Lesson Plan No. 1	Course Name: Design & Analysis of Algorithms Topic: Introduction to algorithms	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the fundamental concepts of algorithms. b. Learn how to represent algorithms using different methods (e.g., pseudocode, flowcharts). c. Differentiate between different types of algorithms (e.g., search, sort, graph algorithms). d. Gain a basic understanding of algorithm analysis and its importance. e. Develop problem-solving and analytical skills.
Teaching Aids (if any)	a. Power point presentation
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - to define what they think an "algorithm" is in their own words. - Briefly discuss real-life examples of algorithms (e.g., recipes, driving directions, search engine algorithms). - Emphasize the importance of efficient algorithms in today's world (e.g., in software development, data science, artificial intelligence). - **Development** (30 minutes) **-What is an Algorithm?** - Define an algorithm formally: A step-by-step procedure for solving a specific problem. - Discuss key characteristics of an algorithm: - Finiteness: Must terminate after a finite number of steps. - Definiteness: Each step must be precisely defined and unambiguous. - Input: May accept zero or more inputs. - Output: Produces at least one output. - Effectiveness: Each step must be executable in a finite amount of time. **-Representing Algorithms:** - Introduce different ways to represent algorithms: - **Pseudocode:** High-level description using a combination of natural language and programming constructs. - **Flowcharts:** Diagrams using symbols to represent different operations and the flow of control. - Present a simple example algorithm (e.g., finding the maximum number in an array) and demonstrate how to represent it using both pseudocode and a flowchart. **-Types of Algorithms:** - Briefly introduce common categories of algorithms:

	- **Search algorithms:** (e.g., linear search, binary search) - **Sorting algorithms:** (e.g., bubble sort, insertion sort, merge sort) - **Graph algorithms:** (e.g., depth-first search, breadth-first search) - **Dynamic programming algorithms** - **Greedy algorithms** - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What are the five essential characteristics of an algorithm? - What are two common ways to represent an algorithm? - Give an example of a real-world application of a search algorithm. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture -NPTEL 3. Reference website: **Introduction to Algorithms (CLRS):** https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/ [1] **GeeksforGeeks - Algorithms:** https://www.geeksforgeeks.org/fundamentals-of-algorithms/ 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - Define the divide-and-conquer strategy and provide an example algorithm that uses it. - What is the difference between Depth-First Search (DFS) and Breadth-First Search (BFS)? - Explain the Greedy algorithm approach and give an example where this approach is used. - Explain the concept of dynamic programming and provide an example where it is more efficient than a naive recursive solution. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to

	answer and discuss. 2. Quiz on Types of Algorithms. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2	Course Name: Design & Analysis of Algorithms Topic: Performance Analysis	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the importance of analyzing algorithm performance. b. Learn key concepts like time complexity and space complexity. c. Be introduced to asymptotic notation (Big O, Big Omega, Big Theta). d. Analyze the performance of simple algorithms (e.g., linear search, binary search). e. Develop critical thinking and analytical skills in evaluating algorithm efficiency.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you have two different ways to travel from your home to school. One route is always faster, regardless of traffic. - Explain that algorithms also have different "speeds" or efficiencies. - Introduce the concept of algorithm analysis as a way to predict and compare the performance of different algorithms. **Development** (30 minutes) - Time Complexity: - Define time complexity as a measure of how the running time of an algorithm grows as the input size increases. - Emphasize that we're interested in the growth rate, not the exact execution time. - Introduce Big O notation (asymptotic upper bound) to describe the growth rate of an algorithm. - Provide examples: - O(1) - Constant time (e.g., accessing an element in an array by index) - O(log n) - Logarithmic time (e.g., binary search) - O(n) - Linear time (e.g., linear search) - O(n log n) - Log-linear time (e.g., merge sort) - O(n^2) - Quadratic time (e.g., bubble sort) - O(2^n) - Exponential time - Space Complexity: - Briefly introduce space complexity as a measure of the amount of memory used by an algorithm. - Discuss how space complexity can also impact the efficiency of an algorithm. - Example: Analyzing Linear Search

	- Present the linear search algorithm and analyze its time complexity. - Explain why its time complexity is O(n) in the worst case. - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What does the time complexity of an algorithm represent? - What is the difference between O(n) and O(n^2) time complexity? - Why is analyzing algorithm performance important for software development? Question and Answer session in between the lesson to check the presence of mind of students.

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these.
	
	2. Suggested NPTEL video lecture
	NPTEL
	3. Reference website:
	- **Introduction to Algorithms (CLRS):**
	https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/
	- **GeeksforGeeks - Algorithm Analysis:**
	
	https://www.geeksforgeeks.org/analysis-of-algorithms/
	4. Suggested Reading books:
	- Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein .
	- Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani
	
	5. Homework
	
	- Define best case, worst case, and average case in the context of algorithm analysis.
	- For the following algorithm, determine its best, worst, and average case time complexities:
	def find(x, arr):
	for i in range(len(arr)):
	if arr[i] == x:
	return i
	return -1
	
	Spend 5 minutes to wrap up and consolidate the learnings

Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on analysis of Algorithms. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 3	Course Name: Design & Analysis of Algorithms Topic: Recursive Procedures	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of recursion and how it works. b. Learn to identify problems that can be effectively solved using recursion. c. Be able to write simple recursive functions. d. Understand the importance of base cases in recursive functions. e. Recognize potential issues with recursion, such as stack overflow.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine a set of Russian nesting dolls. Each doll contains a smaller version of itself within. - Explain that recursion works similarly: a problem is solved by breaking it down into smaller, similar subproblems. - calculating the factorial of a number and how they might approach it. - **Development** (30 minutes) - What is Recursion? - Define recursion: A technique where a function calls itself within its own definition. - Explain the key components of a recursive function: - Base case(s): Conditions that stop the recursion and provide a direct solution. - Recursive step: The part of the function that calls itself with a modified input, moving towards the base case. **- Example: Factorial** -Present the factorial function (n!) as a classic example of recursion. - Write the factorial function in pseudocode or a simple programming language. - Step through the execution of the factorial function for a small input value (e.g., 5!) to illustrate how it works. **- Example: Fibonacci Sequence** - Introduce the Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, ...) and its recursive definition. - Write the Fibonacci function in pseudocode or a simple programming language. - Discuss the potential inefficiencies of the recursive Fibonacci implementation.

	- **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What are the two essential components of a recursive function? - What is a base case in recursion, and why is it important? - Give an example of a problem that can be solved effectively using recursion. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL 3. Reference website: - **Introduction to Algorithms (CLRS):** - https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/ 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - How does tail recursion differ from regular recursion? - What is the time complexity of factorial_tail_recursive(n)? - Would it be more efficient to convert this function into an iterative version? Justify your answer. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on recursive procedure. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 4	Course Name: Design & Analysis of Algorithms Topic: Recurrence Relations, Induction Proof, and Proving Correctness	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of recurrence relations and their significance in algorithm analysis. b. Learn to derive recurrence relations for simple recursive algorithms. c. Master three key methods for solving recurrence relations: i. Substitution Method ii. Iteration Method iii. Master Theorem d. Understand the importance of proving the correctness of algorithms. e. Learn how to use mathematical induction to prove the correctness of recursive algorithms.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - revisiting the concept of recurrence relations: equations that express the nth term of a sequence in terms of previous terms. - Briefly reiterate their importance in analyzing the performance of recursive algorithms. - Introduce the concept of algorithm correctness: ensuring that an algorithm produces the desired output for all valid inputs. - Emphasize that proving the correctness of algorithms is crucial in software development. **Development** (30 minutes) - **Recurrence Relations:** - **Deriving Recurrence Relations:** - Demonstrate how to derive recurrence relations for simple recursive algorithms (e.g., factorial, Fibonacci, merge sort). - **Solving Recurrence Relations:** - Review key methods for solving recurrence relations: - Substitution Method - Iteration Method - Master Theorem (briefly) - **Proving Algorithm Correctness:** - **Mathematical Induction:** - Introduce the principle of mathematical induction as a powerful proof technique. - Explain how to use mathematical induction to prove the correctness of recursive algorithms.

	- Example: Prove the correctness of the recursive factorial function using induction. - **Connecting Recurrence Relations and Correctness:** - Discuss how solving recurrence relations helps in analyzing the time complexity of an algorithm, which is an essential aspect of its correctness (efficiency). - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the principle of mathematical induction? - How can you use mathematical induction to prove the correctness of a recursive algorithm? - Derive the recurrence relation for a simple recursive algorithm (e.g., binary search) and briefly discuss how you would approach proving its correctness. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture 3. **NPTEL :: Mathematics - NOC:Discrete Mathematics** 4. Reference website: **Introduction to Algorithms (CLRS):** https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/ Recurrence Relations | A Complete Guide - GeeksforGeeks 5. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 6. Homework - **Solve the recurrence using the master theorem**: T(n)=2T(n/2) +, with T(1)=1 - **Solve the recurrence using the master theorem**: T(n)=3T(n/3)+n with T(1)=1 Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on recurrence relation. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 5	Course Name: Design & Analysis of Algorithms Topic: Description of Randomized Algorithms	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the fundamental concepts of randomized algorithms. b. Learn to distinguish between deterministic and randomized algorithms. c. Describe the key characteristics and components of randomized algorithms. d. Analyze simple examples of randomized algorithms.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - limitations of deterministic algorithms in certain situations. - Example: Searching for a needle in a haystack - a deterministic approach might be inefficient. - Introduce the idea of introducing randomness into the problem-solving process. - Explain that randomized algorithms make use of random choices during their execution. **Development** (30 minutes) - **Deterministic vs. Randomized Algorithms:** - Define deterministic algorithms: Algorithms that always produce the same output for a given input. - Contrast deterministic algorithms with randomized algorithms, which incorporate random choices. - **Key Characteristics of Randomized Algorithms:** - Randomness: Emphasize the use of random numbers or random choices within the algorithm. - Probability: Discuss how probability plays a crucial role in analyzing the performance of randomized algorithms. - Efficiency: Explain how randomization can sometimes lead to more efficient solutions compared to deterministic approaches. - **Examples of Randomized Algorithms:** - **Randomized Quick Sort:** - Briefly describe the basic idea of Quick Sort and its potential for worst-case performance. - Explain how choosing the pivot element randomly can improve the average-case performance. - **Monte Carlo Algorithm for Primality Testing:** - Briefly explain the concept of primality testing. - Introduce the idea of a Monte Carlo algorithm that can quickly determine with high probability whether a number is prime or

	composite. - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is a deterministic algorithm? - How do randomized algorithms differ from deterministic algorithms? -What is the significance of random choices in the context of randomized algorithms? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL : Computer Science and Engineering - NOC:Randomized Algorithms** **Introduction to Algorithms (CLRS):** **https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/** Randomized Algorithms 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 4. Homework - Define the difference between a Monte Carlo algorithm and a Las Vegas algorithm. - Consider the problem of approximating the maximum independent set in a graph using a randomized algorithm. Describe a simple randomized approach to find a 2-approximation for the maximum independent set. What is the expected approximation ratio of your algorithm? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on randomized algorithm. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 6	Course Name: Design & Analysis of Algorithms Topic: Basic of Probability Theory	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the fundamental concepts of probability. b. Learn to define sample spaces and events. c. Calculate probabilities of simple events. d. Explore basic probability axioms and rules.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Begin with a simple thought experiment: flipping a coin. - to predict the outcome of the coin flip. - Discuss the uncertainty involved and the concept of chance. - Introduce probability as a measure of the likelihood of an event occurring. **Development** (30 minutes) - **Sample Space and Events:** - Define a sample space (S) as the set of all possible outcomes of an experiment. - Example: Flipping a coin: S = {Heads, Tails} - Define an event (E) as any subset of the sample space. - Example: Getting Heads on a coin flip: E = {Heads} - **Probability Axioms:** - Axiom 1: $0 \leq P(E) \leq 1$ for any event E (probability of an event is between 0 and 1). - Axiom 2: P(S) = 1 (probability of the entire sample space is 1). - Axiom 3: If events A and B are mutually exclusive (cannot occur simultaneously), then $P(A \cup B) = P(A) + P(B)$ (addition rule for mutually exclusive events). - **Calculating Probabilities:** - Equally Likely Outcomes: - If all outcomes in the sample space are equally likely, then P(E) = |E| / |S|, where |E| is the number of outcomes in event E and |S| is the number of outcomes in the sample space. - Examples: - Rolling a fair die (calculating the probability of rolling a specific number). - Drawing a card from a standard deck of cards (calculating the probability of drawing a heart). - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students

	- What is a sample space in the context of probability? - State the addition rule for mutually exclusive events. - If you roll a fair six-sided die, what is the probability of rolling an even number? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL : Mathematics - NOC:Probability and Statistics** **Probability Theory 2019 Admn..pdf** 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 4. Homework - **Basic Probability Calculation**: A die is rolled once. What is the probability of getting a number greater than 4? - **Complementary Events**: In a deck of 52 playing cards, what is the probability of drawing a card that is either a spade or a red card? (Note: Red cards are hearts and diamonds). - **Conditional Probability**: A bag contains 5 red balls and 3 green balls. A ball is drawn at random, and without replacement, a second ball is drawn. What is the probability that the second ball is red, given that the first ball drawn was green? - **Joint Probability**: A jar contains 3 blue, 4 red, and 5 green marbles. Two marbles are drawn sequentially without replacement. What is the probability that both marbles are of different colors? - **Independent Events**: Two fair coins are flipped. What is the probability of getting exactly one head and one tail? Are the events "getting a head on the first coin" and "getting a tail on the second coin" independent? - **Advanced Probability Questions:** - **Bayes' Theorem**: In a factory, 95% of the products made by machine A pass the quality control test, while 90% of the products made by machine B pass the test. 70% of the products are made by machine A, and 30% are made by machine B.

	A product passes the test. What is the probability that it was made by machine A? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on probability theory. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 8	Course Name: Design & Analysis of Algorithms Topic: Advantages and Disadvantages of Randomized Algorithms	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the key advantages of using randomized algorithms. b. Identify potential disadvantages and limitations of randomized algorithms. c. Analyze real-world scenarios where randomized algorithms are beneficial. d. Discuss the trade-offs between deterministic and randomized approaches.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - reviewing the concept of randomized algorithms: algorithms that incorporate random choices during their execution. - Briefly discuss examples of randomized algorithms like Randomized Quick Sort. - Why would we choose to use a randomized algorithm instead of a deterministic one? **Development** (30 minutes) - **Advantages of Randomized Algorithms:** - Simplicity: Randomized algorithms can often be simpler to design and implement than their deterministic counterparts. - Efficiency: - Randomized algorithms can often achieve better average-case performance than deterministic algorithms, especially in situations with high variance in input. - Examples: Randomized Quick Sort typically has O(n log n) average-case time complexity. - Dealing with Adversarial Inputs: - Randomized algorithms can be more robust against adversarial inputs designed to make deterministic algorithms perform poorly. - Online Algorithms: Randomized algorithms are often well-suited for online algorithms, where input arrives sequentially and decisions must be made without complete knowledge of the future. - **Disadvantages of Randomized Algorithms:** - Lack of Determinism: - The output of a randomized algorithm may vary on different runs, even with the same input. - This can be undesirable in some applications where

	predictability is crucial. - Potential for Unlucky Runs: - There's always a chance of encountering an "unlucky" sequence of random choices that leads to poor performance. - Debugging Challenges: - Debugging and analyzing the behavior of randomized algorithms can be more challenging than deterministic ones due to their inherent randomness. - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is one key advantage of using randomized algorithms in terms of their design and implementation? - How can randomization help improve the performance of an algorithm in some cases? - What is a potential disadvantage of using a randomized algorithm in an application where consistency is critical? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL : Computer Science and Engineering - NOC:Randomized Algorithms** **Introduction to Algorithms (CLRS):** https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/ 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 4. Homework - Explain why randomized algorithms are often simpler to implement compared to deterministic algorithms. Provide an example where a randomized algorithm offers a simpler solution than a deterministic one. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on randomized algorithm. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 9	Course Name: Design & Analysis of Algorithms Topic: Asymptotic Bounds	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. define and explain the concepts of Big O, Big Omega, and Big Theta notations. b. identify the growth rates of different functions using asymptotic notations. c. compare the efficiency of algorithms based on their asymptotic bounds.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you're traveling between two cities. You have options like driving, flying, or walking. Each has a different "cost" (time, money, energy). - Introduce the idea that in computer science, we're concerned with how the "cost" (time or space) of an algorithm grows as the input size increases. - Briefly mention that asymptotic notation helps us analyze and compare the efficiency of algorithms in a concise and general way. - **Development** (30 minutes) - **Big O Notation (O):** - Define Big O as an upper bound on the growth rate of a function. - Explain that f(n) = O(g(n)) means that f(n) grows no faster than g(n) for sufficiently large values of n. - Provide examples: - Constant time: O(1) (e.g., accessing an element in an array by index) - Linear time: O(n) (e.g., searching for an element in an unsorted array) - Quadratic time: O(n^2) (e.g., nested loops) - Logarithmic time: O(log n) (e.g., binary search) - **Big Omega Notation (Ω):** - Define Big Omega as a lower bound on the growth rate of a function. - Explain that f(n) = Ω(g(n)) means that f(n) grows at least as fast as g(n) for sufficiently large values of n. - **Big Theta Notation (Θ):** - Define Big Theta as both an upper and lower bound on the growth rate of a function. - Explain that f(n) = Θ(g(n)) means that f(n) grows at the same rate as g(n) for sufficiently large values of n. - Emphasize that Θ represents the tightest bound. - **Exercise** (10 minutes) –

	- Group Discussion at the end of the session. - Ask students - **True/False:** If f(n) = O(g(n)), then f(n) = Ω(g(n)). - **Multiple Choice:** Which of the following is the most efficient growth rate? - a) O(n^2) - b) O(log n) - c) O(2^n) - Give an example of an algorithm that has a time complexity of O(n). Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture https://onlinecourses.nptel.ac.in/noc25_cs23/unit?unit=16&lesson=17 **Wikipedia:**https://en.wikipedia.org/wiki/Asymptotic_analysis **GeeksforGeeks:** https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/ 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 4. Homework - a. Show that f(n)=3n2+5n+2f(n) = 3n^2 + 5n + f(n)=3n2+5n+2 is O(n2)O(n^2)O(n2). b. Prove that g(n)=7nlogn+5ng(n) = 7n \log n + 5ng(n)=7nlogn+5n is O(nlogn)O(n \log n)O(nlogn). c. Find the smallest ccc such that 3n2+5n+23n^2 + 5n + 23n2+5n+2 is O(n2)O(n^2)O(n2), and explain the reasoning. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Asymptotic_analysis. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 10	Course Name: Design & Analysis of Algorithms Topic: Concept of Efficiency of an Algorithm, Well Known Asymptotic Functions & Notations	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. define and explain the concept of algorithm efficiency. b. identify and describe different orders of growth (e.g., constant, linear, logarithmic, quadratic, exponential). c. understand and use Big O, Big Omega, and Big Theta notations to represent the time complexity of algorithms. d. compare the efficiency of different algorithms for the same problem..
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - consider two different ways to sort a list of numbers (e.g., bubble sort and merge sort). - Discuss with the class how they might determine which sorting algorithm is "better." - Introduce the concept of algorithm efficiency as a way to measure and compare the performance of different algorithms. **Development** (30 minutes) - **Time Complexity:** - Define time complexity as a measure of the time taken by an algorithm to execute as a function of the input size. - Explain that time complexity is typically expressed using asymptotic notations. - **Asymptotic Notations:** - **Big O Notation (O):** - Define Big O as an upper bound on the growth rate of a function. - Provide examples of functions with different Big O complexities (e.g., O(1), O(log n), O(n), O(n log n), O(n^2), O(2^n)). - **Big Omega Notation (Ω):** - Define Big Omega as a lower bound on the growth rate of a function. - **Big Theta Notation (Θ):** - Define Big Theta as both an upper and lower bound on the growth rate of a function, indicating that the function grows at the same rate as the given bound. - **Well-Known Asymptotic Functions:** - Discuss common functions encountered in algorithm analysis: - Constant (O(1)): Operations that take the same amount of time regardless of the input size. - Logarithmic (O(log n)): Operations that divide the problem size in

	half repeatedly. - Linear (O(n)): Operations that process each element in the input once. - Quadratic (O(n^2)): Operations that involve nested loops over the input. - Exponential (O(2^n)): Operations that double the amount of work for each additional input element. - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the difference between Big O and Big Omega notation? - What is the time complexity of an algorithm that searches for an element in a sorted array using binary search? - Give an example of an algorithm with linear time complexity. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	• Summarize the Lesson Learning Outcomes and get affirmation from students on these. • Suggested NPTEL video lecture https://onlinecourses.nptel.ac.in/noc25_cs23/unit?unit=16&lesson=17 **Wikipedia:**https://en.wikipedia.org/wiki/Asymptotic_analysis **GeeksforGeeks:** https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/ 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani ▪ Homework Given the following functions, order them by their asymptotic growth rate from slowest to fastest: • f1(n)=nlogn • f2(n)=$2n^2$ • f3(n)=5n • f4(n)=n^3 • f5(n)=$3n^2$ Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Asymptotic_analysis.

	Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 11	**Course Name: Design & Analysis of Algorithms** **Topic:** Well Known Sorting Algorithms	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. define and explain the concept of sorting algorithms. b. describe and compare the characteristics and time complexities of different sorting algorithms (e.g., Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort). c. analyze the strengths and weaknesses of different sorting algorithms in various scenarios.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - think about how they would sort a deck of playing cards. - Discuss different approaches they might use, emphasizing the importance of an organized and efficient process. - Introduce the concept of sorting algorithms as computer programs designed to arrange a collection of items (numbers, letters, etc.) in a specific order (e.g., ascending or descending). - **Development** (30 minutes) - **Basic Sorting Algorithms:** - **Bubble Sort:** - Explain the concept of repeatedly swapping adjacent elements if they are in the wrong order. - Discuss its time complexity (O(n^2) in the worst and average case, O(n) in the best case). - Mention its simplicity but inefficiency for large datasets. - **Selection Sort:** - Explain the process of finding the minimum element in the unsorted portion and placing it at the beginning. - Discuss its time complexity (O(n^2) in all cases). - Mention its advantage of making only O(n) swaps. - **Insertion Sort:** - Explain the process of iterating through the array and inserting each element into its correct position in the sorted portion. - Discuss its time complexity (O(n^2) in the worst and average case, O(n) in the best case). - Mention its efficiency for small datasets and its stability. - **Advanced Sorting Algorithms:** - **Merge Sort:** - Explain the divide-and-conquer approach: divide the array into halves, recursively sort each half, and then merge the sorted halves.

	- Discuss its time complexity (O(n log n) in all cases). - Mention its stability and its use in external sorting. - **Quick Sort:** - Explain the partitioning step: choose a pivot element and rearrange the array such that elements smaller than the pivot come before it, and elements larger than the pivot come after it. - Discuss its average-case time complexity (O(n log n)), but its worst-case time complexity (O(n^2)). - Mention its efficiency in practice and its use in many real-world applications. - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - Which sorting algorithm has the best worst-case time complexity? - What is the main idea behind the Bubble Sort algorithm? - In which scenario would Insertion Sort be a good choice? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	• Summarize the Lesson Learning Outcomes and get affirmation from students on these. • Suggested NPTEL video lecture https://onlinecourses.nptel.ac.in/noc25_cs23/ **Wikipedia:** https://en.wikipedia.org/wiki/Sorting_algorithm. 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani ▪ Homework - Compare and contrast the time complexities of Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on sorting algorithms. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 12	**Course Name: Design & Analysis of Algorithms** **Topic:** Comparison of Sorting Algorithms, Best-Case and Worst-Case Analyses, Average-Case Analysis	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. define and explain best-case, worst-case, and average-case scenarios for sorting algorithms. b. compare the time complexity of different sorting algorithms (e.g., Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort) in terms of best-case, worst-case, and average-case scenarios. c. analyze the impact of input data distribution on the performance of sorting algorithms.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - importance of efficient sorting algorithms in various applications (e.g., database systems, search engines, data visualization). - Briefly introduce different sorting algorithms (Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort). - Emphasize that the performance of a sorting algorithm can vary significantly depending on the input data. - **Development** (30 minutes) - **Best-Case, Worst-Case, and Average-Case Analysis:** - **Best-Case:** - Define best-case scenario as the input data that results in the fastest possible execution time for the algorithm. - Analyze the best-case time complexity of different sorting algorithms. - **Worst-Case:** - Define worst-case scenario as the input data that results in the slowest possible execution time for the algorithm. - Analyze the worst-case time complexity of different sorting algorithms. - **Average-Case:** - Define average-case scenario as the expected performance of the algorithm on a random input. - Analyze the average-case time complexity of different sorting algorithms. - **Comparison of Sorting Algorithms:** - Create a table comparing the best-case, worst-case, and average-case time complexities of different sorting algorithms

	(Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort). - Discuss the strengths and weaknesses of each algorithm based on their time complexity analysis. - **Impact of Input Data Distribution:** - Discuss how different input data distributions (e.g., sorted, reverse-sorted, random) can affect the performance of sorting algorithms. - Analyze the impact of input data distribution on the best-case, worst-case, and average-case scenarios. - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the worst-case time complexity of Bubble Sort? - Which sorting algorithm typically exhibits the best average-case performance? - How does the input data distribution affect the performance of Quick Sort? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	• Summarize the Lesson Learning Outcomes and get affirmation from students on these. • Suggested NPTEL video lecture https://onlinecourses.nptel.ac.in/noc25_cs23/ **Wikipedia:** https://en.wikipedia.org/wiki/Sorting_algorithm. 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani ▪ Homework - Compare and contrast the time complexities of Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on sorting algorithms. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 13	**Course Name: Design & Analysis of Algorithms** **Topic:** Amortized Analysis	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. understand the concept of amortized analysis and its importance in analyzing the performance of algorithms. b. learn and apply different techniques of amortized analysis, such as aggregate method, accounting method, and potential method. c. analyze the amortized cost of operations in dynamic data structures, such as stacks, queues, and binary heaps.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - the limitations of traditional worst-case analysis. - Explain that worst-case analysis can sometimes overestimate the actual cost of a sequence of operations, especially in dynamic data structures where individual operations may have high costs but are infrequent. - Introduce amortized analysis as a technique to analyze the average cost of a sequence of operations over time. - **Development** (30 minutes) - **Aggregate Method:** - Explain that the aggregate method analyzes the total cost of a sequence of n operations. - If the total cost of n operations is T(n), then the amortized cost per operation is T(n)/n. - Provide an example: Analyze the cost of n push and pop operations on a stack, where the cost of a push operation is 1 and the cost of a pop operation is 1. - **Accounting Method:** - Explain that the accounting method assigns different "charges" to different operations. - Some operations may be charged more than their actual cost, while others may be charged less. - The extra "credit" from overcharged operations can be used to pay for undercharged operations later. - Provide an example: Analyze the cost of incrementing a binary counter using the accounting method. - **Potential Method:** - Explain that the potential method assigns a "potential" to the data structure after each operation. - The amortized cost of an operation is defined as the actual cost plus the change in potential.

	- The potential function is chosen to ensure that the amortized cost of a sequence of operations is bounded. - Provide an example: Analyze the cost of operations on a dynamic table using the potential method. - **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the main difference between worst-case analysis and amortized analysis? - Describe the aggregate method of amortized analysis. - What is the purpose of a potential function in the potential method? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	• Summarize the Lesson Learning Outcomes and get affirmation from students on these. • Suggested NPTEL video lecture **Design and Analysis of Algorithms - NPTEL** **Wikipedia: Amortized analysis** 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani Homework - a. Consider an implementation of a stack that supports the following operations: - Push: Add an element to the stack. - Pop: Remove the top element from the stack. Suppose that for every **Push** operation, you charge 2 units, and for every **Pop** operation, you charge 1 unit. Prove that the **amortized cost** per operation is constant (i.e., O(1)). Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Amortized Analysis. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 13	**Course Name: Design & Analysis of Algorithms** **Topic:** Divide-and-Conquer, General Method	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. understand the general principle of the divide-and-conquer paradigm. b. identify the three steps involved in the divide-and-conquer approach. c. analyze the time complexity of divide-and-conquer algorithms using recurrence relations.
Teaching Aids (if any)	a. Power point presentation b. Use of whiteboard.
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - presenting a real-world analogy of the divide-and-conquer strategy, such as the "divide and conquer" approach in military tactics. - Explain that divide-and-conquer is a powerful algorithmic paradigm used to solve complex problems by breaking them down into smaller, more manageable subproblems. - **Development** (30 minutes) - **General Divide-and-Conquer Strategy:** - Describe the three key steps involved in the divide-and-conquer approach: - Divide: Divide the original problem into a set of smaller subproblems. - Conquer: Solve each subproblem recursively. If the subproblem is small enough, solve it directly. - Combine: Combine the solutions to the subproblems to obtain the solution to the original problem. - **Examples of Divide-and-Conquer Algorithms:** - Briefly introduce some well-known examples of divide-and-conquer algorithms: - Merge Sort - Quick Sort - Binary Search - Maximum Subarray Problem - Explain how these algorithms follow the divide-and-conquer paradigm. - **Analysis of Divide-and-Conquer Algorithms:** - Introduce the concept of recurrence relations to analyze the time complexity of divide-and-conquer algorithms. - Explain how to set up a recurrence relation based on the divide-and-conquer steps. - Briefly discuss methods for solving recurrence relations (e.g., substitution method, master theorem).

	- **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What are the three steps involved in the divide-and-conquer approach? - How does the merge sort algorithm use the divide-and-conquer strategy? -What is a recurrence relation, and how is it used in the analysis of divide-and-conquer algorithms? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **https://onlinecourses.nptel.ac.in/noc25_cs23/** **Wikipedia: Divide-and-conquer algorithm** 3. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani Homework a. Explain the general strategy of the **Divide-and-Conquer** method. How does it break down a problem, and what are the advantages of using this approach? b. Describe the time complexity of a typical **Divide-and-Conquer** algorithm and explain why it's often expressed using a recurrence relation. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on **Divide-and-Conquer** method. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 14	Course Name: Design & Analysis of Algorithms Topic: Multiplication of two n-bit numbers	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. understand the traditional method of multiplying two n-bit numbers and its time complexity. b. learn and explain the divide-and-conquer approach to multiplying two n-bit numbers (Karatsuba's algorithm). c. analyze the time complexity of Karatsuba's algorithm and compare it with the traditional method.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - understand the traditional method of multiplying two n-bit numbers and its time complexity. - learn and explain the divide-and-conquer approach to multiplying two n-bit numbers (Karatsuba's algorithm). - analyze the time complexity of Karatsuba's algorithm and compare it with the traditional method. - **Development** (30 minutes) **- Grade School Multiplication:** - Explain the traditional method for multiplying two n-bit numbers. - Illustrate with a simple example (e.g., multiplying two 4-bit numbers). - Analyze the time complexity of this method (O(n^2)). **Karatsuba Algorithm:** - Introduce the divide-and-conquer approach. - Explain the core idea of Karatsuba's algorithm (reducing three multiplications to two). - Derive the recurrence relation and solve it to obtain the time complexity (O(n^log3)). **Significance of Efficient Multiplication:** - Discuss the impact of efficient multiplication algorithms on various fields: - Cryptography (e.g., RSA algorithm) - Scientific computing (e.g., matrix multiplication) - Computer graphics (e.g., image processing) **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the time complexity of the grade school multiplication algorithm? - How does the Karatsuba algorithm improve upon the grade school method?

	- What is the significance of efficient multiplication algorithms in cryptography? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture -NPTEL 3. Reference website: Karatsuba Algorithm on Wikipedia **4.** Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - Describe the Karatsuba multiplication algorithm. Given two n-bit numbers A and B, explain how Karatsuba reduces the number of recursive multiplications and the overall time complexity. How does the time complexity compare to the naive long multiplication method? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Types of Algorithms. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 15	Course Name: Design & Analysis of Algorithms Topic: Binary Search	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. understand the concept of binary search. b. identify when binary search can be applied. c. implement binary search in pseudocode or a simple programming language. d. analyze the time complexity of binary search.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - the limitations of linear search: it can be very slow for large datasets. - Introduce the concept of "divide and conquer" as a core principle in algorithm design. - Briefly explain that binary search is a powerful technique that leverages this principle. **Development** (30 minutes) - **Prerequisites:** - Basic understanding of arrays or lists. - Basic understanding of sorting algorithms (e.g., how to sort an array). - **Key Concepts:** - Core Idea: Binary search works only on sorted arrays. It repeatedly divides the search interval in half. If the value being searched is smaller than the middle element, the search continues in the left half. If it's larger, the search continues in the right half. - **Implementation:** o Initialize two pointers: left (pointing to the beginning) and right (pointing to the end) of the array. o While left is less than or equal to right: ▪ Calculate the mid index. ▪ Compare the target value with the element at mid. ▪ If target is equal to mid, return mid. ▪ If target is less than mid, update right to mid - 1. ▪ If target is greater than mid, update left to mid + 1. o If the loop completes without finding the target, return -1 (indicating that the target is not found). - **Time Complexity:**

	- Binary search has a time complexity of O(log n), where 'n' is the number of elements in the array. This makes it significantly faster than linear search,especially for large datasets. **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - **True or False:** Binary search can be applied to any unsorted array. - What is the time complexity of binary search? -In which step of the binary search algorithm do you adjust the search interval? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **https://www.youtube.com/embed/hT1dIbFQoGE** 3. Reference website: **Wikipedia:**https://en.wikipedia.org/wiki/Binary_search **GeeksforGeeks:** https://www.geeksforgeeks.org/binary-search/ **LeetCode:** https://leetcode.com/problems/binary-search/ **4.** Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - Explain binary search with example. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on binary search. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 16	**Course Name: Design & Analysis of Algorithms** **Topic:** Merge Sort	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of Divide and Conquer. b. Learn the Merge Sort algorithm and its steps. c. Analyze the time and space complexity of Merge Sort. d. Apply Merge Sort to solve sorting problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - introducing the concept of sorting algorithms and their importance in computer science. - Briefly discuss the need for efficient sorting algorithms, especially for large datasets. - Introduce the Divide and Conquer paradigm as a powerful problem-solving technique. - Briefly mention other sorting algorithms like Bubble Sort, Insertion Sort, and Selection Sort. **Development** (30 minutes) **Divide and Conquer:** - Explain the core idea of Divide and Conquer: - Divide the problem into smaller subproblems. - Conquer each subproblem recursively. - Combine the solutions of the subproblems to obtain the solution to the original problem. **Merge Sort Algorithm:** - Describe the steps involved in Merge Sort: - Divide: If the given array has more than one element, divide it into two halves. - Conquer: Recursively sort the left and right halves. - Combine: Merge the sorted left and right halves into a single sorted array. - **Merge Operation:** - Explain the key step of merging two sorted arrays efficiently. - Illustrate the merge process with a step-by-step example using a whiteboard or visual aids. **Time and Space Complexity:** - Discuss the time complexity of Merge Sort: O(n log n) in all cases (best, average, and worst). - Explain the space complexity of Merge Sort: O(n) due to the extra space required for the auxiliary array during merging. **Exercise** (10 minutes) –

	- Group Discussion at the end of the session. - Ask students - What is the core principle behind the Merge Sort algorithm? - What is the time complexity of Merge Sort in the best, average, and worst cases? - Why does Merge Sort have a space complexity of O(n)? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms** 3. Reference website: **GeeksforGeeks: https://www.geeksforgeeks.org/merge-sort/** **Wikipedia:https://en.wikipedia.org/wiki/Merge_sort** **Programiz: https://www.programiz.com/dsa/merge-sort** **4.** Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - Explain merge sort with example. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on merge sort . Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 17	**Course Name: Design & Analysis of Algorithms** **Topic:** Quick Sort	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. understand the concept of partitioning in Quick Sort. b. implement the Quick Sort algorithm. c. analyze the time and space complexity of Quick Sort. d. compare Quick Sort with other sorting algorithms (e.g., Bubble Sort, Merge Sort).
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - the importance of efficient sorting algorithms in computer science. - Briefly introduce the concept of divide-and-conquer algorithms. - Introduce Quick Sort as a popular divide-and-conquer sorting algorithm known for its efficiency in practice. - Briefly mention its inventor, C. A. R. Hoare. **Development** (30 minutes) **Partitioning:** - Explain the core concept of partitioning: selecting a "pivot" element and rearranging the array such that elements smaller than the pivot are on one side, and larger elements are on the other. - Illustrate the partitioning process with a simple example using a whiteboard or projector. **Quick Sort Algorithm:** - Describe the recursive nature of Quick Sort: o Partition the array around the pivot. o Recursively sort the sub-array to the left of the pivot. o Recursively sort the sub-array to the right of the pivot. - Present the Quick Sort algorithm in pseudocode or a simple programming language. **Time and Space Complexity:** - Discuss the average-case time complexity of Quick Sort: O(n log n). - Briefly mention the worst-case scenario (O(n^2)) and how to mitigate it (e.g., randomized pivot selection). - Discuss the space complexity of Quick Sort: O(log n) in the average case, O(n) in the worst case. **Exercise** (10 minutes) – - Group Discussion at the end of the session.

	- Ask students - What is the core idea behind the partitioning step in Quick Sort? - Describe the recursive nature of the Quick Sort algorithm. - What is the average-case time complexity of Quick Sort? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms** 3. Reference website: **Wikipedia:** **https://en.wikipedia.org/wiki/Quicksort** **GeeksforGeeks:https://www.geeksforgeeks.org/quick-sort-algorithm/** **Khan Academy:https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort** **4.** Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - Explain Quick sort with example. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Quick sort . Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 18	**Course Name: Design & Analysis of Algorithms** **Topic:** Stassen's Matrix multiplication	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of Stassen's Matrix Multiplication and its applications. b. Learn the steps involved in performing Stassen's Matrix Multiplication. c. apply Stassen's Matrix Multiplication to solve real-world problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - reviewing the concept of traditional matrix multiplication. - Discuss the limitations of traditional matrix multiplication, such as computational complexity and memory requirements. - Introduce Stassen's Matrix Multiplication as an alternative approach that can improve efficiency in certain scenarios. **Development** (30 minutes) **Explain the core concept:** - Describe the fundamental idea behind Stassen's Matrix Multiplication. - Highlight how it differs from traditional matrix multiplication by exploiting specific properties of the matrices involved**.** **Step-by-Step Procedure:** - Outline the steps involved in performing Stassen's Matrix Multiplication. - Use a simple example to demonstrate the process. - Emphasize the importance of identifying and utilizing the specific properties of the matrices to optimize the calculation. **Applications:** - Discuss real-world applications of Stassen's Matrix Multiplication, such as in image processing, machine learning, and scientific computing. - Briefly explain how Stassen's method can lead to performance improvements in these domains. **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the key difference between Stassen's Matrix Multiplication and traditional matrix multiplication? - Given two matrices, can you identify if Stassen's Matrix Multiplication can be applied and why? - Calculate the product of two simple matrices using Stassen's

	Matrix Multiplication. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms** 3. Reference website: **Divide and Conquer | Set 5 (Strassen's Matrix Multiplication) - GeeksforGeeks** **Strassen's Matrix Multiplication** **https://www.youtube.com/embed/pWInbKQiYTY** **4.** Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - Explain Strassen's Matrix Multiplication with example. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Strassen's Matrix Multiplication . Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 19	**Course Name: Design & Analysis of Algorithms** **Topic:** Dynamic Programming, General Method	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the core concept of Dynamic Programming (DP). b. Identify problems that can be effectively solved using DP. c. Learn the general approach and steps involved in solving DP problems. d. Apply the concept of memoization and tabulation to optimize DP solutions.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Start with a real-world analogy, such as climbing stairs. You can take one or two steps at a time. How many different ways can you reach the top? - Explain that this problem can be broken down into smaller subproblems, and the solutions to these subproblems can be reused to find the overall solution. - Introduce Dynamic Programming as a powerful technique for solving problems that exhibit overlapping subproblems and optimal substructure. **Development** (30 minutes) **Overlapping Subproblems:** - Define overlapping subproblems: Solutions to the same subproblem are computed repeatedly. - Example: Fibonacci sequence calculation. Optimal Substructure: - Define optimal substructure: The optimal solution to a problem can be constructed from the optimal solutions to its subproblems. - Example: Shortest path problem. **General Approach to Solving DP Problems:** Step 1: Identify overlapping subproblems. Step 2: Define a recursive relation to express the solution of the problem in terms of solutions to its subproblems. Step 3: Implement the recursive solution using memoization (top-down approach) or tabulation (bottom-up approach). **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What are the two key characteristics of problems that can be effectively solved using Dynamic Programming? - Briefly explain the difference between memoization and

	tabulation. - Give an example of a problem that you think might be solvable using Dynamic Programming. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms** 3. Reference website: **GeeksforGeeks: https://www.geeksforgeeks.org/dynamic-programming/** **Wikipedia:https://simple.wikipedia.org/wiki/Dynamic_programming** **Dynamic Programming - Introduction" by Tushar Roy** **4.** Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani 5. Homework - Explain variety of classic dynamic programming challenges and will help strengthen your understanding of different techniques like memoization, tabulation, and optimization of space complexity. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Dynamic Programming. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 21	**Course Name: Design & Analysis of Algorithms** **Topic:** Chained Matrix Multiplication	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of matrix multiplication and its associativity. b. Learn the significance of efficient matrix multiplication order. c. Apply dynamic programming to find the optimal order of matrix multiplication. d. Analyze the time and space complexity of the algorithm.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - briefly reviewing the concept of matrix multiplication. - Discuss the importance of efficient matrix multiplication in various applications like image processing, data analysis, and machine learning. - Introduce the problem of chained matrix multiplication: Given a sequence of matrices, find the most efficient order to multiply them to minimize the total number of scalar multiplications. **Development** (30 minutes) **Cost of Matrix Multiplication:** - Explain the cost of multiplying two matrices: If matrix A has dimensions p x q and matrix B has dimensions q x r, then the cost of multiplying A and B is p * q * r scalar multiplications. - Associativity of Matrix Multiplication: - Emphasize that matrix multiplication is associative, meaning the order of multiplication can be rearranged without changing the result. - However, the cost of multiplication can vary significantly depending on the order. **Dynamic Programming Approach:** - Introduce the concept of dynamic programming and how it can be applied to solve the chained matrix multiplication problem. - Explain the recursive nature of the problem and how overlapping subproblems can be avoided using memoization or tabulation. - Derive the recurrence relation for the minimum cost of multiplying a chain of matrices. - Discuss the implementation of the dynamic programming algorithm using a 2D table. **Exercise** (10 minutes) – - Group Discussion at the end of the session.

	- Ask students - Given matrices A (10x20), B (20x5), C (5x30), what is the cost of multiplying (A * B) * C? - What is the key idea behind using dynamic programming for the chained matrix multiplication problem? - How does the number of scalar multiplications affect the overall efficiency of matrix multiplication? Give an example of a problem that you think might be solvable using Dynamic Programming. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture **NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms** 3. Reference website: **Matrix Chain Multiplication - GeeksforGeeks** **Matrix Chain Multiplication Algorithm - javatpoint** www.youtube.com/embed/m092HzUZ6Wc **4.** Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - You are given a sequence of matrices A1,A2,...,An with dimensions p0×p1, p1×p2…p-1 ×pn. You need to find the most efficient way to multiply these matrices, minimizing the number of scalar multiplications. - Given the following matrix dimensions: A1:10×20, A2:20×30, A3:30×40, A4:40×30 - **a)** Compute the minimum number of scalar multiplications needed to multiply the matrices together using the matrix chain multiplication algorithm. - **b)** Show the parenthesization of the matrix chain that achieves the minimum scalar multiplications. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. 2. Quiz on Chained Matrix Multiplication. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 25	**Course Name: Design & Analysis of Algorithms** **Topic:** Greedy Algorithms, General Method	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the core concept of greedy algorithms. b. Learn the general method for designing greedy algorithms. c. Analyze the strengths and weaknesses of greedy algorithms. d. Apply greedy algorithms to solve simple problems (e.g., activity selection, fractional knapsack).
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you're making change for a customer. You instinctively give them the largest possible bills/coins first to minimize the number of coins used. This is a simple example of a greedy approach. - Briefly explain that greedy algorithms make locally optimal choices at each step with the hope of finding the global optimum. - not all problems can be solved optimally using greedy algorithms. **Development** (30 minutes) **General Method:** - Define the problem: Clearly understand the goal and constraints. - Make a greedy choice: Select the locally optimal choice at each step. - Reduce the problem: After making the greedy choice, reduce the original problem to a smaller subproblem. - Repeat: Continue making greedy choices and reducing the problem until a solution is reached. **Example: Activity Selection Problem** - Explain the problem: Given a set of activities with start and finish times, select the maximum number of activities that can be performed without any overlaps. - Demonstrate the greedy choice: Select the activity with the earliest finish time. - Show how to reduce the problem: Remove all activities that overlap with the selected activity. - Discuss the correctness of the greedy choice (optional, depending on time). **Example: Fractional Knapsack Problem**

	- Explain the problem: Given a set of items with weights and values, select items to maximize the total value within a given weight limit. - Demonstrate the greedy choice: Select items with the highest value-to-weight ratio first. - Discuss the correctness of the greedy choice (optional, depending on time). **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - A greedy algorithm always finds the optimal solution for every problem. - What is the key characteristic of a greedy choice? - a) It must be the largest possible choice. - b) It must be the smallest possible choice. - c) It must be the locally optimal choice. - d) It must be a random choice. - Briefly describe the general steps involved in designing a greedy algorithm. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **Greedy Algorithms General Structure - GeeksforGeeks** Greedy Algorithms Introduction - javatpoint 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - you are given a set of activities with their start and finish times. You need to select the maximum number of activities that can be performed by a single person, assuming that each activity takes no more than one time unit and that a person can only do one activity at a time. *Example:* Activities: [(1, 4), (2, 5), (3, 6), (4, 7), (5, 8)] Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss.

	- Quiz on greedy algorithm. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 26	**Course Name: Design & Analysis of Algorithms** **Topic:** Knapsack problem	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of the Knapsack Problem. b. Learn the Greedy approach to solving the Knapsack Problem (specifically Fractional Knapsack). c. Analyze the time complexity of the Greedy algorithm for Fractional Knapsack. d. Apply the Greedy algorithm to solve real-world problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you're a hiker going on a long trek. You have a limited backpack capacity (weight limit) and need to choose which items (with different values and weights) to pack for the maximum overall value. - Introduce the Knapsack Problem: A classic optimization problem where you need to select items from a set to maximize their value while staying within a given weight constraint. - Briefly mention the difference between 0/1 Knapsack (where you can't take a fraction of an item) and Fractional Knapsack (where you can take fractions of items). **Development** (30 minutes) **Fractional Knapsack:** - Explain the Greedy approach: o Calculate the value-to-weight ratio (value/weight) for each item. o Select items with the highest value-to-weight ratio first. o If the weight of the current item exceeds the remaining capacity, take a fraction of the item to fill the remaining capacity. - Illustrate with a simple example: Provide a table with items, their weights, and their values. Demonstrate the step-by-step process of applying the Greedy algorithm. - Analyze time complexity: Discuss how the Greedy algorithm for Fractional Knapsack can be implemented in O(n log n) time using sorting (e.g., heapsort). **Real-world applications:** - Briefly discuss potential applications of the Knapsack Problem, such as resource allocation, portfolio optimization, and load balancing.

	Exercise (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the core principle of the Greedy approach to solving the Fractional Knapsack Problem? - Why does the Greedy algorithm work optimally for the Fractional Knapsack Problem? - Given a set of items with their weights and values, how would you determine which item to select first using the Greedy approach? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/Knapsack_problem** **https://www.geeksforgeeks.org/fractional-knapsack-problem/** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - Given a set of items with weights and values, and a knapsack with a weight capacity, determine the maximum value you can carry in the knapsack using a greedy approach. You can take fractions of an item. *Example:* Items: [(60, 10), (100, 20), (120, 30)], Knapsack Capacity = 50 Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Knapsack problem. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 31	**Course Name: Design & Analysis of Algorithms** **Topic:** Dijkstra's Single Source Shortest Path Algorithm	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of graph traversal and shortest paths. b. Learn the Dijkstra's algorithm for finding the shortest paths from a single source node to all other nodes in a weighted graph. c. Analyze the algorithm's time complexity and understand its limitations. d. Apply Dijkstra's algorithm to solve real-world problems (e.g., network routing, transportation).
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you're planning a road trip. You want to find the shortest routes from your starting city to all other cities. How would you approach this problem? - Introduce the concept of weighted graphs: Graphs where edges have associated weights (e.g., distances, costs). - Briefly explain the goal: Find the shortest path from a given source node to all other nodes in a weighted graph. - State that Dijkstra's algorithm is a popular and efficient algorithm for solving this problem. **Development** (30 minutes) **Algorithm Overview:** - Explain the core idea: Iteratively find the shortest path to the nearest unvisited node. - Discuss the use of a priority queue (e.g., min-heap) to efficiently select the next node. **Algorithm Steps:** - Initialize distances to all nodes as infinity, except for the source node (distance 0). - Create a priority queue to store nodes and their tentative distances. - While the priority queue is not empty: o Extract the node with the minimum distance. o For each neighbor of the extracted node: ▪ Calculate the tentative distance to the neighbor. ▪ If the tentative distance is less than the current distance to the neighbor, update the distance. **Example:**

	- Work through a simple example with a small graph (e.g., 5-6 nodes) to demonstrate the step-by-step application of Dijkstra's algorithm. **Limitations:** - Discuss the limitation of Dijkstra's algorithm: It only works correctly with non-negative edge weights. **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the purpose of Dijkstra's algorithm? - What data structure is commonly used to implement Dijkstra's algorithm efficiently? - What is the limitation of Dijkstra's algorithm regarding edge weights? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm** **https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - What is Dijkstra's Algorithm used for? What type of graphs does it work on? - Describe the main steps of Dijkstra's Algorithm for finding the shortest path from a source node to all other nodes in a weighted graph. - Explain how the algorithm selects the next node to process during its execution. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Dijkstra's Single Source Shortest Path Algorithm. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 22	**Course Name: Design & Analysis of Algorithms** **Topic:** Backtracking, General method	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the fundamental concept of backtracking. b. Learn the general algorithm for backtracking problems. c. Identify problems that can be solved using the backtracking approach. d. Apply the backtracking algorithm to solve simple problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - the concept of exhaustive search, where all possible solutions are explored. - Explain that backtracking is a more efficient approach to exhaustive search. - Introduce the idea of making choices and then undoing those choices if they lead to dead ends. - Provide a simple real-world analogy, such as solving a maze. **Development** (30 minutes) General Backtracking Algorithm: Step 1: Initialization: o Start with an empty solution set. o Initialize the starting state of the problem. Step 2: Choice: o Make a choice among the available options at the current state. Step 3: Constraint Check: o Check if the chosen option is valid based on the problem constraints. Step 4: Advance: o If the choice is valid, move to the next state and repeat steps 2-4. Step 5: Backtrack: o If the current state is invalid or a dead end is reached, backtrack to the previous state and try a different choice. Step 6: Solution: o If a complete and valid solution is found, add it to the solution set. Step 7: Termination: o Continue until all possible choices have been explored. - Example Problem (Illustrate with a simple example, such as the N-Queens problem):

	- Demonstrate how the backtracking algorithm works step-by-step for the chosen problem. - Visualize the process using diagrams or a whiteboard. - Emphasize the importance of constraint checking and backtracking to avoid unnecessary exploration. Applications of Backtracking: - Briefly discuss some common applications of backtracking, such as: o Solving puzzles (Sudoku, crossword puzzles) o Finding paths in graphs (maze solving, game AI) o Constraint satisfaction problems (scheduling, resource allocation) **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the core idea behind the backtracking approach? - Describe the two main steps involved in the backtracking algorithm. - Can you name one problem that can be effectively solved using backtracking? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: GeeksforGeeks: https://www.geeksforgeeks.org/backtracking-algorithms/ Wikipedia: https://en.wikipedia.org/wiki/Backtracking https://www.youtube.com/embed/oRTk5mTdXFA 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - Given a graph, implement a backtracking algorithm to check if the graph can be colored with a given number of colors such that no two adjacent nodes have the same color. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on backtracking algorithm.

	Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 23	**Course Name: Design & Analysis of Algorithms** **Topic:** N-Queens Problem	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the N-Queens Problem and its significance in backtracking algorithms. b. Learn to formulate the problem as a constraint satisfaction problem. c. Implement a basic backtracking algorithm to solve the N-Queens problem. d. Analyze the time and space complexity of the algorithm.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - a brief overview of backtracking algorithms and their general approach. - Introduce the N-Queens Problem: "Place N chess queens on an N×N chessboard such that no two queens can attack each other." - Explain the significance of the problem in demonstrating the power and limitations of backtracking. **Development** (30 minutes) **Problem Formulation:** - Discuss how to represent the chessboard (e.g., using a 2D array). - Define the constraints: o No two queens can be in the same row. o No two queens can be in the same column. o No two queens can be on the same diagonal. **Backtracking Algorithm:** - Present a high-level overview of the backtracking approach: o Start with an empty board. o Place a queen in the first row. o Recursively try to place queens in subsequent rows. o If a placement violates any constraints, backtrack and try a different placement. o If all N queens are placed successfully, record the solution. **Implementation (briefly):** - Discuss the key steps involved in the implementation: o Checking for row, column, and diagonal conflicts. o Backtracking mechanism. o Storing and printing solutions. **Exercise** (10 minutes) –

	- Group Discussion at the end of the session. - Ask students - What is the main idea behind backtracking algorithms? - How would you represent the placement of queens on the chessboard in your code? - Explain the concept of diagonal conflicts in the N-Queens problem. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: Wikipedia: N-Queens Problem GeeksforGeeks: N Queen Problem LeetCode: N-Queens 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - How can you optimize the backtracking approach for the N-Queens problem using constraint propagation techniques like forward checking or constraint graphs? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on N-Queens problem. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 24	**Course Name: Design & Analysis of Algorithms** **Topic:** Sum of subsets problem	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of the Sum of Subsets problem. b. Learn different approaches to solving the Sum of Subsets problem, including backtracking. c. Analyze the time and space complexity of the backtracking algorithm. d. Apply the Sum of Subsets problem to real-world scenarios.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - a brief overview of the concept of subsets. - Introduce the Sum of Subsets problem: Given a set of non-negative integers, find all subsets of the set that sum to a given target value. - Motivate the problem with a simple real-world example, such as finding combinations of coins that add up to a specific amount. **Development** (30 minutes) **Backtracking Approach:** - Explain the core idea of backtracking: systematically exploring different combinations by making choices and undoing them if they don't lead to a solution. - Present the backtracking algorithm for the Sum of Subsets problem step-by-step. - Illustrate the algorithm with a concrete example, tracing the recursive calls and the exploration of the solution space. **Time and Space Complexity:** - Discuss the time complexity of the backtracking algorithm, highlighting its exponential nature in the worst case. - Analyze the space complexity, considering both stack space for recursion and the space required to store the subsets. **Applications:** - Discuss potential applications of the Sum of Subsets problem, such as: o Knapsack problem (a variation where each item has a weight and value) o Budgeting and resource allocation o Combinatorial optimization problems **Exercise** (10 minutes) –

	- Group Discussion at the end of the session. - Ask students - What is the core idea behind the backtracking approach to solving the Sum of Subsets problem? - Explain the time complexity of the backtracking algorithm for the Sum of Subsets problem. - Give an example of a real-world scenario that can be modeled using the Sum of Subsets problem.. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: Sum of Subsets Problem on GeeksforGeeks Sum of Subsets Problem on Wikipedia https://www.youtube.com/embed/QTwh6l2Mk2Q 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - Given a set of integers, find the subset whose sum equals the target value, if it exists. **Input**: Set S={1,5,11,5} and target T=10. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Sum of Subsets Problem. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 27	**Course Name: Design & Analysis of Algorithms** **Topic:** Job sequencing with deadlines	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of job scheduling and its importance. b. Learn the Greedy approach to solving the Job Sequencing with Deadlines problem. c. Analyze and implement the Greedy algorithm for job scheduling. d. Apply the algorithm to solve real-world scheduling problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you're a freelancer with multiple projects. Each project has a deadline and a profit associated with it. How do you choose which projects to take on to maximize your overall profit? - Introduce the Job Sequencing problem: Given a set of jobs with deadlines and associated profits, schedule the jobs to maximize the total profit. - Briefly explain the Greedy approach: A problem-solving technique that makes the locally optimal choice at each step, hoping to find the global optimum. **Development** (30 minutes) **Problem Formulation:** - Define the problem formally: Given a set of n jobs, each with a deadline and a profit. Find a schedule that maximizes the total profit of the jobs executed within their deadlines. **Greedy Strategy:** - Sort jobs in descending order of profit. - Create an array 'slots' to represent available time slots. - For each job in the sorted order: o Find the latest available slot that meets the job's deadline. o If a slot is available, schedule the job in that slot. **Algorithm Implementation:** - Discuss the implementation details, including data structures (arrays, lists) and the logic for finding the latest available slot. Example: - Work through a simple example with 4-5 jobs, demonstrating the step-by-step application of the Greedy algorithm. **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the key principle behind the Greedy approach to

	solving the Job Sequencing problem? - How are jobs sorted in the Greedy algorithm for Job Sequencing? - Given a job with a deadline and a profit, how do you determine if it can be scheduled? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://www.geeksforgeeks.org/job-sequencing-problem/** **https://www.youtube.com/watch?v=LX3LU5rrQbw** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - You are given a set of jobs with their deadlines and profits. Each job takes 1 unit of time to complete. Your task is to schedule the jobs to maximize the total profit, ensuring that no two jobs overlap their deadlines. Use a greedy approach based on profit to solve this problem. *Example:* Jobs = [(5, 2), (2, 1), (3, 5), (7, 3)], Deadlines = [3, 1, 2, 3] Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Job sequencing with deadlines. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 28	**Course Name: Design & Analysis of Algorithms** **Topic:** Minimum Spanning Trees	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of graphs and their representations (e.g., adjacency matrix, adjacency list). b. Define and understand the concept of a spanning tree. c. Learn the key properties of Minimum Spanning Trees (MST). d. Explore and compare different algorithms for finding MSTs: e. Prim's Algorithm f. Kruskal's Algorithm g. Apply MST algorithms to solve real-world problems (e.g., network design, transportation).
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine designing a network for a group of computers. You want to connect all computers with the minimum amount of cable. How would you approach this problem? - Introduce the concept of graphs: A collection of nodes (vertices) connected by edges. - Briefly explain the concept of a spanning tree: A subgraph that connects all nodes without forming any cycles. - State the goal: Find the spanning tree with the minimum total weight of edges (Minimum Spanning Tree). **Development** (30 minutes) **Prim's Algorithm:** - Explain the core idea: Start with an arbitrary node and iteratively add the minimum-weight edge that connects the current tree to a new node. - Discuss the implementation using a priority queue (e.g., min-heap) to efficiently find the minimum-weight edge. **Kruskal's Algorithm:** - Explain the core idea: Sort edges by weight in ascending order. Add edges to the tree one by one, ensuring no cycles are formed. - Discuss the use of the Disjoint Set Union (DSU) data structure to efficiently check for cycles. **Comparison of Algorithms:** - Briefly compare Prim's and Kruskal's algorithms in terms of time and space complexity, and discuss their suitability for different graph densities. **Exercise** (10 minutes) –

	- Group Discussion at the end of the session. - Ask students - What is a spanning tree? - What is the key difference between Prim's and Kruskal's algorithms for finding MSTs? - In Kruskal's algorithm, how do we ensure that no cycles are formed when adding edges? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/Minimum_spanning_tree**4. Greedy Algorithms Introduction - javatpoint 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - What is a Minimum Spanning Tree (MST)? Explain the properties of an MST. - What is the difference between a spanning tree and a minimum spanning tree? - Why is a minimum spanning tree unique (if edge weights are distinct) and not necessarily unique if edge weights are not distinct? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on **Minimum Spanning Tree (MST)** Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 29	**Course Name: Design & Analysis of Algorithms** **Topic:** Kruskal's Algorithm	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of graphs and their representations. b. Define and understand the concept of a spanning tree and a Minimum Spanning Tree (MST). c. Learn the logic and steps involved in Kruskal's algorithm for finding MSTs. d. Understand the use of Disjoint Set Union (DSU) data structure in Kruskal's algorithm. e. Apply Kruskal's algorithm to solve simple graph problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you're designing a network for a group of computers. You want to connect all computers with the minimum amount of cable. - Introduce the concept of graphs: A collection of nodes (vertices) connected by edges. - Define a spanning tree: A subgraph that connects all nodes without forming any cycles. - State the goal: Find the spanning tree with the minimum total weight of edges (Minimum Spanning Tree) **Development** (30 minutes) **Kruskal's Algorithm:** - **Core Idea:** o Sort all edges of the graph in ascending order of their weights. o Select edges one by one in this sorted order. o Add an edge to the MST only if it does not create a cycle in the existing tree. - **Disjoint Set Union (DSU):** o Explain how DSU is used to efficiently check if adding an edge creates a cycle. o Briefly explain the key operations of DSU: union() and find(). - **Algorithm Steps:** o Sort all edges of the graph in ascending order of weight. o Create a DSU to keep track of connected components. o Iterate through the sorted edges: ▪ If adding the edge does not create a cycle (i.e., the nodes connected by the edge belong to different sets

	in DSU): ▪ Add the edge to the MST. ▪ Perform union() operation on the sets containing the two nodes. Example: - Work through a simple graph example (e.g., with 5-6 nodes) to demonstrate the step-by-step application of Kruskal's algorithm. **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the key principle behind Kruskal's algorithm for finding MSTs? - Why is the Disjoint Set Union (DSU) data structure used in Kruskal's algorithm? - How does DSU help in determining whether adding an edge creates a cycle? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/Kruskal%27s_algorithm** **Greedy Algorithms Introduction - javatpoint** **https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - Describe the steps of Kruskal's algorithm for finding the Minimum Spanning Tree (MST) in an undirected weighted graph. - Explain the reasoning behind the greedy approach of Kruskal's algorithm. Why does sorting edges by weight and adding them in non-decreasing order ensure the MST? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Kruskal's algorithm Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 30	Course Name: **Design & Analysis of Algorithms** **Topic:** Prim's Algorithm	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of graphs and their representations (e.g., adjacency matrix, adjacency list). b. Define and understand the concept of a spanning tree and a Minimum Spanning Tree (MST). c. Learn the logic and implementation of Prim's Algorithm for finding MSTs. d. Analyze the time and space complexity of Prim's Algorithm. e. Apply Prim's Algorithm to solve real-world problems (e.g., network design).
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Imagine you are building a network connecting several cities. You want to minimize the total cost of the connections while ensuring all cities are reachable. How would you approach this problem? - Introduce the concept of graphs: A collection of nodes (vertices) connected by edges (which may have weights). - Define a spanning tree: A subgraph that connects all nodes without forming any cycles. - Introduce the goal: Find the spanning tree with the minimum total weight of edges (Minimum Spanning Tree). **Development** (30 minutes) **Prim's Algorithm:** - Explain the core idea: Start with an arbitrary node. o Repeatedly select the minimum-weight edge that connects the current tree to a node not yet in the tree. o Continue until all nodes are included in the tree. - Discuss the implementation using a priority queue (e.g., min-heap) to efficiently find the minimum-weight edge. o Explain how to use a priority queue to store nodes and their associated edge weights. - Illustrate the algorithm with a step-by-step example on a small graph. **Exercise** (10 minutes) – - Group Discussion at the end of the session.

	- Ask students - What is a Minimum Spanning Tree (MST)? - What data structure is commonly used to implement Prim's Algorithm efficiently? - In Prim's Algorithm, how is the next node to be added to the tree selected? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/Prim%27s_algorithm** **Prim's Algorithm** **Prim's Algorithm for Minimum Spanning Tree (MST) - GeeksforGeeks** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - What is Prim's Algorithm used for, and what type of graphs does it work on? - Explain the steps involved in Prim's Algorithm. - Given a graph with weighted edges, how does Prim's Algorithm select the next vertex to add to the Minimum Spanning Tree (MST)? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Prim's Algorithm. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 32	**Course Name: Design & Analysis of Algorithms** **Topic:** Non-Deterministic Algorithms	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of determinism and non-determinism in computation. b. Learn how non-deterministic Turing machines work. c. Explore the relationship between non-deterministic and deterministic algorithms. d. Understand the concept of the NP class and its significance. e. Gain an intuitive understanding of NP-complete problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Begin by discussing the limitations of traditional (deterministic) algorithms. - Introduce the concept of "guessing" or "making lucky choices" as a hypothetical computational model. - Explain that non-deterministic algorithms represent this idealized model of computation. **Development** (30 minutes) **Deterministic vs. Non-Deterministic:** - Clearly define determinism: At each step, a deterministic algorithm has a single, well-defined next action. - Introduce non-determinism: A non-deterministic algorithm can explore multiple possible paths simultaneously. **Non-Deterministic Turing Machines:** - Briefly explain the concept of a Turing Machine as a theoretical model of computation. - Describe how a non-deterministic Turing Machine can "magically" make the correct guess at each step. **NP Class:** - Define the NP class: The set of decision problems for which a given solution can be verified in polynomial time by a deterministic algorithm. - Explain that non-deterministic algorithms can solve NP problems in polynomial time. **NP-Completeness:** - Briefly introduce the concept of NP-completeness: The "hardest" problems in NP. - Explain that if one NP-complete problem can be solved in polynomial time by a deterministic algorithm, then all NP problems can.

	Exercise (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the key difference between a deterministic and a non-deterministic algorithm? - What does it mean for a problem to belong to the NP class? - What is the significance of NP-complete problems? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/Nondeterministic_algorithm** **https://en.wikipedia.org/wiki/NP-completeness** **Difference between Deterministic and Non-deterministic Algorithms** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - What is a nondeterministic algorithm? How does it differ from a deterministic algorithm? - Give an example of a problem that can be solved by a nondeterministic algorithm. - What is the role of a "nondeterministic choice" in a nondeterministic algorithm? How does it impact the algorithm's behavior? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on nondeterministic algorithm. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 33	**Course Name: Design & Analysis of Algorithms** **Topic:** Complexity classes	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of computational complexity. b. Learn about different complexity classes (P, NP, NP-Complete, NP-Hard). c. Understand the significance of the P vs. NP problem. d. Analyze the complexity of algorithms. e. Develop an intuition for the difficulty of solving different computational problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Given a list of numbers, can you find a subset that sums up to a specific target value? - Explain that some problems are easy to solve (e.g., sorting a list), while others are much harder. - Introduce the concept of computational complexity: A way to classify problems based on the resources (time and space) required to solve them. **Development** (30 minutes) **Complexity Classes:** P (Polynomial Time): o Define problems that can be solved by a deterministic Turing machine in polynomial time. o Examples: Sorting, searching, matrix multiplication. NP (Non-deterministic Polynomial Time): o Define problems for which a solution can be verified in polynomial time. o Examples: Traveling Salesman Problem, Sudoku, Satisfiability. NP-Complete: o Define problems that are both in NP and as hard as any other problem in NP. o If you can solve one NP-Complete problem efficiently, you can solve all NP problems efficiently. NP-Hard: o Define problems that are at least as hard as any NP-Complete problem. o May or may not be in NP. The P vs. NP Problem: - Explain the significance of this unsolved problem in computer

	science. - Discuss the implications if P = NP or P ≠ NP. **Analyzing Algorithm Complexity:** - Briefly introduce the concept of Big O notation (asymptotic analysis). - Discuss how to analyze the time complexity of simple algorithms (e.g., linear search, binary search). **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What does it mean for a problem to be in the class P? - What is the significance of NP-Complete problems? - What is the P vs. NP problem, and why is it important? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/P_versus_NP_problem** **https://www.geeksforgeeks.org/introduction-to-algorithms/** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - Explain the differences between the following complexity classes: P NP NP-complete NP-hard Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Complexity classes. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 33	**Course Name: Design & Analysis of Algorithms** **Topic:** Introduction to NP-Completeness, Establishing NP-Completeness of Problems, NP-Completeness Proofs,	Course No.: MCA 204

Objectives	At the end of the lesson the student shall be able to: a. Understand the concepts of decision problems, polynomial time algorithms, and the classes P and NP. b. Define NP-completeness and its significance in computer science. c. Learn the concept of polynomial-time reductions and how they are used to prove NP-completeness. d. Explore examples of NP-complete problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Can every problem that can be solved quickly (in polynomial time) also be verified quickly? - Briefly introduce the classes P and NP: - **P:** Class of problems that can be **solved** in polynomial time by a deterministic Turing Machine. - **NP:** Class of problems for which a **solution** can be **verified** in polynomial time by a deterministic Turing Machine. - Is P equal to NP? **Development** (30 minutes) **NP-Completeness:** - Define NP-hard problems: Problems that are at least as hard as any problem in NP. - Define NP-complete problems: Problems that are both NP-hard and in NP. **Polynomial-Time Reductions:** - Explain the concept of reducing one problem to another: Show that if you can solve problem A, you can also solve problem B efficiently. - Emphasize that reductions are used to prove NP-hardness. **Examples of NP-Complete Problems:** - Briefly mention some well-known NP-complete problems: - Satisfiability (SAT) - Traveling Salesman Problem (TSP) - Knapsack Problem - Vertex Cover **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students

	- What is the difference between the classes P and NP? - What is the significance of NP-complete problems? - How are polynomial-time reductions used in the context of NP-completeness? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **https://en.wikipedia.org/wiki/P_versus_NP_problem** **https://www.geeksforgeeks.org/introduction-to-np-completeness/** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - What characteristics must a problem have to be classified as NP-complete? - Explain how an NP-complete problem is both in NP and NP-hard. - Provide examples of NP-complete problems and briefly explain the significance of each. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on NP-complete problem. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 34	**Course Name: Design & Analysis of Algorithms** **Topic:** NP-Hard Problems	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of NP-Completeness. b. Learn about the significance of NP-Hard problems. c. Identify examples of common NP-Hard problems. d. Recognize the challenges and implications of dealing with NP-Hard problems.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - a brief overview of decision problems (problems with a yes/no answer). - Introduce the concept of polynomial time algorithms (algorithms that run in time bounded by a polynomial function of the input size). - Explain the class P: the set of decision problems solvable by polynomial-time algorithms. - Introduce the class NP: the set of decision problems for which a given solution can be verified in polynomial time. **Development** (30 minutes) **NP-Completeness:** - Define NP-Completeness: A problem is NP-Complete if: o It belongs to the class NP. o Every problem in NP can be reduced to it in polynomial time. - Explain the significance of NP-Completeness: If one NP-Complete problem can be solved in polynomial time, then all problems in NP can be solved in polynomial time. **NP-Hard Problems:** - Define NP-Hard problems: A problem is NP-Hard if every problem in NP can be reduced to it in polynomial time. - Note that NP-Hard problems may not necessarily belong to the class NP themselves. **Examples of NP-Hard Problems:** - Traveling Salesperson Problem (TSP): Finding the shortest possible route that visits a given set of cities and returns to the starting city. - Knapsack Problem: Given a set of items, each with a weight and a value, determine the subset of items that can be placed in a knapsack of a given capacity while maximizing the total

	value. - Satisfiability Problem (SAT): Given a Boolean formula, determine if there exists an assignment of truth values to the variables that makes the formula true. **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What does it mean for a problem to be NP-Complete? - What is the relationship between NP-Hard and NP-Complete problems? - Give one example of an NP-Hard problem and briefly describe it. Question and Answer session in between the lesson to check the presence of mind of students.
Closure	b. Summarize the Lesson Learning Outcomes and get affirmation from students on these. c. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms d. Reference website: **Wikipedia: NP-completeness** **GeeksforGeeks: NP-Complete Problems** 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 5. Homework - What does it mean for a problem to be NP-Hard? Explain the difference between NP-Hard and NP-Complete problems. - Describe the process of reducing one problem to another to show NP-Hardness. Give an example of a problem that has been proven NP-Hard through reduction. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on NP-Hard problem. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 35	**Course Name: Design & Analysis of Algorithms** **Topic:** Graphs Algorithms: Traversing Trees, Depth-First Search (DFS)	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of tree data structures. b. Learn the Depth-First Search (DFS) algorithm for traversing trees. c. Apply DFS to solve real-world problems. d. Analyze the time and space complexity of the DFS algorithm.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Briefly introduce the concept of graphs and their importance in computer science. - Define trees as a special type of graph with no cycles. - Discuss the various applications of trees, such as: - Representing hierarchical data (file systems, organizational structures) - Implementing efficient searching algorithms (binary search trees) - Representing game trees for AI **Development** (30 minutes) **Depth-First Search (DFS):** - Explain the core idea of DFS: explore as deep as possible along each branch before backtracking. - Discuss common DFS implementations: o Recursive approach o Iterative approach using a stack - Illustrate DFS with a simple tree example, step-by-step, using both recursive and iterative approaches. - Analyze the time and space complexity of DFS (usually O(V + E) where V is the number of vertices and E is the number of edges). **Applications of DFS:** - Discuss real-world applications of DFS: o Finding connected components in a graph o Topological sorting o Solving maze problems o Detecting cycles in a graph - **DFS Variations (Optional):** -Briefly touch upon variations of DFS, such as: o DFS with backtracking for solving constraint satisfaction problems

	Exercise (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the core principle of Depth-First Search? - Which data structure is typically used to implement DFS iteratively? - What is the time complexity of DFS in most cases? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: GeeksforGeeks: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/ Wikipedia: https://en.wikipedia.org/wiki/Depth-first_search 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 4. Homework - Write the pseudocode for the Depth-First Search (DFS) algorithm. How does DFS traverse a graph, and what data structures does it rely on (e.g., stack, recursion)? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Depth First Search. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 36	**Course Name: Design & Analysis of Algorithms** **Topic:** Graphs Algorithms: Traversing Trees, Breadth-First Search (BFS)	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of graph traversal algorithms. b. Learn the Breadth-First Search (BFS) algorithm for exploring graphs. c. Implement BFS using a queue data structure. d. Analyze the time and space complexity of BFS. e. Apply BFS to solve real-world problems like finding shortest paths in unweighted graphs.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - a brief overview of graph data structures, emphasizing their importance in representing various real-world scenarios like social networks, transportation systems, and computer networks. - Introduce the concept of graph traversal as the process of systematically visiting all the vertices in a graph. - Briefly mention other graph traversal algorithms like Depth-First Search (DFS) and highlight the key differences between BFS and DFS. **Development** (30 minutes) **BFS Algorithm:** - Explain the core idea of BFS: Explore all the neighbors of the current vertex before moving to the next level of the graph. - Illustrate the BFS algorithm using a simple, step-by-step example on a small graph. Use a diagram to visualize the order in which vertices are visited. - Discuss the implementation of BFS using a queue data structure: o Enqueue the starting vertex. o Dequeue a vertex from the front of the queue. o Explore all its unvisited neighbors and enqueue them. o Repeat until the queue is empty. **Time and Space Complexity:** - Analyze the time complexity of BFS: O(V + E) where V is the number of vertices and E is the number of edges in the graph. - Discuss the space complexity of BFS: O(V) in the worst case, as all vertices may need to be stored in the queue. **Applications of BFS:**

	- Discuss real-world applications of BFS: o Finding the shortest path in an unweighted graph. o Finding connected components in a graph. o Breadth-first search in a tree. o Level order traversal of a tree. o Social network analysis (finding connections between people). **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students - What is the core principle behind the BFS algorithm? - Which data structure is essential for implementing BFS? - What is the time complexity of BFS in a graph with V vertices and E edges? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/** **Wikipedia:** https://en.wikipedia.org/wiki/Breadth-first_search Breadth First Search (BFS) Algorithm 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 4. Homework - Write the pseudocode for the Breadth-First Search (BFS) algorithm. How does BFS traverse a graph, and what data structures does it rely on (e.g., queue)? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on Breadth First Search. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 36	**Course Name: Design & Analysis of Algorithms** **Topic:** Graphs Algorithms: Traversing Trees, Best-First Search (BFS)	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of informed search algorithms. b. Learn the Best-First Search algorithm and its core principle: using a heuristic function to guide the search. c. Analyze the time and space complexity of Best-First Search. d. Apply Best-First Search to solve pathfinding problems, such as finding the shortest path in a weighted graph.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - Briefly review the concept of graph traversal algorithms, particularly Depth-First Search (DFS) and Breadth-First Search (BFS). - Discuss the limitations of uninformed search algorithms like DFS and BFS: - DFS can get stuck in deep branches and may not find the optimal solution. - BFS explores all nodes at a given level, which can be inefficient for large graphs. - Introduce the idea of informed search algorithms, which use additional information to guide the search process. **Development** (30 minutes) **Best-First Search (BFS):** - Explain the core principle of Best-First Search: o Use a priority queue to order nodes based on a heuristic function. o The heuristic function estimates the "cost" or "distance" to the goal. - Discuss common heuristic functions: o Manhattan distance (for grid-based problems) o Euclidean distance (for grid-based problems) o *A search** (a special case of Best-First Search that combines heuristic and path cost) - Illustrate Best-First Search with a simple graph example, highlighting how the heuristic guides the search process. - Analyze the time and space complexity of Best-First Search (highly dependent on the heuristic function and the graph structure). **Applications of Best-First Search:**

	- Discuss real-world applications of Best-First Search:
	  o Pathfinding in games (e.g., finding the shortest path for a character)
	  o Robotics (e.g., motion planning)
	  o AI planning (e.g., searching for optimal solutions in game trees)
	Limitations of Best-First Search:
	- Discuss potential limitations of Best-First Search:
	  o The quality of the search heavily relies on the accuracy of the heuristic function.
	  o Can get stuck in local optima if the heuristic is not well-defined.
	Exercise (10 minutes) –
	- Group Discussion at the end of the session.
	- Ask students
	- What is the key difference between Best-First Search and other graph traversal algorithms like BFS?
	- What data structure is typically used to implement Best-First Search?
	- Can you name one example of a heuristic function used in pathfinding problems?
	Question and Answer session in between the lesson to check the presence of mind of students.

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: **Best First Search (Informed Search) - GeeksforGeeks** **Wikipedia:https://en.wikipedia.org/wiki/Best-first_search** Best First Search in Artificial Intelligence - Javatpoint 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 4. Homework - Explain the difference between Best-First Search (BFS) and Depth-First Search (DFS) in terms of algorithm behavior, performance, and typical use cases. Provide examples of problems where each algorithm would be preferred. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss.

	- Quiz on Best First Search. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 37	**Course Name: Design & Analysis of Algorithms** **Topic:** Topological Sort	**Course No.: MCA 204**

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of directed acyclic graphs (DAGs). b. Learn the definition and applications of topological sort. c. Implement topological sort using Depth-First Search (DFS). d. Analyze the time and space complexity of topological sort.
Teaching Aids (if any)	a. Power point presentation b. Use of Whiteboard
Teaching Development	- **Introduction** (5 minutes) - Ask questions. - introduce the concept of directed graphs. - Define acyclic graphs as graphs without cycles. - Discuss the importance of topological sort in various applications, such as: - Task scheduling (e.g., building construction, software compilation) - Course scheduling (determining the order in which courses must be taken) - Data serialization **Development** (30 minutes) **Directed Acyclic Graphs (DAGs):** - Explain the properties of DAGs. - Illustrate DAGs with real-world examples. **Topological Sort:** - Define topological sort as a linear ordering of vertices in a DAG such that if there is an edge from vertex 'u' to vertex 'v', then 'u' comes before 'v' in the ordering. - Explain that topological sort is not possible for graphs with cycles. - **Topological Sort Algorithm:** - Describe the DFS-based algorithm for topological sort: o Perform DFS on the graph. o Maintain a stack to store the vertices in the order they are finished (post-order traversal). o Pop vertices from the stack to obtain the topological order. - Illustrate the algorithm with a simple DAG example. - Analyze the time and space complexity of the algorithm (usually O(V + E) where V is the number of vertices and E is the number of edges). **Exercise** (10 minutes) – - Group Discussion at the end of the session. - Ask students

	- What is a DAG? - Is it possible to perform topological sort on a graph with a cycle? - Which graph traversal algorithm is commonly used to implement topological sort? Question and Answer session in between the lesson to check the presence of mind of students.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested NPTEL video lecture NPTEL :: Computer Science and Engineering - NOC:Design and Analysis of Algorithms 3. Reference website: GeeksforGeeks:https://www.geeksforgeeks.org/topological-sorting/ Wikipedia: https://en.wikipedia.org/wiki/Topological_sorting 4. Suggested Reading books: - Introduction to Algorithms by T.Cormen, C. Lieserson, R.Rivest, C.Stein . - Algorithms by S. Dasgupta, C. Papadimitriou, Umesh Vazirani - Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni 4. Homework - What is the purpose of topological sorting in directed acyclic graphs (DAGs), and why is it not applicable for graphs with cycles? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, Why, Who?). Allow students to answer and discuss. - Quiz on topological sorting. Spend 5 minutes to evaluate student assimilation of the lesson contents