DESIGN AND ANALYSIS OF ALGORITHMS USING JAVA

(EFFECTIVE FROM THE ACADEMIC YEAR 2022-2023)

DESIGN AND ANALYSIS OF ALGORITHMS USING JAVA			
Course code:	**IPCC21CG42**	**Credits:**	**4**
L:T:P:	**2: 2:0**	**CIE Marks:**	**50**
Exam Hours:	**03**	**SEE Marks:**	**50**
Total Hours:	**40**		

COURSEOBJECTIVES	
1.	Understand the fundamentals of algorithm design and analysis. Define an algorithm and its role in problem-solving. Demonstrate knowledge of asymptotic notations and analyse the efficiency of algorithms using mathematical analysis.
2.	Apply various algorithmic problem-solving techniques, including brute force, decrease and conquer, divide and conquer, transform-and-conquer, and space and time trade-offs. Implement and analyse algorithms such as sorting, searching, graph traversal, and string matching.
3.	Explore dynamic programming and greedy technique for solving optimization problems. Apply dynamic programming to solve problems such as binomial coefficient and the Knapsack problem. Understand the principles of greedy algorithms and apply them to solve problems like Prim's Algorithm, Kruskal's Algorithm, and Dijkstra's Algorithm.
4.	Develop an understanding of backtracking and branch-and-bound techniques for solving combinatorial problems. Apply backtracking to solve problems like the n-Queens problem and the Subset-Sum problem. Utilize branch-and-bound for solving problems like the Traveling Salesperson problem and the 0/1 Knapsack problem.
5.	Understand the concepts of NP and NP-complete problems. Differentiate between deterministic and nondeterministic algorithms. Define the classes P, NP, NP-complete, and NP-hard. Analyse the complexity of problems and identify if they belong to the NP-complete class.

COURSEOUTCOMES:ATTHE ENDOFTHECOURSE,STUDENT WILL BE ABLE TO	
CO1	Develop a solid understanding of algorithmic problem-solving techniques and their efficiency analysis. Gain proficiency in analyzing the time and space complexity of algorithms using mathematical analysis and asymptotic notations. Apply these skills to evaluate and compare the performance of brute force algorithms such as selection sort, bubble sort, sequential search, and brute-force string matching.
CO2	Demonstrate proficiency in applying decrease and conquer and divide and conquer techniques to solve algorithmic problems. Implement and analyze algorithms such as insertion sort, depth-first search (DFS), breadth-first search (BFS), topological sorting, merge sort, quicksort, multiplication of long integers, and Strassen's matrix multiplication.
CO3	Apply transform-and-conquer techniques, including pre-sorting, heapsort, and Horner's rule. Understand the space and time trade-offs in algorithms, and apply techniques such as sorting

	by counting, naive string matching, Horspool's algorithm, and the Boyer-Moore algorithm.
CO4	Develop a strong understanding of dynamic programming and greedy techniques in algorithm design. Apply dynamic programming to solve problems such as the binomial coefficient, the Knapsack problem, and the algorithms of Warshall and Floyd. Apply greedy techniques to solve problems such as Prim's Algorithm, Kruskal's Algorithm, and Dijkstra's Algorithm. Gain proficiency in analysing problem characteristics and selecting appropriate algorithmic approaches for efficient problem-solving.
CO5	Apply advanced algorithmic problem-solving techniques such as backtracking and branch-and-bound to solve complex problems like the n-Queens problem, the Subset-Sum problem, the Traveling Salesperson problem, and the 0/1 Knapsack problem. Understand the concepts of NP and NP-complete problems, including basic concepts, nondeterministic algorithms, and the classes P, NP, NP-complete, and NP-hard.

MAPPING OF COURSE OUTCOMES TO PROGRAM OUTCOMES

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	2	-	2	-	-	-	-	-	-	-	-	1	-	-
CO2	3	2	-	-	-	2	2	-	-	-	-	-	1	-	-
CO3	-	3	-	-	-	2	2	-	-	-	-	1	2	1	-
CO4	-	3		-	-	2	-	2	-	-	-	1	1	1	-
CO5	3	2	2	-	-	-	-	-	-	-	-	1	1	1	-

Module	Module Contents	Hours	CO's
	Module 1		
1.	**Introduction:** What is an Algorithm? Fundamentals of Algorithmic Problem Solving, **Fundamentals of the Analysis of Algorithm Efficiency:** The Analysis Framework, Asymptotic Notations and Basic Efficiency Classes, Mathematical Analysis of Non recursive Algorithm, Mathematical Analysis of Recursive Algorithms. **Brute Force**: Selection Sort and Bubble Sort, Sequential Search and Brute-Force String Matching	**08**	**CO1**

Laboratory Component **1.** Implement Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N. **2.** Sort a given set of integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.			
Module 2			
2.	**Decrease and Conquer:** Insertion Sort, Depth First Search, Breadth First Search, Topological Sorting, Applications of DFS and BFS. **Divide and Conquer**: Merge sort, Quick sort, Multiplication of long integers, Strassen's Matrix multiplication.	**08**	**CO2**
Laboratory Component 1. Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. 2. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. 3. Sort a given set of N integer elements using Quick Sort technique and compute its time taken. 4. Write program to do the following: a. Print all the nodes reachable from a given starting node in a digraph using BFS method. b. Check whether a given graph is connected or not using DFS method. 5. Write program to obtain the Topological ordering of vertices in a given digraph.			
Module 3			
3.	**Transform-and-Conquer:** Presorting, Heaps and Heapsort, Horner's Rule **Space and Time Tradeoffs**: Sorting by Counting, Naive String Matching, Horspool's and Boyer-Moore algorithm	**08**	**CO3**
Laboratory Component 1. Write a program to implement Horspool's algorithm for String Matching. 2. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.			
Module 4			
4.	**Dynamic Programming:** Binomial Coefficient, The Knapsack Problem, Warshall's and Floyd's Algorithms **Greedy Technique:** Prim's Algorithm, Kruskal's Algorithm, Dijkstra's Algorithm	**08**	**CO4**

Laboratory Component 1. Implement Warshall's algorithm using dynamic programming. 2. Implement All Pair Shortest paths problem using Floyd's algorithm. 3. Implement Single source shortest path using Dijkstra's algorithm. 4. Find Minimum Cost Spanning Tree of a given undirected graph using a. Prim's algorithm. b. Kruskals algorithm.			
Module 5			
5.	**Backtracking**: n-Queens Problem, Subset-Sum Problem **Branch-and-Bound**: Travelling Sales Person problem,0/1 Knapsack problem **NP and NP-Complete Problems** : Basic concepts, nondeterministic algorithms, P, NP, NP Complete, and NP-Hard classes.	**08**	**CO5**
Laboratory Component 1. Implement "Sum of Subsets" using Backtracking.: Find a subset of a given set S = {s1, s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution. 2. Implement "N-Queens Problem" using Backtracking.			

TEXTBOOKS	
TBNo.	**Author/Edition/Publication/Year**
1.	Introduction to the Design and Analysis of Algorithms, Anany Levitin: 2nd Edition, 2009. Pearson.
2.	Computer Algorithms/C++, Ellis Horowitz, SatrajSahni and Rajasekaran, 2nd Edition, 2014, Universities Press.

REFERENCEBOOKS	
RBNo.	**Author/ Edition/Publication/Year**
1.	Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronal L. Rivest, Clifford Stein, 3rd Edition, PHI.
2.	Design and Analysis of Algorithms, S. Sridhar, Oxford (Higher Education)