Factorial Of a Number

Calculate the factorial of a given non-negative integer 'n' using three different approaches: iterative, recursive, and memoization.

Approach 1: Function to compute factorial using iterative approach

- The **factorialIterative** function calculates the factorial of 'n' using an iterative approach. It initializes the result **res** to 1 and then iteratively multiplies it with all numbers from 1 to 'n'. The final result is returned.
- **Time Complexity: O(n) - The loop runs 'n' times, performing constant time operations in each iteration.**
- **Space Complexity: O(1) - The function uses only a single variable to store the result.**

Approach 2: Function to compute factorial using recursive approach

- The factorialRecursive function calculates the factorial of 'n' using a recursive approach. It checks if 'n' is either 0 or 1, in which case the factorial is 1 (base case). Otherwise, it recursively calls the factorialRecursive function with 'n - 1' and multiplies the result by 'n'.
- **Time Complexity: O(n) - The function makes 'n' recursive calls, and each call performs constant time operations.**
- **Space Complexity: O(n) - The function uses the call stack, which grows with 'n' recursive calls, so the space complexity is proportional to 'n'.**

Approach 3: Function to compute factorial using memoization approach

- The factorialMemoization function calculates the factorial of 'n' using a memoization approach. It maintains a memoization table (unordered_map) to store previously computed factorial results to avoid redundant calculations. It follows the same logic as the recursive approach, but before making a recursive call, it checks if the result is already available in the memoization table.
- **Time Complexity: O(n) - The function still makes 'n' recursive calls, but the memoization table ensures that each factorial value is calculated only once.**
- **Space Complexity: O(n) - The memoization table can store up to 'n' factorial values, so the space complexity is proportional to 'n'.**