GUJARAT TECHNOLOGICAL UNIVERSITY

Syllabus for Master of Computer Applications, 5th Semester

Subject Name: Design and Analysis of Algorithms (DAA)

Subject Code: 4659301

With effective from academic year 2018-19

1. Learning Objectives:

To understand alternate methods of writing algorithms under various categories, such as Divide-and-Conquer, Dynamic Programming, Greedy Methods, Backtracking, Branch & Bound, etc. To be able to analyze algorithms by working out complexity of algorithms. To understand the basics of P, NP, and NP Complete problems

2. Prerequisites: Programming Language: C, Data Structure

3. Course Contents:

Sr. No.	Course Content	Percentage Weightage
1	**Basic Concepts of Analysis and Design of Algorithms** Introduction; Characteristics of iterative algorithms; Efficiency of algorithms; Estimating and specifying execution time; Order notation: Big-Oh, Theta, Omega, Small-Oh, Small-Omega notations; Algorithm strategies	**7%**
2	**Algorithms Using Divide-and-Conquer Strategy** Introduction; Examples: x**n; Multiplication algorithm and its analysis; Binary search and its analysis; Closest pair; Merge sort and its analysis; Limitations of Divide-and-Conquer strategy; Decrease-and-Conquer approach: Permutation generation	**20%**
3	**Greedy Methods** Introduction; Knapsack problem; Job sequencing with deadlines; Minimum spanning trees: Prim's algorithm, and Kruskal's algorithm; Shortest path, Dijkstra's shortest path algorithm, Optimal merge patterns	**20%**
4	**Dynamic Programming** Introduction; Examples: Coin exchange problem; Principle of Optimality; Rod cutting problem, Multistage graphs, Traveling salesman problem, Matrix multiplication, Longest common sub-sequence, Maximum flow problem	**20%**
5	**Backtracking, Branch and Bound Algorithms** Combinatorial search; Search and traversal: Breadth First Search (BFS), Depth First Search (DFS); 8-Queen problem; M-Coloring problem; Hamiltonian circuits; Branch-and-Bound algorithms, Examples: Shortest path; 16-Puzzle and 8-Puzzle; Scale balancing, 0/1 Knapsack problem, Traveling salesman problem; Limitations of Branch-and-Bound	**23%**

6	**Efficiency of Algorithms; Complexity Calculation and Categorization** Polynomial-time and Non-polynomial-time algorithms; Worst and average case behavior, Probabilistic average case analysis; Time analysis of algorithms; Examples: Matrix multiplication; Efficiency of recursion; Complexity: Notion of complexity, Profiling, Suppressing multiplicative constants, Counting dominant operations, Growth rate, Upper bounds, Asymptotic growth rate; The 'O' notation; Simplified definition of 'O'; 'O' notation rules **Examples of Complexity Calculation**: Sorting examples: Bucket sort, Radix sort, Simple Insertion sort, Quick sort, Heap sort, Merge sort, Counting Sort; Binary, Binomial and Fibonacci Heaps; Binomial Heap; Dijkstra's shortest path algorithm; **Complexity Categorization of Problems**: Introduction: P, NP, NPC, NPH; Upper and lower bounds; Four levels of algorithmic behavior; summary	**10%**

4. Text Book(s):

1. Parag H Dave, Himanshu B Dave, “Design and Analysis of Algorithms”, Pearson, 2nd Edition (2014)

5. Reference Books:

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L Rivest, Clifford Stein, “Introduction to Algorithms”, PHI, 2nd Edition
2. Anany Levitin, “Introduction to Design and Analysis of Algorithms”, Pearson (2014)
3. S. Baase, “Computer Algorithms: Introduction to Design and Analysis”, Pearson (2002)
4. Kleinberg, “Algorithm Design”, Pearson (2013)
5. Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, “Fundamentals of Computer Algorithms”, Universities Press, 2nd Edition (2008)
6. Thomas H. Cormen, “Algorithms Unlocked”, MIT Press (2013)
7. Sanjay Dasgupta, “Algorithms”, McGraw-Hill (2006)
8. Gerard Tel, “Introduction to Distributed Algorithms”, Cambridge University Press (2004)

6. Unit wise coverage from Text book(s):

Unit 1	**Topics**
I	Chapter-1; Chapter-4
II	Chapter-9 (9.1: Ex-1; 9.2: Ex-5, Ex-7, Ex-9; 9.4; 9.6.1, 9.6.2)
III	Chapter-10 (10.1, 10.2, 10.3, 10.4.1-10.4.4, 10.5, 10.6)
IV	Chapter-11 (11.1, 11.2, 11.3, 11.4, 11.5, 11.6, 11.9)
V	Chapter-12 (12.1; 12.2; 12.3; 12.4.2, 12.4.3; 12.6.1, 12.6.3, 12.6.4, 12.6.6, 12.6.7)
VI	Chapter-14 (14.1, 14.2, 14.3, 14.3.1, 14.4, 14.5, 14.5.1-14.5.11); Chapter-15 (15.1, 15.2, 15.4, 15.5); Chapter-17 (17.1, 17.2, 17.3, 17,18)

7. Accomplishment of the student after completing the course:

The student will be able to decide on an appropriate category of algorithms for solving a given problem. With an understanding of more than one method of solving problems using algorithms, (s)he will be able to carry out complexity of algorithms and decide on an efficient algorithm for the task on hand. (S)he will also have an idea about categorization of problems into P, NP, NPC, NPH.

<u>Practicals List</u>

Objectives: To develop ability to write algorithms (often more than one different algorithms) for various problems and implement them using C language.

Prerequisites: C Language, Data Structures

Advice (Note) to Teachers:

.The list of exercises given below is an indicative list. More than one algorithm is possible for many problems. Teachers should encourage students to try out multiple algorithms for the given problems and discuss among them about relative efficiency (and complexity) of algorithms.

.All the algorithms are to be implemented in C language

.First list of exercises have been labeled as “Mandatory” while the exercises in the second list have been marked as “Desirable”. Teachers should encourage bright students to complete desirable exercises as well.

A) List of Mandatory Lab Exercises (Write Algorithms and Implement in C Language)

For the following problems, students are expected to write one or more (as the case may be) algorithms along with the complexity of these algorithms, and implement them in C Language.

1. Find square root of a number. Can we use Divide & Conquer approach for this problem? Can we have a still better algorithm to solve the problem?
2. Determine smallest divisor of an integer.
3. For a given value of n, generate prime numbers <= n (more than one algorithms are possible)
4. Find X^n. Iterative and recursive algorithms are possible with complexity $\log_2 n$
5. Determine product of 2 integers (a * b) as repeated sums. Iterative and recursive algorithms are possible.
6. Determine product of 2 large integers using multiplication of their digits. For simplicity, assume both numbers to have same number of digits. This assumption can be relaxed subsequently.
7. Binary Search of an ordered array. Iterative and Recursive algorithms are possible.
8. Sort a given sequence of numbers using (a) Bubble Sort, and (b) Merge Sort
9. Knapsack problem using Greedy algorithm.
10. Solution of Rod-cutting problem using Dynamic Programming algorithm.
11. Multiplication of n Matrices using Dynamic Programming algorithm.
12. Breadth First Search (BFS) in a binary tree.
13. Depth First Search (DFS) in a binary tree.
14. 8-Puzzle and 16-Puzzle
15. Solve 8 Queens problem.

16. Solve Scale Balancing problem.
17. Prim's algorithm to find minimum spanning (cost) tree (shortest path in a tree).

B) List of Lab Exercises (Write optimized code)

18. **Number of ways:**
    You are given three strings A,B and C. From the strings A and B, you can create all possible strings X such that X contains atleast one character from both the strings, and the order of all the selected characters in individual strings is preserved.

    For example:
    A = "ab", B = "cd"
    All possible strings are: { abc, abcd, abd, ac, acb, acbd, acd, acdb, ad, adb, bc, bcd, bd, ca, cab, cabd, cad, cadb, cb, cbd, cda, cdab, cdb, da, dab, db }

19. **Compute** the sum of the Bitwise OR of all the subarrays present in the array.
    Input Array: 1 2 3
    all possible subsets are {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}
    ` Bitwise OR of these subsets are, 1 + 2 + 3 + 3 + 3 + 3 + 3 = 9.
    Output : 18

20. **Given** an array, we need to calculate the Sum of Bit-wise AND of all possible subsets of given array.
    Input Array: 1 2 3
    all possible subsets are {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}
    ` Bitwise AND of these subsets are, 1 + 2 + 3 + 0 + 1 + 2 + 0 = 9.
    Output : 9
21. Without using divide operator, write a function to divide two integers.
22. Find if a number is divisible by 17 using bitwise operators
23. Compute Subtract using plus operator.
24. Find maximum of two numbers without any comparison.
25. De Bruijn Sequence : Given a string (like AB), generate shortest String containing all combinations of the given string.

    Example:
    For given string "AB", all combinations are {AA, AB, BA, BB}
    One string containing all these combinations is AAABBABB.
    But this is not the shortest.
    Shortest string containing all combinations is AABBA
    Such a string is called
26. Given a number N, find all pairs of numbers such that $N^2 = X^2 + Y^2$
27. Given some integer N, find all the prime factors of that number.

C) List of Desirable Lab Exercises (Write Algorithms and Implement in C Language)
For the following problems, students are expected to write one or more (as the case may be) algorithms along with the complexity of these algorithms, and implement them in C Language.

28. Generate pseudo-random numbers.
29. Strassen matrix multiplication.

30. Find the closest pair out of given n points in 2-dimensional space.
31. Unique partitions of a positive integers.
32. Generate permutations of given n numbers. Iterative and recursive algorithms are possible.
33. Generate Gray Code.
34. Kruskal's algorithm to find minimum spanning (cost) tree (shortest path in a tree).
35. Determine largest common subsequence
36. Implement Twister game

Reference Books:

.Parag H Dave, Himanshu B Dave, “Design and Analysis of Algorithms”, Pearson (2014)
.Anany Levitin, “Introduction to Design and Analysis of Algorithms”, Pearson (2014)
.Thomas H. Cormen, “Algorithms Unlocked”, MIT Press (2013)

Accomplishment of the student after completing the course:

The student will be able to write one (and sometimes more than one) algorithm to solve a given problem. (S)he will be able to determine complexity of algorithms and select the most efficient algorithm for a given task.