INTRODUCTION

An algorithm is a description of a procedure which terminates with a result. It is a step by step procedure designed to perform an operation, and which will lead to the sought result if followed correctly. Algorithms have a definite beginning and a definite end, and a finite number of steps. An algorithm produces the same output information given the same input information, and several short algorithms can be combined to perform complex tasks.

PROPERTIES OF ALGORITHMS

1. Finiteness – an algorithm must terminate after a finite number of steps.
2. Definiteness – each step must be precisely defined.
3. Effective – all operations can be carried out exactly in finite time.
4. Input – an algorithm has one or more outputs.

MODELS OF COMPUTATION

There are many ways to compute the algorithm's complexity; they are all equivalent in some sense.

1. Turing machines.
2. Random access machines (RAM).
3. λ Calculus.
4. Recursive functions.
5. Parallel RAM (PRAM).

For the analysis of algorithms, the RAM model is most often employed. Note time does not appear to be reusable, but space often is.

MEASURING AN ALGORITHM'S COMLEXITY

General consideration of time and space is:

- One time unit per operation.
- One space unit per value (all values fit in fixed size register).

There are other considerations:

- On real machines, different operations typically take different times to execute. This will generally be ignored, but sometimes we may wish to count different types of operations, e.g., Swaps and compares, or additions and multiplications.
- Some operations may be "noise" not truly affecting the running time; we typically only count "major" operations example, for loop index arithmetic and boundary checking is "noise." operations inside for loops are usually "major."
- Logarithmic cost model: it takes $\lfloor lgn \rfloor + 1$ bits to represent natural number *n* in binary notation. Thus the uniform model of time and space may not bias results for large integers (data).

An algorithm solves an instance of a problem. There is, in general, one parameter, the input size, denoted by *n*, which is used to characterize the problem instance. The input size *n* is the number of registers needed to hold input (data segment size).

Given n, we'd like to find:

1. The time complexity, denoted by T(n), which is the count of operations the algorithm performs on the given input.
2. The space complexity, denoted by S(n), which is the number of memory registers used by the algorithm (stack/heap size, registers).

Note that T(n) and S(n) are relations rather than functions. That is, for different input of the same size n, T(n) and S(n) may provide different answers.

WORST, AVERAGE, BEST, AND MORTIZED COMPLEXITY

Complexities usually not measured exactly: big- O, Ω, and Θ notation is used.

WORST CASE:

This is the longest time (or most space) that the algorithm will use over all instances of size n. Often this can be represented by a function f(n) such as **$f(n)=n^2$** or **f(n)=n lg n.** We write T(n)= O(f(n)) for the worst case time complexity. Roughly, this means the algorithm will take mo more than f(n) operations.

BEST CASE:

This is the shortest time that the algorithm will use over all instances of size n. often this can be represented by a function f(n) such as **$f(n)=n^2$** or **f(n)=n lg n**. We write **T(n)= Ω (f(n))** for the best case. Roughly, this means the algorithm will take no less than f(n) operations. The best case is seldom interesting.

When the worst and best case performance of an algorithm are the same we can write **T(n)= Θ (f(n))**. Roughly, this says the algorithm always uses f(n) operations on all instances of size n.

AVERAGE CASE:

This is the average time that the algorithm will use over all instances if size n. it depends on the probability distribution of instances of the problem.

AMORTIZED COST:

This is used when a sequence of operations occur, e.g., inserts and deletes in a tress, where the costs vary depending on the operations and their order. For example, some may take a few steps, some many.

TYPES OF ALGORITHMS

1. Off-line algorithms: all input in memory before time starts, want final result.
2. On-line: input arrives at discrete time steps, intermediate result furnished before next input.
3. Real-time: elapsed time between two inputs (outputs) is a constant O(1).

COMPLEXITY CLASSES

Collection of problems that required roughly the same amount of resources from complexity classes. Here is a list of the most important:

1. The class P of problems that can be solved in a polynomial number of operations of the input size on a deterministic Turing machine.
2. The class NP of problems that can be solved in a polynomial number of operations of the input size on a non-deterministic Turing machine.

3. The class of problems that can be solved in a constant amount of space.
4. The class L that can be solved in a logarithmic amount of space based on the input size.
5. The class PSPACE of problems that can be solved in a polynomial amount of space based on the input size.
6. The class NC of problems that can be solved in poly-logarithmic time on a polynomial number of processors.

ALGORITHM PARADIGMS

Often there are large collections of problems that can be solved using the same general techniques or paradigms. A few of the most common are described below:

Brute Force:

A straightforward approach to solving a problem based on the problem statement and concepts involved. Brute force algorithms are rarely efficient. Example algorithms include:

- Bubble sort.
- Computing the sum of n numbers by direct addition.
- Standard matrix multiplication.
- Linear search.

Divide and Conquer:

Perhaps the most famous algorithm paradigm, divide and conquer is based on partitioning the problem into two or more smaller sub-problems, solving them and combining the sub-problem solutions into a solution for the original problem. Example algorithms include:

- Merge sort and quick sort.
- The Fast Fourier Transform (FFT).
- Stassen's matrix multiplication.

Greedy Algorithms:

Greedy algorithms always make the choice that seems best at the moment. This is locally optimal choice is made with the hope that it leads to a globally optimal solution. Some greedy algorithms may not be guaranteed to always produce an optimal solution.

Greedy algorithms are often applied to combinatorial optimization problems.

- Given an instance 1 of the problem.
- There is a set of candidates or feasible solutions that satisfy the constraints of the problem.
- For each feasible solution there is a value determined byan objective function.
- An optimal solution minimizes (or maximizes) the value of objective function.

Example algorithms include:

- Kruskal's and Prim's minimal spanning tree algorithms.
- Dijkstra's single source shortest path algorithm.
- Huffman coding.

Dynamic Programming:

A nutshell definition of dynamic programming is difficult, but to summarize, problems which lend themselves to a dynamic programming attack have the following characteristics:

- We have to search over a large space for an optimal solution.
- The optimal solution can be expressed in terms of optimal solution to sub-problem.
- The number of sub-problems that must be solved is small.

Dynamic programming algorithms have the following features:

- A recurrence that is implemented iteratively.
- A table, built to support the iteration.
- Tracing through the table to find the optimal solution.

Example algorithms include:

- Efficient Fibonacci number computation.
- The Floyd's, warshall's, all pairs shortest path algorithm.
- The minimal edit algorithm.

Optimal polygon triangulation.

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

```
import java.util.Scanner;

class Student {

        String USN, Name, Branch, Phone;

        Scanner input = new Scanner(System.in);

        void read() {
                System.out.println("Enter Student Details");
                System.out.println("Enter USN");
                USN = input.nextLine();

                System.out.println("Enter Name");
                Name = input.nextLine();

                System.out.println("Enter Branch");
                Branch = input.nextLine();

                System.out.println("Enter Phone");
                Phone = input.nextLine();
        }

        void display() {
                System.out.printlnUSN+"\t"+ Name+"\t"+Branch+"\t"+Phone);
        }
}

class studentdetails {
        public static void main(String[] args) {

                Scanner input = new Scanner(System.in);
                System.out.println("Enter number of student details to be created");
                int number = input.nextInt();

                Student s[] = new Student[number];

                // Read student details into array of student objects

                for (int i = 0; i < number; i++) {
                        s[i] = new Student();
                        s[i].read();
                }
```

```
                // Display student information

                System.out.println( "USN\tNAME\tBRANCH\tPHONE");
                for (int i = 0; i < number; i++) {
                        System.out.println();
                        s[i].display();
                }
         input.close();
        }
}
```

Output:

Enter Number of Students: 3

Enter Student Details

Enter Student USN: 1DT14cs001

Enter Student NAME: AAA

Enter Student BRANCH: CSE

Enter Student PHONENUMBER: 9999900000

Enter Student Details

Enter Student USN:1DT14cs002

Enter Student NAME: BBB

Enter Student BRANCH: CSE

Enter Student PHONENUMBER: 9999911111

Enter Student Details

Enter Student USN: 1DT14CS003

Enter Student NAME: CCC

Enter Student BRANCH: CSE

Enter Student PHONENUMBER: 9999922222

USN	NAME	BRANCH	PHONENUMBER
1DT14cs001	AAA	CSE	9999900000
1DT14cs002	BBB	CSE	9999911111
1DT14CS003	CCC	CSE	9999922222

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

```
import java.util.*;

class arrayStack {
	int arr[];
	int top, max;

	arrayStack(int n) {
		max = n;
		arr = new int[max];
		top = -1;
	}

	void push(int i) {
		if (top == max - 1)
			System.out.println("Stack Overflow");
		else
			arr[++top] = i;
	}

	void pop() {
		if (top == -1) {
			System.out.println("Stack Underflow");
		} else {
			int element = arr[top--];
			System.out.println("Popped Element:  " + element);
		}
	}

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

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

```
                arrayStack stk = new arrayStack(n);

                char ch;
                do {
                        System.out.println("\nStack Operations");
                        System.out.println("1. push");
                        System.out.println("2. pop");
                        System.out.println("3. display");
                        System.out.println("4. Exit");

                        int choice = scan.nextInt();
                        switch (choice) {
                        case 1:
                                System.out.println("Enter integer element to push");
                                stk.push(scan.nextInt());
                                break;

                        case 2:
                                stk.pop();
                                break;

                        case 3:
                                stk.display();
                                break;

                        case 4:
                                done = true;
                                break;

                        default:
                                System.out.println("Wrong Entry \n ");
                                break;

                        }

                } while (!done);
        }
}
```

Output:

Enter the stack size : 4

Enter the stack operation :

1.Push

2.Pop

3.Dispaly

```
4.Exit
1
Enter the element to be pushed: 10
Enter the stack operation: 1
Enter the element to be pushed: 20
Enter the stack operation: 1
Enter the element to be pushed: 30
Enter the stack operation: 1
Enter the element to be pushed: 40
Enter the stack operation: 1
Enter the element to be pushed: 50
                        stack is full
Enter the stack operation:3
Elements in stack :
10
20
30
40
Enter the stack operation:2
Deleted element is:40
Enter the stack operation:2
Deleted element is:30
Enter the stack operation:2
Deleted element is:20
Enter the stack operation:2
Deleted element is:10
Enter the stack operation:2
                        stack is empty
Enter the stack operation:3
                        stack is empty
Enter the stack operation :4
```

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

```
import java.util.Scanner;

class Staff {
	String StaffID, Name, Phone, Salary;

	Scanner input = new Scanner(System.in);

	void read() {
		System.out.println("Enter StaffID");
		StaffID = input.nextLine();

		System.out.println("Enter Name");
		Name = input.nextLine();

		System.out.println("Enter Phone");
		Phone = input.nextLine();

		System.out.println("Enter Salary");
		Salary = input.nextLine();
	}

	void display() {
		System.out.printf("\n%-15s", "STAFFID: ");
		System.out.printf("%-15s \n", StaffID);
		System.out.printf("%-15s", "NAME: ");
		System.out.printf("%-15s \n", Name);
		System.out.printf("%-15s", "PHONE:");
		System.out.printf("%-15s \n", Phone);
		System.out.printf("%-15s", "SALARY:");
		System.out.printf("%-15s \n", Salary);
	}
}


class Teaching extends Staff {
	String Domain, Publication;

	void read_Teaching() {
		super.read(); // call super class read method
		System.out.println("Enter Domain");
		Domain = input.nextLine();
		System.out.println("Enter Publication");
		Publication = input.nextLine();
```

```
        }

        void display() {
                super.display(); // call super class display() method
                System.out.printf("%-15s", "DOMAIN:");
                System.out.printf("%-15s \n", Domain);
                System.out.printf("%-15s", "PUBLICATION:");
                System.out.printf("%-15s \n", Publication);
        }
}


class Technical extends Staff {
        String Skills;

        void read_Technical() {
                super.read(); // call super class read method
                System.out.println("Enter Skills");
                Skills = input.nextLine();
        }

        void display() {
                super.display(); // call super class display() method
                System.out.printf("%-15s", "SKILLS:");
                System.out.printf("%-15s \n", Skills);
        }
}


class Contract extends Staff {
        String Period;

        void read_Contract() {
                super.read(); // call super class read method
                System.out.println("Enter Period");
                Period = input.nextLine();
        }

        void display() {
                super.display(); // call super class display() method
                System.out.printf("%-15s", "PERIOD:");
                System.out.printf("%-15s \n", Period);
        }
}

class Staffdetails {
        public static void main(String[] args) {

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

```
System.out.println("Enter number of staff details to be created");
int n = input.nextInt();

Teaching steach[] = new Teaching[n];
Technical stech[] = new Technical[n];
Contract scon[] = new Contract[n];

// Read Staff information under 3 categories

for (int i = 0; i < n; i++) {
        System.out.println("Enter Teaching staff information");
        steach[i] = new Teaching();
        steach[i].read_Teaching();
}

for (int i = 0; i < n; i++) {
        System.out.println("Enter Technical staff information");
        stech[i] = new Technical();
        stech[i].read_Technical();
}

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

        System.out.println("Enter Contract staff information");
        scon[i] = new Contract();
        scon[i].read_Contract();
}

// Display Staff Information
System.out.println("\n STAFF DETAILS: \n");
System.out.println("-----TEACHING STAFF DETAILS----- ");

for (int i = 0; i < n; i++) {
        steach[i].display();
}

System.out.println();
System.out.println("-----TECHNICAL STAFF DETAILS-----");
for (int i = 0; i < n; i++) {
        stech[i].display();
}

System.out.println();
System.out.println("-----CONTRACT STAFF DETAILS-----");
for (int i = 0; i < n; i++) {
        scon[i].display();
}
```

```
                input.close();
            }
        }
```

Output:

```
run:
Enter number of staff details to be created
3
Enter Teaching staff information
Enter StaffID
11
Enter Name
aaa
Enter Phone
9999900000
Enter Salary
100000
Enter Domain
Network
Enter Publication
4
Enter Teaching staff information
Enter StaffID
22
Enter Name
BBB
Enter Phone
9999911111
Enter Salary
100000
Enter Domain
Java
Enter Publication
3
Enter Teaching staff information
Enter StaffID
33
Enter Name
CCC
Enter Phone
9999922222
Enter Salary
100000
Enter Domain
C++
Enter Publication
5
Enter Technical staff information
Enter StaffID
```

```
44
Enter Name
DDD
Enter Phone
9999933333
Enter Salary
10000
Enter Skills
Programing
Enter Technical staff information
Enter StaffID
55
Enter Name
EE
Enter Phone
9999944444
Enter Salary
20000
Enter Skills
C++ Prog
Enter Technical staff information
Enter StaffID
66
Enter Name
FF
Enter Phone
9999966666
Enter Salary
30000
Enter Skills
Java Prog
Enter Contract staff information
Enter StaffID
77
Enter Name
XYZ
Enter Phone
9999977777
Enter Salary
10000
Enter Period
4
Enter Contract staff information
Enter StaffID
88
Enter Name
GGG
Enter Phone
9999988888
```

```
Enter Salary
10000
Enter Period
3
Enter Contract staff information
Enter StaffID
99
Enter Name
HHH
Enter Phone
99999010101
Enter Salary
20000
Enter Period
5

 STAFF DETAILS:

-----TEACHING STAFF DETAILS-----

STAFFID:      11
NAME:         aaa
PHONE:        9999900000
SALARY:       100000
DOMAIN:       Network
PUBLICATION:  4

STAFFID:      22
NAME:         BBB
PHONE:        9999911111
SALARY:       100000
DOMAIN:       Java
PUBLICATION:  3

STAFFID:      33
NAME:         CCC
PHONE:        9999922222
SALARY:       100000
DOMAIN:       C++
PUBLICATION:  5

-----TECHNICAL STAFF DETAILS-----

STAFFID:      44
NAME:         DDD
PHONE:        9999933333
SALARY:       10000
SKILLS:       Programing
```

```
STAFFID:    55
NAME:       EE
PHONE:      9999944444
SALARY:     20000
SKILLS:     C++ Prog

STAFFID:    66
NAME:       FF
PHONE:      9999966666
SALARY:     30000
SKILLS:     Java Prog

-----CONTRACT STAFF DETAILS-----

STAFFID:    77
NAME:       XYZ
PHONE:      9999977777
SALARY:     10000
PERIOD:     4

STAFFID:    88
NAME:       GGG
PHONE:      9999988888
SALARY:     10000
PERIOD:     3

STAFFID:    99
NAME:       HHH
PHONE:      99999010101
SALARY:     20000
PERIOD:     5
BUILD SUCCESSFUL (total time: 4 minutes 32 seconds)
```

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

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

public class Customer {
	public static void main(String[] args) {
		String name;
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter Name and Date_of_Birth in the format <Name,DD/MM/YYYY>");
		name = scan.next();

		// create stringTokenizer with delimiter "/"
		StringTokenizer st = new StringTokenizer(name, ",/");

		// Count the number of tokens
		int count = st.countTokens();

		// Print one token at a time and induce new delimiter ","
		for (int i = 1; i <= count && st.hasMoreTokens(); i++) {
			System.out.print(st.nextToken());
			if (i < count)
				System.out.print(",");
		}
	}
}
```

Output:

```
Enter Name and Date_of_Birth in the format <Name,DD/MM/YYYY>
AAA,30/06/1989
AAA,30,06,1989
```

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

```
import java.util.Scanner;

class exception {

public static void main(String[] args) {
        int a, b, result;

        Scanner input = new Scanner(System.in);
        System.out.println("Input two integers");

        a = input.nextInt();
        b = input.nextInt();

        try {
                result = a / b;
                System.out.println("Result = " + result);
        }

        catch (ArithmeticException e) {
                System.out.println("Exception caught: Division by zero.");
        }
}
}
```

Output:

```
Input two integers
10
2
Result = 5


Input two integers
10
0
Exception caught: Division by zero.
```

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

```
import java.util.Random;

class SquareThread implements Runnable {
  int x;

  SquareThread(int x) {
        this.x = x;
  }

  public void run() {
        System.out.println("Thread Name:Square Thread and Square of " + x + " is: "
+ x * x);
  }
}

class CubeThread implements Runnable {
  int x;

  CubeThread(int x) {
        this.x = x;
  }

  public void run() {
        System.out.println("Thread Name:Cube Thread and Cube of " + x + " is: " + x
* x * x);
  }
}

class RandomThread implements Runnable
{
  Random r;
```

```
    Thread t2, t3;

    public void run() {
            int num;
            r = new Random();
            try {

                boolean flag=true;
                int count=0;

                    while (flag) {
                            num = r.nextInt(100);
                    System.out.println("Main Thread and Generated Number is " + num);
                            t2 = new Thread(new SquareThread(num));
                            t2.start();

                            t3 = new Thread(new CubeThread(num));
                            t3.start();

                            Thread.sleep(1000);
                            System.out.println("--------------------------------------");
                        count++;
                             if (count==10)
                                flag=false;
                    }
            } catch (Exception ex) {
                    System.out.println("Interrupted Exception");
            }
    }
}


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

```
            RandomThread thread_obj = new RandomThread();
            Thread t1 = new Thread(thread_obj);
            t1.start();
    }
}
```

Output:

```
Main Thread and Generated Number is 73
Thread Name:Square Thread and Square of 73 is: 5329
Thread Name:Cube Thread and Cube of 73 is: 389017
-------------------------------------
Main Thread and Generated Number is 96
Thread Name:Square Thread and Square of 96 is: 9216
Thread Name:Cube Thread and Cube of 96 is: 884736
-------------------------------------
Main Thread and Generated Number is 30
Thread Name:Square Thread and Square of 30 is: 900
Thread Name:Cube Thread and Cube of 30 is: 27000
-------------------------------------
Main Thread and Generated Number is 77
Thread Name:Square Thread and Square of 77 is: 5929
Thread Name:Cube Thread and Cube of 77 is: 456533
-------------------------------------
Main Thread and Generated Number is 63
Thread Name:Square Thread and Square of 63 is: 3969
Thread Name:Cube Thread and Cube of 63 is: 250047
-------------------------------------
Main Thread and Generated Number is 97
Thread Name:Square Thread and Square of 97 is: 9409
Thread Name:Cube Thread and Cube of 97 is: 912673
-------------------------------------
Main Thread and Generated Number is 75
Thread Name:Square Thread and Square of 75 is: 5625
Thread Name:Cube Thread and Cube of 75 is: 421875
-------------------------------------
Main Thread and Generated Number is 79
Thread Name:Square Thread and Square of 79 is: 6241
Thread Name:Cube Thread and Cube of 79 is: 493039
-------------------------------------
Main Thread and Generated Number is 22
Thread Name:Square Thread and Square of 22 is: 484
Thread Name:Cube Thread and Cube of 22 is: 10648
-------------------------------------
Main Thread and Generated Number is 31
```

```
Thread Name:Square Thread and Square of 31 is: 961
Thread Name:Cube Thread and Cube of 31 is: 29791
-------------------------------------
Main Thread and Generated Number is 37
Thread Name:Square Thread and Square of 37 is: 1369
Thread Name:Cube Thread and Cube of 37 is: 50653
-------------------------------------
Main Thread and Generated Number is 98
Thread Name:Square Thread and Square of 98 is: 9604
Thread Name:Cube Thread and Cube of 98 is: 941192
-------------------------------------
Main Thread and Generated Number is 9
Thread Name:Square Thread and Square of 9 is: 81
Thread Name:Cube Thread and Cube of 9 is: 729
-------------------------------------
```

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

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

public class QuickSortComplexity {
  static final int MAX = 200000;
  static int[] a = new int[MAX];
  public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.print("Enter Max array size: ");
        int n = input.nextInt();
        Random random = new Random();
        System.out.println("Enter the array elements: ");
        for (int i = 0; i < n; i++)
              // a[i] = input.nextInt();   // for keyboard entry
              a[i] = random.nextInt(10000); // generate
    // random numbers ñ uniform distribution

        // a = Arrays.copyOf(a, n); // keep only non zero elements
        // Arrays.sort(a); // for worst-case time complexity

        System.out.println("Input Array:");
        for (int i = 0; i < n; i++)
              System.out.print(a[i] + " ");
        // set start time
        long startTime = System.nanoTime();
        QuickSortAlgorithm(0, n - 1);
        long stopTime = System.nanoTime();
        long elapsedTime = stopTime - startTime;
        System.out.println("\nSorted Array:");
        for (int i = 0; i < n; i++)
              System.out.print(a[i] + " ");
        System.out.println();
        System.out.println("Time Complexity in ms for
            n=" + n + " is: " + (double) elapsedTime / 1000000);
  }

  public static void QuickSortAlgorithm(int p, int r) {
        int i, j, temp, pivot;
        if (p < r) {
              i = p;
```

```
                j = r + 1;
                pivot = a[p]; // mark first element as pivot
                while (true) {
                        i++;
                        while (a[i] < pivot && i < r)
                                i++;
                        j--;
                        while (a[j] > pivot)
                                j--;
                        if (i < j) {
                                temp = a[i];
                                a[i] = a[j];
                                a[j] = temp;
                        } else
                                break; // partition is over
                }
                a[p] = a[j];
                a[j] = pivot;
                QuickSortAlgorithm(p, j - 1);
                QuickSortAlgorithm(j + 1, r);
        }
  }
}
```

Output

```
Enter Max array size: 20
Enter the array elements:
Input Array:
326 719 983 701 490 230 595 474 341 75 916 173 324 852 728 434 758 445 303 566
Sorted Array:
75 173 230 303 324 326 341 434 445 474 490 566 595 701 719 728 758 852 916 983
Time Complexity in ms for n=20 is: 0.023225

Enter Max array size: 20000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=20000 is: 4.953809

Enter Max array size: 30000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=30000 is: 7.141865

Enter Max array size: 40000
Enter the array elements:
Input Array:
Sorted Array:
```

Time Complexity in ms for n=40000 is: 8.698231

Enter Max array size: 50000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=50000 is: 9.103897

Enter Max array size: 60000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=60000 is: 12.380137

Enter Max array size: 70000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=70000 is: 24.719828

Enter Max array size: 80000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=80000 is: 21.150887

Enter Max array size: 90000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=90000 is: 35.894418

Enter Max array size: 100000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=100000 is: 31.430762

Enter Max array size: 200000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=200000 is: 47.498161

Plot Graph: time taken versus n on graph sheet

Time Complexity Analysis:

Quick Sort Algorithm
Average performance O(n log n)

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.

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

public class MergeSort{
        static final int MAX = 200000;
        static int[] a = new int[MAX];

        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.print("Enter Max array size: ");
                int n = input.nextInt();
                Random random = new Random();
                System.out.println("Enter the array elements: ");
                for (int i = 0; i < n; i++)
        {
                //      a[i] = input.nextInt(); // for keyboard entry
                 a[i] = random.nextInt(100000);
               System.out.print(a[i] + " ");
        }
                long startTime = System.nanoTime();
                MergeSortAlgorithm(0, n - 1);
                long stopTime = System.nanoTime();
                long elapsedTime = stopTime - startTime;
                System.out.println("Time Complexity (ms) for n = " +
n + " is : " + (double) elapsedTime / 1000000);
                System.out.println("Sorted Array (Merge Sort):");
                for (int i = 0; i < n; i++)
                        System.out.print(a[i] + " ");
                input.close();
        }

        public static void MergeSortAlgorithm(int low, int high) {
                int mid;
                if (low < high) {
                        mid = (low + high) / 2;
                        MergeSortAlgorithm(low, mid);
                        MergeSortAlgorithm(mid + 1, high);
                        Merge(low, mid, high);
                }
        }
```

```
        public static void Merge(int low, int mid, int high) {
                int[] b = new int[MAX];
                int i, h, j, k;
                h = i = low;
                j = mid + 1;
                while ((h <= mid) && (j <= high))
                        if (a[h] < a[j])
                                b[i++] = a[h++];
                        else
                                b[i++] = a[j++];

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

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

Output

```
Enter Max array size: 5
Enter the array elements:
856 604 528 287 321 Time Complexity (ms) for n = 5 is : 0.090071
Sorted Array (Merge Sort):
287 321 528 604 856

Enter Max array size: 10000
Enter the array elements:
Time Complexity (ms) for n = 10000 is : 1194.135767
Sorted Array (Merge Sort):

Enter Max array size: 20000
Enter the array elements:
Time Complexity (ms) for n = 20000 is : 2040.96632
Sorted Array (Merge Sort):

Enter Max array size: 30000
Enter the array elements:
Time Complexity (ms) for n = 30000 is : 3098.642188
Sorted Array (Merge Sort):

Enter Max array size: 40000
Enter the array elements:
Time Complexity (ms) for n = 40000 is : 3914.650313
```

Sorted Array (Merge Sort):

Enter Max array size: 50000
Enter the array elements:
Time Complexity (ms) for n = 50000 is : 4700.729745
Sorted Array (Merge Sort):

Enter Max array size: 60000
Enter the array elements:
Time Complexity (ms) for n = 60000 is : 5457.318457
Sorted Array (Merge Sort):

Enter Max array size: 70000
Enter the array elements:
Time Complexity (ms) for n = 70000 is : 6630.648568
Sorted Array (Merge Sort):

Enter Max array size: 80000
Enter the array elements:
Time Complexity (ms) for n = 80000 is : 7419.150889
Sorted Array (Merge Sort):

Enter Max array size: 90000
Enter the array elements:
Time Complexity (ms) for n = 90000 is : 8119.913672
Sorted Array (Merge Sort):

Enter Max array size: 100000
Enter the array elements:
Time Complexity (ms) for n = 100000 is : 8865.6302
Sorted Array (Merge Sort):

Plot Graph: time taken versus n on graph sheet

Time Complexity Analysis:

Merge Sort Algorithm

Average performance O(n log n)

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

(a) Dynamic Programming method

```
import java.util.Scanner;

public class KnapsackDP {
	static final int MAX = 20; // max. no. of objects
	static int w[];   // weights 0 to n-1
	static int p[];    // profits 0 to n-1
	static int n;              // no. of objects
	static int M;              // capacity of Knapsack
	static int V[][];          // DP solution process - table
	static int Keep[][]; // to get objects in optimal solution

	public static void main(String args[]) {
		w = new int[MAX];
		p = new int[MAX];
		V = new int [MAX][MAX];
		Keep = new int[MAX][MAX];
		int optsoln;
		ReadObjects();
		for (int i = 0; i <= M; i++)
			V[0][i] = 0;
		for (int i = 0; i <= n; i++)
			V[i][0] = 0;
		optsoln = Knapsack();
		System.out.println("Optimal solution = " + optsoln);
	}

	static int Knapsack() {
		int r; // remaining Knapsack capacity
		for (int i = 1; i <= n; i++)
			for (int j = 0; j <= M; j++)
				if ((w[i] <= j) && (p[i] + V[i - 1][j - w[i]] > V[i - 1][j]))
				{
					V[i][j] = p[i] + V[i - 1][j - w[i]];
					Keep[i][j] = 1;
				} else {
					V[i][j] = V[i - 1][j];
					Keep[i][j] = 0;
				}

		// Find the objects included in the Knapsack
		r = M;
		System.out.println("Items = ");
		for (int i = n; i > 0; i--) // start from Keep[n,M]
```

```
                        if (Keep[i][r] == 1) {
                                System.out.println(i + " ");
                                r = r - w[i];
                        }
                System.out.println();
                return V[n][M];
        }

        static void ReadObjects() {
                Scanner scanner = new Scanner(System.in);
                System.out.println("Knapsack Problem - Dynamic Programming
Solution: ");
                System.out.println("Enter the max capacity of knapsack: ");
                M = scanner.nextInt();
                System.out.println("Enter number of objects:  ");
                n = scanner.nextInt();
                System.out.println("Enter Weights: ");
                for (int i = 1; i <= n; i++)
                        w[i] = scanner.nextInt();
                System.out.println("Enter Profits: ");
                for (int i = 1; i <= n; i++)
                        p[i] = scanner.nextInt();
                scanner.close();
        }
}
```

Output

```
Knapsack Problem - Dynamic Programming Solution:
Enter the max capacity of knapsack:
5
Enter number of objects:
4
Enter Weights:
1
2
2
1
Enter Profits:
15
20
10
12
Items =
4
2
1

Optimal solution = 47
```

(b) Greedy method.

```
import java.util.Scanner;
class KObject {                                    // Knapsack object details
        float w;
        float p;
        float r;
}
public class KnapsackGreedy {
        static final int MAX = 20;      // max. no. of objects
        static int n;                           // no. of objects
        static float M;                         // capacity of Knapsack

        public static void main(String args[]) {
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter number of objects:  ");
                n = scanner.nextInt();
                KObject[] obj = new KObject[n];
                for(int i = 0; i<n;i++)
                        obj[i] = new KObject();// allocate memory for members

                ReadObjects(obj);
                Knapsack(obj);
                scanner.close();
        }

        static void ReadObjects(KObject obj[]) {
                KObject temp = new KObject();
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter the max capacity of knapsack: ");
                M = scanner.nextFloat();

                System.out.println("Enter Weights: ");
                for (int i = 0; i < n; i++)
                        obj[i].w = scanner.nextFloat();

                System.out.println("Enter Profits: ");
                for (int i = 0; i < n; i++)
                        obj[i].p = scanner.nextFloat();

                for (int i = 0; i < n; i++)
                        obj[i].r = obj[i].p / obj[i].w;

                // sort objects in descending order, based on p/w ratio
                for(int i = 0; i<n-1; i++)
                        for(int j=0; j<n-1-i; j++)
                                if(obj[j].r < obj[j+1].r){
                                        temp = obj[j];
                                        obj[j] = obj[j+1];
```

```
                                        obj[j+1] = temp;
                                    }
                scanner.close();
        }

        static void Knapsack(KObject kobj[]) {
                float x[] = new float[MAX];
                float totalprofit;
                int i;
                float U;  // U place holder for M
                U = M;
                totalprofit = 0;
                for (i = 0; i < n; i++)
                        x[i] = 0;
                for (i = 0; i < n; i++) {
                        if (kobj[i].w > U)
                                break;
                        else {
                                x[i] = 1;
                                totalprofit = totalprofit + kobj[i].p;
                                U = U - kobj[i].w;
                        }
                }
                System.out.println("i = " + i);
                if (i < n)
                        x[i] = U / kobj[i].w;
                totalprofit = totalprofit + (x[i] * kobj[i].p);
                System.out.println("The Solution vector, x[]: ");
                for (i = 0; i < n; i++)
                        System.out.print(x[i] + " ");
                System.out.println("\nTotal profit is = " + totalprofit);
        }
}
```

Output

```
Enter number of objects:
4
Enter the max capacity of knapsack:
5
Enter Weights: 1 2 2 1
Enter Profits:
15
20
10
12
i = 3
The Solution vector, x[]:
1.0 1.0 1.0 0.5
```

Total profit is = 52.0

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

```
import java.util.*;

public class DijkstrasClass {

	final static int MAX = 20;
	final static int infinity = 9999;
	static int n;		// No. of vertices of G
	static int a[][]; // Cost matrix
	static Scanner scan = new Scanner(System.in);

	public static void main(String[] args) {
		ReadMatrix();
		int s = 0;		// starting vertex
		System.out.println("Enter starting vertex: ");
		s = scan.nextInt();
		Dijkstras(s);	// find shortest path
	}

	static void ReadMatrix() {
		a = new int[MAX][MAX];
		System.out.println("Enter the number of vertices:");
		n = scan.nextInt();
		System.out.println("Enter the cost adjacency matrix:");
		for (int i = 1; i <= n; i++)
			for (int j = 1; j <= n; j++)
				a[i][j] = scan.nextInt();
	}

	static void Dijkstras(int s) {
		int S[] = new int[MAX];
		int d[] = new int[MAX];
		int u, v;
		int i;
		for (i = 1; i <= n; i++) {
			S[i] = 0;
			d[i] = a[s][i];
		}
		S[s] = 1;
		d[s] = 1;
		i = 2;
		while (i <= n) {
			u = Extract_Min(S, d);
			S[u] = 1;
			i++;
			for (v = 1; v <= n; v++) {
```

```
                        if (((d[u] + a[u][v] < d[v]) && (S[v] == 0)))
                            d[v] = d[u] + a[u][v];
                    }
                }
                for (i = 1; i <= n; i++)
                    if (i != s)
                        System.out.println(i + ":" + d[i]);
            }

            static int Extract_Min(int S[], int d[]) {
                int i, j = 1, min;
                min = infinity;
                for (i = 1; i <= n; i++) {
                    if ((d[i] < min) && (S[i] == 0)) {
                        min = d[i];
                        j = i;
                    }
                }
                return (j);
            }
}
```

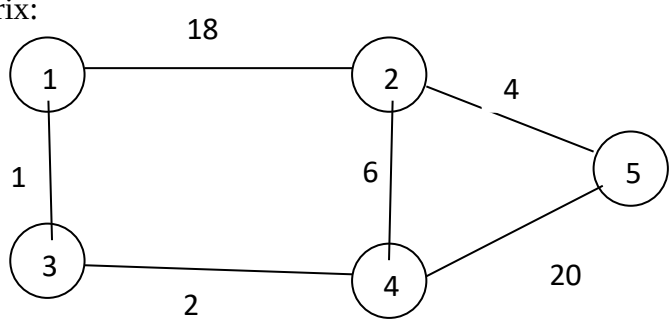
Output

Enter the number of vertices:

5

Enter the cost adjacency matrix:

0 18 1 9999 9999
18 0 9999 6 4
1 9999 0 2 9999
9999 6 2 0 20
9999 4 9999 20 0



Enter starting vertex:

1

2:9

3:1

4:3

5:13

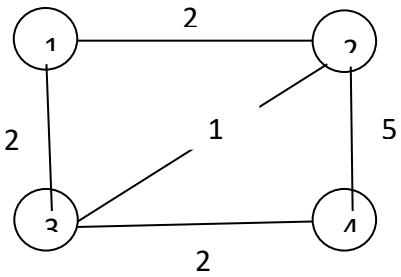
8. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal'salgorithm. Use Union-Find algorithms in your program.

```
import java.util.Scanner;

public class KruskalsClass {

        final static int MAX = 20;
        static int n; // No. of vertices of  G
        static int cost[][]; // Cost matrix
        static Scanner scan = new Scanner(System.in);
         static int parent [] = new int [10];
        public static void main(String[] args) {
                ReadMatrix();
                Kruskals();
        }

        static void ReadMatrix() {

                int i, j;
                cost = new int[MAX][MAX];

                System.out.println("Implementation of Kruskal's algorithm");
                System.out.println("Enter the no. of vertices");
                n = scan.nextInt();

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

        sstatic void Kruskals() {

                int a = 0, b = 0, u = 0, v = 0, i, j, ne = 1, min, mincost = 0;
```

```
        System.out.println("The edges of Minimum Cost Spanning Tree are");
        while (ne < n) {
            for (i = 1, min = 999; i <= n; i++) {
                for (j = 1; j <= n; j++) {
                    if (cost[i][j] < min) {
                        min = cost[i][j];
                        a = u = i;
                        b = v = j;
                    }
                }
            }
            u = find(u);
            v = find(v);
            if (u != v) {
                uni(a, v);
        System.out.println(ne++ + "edge (" + a + "," + b + ") =" + min);
                mincost += min;
            }
            cost[a][b] = cost[b][a] = 999;
        }
        System.out.println("Minimum cost :" + mincost);
}

static int find(int i) {

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

static void uni(int i, int j) {

        parent[j] = i;
}}
```

Output

Enter the number of vertices: 4
Enter the cost adjacency matrix:
0 20 2 999
20 0 15 5
2 15 0 25
999 5 25 0



The edges of Minimum Cost Spanning Tree are

1edge (1,3) =2

2edge (2,4) =5

3edge (2,3) =15

Minimum cost :22

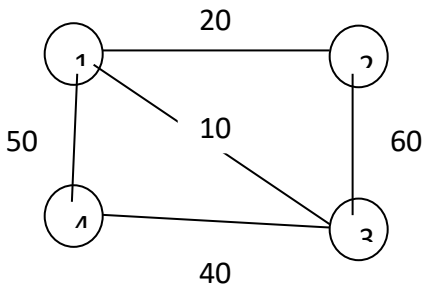
9. Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
import java.util.Scanner;

public class PrimsClass {

        final static int MAX = 20;
        static int n; // No. of vertices of G
        static int cost[][]; // Cost matrix
        static Scanner scan = new Scanner(System.in);

        public static void main(String[] args) {
                ReadMatrix();
                Prims();
        }

        static void ReadMatrix() {
                int i, j;
                cost = new int[MAX][MAX];

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

        static void Prims() {

                int visited[] = new int[10];
                int ne = 1, i, j, min, a = 0, b = 0, u = 0, v = 0;
                int mincost = 0;

                visited[1] = 1;
                while (ne < n) {
```

```
                for (i = 1, min = 999; i <= n; i++)
                        for (j = 1; j <= n; j++)
                                if (cost[i][j] < min)
                                        if (visited[i] != 0) {
                                                min = cost[i][j];
                                                a = u = i;
                                                b = v = j;
                                        }
                if (visited[u] == 0 || visited[v] == 0) {
        System.out.println("Edge" + ne++ + ":(" + a + "," + b + ")" + "cost:" + min);
                        mincost += min;
                        visited[b] = 1;
                }
                cost[a][b] = cost[b][a] = 999;
        }
        System.out.println("\n Minimun cost" + mincost);
}

}
```

Output

```
 Enter the number of nodes: 4
Enter the cost matrix:
 0        20       10       50
20        0        60       999
10        60       0        40
50        999      40       0

Enter Source: 1
1 --> 3 = 10    Sum = 10
1 --> 2 = 20    Sum = 30
3 --> 4 = 40    Sum = 70

Total cost: 70
```



10. Write Java programs to

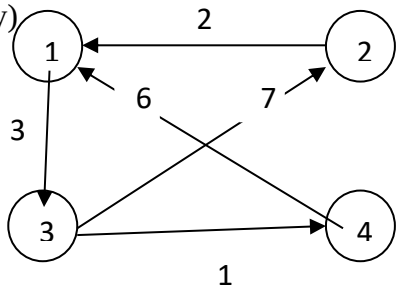
(a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

```
import java.util.Scanner;
public class FloydsClass {
static final int MAX = 20;      // max. size of cost matrix
static int a[][];               // cost matrix
static int n;                           // actual matrix size

public static void main(String args[]) {
        a = new int[MAX][MAX];
        ReadMatrix();
        Floyds();                               // find all pairs shortest path
        PrintMatrix();
}

static void ReadMatrix() {
        System.out.println("Enter the number of vertices\n");
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        System.out.println("Enter the Cost Matrix (999 for infinity) \n");
        for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= n; j++) {
                        a[i][j] = scanner.nextInt();
                }
        }
        scanner.close();
}

static void Floyds() {
        for (int k = 1; k <= n; k++) {
                for (int i = 1; i <= n; i++)
                        for (int j = 1; j <= n; j++)
                                if ((a[i][k] + a[k][j]) < a[i][j])
                                        a[i][j] = a[i][k] + a[k][j];
        }
}

static void PrintMatrix() {
        System.out.println("The All Pair Shortest Path Matrix is:\n");
        for(int i=1; i<=n; i++)
        {
                for(int j=1; j<=n; j++)
                        System.out.print(a[i][j] + "\t");
                System.out.println("\n");
        }
        }
}
```

Output

Enter the number of vertices: 4
EEnter the Cost Matrix (999 for infinity)

0	999	3	999
2	0	999	999
999	7	0	1
6	999	999	0



TThe All Pair Shortest Path Matrix is:

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

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

```
import java.util.Scanner;

public class TravSalesPerson {
static int MAX = 100;
static final int infinity = 999;

public static void main(String args[]) {
        int cost = infinity;
        int c[][] = new int[MAX][MAX];      // cost matrix
        int tour[] = new int[MAX];          // optimal tour
        int n;                                                  // max. cities
        System.out.println("Travelling Salesman Problem using Dynamic
Programming\n");
        System.out.println("Enter number of cities: ");
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        System.out.println("Enter Cost matrix:\n");
        for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++) {
                        c[i][j] = scanner.nextInt();
                        if (c[i][j] == 0)
                                c[i][j] = 999;
                }
        for (int i = 0; i < n; i++)
                tour[i] = i;
        cost = tspdp(c, tour, 0, n);
        // print tour cost and tour
        System.out.println("Minimum Tour Cost: " + cost);
        System.out.println("\nTour:");
        for (int i = 0; i < n; i++) {
                System.out.print(tour[i] + " -> ");
        }
        System.out.println(tour[0] + "\n");
        scanner.close();
}

static int tspdp(int c[][], int tour[], int start, int n) {
        int i, j, k;
        int temp[] = new int[MAX];
        int mintour[] = new int[MAX];
        int mincost;
        int cost;
        if (start == n - 2)
                return c[tour[n - 2]][tour[n - 1]] + c[tour[n - 1]][0];
        mincost = infinity;
        for (i = start + 1; i < n; i++) {
                for (j = 0; j < n; j++)
```

```
                    temp[j] = tour[j];
                temp[start + 1] = tour[i];
                temp[i] = tour[start + 1];
                if (c[tour[start]][tour[i]] + (cost = tspdp(c, temp, start + 1, n)) <
mincost) {
                    mincost = c[tour[start]][tour[i]] + cost;
                    for (k = 0; k < n; k++)
                        mintour[k] = temp[k];
                }
        }
        for (i = 0; i < n; i++)
                tour[i] = mintour[i];
        return mincost;
}
}
```

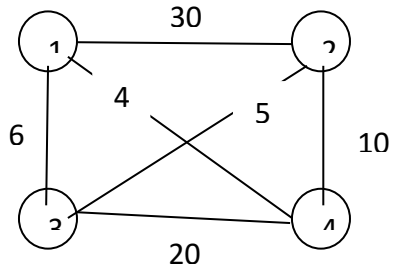
Output

Travelling Salesman Problem using Dynamic Programming

Enter number of cities:

Enter cost matrix:

0	30	6	4
30	0	5	10
6	5	0	20
4	10	20	0



Minimum Tour Cost: 25

Tour:

0 -> 2 -> 1 -> 3 -> 0

11. Design and implement in Java to find a subset of a given set S = {S_1, S_2,.....,S_n} of *n* positive integers whose SUM is equal to a given positive integer *d*. For example, if S ={1, 2, 5, 6, 8} and *d*= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

```
import java.util.Scanner;

public class SumOfsubset {
        final static int MAX = 10;
        static int n;
        static int S[];
        static int soln[];
        static int d;

        public static void main(String args[]) {
                S = new int[MAX];
                soln = new int[MAX];
                int sum = 0;
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter number of elements: ");
                n = scanner.nextInt();

                System.out.println("Enter the set in increasing order: ");
                for (int i = 1; i <= n; i++)
                        S[i] = scanner.nextInt();
                System.out.println("Enter the max. subset value(d): ");
                d = scanner.nextInt();
                for (int i = 1; i <= n; i++)
                        sum = sum + S[i];
                if (sum < d || S[1] > d)
                        System.out.println("No Subset possible");
                else
                        SumofSub(0, 0, sum);
                scanner.close();
        }

        static void SumofSub(int i, int weight, int total) {
                if (promising(i, weight, total) == true)
                        if (weight == d) {
                                for (int j = 1; j <= i; j++) {
```

```
                                        if (soln[j] == 1)
                                                System.out.print(S[j] + "  ");
                                }
                                System.out.println();
                        }
                        else {
                                soln[i + 1] = 1;
                                SumofSub(i + 1, weight + S[i + 1], total - S[i + 1]);
                                soln[i + 1] = 0;
                                SumofSub(i + 1, weight, total - S[i + 1]);
                        }
        }

        static boolean promising(int i, int weight, int total) {
        return ((weight + total >= d) && (weight == d || weight + S[i + 1] <= d));
        }
}
```

Output

```
Enter number of elements:
5
Enter the set in increasing order:
2
3
4
5
6
Enter the max. subset value(d): 9

2  3  4
3  6
4  5
```

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

```
package hamiltoniancycleexp;
import java.util.*;

public class HamiltonianCycleExp {
   public static void main(String[] args) {
      // TODO code application logic here
        HamiltonianCycle obj=new HamiltonianCycle();
 obj.getHCycle(1);
   }

}

class HamiltonianCycle
{
 private int adj[][],x[],n;
 public HamiltonianCycle()
 {
          Scanner src = new Scanner(System.in);
          System.out.println("Enter the number of nodes");
          n=src.nextInt();
          x=new int[n];
          x[0]=0;
          for (int i=1;i<n; i++)
                  x[i]=-1;
          adj=new int[n][n];

 System.out.println("Enter the adjacency matrix");
  for (int i=0;i<n; i++)
    for (int j=0; j<n; j++)
          adj[i][j]=src.nextInt();
 }

public void nextValue (int k)
{
int i=0;
 while(true)
{
```

```
        x[k]=x[k]+1;
        if (x[k]==n)
                x[k]=-1;
        if (x[k]==-1)
                return;
        if (adj[x[k-1]][x[k]]==1)
                for (i=0; i<k; i++)
                        if (x[i]==x[k])
                                break;
        if (i==k)
                if (k<n-1 || k==n-1 && adj[x[n-1]][0]==1)
                        return;
}
}
public void getHCycle(int k)
{
        while(true)
        {
                nextValue(k);
                if (x[k]==-1)
                        return;
                if (k==n-1)
                {
                        System.out.println("\nSolution : ");
                        for (int i=0; i<n; i++)
                                System.out.print((x[i]+1)+" ");
                        System.out.println(1);
                }
                else getHCycle(k+1);
        }
}
}
```

Output

```
Enter the number of nodes
4
Enter the adjacency matrix
0 1
1 1
1 0 1 1
1 1 0 1
1 1 1 0
```

Solution :
1 2 3 4 1

Solution :
1 2 4 3 1

Solution :
1 3 2 4 1

Solution :
1 3 4 2 1

Solution :
1 4 2 3 1

Solution :
1 4 3 2 1

Viva Questions:

1. What is an algorithm? What is the need to study Algorithms?
2. Explain Euclid's Algorithm to find GCD of two integers with an e.g.
3. Explain Consecutive Integer Checking algorithm to find GCD of two numbers with an e.g.
4. Middle School Algorithm with an e.g.
5. Explain the Algorithm design and analysis process with a neat diagram.
6. Define: a) Time Efficiency b) Space Efficiency.
7. What are the important types of problems that encounter in the area of computing?
8. What is a data structure? How are data structures classified?
9. Briefly explain linear and non-linear data structures.
10. What is a set? How does it differ from a list?
11. What are the different operations that can be performed on a set?
12. What are the different ways of defining a set?
13. How can sets be implemented in computer application?
14. What are different ways of measuring the running time of an algorithm?
15. What is Order of Growth?
16. Define Worst case, Average case and Best case efficiencies.
17. Explain the Linear Search algorithm.
18. Define O, Ω, Θ notations.
19. Give the general plan for analyzing the efficiency of non-recursive algorithms with an e.g.
20. Give an algorithm to find the smallest element in a list of n numbers and analyze the efficiency.
21. Give an algorithm to check whether all the elements in a list are unique or not and analyze the efficiency
22. Give an algorithm to multiply two matrices of order N*N and analyze the efficiency.
23. Give the general plan for analyzing the efficiency of Recursive algorithms with an e.g.
24. Give an algorithm to compute the Factorial of a positive integer n and analyze the efficiency.
25. Give an algorithm to solve the Tower of Hanoi puzzle and analyze the efficiency.
26. Define an explicit formula for the nth Fibonacci number.
27. Define a recursive algorithm to compute the nth Fibonacci number and analyze its efficiency.
28. What is Exhaustive Search?
29. What is Traveling Salesmen Problem (TSP)? Explain with e.g.
30. Give a Brute Force solution to the TSP. What is the efficiency of the algorithm?
31. What is an Assignment Problem? Explain with an e.g.
32. Give a Brute Force solution to the Assignment Problem. What is the efficiency of the algorithm?
33. Explain Divide and Conquer technique and give the general divide and conquer recurrence.

34. Define: a)Eventually non-decreasing function b)Smooth function c)Smoothness rule d)Masters theorem
35. Explain the Merge Sort algorithm with an e.g. and also draw the tree structure of the recursive calls made.
36. Analyze the efficiency of Merge sort algorithm.
37. Explain the Quick Sort algorithm with an example and also draw the tree structure of the recursive calls made.
38. Analyze the efficiency of Quick sort algorithm.
39. Give the Binary Search algorithm and analyze the efficiency.
40. Give an algorithm to find the height of a Binary tree and analyze the efficiency.
41. Give an algorithm each to traverse the Binary tree in Inorder, Preorder and Postorder.
42. Explain how do you multiply two large integers and analyze the efficiency of the algorithm. Give an e.g.
43. Explain the Stassen's Matrix multiplication with an e.g. and analyze the efficiency.
44. Explain the concept of Decrease and Conquer technique and explain its three major variations.
45. Give the Insertion Sort algorithm and analyze the efficiency.
46. Explain DFS and BFS with an e.g. and analyze the efficiency.
47. Give two solutions to sort the vertices of a directed graph topologically.
48. Discuss the different methods of generating Permutations.
49. Discuss the different methods of generating Subsets.
50. What is Heap? What are the different types of heaps?
51. Explain how do you construct heap?
52. Explain the concept of Dynamic programming with an e.g.
53. What is Transitive closure? Explain how do you find out the Transitive closure with an e.g.
54. Give the Warshall's algorithm and analyze the efficiency.
55. Explain how do you solve the All-Pairs-Shortest-Paths problem with an e.g.
56. Give the Floyd's algorithm and analyze the efficiency.
57. What is Knapsack problem? Give the solution to solve it using dynamic programming technique.
58. What are Memory functions? What are the advantages of using memory functions?
59. Give an algorithm to solve the knapsack problem.
60. Explain the concept of Greedy technique.
61. Explain Prim's algorithm with e.g.
62. Prove that Prim's algorithm always yields a minimum spanning tree.
63. Explain Kruskal's algorithm with an e.g.
64. Explain Dijkstra's algorithm with an e.g.
65. What are Huffman trees? Explain how to construct a Huffman trees with an e.g.
66. Explain the concept of Backtracking with an e.g.
67. What is state space tree? Explain how to construct a state space tree?
68. What is n-Queen's problem? Generate the state space tree for n=4.

69. Explain the subset sum problem with an e.g.
70. What are Decision Trees? Explain.
71. Define P, NP, and NP-Complete problems.
72. Explain the Branch and Bound technique with an e.g.
73. What are the steps involved in quick sort?
74. What is the principle used in the quick sort?
75. What are the advantages and disadvantages of quick sort?
76. What are the steps involved in merge sort?
77. What is divide, conquer, combine?
78. xplain the concept of topological ordering?
79. What are the other ordering techniques that you know?
80. How topological ordering is different from other ordering techniques?
81. What is transitive closure?
82. Time complexity of warshall's algorithm?
83. Define knapsack problem?
84. What is dynamic programming?
85. hat is single-source shortest path problem?
86. What is the time complexity of dijkstra's algorithm?
87. What is the purpose of kruskal's algorithm?
88. How is kruskal's algorithm different from prims?
89. What is BACK TRACKING?
90. What is branch and bound?
91. What is the main idea behind solving the TSP?
92. Do you know any other methodology for implementing a solution to this problem?
93. What does the term optimal solution of a given problem mean?
94. What is a spanning tree?
95. What is a minimum spanning tree?
96. Applications of spanning tree?