**Course title: Algorithms**
**Code: CS402**
**Type: PCC/DSC**
**Credits: 4**
**Semester: 1**

| **Course** | Algorithms | **Credits** | 4 |
|---|---|---|---|
| **Course Type** | **PCC/DSC** | | |

**Course Description**

This course provides a comprehensive study of algorithmic design and analysis techniques for solving computational problems efficiently. It introduces mathematical foundations of algorithm analysis and explores major design paradigms including Divide and Conquer, Greedy Method, Dynamic Programming, Backtracking, and Branch and Bound. The course emphasizes correctness proofs, complexity analysis, and the selection of appropriate data structures for algorithm development. It also introduces computational complexity theory and NP-completeness, enabling students to understand the inherent difficulty of problems and the limitations of efficient computation. Through a variety of classical and real-world problems, students develop the ability to design, analyze, and evaluate algorithms for diverse applications.

**Course Objectives**

After completing this course, students will be able to:

1. Understand the mathematical foundations and principles of algorithm analysis, including asymptotic notations, recurrence relations, and performance evaluation.
2. Apply fundamental algorithm design paradigms such as Divide and Conquer, Greedy Strategy, Dynamic Programming, Backtracking, and Branch and Bound to solve computational problems efficiently.
3. Design and analyze algorithms using appropriate data structures, and evaluate their correctness, time complexity, and space complexity.
4. Develop problem-solving skills for graph, combinatorial, decision and optimization problems by selecting suitable algorithmic strategies.
5. Understand the concepts of computational complexity and NP-completeness, and recognize the tractability and limitations of algorithmic solutions.
6. Cultivate the ability to compare alternative algorithmic approaches and choose efficient solutions for practical and large-scale computing applications.

**Course Learning Outcomes**

CLO-1: Assess the inherent structure/hardness of a problem (Evaluate)

CLO-2: Select an appropriate strategy to solve a problem (Understand)

CLO-3: Design an algorithm that suits the time complexity requirements of the problem. (Create)

CLO-4: Estimate the time and space complexities of an algorithm along with the mathematical proofs when necessary. (Evaluate)

CLO-5: Devise algorithms by choosing appropriate data structures (Create)

| Mapped to Program Level Outcomes | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PLO 1 | PLO 2 | PLO 3 | PLO 4 | PLO 5 | PLO 6 | PLO 7 | PLO 8 | PLO 9 | PLO 10 | PLO 11 | PLO 12 |
| CLO1 | 2 | 3 | | | | 1 | | | | | | |
| CLO2 | | 1 | 2 | | 3 | | | | | | | |
| CLO3 | 2 | | 3 | 1 | | | | | | | | |
| CLO4 | 1 | | 2 | 3 | | | | | | | | |
| CLO5 | 2 | | 1 | | | 3 | | | | | | |
| CLO6 | 2 | 3 | | | | 1 | | | | | | |

**Syllabus**

**UNIT I: Foundations of Algorithm Analysis**
Introduction to algorithms and problem-solving methodologies. Definition of algorithms, Pseudocode vs Algorithm, performance analysis, time and space complexity, best-case, average-case, and worst-case analysis. Asymptotic notations including Big-O, Big-Ω, Big-Θ, little-o, and little-ω. Mathematical tools for complexity analysis. Recurrence relations and their solutions using substitution method, iteration method, recursion tree method, generating functions, and Master's Theorem. Introduction to heap data structures, binary heaps, heap operations, and priority queues with applications. Complexity analysis of sorting algorithms including Insertion Sort, Bubble Sort, Selection Sort, Heap Sort, Counting Sort, and Radix Sort. Comparative study of sorting techniques and introduction to algorithm design paradigms and general strategies for problem solving.

**UNIT II: Divide and Conquer Strategy**
Fundamentals of the Divide and Conquer paradigm, design methodology, recurrence formulation, and complexity analysis. Binary Search, Finding Maximum and Minimum, Merge Sort, Quick Sort, Randomized Quick Sort, and their correctness proofs and complexity analysis, Median and Order Statistics. Applications of Divide and Conquer to problems such as Closest Pair of Points, Convex Hull, Strassen's Matrix Multiplication, etc.

**UNIT 3: Greedy Strategy**

Theoretical foundations of greedy algorithms including greedy-choice property, optimal substructure, and proof techniques. Introduction to Matroids and greedy algorithms on matroid structures. Design, correctness proofs, and complexity analysis of greedy algorithms for Fractional Knapsack Problem, Job Sequencing with Deadlines, Activity Selection Problem, Coin Change Problem (for canonical coin systems), Huffman Coding, Minimum Spanning Tree algorithms (Kruskal's and Prim's), Single Source Shortest Path using Dijkstra's Algorithm.

**UNIT 4: Dynamic Programming Strategy**

Principle of optimality, overlapping subproblems, optimal substructure, memoization, tabulation, and dynamic programming design methodology. Comparison of Dynamic Programming with Greedy and Divide and Conquer approaches, identifying situations where those strategies fail. Applications including Fibonacci Numbers, Coin Change Problem, 0/1 Knapsack Problem, Unbounded Knapsack (0/n Knapsack), Subset Sum Problem, Rod Cutting Problem, Matrix Chain Multiplication, Longest Common Subsequence (LCS), Optimal Binary Search Tree (OBST), Bellman–Ford Algorithm, Floyd–Warshall Algorithm, Traveling Salesperson Problem (TSP). Correctness proofs and complexity analysis of dynamic programming algorithms.

**UNIT 5: Backtracking and Branch & Bound Strategies**

State-space representation, state-space tree construction, traversal techniques, pruning methods, and bounding functions. Backtracking algorithms for N-Queens Problem, Sum of Subsets Problem, Graph Coloring, etc. Branch and Bound techniques including FIFO, LIFO, and Least-Cost Branch and Bound. Applications to 0/1 Knapsack Problem, Traveling Salesperson Problem, Depth First Search and its applications including Topological Sorting, Strongly Connected Components, Cycle Detection, Articulation Points. Complexity analysis and comparison with exhaustive search methods.

**UNIT 6: Theory of NP-Completeness**

Introduction to computational complexity and tractability. Decision problems, optimization problems, and verification algorithms. Complexity classes P, NP, co-NP, NP-Hard, and NP-Complete. Polynomial-time reductions, reducibility concepts, and Cook–Levin Theorem. Techniques for proving NP-Completeness and construction of reductions. Study of classical NP-Complete problems including SAT, CNF-SAT, 3-SAT, Vertex Cover, Independent Set, Clique, Graph Coloring. NP-Completeness proofs for selected problems and discussion of approximation algorithms and heuristic methods for computationally hard problems.

**Reference Books**


1. Introduction to Algorithms-T.Cormen, C.E.Leiserson, R.L.Rivest, PHI, 3rdEdition 2009.
2. Algorithms- R.Johnsonbaugh and M.Schaefer, Pearson, 2004.
3. Fundamentals of Algorithmics - G.Brassard and P.Bratley, PH, 1996
4. The Algorithm Design Manual- Steven S. Skiena, Springer, 2009