B.P. Poddar Institute of Management and Technology
Department of Information Technology
Academic Year: 2018-2019 [Odd Semester]

List of Experiments:

S.No.	***Name of Experiment***	***CO***	***PO***	***PSO***
1	Implement binary search using Divide and Conquer approach.	***CO1***	***PO3,PO5***	***PSO1, PSO2***
2	Implement Merge Sort using Divide and Conquer approach.	***CO1***	***PO1,PO5***	***PSO1, PSO2***
3	Implement Quick Sort using Divide and Conquer approach.	***CO3***	***PO1,PO5***	***PSO1, PSO2***
4	Find Maximum and Minimum element from an array of integer using Divide and Conquer Approach.	***CO2***	***PO1,PO5***	***PSO1, PSO2***
5	Find the minimum number of scalar multiplication needed for chain of matrix.	***CO3***	***PO1,PO5***	***PSO1***
6	Implement all pair of Shortest path for a graph (Floyed- Warshall Algorithm).	***CO3***	***PO1,PO5***	***PSO1***
7	Implement Traveling Salesman Problem.	***CO3***	***PO1,PO5***	***PSO1***
8	Implement Single Source shortest Path for a graph using (Dijkstra /Bellman Ford Algorithm).	***CO3***	***PO1,PO3, PO5***	***PSO1***
9	Implement 15 Puzzle Problem.	***CO4***	***PO1,PO3, PO5***	***PSO1, PSO2***
10	Implement 8 Queen problem.	***CO4***	***PO1,PO3, PO5***	***PSO1***
11	Implement backtracking method in Graph Coloring Problem	***CO4***	***PO1,PO5***	***PSO1***
12	Implement greedy method in Knapsack Problem.	***CO5***	***PO1,PO5***	***PSO1,PSO2***

13	Implement greedy method in Job sequencing with deadlines.	***CO5***	***PO1,PO5***	***PSO1***
14	Implement greedy method to find Minimum Cost Spanning Tree by applying Prim's Algorithm.	***CO5***	***PO1,PO5***	***PSO1,PSO2***
15	Implement greedy method to find Minimum Cost Spanning Tree by applying Kruskal's Algorithm.	***CO5***	***PO1,PO5***	***PSO1,PSO2***
16	Implement graph traversal algorithm by applying Breadth First Search (BFS).	***CO5***	***PO1,PO5***	***PSO1***
17	Implement graph traversal algorithm by applying Depth First Search (DFS)	***CO5***	***PO1,PO5***	***PSO1***
18	**Case study: 1**. Perrin Number problem: p(0)=3,p(1)=0,p(2)=2,p(n)=p(n 2)+p(n-3), Illustrate time and space trade-off. Design/State at least three algorithms to study the timing and complexity analysis for that problem. **Case Study: 2** Design algorithms for integer multiplication which multiplies n-bit numbers by recursively multiplying n/ 2 bit numbers. Calculate the time complexity o your algorithm. Execute the Program in C. Can you propose any optimization technique for this problem. **Case Study: 3** You are given an infinite array A[·] in which the first n cells contain integers in sorted order and the rest of the cells are filled with ∞. You are not given the value of n. Describe an algorithm that takes an integer x as input and finds a position in the array containing x, if such a position exists, in O(log n) time. Execute the Program in C. **Case Study: 4** There are 3 (non-decreasing) sorted arrays, namely A, B and C. Define a triplet (a, b, c) such that a is in A, b is in B and c is in C. Also, define dist (a, b, c) = max (|a-b|, |b-c|, |c-a|). Now find the triplet (a_min, b_min,		***PO1,PO3, PO5***	***PSO1,PSO2***

	c_min) from A, B and C such that dist (a_min, b_min, c_min) is minimum among all possible triplets. Can you propose an algorithm which will takes O (n(A)+n(B)+n(C)). Execute the Program in C. **Case Study: 5** You have n = 2^k coins and a pan balance. One of these coins is counterfeit and is lighter (in weight) than the rest. Design a divide-and-conquer algorithm to find the counterfeit coin. You may put any number of coins in each pan of the balance, and, it tells you which side is heavier. Analyze your algorithm. Execute the Program in C. **Case Study: 6** Generate the power of even number, for example 128, How many minimum numbers of multiplications you needed. Design the algorithm. Calculate the time complexity. **Case Study: 7** You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time O(n log n).			