K.S Institute of Technology, Bangalore-109

"DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY"

Subcode: 18CSL47

Prepared by:-

Mr. K.Venkata Rao
Associate Professor
Dept of CSE, KSIT

Mrs. Vijayalaxmi Mekali
Assistant Professor
Dept of CSE, KSIT

Mr. Raghavendrachar S
Assistant Professor
Dept of CSE, KSIT

Mrs. Ranjitha.K.N
Assistant Professor
Dept of CSE, KSIT

Mrs. Mamatha. R
Assistant Professor
Dept of CSE, KSIT

Department of Computer Science & Engineering
K.S Institute of Technology, Bangalore-109.

K. S. INSTITUTE OF TECHNOLOGY

#14, Raghuvanahalli, Kanakapura Main Road, Bengaluru-560109

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course: DESIGN AND ANALYSIS OF ALGORITHM LABORATORY		
Type: Core	**Course Code: 18CSL47**	**Academic Year: 2019-2020**

No of Hours per week			
Theory (Lecture Class)	Practical/Field Work/Allied Activities	Total/Week	Total teaching hours
0	3	3	40 hours

Marks			
Internal Assessment	Examination	Total	Credits
40	60	100	3

Aim/Objective of the Course:

1. Get the practical exposures to design, develop, and implement the specified algorithms for the following problems using Java language under LINUX /Windows environment.
2. Make use of Netbeans/Eclipse IDE tool be used for development and demonstration.
3. Illustrate solutions to Lab programs.

Course Learning Outcomes:

After completing the course, the students will be able to create, insert, manipulate and querying different types of database and implement simple database data base applications.

CO1	**Experiment** with object oriented concepts of JAVA programming language.	**Applying** (K3)
CO2	**Construct** the JAVA program by using the approach of Divide and Conquer such as Merge Sort, Quick Sort.	**Applying** (K3)
CO3	**Make use of** Greedy method to solve knapsack and minimum cost spanning tree using JAVA programming.	**Applying** (K3)
CO4	**Apply** Dynamic Programming techniques to solve All pair's shortest path (Floyd's algorithm) and Travelling sales person (TSP) problem using JAVA programming.	**Applying** (K3)
CO5	**Choose** the Backtracking techniques to solve Sum of subset problem and Hamiltonian cycles using JAVA programming.	**Applying** (K3)

Institution Vision & Mission

Vision: **“**To strive continuously to impart quality technical education with ethical values, employable skills and research of global standards”

Mission:

- To attract and retain highly qualified, experienced and committed Faculty.
- To create relevant infrastructure of global standard.
- Network with industry and premier institutions to encourage emergence of new ideas by providing Research and Development Facilities to achieve excellence.
- To inculcate the professional and ethical values among young students so that, they utilize the knowledge and skills acquired in transforming society.

Department Vision & Mission

Vision: “To provide competent and responsible professionals in the field of Computer Science and Engineering with knowledge and skills required for country in its quest for development.”

Mission:

- Inculcate strong theoretical and practical knowledge for continuous learning.
- Prepare students to find Computer Solutions for the society through research and entrepreneurship with professional ethics.
- Encourage team work in inter-disciplines and evolve as leaders with social concerns.

DESIGN AND ANALYSIS OF ALGORITHM LABORATORY
[As per Choice Based Credit System (CBCS) scheme]
SEMESTER - IV

Subject Code:	**18CSL47**	IA Marks:	40
Number of Lecture Hours/Week:	01 I + 02 P	Exam Marks:	60
Total Number of Lecture Hours:	40	Exam Hours:	03

CREDITS - 02

Course objectives: This course 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.

Example programs are given from page 3 – 18 and viva questions are attached at last.

Exp No.	Part	Description	Page No.
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.	19
	B	Write a Java program to implement the Stack using arrays. Write Push(), Pop(), and Display() methods to demonstrate its working.	21
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.	24
	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 "/".	28
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.	29
	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.	30
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.	32

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.	38
6	Implement in Java, the **0/1 Knapsack** problem using (a) Dynamic Programming method (b) Greedy method.	44
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.	50
8	Find Minimum Cost Spanning Tree of a given connected undirected graph using **Kruskal's algorithm.** Use Union-Find algorithms in your program.	54
9	Find Minimum Cost Spanning Tree of a given connected undirected graph using **Prim's algorithm**.	57
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.	61 63
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.	65
12	Design and implement in Java to find all **Hamiltonian Cycles** in a connected undirected Graph G of *n* vertices using backtracking principle.	68

Course Outcomes: The students should be able to:

- Design algorithms using appropriate design techniques (brute-force, greedy, dynamic programming, etc.)
- Implement a variety of algorithms such assorting, 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.

Graduate Attributes

- Engineering Knowledge
- Problem Analysis
- Modern Tool Usage
- Conduct Investigations of Complex Problems
- Design/Development of Solutions

Conduction of Practical Examination:

All laboratory experiments (Twelve problems) are to be included for practical examination. Students are allowed to pick one experiment from the lot.
To generate the data set use random number generator function.
Strictly follow the instructions as printed on the cover page of answer script for breakup of marks
Marks distribution: Procedure + Conduction + Viva: 15 + 70 + 15 (100). Change of experiment is allowed only once and marks allotted to the procedure

JAVA example programs:

Example 1

JAVA program to print “Welcome to KSIT”.

CODE :

```
public class Example1 {

      public static void main(String[] args) {

            System.out.println("Welcome to JAVA");

      }

}
```

OUTPUT :

```
Welcome to JAVA
```

Example 2

JAVA program to print Name, College and Place.

CODE :

```
public class Example2 {

      public static void main(String[] args) {
            System.out.println("Name : Raghu");
            System.out.println("College : KSIT");
            System.out.println("Place : Bengaluru");

      }

}
```

OUTPUT :

```
Name : Raghu
College : KSIT
Place : Bengaluru
```

Example 3

JAVA program to accept emp_name, emp_id, emp_salary and print them.

CODE :

```
import java.util.Scanner;

public class Example3 {

      public static void main(String[] args) {
            int emp_id;
            String emp_name;
            double emp_salary;
            Scanner read = new Scanner(System.in);
            System.out.println("Enter the employee name");
            emp_name=read.nextLine();
            System.out.println("Enter the employee ID");
            emp_id=read.nextInt();
            System.out.println("Enter the employee salary");
            emp_salary=read.nextDouble();
            System.out.println("\nEntered details are");
            System.out.println("Employee Name : "+emp_name);
            System.out.println("Employee ID : "+emp_id);
            System.out.println("Employee Salary : "+emp_salary);
      }

}
```

OUTPUT :

```
Enter the employee name
Kushal
Enter the employee ID
46
Enter the employee salary
87000.50

Entered details are
Employee Name : Kushal
Employee ID : 46
Employee Salary : 87000.5
```

Example 4

JAVA program to perform mathematical operations on an expression.

CODE :

```
import java.util.Scanner;

public class Example4 {

	public static void main(String[] args) {
		double a,b,res=0;
		char c;
		Scanner read = new Scanner(System.in);
		System.out.println("Enter the expression  ");
		a=read.nextDouble();
		c=read.next().charAt(0);
		b=read.nextDouble();
		switch(c)
		{
			case '+' :
				res=a+b;
				break;
			case '-' :
				res=a-b;
				break;
			case '*' :
				res=a*b;
				break;
			case '/' :
				if(b==0)
				{
					System.out.println("Divide by zero error");
					System.exit(0);
				}
				else
					res=a/b;
				break;
			default :
				System.out.println("Invalid Entry");
				System.exit(0);
		}
		System.out.println("Result is : "+res);
	}
}
```

OUTPUT :

```
Enter the expression
4 - 12
Result is : -8.0
```

```
Enter the expression
4.5 / 3
Result is : 1.5
```

```
Enter the expression
5 / 0
Divide by zero error
```

Example 5

JAVA program to read and print array elements.

CODE :

```
import java.util.Scanner;

public class Example5 {

      public static void main(String[] args) {
            int []a = new int[10];
            Scanner read = new Scanner(System.in);
            System.out.println("Enter the array elements");
            for(int i=0;i<a.length;i++)
                  a[i]=read.nextInt();
            System.out.println("Entered array elements : ");
            for(int i=0;i<a.length;i++)
                  System.out.print(a[i]+"\t");
      }

}
```

OUTPUT :

```
Enter the array elements
1 3 2 4 6 5 8 7 9 0
Entered array elements :
1     3     2     4     6     5     8     7     9     0
```

Example 6

JAVA program to print array elements.

CODE :

```
public class Example6 {

      public static void main(String[] args) {
            int []a = {1,2,3,4,5};
            System.out.println("Entered array elements are");
            for(int i=0;i<a.length;i++)
                  System.out.print(a[i]+"\t");
      }
}
```

OUTPUT :

```
Entered array elements are
1     2     3     4     5
```

Example 7

JAVA program to read ***n*** array elements and print the sum of array elements.

CODE :

```
import java.util.Scanner;

public class Example7 {

      public static void main(String[] args) {
            Scanner read = new Scanner(System.in);
            int n,sum=0;
            System.out.println("Enter the size of array ");
            n=read.nextInt();
            int []a=new int[n];
            System.out.println("Enter the array elements");
            for(int i=0;i<a.length;i++)
                  a[i]=read.nextInt();
            for(int i=0;i<a.length;i++)
                  sum+=a[i];
            System.out.println("Sum of array elements is "+sum);
      }
}
```

OUTPUT :

```
Enter the size of array
4
Enter the array elements
2 6 7 22
Sum of array elements is 37
```

Example 8

JAVA program to read and print 2 x 2 matrix.

CODE :

```
import java.util.Scanner;

public class Example8 {

      public static void main(String[] args) {
            Scanner read = new Scanner(System.in);
            int [][]a = new int[2][2];
            System.out.println("Enter the matrix");
            for(int i=0;i<2;i++)
                  for(int j=0;j<2;j++)
                        a[i][j]=read.nextInt();
            System.out.println("Entered matrix is ");
            for(int i=0;i<2;i++)
            {
                  for(int j=0;j<2;j++)
                        System.out.print(a[i][j]+"\t");
                  System.out.println();
            }
      }
}
```

OUTPUT :

```
Enter the matrix
34    -12
44    90
Entered matrix is
34    -12
44    90
```

Example 9

JAVA program to read and print M x N matrix.

CODE :

```
import java.util.Scanner;

public class Example9 {

      public static void main(String[] args) {
            Scanner read = new Scanner(System.in);
            System.out.println("Enter the number of rows");
            int m = read.nextInt();
            System.out.println("Enter the number of columns");
            int n = read.nextInt();
            int [][]a = new int[m][n];
            System.out.println("Enter the matrix");
            for(int i=0;i<m;i++)
                  for(int j=0;j<n;j++)
                        a[i][j]=read.nextInt();
            System.out.println("Entered matrix is ");
            for(int i=0;i<m;i++)
            {
                  for(int j=0;j<n;j++)
                        System.out.print(a[i][j]+"\t");
                  System.out.println();
            }

      }
}
```

OUTPUT :

```
Enter the number of rows
3
Enter the number of columns
3
Enter the matrix
12    56    4
7     13    -3
77    14    6
Entered matrix is
12    56    4
7     13    -3
77    14    6
```

Example 10

JAVA program to print 2 x 2 matrix.

CODE :

```
public class Example10 {

      public static void main(String[] args) {
            int [][]a = {{1,2},{3,4}};
            System.out.println("Entered matrix is ");
            for(int i=0;i<2;i++)
            {
                  for(int j=0;j<2;j++)
                        System.out.print(a[i][j]+"\t");
                  System.out.println();
            }
      }
}
```

OUTPUT :

```
Entered matrix is
1     2
3     4
```

Example 11

JAVA program to find the biggest element in the given M x N matrix.

CODE :

```
import java.util.Scanner;
public class Example11
{
	public static void main(String[] args)
	{
		Scanner read=new Scanner(System.in);
		System.out.println("Enter the number of rows");
		int m=read.nextInt();
		System.out.println("Enter the number of columns");
		int n=read.nextInt();
		int[][] a=new int[m][n];
		System.out.println("Enter the matrix");
		for(int i=0;i<m;i++)
			for(int j=0;j<n;j++)
				a[i][j]=read.nextInt();
		int big=a[0][0];
		for(int i=0;i<m;i++)
		{
			for(int j=0;j<n;j++)
			{
				if(a[i][j]>big)
					big=a[i][j];
			}
		}
		System.out.println("The biggest element in the matrix is
		"+big);
	}
}
```

OUTPUT :

```
Enter the matrix
4     8
3     9
The biggest element in the matrix is 9
```

Example 12

JAVA program to print volume using an object.

CODE :

```
class Box1
{
      double width,height,depth;
}
public class Example12
{
      public static void main(String[] args)
      {
            Box1 b1=new Box1();
            b1.height=10;
            b1.width=20;
            b1.depth=30;
            double vol=b1.height* b1.width* b1.depth;
            System.out.println("Volume is "+vol);
      }
}
```

OUTPUT :

```
Volume is 6000.0
```

Example 13

JAVA program to print volume.

CODE :

```
class Box2
{
	double width,height,depth;
	void setDim(double w, double h, double d)
	{
		width=w;
		height=h;
		depth=d;
	}
	double volume()
	{
		return width*height*depth;
	}
}
public class Example13
{
	public static void main(String[] args)
	{
		Box2 mybox=new Box2();
		double vol;
		mybox.setDim(10,20,15);
		vol=mybox.volume();
		System.out.println("Volume is "+vol);
	}
}
```

OUTPUT :

```
Volume is 3000.0
```

Example 14

JAVA program to show the use of constructor.

CODE :

```
class Box3
{
	double width,height,depth;
	Box3()
	{
		System.out.println("Constructing Box");
		width=10;
		height=10;
		depth=10;
	}
	double volume()
	{
		return width*height*depth;
	}
}
public class Example14
{
	public static void main(String[] args)
	{
		Box3 mybox=new Box3();
		double vol;
		vol=mybox.volume();
		System.out.println("Volume is "+vol);
	}
}
```

OUTPUT :

```
Constructing Box
Volume is 1000.0
```

Example 15

JAVA program to show the use of ‘this’.

CODE :

```
class Box4
{
	double width,height,depth;
	Box4(double width, double height, double depth)
	{
		this.width=width;
		this.height=height;
		this.depth=depth;
	}
	double volume()
	{
		return width*height*depth;
	}
}
public class Example15
{
	public static void main(String[] args)
	{
		Box4 b1=new Box4(10,20,30);
		double vol=b1.volume();
		System.out.println("Volume is "+vol);
	}
}
```

OUTPUT :

```
Volume is 6000.0
```

Example 16

JAVA program to show the concept of inheritance.

CODE :

```
class Base
{
	int i,j;
	void showBase()
	{
		System.out.println(i+" "+j);
	}
}
class Derived extends Base
{
	int k;
	void showDerived()
	{
		System.out.println(k);
	}
}
public class Example16
{
	public static void main(String[] args)
	{
		Base a=new Base();
		Derived b=new Derived();
		a.i=10;
		a.j=20;
		a.showBase();
		b.i=30;
		b.k=50;
		b.j=40;
		b.showBase();
		b.showDerived();
	}
}
```

OUTPUT :

```
10 20
30 40
50
```

Example 17

JAVA program to print area and volume.

CODE :

```
class Room
{
	int length,breadth,a;
	Room(int x,int y)
	{
		length=x;
		breadth=y;
	}
	void area()
	{
		a=(length*breadth);
		System.out.println("Area: "+a);
	}
}
class Room1 extends Room
{
	int height,vol;
	Room1(int x,int y,int z)
	{
		super(x,y);
		height=z;
	}
	void volume()
	{
		vol=(length*breadth*height);
		System.out.println("Volume: "+vol);
	}
}
public class Example17
{
	public static void main(String[] args)
	{
		Room1 r1=new Room1(4,5,6);
		r1.area();
		r1.volume();
	}
}
```

OUTPUT :

```
Area: 20
Volume: 120
```

Example 18

JAVA program to show the use of ‘super’.

CODE :

```
class A
{
      int i=20;
}
class B extends A
{
      int i=30;
      void display()
      {
            System.out.println(i);
            System.out.println(super.i);
      }
}
public class Example18
{
      public static void main(String[] args)
      {
            B b1=new B();
            b1.display();
      }
}
```

OUTPUT :

```
30
20
```

Lab Programs :

Program 1.a

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

- USN
- Name
- Branch
- Phone

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

CODE :

```
import java.util.Scanner;
class Student
{
	String usn,name,branch,phone;
	Student(String usn,String name,String branch,String phone)
	{
		this.usn=usn;
		this.name=name;
		this.branch=branch;
		this.phone=phone;
	}
	void display()
	{
		System.out.println(usn+"\t\t"+name+"\t\t"+branch+"\t\t"+phone);
	}
}
public class P1a {

	public static void main(String[] args) {
		Scanner read=new Scanner(System.in);
		Scanner read1=new Scanner(System.in);
		System.out.println("Enter the number of students:");
		int n=read.nextInt();
		Student []S=new Student[n];
		for(int i=0;i<n;i++)
		{
			System.out.println("Enter details of student-->"+(i+1));
			System.out.println("USN:");
			String usn=read.next();
			System.out.println("Name:");
			String name=read1.nextLine();
			System.out.println("Branch:");
			String branch=read.next();
			System.out.println("Phone:");
			String phone=read.next();
```

```
                S[i]=new Student(usn,name,branch,phone);
            }
            System.out.println("Student details  are:");
            System.out.println("------------------------------------------------------------
------------------------------------");
            System.out.println("USN\t\t\tName\t\t\tBranch\t\t\tPhone");
            System.out.println("------------------------------------------------------------
-----------------------------------");
            for(int i=0;i<n;i++)
            {
                S[i].display();
            }
        }

}
```

Output:

```
Enter the number of students:
2
Enter details of student-->1
USN:
1KS15CS005
Name:
Adithi
Branch:
CSE
Phone:
9742507400
Enter details of student-->2
USN:
1KS15CS018
Name:
Brunda
Branch:
CSE
Phone:
9900567891
Student details are:
-----------------------------------------------------------------------------------
USN                  Name                   Branch                  Phone
-----------------------------------------------------------------------------------
1KS15CS005       Adithi                   CSE              9742507400
1KS15CS018       Brunda                   CSE              9900567891
```

Program 1.b

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

CODE :

```
import java.util.Scanner;
class Stack
{
      int []Stack;
      int top=-1,size;
      Stack(int size)
      {
            this.size=size;
            Stack=new int[size];
      }
      void push()
      {
            if(top==size-1)
                  System.out.println("Stack overflow");
            else
            {
                  Scanner read=new Scanner(System.in);
                  System.out.println("Enter the element to be inserted:");
                  int element=read.nextInt();
                  Stack[++top]=element;
            }
      }

      void pop()
      {
            if(top==-1)
                  System.out.println("Stack underflow");
            else
                System.out.println("Popped element is:"+Stack[top--]);
      }
      void display()
      {
            if(top==-1)
                  System.out.println("Stack is empty");
            Scanner read=new Scanner(System.in);
            System.out.println("The elements of Stack are:");
            for(int i=top;i>=0;i--)
                  System.out.println(Stack[i]);
      }
}
public class P1b {

      public static void main(String[] args) {
            Scanner read=new Scanner(System.in);
            System.out.println("Enter the size of stack:");
            int size=read.nextInt();
            Stack obj=new Stack(size);
            for(;;)
            {
                  System.out.println("Stack operations:");
                  System.out.println("1-->Push");
                  System.out.println("2-->Pop");
                  System.out.println("3-->Display");
                  System.out.println("4-->Exit");
                  System.out.println("Enter your choice:");
                  int  choice=read.nextInt();
                  switch(choice)
                  {
                  case 1:
                        obj.push();
                        break;
                  case 2:
                        obj.pop();
                        break;
                  case 3:
                        obj.display();
                        break;
                  case 4:System.exit(0);
                  }
            }
      }
}
```

Output:

```
Enter the size of stack:
4
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
1
Enter the element to be inserted:
20
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
1
Enter the element to be inserted:
30
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
1
Enter the element to be inserted:
50
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
1
Enter the element to be inserted:
81
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
1
Stack overflow
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
3
The elements of Stack are:
81
50
30
20
```

```
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
2
Popped element is:81
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
2
Popped element is:50
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
2
Popped element is:30
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
2
Popped element is:20
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
2
Stack underflow
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
3
Stack is empty
The elements of Stack are:
Stack operations:
1-->Push
2-->Pop
3-->Display
4-->Exit
Enter your choice:
4
```

Program 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.

CODE :

```
import java.util.Scanner;
class Staff
{
	String sid,name,ph;
	double sal;
	Scanner read=new Scanner(System.in);
	Scanner read1=new Scanner(System.in);

	void read()
	{
		System.out.println("Staff id:");
		sid=read.next();
		System.out.println("Name:");
		name=read1.nextLine();
		System.out.println("Phone:");
		ph=read.next();
		System.out.println("Salary:");
		sal=read.nextDouble();
	}

	void display()
	{
		System.out.println("Staff id: "+sid);
		System.out.println("Name: "+name);
		System.out.println("Phone: "+ph);
		System.out.println("Salary: "+sal);
	}
}

class Teaching extends Staff
{
	String Domain;
	int publication;

	void read()
	{
		super.read();
		System.out.println("Domain:");
		Domain=read1.nextLine();
		System.out.println("Publication:");
		publication=read.nextInt();
	}
```

```
      void display()
      {
            super.display();
            System.out.println("Domain: "+Domain);
            System.out.println("Publications: "+publication);
      }
}

class Technical extends Staff
{
      String skills;

      void read()
      {
            super.read();
            System.out.println("Skills:");
            skills=read1.nextLine();
      }

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

class Contract extends Staff
{
      int period;

      void read()
      {
            super.read();
            System.out.println("Period:");
            period=read.nextInt();
      }

      void display()
      {
            super.display();
            System.out.println("Period: "+period);
      }
}
public class P2a {

      public static void main(String[] args) {

            Teaching teach=new Teaching();
            Technical tech=new Technical();
            Contract cont=new Contract();

            System.out.println("Enter teaching staff details:");
            teach.read();
            System.out.println("\n\nEnter technical staff details:");
            tech.read();
            System.out.println("\n\nEnter Contract staff details:");
            cont.read();

            System.out.println("\n\nHere are teaching staff details:");
            System.out.println("-----------------------------------------------------------
            ");
            teach.display();
            System.out.println("\n\nHere are technical staff details:");
            System.out.println("-----------------------------------------------------------
            ");
            tech.display();
            System.out.println("\n\nHere are contract staff details:");
            System.out.println("-----------------------------------------------------------
            ");
            cont.display();
      }
}
```

Output:

```
Enter teaching staff details:
Staff id:
KS01
Name:
K Venkata Rao
Phone:
9901234567
Salary:
800000
Domain:
Formal languages and Automata
Theory
Publication:
8


Enter technical staff details:
Staff id:
KS06
Name:
Asar
Phone:
8098765432
Salary:
700000
Skills:
C++, Java, Python


Enter Contract staff details:
Staff id:
KS18
Name:
Amruth
Phone:
7708347692
Salary:
750000
Period:
5
```

```
Here are teaching staff details:
-----------------------------------------------------------
Staff id : KS01
Name : K Venkata Rao
Phone : 9901234567
Salary : 800000.0
Domain : Formal languages and Automata
Theory
Publications : 8


Here are technical staff details:
-----------------------------------------------------------
Staff id: KS06
Name: Asar
Phone: 8098765432
Salary: 700000.0
Skills: C++, Java, Python


Here are contract staff details:
-----------------------------------------------------------
Staff id: KS18
Name: Amruth
Phone: 7708347692
Salary: 750000.0
Period: 5
```

Program 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 "/".

CODE :

```
import java.util.*;
class Customer
{
      String data,name,dob;
      void read()
      {
            Scanner read=new Scanner(System.in);
            System.out.println("Enter data in the form <name,dd/mm/yyyy>");
            data=read.nextLine();
      }
      void display()
      {
            String []S=data.split(",");
            name=S[0];
            dob=S[1];
            StringTokenizer str=new StringTokenizer(dob,"/");
            System.out.println(name+","+str.nextToken()+","+str.nextToken()
            +","+str.nextToken());
      }
}
public class P2b {

      public static void main(String[] args) {

            Customer obj=new Customer();
            obj.read();
            obj.display();
      }
}
```

Output:

```
Enter data in the form <name,dd/mm/yyyy>
Arjit, 25/04/1987
Arjit, 25,04,1987
```

```
Enter data in the form <name,dd/mm/yyyy>
Mahatma Gandhi, 02/10/1869
Mahatma Gandhi, 02,10,1869
```

Program 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.

CODE :

```
import java.util.*;
public class P3a {
      public static void main(String[] args) {
            int a,b;
            float res;
            Scanner read=new Scanner(System.in);
            System.out.println("Enter the values:");
            a=read.nextInt();
            b=read.nextInt();
            try
            {
                  if(b==0)
                        throw new ArithmeticException("Divide by zero
                        error");
                  res= (float)a/b;
                  System.out.println("Result is:"+res);
            }
            catch(Exception e)
            {
                  System.out.println(e);
            }
      }

}
```

Output:

```
Enter the values:
5 3
Result is:1.6666666
```

```
Enter the values:
6 0
java.lang.ArithmeticException: Divide by zero error
```

Program 3.b

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

CODE :

```
import java.util.*;
class MyThread1 extends Thread
{
	public void run()
	{
		int i=1;
		Scanner read=new Scanner(System.in);
		System.out.println("Enter the number of random numbers to be
		generated:");
		int n=read.nextInt();
		try
		{
			while(i<=n)
			{
				Random random=new Random();
				P3b.r=random.nextInt(100);
				System.out.println(i+"--> random integer generated
				is: "+P3b.r);
				new MyThread2().start();
				new MyThread3().start();
				Thread.sleep(1000);
				System.out.println("\n\n");
				i++;
			}
		}
		catch(Exception e)
		{
			System.out.println(e);
		}
	}
}
class MyThread2 extends Thread
{
	public void run()
	{
		System.out.println("Square of " +P3b.r +" is equal to
		"+(P3b.r*P3b.r));
	}
}
```

```
class MyThread3 extends Thread
{
	public void run()
	{
		System.out.println("Cube of " +P3b.r +" is equal to
		"+P3b.r*P3b.r*P3b.r);
	}
}
public class P3b {
	static int r;
	public static void main(String[] args) {

		MyThread1 thread1=new MyThread1();
		thread1.start();

	}

}
```

Output:

```
Enter the number of random numbers to be generated:
4
1--> random integer generated is: 68
Square of 68 is equal to 4624
Cube of 68 is equal to 314432



2--> random integer generated is: 96
Square of 96 is equal to 9216
Cube of 96 is equal to 884736



3--> random integer generated is: 25
Square of 25 is equal to 625
Cube of 25 is equal to 15625



4--> random integer generated is: 70
Square of 70 is equal to 4900
Cube of 70 is equal to 343000
```

Program 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.

ALGORITHM :

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

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

CODE :

```
import java.util.*;
import java.io.*;
public class P4 {

      static int [] a;
      static int n;
      static boolean flag = true;

      public static void quicksort(int [] a,int low,int high)
      {
            int i=low,j=high;
            int temp;
            int pivot = a[(low+high)/2];

            if(flag)
            {
                  while(i<=j)
                  {
                        while(a[i]<pivot)
                              i++;

                        while(a[j]>pivot)
                              j--;

                        if(i<=j)
                        {
                              temp=a[i];
                              a[i]=a[j];
                              a[j]=temp;

                              i++;
                              j--;
                        }
                  }
                  if(low<j)
                        quicksort(a,low,j);

                  if(i<high)
                        quicksort(a,i,high);
            }
            else
            {
                  while(i<=j)
                  {
                        while(a[i]>pivot)
                              i++;

                        while(a[j]<pivot)
                              j--;

                        if(i<=j)
```

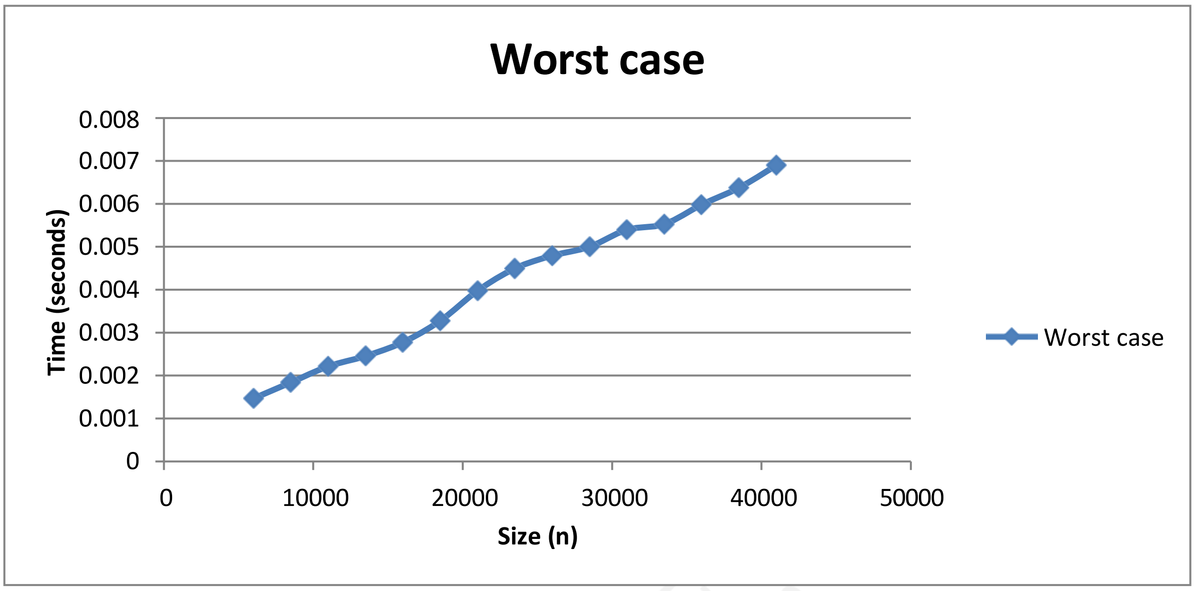
```
                    {
                          temp=a[i];
                          a[i]=a[j];
                          a[j]=temp;

                          i++;
                          j--;
                    }
              }
              if(low<j)
                    quicksort(a,low,j);

              if(i<high)
                    quicksort(a,i,high);
        }
  }
  public static void main(String[] args)throws IOException
  {
        int i;
        long st,et;

        Scanner read=new Scanner(System.in);

        Random random= new Random();

        PrintWriter out = new PrintWriter(new File("Random.txt"));

        System.out.println("Enter the no elements (>5000)");
        n=read.nextInt();
        a= new int[n];

        for(i=0;i<n;i++)
        {
              a[i]=random.nextInt(n)+1;
              out.print(a[i]+"\t");
        }

        System.out.println("The total numbers generated: " + i );

        out.close();

        st = System.nanoTime();
        quicksort(a,0,n-1);
        et= System.nanoTime() - st;

        PrintWriter outA = new PrintWriter(new File("Ascending.txt"));

        for(i=0;i<n;i++)
              outA.print(a[i]+"\t");

        outA.close();
```

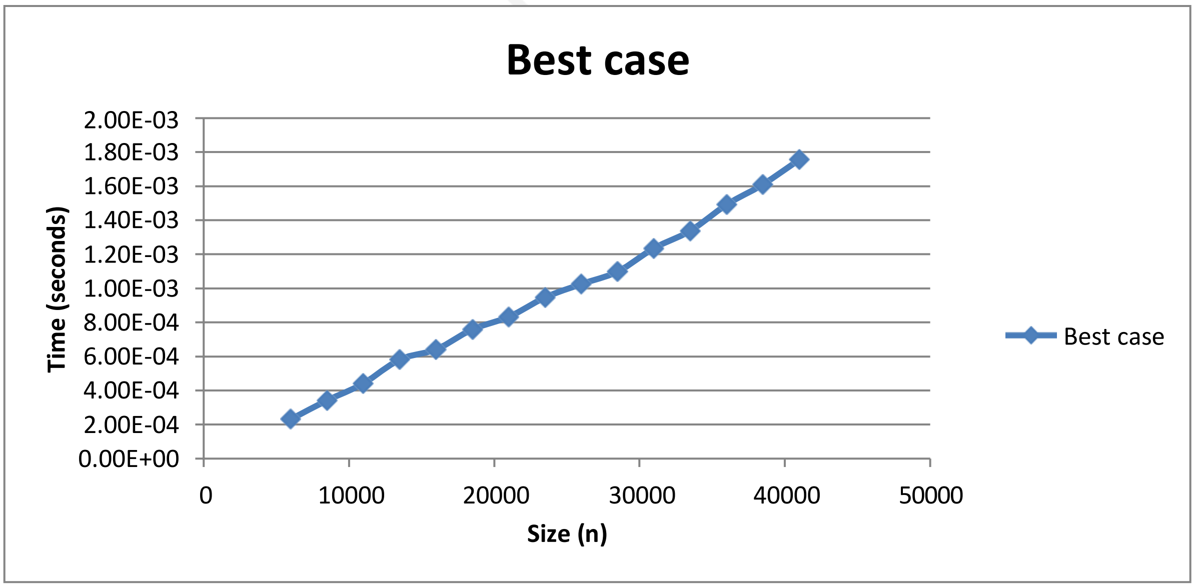
```
            System.out.println("THE TIME COMPLEXITY FOR WORST CASE IS... "
+ (et/1000000000.0) + " secs");

            st = System.nanoTime();
            quicksort(a,0,n-1);
            et= System.nanoTime() - st;

            System.out.println("THE TIME COMPLEXITY FOR BEST CASE IS... " +
(et/1000000000.0)+ " secs");

            flag=false;

            st = System.nanoTime();
            quicksort(a,0,n-1);
            et = System.nanoTime()- st;

            PrintWriter outD = new PrintWriter(new File("Descending.txt"));

            for(i=0;i<n;i++)
                  outD.print(a[i]+"\t");

            outD.close();

            System.out.println("THE TIME COMPLEXITY FOR AVERAGE CASE IS ...
" + (et/1000000000.0) + " secs");
      }
}
```

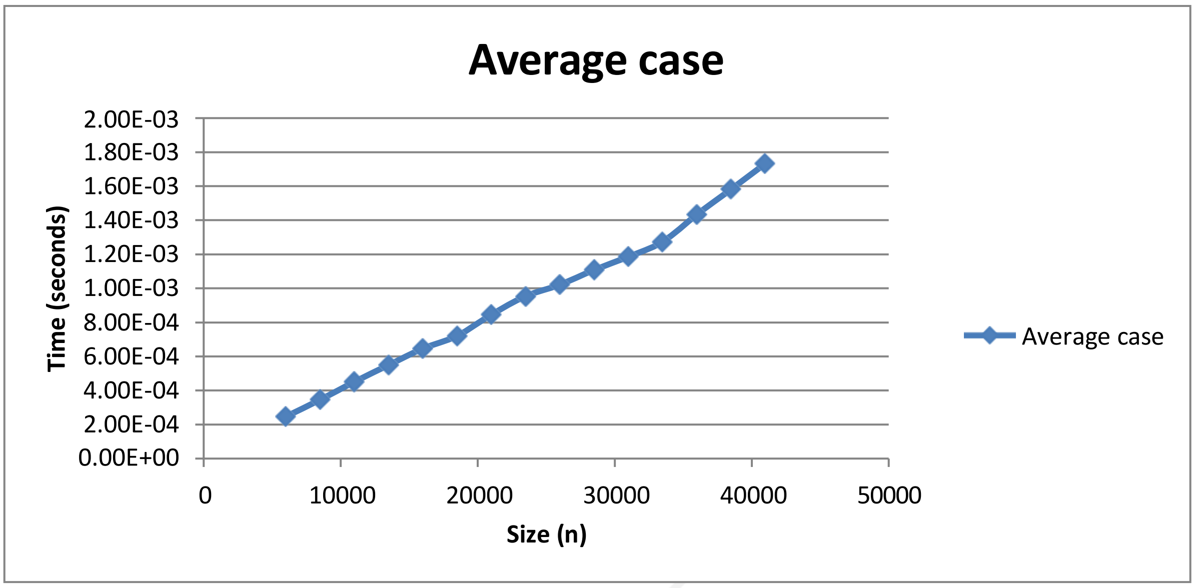
Data Table :

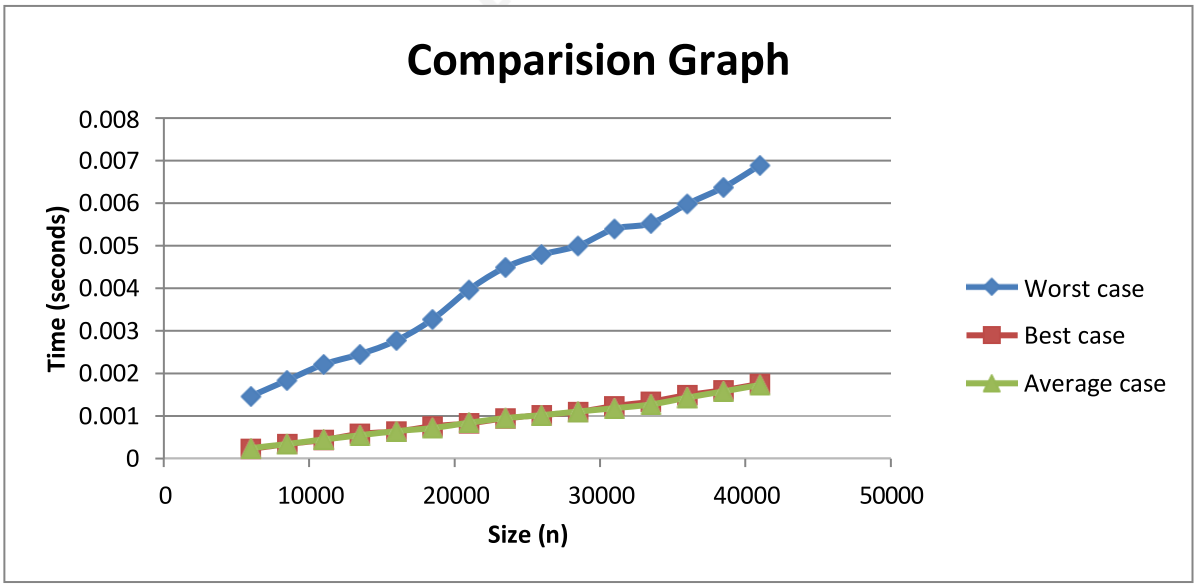
Size (n)	Time (in seconds)		
	Worst case	Best case	Average case
6000	0.001466	2.32E-04	2.47E-04
8500	1.84E-03	3.42E-04	3.46E-04
11000	0.002214	4.42E-04	4.51E-04
13500	0.002455	5.83E-04	5.50E-04
16000	0.002775	6.42E-04	6.47E-04
18500	0.003276	7.60E-04	7.21E-04
21000	0.003973	8.33E-04	8.46E-04
23500	0.004496	9.48E-04	9.54E-04
26000	4.79E-03	1.03E-03	0.001022
28500	0.005	0.001099	0.001108
31000	0.005399	1.24E-03	0.001186
33500	5.53E-03	1.34E-03	0.001273
36000	0.005982	0.001493	0.001434
38500	0.006375	0.001611	0.001582
41000	0.006895	0.001757	0.001732

Graphs :


Worst case
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
Time (seconds)
0
10000
20000
30000
40000
50000
Size (n)
Worst case


Best case
2.00E-03
1.80E-03
1.60E-03
1.40E-03
1.20E-03
1.00E-03
8.00E-04
6.00E-04
4.00E-04
2.00E-04
0.00E+00
Time (seconds)
0
10000
20000
30000
40000
50000
Size (n)
Best case


Average case
2.00E-03
1.80E-03
1.60E-03
1.40E-03
1.20E-03
1.00E-03
8.00E-04
6.00E-04
4.00E-04
2.00E-04
0.00E+00
Time (seconds)
0
10000
20000
30000
40000
50000
Size (n)
Average case


Comparision Graph
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
Time (seconds)
0
10000
20000
30000
40000
50000
Size (n)
Worst case
Best case
Average case

Program 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.

ALGORITHM :

```
ALGORITHM  Mergesort(A[0...n-1])
    //Sorts array A[0..n-1] by recursive mergesort
    //Input: An array A[0...n-1] of orderable elements
    //Output: Array A[0...n-1] sorted in non decreasing order
    If n>1
        copy A[0...⌊□/2⌋-1] to B[0... ⌊□/2⌋-1]
        copy A[ ⌊□/2⌋...n-1] to C[⌈□/2⌉...-1]
        Mergesort(B[0... ⌊□/2⌋-1])
        Mergesort(C[0...⌈□/2⌉-1])
        Merge(B, C, A)
```

```
ALGORITHM Merge(B[0...p-1], C[0...q-1], A[0...p+q-1])
    //Merges two sorted arrays into one sorted array
    //Input: Arrays B[0...p-1] and C[0...q-1] both sorted
    //Output: Sorted array A[0...p+q-1]of the elements of B and C
    i ← 0; j ← 0; k ← 0
    while i < p and j < q do
        if B[i]≤ C[j]
            A[k] ← B[i]; i ← i+1
        else   A[k] ← C[j]; j ← j+1 k
        ← k+1
    if i = p
        copy C[j...q-1] to A[k...p+q-1]
    else
        copy B[i...p-1] to A[k...p+q-1]
```

CODE :

```
import java.util.*;
import java.io.*;
public class P5 {
      static int [] a;
      static int n;
      static boolean flag = true;

      static void mergesort(int a[],int low,int high)
      {
            int mid;
            if(low<high)
            {
                  mid=(low+high)/2;

                  mergesort(a,low,mid);
                  mergesort(a,mid+1,high);
                  merge(a,low,mid,high);
            }
      }

      static void merge(int a[],int low,int mid,int high)
      {
            int i1,i2,j,k;
            int [] b=new int [n];

            i1=j=low;
            i2=mid+1;

            if(flag)
            {
                  while((i1<=mid)&&(i2<=high))
                  {
                        if(a[i1]<=a[i2])
                        {
                              b[j]=a[i1];
                              i1++;
                        }
                        else
                        {
                              b[j]=a[i2];
                              i2++;
                        }
                        j++;
                  }



                  if(i1>mid)
                        for(k=i2;k<=high;j++,k++)
                              b[j]=a[k];
                  else
                        for(k=i1;k<=mid;j++,k++)
                              b[j]=a[k];
```

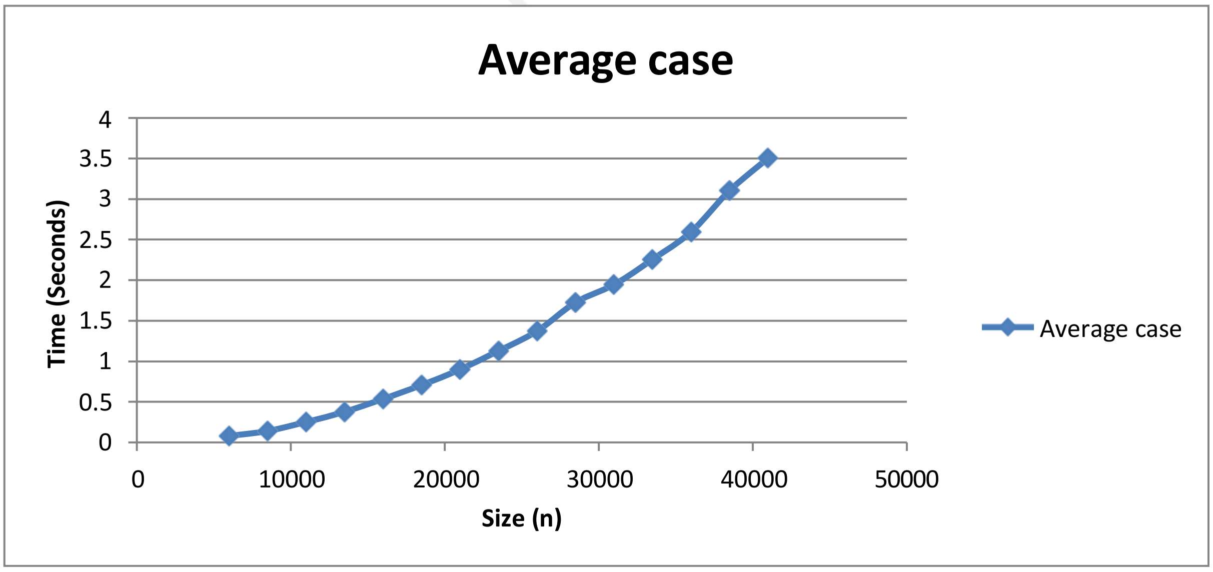
```
                for(k=low;k<=high;k++)
                      a[k]=b[k];

          }
          else
          {
                while((i1<=mid)&&(i2<=high))
                {
                      if(a[i1]>=a[i2])
                      {
                            b[j]=a[i1];
                            i1++;
                      }
                      else
                      {
                            b[j]=a[i2];
                            i2++;
                      }
                      j++;
                }

                if(i1>mid)
                      for(k=i2;k<=high;j++,k++)
                            b[j]=a[k];
                else
                      for(k=i1;k<=mid;j++,k++)
                            b[j]=a[k];

                for(k=low;k<=high;k++)
                      a[k]=b[k];

          }
    }

    public static void main(String[] args)throws IOException
    {
          int i;
          long st,et;

          Scanner read=new Scanner(System.in);

          Random random= new Random();

          PrintWriter out = new PrintWriter(new File("Random.txt"));

          System.out.println("Enter the no elements (>5000)");
          n=read.nextInt();

          a= new int[n];
```

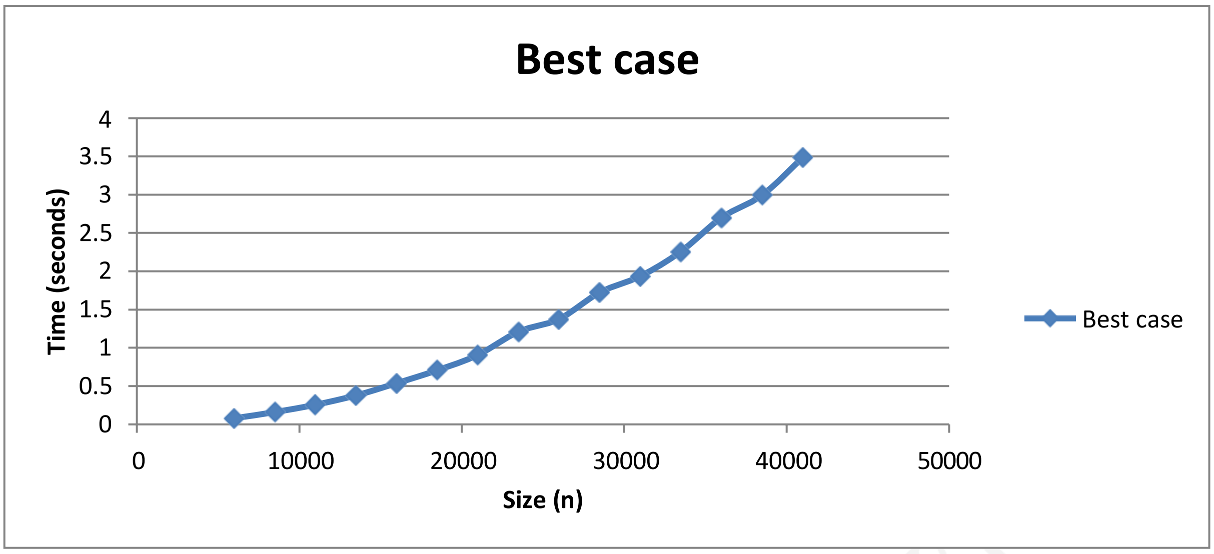
```
            for(i=0;i<n;i++)
            {
                  a[i]=random.nextInt(n)+1;
                  out.print(a[i]+"\t");
            }

            System.out.println("The total numbers generated: " + i );

            out.close();

            st = System.nanoTime();
            mergesort(a,0,n-1);
            et= System.nanoTime() - st;

            PrintWriter outA = new PrintWriter(new File("Ascending.txt"));

            for(i=0;i<n;i++)
                  outA.print(a[i]+"\t");

            outA.close();

            System.out.println("THE TIME COMPLEXITY FOR WORST CASE IS... "
+ (et/1000000000.0) + " secs");

            st = System.nanoTime();
            mergesort(a,0,n-1);
            et= System.nanoTime() - st;

            System.out.println("THE TIME COMPLEXITY FOR BEST CASE IS... " +
(et/1000000000.0)+ " secs");

            flag=false;

            st = System.nanoTime();
            mergesort(a,0,n-1);
            et = System.nanoTime()- st;

            PrintWriter outD = new PrintWriter(new File("Descending.txt"));

            for(i=0;i<n;i++)
                  outD.print(a[i]+"\t");

            outD.close();

            System.out.println("THE TIME COMPLEXITY FOR AVERAGE CASE IS ...
" + (et/1000000000.0) + " secs");
      }
}
```

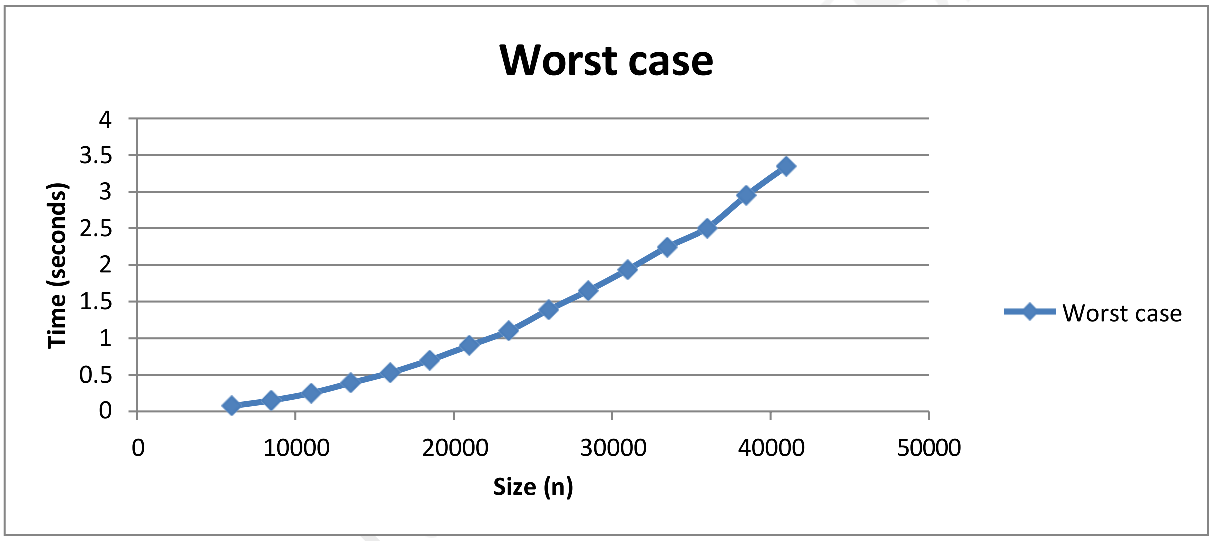
Data Table

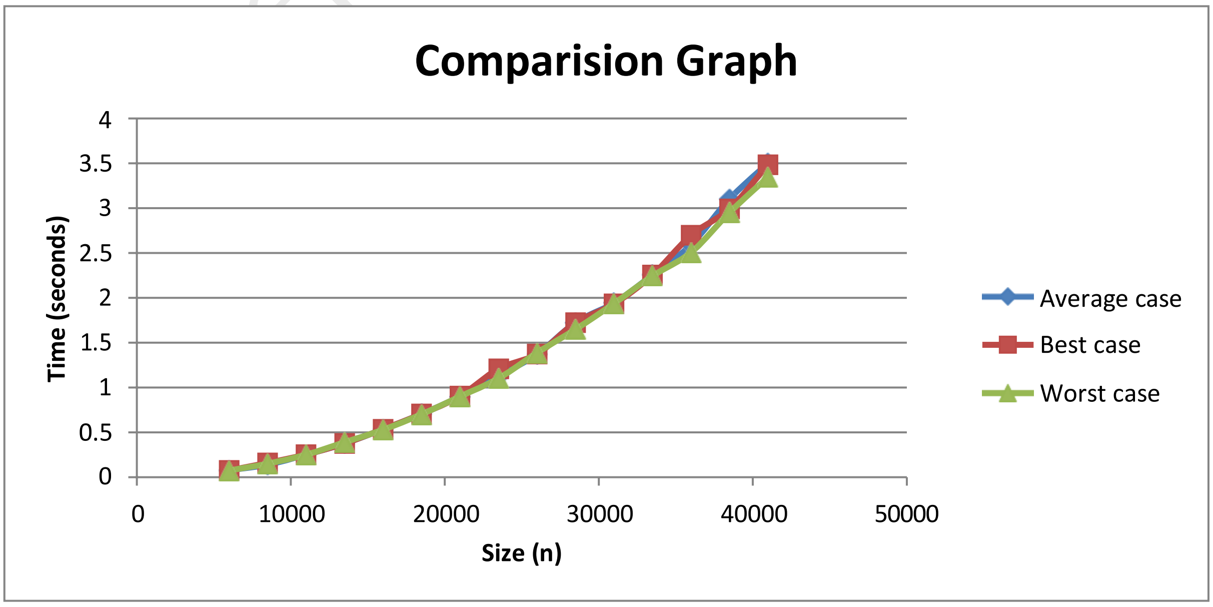
Size (n)	Time (in seconds)		
	Average case	Best case	Worst case
6000	0.079857	0.078119	0.074417
8500	0.140399	0.16106	0.150676
11000	0.252403	0.255474	0.251018
13500	0.37724	0.377781	0.389371
16000	0.536145	0.537254	0.529286
18500	0.706898	0.70681	0.701618
21000	0.900263	0.906104	0.900545
23500	1.129271	1.211591	1.102181
26000	1.372421	1.374205	1.387737
28500	1.726517	1.724516	1.649343
31000	1.945343	1.934265	1.936539
33500	2.257244	2.256724	2.245219
36000	2.593985	2.700119	2.505199
38500	3.104938	2.993526	2.952188
41000	3.503476	3.485765	3.346788

Graphs:




Best case
Time (seconds)
4
3.5
3
2.5
2
1.5
1
0.5
0
0
10000
20000
30000
40000
50000
Size (n)
Best case


Worst case
Time (seconds)
4
3.5
3
2.5
2
1.5
1
0.5
0
0
10000
20000
30000
40000
50000
Size (n)
Worst case


Comparision Graph
Time (seconds)
4
3.5
3
2.5
2
1.5
1
0.5
0
0
10000
20000
30000
40000
50000
Size (n)
Average case
Best case
Worst case

Program 6

Implement in Java, the **0/1 Knapsack** problem using

- Dynamic Programming method
- Greedy method

ALGORITHM :

```
ALGORITHM MFKnapsack(i, j)
    // Implements the memory function method for the knapsack problem
    // Input: A nonnegative integer i indicating the number of the first items being
    // considered and a nonnegative integer j indicating the knapsack's capacity
    // Output: The value of an optimal feasible subset of the first i items
    // Note: Uses as global variables input arrays Weights[1..n], Values[1..n], and table
    // V[0..n, 0..W] whose entries are //initialized with -1's except for row 0 and column 0
    // initialized with 0's
    if V[i, j] < 0
        if j < Weights[i]
            value ← MFKnapsack(i – 1, j)
        else
            value ← max(MFKnapsack(i – 1, j), Values[i] +
            MFKnapsack(i – 1, j-Weights[i]))
        V[i, j] ← value
    return V[i, j]
```

ALGORITHM *GreedyKnapsack(*float *m*, int *n)*

```
//P[1:n] and w[1:n] contain the profits and weights respectively of the n objects
// ordered such that $P[i]/w[i] \geq P[i+1]/w[i+1]$. m is the knapsack size and x[1:n] is
// the solution vector

{
    for (int i = 1; i ≤ n; i++)
        x[i] = 0.0;            //Initialize x
    float u = m;
    for (i = 1; i ≤ n; i++) {
        if (w[i] > u)    continue;
        x[i] = 1.0;
        u - = w[i];
    }

}
```

CODE :

```
import java.util.*;
public class Knapsack {
      static int []p =new int[50];
      static int []w =new int[50];
      static int []x =new int[50];
      static int []t =new int[50];
      static double maxprofit;
      static int n,m,i,j;

      static void dynamicknapsack(int n, int w[], int p[], int m)
      {
            int [][]v= new int[n+1][m+1];

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

            for(j=0;j<=m;j++)
                  v[0][j]=0;
```

```
        for(i=1;i<=n;i++)
              for(j=1;j<=m;j++)
                    if(j<w[i])
                          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("Solution Table : ");
        for(i=0;i<=n;i++)
        {
              for(j=0;j<=m;j++)
                    System.out.print(v[i][j]+"\t");
              System.out.println("\n");
        }

        System.out.println("Optimal solution for dynamic method :"
        +v[n][m]);

        i=n;
        j=m;

        while(i!=0 && j!=0)
        {
              if(v[i][j]!=v[i-1][j])
              {
                    x[i]=1;
                    j=j-w[i];
              }
              i=i-1;
        }
        System.out.println("The Solution Vector for Dynamic Method is :
        ");
        for(i=1;i<=n;i++)
              System.out.print(x[i]+"\t");
        System.out.println("\n");
  }
  static int max(int a, int b)
  {
        return (a>b)?a:b;
  }
  static void greedyknapsack(int n, int w[], int p[], int m)
  {
        int rc=m;

        bubblesort(n,w,p,t);

        for(i=1;i<=n;i++)
        {
              if(w[t[i]]>rc)
                    continue;
```

```
                x[t[i]]=1;
                rc-=w[t[i]];
                maxprofit+=p[t[i]];
            }

            System.out.println("Optimal solution for greedy method :
            "+maxprofit);

            System.out.println("Solution vector for greedy method : ");
            for(i=1;i<=n;i++)
                System.out.print(x[i]+"\t");
        }
        static void bubblesort(int n,int w[],int p[],int t[])
        {
            int temp;

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

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

        if((double)p[t[j]]/w[t[j]]<(double)p[t[j+1]]/w[t[j+1]])
                    {
                        temp=t[j];
                        t[j]=t[j+1];
                        t[j+1]=temp;
                    }
        }
        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);

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

            System.out.println("Enter the object's weights : ");
            for(i=1;i<=n;i++)
                w[i]=sc.nextInt();

            System.out.println("Enter the object's profits : ");
            for(i=1;i<=n;i++)
                p[i]=sc.nextInt();

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

            dynamicknapsack(n,w,p,m);

            for(i=1;i<=n;i++)
                x[i]=0;

            greedyknapsack(n,w,p,m);
        }
```

```
}
```

Output:

```
Enter the number of Objects :
3
Enter the object's weights :
18 15 10
Enter the object's profits :
25 24 15
Enter the maximum capacity :
20
Solution Table :
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  25  25  25

0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  24  24  24  25  25  25

0  0  0  0  0  0  0  0  0  0  15  15  15  15  15  24  24  24  25  25  25

Optimal solution for dynamic method :25
The Solution Vector for Dynamic Method is :
1      0      0

Optimal solution for greedy method : 24.0
Solution vector for greedy method :
0      1      0
```

```
Enter the number of Objects :
7
Enter the object's weights :
2 3 5 7 1 4 1
Enter the object's profits :
10 5 15 7 6 18 3
Enter the maximum capacity :
15
Solution Table :
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

0  0  10  10  10  10  10  10  10  10  10  10  10  10  10  10

0  0  10  10  10  15  15  15  15  15  15  15  15  15  15  15

0  0  10  10  10  15  15  25  25  25  30  30  30  30  30  30

0  0  10  10  10  15  15  25  25  25  30  30  30  30  32  32

0  6  10  16  16  16  21  25  31  31  31  36  36  36  36  38

0  6  10  16  18  24  28  34  34  34  39  43  49  49  49  54

0  6  10  16  19  24  28  34  37  37  39  43  49  52  52  54

Optimal solution for dynamic method :54
The Solution Vector for Dynamic Method is :
1      1      1      0      1      1      0

Optimal solution for greedy method : 52.0
Solution vector for greedy method :
1      0      1      0      1      1      1
```

```
Enter the number of Objects :
4
Enter the object's weights :
2 1 3 2
Enter the object's profits :
12 10 20 15
Enter the maximum capacity :
5
Solution Table :
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

Optimal solution for dynamic method :37
The Solution Vector for Dynamic Method is :
1      1      0      1

Optimal solution for greedy method : 37.0
Solution vector for greedy method :
1      1      0      1
```

Program 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.

ALGORITHM :

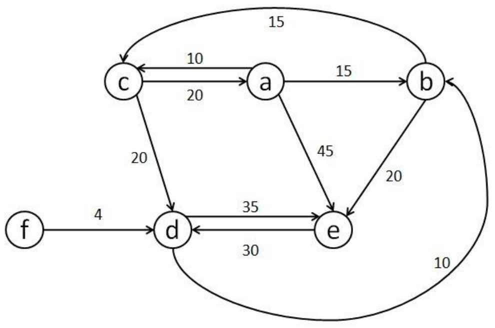
```
ALGORITHM  Dijkstra(G)
    // Dijkstra's algorithm for single-source shortest paths
    // Input: A weighted connected graph $G = <V, E>$ with
    // non negative weights and its vertex $s$
    // Output: The length $d_v$ of a shortest path from $s$ to $v$ and its
    // penultimate vertex $p_v$ for every vertex $v$ in $V$

    Initialize(Q)    //initialize vertex priority queue to empty
    for every vertex $v$ in $V$ do
        $d_v \leftarrow \infty$; $p_v \leftarrow$ null
        Insert(Q, v, $d_v$)            //initialize vertex priority in the priority queue
    $d_s \leftarrow 0$; Decrease(Q, s, $d_s$) //update priority of s with $d_s$
    $V_T \leftarrow \emptyset$
    for $i \leftarrow 0$ to $|V| - 1$ do
        $u^* \leftarrow$ DeleteMin(Q) //delete the minimum priority element
        $V_T \leftarrow V_T \cup \{u^*\}$
        for every vertex $u$ in $V - V_T$ that is adjacent to $u^*$ do
            if $d_{u^*} + w(u^*, u) < d_u$
                $d_u \leftarrow d_{u^*} + w(u^*, u)$;    $p_u \leftarrow u^*$
              Decrease(Q, u, $d_u$)
```

CODE :

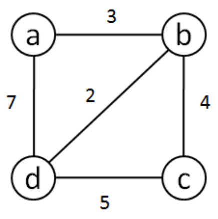
```
import java.util.Scanner;
public class P7 {
      static void shortest(int v,int cost[][],int dist[],int n)
      {
            boolean [] s = new boolean[10];
            int i,w,u,num;
            for(i=1;i<=n;i++)
            {
                  s[i]=false;
                  dist[i]=cost[v][i];
            }
            s[v]=true;
            dist[v]=0;
            num=2;

            while(num<=n)
            {
                  u=choose(dist,s,n);
                  s[u]=true;
                  num++;
                  for(w=1;w<=n;w++)
                  {
                              if(((dist[u]+cost[u][w])<dist[w])&&!s[w])
                                    dist[w]=dist[u]+cost[u][w];
                  }
            }
      }

      static int choose(int dist[],boolean s[],int n)
      {
            int w,j=0,min=9999;

            for(w=1;w<=n;w++)
                  if((dist[w]<min)&&(s[w]==false))
                  {
                        min=dist[w];
                        j=w;
                  }
            return j;
      }


      public static void main(String [] args)
      {
            int [][]cost = new int[50][50];
            int []dist = new int[50];
            int i,j,n,v;

            Scanner read=new Scanner(System.in);

            System.out.println("enter the no. of nodes:");
            n=read.nextInt();
            System.out.println("enter the cost adjacency matrix,'9999' for
            no direct path\n");
            for(i=1;i<=n;i++)
                  for(j=1;j<=n;j++)
                        cost[i][j]=read.nextInt();

            System.out.println("enter the starting vertex:");
            v=read.nextInt();

            shortest(v,cost,dist,n);
            System.out.println("Shortest path from starting vertex to other
vertices are ");

            for(j=1;j<=n;j++)
                  System.out.println(v+"->"+j+"="+dist[j]);
      }
}
```

Output:

```
enter the no. of nodes:
6
enter the cost adjacency matrix,'9999' for no direct path

0     15    10    9999  45    9999
9999  0     15    9999  20    9999
20    9999  0     20    9999  9999
9999  10    9999  0     35    9999
9999 9999 9999 30     0     9999
9999  9999 9999  4     9999  0
enter the starting vertex:
6
Shortest path from starting vertex to other vertices are
6->1=49
6->2=14
6->3=29
6->4=4
6->5=34
6->6=0
```



```
enter the no. of nodes:
4
enter the cost adjacency matrix,'9999' for no direct path

0     3     9999  7
3     0     4     2
9999  4     0     5
7     2     5     0
enter the starting vertex:
1
Shortest path from starting vertex to other vertices are
1->1=0
1->2=3
1->3=7
1->4=5
```



Program 8

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

ALGORITHM :

```
ALGORITHM Kruskal(G)
    // Kruskal's algorithm for constructing a minimum spanning tree
    // Input: A weighted connected graph $G = <V, E>$
    // Output: $E_T$, the set of edges composing a minimum spanning tree of $G$

    Sort $E$ in non-decreasing order of the edge weights $w(e_{i1}) \le . . . \le w(e_{i|E|})$
    $E_T \leftarrow \emptyset$; $ecounter \leftarrow 0$          //initialize the set of tree edges and its size
    $k \leftarrow 0$
    while $ecounter < |V| - 1$ do
        $k \leftarrow k+1$
        if $E_T \cup \{e_{ik}\}$ is acyclic
            $E_T \leftarrow E_T \cup \{e_{ik}\}$;   $ecounter \leftarrow ecounter + 1$
    return $E_T$
```

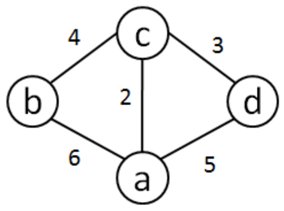
CODE:

```
import java.util.Scanner;

public class P8 {

	static int [] parent =new int [50];
	static int [][]cost = new int [10][10];
	static int a,b,i,j,u,v,n,min,noe=1,mincost=0;

	static int find(int w)
	{
		while(parent[w]!=0)
			w=parent[w];

		return w;
	}



	static void union()
	{
```

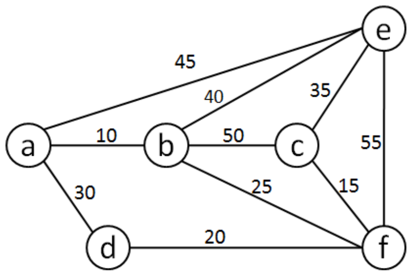
```
            if(u!=v)
            {
                  noe++;

                  System.out.println(noe-1 + "Edge("+a+","+b+")="+min);
                  mincost+=min;
                  parent[v]=u;
            }

            cost[a][b]=cost[b][a]=9999;
      }

      public static void main(String [] args)
      {
            Scanner read = new Scanner(System.in);
            System.out.println("Enter the no. of vertices:");
            n=read.nextInt();

            System.out.println("Enter the cost adjacency matrix, 9999 for
no direct path:");
            for(i=1;i<=n;i++)
                  for(j=1;j<=n;j++)
                        cost[i][j]=read.nextInt();



            while(noe<n)
            {
                  min=9999;

                  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);
            union();
            }


            System.out.println("minimum cost ="+mincost);

      }

}
```

Output:

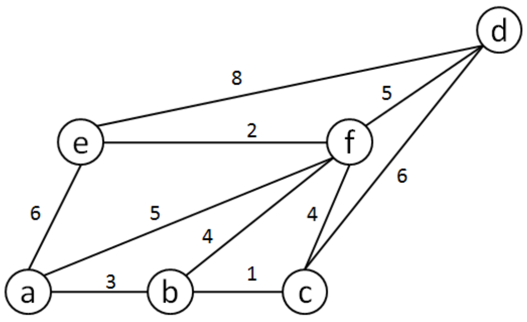
```
Enter the no. of vertices:
4
Enter the cost adjacency matrix, 9999 for no direct path:
0       6       2       5
6       0       4       9999
2       4       0       3
5       9999    3       0
1Edge(1,3)=2
2Edge(3,4)=3
3Edge(2,3)=4
minimum cost =9
```



```
Enter the no. of vertices:
6
Enter the cost adjacency matrix, 9999 for no direct path:
0       10      9999    30      45      9999
10      0       50      9999    40      25
9999    50      0       9999    35      15
30      9999    9999    0       9999    20
45      40      35      9999    0       55
9999    25      15      20      55      0
1Edge(1,2)=10
2Edge(3,6)=15
3Edge(4,6)=20
4Edge(2,6)=25
5Edge(3,5)=35
minimum cost =105
```



```
Enter the no. of vertices:
6
Enter the cost adjacency matrix, 9999 for no direct path:
0       3       9999    9999    6       5
3       0       1       9999    9999    4
9999    1       0       6       9999    4
9999    9999    6       0       8       5
6       9999    9999    8       0       2
5       4       4       5       2       0
1Edge(2,3)=1
2Edge(5,6)=2
3Edge(1,2)=3
4Edge(2,6)=4
5Edge(4,6)=5
minimum cost =15
```



Program 9

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

ALGORITHM :

```
ALGORITHM  Prim(G)
    //Prim's algorithm for constructing a minimum spanning tree
    //Input: A weighted connected graph $G = <V, E>$
    //Output: $E_T$, the set of edges composing a minimum spanning tree of $G$

    $V_T \leftarrow \{v_0\}$            //the set of tree vertices can be initialized
                                 //with any vertex
    $E_T \leftarrow Ø$
    for $i \leftarrow 1$ to $|V| - 1$ do
        find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges $(v, u)$ such that
        $v$ is in $V_T$ and $u$ is in $V - V_T$
        $V_T \leftarrow V_T \cup \{u^*\}$
        $E_T \leftarrow E_T \cup \{e^*\}$
    return $E_T$
```

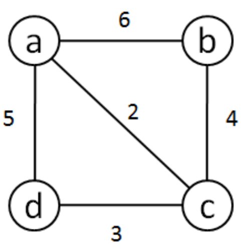
CODE :

```
import java.util.Scanner;

public class P9
{
      public static void main(String [] args)
      {
          int n,i,j;
          int [][]cost=new int[10][10];
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter the no of vertices");
          n=sc.nextInt();

          System.out.println("Enter the cost adjacency matrix,'9999' for no
direct path ");
          for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                  cost[i][j]=sc.nextInt();
          prims(cost,n);
      }
```

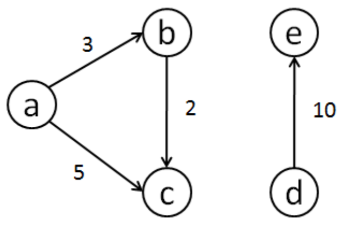
```
    static void prims(int cost[][],int n)
    {
        int []v=new int[10];
        int min,p,q,i,j,flag=0,mincost=0,noe=1;

        for(i=1;i<=n;i++)
          v[i]=0;

        v[1]=1;

        System.out.println("The spanning tree has the following edges");

        while(noe!=n)
        {
            min=9999;
            i=j=-1;
            flag=0;

            for(p=1;p<=n;p++)
                for(q=1;q<=n;q++)
                {
                      if(p==q)
                          continue;

                      if((cost[p][q]<min) && (v[p]==1) && (v[q]!=1))
                      {
                          min=cost[p][q];
                          i=p;
                          j=q;
                          flag=1;
                      }
                }

            if(flag==0)
            {
                System.out.println("Graph is disconnected\n");
                System.exit(0);
            }

            if(i!=-1)
            {
                v[j]=1;
                System.out.println("("+i+","+j+") ="+ cost[i][j]);
                mincost+=cost[i][j];
                noe++;
            }
        }

        System.out.println("Cost of spanning tree:"+ mincost);
    }
}
```

Output:

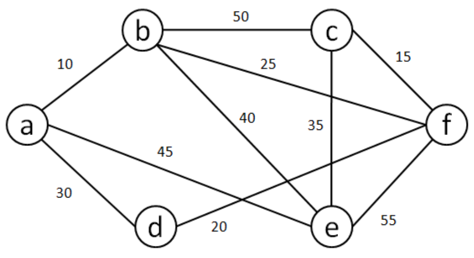
```
Enter the no of vertices
4
Enter the cost adjacency matrix,'9999' for no direct path
0      6       2      5
6      0       4      9999
2      4       0      3
5      9999    3      0
The spanning tree has the following edges
(1,3) =2
(3,4) =3
(3,2) =4
Cost of spanning tree:9
```



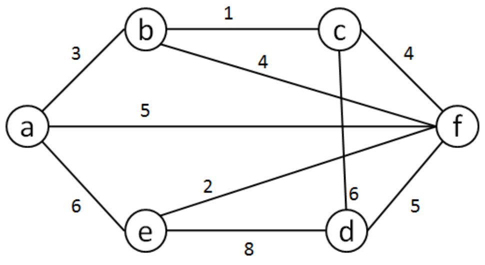
```
Enter the no of vertices
5
Enter the cost adjacency matrix,'9999' for no direct path
0      3       5      9999   9999
9999   0       2      9999   9999
9999   9999    0      9999   9999
9999   9999    9999   0      10
9999   9999    9999   9999   0
The spanning tree has the following edges
(1,2) =3
(2,3) =2
Graph is disconnected
```



```
Enter the no of vertices
6
Enter the cost adjacency matrix,'9999' for no direct path
0      10      9999   30     45     9999
10     0       50     9999   40     25
9999   50      0      9999   35     15
30     9999    9999   0      9999   20
45     40      35     9999   0      55
9999   25      15     20     55     0
The spanning tree has the following
edges
(1,2) =10
(2,6) =25
(6,3) =15
(6,4) =20
(3,5) =35
Cost of spanning tree:105
```



```
Enter the no of vertices
6
Enter the cost adjacency matrix,'9999' for no direct path
0      3      9999  9999  6     5
3      0      1     9999  9999  4
9999  1      0     6     9999  4
9999  9999   6     0     8     5
6      9999   9999  8     0     2
5      4      4     5     2     0
The spanning tree has the following edges
(1,2) =3
(2,3) =1
(2,6) =4
(6,5) =2
(6,4) =5
Cost of spanning tree:15
```



Program 10.a

Write Java program to implement All-Pairs Shortest Paths problem using **Floyd's algorithm**.

ALGORITHM :

```
ALGORITHM Floyd( W[1...n, 1...n] )
    // Implement Floyd's algorithm for all-pairs shortest-paths problem
    // Input: The weight matrix W of a graph with no negative-length cycle
    // Output: The distance matrix of the shortest path's lengths

    D ← W          //is not necessary if W can be overwritten
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i, j] ← min{ D[i, j], D[i, k] + D[k, j] }
    return D
```

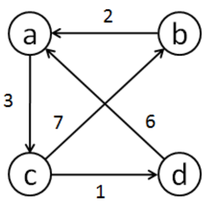
CODE:

```
import java.util.Scanner;

public class P10a {

      public static void main(String [] args)
      {
         int i,j,k,n;
         int [][] a= new int[10][10];

         Scanner read = new Scanner(System.in);
         System.out.println("enter the no. of nodes");
         n=read.nextInt();

         System.out.println("enter the cost adjacency matrix,'9999' for no
direct path");
         for(i=1;i<=n;i++)
            {
               for(j=1;j<=n;j++)
                     a[i][j]=read.nextInt();
               a[i][i]=0;
            }

         for(k=1;k<=n;k++)
            for(i=1;i<=n;i++)
                  for(j=1;j<=n;j++)
                        if(a[i][k]+a[k][j]<a[i][j])
                           a[i][j]=a[i][k]+a[k][j];
```

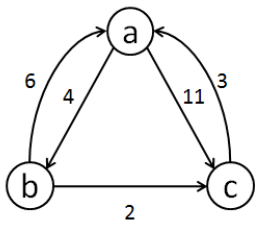
```
          System.out.println("output path matrix");
          for(i=1;i<=n;i++)
          {
              for(j=1;j<=n;j++)
                   System.out.print(a[i][j]+"\t");
              System.out.println();
          }
       }
}
```

Output:

```
enter the no.of nodes
4
enter the cost adjacency matrix,'9999' for no direct path
0      9999   3      9999
2      0      9999   9999
9999   7      0      1
6      9999   9999   0
output path matrix
0      10     3      4
2      0      5      6
7      7      0      1
6      16     9      0
```



```
enter the no.of nodes
3
enter the cost adjacency matrix,'9999' for no direct path
0       4      11
6       0      2
3       9999   0
output path matrix
0      4      6
5      0      2
3      7      0
```



Program 10.b

Write Java program to implement **Travelling Sales Person problem** using Dynamic programming.

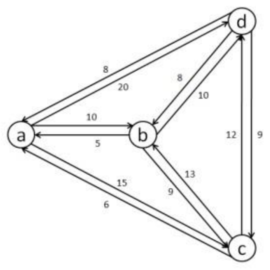
CODE:

```
import java.util.Scanner;

public class P10b {
	static int [][] cost = new int [20][20];
	static int [] visited = new int [20];
	static int n,min_cost;

	static int Tsp_Dynamic(int i,int copy [])
	{
		int min=999,val,j;
		int [] s = new int [20];

		boolean flag=false;

		for(j=1;j<=n;j++)
			s[j]=copy[j];

		s[i]=1;

		if(n==1)
			return cost[i][1];

		for(j=1;j<=n;j++)
		{
			if(s[j]==0)
			{
				flag=true;
				val=cost[i][j]+Tsp_Dynamic(j,s);

				if(val<min)
					min=val;
			}
		}
		if(!flag)
			min=cost[i][1];

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

		Scanner read=new Scanner(System.in);

		System.out.println("Enter the number of cities");
```

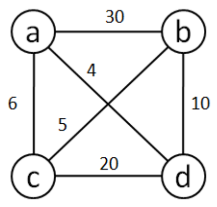
```
            n=read.nextInt();

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

            min_cost=Tsp_Dynamic(1,visited);

            System.out.println("The cost of optimal tour is "+ min_cost);
      }

}
```

Output:

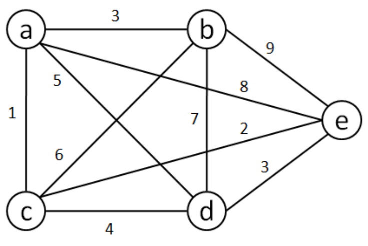
```
Enter the number of cities
4
Enter the cost adjacency matrix
0        10      15      20
5        0       9       10
6        13      0       12
8        8       9       0
The cost of optimal tour is 35
```



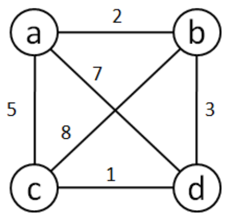
```
Enter the number of cities
4
Enter the cost adjacency matrix
0       30       6       4
30      0        5       10
6       5        0       20
4       10       20      0
The cost of optimal tour is 25
```



```
Enter the number of cities
5
Enter the cost adjacency matrix
0 3 1 5 8
3 0 6 7 9
1 6 0 4 2
5 7 4 0 3
8 9 2 3 0
The cost of optimal tour is 16
```



```
Enter the number of cities
4
Enter the cost adjacency matrix
0 2 5 7
2 0 8 3
5 8 0 1
7 3 1 0
The cost of optimal tour is 11
```



Program 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.

ALGORITHM :

void SumOfSub (float s, int k, float r)

```
        // Find all subsets of w[1:n] that sum to m. The values of x[j], 1<=j<k, have already
    // been determined, s= $\sum_{□=1}^{□-1} □[□] * □[□]$ and r= $\sum_{□=□}^{□} □[□]$. The w[j]'s are in
        // nondecreasing order. It is assumed that w[1] ≤ m and $\sum_{□=1}^{□} □[□] \geq m$.
        {
                // Generate left child. Note that s+w[k] <=m
                // because $B_{k-1}$ is true.
                 x[k] =1;
                 if (s+w[k] == m)
                 {       // Subset found
                         for (int j=1; j<=k; j++) cout << x[j] << ' ' ;
                                 cout << end1;
                 }
                  // There is no recursive call here
                  // as w[j] > 0, 1 <= j <+ n.
                 else if (s+w[k]+w[k+1] <= m)
                         SubOfSub(s+w[k], k+1, r-w[k]);
                 // Generate right child and evaluate $B_k$ .
                 if ((s+r-w[k] >= m) && )s+w[k+1] <= m)) {
                         x[k] = 0;
                         SumOfSub(S, k+1, r-w[k]);
                }
        }
```

CODE:

```
import java.util.Scanner;

public class P11 {

       static int d,flag=0;
       static int []S = new int [10];
       static int []x = new int [10];
```

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

		x[k]=1;

		if((s+S[k]) == d)
		{
			flag=1;
			for(i=1;i<=k;i++)
			if(x[i]==1)
				System.out.print(S[i]+"\t");
			System.out.println();
		}



		else


		if(s+S[k]+S[k+1]<=d)
			sumofsub(s+S[k],k+1,r-S[k]);

		if((s+r-S[k]>=d) && (s+S[k+1]<=d))
		{
			x[k]=0;
			sumofsub(s,k+1,r-S[k]);
		}
	}

	public static void main(String [] args)
	{
		int i,n,sum=0;

		Scanner read= new Scanner(System.in);

		System.out.println("enter the no. of elements in the set");
		n=read.nextInt();

		System.out.println("enter the set in increasing order");
		for(i=1;i<=n;i++)
			S[i]=read.nextInt();

		System.out.println("enter the max subset value");
			d=read.nextInt();

		for(i=1;i<=n;i++)
			sum=sum+S[i];

		if(sum<d||S[1]>d)
			System.out.println("no subset possible");
		else
		{
			System.out.println("The possible subsets are");
			sumofsub(0,1,sum);
			if(flag==0)
				System.out.println("no subset possible ");
		}
	}
}
```

Output:

```
enter the no. of elements in the set
5
enter the set in increasing order
1       2       5       6       8
enter the max subset value
9
The possible subsets are
1     2     6
1     8
```

```
enter the no. of elements in the set
5
enter the set in increasing order
1       2       5       6       8
enter the max subset value
25
no subset possible
```

```
enter the no. of elements in the set
4
enter the set in increasing order
3 5 6 7
enter the max subset value
15
The possible subsets are
3     5     7
```

```
enter the no. of elements in the set
4
enter the set in increasing order
3 5 6 7
enter the max subset value
2
no subset possible
```

Program 12

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

ALGORITHM :

```
void Hamiltonian (int k)
        // This program uses the recursive formulation of  backtracking to find all
        // the Hamiltonian cycles of a graph.  The graph is stored as an adjacency
        // matrix G[1:n] [1:n]. All cycles begin at node 1.
        {
                do {    // Generate values for x[k].
                        NextValue(k) ; // Assign a legal next value to x[k].
                          if (!x[k]) return;
                          if (k == n)    {
                                  for (int i=1; i<=n; i++) cout << x[i] << ' ' ;
                                  cout << "1\n";
                          }
                          else Hamiltonian (k+1);
                } while (1);
        }

void NextValue(int k)
// x[1],..., x[k-1] is a path of k-1 distinct vertices. If x[k]==0, then no vertex has as yet
// been assigned to x[k]. After execution x[k] is assigned to the next highest
// numbered vertex which i) does not already appear in x[1], x[2],..., x[k-1]; and
// ii) is connected by an edge to x[k-1]. Otherwise x[k]==0. If k==n, then
// in addition x[k] is connected to x[1].
{
        do{
            x[k] = (x[k]+1) % (n+1); // Next vertex
            if (!x[k]) return;
            if (G[x[k-1]] [x[k]]) {        // Is there an edge?
                for (int  j=1; j<=k-1; j++)       // Check for distinctness.
                        if (x[j]==x[k])
                                break;
                if (j==k)          // If true, then the vertex is distinct.
                        if ((k<n) || ((k==n) && G[x[n]] [x[1]]))
                                return;
                }
        } while (1);
}
```

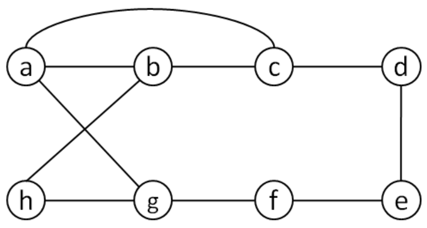
CODE:

```
import java.util.Scanner;

public class P12new {

static int [] x = new int [25];

	static void Next_Vertex(int G[][],int n,int k)
	{
		int j;

		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;
				}
			}
		}
	}
	static void H_Cycle(int G[][],int n,int k)
	{
		int i;

		while(true)
		{
			Next_Vertex(G,n,k);

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

			if(k==n)
			{
				System.out.println("\n");

				for(i=1;i<=n;i++)
					System.out.print(x[i] +"-->");
```

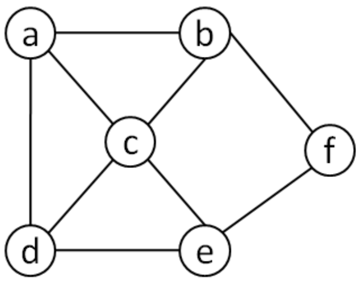
```
                        System.out.print(x[1]);
                  }
                  else
                        H_Cycle(G,n,k+1);
            }
      }

      public static void main(String[] args) {
            int i,j,n;
            int [][] G = new int [25][25];

            Scanner read = new Scanner(System.in);

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

            System.out.println("Enter the Path adjacency matrix");

            for(i=1;i<=n;i++)
            {
                  for(j=1;j<=n;j++)
                  {
                        G[i][j]=read.nextInt();
                        x[i]=0;
                  }
            }

            x[1]=1;

            System.out.println(" \n Hamiltonian Cycles are");

            H_Cycle(G,n,2);
      }

}
```

Output:

```
Enter the number of vertices of the graph
8
Enter the Path adjacency matrix
0 1 1 0 0 0 1 0
1 0 1 0 0 0 0 1
1 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 1 0 0
0 0 0 0 1 0 1 0
1 0 0 0 0 1 0 1
0 1 0 0 0 0 1 0

 Hamiltonian Cycles are


1-->2-->8-->7-->6-->5-->4-->3-->1

1-->3-->4-->5-->6-->7-->8-->2-->1
```



```
Enter the number of vertices of the graph
6
Enter the Path adjacency matrix
0 1 1 1 0 0
1 0 1 0 0 1
1 1 0 1 1 0
1 0 1 0 1 0
0 0 1 1 0 1
0 1 0 0 1 0

 Hamiltonian Cycles are


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

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

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

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

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

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



DAA Laboratory viva questions

Module 1

INTRODUCTION

1. What is an algorithm?

An algorithm is a sequence of unambiguous instructions for solving a problem. i.e., for obtaining a required output for any legitimate input in a finite amount of time.

2. State the Euclid's algorithm for finding GCD of two given numbers. ALGORITHM Euclid (m, n)

```
// Computes gcd(m,n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
r← mod n
m←n
n←r
return m.
```

3. What do you mean by Amortized Analysis?

Amortized analysis finds the average running time per operation over a worst case sequence of operations. Amortized analysis differs from average-case performance in that probability is not involved; amortized analysis guarantees the time per operation over worst-case performance.

4. What are important problem types? (or) Enumerate some important types of problems.

1. Sorting	2. Searching
3. Numerical problems	4. Geometric problems
5. Combinatorial Problems	6. Graph Problems
7. String processing Problems	

5. Name some basic Efficiency classes

1. Constant 2. Logarithmic 3. Linear 4. nlogn

5. Quadratic 6. Cubic 7. Exponential 8. Factorial

6. What are algorithm design techniques?

Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmatically, applicable to a variety of problems from different areas of computing. General design techniques are:

(i) Brute force (ii) divide and conquer
(iii) decrease and conquer (iv) transform and concquer
(v) greedy technique (vi) dynamic programming
(vii) backtracking (viii) branch and bound

7. How is an algorithm's time efficiency measured?

Time efficiency indicates how fast the algorithm runs. An algorithm's time efficiency is measured as a function of its input size by counting the number of times its basic operation (running time) is executed. Basic operation is the most time consuming operation in the algorithm's innermost loop.

8. What is Big 'Oh' notation?

A function t(n) is said to be in O(g(n)), denoted t(n) O (g(n)) , if t(n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some nonnegative integers n_0 such that $t(n) \leq cg(n)$ for all $n \geq n_0$

9. What is an Activation frame?

It is a storage area for an invocation of recursive program (parameters, local variables, return address/value etc.). Activation frame allocated from frame stack pointed by frame pointer.

10. Define order of an algorithm

Measuring the performance of an algorithm in relation with the input size n is known as order of growth.

11. What is recursive call?

Recursive algorithm makes more than a single call to itself is known as recursive call. An algorithm that calls itself is direct recursive. An algorithm"A" is said to be indirect recursive if it calls another algorithm which in turn calls "A".

12. What do you mean by stepwise refinement?

In top down design methodology the problem is solved in sequence (step by step) is known as stepwise refinement.

13. How is the efficiency of the algorithm defined?

The efficiency of an algorithm is defined with the components.
(i) Time efficiency -indicates how fast the algorithm runs
(ii) Space efficiency -indicates how much extra memory the algorithm needs

14. Define direct recursive and indirect recursive algorithms.

Recursion occurs where the definition of an entity refers to the entity itself. Recursion can be direct when an entity refers to itself directly or indirect when it refers to other entities which refers to it. A (Directly) recursive routine calls itself. Mutually recursive routines are an example of indirect recursion. A (Directly) recursive data type contains pointers to instances of the data type.

15. What are the characteristics of an algorithm?

Every algorithm should have the following five characteristics

(i) Input (ii) Output (iii)Definiteness (iv) Effectiveness (v) Termination

Therefore, an algorithm can be defined as a sequence of definite and effective instructions, which terminates with the production of correct output from the given input. In other words, viewed little more formally, an algorithm is a step by step formalization of a mapping function to map input set onto an output set.

16. What do you mean by time complexity and space complexity of an algorithm?

Time complexity indicates how fast the algorithm runs. Space complexity deals with extra memory it require. Time efficiency is analyzed by determining the number of repetitions of the basic operation as a function of input size. Basic operation: the operation that contributes most towards the running time of the algorithm The running time of an algorithm is the function defined by the number of steps (or amount of memory) required to solve input instances of size n.

17. Define Big Omega Notation

\A function t(n) is said to be in Ω(g(n)) , denoted t(n) Є Ω((g(n)) , if t(n) is bounded below by some positive constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that $t(n) \geq cg(n)$ for all for all $n \geq n_0$

18. What are the different criteria used to improve the effectiveness of algorithm?

(i) The effectiveness of algorithm is improved, when the design, satisfies the following constraints to be minimum.Time efficiency - how fast an algorithm in question runs. Space efficiency – an extra space the algorithm requires
(ii) The algorithm has to provide result for all valid inputs.

19. Analyze the time complexity of the following segment:

```
for(i=0;i<N;i++)
for(j=N/2;j>0;j--)
sum++;
```

Time Complexity= (N * N) / 2= N^2 /2

Є O (N^2)

20. write general plan for analyzing non-recursive algorithms.

- Decide on parameter indicating an input's size.
- Identify the algorithm's basic operation
- Checking the no.of times basic operation executed depends on size of input.if it depends on some additional property,then best,worst,avg.cases need to be investigated
- Set up sum expressing the no. of times the basic operation is executed. (establishing order of growth)

21. How will you measure input size of algorithms?

The time taken by an algorithm grows with the size of the input. So the running time of the program depends on the size of its input. The input size is measured as the number of items in the input that is a parameter n is indicating the algorithm's input size.

22. Define the terms: pseudocode, flow chart

A pseudocode is a mixture of a natural language and programming language like constructs. A pseudocode is usually more precise than natural language. A flowchart is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's step

23. write general plan for analyzing recursive algorithms.

I. Decide on parameter indicating an input's size.
II. Identify the algorithm's basic operation
III. **Checking** the no.of times basic operation executed depends on size of input.if it depends on some additional property,then best,worst,avg.cases need to be investigated
IV. Set up the recurrence relation,with an appropriate initial condition,for the number of times the basic operation is executed
V. Solve recurrence (establishing order of growth)

24. What do you mean by Combinatorial Problem?

Combinatorial Problems are problems that ask to find a combinatorial object-such as permutation, a combination, or a subset--that satisfies certain constraints and has some desired property.

25. Define Big Theta Notations

A function t(n) is said to be in Θ(g(n)) , denoted t(n) Є Θ (g(n)) , if t(n) is bounded both above and below by some positive constant multiple of g(n) for all large n, i.e., if there exist some positive constants c1 and c2 and some nonnegative integer n0 such that c1 g(n) ≤t(n) ≤ c2g(n) for all n ≥ n0.

26. What is performance measurement?

Performance measurement is concerned with obtaining the space and the time requirements of a particular algorithm.

27. What is an algorithm?

An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:
1) input 2) Output 3) Definiteness 4) Finiteness 5)Effectiveness.

28. Define Program.

A program is the expression of an algorithm in a programming language. Sometimes works such as procedure, function and subroutine are used synonymously program.

29. What is recursive algorithm?

An algorithm is said to be recursive if the same algorithm is invoked in the body.

30. What is space complexity and time complexity ?

The space complexity of an algorithm is the amount of memory it needs to run to completion. The time complexity of an algorithm is the amount of computer time it needs to run to completion.

31. Give the two major phases of performance evaluation

Performance evaluation can be loosely divided into two major phases: (i) a prior estimates (performance analysis) (ii) a Posterior testing(performance measurement)

32. Define input size.

The input size of any instance of a problem is defined to be the number of words(or the number of elements) needed to describe that instance.

33. Define best-case step count.

The best-case step count is the minimum number of steps that can be executed for the given parameters.

34. Define worst-case step count.

The worst-case step count is the maximum number of steps that can be executed for the given parameters.

35. Define average step count.

The average step count is the average number of steps executed an instances with the given parameters.

36. Define Little "oh".

The function f(n) = 0(g(n)) iff

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

37. Define Little Omega.

The function f(n) = ω (g(n)) iff

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

38. Write algorithm using iterative function to fine sum of n numbers.

```
Algorithm sum(a,n)
{ S : = 0.0
For i=1 to n do S : - S
+ a[i]; Return S;    }
```

39. Write an algorithm using Recursive function to fine sum of n numbers,

```
Algorithm Rsum (a,n)
{
if(n≤0) then
Return 0.0;
Else Return Rsum(a, n- 1) + a(n);
```

}

40. What are Sequential Algorithms?

The central assumption of the RAM model is that instructions are executed one after another, one operation at a time. Accordingly, algorithms designed to be executed on such machines are called Sequential algorithms.

41. What are Parallel Algorithms?

The central assumption of the RAM model does not hold for some newer computers that can execute operations concurrently, i.e., in parallel algorithms that take advantage of this capability are called Parallel algorithms.

42. What is Exact and Approximation algorithm?

The principal decision to choose solving the problem exactly is called exact algorithm.The principal decision to choose solving the problem approximately is called Approximation algorithm.

43. What is Algorithm Design Technique? Nov/Dec 2005

An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.

44. Define Pseudo code.

A Pseudo code is a mixture of a natural language and programming language like constructs. A pseudo code is usually more precise than a natural language, and its usage often yields more succinct algorithm descriptions.

45. Define Flowchart.

A method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm’s steps.

46. Explain Algorithm’s Correctness

To prove that the algorithm yields a required result for every legitimate input in a finite amount of time.

Example: Correctness of Euclid’s algorithm for computing the greatest common divisor stems from correctness of the equality gcd (m, n) = gcd (n, m mod n).

47. What is generality of an algorithm?

It is a desirable characteristic of an algorithm. Generality of the problem the algorithm solves is sometimes easier to design an algorithm for a problem posed in more general terms.

48. What is algorithm’s Optimality?
Optimality is about the complexity of the problem that algorithm solves. What is the minimum amount of effort any algorithm will need to exert to solve the problem in question is called algorithm’s Optimality.

49. What do you mean by “Sorting” problem?
The sorting problem asks us to rearrange the items of a given list in ascending order (or descending order)

50. What do you mean by “Searching” problem?
The searching problem deals with finding a given value, called a search key, in a given set.

51. What do you mean by “Worst case-Efficiency” of an algorithm?
The ²Worst case-Efficiency” of an algorithm is its efficiency for the worst-case input of size n, which is an input (or inputs) of size n for which the algorithm runs the longest among all possible inputs of that size.
Ex: if you want to sort a list of numbers in ascending order when the numbers are given in descending order. In this running time will be the longest.

52. What do you mean by “Best case-Efficiency” of an algorithm?
The ²Best case-Efficiency” of an algorithm is its efficiency for the Best-case input of size n, which is an input(or inputs) of size n for which the algorithm runs the fastest among all possible inputs of that size.
Ex: if you want to sort a list of numbers in ascending order when the numbers are given in ascending order. In this running time will be the smallest.

53. Define the “Average-case efficiency” of an algorithm?
The ²Average-case efficiency” of an algorithm is its efficiency for the input of size n, for which the algorithm runs between the best case and the worst case among all possible inputs of that size.

54. What do you mean by “Amortized efficiency”?
The “Amortized efficiency” applies not only a single run of an algorithm but rather to a sequence of operations performed on the same data structure. It turns out that in some situations a single operation can be expensive ,but the total time for an entire sequence of n such operations is always significantly better than the worst case efficiency of that single operation multiplied by n. This is known as “Amortized efficiency

55. How to measure the algorithm’s efficiency?
It is logical to investigate the algorithm’s efficiency as a function of some parameter n indicating the algorithm’s input size.

Example: It will be the size of the list for problems of sorting, searching, finding the list's smallest element, and most other problems dealing with lists.

56. What is called the basic operation of an algorithm?

The most important operation of the algorithm is the operation contributing the most to the total running time is called basic operation of an algorithm.

57. How to measure an algorithm's running time?

Let C_{op} be the time of execution of an algorithm's basic iteration on a particular computer and let C (n) be the number of times this operation needs to be executed for this algorithm. Then we can estimate the running time T(n) of a program implementing this algorithm on that computer by the formula

$T(n) \approx C_{op} C(n)$

58. Define order of growth.

The efficiency analysis framework concentrates on the order of growth of an algorithm's basic operation count as the principal indicator of the algorithm's efficiency. To compare and rank such orders of growth we use three notations

- (Big oh) notation

- Ω (Big Omega) notation &

- Θ (Big Theta) notation

59. Prove that(½)n(n-1) Î Q(n^2)

1/2n(n-1)=(1/2)n^2-1/2n £ 1/2 n^2 for all n³0.(we have proved upper inequality) now 1/2n(n-1)=(1/2)n^2 - 1/2n³(1/2)n^2-1/2n*1/2n(for all n³2)=1/4 n^2 hence we can select c2=1/4,c1=1/2 and n0=2.

60. What is the use of Asymptotic Notations?

The notations O, W and Q and are used to indicate and compare the asymptotic orders of growth of functions expressing algorithm efficiencies

MODULE 2

DIVIDE AND CONQUER

1. Define the divide and conquer method.

Given a function to compute on 'n' inputs the divide-and-comquer strategy suggests splitting the inputs in to'k' distinct susbsets, 1<k <n, yielding 'k' subproblems. The subproblems must be solved, and then a method must be found to combine subsolutions into a solution of the whole. If the subproblems are still relatively large, then the divide-and conquer strategy can possibly be reapplied.

2. Define control abstraction.

A control abstraction we mean a procedure whose flow of control is clear but whose primary operations are by other procedures whose precise meanings are left undefined.
Write the Control abstraction for Divide-and conquer.

```
Algorithm DAndC(P)
{
if small(p) then return S(P);
else
{
divide P into smaller instance p1,p2,…pk, k ≥1; Apply DAndC to each of these subproblems
Return combine (DAnd C(p1) DAnd C(p2), ---, DAnd (pk));
}
}
```

3.What is the substitution method?

One of the methods for solving any such recurrence relation is called the substitution method.

4. What is the binary search?

If 'q' is always chosen such that 'q' is the middle element(that is, q=[(n+1)/2], then the resulting search algorithm is known as binary search.

5. Give computing time for Binary search?

In conclusion we are now able completely describe the computing time of binary search by giving formulas that describe the best, average and worst cases.
Successful searches

$\Theta(1)$	Θ (logn)	Θ (Logn)
best	average	worst

Unsuccessful searches
Θ (logn)

best, average, worst

6. Define external path length?

The external path length E, is defines analogously as sum of the distance of all external nodes from the root.

7. Define internal path length.

The internal path length 'I' is the sum of the distances of all internal nodes from the root.

8. What is the maximum and minimum problem?

The problem is to find the maximum and minimum items in a set of 'n' elements. Though this problem may look so simple as to be contrived, it allows us to demonstrate divideand-conquer in simple setting.

9. What is the Quick sort?

Quicksort is divide and conquer strategy that works by partitioning it's input elements according to their value relative to some preselected element(pivot). it uses recursion and the method is also called partition –exchange sort.

10. Write the Analysis for the Quick sort.

O(nlogn) in average and best cases

O(n^2) in worst case

11. what is Merge sort? and Is insertion sort better than the merge sort?

Merge sort is divide and conquer strategy that works by dividing an input array in to two halves,sorting them recursively and then merging the two sorted halves to get the original array sorted Insertion sort works exceedingly fast on arrays of less then 16 elements, though for large'n' its computing time is O(n^2).

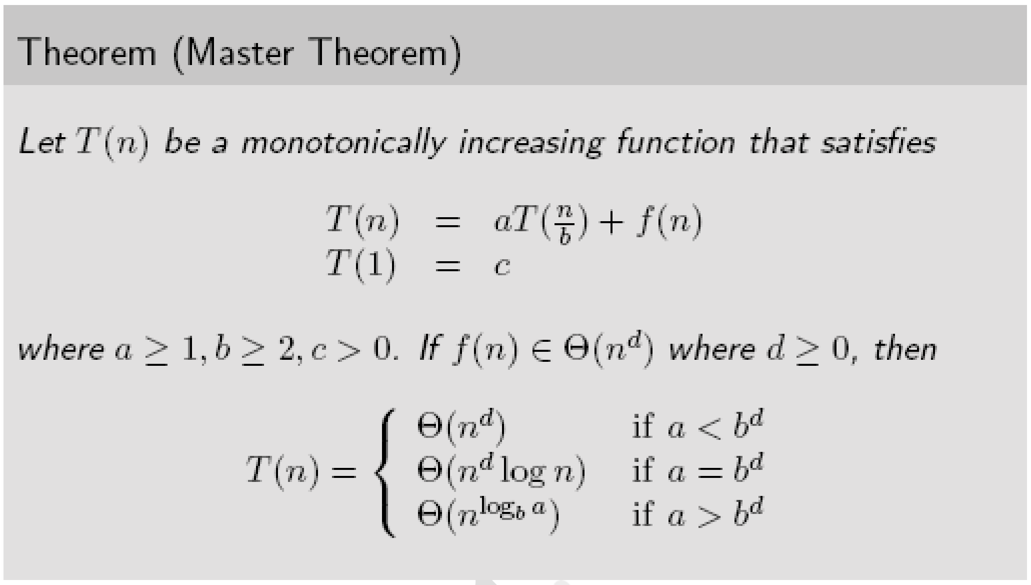
12. Write a algorithm for straightforward maximum and minimum?

```
Algorithm straight MaxMin(a,n,max,min)
//set max to the maximum and min to the minimum of a[1:n]
{
max := min: = a[i];
for i = 2 to n do
{
if(a[i] >max) then max: = a[i];
if(a[i] >min) then min: = a[i];
}     }
```

13. what is general divide and conquer recurrence?

Time efficiency T(n)of many divide and conquer algorithms satisfies the equation T(n)=a.T(n/b)+f(n).This is the general recurrence relation.

14. What is Master's theorem?



Theorem (Master Theorem)

Let $T(n)$ be a monotonically increasing function that satisfies

$$\begin{aligned} T(n) &= aT(\tfrac{n}{b}) + f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1, b \geq 2, c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

15. Write the algorithm for Iterative binary search?

```
Algorithm BinSearch(a,n,x)
//Given an array a[1:n] of elements in nondecreasing
// order, n>0, determine whether x is present
{
low : = 1;
high : = n;
while (low < high) do
{
mid : = [(low+high)/2];
if(x < a[mid]) then high:= mid-1;
else if (x >a[mid]) then low:=mid + 1;
else return mid; } } return 0;
```

16. Describe the recurrence relation for merge sort?

If the time for the merging operation is proportional to n, then the computing time of merge sort is described by the recurrence relation

$$T(n) = \begin{cases} a & n = 1, \quad a \text{ a constant} \\ 2T(n/2) + n & n > 1, \quad c \text{ a} \end{cases}$$

17. What can we say about the average case efficiency of binary search?

A sophisticated analysis shows that the average number of key comparisons made by binary search is only slightly smaller than that in the worst case

$C_{avg}(n) \approx \log_2 n$

18.Define Binary Tree

A binary tree T is defined as a finite set of nodes that is either empty or consists of s root and two disjoint binary trees TL, and T_R called, respectively the left and right subtree of the root.

19.How divide and conquer technique can be applied to binary trees?

Since the binary tree definition itself divides a binary tree into two smaller structures of the same type, the left subtree and the right subtree, many problems about binary trees can be solved by applying the divide-conquer technique.

20. Explain Internal and External Nodes

To draw the tree's extension by replacing the empty subtrees by special nodes.The extra nodes shown by little squares are called external. The original nodes shown by littile circles are called internal.

21. Define Preorder, inorder and postorder Traversal

PREORDER -The root is visited before the left and right subtrees are visited (in that order)
INORDER -The root is visited after visiting its left subtree but before visiting the right Subtree
POSTORDER - The root is visited after visiting the left and right subtrees(in that order)

DAA Laboratory viva questions

Module 2

Note: For answers refer theory notes

1. Explain divide and conquer method.
2. Explain Binary search algorithm with an example by constructing recursive call tree.justify how it uses divide and conquer method.
3. Explain Merge sort algorithm with an example by constructing recursive call tree.justify how it uses divide and conquer method.
4. Explain Quick sort algorithm with an example by constructing recursive call tree.justify how it uses divide and conquer method.
5. Differentiate Merge sort and quick sort.
6. what is the best,worst and average cases in merge sort?
7. what is the best,worst and average cases in quick sort?
8. Expalin with an example how stragens's matrix multiplication uses divide and conquer approach.write its time complexity.
9. list the problems that can be solved using divide and conquer method.
10. Explain how conquering happens in the quick sort.
11. Explain Decrease and conquer method with its variations.
12. Explain Topoloical sort using both OFS and source removal method with an example.justify how it uses decrease and conquer approach.
13. Differentiate divide and conquer,decrease and conquer.
14. List the problems that can be solved using decrease and conquer method.

MODULE 3

GREEDY METHOD

1. Explain the greedy method.

Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given 'n' inputs are required us to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one can device an algorithm that works in stages considering one input at a time.

2. Define feasible and optimal solution.

Given n inputs and we are required to form a subset such that it satisfies some given constraints then such a subset is called feasible solution. A feasible solution either maximizes or minimizes the given objective function is called as optimal solution.

3. Write the control abstraction for greedy method.

```
Algorithm Greedy (a, n)
{
solution=0;
for i=1 to n do
{
x= select(a);
if feasible(solution ,x) then solution=Union(solution ,x);
}
return solution;
}
```

4. What are the constraints of knapsack problem?

To maximize $\sum_{1 \le i \le n} p_i x_i$

The constraint is : $\sum_{1 \le i \le n} w_i x_i \ge m$ and $0 \le x_i \le 1$ $1 \le i \le n$

where m is the bag capacity, n is the number of objects and for each object i w_i and p_i are the weight and profit of object respectively.

5. What is a minimum cost spanning tree?

A spanning tree of a connected graph is its connected acyclic subgraph that contains all vertices of a graph. A minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight where bweight of the tree is the sum of weights on all its edges. A minimum spanning subtree of a weighted graph (G,w) is a spanning subtree of G of minimum weight w(T)= Σ w(e) e€ T Minimum Spanning Subtree Problem: Given a weighted connected undirected graph (G,w), find a minimum spanning subtree

6. Specify the algorithms used for constructing Minimum cost spanning tree.

a) Prim's Algorithm **b)** Kruskal's Algorithm

7. State single source shortest path algorithm (Dijkstra's algorithm).

For a given vertex called the source in a weigted connected graph,find shotrtest paths to all its other vertices.Dijikstra's algorithm applies to graph with non-negative weights only.

8. What is Knapsack problem?

A bag or sack is given capacity and n objects are given. Each object has weight wi and profit pi .Fraction of object is considered as xi (i.e) 0<=xi<=1 .If fraction is 1 then entire object is put into sack. When we place this fraction into the sack we get wixi and pixi.

9. Write any two characteristics of Greedy Algorithm?

To solve a problem in an optimal way construct the solution from given set of candidates. As the algorithm proceeds, two other sets get accumulated among this one set contains the candidates that have been already considered and chosen while the other set contains the candidates that have been considered but rejected.

10. What is the Greedy approach?

The method suggests constructing solution through sequence of steps,each expanding partially constructed solution obtained so far,until a complete solution is reached. On each step,the choice must be

- Feasible(satisfy problem constraints)
- Locally optimal(best local choice among all feasible choices available on that step)
- Irrevocable(once made,it cant be changed)

11. What are the steps required to develop a greedy algorithm?

1.Determine the optimal substructure of the problem.
2.Develop a recursive solution.
3.Prove that at any stage of recursion one of the optimal choices is greedy choice. Thus it is always safe to make greedy choice.
4.Show that all but one of the sub problems induced by having made the greedy choice are empty.
5.Develop a recursive algorithm and convert into iterative algorithm.

12. Define forest.

Collection of sub trees that are obtained when root node is eliminated is known as forest

13. Write the difference between the Greedy method and Dynamic programming.

Greedy method	Dynamic programming
Only one sequence of decision is	Many number of decisions are generated.
It does not guarantee to give an	It definitely gives an optimal solution

14.state the requirement in optimal storage problem in tapes.

Finding a permutation for the n programs so that when they are stored on the tape in this order the MRT is minimized.This problem fits the ordering paradigm.

15. state efficiency of prim's algorithm.

O(|v|^2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY) O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP).

16. State Kruskal Algorithm.

The algorithm looks at a MST for a weighted connected graph as an acyclic subgraph with |v|-1 edges for which the sum of edge weights is the smallest.

17. state efficiency of Dijkstra's algorithm.

O(|v|^2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY) O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

18. Differentiate subset paradigm and ordering paradigm

subset paradigm	ordering paradigm
At each stage a decision is made regarding whether a particular input is in an optimal solution (generating sub optimal	For problems that do not call for selection of optimal subset,in the greedy manner we make decisions by considering inputs in
Example kNAPSACK,MST	Optimal storage on tapes

19. Define Spanning Tree

A Spanning Tree of a connected graph is its connected acyclic subgraph(i.e., a tree) that contains all the vertices of the graph.

20. Define Minimum Spanning Tree

A minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight ,where the weight of a tree is defined as the sum of the weights on all its edges.

21. Define min-heap

A min-heap is a complete binary tree in which every element is less than or equal to its children. All the principal properties of heaps remain valid for min-heaps, with some obvious modifications.

22. Define Kruskal's Algorithm

Kruskal's algorithm looks at a minimum spanning tree for a weighted connected graph G =(V,E) as an acyclic subgraph with | V| - 1 edges for which the sum of the edge weights is the smallest.

23. Define Prim's Algorithm

Prim's algorithm is a greedy algorithm for constructing a minimum spanning tree of a weighted connected graph.It works by attaching to a previously constructed subtree a vertex to the vertices already in the tree.

24. Explain Dijkstra's Algorithm

Dijkstra's algorithm solves the single - source shortest - path problem of finding shortest paths from a given vertex (the source) to all the other vertices of a weighted graph or digraph. It works as prim's
algorithm but compares path lengths rather than edge lengths. Dijkstra's algorithm always yields a correct solution for a graph with nonnegative weights.

DAA Laboratory viva questions

Module 3

Note:For answers refer theory notes

1. Explain Greedy method.
2. List the different problems that can be solved using greedy method.
3.What is feasible solution?
4.what is optimal solution?
5.What is coin change problem?
6. What are different types of knapsack problems,explain with an example.justify how the knapsack problem uses greedy method.
7. Expain job requencing with dead lines with an example.justify how it uses greedy method.
8.What is spanning tree?Explian with an example.
9. What is minimum cost Spanning tree? Explian with an example.
10. Explain prims algorithm with an example .justify how it uses greedy method.write its complexity.
11. Explain Kruskal's algorithm with an example .justify how it uses greedy method.write its complexity.
12. what are the proceduaral difference between prims and kruskal's algorithm.
13. Explain single source shortest path problem using dijkstras algorithm with an example.justify how it uses greedy method.
14. How single source shortest path problem can be converted in to all pairs shortest path problem.justify with an example.
15. Give real time example of prime and krurkals algorithm.
16.What is Huffman trees and codes.
17. Explain transform and conquer method.
18. Define Heap,Min heap,Max heap.
19. Show the construction of max/min heap with an example.
20. Once the max/min heap constructed show how the inserution or deletion will taker place. explain with an example.
21. Explain the Heap sort with an example .justify how it uses transform and conquer approach.

MODULE 4

DYNAMIC PROGRAMMING

1. Write the difference between the Greedy method and Dynamic programming.

Greedy method	Dynamic programming
1.Only one sequence of decision is	1.Many number of decisions are generated.
2.It does not guarantee to give an optimal	2.It definitely gives an optimal solution

2. Define dynamic programming.

Dynamic programming is an algorithm design method that can be used when a solution to the problem is viewed as the result of sequence of decisions. It is technique for solving problems with overlapping subproblems.

3. What are the features of dynamic programming?

- Optimal solutions to sub problems are retained so as to avoid recomputing of their values.
- Decision sequences containing subsequences that are sub optimal are not considered.
- It definitely gives the optimal solution always.

4. What are the drawbacks of dynamic programming?

- Time and space requirements are high, since storage is needed for all level.
- Optimality should be checked at all levels.

5. Write the general procedure of dynamic programming.

The development of dynamic programming algorithm can be broken into a sequence of 4 steps.
1. Characterize the structure of an optimal solution.
2. Recursively define the value of the optimal solution.
3. Compute the value of an optimal solution in the bottom-up fashion.
4. Construct an optimal solution from the computed information.

6. Define principle of optimality.

It states that an optimal solution to any of its instances must be made up of optimal solutions to its subinstances.

7. Define multistage graph

A multistage graph G =(V,E) is a directed graph in which the vertices are partitioned in to K>=2 disjoint sets Vi,1<=i<=k.The multi stage graph problem is to find a minimum cost paths from s(source) to t(sink)Two approach(forward and backward)

8. Define All pair shortest path problem

Given a weighted connected graph, all pair shortest path problem asks to find the lengths of shortest paths from each vertex to all other vertices.

9.Define Distance matrix

Recording the lengths of shortest path in n x n matrix is called Distance matrix(D)

10.Define floyd's algorithm

To find all pair shortest path.The algorithm computes the distance matrix of a weighted graph with n vertices through series of n by n matrices :D(0)…D(k-1),D(k)…..D(n)

11. State the time efficiency of floyd's algorithm

$O(n^3)$ It is cubic

12. Define OBST

Dynammic pgmg. Used. If probabilities of searching for elements of a set are known then finding optimal BST for which the average number of comparisons in a search is smallest possible.

13. Define Catalan number

The total number of binary search trees with n keys is equal to nth Catalan number

C(n)=(2n to n) 1/(n+1) for n>0,c(0)=1

14. Define Transitive closure

The transitive closure of a directed graph with n vertices can be defined as the n by n Boolean matrix T = {ti,}, in which the element in the ith row ($1 \le i \le n$) and the j th column ($l \le j \le n$) is 1 if there exists a non trivial directed path from the ith vertex to jth vertex ; otherwise , t_{ij} is 0.

.

15. Explain Warshalls algorithm

Warshall's algorithm constructs the transitive closure of a given digraph with n vertices through a series of n by n Boolean matrices $R^{(0)}$, ………, $R^{(k-l)}R^{(k)}$,… , $R^{(n)}$
Each of these matrices provides certain information about directed paths in the digraph.

16. What does Floyd's algorithm do?

It is used to find the distances (the lengths of the shortest paths) from each vertex to all other vertices of a weighted connected graph.

17. Explain principle of Optimality

It says that an optimal solution to any instance of an optimization problem is composed of optimal solutions to its subinstances.

18. Explain Knapsack problem

Given n items of known weights w_1,w_2..........w_n and values v_1,v_2...........v_n and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack.(Assuming all the weights and the knapsack's capacity are positive integers the item values do not have to be integers.)

19.Explain the Memory Function technique

The Memory Function technique seeks to combine strengths of the top down and bottom-up approaches to solving problems with overlapping subproblems. It does this by solving, in the top-down fashion but only once, just necessary sub problems of a given problem and recording their solutions in a table.

20. Explain ²Traveling salesman problem"?

A salesman has to travel n cities starting from any one of the cities and visit the remaining cities exactly once and come back to the city where he started his journey in such a manner that either the distance is minimum or cost is minimum. This is known as traveling salesman problem.

DAA Laboratory viva questions

Module 4

Note:For answers refer theory notes

1. Explain Dynamic programming.
2. List the problems which can be solved using dynamic programming method.
3.Explain multistage graph with an example.
4.What is transitive closure of a graph? Explain with example.
5.What is adjacency matrix?
6.What are the different types of adjacency matrix?
7.Explain warshall's algorithm with an example.Justify how it uses dynamic programming method.
8. Explain Floud's Algorithm with an example.Justify how it uses dynamic programming method.
9.What is the outcome of warshall's algorithm?
10.What is the outcome of floyd's algorith
11.What is optimal binary search tree? How it can be differentiate with binary search tree justify with an example.
12.How knoaprack can be solved using dynamic programming?explain withan example.
13.suggest different methods which can be solved knaprack problem using dynamic programming?
14.What is the advantage of Bellman ford algorithm with dijkrtra's algorithm.
15.Explain travelling salesperson problem with an example.justify how it user dynamic programming method.
16.Explain Reliability design.

MODULE 5

BACKTRACKING

1. what are the requirements that are needed for performing Backtracking?

To solve any problem using backtracking, it requires that all the solutions satisfy a complex set of constraints. They are: i. Explicit constraints. ii. Implicit constraints.

i

2. Define state space tree.

The tree organization of the solution space is referred to as state space tree.

3. Define state space of the problem.

All the paths from the root of the organization tree to all the nodes is called as state space of the problem

4. Define answer states.

Answer states are those solution states s for which the path from the root to s defines a tuple that is a member of the set of solutions of the problem.

5. What are static trees?

The tree organizations that are independent of the problem instance being solved are called as static tree.

6. What are dynamic trees?

The tree organizations those are independent of the problem instance being solved are called as static tree.

7. Define a live node.

A node which has been generated and all of whose children have not yet been generated is called as a live node

8. Define a E – node.

E – node (or) node being expanded. Any live node whose children are currently being generated is called as a E –node.

9. Define a dead node.

Dead node is defined as a generated node, which is to be expanded further all of whose children have been generated.

10. Define Branch-and-Bound method.

The term Branch-and-Bound refers to all the state space methods in which all children of the E-node are generated before any other live node can become the E- node.

11. What are the searching techniques that are commonly used in Branch-and-Bound method

The searching techniques that are commonly used in Branch-and-Bound method.
I FIFO ii. LIFO iii. LC iv. Heuristic search

12. State 8 – Queens problem.

The problem is to place eight queens on a 8 x 8 chessboard so that no two queen "attack" that is, so that no two of them are on the same row, column or on the diagonal.

13. State Sum of Subsets problem.

Given n distinct positive numbers usually called as weights , the problem calls for finding all the combinations of these numbers whose sums are m.

14. State m – colorability decision problem.

Let G be a graph and m be a given positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used.

15. Define chromatic number of the graph.

The m – colorability optimization problem asks for the smallest integer m for which the graph G can be colored. This integer is referred to as the chromatic number of the graph.

16. Define a planar graph.

A graph is said to be planar iff it can be drawn in such a way that no two edges cross each other.

17. What are NP- hard and Np-complete problems?

The problems whose solutions have computing times are bounded by polynomials of small degree.

18. What is a decision problem?

Any problem for which the answer is either zero or one is called decision problem.

19. what is approximate solution?

A feasible solution with value close to the value of an optimal solution is called approximate solution.

20. what is promising and non-promising nodes?

a node in a state space tree is said to be promising if it corresponds to a partially constructed solution from which a complete solution can be obtained.

The nodes which are not promising for solution in a state space tree are called non- promising nodes.

21. Write formula for bounding function in Knapsack problem

In knapsack problem upper bound value is computed by the formula

UB = v + (W-w) * (vi+1/wi+1)

22. Write about traveling salesperson problem.

Let g = (V, E) be a directed. The tour of G is a directed simple cycle that includes every vertex in V. The cost of a tour is the sum of the cost of the edges on the tour. The traveling salesperson problem is to find a tour of minimum cost.In branch and bound technique of TSP problem Lower bound lb= s/2

23. Write some applications of traveling salesperson problem.

- ➔ Routing a postal van to pick up mail from boxes located at n different sites.
- ➔ Using a robot arm to tighten the nuts on some piece of machinery on an assembly line.
- ➔ Production environment in which several commodities are manufactured on the same set of machines.

24. Give the time complexity and space complexity of traveling salesperson problem.

Time complexity is O ($n^2 2^n$). Space complexity is O (n 2^n).

25. Differentiate decision problem and optimization problem

Any problem for which the answer is either zero or one is called decision problem.Any problem that involves the identification of an optimal (maximum or minimum) value of a given cost function is called optimization problems

26. what is class P and NP?

P is set of all decision problems solvable by deterministic algorithms in polynomial time. NP is set of all decision problems solvable by non deterministic algorithms in polynomial time.

27. Define NP-Hard and NP-Complete problems

Problem L is NP-Hard if and only if satisfiability reduces to L.
A Problem L is NP-Complete if and only if L is NP-Hard and L belongs to NP.

28. Explain promising and nonpromising node

A node in a state space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution;otherwise it is called nonpromising.

29. Explain n-Queens problem

The problem is to place n queens on an n by n chessboard so that no two queens attack each other by being in the same row or same column or on the same diagonal.

30. Explain Subset-Sum Problem

We consider the subset-sum problem: Find a subset of a given

set S={S1,S2,.........Sn} of n positive integers whose sum is equal to a given positive integer d.

31. Explain Branch and Bound Technique

Compared to backtracking, branch and bound requires .The idea to be strengthened further if we deal with an optimization problem, one that seeks to minimize or maximize an objective function, usually subject to some constraints.

32. Define e Feasible Solution

A feasible solution is a point in the problem's search space that satisfies all the problem's constraints.Ex: A Hamiltonian Circuit in the traveling salesman problem. A subset of items whose total weight does not exceed the knapsack's Capacity

33. Define Optimal solution

Is a feasible solution with the best value of the objective function

Eg: The shortest Hamiltonian Circuit m, The most valuable subset of items that fit the knapsack

34. Mention two reasons to terminate a search path at the current node in a state-space tree of a branch and bound algorithm.

The value of the node's bound is not better than the value of the best solution seen so far. The node represents no feasible solutions because the constraints of the problem are already violated.

35. Explain ²Graph coloring” problem.

The graph coloring problem asks us to assign the smallest number of colors to vertices of a graph so that no two adjacent vertices are the same color.

36. Explain Knapsack Problem

Find the most valuable subset of n items of given positive integer weights and values that fit into a knapsack of a given positive integer capacity.

37. Define tractable and intractable problems

Problems that can be solved in polynomial time are called tractable problems, problems that cannot be solved in polynomial time are called intractable problems.

38. Explain the theory of computational complexity

A problem's intractability remains the same for all principal models of computations and all reasonable input encoding schemes for the problem under consideration

39. Explain class P problems

Class P is a class of decision problems that can be solved in polynomial time by(deterministic) algorithms. This class of problems is called polynomial.

40. Explain undecidable problems

If the decision problem cannot be solved in polynomial time, and if the decision problems cannot be solved at all by any algorithm. Such problems are called Undecidable.

41. Explain the halting problem

Given a computer program and an input to it,determine whether the program will halt on that input or continue working indefinitely on it.

42. Explain class NP problems

Class NP is the class of decision problems that can be solved by nondeterministic polynomial algorithms.Most decision problems are in NP. First of all, this class includes all the problems in P. This class of problems is called Nondeterministic polynomial.

43. Explain NP-complete problems

A decision problem d is said to be NP-complete if

1) it belongs to class NP 2)every problem in NP is polynomially reducible to D.

44. When a decision problem is said to be polynomially reducible

A decision problem Dl is said to be polynomially reducible to a decision problem D2 if there exists a function t that transforms instances of Dl to instances ofD2 such that

i) t maps all yes instances of d1 to yes instances odf d2 and all no instances of dl to no instances of d2

ii) t is computable by a polynomial time algorithm

45. Define a Heuristic

A heuristic is a common-sense rule drawn from experience rather than from a mathematically proved assertion. Ex: Going to the nearest unvisited city in the traveling salesman problem is a good illustration for Heuristic

46. Explain NP-Hard problems

The notion of an NP-hard problem can be defined more formally by extending the notion of polynomial reducability to problems that are not necessary in class NP including optimization problems.

47. Define Traversals.

When the search necessarilyinvolves the examination of every vertex in the object being searched it is called a traversal.

48. List out the techniques for traversals in graph.

i) Breadth first search ii) Depth first search

49. What is articulation point.

A vertex v in a connected graph G is an articulation point if and only if the deletion of vertex v together with all edged incident to v disconnects the graph in to two or more nonempty components.

DAA Laboratory viva questions

Module 5

Note:For answers refer theory notes

1. Explain Backtracking.

2. List the problems that can be solved using backtracking method.

3. Explain N-queens problem, write state space tree for 4_queens problem and justify How it uses backtracking method.

4. Explain sum subset problem, write state space tree and justify how it uses backtracking method.

5. Explain Graph closing, write state space tree and justify how it uses backtracking method.

6. Explain Hamiltonian cycle problem, write state space tree and justify how it uses backtracking method.

7. Explain branch and bound.

8. List the different problems that can be solved using branch and bound method.

9. Differentiate Branch & bound, Backtracking.