2. Prove the time complexity of the algorithms

Ans :

In the standard recursive method for calculating Fibonacci numbers, each function call leads to two more calls, creating an expansive tree of function invocations. This results in an exponential increase in the number of calls, specifically 2^n, where n is the input number. Therefore, the time complexity is O(2^n), reflecting a rapid growth in computational effort with larger inputs.

3. Comment on ways you could improve your implementation (you don't need to do it just discuss it)

Memorization :

Store previously computed Fibonacci values to avoid redundant calculations, reducing time complexity to O(n).

Dynamic Programming:

Use an iterative approach to build the sequence from the ground up, which also achieves O(n) time complexity and is more efficient than recursion.

Iterative Optimization:

Maintain only the last two values in memory, optimizing both time and space usage with a linear time complexity and constant space requirement.

These techniques significantly enhance performance, especially for large values of n.