Lab Assignment: Chain Matrix Multiplication

Objective: To understand and implement the chain matrix multiplication problem using four different methods: recursive, memoization, dynamic programming, and optimal parenthesization. Analyze the time complexity and efficiency of each approach.

Instructions:

1. **Introduction:**
   - Briefly explain the chain matrix multiplication problem.
   - Discuss the significance of optimizing matrix multiplication order in computational efficiency.
2. **Task 1: Recursive Approach**
   - Implement a recursive solution to the chain matrix multiplication problem.
   - Write a function matrixChainRecursive(p) where p is an array representing the dimensions of the matrices.
   - Analyze the time complexity of the recursive solution.
3. **Task 2: Memoization Approach**
   - Implement a memoized solution to the chain matrix multiplication problem.
   - Write a function matrixChainMemoized(p) that uses a memoization table to store intermediate results.
   - Compare the time complexity and space complexity with the recursive approach.
4. **Task 3: Dynamic Programming Approach**
   - Implement a dynamic programming solution to the chain matrix multiplication problem.
   - Write a function matrixChainDP(p) that uses a dynamic programming table to compute the optimal multiplication order.
   - Analyze the time complexity and space complexity of the dynamic programming approach.
5. **Task 4: Comparative Analysis**
   - Execute all three implementations on the same set of input matrices.
   - Record the execution time for each approach.
   - Compare and discuss the results, highlighting the advantages and disadvantages of each approach.
6. **Task 5: Optimal Parenthesization**
   - Implement a solution to print the optimal parenthesization of the matrices in the chain multiplication problem.
   - Write a function printOptimalParens(s, i, j) that prints the optimal parenthesization based on a table s obtained from the dynamic programming approach.
   - Integrate this function with matrixChainDP(p) to produce the optimal parenthesization along with the optimal cost.

Example Input:

```
# Example dimensions of matrices
p = [10, 20, 30, 40, 30]
```