<table>
<tr><td>Course Title</td><td colspan="3"><b>Design & Analysis of Algorithms</b></td><td colspan="4">Course Type</td><td colspan="2"><b>Comprehensive</b></td></tr>
<tr><td>Course Code</td><td colspan="3"></td><td colspan="4">Class</td><td colspan="2">B.Tech</td></tr>
<tr><td rowspan="6">Instruction delivery</td><td>Activity</td><td>Credits</td><td>Credit Hours</td><td colspan="4" rowspan="2">Total Number of Classes per Semester</td><td colspan="2" rowspan="2">Assessment in Weightage</td></tr>
<tr><td>Lecture</td><td>3</td><td>3</td></tr>
<tr><td>Tutorial</td><td>0</td><td>0</td><td rowspan="3">Theory</td><td rowspan="3">Tutorial</td><td rowspan="3">Practical</td><td rowspan="3">Self-study</td><td rowspan="3">CIE</td><td rowspan="3">SEE</td></tr>
<tr><td>Practical</td><td>1</td><td>1</td></tr>
<tr><td>Self-study</td><td>1</td><td>1</td></tr>
<tr><td>Total</td><td>5</td><td>5</td><td>37</td><td>0</td><td>12</td><td>15</td><td>50%</td><td>50%</td></tr>
<tr><td rowspan="3">Names Course Instructors</td><td>Course Lead</td><td colspan="8">Dr. Medhavi Malik</td></tr>
<tr><td colspan="3">Theory</td><td colspan="2"></td><td colspan="4">Practice</td></tr>
<tr><td colspan="3">1. Parvesh
2. Dr. Medhavi Malik
3. Dr. Arvind Dagur
4. Dr. Gambhir Singh
5. Sanjay Sonker
6. Ravi Sharma
7. Dr. D. Rajesh
8. Dr. SPS Chauhan
9. Shailendra Pratap Singh
10. Dr. Santosh Kumar
11. Swapnita Srivastava
12. Aditi Gaur
13. Dr. Anupam Sharma
14. Pragya</td><td colspan="2"></td><td colspan="4">1. Parvesh
2. Dr. Medhavi Malik
3. Dr. Arvind Dagur
4. Dr. Gambhir Singh
5. Sanjay Sonker
6. Ravi Sharma
7. Dr. D. Rajesh
8. Dr. SPS Chauhan
9. Shailendra Pratap Singh
10. Dr. Santosh Kumar
11. Swapnita Srivastava
12. Aditi Gaur
13. Dr. Anupam Sharma
14. Pragya</td></tr>
</table>

	15. Pooja 16. Mili Dhar 17. Vartika Mishra 18. Ambika Gupta 19. Nitin Jain 20. Harshit Jain 21. K. Rajkannan		15. Pooja 16. Mili Dhar 17. Vartika Mishra 18. Ambika Gupta 19. Nitin Jain 20. Harshit Jain 21. K. Rajkannan

Course Overview:

An Algorithm is a sequence of steps to solve a problem. Design and Analysis of Algorithm is very important for designing algorithm to solve different types of problems in the branch of computer science and information technology. This tutorial introduces the fundamental concepts of Designing Strategies, Complexity analysis of Algorithms, followed by problems on Graph Theory and Sorting methods. This tutorial also includes the basic concepts on Complexity theory.

Course Objective:

- To analyze and design different searching and sorting algorithm algorithms based upon different designing approaches.
- To analyze and design different tree algorithms based upon different designing approaches.
- To choose appropriate algorithm design techniques for solving Greedy Algorithm real time problems.
- To choose appropriate algorithm design techniques for solving Dynamic Programming real time problems.
- To apply and synthesize efficient algorithms**.**

Course Outcome (COs)- After the completion of the course, the student will be able to

CO1	Design new algorithms, prove them correct, and analyze their asymptotic and absolute runtime and memory demands.
CO2	Find an algorithm to solve the problem (create) and prove that the algorithm solves the problem correctly (validate).
CO3	Understand basic techniques for designing algorithms, including the techniques of recursion, divide-and-conquer, and greedy.
CO4	Apply classical sorting, searching, optimization and graph algorithms.
CO5	Understand basic techniques for designing algorithms, including the techniques of recursion, divide-and-conquer, and greedy.

BLOOM’S LEVEL OF THE COURSE OUTCOMES: Bloom's taxonomy is a set of hierarchical models used for the classification of educational learning objectives into levels of complexity and specificity. The learning domains are cognitive, affective, and psychomotor**.**

	Remember (L1)	**Understand (L2)**	**Apply (L3)**	**Analyze (L4)**	**Evaluate (L5)**	**Create (L6)**
CO1				✓		✓
CO2					✓	✓
CO3		✓			✓	
CO4		✓		✓		
CO5		✓	✓			

Program Outcomes:

PO1 Computing Science knowledge: Apply the knowledge of mathematics, statistics, computing science and information science fundamentals to the solution of complex computer application problems.

PO2 Problem analysis: Identify, formulate, review research literature, and analyze complex computing science problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and computer sciences**.**

PO3 Design/development of solutions: Design solutions for complex computing problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern computing science and IT tools including prediction and modeling to complex computing activities with an understanding of the limitations.

PO6 IT specialist and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional computing science and information science practice.

PO7 Environment and sustainability: Understand the impact of the professional computing science solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the computing science practice.

PO9 Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 Communication: Communicate effectively on complex engineering activities with the IT analyst 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.

PO11 Project management and finance: Demonstrate knowledge and understanding of the computing science and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 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.

Program Specific Outcomes(PSO’s)

PSO1: Have the ability to work with emerging technologies in computing requisite to Industry 4.0.

PSO2: Demonstrate Engineering Practice learned through industry internship and research project to solve live problems in various domains.

COURSE ARTICULATION MATRIX: The Course articulation matrix indicates the correlation between Course Outcomes and Program Outcomes and their expected strength of mapping in three levels (low, medium, and high).

COs#/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3	3	3										2	
CO2	3	2	2		2								2	
CO3	3	2	2										2	
CO4	3	2	2										2	
CO5	3	3	3	3	3			3	3				2	

COURSE ASSESSMENT: The course assessment patterns are the assessment tools used both in formative and summative examinations.

SNo	Assessment Tools	CIE							Total CIE marks	SEE
		QUIZ 1/AAT	CAT1	QUIZ 2/AAT	CAT2	LAB	LAB EXAM	Course-based Project		
1.	Comprehensive		A1		A2	A3		A4		
		0	30	0	30	20	0	20	100	100

Course Content (Theory)

Session	Topics	Skills to be Learned
1.	Introduction, Fundamentals of algorithm(Line count, operation count)	Design new Algorithm prove them correct, and analyze their asymptotic Complexities.
2.	Algorithm Design Techniques (Approaches, Design Paradigms)	
3.	Designing an algorithm and its Analysis(Best ,Worst & Average case)	
4.	Asymptotic Notations(O, Ω, Θ) based on Orders of Growth	
5.	Mathematical Analysis - Induction	
6.	Recurrence Relation - Substitution method	
7.	Recurrence Relation - Recursion method	
8.	Recurrence Relation - Master's Theorem	
9.	Introduction, Binary Search	Analyse Complexity of Searching and Sorting Algorithms
10.	Merge sort and its algorithm analysis	
11.	Quick sort and its algorithm analysis	
12.	Strassen's Matrix multiplication	
13.	Finding Maximum and minimum	
14.	Algorithm for finding closest pair	
15.	Convex Hull Problem	
16.	Red-Black trees	Analysis of complexity of Trees and Heaps that helps in storage of data in Memory, Dynamic Approch for Searching and Storing
17.	B – trees	
18.	Binomial Heaps	
19.	Fibonacci Heaps	
20.	M-way search tree	
21.	Dynamic Programming - 0/1 Knapsack Problem	
22.	Dynamic Programming- Travelling Salesman Problem	
23.	Dynamic Programming- Multistage Graph- Forward path	
	and backward path	
24.	N Queen's Problem	Understand the utilization of resource allocation.
25.	Sum Of Subsets	
26.	Graph Coloring	
27.	Hamiltonian's Circuit	
28.	Travelling Salesman Problem	
29.	String Matching	
30.	Branch and bound - 0/1 Knapsack	Create the algorithms so that

31.	Branch and Bound - Travelling Sales man Problem	either running time or time complexity; or memory used will be reduced.
32.	Randomized algorithm- Hiring Problem	
33.	Randomized algorithm- Matrix Chain Multiplication	
34.	Randomized Quick Sort	
35.	Introduction to PN problems	
36.	Introduction to NP problems	
37.	NP Complete	

Course Content (Practical)

1. Write a program to sort given set of numbers in ascending/descending order using Bubble sort and also search a number using binary search.
2. Write a program to sort given set of numbers in ascending/descending order using Insertion sort and also search a number using linear search.
3. Write a program to sort given set of numbers in ascending/descending order using Quick sort and any other sorting algorithm. Also record the time taken by these two programs and compare them.
4. Write a program to sort given set of numbers using Heap sort.
5. Write a program to sort given set of numbers Merge Sort.
6. Write a program to sort given set of numbers Counting Sort.
7. Write a program to implement Matrix Chain Multiplication.
8. Write a program to implement Knapsack using Greedy technique.
9. Write a program to implement Knapsack using Dynamic programming.
10. Write a program to implement Dijkstra’s Algorithm.
11. Write a program to implement Bellman-Ford Algorithm.
12. Write a program to implement n-Queen Problem using backtracking.
13. Write a program to implement Naive string matching algorithm.
14. Write a program to implement String Matching using Rabin-Karp algorithm.
15. Obtain the Topological ordering of vertices in a given digraph.
16. Write a program to implement Minimum Cost spanning tree.
17. Write a program to implement Sum of subset problem.
18. Write a program to implement All Pairs Shortest Paths using Floyd’s Algorithm.
19. Write a program to implement String Matching using Knuth Morris Pratt Algorithm.
20. Write a program to implement Greedy algorithm using Task Scheduling Problem.
21. Write a program to implement Greedy algorithm using Acitivity Selection Problem.
22. Compute the transitive closure of a given directed graph using Warshall's algorithm. Write a program to implement shortest path algorithm.
23. Write a program to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.
24. Write a program to implement solve LCS problem.
25. Write a program to implement Huffman-code.
26. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.
27. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

28. Write programs to (a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.
(b) Implement Travelling Sales Person problem using Dynamic programming.

29. Design and implement 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.

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

BIBLIOGRAPHY

Text Book:

1. Thomas H. Coreman, Charles E. Leiserson and Ronald L. Rivest, “Introduction to Algorithms”, The MIT Press, 3rd edition, 2009.

Reference Book

1. Michael T. Goodrich and Roberto Tamassia: Algorithm Design: Foundations, Analysis and Internet examples (John Wiley &Sons, Inc., 2002).
2. Ellis Horowitz, SartajSahni, SanguthevarRajasekaran. Fundamentals of Computer Algorithms, MIT Press, Second Edition (Indian reprint: Prentice-Hall), 2008.
3. Aho, Hopcraft, Ullman, “The Design and Analysis of Computer Algorithms” Pearson Education.

STUDENT-CENTERED LEARNING (SELF-LEARNING TOWARDS LIFE-LONG-LEARNING)

A) COURSE-BASED PROJECT (Psychomotor skills)

1. Comparing Kruskal and Prim’s algorithm in MST.
2. Greedy and Backtracking Algorithm Comparison in Graph Coloring Problem.
3. Capsa Susun using Greedy Algorithm.
4. Solving Travelling Salesman Problem Using Greedy Algorithm and Brute Force Algorithm.
5. Design and Analysis 0/1 Knapsack Problem.
6. Coin Change Problem using Brute Forceand Greedy Algorithm.
7. Comparing Exhaustive Search Algorithm and Greedy Algorithm in job Scheduling Problem.
8. Build a Cash Flow Minimiser.
9. Build a CB Mario game using Dynamic Programming Optimisation.
10. Build an application of Sudoku game using Backtracking method.
11. Build a Snakes & Ladders game, challenge the player to win in minimum number of moves. You can use BFS to compute it.

12. Make an application like Google Maps - You can use Dijkshtra’s algorithm to find the shortest paths, A* Search for more efficient & real time use.

13. Binary Search Algorithm to Search for a Value with a Certain Precision

14. Space Efficient Algorithm for Subset Sum

15. Place eight queens on an 8 × 8 chessboard so that no queen attacks another queen.

16. Build a game like SPACE- SHOOTER.

17. Make a PHONEBOOK using TRIE Data structure.

18. Use a string compression algorithm, like run lenghth encoding or Huffman coding.

19. Build. Game like Flappy Bird.

20. Build a snake and Ladders game, challenge the player to win in minimum number of moves. You can use BFS to compute it.

B) SELF-LEARNING THROUGH MOOCs

- CodeChef
- Coding Ninjas
- https://leetcode.com