DAYANANDA SAGAR ACADEMY OF TECHNOLOGY & MANAGEMENT

Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082

Affiliated to VTU, Approved by AICTE, Accredited by NAAC with A+ Grade CSE, ISE, ECE, EEE, MECH, CIVIL Programs Accredited by NBA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

\

2022-2023

IV Semester CBCS

2021 scheme

DESIGN & ANALYSIS OF ALGORITHMS

Subject code: 21CS42

COMPILED BY:
Ms. Jahnavi S
(Asst. Prof, CSE, DSATM)
Ms. Shylaja B
(Asst. Prof, CSE, DSATM)
Mrs. Divya H N
(Asst. Prof, CSE, DSATM)

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

(Affiliated to Visvesvaraya Technological University, Belagavi & Approved By AICTE, New Delhi)
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore-560082

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision and Mission of the Institution Vision

"To strive at creating the institution a center of highest caliber of learning, so as to create an overall intellectual atmosphere with each deriving strength from the other to be the best of the engineers, scientists with management & design skills."

Mission

- To serve its region, state, the nation and globally by preparing students to make meaningful contributions in an increasing global society challenges.
- To encourage, reflection on and evaluation of emerging needs and priorities with the state of art infrastructure at institution.
- To support research and services establishing enhancements in technical, economic, human and cultural development.
- To establish inter disciplinary center of excellence, supporting/ promoting students implementation.
- To increase the number of Doctorate holders to promote research culture in the campus.
- To increase IPC, IPR, EDC, innovation cells with functional MOU's supporting students quality growth.

Vision and Mission of the CSE Department Vision

"Epitomize CSE graduate to crave a niche globally in the field of computer science to excel in the world of information technology and automation by imparting knowledge to sustain skills for the changing trends in the society and industry."

Mission

- To educate students to become excellent engineers in a confident and creative environment through world class pedagogy.
- Enhancing the knowledge in the changing technology trends by giving hands-on experience through continuous education and by making them to organize & participate in various events.
- Impart skills in the field of IT and its related areas with a focus on developing the required competencies and virtues to meet the industry expectations.
- Ensure quality research and innovations to fulfil industry, government and social needs.
- Impart entrepreneurship and consultancy skills to students to develop self-sustaining life skills in multi-disciplinary areas.

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

(Affiliated to Visvesvaraya Technological University, Belagavi & Approved By AICTE, New Delhi)
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore-560082

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program Outcomes	
a.	**Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems
b.	**Problem Analysis:** Identify, formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
c.	**Design/ Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
d.	**Conduct investigations of complex problems:** Using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
e.	**Modern Tool Usage**: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to Complex engineering activities with an under- standing of the limitations.
f.	**The Engineer and Society:** Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the Consequent responsibilities relevant to professional engineering practice
g.	**Environment and Sustainability:** Understand the impact of professional Engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development
h.	**Ethics:** Apply ethical principles and commit to professional ethics and Responsibilities and norms of engineering practice
i.	**Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.
j.	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
k.	**Life-long Learning**: Recognize the need for and have the preparation and ability to engage in independent and life- long learning in the broadest context of technological change
l.	Project Management and Finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environment
Program Specific Outcomes	
m.	**PSO1:** Adapt, Contribute Innovate ideas in the field of Artificial Intelligence and Machine Learning
n.	**PSO2:** Enrich the abilities to qualify for Employment, Higher studies and Research in various domains of Artificial Intelligence and Machine Learning such as Data Science, Computer Vision, Natural Language Processing with ethical values
o.	**PSO3:** Acquire practical proficiency with niche technologies and open-source platforms and become Entrepreneur in the domain of Artificial Intelligence and Machine Learning

DESIGN AND ANALYSIS OF ALGORITHMS			
Course Code	**21CS42**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 T + 20 P	Total Marks	100
Credits	04	Exam Hours	03

Course Learning Objectives:

CLO 1. Explain the methods of analysing the algorithms and to analyze performance of algorithms.
CLO 2. State algorithm's efficiencies using asymptotic notations.
CLO 3. Solve problems using algorithm design methods such as the brute force method, greedy method, divide and conquer, decrease and conquer, transform and conquer, dynamic programming, backtracking and branch and bound.
CLO 4. Choose the appropriate data structure and algorithm design method for a specified application.
CLO 5. Introduce P and NP classes.

Teaching-Learning Process (General Instructions)

These are sample Strategies, which teachers can use to accelerate the attainment of the various course outcomes.

1. Lecturer method (L) does not mean only traditional lecture method, but different type of teaching methods may be adopted to develop the outcomes.
2. Show Video/animation films to explain functioning of various concepts.
3. Encourage collaborative (Group Learning) Learning in the class.
4. Ask at least three HOT (Higher order Thinking) questions in the class, which promotes critical thinking.
5. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop thinking skills such as the ability to evaluate, generalize, and analyze information rather than simply recall it.
6. Topics will be introduced in a multiple representation.
7. Show the different ways to solve the same problem and encourage the students to come up with their own creative ways to solve them.
8. Discuss how every concept can be applied to the real world - and when that's possible, it helps improve the students' understanding.

Module-1

Introduction: What is an Algorithm? It's Properties. Algorithm Specification-using natural language, using Pseudo code convention, Fundamentals of Algorithmic Problem solving, Analysis Framework-Time efficiency and space efficiency, Worst-case, Best-case and Average case efficiency.

Performance Analysis: Estimating Space complexity and Time complexity of algorithms.

Asymptotic Notations: Big-Oh notation (O), Omega notation (Ω), Theta notation ( ) with examples, Basic efficiency classes, Mathematical analysis of Non-Recursive and Recursive Algorithms with Examples.

Brute force design technique: Selection sort, sequential search, string matching algorithm with complexity Analysis.

Textbook 1: Chapter 1 (Sections 1.1,1.2), Chapter 2(Sections 2.1,2.2,2.3,2.4), Chapter 3(Section 3.1,3.2)

Textbook 2: Chapter 1(section 1.1,1.2,1.3)

Laboratory Component:

1. Sort a given set of n integer elements using Selection 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case.

Teaching-Learning Process	1. Problem based Learning. 2. Chalk & board, Active Learning. 3. Laboratory Demonstration.

Module-2

Divide and Conquer: General method, Recurrence equation for divide and conquer, solving it using Master's theorem. , Divide and Conquer algorithms and complexity Analysis of Finding the maximum & minimum, Binary search, Merge sort, Quick sort.

Decrease and Conquer Approach: Introduction, Insertion sort, Graph searching algorithms, Topological Sorting. It's efficiency analysis.

Textbook 2: Chapter 3(Sections 3.1,3.3,3.4,3.5,3.6)

Textbook 1: Chapter 4 (Sections 4.1,4.2,4.3), Chapter 5(Section 5.1,5.2,5.3)

Laboratory Component:

1. 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.
2. 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

Teaching-Learning Process	1. Chalk & board, Active Learning, MOOC, Problem based Learning. 2. Laboratory Demonstration.

Module-3

Greedy Method: General method, Coin Change Problem, Knapsack Problem, solving Job sequencing with deadlines Problems.

Minimum cost spanning trees: Prim's Algorithm, Kruskal's Algorithm with performance analysis.

Single source shortest paths: Dijkstra's Algorithm.

Optimal Tree problem: Huffman Trees and Codes.

Transform and Conquer Approach: Introduction, Heaps and Heap Sort.

Textbook 2: Chapter 4(Sections 4.1,4.3,4.5)

Laboratory Component:

1. Sort a given set of n integer elements using Selection 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case.

Teaching-Learning Process	1. Problem based Learning. 2. Chalk & board, Active Learning. 3. Laboratory Demonstration.

Module-2

Divide and Conquer: General method, Recurrence equation for divide and conquer, solving it using Master's theorem. , Divide and Conquer algorithms and complexity Analysis of Finding the maximum & minimum, Binary search, Merge sort, Quick sort.

Decrease and Conquer Approach: Introduction, Insertion sort, Graph searching algorithms, Topological Sorting. It's efficiency analysis.

Textbook 2: Chapter 3(Sections 3.1,3.3,3.4,3.5,3.6)

Textbook 1: Chapter 4 (Sections 4.1,4.2,4.3), Chapter 5(Section 5.1,5.2,5.3)

Laboratory Component:

1. 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

2. 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

Teaching-Learning Process	1. Chalk & board, Active Learning, MOOC, Problem based Learning. 2. Laboratory Demonstration.

Module-3

Greedy Method: General method, Coin Change Problem, Knapsack Problem, solving Job sequencing with deadlines Problems.

Minimum cost spanning trees: Prim's Algorithm, Kruskal's Algorithm with performance analysis.

Single source shortest paths: Dijkstra's Algorithm.

Optimal Tree problem: Huffman Trees and Codes.

Transform and Conquer Approach: Introduction, Heaps and Heap Sort.

Textbook 2: Chapter 4(Sections 4.1,4.3,4.5)

Textbook 1: Chapter 9(Section 9.1,9.2,9.3,9.4), Chapter 6(section 6.4)

Laboratory Component:

Write & Execute C++/Java Program

1. To solve Knapsack problem using Greedy method.
2. To find shortest paths to other vertices from a given vertex in a weighted connected graph, using Dijkstra's algorithm.
3. To find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.
4. To find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

Teaching-Learning Process	1. Chalk & board, Active Learning, MOOC, Problem based Learning. 2. Laboratory Demonstration.

Module-4

Dynamic Programming: General method with Examples, Multistage Graphs.

Transitive Closure: Warshall's Algorithm. **All Pairs Shortest Paths**: Floyd's Algorithm,

Knapsack problem, Bellman-Ford Algorithm, Travelling Sales Person problem.

Space-Time Tradeoffs: Introduction, Sorting by Counting, Input Enhancement in String Matching-Harspool's algorithm.

Textbook 2: Chapter 5 (Sections 5.1,5.2,5.4,5.9)

Textbook 1: Chapter 8(Sections 8.2,8.4), Chapter 7 (Sections 7.1,7.2)

Laboratory Component:

Write C++/ Java programs to

1. Solve All-Pairs Shortest Paths problem using Floyd's algorithm.
2. Solve Travelling Sales Person problem using Dynamic programming.
3. Solve 0/1 Knapsack problem using Dynamic Programming method.

Teaching-Learning Process	1. Chalk & board, Active Learning, MOOC, Problem based Learning. 2. Laboratory Demonstration.

Module-5

Backtracking: General method, solution using back tracking to N-Queens problem, Sum of subsets problem, Graph coloring, Hamiltonian cycles Problems.

Branch and Bound: Assignment Problem, Travelling Sales Person problem, 0/1 Knapsack problem

NP-Complete and NP-Hard problems: Basic concepts, non- deterministic algorithms, P, NP, NP-Complete, and NP-Hard classes.

Textbook 1: Chapter 12 (Sections 12.1,12.2) Chapter 11(11.3)

Textbook 2: Chapter 7 (Sections 7.1,7.2,7.3,7.4,7.5) Chapter 11 (Section 11.1)

Laboratory Component:

1. Design and implement C++/Java Program 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.
2. Design and implement C++/Java Program to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.

Teaching-Learning Process	1. Chalk & board, Active Learning, MOOC, Problem based learning. 2. Laboratory Demonstration.

Course outcome (Course Skill Set)

At the end of the course the student will be able to:

CO 1. Analyze the performance of the algorithms, state the efficiency using asymptotic notations and analyze mathematically the complexity of the algorithm.
CO 2. Apply divide and conquer approaches and decrease and conquer approaches in solving the problems analyze the same
CO 3. Apply the appropriate algorithmic design technique like greedy method, transform and conquer approaches and compare the efficiency of algorithms to solve the given problem.
CO 4. Apply and analyze dynamic programming approaches to solve some problems. and improve an algorithm time efficiency by sacrificing space.
CO 5. Apply and analyze backtracking, branch and bound methods and to describe P, NP and NP-Complete problems.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures not less than 35% (18 Marks out of 50) in the semester-end examination (SEE), and a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation:

Three Unit Tests each of **20 Marks (duration 01 hour**)

1. First test at the end of 5th week of the semester
2. Second test at the end of the 10th week of the semester
3. Third test at the end of the 15th week of the

semester Two assignments each of **10 Marks**

4. First assignment at the end of 4th week of the semester
5. Second assignment at the end of 9th week of the semester

Practical Sessions need to be assessed by appropriate rubrics and viva-voce method. This will contribute to **20 marks**.

- Rubrics for each Experiment taken average for all Lab components – 15 Marks.
- Viva-Voce– 5 Marks (more emphasized on demonstration topics)

The sum of three tests, two assignments, and practical sessions will be out of 100 marks and will be **scaled down to 50 marks**
(to have a less stressed CIE, the portion of the syllabus should not be common /repeated for any of the methods of the CIE. Each method of CIE should have a different syllabus portion of the course).

CIE methods /question paper has to be designed to attain the different levels of Bloom's taxonomy as per the outcome defined for the course.

Semester End Examination:

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the subject (**duration 03 hours)**

1. The question paper will have ten questions. Each question is set for 20 marks. Marks scored shall be proportionally reduced to 50 marks
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.

The students have to answer 5 full questions, selecting one full question from each module

Suggested Learning Resources:

Textbooks

1. Introduction to the Design and Analysis of Algorithms, Anany Levitin: 2nd Edition, 2009. Pearson.
2. Computer Algorithms/C++, Ellis Horowitz, SatrajSahni and Rajasekaran, 2nd Edition, 2014, Universities Press.

Reference Books

1. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronal L. Rivest, Clifford Stein, 3rd Edition, PHI.
2. Design and Analysis of Algorithms, S. Sridhar, Oxford (Higher Education)

Weblinks and Video Lectures (e-Resources):

1. http://elearning.vtu.ac.in/econtent/courses/video/CSE/06CS43.html
2. https://nptel.ac.in/courses/106/101/106101060/
3. http://elearning.vtu.ac.in/econtent/courses/video/FEP/ADA.html
4. http://cse01-iiith.vlabs.ac.in/
5. http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms

Activity Based Learning (Suggested Activities in Class)/ Practical Based learning

1. Real world problem solving and puzzles using group discussion. E.g., Fake coin identification, Peasant, wolf, goat, cabbage puzzle, Konigsberg bridge puzzle etc.,
2. Demonstration of solution to a problem through programming.

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.

Program No.	Topics	Course Outcomes
1.	Sort a given set of n integer elements using Selection 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case.	CO5
2.	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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.	CO5
3.	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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.	CO5
4.	Write & Execute C++/Java Program To solve Knapsack problem using Greedy method.	CO5
5.	Write & Execute C++/Java Program To find shortest paths to other vertices from a given vertex in a weighted connected graph, using Dijkstra's algorithm.	CO5
6.	Write & Execute C++/Java Program To find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.	CO5
7.	Write & Execute C++/Java Program To find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.	CO5
8.	Write C++/ Java programs to Solve All-Pairs Shortest Paths problem using Floyd's algorithm.	CO5
9.	Write C++/ Java programs to Solve Travelling Sales Person problem using Dynamic programming.	CO5
10.	Write C++/ Java programs to Solve 0/1 Knapsack problem using Dynamic Programming method.	CO5

11.	Design and implement C++/Java Program 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.	CO5
12.	Design and implement C++/Java Program to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.	CO5

1. Sort a given set of n integer elements using Selection 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. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case.

```
package sorting;

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

public class SelectionSortComplexity
{
   static final int MAX = 200000;
   static int[] a = new int[MAX];
   public static void SelectionSortAlgorithm(int n)
   {
      for (int i = 0; i < n - 1; i++) {
         int minIndex = i;
         for (int j = i + 1; j < n; j++) {
            if (a[j] < a[minIndex]) {
               minIndex = j;
            }
         }
         if (minIndex != i) {
            int temp = a[i];
            a[i] = a[minIndex];
            a[minIndex] = temp;
         }
      }
   }



   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();
      }

      System.out.println("Input Array:");
      for (int i = 0; i < n; i++)
      {
```

```
            System.out.print(a[i] + " ");
        }

        long startTime = System.nanoTime();
        SelectionSortAlgorithm(n);
        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);
    }
}
```

```
run:
Enter Max array size:
5
Enter the array elements:
10
20
30
40
50
Input Array:
10 20 30 40 50
Sorted Array:
10 20 30 40 50
Time Complexity in ms for n=5 is: 0.0048


Enter Max array size: 5
Enter the array elements:
500
6000
800
920
502
Input Array:
500 6000 800 920 502
Sorted Array:
500 502 800 920 6000
Time Complexity in ms for n=5 is: 0.0056

run:
Enter Max array size: 10
Enter the array elements:
```

```
8999
444545454
45454
787
865
45454
89989
154875
98951
56568
Input Array:
8999 444545454 45454 787 865 45454 89989 154875 98951 56568
Sorted Array:
787 865 8999 45454 45454 56568 89989 98951 154875 444545454
Time Complexity in ms for n=10 is: 0.0093
```

2. 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)

3. 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)

4. Write & Execute C++/Java Program
To solve Knapsack problem using 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
```

5. To find shortest paths to other vertices from a given vertex in a weighted connected graph, using Dijkstra's algorithm

```
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);
        }
}
```

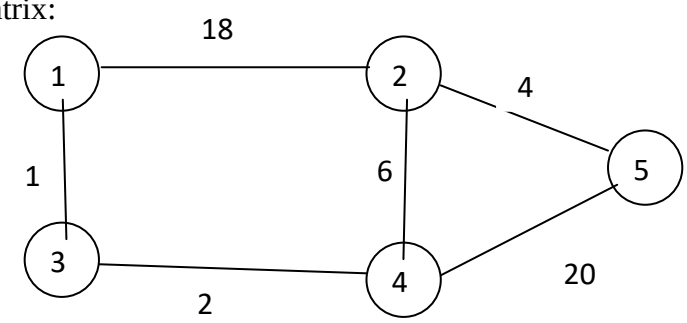
Output

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

6. Write & Execute C++/Java Program

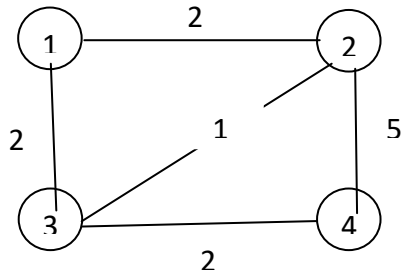
a) To find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.

```
import java.util.Scanner;

public class KruskalsClass {

public static void main(String[] args)
        final int MAX = 20;
        int n; // No. of vertices of G
        int cost[][]; // Cost matrix
        int a = 0, b = 0, u = 0, v = 0, ne = 1, min, mincost = 0;
```

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

        cost = new int[MAX][MAX];

        System.out.println("Enter the no. 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++) {
                        cost[i][j] = scan.nextInt();
                        if (cost[i][j] == 0)
                                cost[i][j] = 999;


System.out.println("The edges of Minimum Cost Spanning Tree are");
        while (ne < n) {
            min= 999;
                for (int i =1; i <= n; i++) {
                        for (int j = 1; j <= n; j++) {
                                if (cost[i][j] < min) {
                                        min = cost[i][j];
                                        a = u = i;
                                        b = v = j;
                                }
                        }
                }
                while (parent[u] != 0)
                u = parent[u];

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

                if (u != v) {
                 ne++;
        System.out.println("edge found " + a + "-->" + b + " =" + min);
                        mincost += min;
                  parent[v]=u;

                }
                cost[a][b] = cost[b][a] = 999;
        }
        System.out.println("Minimum cost :" + mincost);
}

}}
```

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



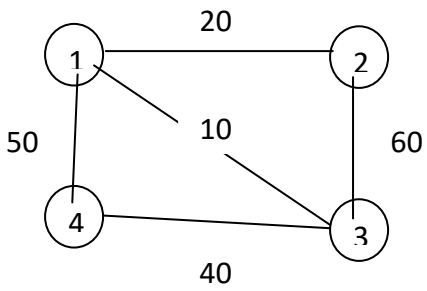
7. To find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
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
```



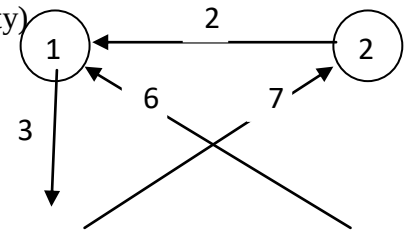
8. Solve All-Pairs Shortest Paths problem using Floyd's algorithm.

```
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
```

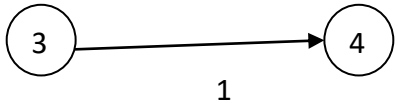
```
        public static void main(String args[]) {
                a = new int[MAX][MAX];
                ReadMatrix();
                Floyds();                               // find all pairs shortest path
                PrintMatrix();
        }

        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

9. Solve 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;
}
}
```

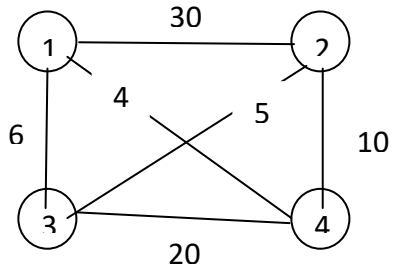
Output

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

10. Solve 0/1 Knapsack problem using 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
```

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

```
import java.util.Scanner;

public class 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?