<table>
<tr><td colspan="6">Devi Ahilya Vishwavidhyalaya, Indore, India
Institute of Engineering & Technology</td><td>II Year B.E. (Information Technology)</td></tr>
<tr><td>Course Code & Name</td><td colspan="5">Instructions Hours per Semester and Credits</td></tr>
<tr><td rowspan="2">4RIPC2

ANALYSIS AND DESIGN OF ALGORITHMS</td><td colspan="2">Classroom Instruction (CI)</td><td>Lab Instruction (LI)</td><td>Term Work (TW) and Self Learning (SL)</td><td>Total no. of Hours Per semester</td><td>Total Credits (Total Hours/30)</td></tr>
<tr><td>L</td><td>T</td><td>P</td><td>TW+SL</td><td rowspan="2">120</td><td rowspan="2">4</td></tr>
<tr><td>20</td><td>10</td><td>20</td><td>70</td></tr>
</table>

Course Learning Objectives:

• To develop a strong understanding of algorithmic thinking, mathematical foundations, and performance analysis required for designing efficient algorithms in computer science and information technology.

• To enable students to analyse, design, and implement algorithms using different strategies such as divide and conquer, greedy, dynamic programming, and backtracking for solving computational problems efficiently.

• To build the ability to evaluate algorithms using time and space complexity, asymptotic analysis, and recurrence relations for performance comparison.

• To prepare students for advanced study and practical applications involving sorting, searching, graph algorithms, string processing, matrix algorithms, and complexity theory.

• To enhance problem-solving skills using systematic algorithm design techniques applicable to real-world IT and software development problems.

Prerequisites:

Basic knowledge of mathematics including algebra, logarithms, functions, and matrices; fundamentals of programming in any high-level language; and basic understanding of data structures such as arrays, stacks, queues, and linked lists.

COURSE CONTENTS

Unit-I

Foundations of Algorithm Analysis: Introduction to Algorithms and their role in problem solving. Fundamental issues in algorithm analysis. Types of algorithms and performance issues. Time and space complexity. Asymptotic notations: Big-O, Big-Ω, Big-Θ. Mathematical preliminaries: functions, logarithms, exponentials, factorials and their growth rates. Comparison of growth rates. Recurrence relations and their formulation. Methods for solving recurrences: Substitution method, Recurrence Tree method, and Master Theorem.

Unit-II

Sorting and Searching Techniques

Elementary sorting techniques and their analysis: Selection Sort, Insertion Sort. Divide and Conquer approach. Advanced sorting techniques and their analysis: Merge Sort, Heap Sort, Priority Queue operations, Quick Sort, Radix Sort, and Bucket Sort.
Searching techniques: Linear Search and Binary Search. Algorithms for finding minimum and maximum elements. Performance comparison of searching and sorting algorithms.

Unit-III

Greedy, Dynamic Programming and Backtracking

Greedy Method: Concept and strategy, Knapsack Problem, Minimum Cost Spanning Tree, Optimal Storage on Tapes, Single Source Shortest Path. Dynamic Programming: Concept and strategy, All-Pairs Shortest Path, 0/1 Knapsack Problem, Traveling Salesperson Problem.
Basic Traversal and Search Techniques. Backtracking: Concept, 8-Queens Problem.
Branch and Bound Techniques: Basic idea and applications.

Unit-IV

Matrix and String Algorithms

Matrix Multiplication Problem. Strassen's Matrix Multiplication Algorithm. Matrix Chain Multiplication Problem. String Matching Problems and Applications. Algorithms for String Matching: Naive String Matching Algorithm, Rabin-Karp Algorithm, String Matching with Finite Automata, Knuth-Morris-Pratt (KMP) Algorithm.

Unit-V

Non-Deterministic Algorithms and Complexity Classes

Introduction to Non-Deterministic Algorithms. Nondeterministic Complexity. Computational Complexity Classes: P, NP, NP-Complete, NP-Hard. Reducibility. Decision and Optimization Problems. Examples of NP and NP-Hard Problems: Hamiltonian Cycle, Traveling Salesperson Problem (TSP), Satisfiability Problem, Clique Problem.

Course Outcomes:

CO.No.	CO
CO1	Apply fundamental concepts of algorithm design and analysis to evaluate time and space complexity using asymptotic notations and mathematical preliminaries.
CO2	Analyze and compare different types of algorithms, including recursive and divide-and-conquer algorithms, using recurrence relations and standard solving methods.
CO3	Implement and analyze sorting and searching algorithms to select efficient techniques for solving real-world computational problems.
CO4	Apply greedy, dynamic programming, backtracking, and branch-and-bound strategies to solve optimization and decision-making problems.

CO5	Evaluate advanced algorithms related to matrices, strings, and computational complexity classes (P, NP, NP-Complete, NP-Hard) for solving practical and theoretical problems in computer science..

Books Recommended:

[1] T.H. coreman, C.E. Leiserson and R. L. Rivest, Inroduction to Algorithms, Prentice Hall of India, 1990.
[2] E. Horowitz, S. Sahni, S Rajasekaran, Computer Algorithm, Galgotia Publications.
[3] A. V. Aho, J. E. Hopcroft & J. D. Ullman, The design and Analysis of Computer Algorithms, Addision Wesley 1974
[4] Knuth, D, The art of computer programming , Vol. 1-2-3, A ddision Wesley, 2/e, 1988.

CO-PO-PSO Relationship

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
4RIPC2.CO1	3	3	2	2	3			**1**	**1**	**1**	**1**	3	2	1
4RIPC2.CO2	3	3	2	2	3			**1**	**1**	**1**	**1**	2	3	1
4RIPC2.CO3	3	3	2	2	3			**1**	**1**	**1**	**1**	3	2	1
4RIPC2.CO4	3	3	2	2	3			**1**	**1**	**1**	**1**	2	3	1
4RIPC2.CO5	3	3	2	2	3			**1**	**1**	**1**	**1**	3	2	1

List of Practical Assignments :

During the learning of course, students need to do assignments:

Assignment 1: Analysis of Algorithm Efficiency and Asymptotic Notations
Problem 1: Study of algorithm efficiency.
Write algorithms for simple problems (sum of n numbers, finding maximum).
Analyze time and space complexity.
Problem 2: Asymptotic notations.
Demonstrate Big-O, Big-Ω, and Big-Θ for given algorithms.
Compare growth rates of linear, quadratic, and logarithmic functions.

Assignment 2: Recurrence Relations and Their Solutions
Problem 1: Formulation of recurrence relations.
Formulate recurrence relations for recursive algorithms.
Problem 2: Solving recurrences.
Solve recurrences using Substitution Method.
Solve recurrences using Recurrence Tree Method.
Solve recurrences using Master Theorem.

Assignment 3: Implementation and Analysis of Elementary Sorting Algorithms
Problem 1: Selection Sort.
Implement Selection Sort.
Analyze best, average, and worst-case complexity.

Problem 2: Insertion Sort.
Implement Insertion Sort.
Perform complexity analysis and comparison.

Assignment 4: Divide and Conquer Sorting Techniques
Problem 1: Merge Sort.
Implement Merge Sort using Divide and Conquer approach.
Analyze time and space complexity.
Problem 2: Quick Sort.
Implement Quick Sort.
Analyze best, average, and worst-case performance.

Assignment 5: Advanced Sorting Techniques
Problem 1: Heap Sort.
Implement Heap Sort using binary heap.
Perform Priority Queue operations.
Problem 2: Non-comparison based sorting.
Implement Radix Sort.
Implement Bucket Sort and analyze constraints.

Assignment 6: Searching Techniques and Performance Comparison
Problem 1: Linear Search.
Implement Linear Search.
Analyze performance for different input sizes.
Problem 2: Binary Search.
Implement Binary Search (iterative and recursive).
Compare Linear and Binary Search performance.

Assignment 7: Greedy Algorithms
Problem 1: Knapsack Problem.
Implement Fractional Knapsack using Greedy method.
Problem 2: Graph-based greedy algorithms.
Implement Minimum Cost Spanning Tree (Prim's or Kruskal's).
Implement Single Source Shortest Path (Dijkstra's Algorithm).

Assignment 8: Dynamic Programming Techniques
Problem 1: 0/1 Knapsack Problem.
Implement 0/1 Knapsack using Dynamic Programming.
Problem 2: Shortest path and optimization problems.
Implement All-Pairs Shortest Path (Floyd–Warshall).
Solve Traveling Salesperson Problem using DP approach.

Assignment 9: Backtracking and Branch & Bound Techniques
Problem 1: Backtracking.
Implement 8-Queens Problem using Backtracking.
Problem 2: Branch and Bound.
Solve Knapsack or TSP using Branch and Bound technique.

Assignment 10: String and Matrix Algorithms
Problem 1: Matrix algorithms.
Implement Matrix Multiplication.

Implement Strassen's Matrix Multiplication Algorithm.
Solve Matrix Chain Multiplication Problem.
Problem 2: String matching algorithms.
Implement Naive String Matching Algorithm.
Implement Rabin-Karp Algorithm.
Implement Knuth-Morris-Pratt (KMP) Algorithm.