## UE16CS255 – Design and Analysis of Algorithms Laboratory

**# of Credits: 1** **# of Weeks: 13**

| | |
|---|---|
| Laboratory Title | Design and Analysis of Algorithms Laboratory (UE16CS255) |
| For the Class | B. Tech. 4th Semester 2016-2020 batch |
| Preamble | Design and Analysis of Algorithms Laboratory is a core course and complements the theory course in Design and Analysis of Algorithms. In the theory course, the students are given a fresh approach in both design and analysis of algorithms. In the lab course, the students implement the algorithms in a programming language and do empirical analysis of the algorithms learnt in the theory course. |
| Objective | 1. Learn to design and implement Brute Force algorithms.<br>2. Learn to design, analyze, implement and compare the actual running time of algorithms designed in Brute Force and Divide-and-Conquer techniques.<br>3. To design and implement algorithms with Decrease-and-Conquer, and Transform-and-Conquer techniques.<br>4. To design and implement algorithms with Space and Time Tradeoff with an emphasis on the resource utilization in terms of time and space.<br>5. To design and implement optimization algorithms using advanced design techniques such as Dynamic Programming and Greedy technique. |
| Outcome | 1. Design and implement Brute Force algorithms.<br>2. Design and implement algorithms with Divide-and-Conquer technique, and appreciate the efficiency with their Brute Force counterparts.<br>3. Design and implement algorithms with Decrease and Conquer, and Transform and Conquer techniques.<br>4. Design and implement algorithms with Space and Time Tradeoff, and appreciate the tradeoff between utilization of space and time.<br>5. Design and implement optimization algorithms using Dynamic Programming and Greedy technique. |

| Session | Tasks |
|---|---|
| 1 | Introduction to the lab environment.<br>- Compile and execution of a C program in Linux.<br>- Handling Input-Output formats with large number of test-cases.<br>Brute Force: Implementation of **Sequential Search** algorithm<br>Find the key element in an array of integers using the sequential search algorithm. |
| 2 | Brute Force: Implementation of **String Matching** algorithm<br>Find a pattern of length m in a text of length n using the naive string matching algorithm. |
| 3 | Brute Force: Implementation of **Bubble Sort** algorithm<br>Sort a given array of integers using the bubble sort algorithm.<br><br>Brute Force: Implementation of **Selection Sort** algorithm<br>Sort a given array of integers using the selection sort algorithm. |
| 4 | Brute Force: Solution for **Traveling Salesperson Problem**<br>Find a solution to the traveling salesperson problem using the exhaustive search method. |

| | |
|---|---|
| 5 | Divide and Conquer: Implementation of **Merge Sort**<br>Sort a given array of integers using the merge sort algorithm.<br><br>Divide and Conquer: Implementation of **Binary Search**<br>Search for a key element in a sorted array of integers. |
| 6 | Divide and Conquer: Implementation of **Quick Sort**<br>Sort a given array of student records using the quick sort algorithm. |
| 7 | Decrease and conquer: Implementation of **Insertion Sort** algorithm<br>Sort a given array of student records using the insertion sort algorithm.<br><br>Decrease and conquer: Demonstration of **BFS**<br>Find the number of components of an undirected graph given in the form of an adjacency matrix using BFS technique. |
| 8 | Decrease and conquer: Demonstration of **DFS** algorithms<br>Find the number of components of an undirected graph given in the form of an adjacency matrix using DFS technique.<br><br>Decrease and conquer: **Topological Sorting** of vertices in a digraph<br>Find a topological order of a directed acyclic graph using the DFS technique. |
| 9 | Transform and Conquer: Implementation of **Heap Sort** algorithm<br>Sort a given array of student records using the heap sort algorithm. Use bottom up approach for the heap construction.<br><br>Space and Time Tradeoffs: Implementation of **Sorting by Distribution Counting** algorithm<br>Sort a given array of student records using the distribution counting sort algorithm. |
| 10 | Space and Time Tradeoffs: Implementation of **Horspool's** algorithm<br>Find a pattern of length m in a text of length n using the Horspool's algorithm. |
| 11 | Dynamic Programming: Implementation of **Warshall's** algorithm<br>Find the transitive closure of a graph given in the form of an adjacency matrix.<br><br>Dynamic Programming: Implementation of **Floyd's** algorithm<br>Find all-pairs-shortest-paths of a weighted graph given in the form of a cost matrix. |
| 12 | Dynamic Programming: Solution for the **Knapsack Problem**<br>Find the solution to a 0/1 Knapsack problem using Dynamic Programming technique. |
| 13 | Greedy Technique: Implementation of **Prim's** algorithm<br>Find a minimum spanning tree of a weighted connected undirected graph using the Prim's algorithm.<br><br>Greedy Technique: Implementation of **Dijkstra's** algorithm<br>Find single-source-shortest-paths of a weighted connected graph using the Dijkstra's algorithm. |