4

from Vertex 4 to Vertex 6-mincost:5

MINIMUM COST SPANNING TREE (MCST)=15

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

```
package prim.s;
import java.util.*; /*author PROF. SAKEENA */

public class PrimS {
    public int isVisited[] = new int[15];
    public int cost[][] = new int[10][10];
    public int mincost;

    public void calc(int n)
    {
        int flag[] = new int[n+1];
        int i,j,min=999,num_edges=1,a=1,b=1,minpos_i=1,minpos_j=1;

        while(num_edges < n)
        {

            for(i=1 , min=999; i<=n; i++)
            for(j=1 ; j<=n; j++)
            if(this.cost[i][j] < min)
            if(this.isVisited[i] != 0)
            {
                min = this. cost[i][j];
                a = minpos_i = i;
                b = minpos_j = j;
            }
            if(this. isVisited[minpos_i]== 0 || this.isVisited[minpos_j] == 0)
            {
                System.out.println"\t from Vertex \t"+a+"\t to Vertex \t"+b+"-mincost:"+min+" \n");
                this.mincost=this.mincost+min;
                num_edges=num_edges+1;
                this.isVisited[b]=1;
            }
            this.cost[a][b]=this.cost[b][a]=999;
        }
        System.out .println("MINIMUM COST SPANNING TREE (MCST) = "+mincost);
    }

    public static void main(String args[])
    {
        int nodes,i,j;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the Number of Nodes \n");
        nodes = in.nextInt ();
        PrimS p = new PrimS();
```

```

        System.out.println ("Enter the Cost Matrix Weights : \n");
        for(i=1; i<=nodes; i++)
            for(j=1; j<=nodes; j++)
            {
                p.cost[i][j] = in.nextInt();
                if(p.cost[i][j] == 0)
                    p.cost[i][j] = 999;
            }
        p.isVisited[1] = 1; // Initialization
        p.calc(nodes);
    }
}

```

run:

Enter the Number of Nodes

4

Enter the Cost Matrix Weights :

```

0   3  999  7
3   0   4   2
999  4   0   5
7   2   5   0

```

from Vertex 1 to Vertex 2-mincost:3

from Vertex 2 to Vertex 4-mincost:2

from Vertex 2 to Vertex 3-mincost:4

MINIMUM COST SPANNING TREE (MCST) = 9

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.

```
package floyd;

import java.util.*; /*@author Prof SAKEENA */

public class Floyd {

    public static void main(String[] args) {

        int wt[][]=new int[10][10];

        int n,i,j;

        System.out.print("\n create a graph using adjucency matrix");

        System.out.print("\n enter no of vertices");

        Scanner in= new Scanner(System.in);

        n=in.nextInt();

        System.out.print("enter elements   enter 999 as infinity value");

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

        {

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

            {

                System.out.print("\nwt["+i+""]["+j+"]");

                wt[i][j]=in.nextInt();

            }

        }

        System.out.print("\n\t Computing ALl pairs shortest path..\n");

        Floyd_shortest_path(wt,n);

    }

    public static void Floyd_shortest_path(int wt[][],int n)

    {

        int d[][]=new int[10][10];
```



```
int i,j,k;
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        d[i][j]= wt[i][j];
    }
}
for(k=1;k<=n;k++)
{
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            d[i][j] = min(d[i][j],(d[i][k]+d[k][j]));
        }
    }
}
for(k=0;k<=n;k++)
{
    System.out.print("d("+k+")\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            System.out.print(" "+d[i][j]);
        }
    }
}
```

```

        }
        System.out.print("\n");
    }
}

```

```

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

```

create a graph using adjacency matrix

enter no of vertices3

enter elements enter 999 as infinity value

create a graph using adjacency matrix

enter no of vertices

3

enter elements

enter 999 as infinity value

0 8 5

2 0 999

999 1 0

Computing ALL pairs shortest path..

d(0)

0 6 5

2 0 7

3 1 0

d(1)

0 6 5

2 0 7

3 1 0

d(2)

0 6 5

2 0 7

3 1 0

d(3)

0 6 5

2 0 7

3 1 0

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

```
package tsp;

import java.util.*;

public class TSP{

    static int cost =0;

    public static void main(String[] args)
    {
        int a[][]=new int[10][10];

        int visited[]=new int[10];

        int i,j,n;

        System.out.print("\n enter no of cities");

        Scanner in= new Scanner(System.in);

        n= in.nextInt();

        //create(a,visited,n);

        System.out.println("enter cost matrix");

        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                a[i][j]=in.nextInt();
            }
            visited[i]=0;
        }

        System.out.println("The Path is");

        mincost(a,n,0,visited);

        display();
    }
}
```

```
public static void mincost(int a[][],int n,int city,int visited[])
{
    int i,cityno;
    visited[city]=1;
    System.out.print((city+1)+"->");
    cityno=least(a,visited,n,city);
    if(cityno==999)
    {
        cityno=0;
        System.out.print(" "+(cityno+1));
        cost+=a[city][cityno];
        return;
    }

    mincost(a,n,cityno,visited) ;
}

public static int least(int a[][],int visited[],int n,int c)
{
    int i,minnode=999,min=999,newmin=0;
    for(i=0;i<=n;i++)
    {
        if ( (a[c][i]!=0) && (visited[i]==0) )
            if(a[c][i]< min)
            { min=a[i][0]+a[c][i];
              newmin = a[c][i];
              minnode=i;
            }
    }
}
```

```
        }  
    }  
    if (min!=999)  
        cost+=newmin;  
    return minnode;  
}  
public static void display()  
{  
    System.out.println("\n total cost of tour "+cost);  
}  
}  
eneter no of cities  
4  
enter cost matrix  
0 10 15 20  
5 0 9 10  
6 13 0 12  
8 8 9 0  
The Path is  
1->2->4->3->1  
total cost of tour 35
```

11) Design and implement in Java to find a **subset** of a given set  $S = \{S_1, S_2, \dots, S_n\}$  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.

```
package subset; /* @author prof sakeena */
```

```
public class Subset {

    static int count,d =9;

    static int a[]= new int[10];

    static int w[]={ 1,2,5,6,8};

    public static void main(String[] args) {

        int sum = 0,i,n;

        n = 5;

        System.out.print("\n the set for sum is S= {");

        for(i=0; i<n-1; i++)

            System.out.print(w[i]+",");

        System.out.println(w[i]+"}");

        System.out.println("d=" +d);

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

            sum += w[i];

        if (sum < d)

        {

            System.out.println("there is no solution");

        }

        System.out.println("The solution is ");

        count=0;

        sumsubset(0,0,sum);

    }

    public static void sumsubset(int sum,int index,int remainingsum)

    {
```

```

int i;
a[index]=1;
if (sum+w[index]== d)
{
System.out.println("\n solution ="(++count));
for(i=0;i<=index;i++)
{
if (a[i]== 1)
System.out.print(" "+w[i]);
}
}
else if (sum + w[index] + w[index+1] <= d )
    sumsubset(sum + w[index],index+1,remainingsum-w[index] );
if( (sum + remainingsum - w[index] >= d) && (sum+w[index+1]) <=d)
{
a[index] =0;
sumsubset(sum,(index+1),remainingsum- w[index]);
}
}
}
the set for sum is S= {1,2,5,6,8}
d=9
The solution is
solution =1
1 2 6
solution =2
1 8BUILD SUCCESSFUL (total time: 0 seconds)

```



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

```
package hamiltonian; /* @author PROF SAKEENA */
```

```
import java.util.*;
```

```
public class Hamiltonian {
```

```
    static int MAX = 25;
```

```
    static int vertex[]=new int [MAX];
```

```
    public static void main(String[] args) {
```

```
        int i,j,v1,v2,Edges,n;
```

```
        int G[][] =new int[MAX][MAX];
```

```
        System.out.println("\n Program for hamiltonian cycle");
```

```
        System.out.println("Enter number of vertices of graph");
```

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

```
        n= in.nextInt();
```

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

```
        {
```

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

```
            {
```

```
                G[i][j] =0;
```

```
                vertex[i]=0;
```

```
            }
```

```
        }
```

```
        System.out.println("enter the total no of edges");
```

```
        Edges= in.nextInt();
```

```
        for(i=1;i<=Edges;i++)
```

```
        {
```

```
            System.out.println("enter the edge ");
```

```
            v1=in.nextInt();
```

```
        v2=in.nextInt();
        G[v1][v2]= 1;
        G[v2][v1]= 1;
    }
    vertex[1]=1;
    System.out.println("Hamiltonian cycle");
    H_cycle(G,n,2);
}

public static void Nextvertex(int G[][],int n,int k)
{
    int j;
    while(true)
    {
        vertex[k]= (vertex[k] +1)%(n+1);
        if(vertex[k]==0)
            return;
        if(G[vertex[k-1]][vertex[k]]!=0)
        {
            for(j=1;j<=k-1;j++)//every adjacent vertex
            {
                if(vertex[j] == vertex[k])// not a distinct vertex
                    break;
            }
            if(j==k)// obtain a distinct vertex
            {
                if ((k < n) || (k == n) && (G[vertex[n]][vertex[1]] !=0))
                    return;//return a distinct vertex
            }
        }
    }
}
```

```

        }
    }
}

}

public static void H_cycle(int G[][],int n ,int k)
{
    int i;
    while(true)
    {
        Nextvertex(G,n,k);
        if (vertex[k]==0)
            return;
        if(k==n)
        {
            System.out.print("\n");
            for(i=1;i<=n;i++)
                System.out.print(vertex[i]+"->");
            System.out.print(""+vertex[1]);
        }
        else
            H_cycle(G,n,k+1);
    }
}

```

Program for hamiltonian cycle    Enter number of vertices of graph

6

enter the total no of edges

10

enter the edge

1 2

enter the edge

1 3

enter the edge

1 6

enter the edge

2 4

enter the edge

2 6

enter the edge

3 4

enter the edge

3 5

enter the edge

3 6

enter the edge

4 5

enter the edge

4 6

Hamiltonian cycle

1->2->4->5->3->6->1

1->2->6->4->5->3->1

1->3->5->4->2->6->1

1->3->5->4->6->2->1

1->6->2->4->5->3->1

1->6->3->5->4->2->1