ECM 1414 - Weekly Exercises and Challenges

Week 1: Introduction to Algorithms

Exercise: Write a Python function to compute the Fibonacci sequence iteratively for n.

Challenge: Implement a program to compute the Fibonacci sequence recursively and iteratively. Then, modify it to cache previous results (using memorization/dynamic programming) for the recursive version and compare the performance.

Week 2: Analysis of Algorithms

Exercise: Write a program to count the number of comparisons in a sequential search for an unsorted array.

Challenge: Design and implement a program to measure and graph the runtime of three algorithms with O(n), O(n^2), and O(n log n) complexities for various input sizes (e.g., n = 100, 500, 1000, 5000).

Week 3: Asymptotic Notations

Exercise: Write a function to compute the number of comparisons in Bubble Sort for a given array size.

Challenge: Write a program to analyse and classify $f(n) = 5n^3 + 3n^2 - 4n + 10$ against $g(n) = n^3$ and $h(n) = n^2$ using asymptotic notation. Validate the results with computational experiments.

Week 4: Recurrences

Exercise: Solve the Tower of Hanoi problem for n = 3 disks and display the moves.

Challenge: Implement the Tower of Hanoi problem iteratively (without recursion) for $n \leq 10$. Include a feature to count the number of moves and show the state of the rods after each step.

Week 5: Introduction to Data Structures

Exercise: Implement a program to rotate a 1D array to the right by k positions using extra memory.

Challenge: Implement a dynamic 2D matrix with operations for insertion, deletion, resizing rows and columns, and accessing elements efficiently.

Week 6: Singly Linked Lists

Exercise: Implement a singly linked list with basic operations: insertion at the head and deletion from the tail.

Challenge: Implement a function that reverses a singly linked list in place.

Week 7: Stacks and Queues

Exercise: Implement a stack and use it to reverse a string input by the user.

Challenge: Implement a deque (double-ended queue) with linked list representation. Include operations for insertion and deletion at both ends and test it with a palindromic check on strings.

Week 8: Sorting Algorithms

Exercise: Write a program to sort an array of integers using Insertion Sort and print the sorted array.

Challenge: Implement a hybrid sorting algorithm that switches from Merge Sort to Insertion Sort when the subarray size falls below a threshold (e.g., k = 10). Compare its performance with plain Merge Sort.

Week 9: Trees

Exercise: Implement a binary tree with functions for insertion and inorder traversal.

Challenge: Implement an AVL Tree (or Red-Black Tree) with insertion and rotation operations. Test it with a series of input values to demonstrate self-balancing properties.

Week 10: Graphs

Exercise: Implement a graph using an adjacency list and perform a Breadth-First Search (BFS).

Challenge: Implement Kruskal's algorithm to find the Minimum Spanning Tree of a graph and compare its results with Prim's algorithm.

Week 11: Advanced Graph Algorithms

Exercise: Implement Prim's algorithm to find the minimum spanning tree of a simple weighted graph.

Challenge: Implement Floyd-Warshall's algorithm to find the shortest paths between all pairs of nodes in a weighted graph. Test it on graphs with both positive and negative edge weights (no negative cycles).