Part 1: Advanced Array Manipulation (10 questions)

1. **Flatten a Multi-dimensional Array:** Write a function that takes an array of arrays and flattens it into a single array. The function must handle nested arrays of any depth without using Array.prototype.flat() or Array.prototype.flatMap().
2. **Group By Property:** Given an array of objects, write a function that groups the objects by a specified property. The function should return an object where keys are the unique property values and each value is an array of corresponding objects.
3. **Find Pairs with a Given Sum:** Given an array of integers and a target sum, write a function to find all unique pairs of numbers in the array that add up to the target sum. The function should return an array of these pairs.
4. **Rotate Array In-Place:** Write a function that rotates an array to the right by k steps without creating a new array. The rotation should be performed directly on the input array.
5. **Calculate Median:** Write a function that takes an array of numbers and returns the median. The median is the middle value of a sorted list of numbers. If the list has an even number of elements, the median is the average of the two middle elements.
6. **Chunk an Array:** Write a function that splits an array into smaller arrays of a specified size (chunk).
7. **Find Symmetrical Difference:** Write a function that takes two arrays and returns a new array of elements that are found in exactly one of the two input arrays.
8. **Implement a Custom map Function:** Write a function called myMap that takes an array and a callback function as arguments. myMap should behave like the native Array.prototype.map() method, applying the callback to each element and returning a new array with the results. You cannot use the native map method.
9. **Deep Comparison of Arrays:** Write a function that performs a deep comparison between two arrays. It should return true if the arrays contain the same elements in the same order, including nested arrays, and false otherwise.
10. **Find the Longest Common Subsequence:** Given two arrays, write a function that finds the longest common subsequence (LCS). The LCS is the longest sequence that can be obtained by deleting some elements from the first and second arrays.

Part 2: Advanced Object Manipulation (10 questions)

1. **Deep Clone an Object:** Write a function that performs a deep clone of an object. The function should be able to handle nested objects, arrays, and primitive types without using JSON.parse(JSON.stringify()) or any third-party libraries.
2. **Filter Object by Value:** Write a function that takes an object and a callback function. The function should return a new object containing only the key-value pairs where the value satisfies the condition defined by the callback function.
3. **Merge Multiple Objects:** Write a function that takes an arbitrary number of objects as arguments and merges them into a single new object. If a key appears in multiple objects, the value from the last object in the arguments should be used.

4. **Check for Circular References:** Write a function that takes an object and determines if it contains any circular references. A circular reference occurs when an object's property directly or indirectly references itself.
5. **Count Nested Key Occurrences:** Write a function that takes a deeply nested object and a key as arguments. The function should count and return the total number of times the key appears throughout the entire object, at any level of nesting.
6. **Implement a Simple Memoization Function:** Write a function called memoize that takes a function as an argument and returns a memoized version of it. The memoized function should cache the results of expensive calculations based on their input parameters.
7. **Key-Value Swapping:** Write a function that takes an object and returns a new object with its keys and values swapped. Assume all original values are unique and can be used as keys.
8. **Object Path Access:** Write a function getPathValue(obj, path) that safely retrieves a value from a nested object using a string path (e.g., 'a.b.c'). The function should handle non-existent paths gracefully and return undefined if any part of the path is not found.
9. **Check for Subset Object:** Write a function isSubset(mainObj, subsetObj) that returns true if subsetObj is a subset of mainObj. A subset means that mainObj contains all the same properties and values (at the same nesting levels) as subsetObj.
10. **Implement a Custom reduce on Objects:** Write a function called objectReduce that takes an object, a callback function, and an initial value. It should apply the callback to each key-value pair of the object and return a single accumulated value, similar to Array.prototype.reduce().