Fibonacci Series

This C++ program calculates the nth term of the Fibonacci series using both a simple recursive function and a recursive function with memoization. The Fibonacci series is a sequence of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1.

Recursive function to calculate the nth Fibonacci number

1. The program starts by including the necessary header files for input/output and an unordered map (used for memoization).
2. The **fibonacciSeries** function is defined, which takes an integer **sequence** as input and returns the nth term of the Fibonacci series.
3. In the **fibonacciSeries** function, there are two cases:
    - Base Case: If **sequence** is 0 or 1, the function returns **sequence**. For **sequence = 0**, the Fibonacci series starts with 0, and for **sequence = 1**, it starts with 0, 1.
    - Recursive Case: If **sequence** is greater than 1, the function calculates the nth term of the Fibonacci series recursively by adding the **(sequence - 1)**th and **(sequence - 2)**th terms.

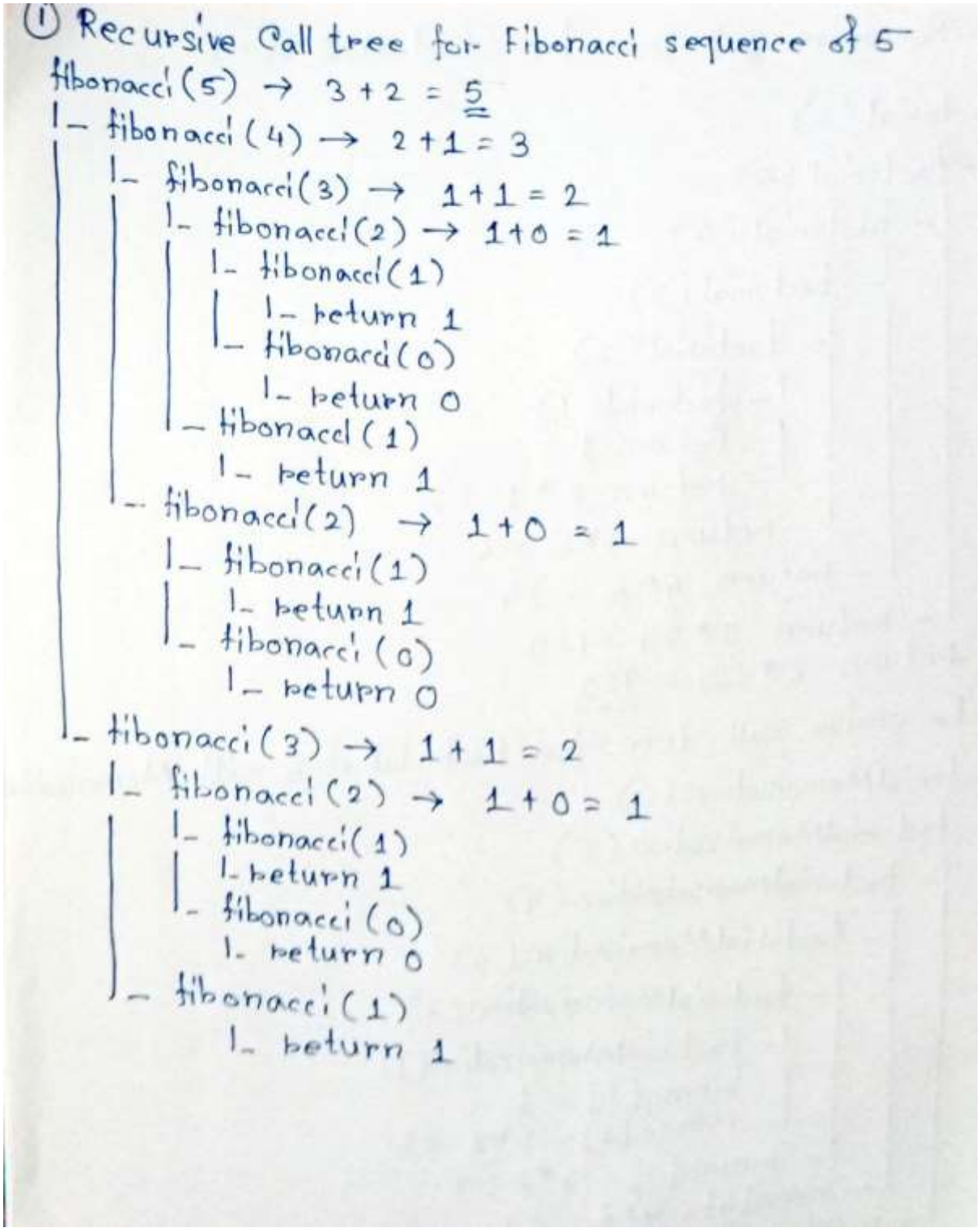
Time Complexity:

- **The time complexity of the fibonacciSeries function without memoization is exponential, specifically O(2^n). This is because the function makes two recursive calls for each sequence number greater than 1, leading to redundant calculations.**

Space Complexity:

- **The space complexity of the fibonacciSeries function without memoization is O(n). This is because it requires space for n recursive calls on the call stack.**

Recursive call stack for the approach:



Recursive function to calculate the nth Fibonacci number with memoization

1. The **fibonacciSeriesMemoization** function is defined, which takes an integer **sequence** as input and returns the nth term of the Fibonacci series using memoization.

2. Inside the **fibonacciSeriesMemoization** function, there is a memoization step using an unordered map named **memo**.
3. In the **fibonacciSeriesMemoization** function, there are three cases:
    - Base Case: If **sequence** is 0 or 1, the function returns **sequence**, same as the **fibonacciSeries** function.
    - Memoization Check: The function checks if the Fibonacci value for the given **sequence** is already calculated and stored in the **memo** map using **memo.count(sequence)**. If it is present, it directly returns the precalculated value.
    - Recursive Case: If the Fibonacci value is not already calculated, the function calculates it recursively by adding the **(sequence - 1)**th and **(sequence - 2)**th terms of the Fibonacci series. The result is then stored in the **memo** map for future use.

Time Complexity:

- **The time complexity of the fibonacciSeriesMemoization function with memoization is linear, specifically O(n). The memoization approach reduces redundant calculations by storing previously calculated Fibonacci values in the memo map. As a result, each Fibonacci number is computed only once, and subsequent calls retrieve the precalculated values in constant time**.

Space Complexity:

- **The space complexity of the fibonacciSeriesMemoization function with memoization is also O(n). The memoization approach optimizes the recursion and reduces redundant calculations, but it requires additional space to store the memo map,** which can hold up to n Fibonacci values.

Recursive call stack for the approach: