Pimpri Chinchwad Education Trust's
Pimpri Chinchwad College of Engineering An
Autonomous Institute
(Affiliated to Savitribai Phule Pune University)

Question Bank :Design and Analysis of Algorithm [BCER5411]

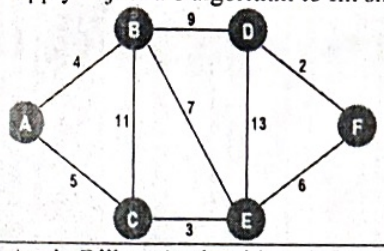
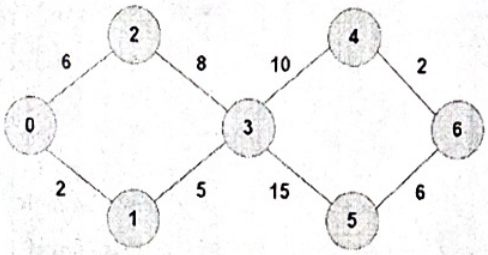
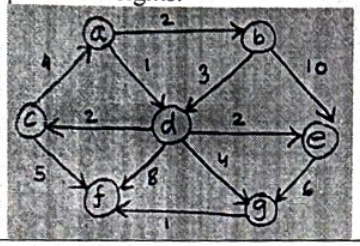
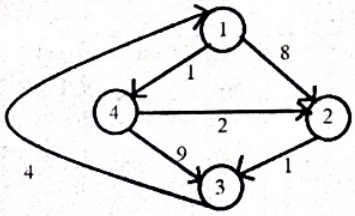
Sr. No.	Question	Bloom's Level	CO	Marks
	Unit I			
1	Explain the Amortized complexity for inserting array element in dynamic array	Understand (L2)	CO2	3
2	Describe the aggregate method in Amortized Analysis with an example	Understand (L2)	CO2	3
3	Apply the accounting method to get the amortized complexity for push and pop operations in a stack	Apply (L3)	CO2	3
4	Explain the concept of Amortized Analysis in algorithm design	Understand (L2)	CO2	3
5	Describe different techniques used in Amortized Analysis	Understand (L2)	CO2	3
6	Show Amortized analysis using Stack operations	Apply (L3)	CO2	3
7	Explain the Amortized complexity for 4 bit binary incrementor from 0 to 8	Understand (L2)	CO2	3
8	Define asymptotic notation and explain its significance in algorithm analysis.	Remember (L1)	CO1	3
9	Describe best-case, worst-case, and average-case time complexities with examples.	Understand (L2)	CO1	3
10	Explain the significance of time complexity in evaluating algorithms.	Understand (L2)	CO1	3
11	Differentiate between lower bound and upper bound in algorithm analysis.	Understand (L2)	CO1	5
12	Identify various techniques for proving the correctness of algorithms.	Remember (L1)	CO1	5
13	Explain the process of solving a recurrence equation using Master Method.	Understand (L2)	CO1, CO4	5
14	List different types of asymptotic notations and their use cases.	Remember (L1)	CO1	7
15	Classify the time complexities of various algorithms as linear, logarithmic, quadratic, etc., with examples.	Understand (L2)	CO1, CO6	7
16	Identify different cases of algorithm analysis and give examples for each.	Remember (L1)	CO1	7
17	Illustrate how asymptotic analysis helps in comparing the performance of algorithms.	Understand (L2)	CO1	7
18	Describe the process of formulating and solving recurrence relations using the Master Theorem.	Understand (L2)	CO1, CO4	7
19	Describe different methods to solve recurrence relations.	Remember (L1)	CO1	5
20	Define Big-O, Big-Ω, and Big-Θ notations and provide examples.	Remember (L1)	CO1	8
21	Explain the role of asymptotic notation in the analysis of algorithms.	Understand (L2)	CO1	5
22	Explain the Master Theorem and its applications in solving recurrence relations.	Understand (L2)	CO1, CO4	8
23	Illustrate how worst-case time complexity affects the choice of an algorithm in a real-time system.	Understand (L2)	CO1	8
24	Apply Master theorem to fin complexity of following recurrence relation $T(n) = 4T(n/2) + n^2$	Apply (L3)	CO4, CO5	7
25	Formulate the recurrence relation for Merge Sort and solve it using the Master Theorem.	Apply (L3)	CO1, CO4	7
26	Design an algorithm for binary search and Analyze its time complexity.	Apply (L3)	CO3, CO5	8
27	Compare time complexities of different algorithms with different types methods, also give example of master method	Analyze (L4)	CO1	8
28	Apply Master theorem to find complexity of following recurrence relation	Analyze (L4)	CO1	7

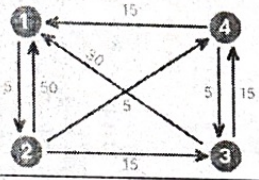
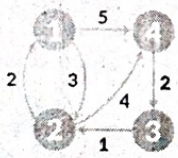
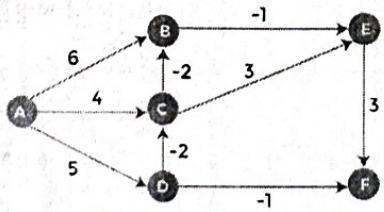
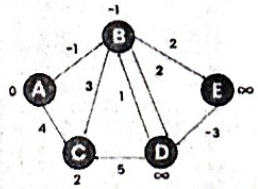
	$T(n) = 2T(n/2) + n \log n$			
29	Evaluate complexity of following recurrence relation using master theorem $T(n) = T(n/2) + 2^n$	Evaluate (L5)	CO1, CO3	7
30	Prove that the worst-case time complexity of Quick Sort is O(n^2).	Analyze (L4)	CO1, CO4	8
31	Analyze the importance of understanding algorithm complexities for real-time applications.	Analyze (L4)	CO1, CO3	8
32	Apply master theorem to find complexity of given recurrence relation $T(n) = 6T(n/3) + n^2 \log n$	Analyze (L4)	CO3, CO5	8
33	Construct recurrence equations for Merge Sort	Apply (L3)	CO1, CO3	7
34	Describe the process of solving recurrence relations	Remember (L1)	CO1	3
35	Classify algorithms based on their time complexity and give examples for each type.	Understand (L2)	CO1, CO6	7
36	Make use of Master theorem to find complexity of given recurrence relation $T(n) = 3T(n/3) + \sqrt{n}$	Apply (L3)	CO3, CO5	7
37	Make use of Master theorem to find complexity of given recurrence relation. $T(n) = 3T(n/3) + n/2$	Apply (L3)	CO4, CO5	7
38	Explain the impact of Big-O notation on software performance optimization.	Understand (L2)	CO1	5
39	Discuss the importance of choosing the right algorithm for specific types of problems.	Understand (L2)	CO1, CO3	3
40	Describe the significance of average-case analysis in comparing algorithm performance.	Understand (L2)	CO1	3
41	Apply Master Theorem to find complexity of Binary Search by using recurrence relation	Understand (L2)	CO1	8
42	Compare the efficiency of algorithms for solving the same problem using different strategies.	Analyze (L4)	CO1, CO3	8
43	Build recurrence equations using Master Theorem for Fibonacci series.	Apply (L3)	CO3, CO5	8
44	Illustrate the use of dynamic programming in solving complex computational problems.	Understand (L2)	CO1, CO3	8
45	Explain how the choice of data structures impacts the performance of algorithms.	Understand (L2)	CO1	7
46	Describe correctness of algorithm along with example.	Understand (L2)	CO3, CO5	5
47	Compare the performance of divide-and-conquer algorithms versus dynamic programming algorithms along with example	Analyze (L4)	CO1, CO3	8
48	Explain the concept of amortized analysis and its application in algorithms.	Understand (L2)	CO1	7
49	Analyze the trade-offs between time complexity and space complexity in algorithm design.	Analyze (L4)	CO1, CO3	5
50	Compare time complexity and amortize analysis along with examples How both techniques are differ.	Analyze (L4)	CO3, CO5	8

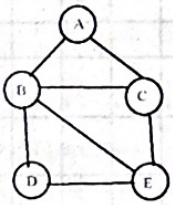
Unit II

Sr. No.	Question	Bloom's Level	CO Mapped	
1	Use the divide and conquer approach to solve Maximum sub array for given array A = [−2,1, −3,4, −1,2,1, −5,4]	Apply (L3)	CO1	3
2	Apply Master Theorem to get time complexity of Quick sort	Apply (L3)	CO1	3
3	Consider a situation where divide and conquer is not the most efficient approach. Provide an example	Apply (L3)	CO1	3
4	Compare time complexity of Quick sort and Merge sort	Understand (L2)	CO1	3
5	Apply Master Theorem to get time complexity of Merge sort	Apply (L3)	CO1	3
6	What is control abstraction for greedy approach and time analysis of control abstraction	Understand (L2)	CO1	3
7	Show how Master Theorem is useful to solve recurrence relation	Apply (L3)	CO1	3
8	Illustrate Integer arithmetic in Divide and Concur with example	Apply (L3)	CO1	3
9	Define Divide and Conquer strategy and explain its significance with examples.	Remember (L1)	CO1	7
10	Build recurrence relation for Merge Sort and apply master theorem to find complexity of algorithm.	Apply (3)	CO1	8

11	Explain the principle of the Greedy Strategy with examples.	Understand (L2)	CO1	3
12	What is the time complexity of the Quick Sort algorithm?	Remember (L1)	CO1	3
13	Describe the Binary Search algorithm and its efficiency.	Understand (L2)	CO1	6
14	Compare the time complexities of Merge Sort and Quick Sort algorithms.	Understand (L2)	CO1	7
15	Explain the concept of "Divide and Conquer" with the example of the Maximum Sub-array problem.	Understand (L2)	CO1	7
16	Discuss the advantages and disadvantages of using Greedy Algorithms.	Understand (L2)	CO1	3
17	Describe how the Knapsack problem can be solved using a Greedy approach.	Understand (L2)	CO1	7
18	Apply divide and conquer strategy to find maximum sub array for given elements {-2, -5, 6, -2, -3, 1, 5, -6}	Apply (L3)	CO4	7
19	Apply quick sort to sort given elements of array {4, 9, 6, 8, 3, 1, 5,7}	Apply (L3)	CO4	7
20	Illustrate Greedy Strategy to solve the Job Scheduling problem.	Apply (L3)	CO3	7
21	Analyze the time complexity of Binary Search.	Analyze (L4)	CO1	7
22	Explain how the Divide and Conquer approach can be used to improve the efficiency of an algorithm along with example.	Understand (L2)	CO1	8
23	Prove that Quick Sort has an average-case time complexity of O(n log n).	Analyze (L4)	CO1	8
24	Apply Quick sort to sort given elements of array 44 33 11 55 77 90 40 60 99 22 88	Apply (L3)	CO3	7
25	Apply Merge sort to sort given elements of array {4, 9, 6, 8, 3, 1, 5,7}	Apply (L3)	CO4	7
26	Compare and contrast the Greedy approach with the Dynamic Programming approach for solving optimization problems.	Analyze (L4)	CO6	8
27	Discuss the role of the 'Divide and Conquer' strategy in the analysis of Merge Sort along with example.	Understand (L2)	CO1	7
28	Analyze the time complexity of the Merge Sort algorithm in best, worst, and average cases.	Analyze (L4)	CO1	8
29	Apply binary search to search 40 in given elements of array 44 33 11 55 77 90 40 60 99 22 88	Apply (L3)	CO3	8
30	Determine the time complexity of the Quick Sort algorithm in the worst-case scenario.	Analyze (L4)	CO1	8
31	Apply fractional knapsack to find maximum profit for given items consider weight of knapsack is 90 Items: 1 2 3 4 5 Profit: 90 130 60 110 42 Weight: 20 60 20 50 15	Analyze (L4)	CO1	7
32	Build recursive algorithm for solving the Maximum Sub-array problem using Divide and Conquer.	Apply (L3)	CO3	8
33	Analyze the time complexity of binary search.	Analyze (L4)	CO1	7
34	Apply Quick sort to sort given elements of array 44 33 11 55 77 90 40 60 99 22 88	Apply (L3)	CO3	7
35	Discuss the use of the Master Theorem in solving complex recurrence relations along with example.	Understand (L2)	CO1	8
36	For the given set of items and the knapsack capacity of 10 kg, Analyze the subset of the items to be added in the knapsack such that the profit is maximum. Items: 1 2 3 4 5 Weights (in kg): 3 3 2 5 1 Profits: 10 15 10 12 8	Analyze (L4)	CO3	8
37	Analyze the effectiveness of Greedy algorithms for solving Dijkstra's algorithm	Analyze (L4)	CO6	7
38	Design a Greedy algorithm for solving the Fractional Knapsack problem with a capacity constraint.	Creating	CO3	7
39	Analyze the complexity of the Integer Arithmetic problem using Divide and Conquer approach.	Analyze (L4)	CO6	8

40	Explain how Divide and Conquer can be applied to solve complex optimization problems.	Understand (L2)	CO1	7
41	Analyze the performance of the Greedy approach for the Job Scheduling problem with different types of jobs.	Analyze (L4)	CO1	7
42	Apply Dijkstra's algorithm to fin shortest distance 	Apply (L3)	CO3	7
43	Apply Dijkstra's algorithm to fin shortest distance 	Apply (L3)	CO3	7
44	Explain how the Master Theorem helps in Analyze the time complexity of recursive algorithms.	Understand (L2)	CO1	7
45	Make a use of Greedy algorithm to solve the Single Source Shortest Path problem with positive weights. 	Apply (L3)	CO3	7
	Unit III			
1	Using dynamic programming to determine if there exists a subset of the set [6,1,8,3,5,2] and a target sum of 10	Apply (L3)	CO4	7
2	Explain how does a dynamic programming knapsack problem is differed from greed knapsack problem	Understand (L2)	CO4	7
3	Use Floyd-Warshall Algorithm to get the shortest path distance between pair of vertices? 	Apply (L3)	CO4	7
4	Construct an optimal binary search tree to minimize the expected search time (a1 a2 a3) = (read, write, delete) p(1: 3) = (5,2,4) q(0: 3) = (1,3,2,4).	Apply (L3)	CO4	7
5	Use dynamic programming to get the optimal solution for the sum of subset problems for the set [3, 34, 4, 12, 5, 2] and a target sum of 9	Apply (L3)	CO4	7
6	Illustrate Bellman-Ford algorithm along with example	Apply (L3)	CO4	7
7	Construct an optimal binary search tree to minimize the expected search time (a1 a2 a3) = (do, if, while) p(1: 3) = (3, 3 , 1) q(0: 3) = (2, 3, 1, 1).	Apply (L3)	CO4	7
8	Use dynamic programming to get the optimal solution for the sum of subset problems for the set [1,2,3,4,5] and a target sum of 7	Apply (L3)	CO4	7
9	What is the 0/1 knapsack problem? How does a dynamic programming method gives optimal solutions as compared to greedy method?	Understand (L2)	CO4	7

10	Illustrate Floyd-Warshall algorithm along with example	Apply (L3)	CO4	7
11	Construct an optimal binary search tree to minimize the expected search time. Keys (a1 a2 a3) = (search, select, update) p(1: 3) = (4, 2 , 3) q(0: 3) = (1, 2, 2, 3).	Apply (L3)	CO4	7
12	Define Dynamic Programming and explain its significance with examples.	Remember (L1)	CO1	7
13	Describe the principle of Dynamic Programming with examples of its application.	Understand (L2)	CO1	7
14	Explain the concept of optimal substructure in Dynamic Programming.	Understand (L2)	CO1	7
15	What is the time complexity of the Dynamic Programming approach for the 0/1 Knapsack problem?	Understand (L2)	CO1	7
16	Describe the difference between Dynamic Programming and Greedy algorithms with example.	Understand (L2)	CO1	8
17	Explain how to solve the All-Pairs Shortest Path problem using Dynamic Programming.	Understand (L2)	CO1	8
18	Describe how the Floyd-Warshall algorithm solves the All-Pairs Shortest Path problem with example.	Understand (L2)	CO1	7
19	Apply Floyd-Warshall algorithm to find all pair shortest path. 	Apply (L3)	CO3	7
20	Apply Floyd-Warshall algorithm solves given problem 	Apply (L3)	CO3	7
21	Explain the Bellman-Ford algorithm for finding shortest paths in a graph along with example.	Understand (L2)	CO1	5
22	Solve a simple Dynamic Programming problem, such as the Fibonacci sequence, and discuss its time complexity.	Apply (L3)	CO4	7
23	Analyze the time complexity of the Floyd-Warshall algorithm for All-Pairs Shortest Path.	Analyze (L4)	CO1	8
24	Make use of Bellman ford algorithm to find single source shortest path for given graph 	Apply (L3)	CO3	8
25	Make use of Bellman ford algorithm to find single source shortest path for given graph 	Apply (L3)	CO3	8
26	Compare Dynamic Programming with Divide and Conquer in terms of problem-solving approach.	Analyze (L4)	CO6	8
27	Analyze the space complexity of Dynamic Programming algorithms with examples.	Analyze (L4)	CO1	8
28	Describe how Dynamic Programming can be used to solve complex optimization problems.	Understand (L2)	CO1	7
29	Explain how Dynamic Programming can be used for the Sum of Subsets problem.	Understand (L2)	CO1	7

30	Compare the efficiency of Dynamic Programming algorithms to Greedy algorithms for the 0/1 Knapsack problem.	Analyze (L4)	CO6	8
31	Explain how Dynamic Programming can solve the Optimal Binary Search Tree (OBST) problem.	Understand (L2)	CO1	8
32	Explain a Dynamic Programming approach for the Subset Sum problem.	Analyze (L4)	CO3	7
33	Justify how Dynamic Programming can be applied to solve problems with multiple constraints.	Evaluate (L5)	CO1	7
34	Outline a Dynamic Programming solution for the Knapsack problem	Analyze (L4)	CO3	7
35	Compare and contrast Dynamic Programming and Divide and Conquer approaches for solving optimization problems.	Analyze (L4)	CO6	8
36	Analyze the time complexity of Dynamic Programming algorithms using specific examples.	Analyze (L4)	CO1	7
37	Explain how the principles of Dynamic Programming can be applied to improve algorithm efficiency.	Analyze (L4)	CO1	7
	Unit IV			
1	Describe how backtracking is used to find Hamiltonian cycle in a graph	Understand (L2)		3
2	Use the backtracking strategy to colour a graph with 5 vertices (A, B, C, D, E) using three colours (1, 2, 3) such that no two adjacent vertices have the same colour 	Apply (L3)	CO6	3
3	Describe how backtracking is used to solve the N-Queens problem	Understand (L2)	CO6	3
4	Solve the sum of subsets problem using backtracking to find all subsets of the set {2, 4, 6, 10} that sum to 12	Apply (L3)	CO6	3
5	Using backtracking, show how you would assign colors to the vertices of a graph such that no two adjacent vertices share the same color. Consider 5 vertices and 3 colors	Apply (L3)	CO6	3
6	Describe how backtracking is used to solve the N-Queens problem	Understand (L2)	CO6	3
7	Apply the sum of subsets problem using backtracking to find all subsets of the set {1,2,3,4} that sum to 6	Apply (L3)	CO6	3
8	Describe the pseudocode to find a Hamiltonian cycle using Backtracking	Understand (L2)	CO6	3
9	Define the principle of Backtracking.	Remember (L1)	CO1	3
10	Explain the 8-Queen problem in backtracking.	Understand (L2)	CO3	3
11	Recall the time complexity of the graph coloring problem.	Remember (L1)	CO1	3
12	Illustrate the control abstraction in backtracking.	Understand (L2)	CO2	3
13	Describe the sum of subsets problem.	Understand (L2)	CO1	3
14	Explain how pruning helps in backtracking.	Understand (L2)	CO2	3
15	Describe a real-life scenario where backtracking could be applied.	Understand (L2)	CO3	3
16	Explain the backtracking control abstraction for a general search problem.	Understand (L2)	CO2	3
17	Solve the 8-Queen problem for a 4x4 chessboard.	Apply (L3)	CO3	7
18	Demonstrate backtracking with the graph coloring problem.	Apply (L3)	CO3	7
19	Apply backtracking to solve the sum of subsets problem.	Apply (L3)	CO4	7
20	Solve a Sudoku puzzle using a backtracking approach.	Apply (L3)	CO4	7
21	Use backtracking to find a Hamiltonian cycle in a given graph.	Apply (L3)	CO3	7
22	Implement backtracking to find all possible solutions to a N-Queen problem.	Apply (L3)	CO4	7
23	Discuss the control abstraction for backtracking.	Apply (L3)	CO1	7
24	Apply backtracking to solve the Knight's Tour problem.	Apply (L3)	CO4	7
25	Solve the m-coloring problem using a backtracking algorithm.	Apply (L3)	CO4	7
26	Discuss the time complexity of backtracking for solving the graph coloring problem.	Apply (L3)	CO2	7
27	Illustrate the process of solving a subset-sum problem using backtracking.	Apply (L3)	CO3	8
28	Write a pseudocode for solving the 8-Queen problem using backtracking.	Apply (L3)	CO4	7

29	Apply backtracking to solve a knapsack problem (with integer weights).	Apply (L3)	CO4	7
30	Use backtracking to enumerate all possible knight moves in a given chessboard.	Apply (L3)	CO4	8
31	Use backtracking to solve a combinatorial optimization problem.	Apply (L3)	CO3	7
32	Illustrate backtracking for solving the word search puzzle.	Apply (L3)	CO4	8
33	Solve a Hamiltonian cycle problem with backtracking for a small graph.	Apply (L3)	CO4	7
34	Analyze the advantages of applying backtracking in optimization problems.	Apply (L3)	CO1	7
35	Analyze the sum of subsets problem using backtracking.	Analyze (L4)	CO4	8
36	Compare the time complexities of the 8-Queen and graph coloring problems.	Analyze (L4)	CO1	8
37	Evaluate the time complexity of the 8-Queen problem.	Analyze (L4)	CO2	8
38	Compare backtracking and dynamic programming for solving optimization problems.	Analyze (L4)	CO2	8
39	Evaluate the effectiveness of backtracking vs. branch-and-bound for the knapsack problem.	Evaluate	CO5	8
40	Analyze how backtracking performs for solving the Hamiltonian path problem.	Analyze (L4)	CO4	8
1	Explain how Least Cost Branch and Bound strategy helps to give optimal solution in the Branch and Bound algorithm	Understand (L2)	CO5	3
2	Apply the Branch and Bound method to solve followig 0/1 Knapsack problem Item 1: Value = 60, Weight = 10 Item 2: Value = 100, Weight = 20 Item 3: Value = 120, Weight = 30 Knapsack Capacity: 50	Apply (L3)	CO5	3
3	Apply Least Cost Branch and Bound to find a solution to a Traveling Salesman Problem (TSP) of a given matrix A B C A 0 10 15 B 10 0 35 C 15 35 0	Apply (L3)	CO5	3
4	Explain various techniques used for branch and bound strategies to get solution to the problems	Understand (L2)	CO5	3
5	Describe how Branch and Bound is applied to solve Travelling salesperson Problem	Understand (L2)	CO5	3
6	Apply LC Branch and Bound to find a solution to a Traveling Salesman Problem (TSP) of a given matrix A B C A 0 5 10 B 5 0 15 C 10 15 0	Apply (L3)	CO5	3
7	Explain time analysis of control abstraction for branch and bound strategy	Understand (L2)	CO5	3
8	Apply the Branch and Bound method to solve followig 0/1 Knapsack problem Item 1: Value = 60, Weight = 10 Item 2: Value = 100, Weight = 20 Item 3: Value = 120, Weight = 30 Knapsack Capacity: 50	Apply (L3)	CO5	3
9	Apply LC Branch and Bound to find a solution to a Traveling Salesman Problem (TSP) of a given matrix A B C A 0 4 8 B 4 0 6 C 8 6 0	Apply (L3)	CO5	3
10	Define the principle of Branch and Bound.35	Remember (L1)	CO1	3
11	List the key features of the Branch and Boun3d algorithm.	Remember (L1)	CO2	3
12	State two differences between Branch and Bound and Backtracking.	Remember (L1)	CO1	3
13	Recall the general structure of the Branch and Bound algorithm.	Remember (L1)	CO2	3
14	Explain the role of bounding functions in Branch and Bound.	Understand (L2)	CO3	3
15	Describe the knapsack problem in the context of Branch and Bound.	Understand (L2)	CO4	3
16	What is the purpose of the Branch and Bound method in solving optimization problems?	Remember (L1)	CO1	3
17	Identify the primary advantage of Branch and Bound over brute-force methods.	Understand (L2)	CO2	3

18	Define the term "feasible solution" in the context of Branch and Bound.	Remember (L1)	CO1	5
19	Recall how the Branch and Bound algorithm explores the solution space.	Remember (L1)	CO1	5
20	Solve a 0/1 knapsack problem using Branch and Bound.	Apply (L3)	CO3	7
21	Implement Branch and Bound for the traveling salesman problem (TSP).	Apply (L3)	CO4	7
22	Solve the 0/1 knapsack problem with 4 items using the Branch and Bound method.	Apply (L3)	CO4	7
23	Write the pseudocode for Branch and Bound applied to the knapsack problem.	Apply (L3)	CO4	7
24	Implement Branch and Bound to solve the Maximum Clique problem.	Apply (L3)	CO4	7
25	Solve the assignment problem using the Branch and Bound strategy.	Apply (L3)	CO4	7
26	Show how Branch and Bound can optimize resource allocation problems.	Apply (L3)	CO3	7
27	Solve the knapsack problem where the profit and weights are fractional using Branch and Bound.	Apply (L3)	CO4	7
28	Apply Branch and Bound to solve a shortest path problem in a weighted graph.	Apply (L3)	CO4	7
29	Explain the bounding function used in the TSP when applying Branch and Bound.	Apply (L3)	CO4	7
30	Write a pseudocode for Branch and Bound for TSP problem.	Apply (L3)	CO4	7
31	Analyze the computational complexity of Branch and Bound in a TSP.	Apply (L3)	CO2	7
32	Analyze the time complexity of the Branch and Bound method in solving TSP.	Analyze (L4)	CO4	8
33	Compare Branch and Bound with dynamic programming in solving optimization problems.	Analyze (L4)	CO2	8
34	Evaluate the performance of Branch and Bound in solving the knapsack problem.	Evaluate	CO4	8
35	Compare the efficiency of Branch and Bound versus Backtracking for TSP.	Analyze (L4)	CO3	8
36	Evaluate the time complexity of Branch and Bound for different types of knapsack problems.	Evaluate	CO4	8
37	Analyze how Branch and Bound performs on real-world scheduling problems.	Analyze (L4)	CO4	8

Unit V

1	Elaborate polynomial and non-polynomial class problems along with example. (CO3)	Understand (L2)	CO6	7
2	Prove Vertex cover problem is NP Complete. (CO3)	Analyze (L4)	CO6	7
3	Show 3 SAT Problem is NP Complete. (CO3)	Apply (L3)	CO6	7
4	Explain deterministic and non-deterministic algorithms along with example (CO3)	Understand (L2)	CO6	7
5	Prove importance of P verses NP problems. (CO3)	Analyze (L4)	CO6	7
6	Show the relation between P , NP class,NP-Hard & NP-Complete problems. (CO3)	Apply (L3)	CO6	7
7	Discuss Clique Problem and Prove Clique problem is NP Complete. (CO3)	Analyze (L4)	CO6	7
8	Elaborate deterministic and non-deterministic algorithms with example (CO3)	Understand (L2)	CO6	7
9	Discuss NP problem along with example (CO3)	Understand (L2)	CO6	7
10	Prove how all NP problems are not NP complete along with example. (CO3)	Analyze (L4)	CO6	7
11	Define P and NP in the context of complexity theory.	Remember (L1)	CO1	7
12	Explain the concept of NP-completeness.	Understand (L2)	CO2	7
13	Recall the definition of polynomial time.	Remember (L1)	CO1	7
14	Define a reduction in complexity theory.	Remember (L1)	CO1	7
15	Explain what is meant by the class NP-Hard.	Understand (L2)	CO1	7
16	State the difference between P and NP problems.	Remember (L1)	CO2	7
17	Recall the Cook-Levin theorem.	Remember (L1)	CO1	7
18	Describe the significance of NP-complete problems.	Understand (L2)	CO3	7
19	State an example of an NP-complete problem.	Remember (L1)	CO2	7
20	Explain what is meant by a deterministic Turing machine.	Understand (L2)	CO1	7
21	Apply reduction techniques to prove that a problem is NP-complete.	Apply (L3)	CO4	7
22	Solve a problem using the concept of polynomial time reduction.	Apply (L3)	CO3	7
23	Apply the Cook-Levin theorem to demonstrate NP-completeness.	Apply (L3)	CO4	7
24	Demonstrate how a problem can be transformed into a known NP-complete problem.	Apply (L3)	CO4	7
25	Solve a graph-based problem to demonstrate NP-hardness.	Apply (L3)	CO3	7
26	Use reduction techniques to show that a certain problem belongs to class NP.	Apply (L3)	CO4	7
27	Implement a heuristic algorithm for an NP-complete problem and discuss its efficiency.	Apply (L3)	CO4	7
28	Apply approximation algorithms to solve an NP-hard problem and compare the results.	Apply (L3)	CO3	7
29	Solve the vertex cover problem using approximation and discuss the complexity.	Apply (L3)	CO3	7
30	Apply complexity analysis to determine the feasibility of solving an NP-hard problem in	Apply (L3)	CO4	7

	polynomial time.			
31	Apply the concept of a certificate to verify the correctness of an NP problem.	Apply (L3)	CO3	7
32	Solve a traveling salesman problem using an approximation algorithm and analyze its performance.	Apply (L3)	CO4	7
33	Demonstrate how a reduction can simplify a graph problem in complexity theory.	Apply (L3)	CO3	7
34	Use the concept of computational complexity to analyze a real-world problem.	Apply (L3)	CO4	7
35	Apply time complexity analysis to an NP-complete problem and evaluate its computational feasibility.	Apply (L3)	CO3	7
36	Show how a known NP-complete problem can be used to prove the NP-completeness of a new problem.	Apply (L3)	CO4	7
37	Apply reductions to show how problems in NP can be transformed into each other.	Apply (L3)	CO3	7
38	Implement an approximation algorithm for the bin packing problem and discuss its efficiency.	Apply (L3)	CO4	7
39	Demonstrate the use of reduction to transform a decision problem into another.	Apply (L3)	CO4	7
40	Apply the concept of complexity classes to determine the hardness of a combinatorial optimization problem.	Apply (L3)	CO3	7
41	Use reduction techniques to classify a scheduling problem as NP-hard.	Apply (L3)	CO4	7
42	Analyze the time complexity of a problem in both deterministic and non-deterministic models.	Analyze (L4)	CO4	7
43	Compare the space and time complexity of solving NP-complete problems using brute force and approximation algorithms.	Analyze (L4)	CO3	7
44	Analyze the significance of the P vs. NP question in complexity theory.	Analyze (L4)	CO4	7
45	Analyze how the complexity of problems in NP influences real-world computational challenges.	Analyze (L4)	CO4	7
46	Compare the performance of exact algorithms and heuristic methods for solving NP-hard problems.	Analyze (L4)	CO4	7
47	Analyze the trade-offs between space and time complexity in solving NP-hard problems using various techniques.	Analyze (L4)	CO4	7
48	Compare the efficiency of dynamic programming with greedy algorithms in solving specific NP-hard problems.	Analyze (L4)	CO3	7
49	Analyze the limitations of existing approximation algorithms for NP-hard problems and propose enhancements.	Analyze (L4)	CO5	7
50	Evaluate the performance of Branch and Bound methods in solving NP-hard problems and discuss its scalability.	Evaluate (L5)	CO4	7
51	Analyze the effect of various heuristics on the time complexity of solving NP-complete problems.	Analyze (L4)	CO4	7
52	Analyze the trade-offs involved in using approximation algorithms versus exact algorithms for NP-hard problems.	Analyze (L4)	CO4	7
53	Analyze the potential of bio-inspired algorithms, such as genetic algorithms, in solving NP-complete problems.	Analyze (L4)	CO5	7

Note: As per complexity of Question and Content Marks may vary

Mr. V. N. Malavade

Course Coordinator