Program to calculate Factorial of a number

This C++ program calculates the factorial of a given number using both a simple recursive function and a recursive function with memoization. Factorial of a non-negative integer n (denoted as n!) is the product of all positive integers less than or equal to n.

Recursive function to calculate the factorial of a number

1. The program starts by including the necessary header files for input/output and an unordered map (used for memoization).
2. The **factorial** function is defined, which takes an integer **num** as input and returns the factorial of **num**.
3. In the **factorial** function, there are two cases:
    - Base Case: If **num** is 0 or 1, the function returns 1, as the factorial of 0 and 1 is 1.
    - Recursive Case: If **num** is greater than 1, the function calculates the factorial recursively by multiplying **num** with the factorial of **(num - 1)**.

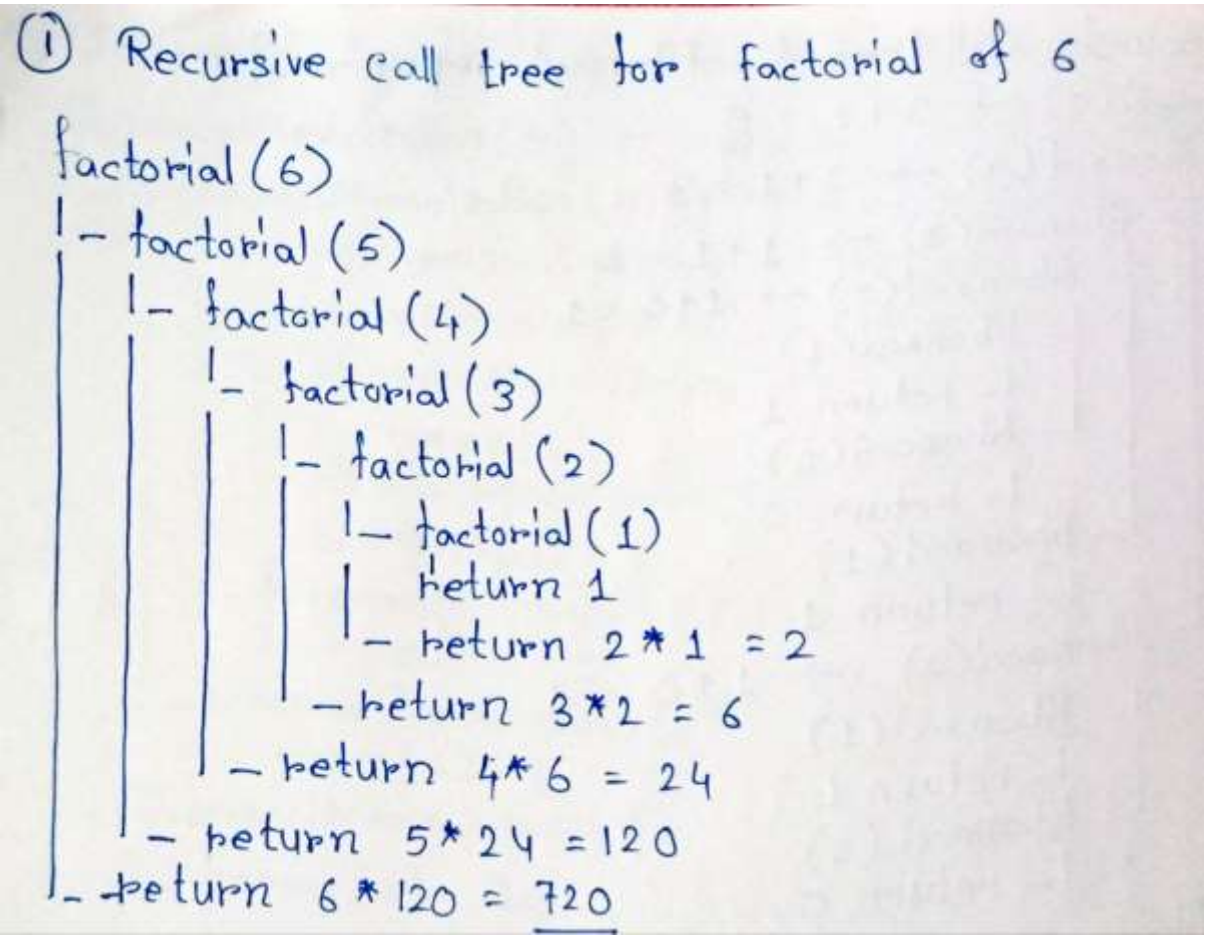
Time Complexity:

- **The time complexity of the factorial function without memoization is O(n) since it calculates the factorial in a straightforward recursive manner, making n recursive calls.**

Space Complexity:

- **The space complexity of the factorial function without memoization is O(n) because it needs to store n recursive calls on the call stack.**

Recursive call stack for this approach:



Recursive function to calculate the factorial of a number with memoization

1. The **factorialMemoization** function is defined, which takes an integer **num** as input and returns the factorial of **num** using memoization.
2. Inside the **factorialMemoization** function, there is a memoization step using an unordered map named **memo**.
3. In the **factorialMemoization** function, there are three cases:
    - Base Case: If **num** is 0 or 1, the function returns 1, as the factorial of 0 and 1 is 1.
    - Memoization Check: The function checks if the factorial of **num** is already calculated and stored in the **memo** map using **memo.count(num)**. If it is present, it directly returns the precalculated value.

- Recursive Case: If the factorial of **num** is not already calculated, the function calculates it recursively by multiplying **num** with the factorial of **(num - 1)**. The result is then stored in the **memo** map for future use.

Time Complexity:

- **The time complexity of the factorialMemoization function with memoization is also O(n). However, the use of memoization significantly reduces the number of redundant recursive calls. Once a factorial is calculated, it is stored in the memo map, so any subsequent calculation for the same number can be retrieved in constant time from the map. Thus, the memoization optimizes the recursion and reduces the number of calculations.**

Space Complexity:

- **The space complexity of the factorialMemoization function with memoization is also O(n) due to the space used by the call stack for recursive calls and the memo map to store the factorial values. However, the memoization approach reduces redundant calculations and makes the algorithm more efficient, especially for larger values of num.**

Recursive call stack for this approach: