Explain the concept of recursion and how it can simplify certain problems.

Recursion is a way in programming which calls its function directly (e.g. itself) to solve the problem. It is most commonly used for problems which are a part of larger, more general problems which, therefore, can be broken down into smaller ones. At each step of the recursion, the problem is split into a smaller part and is called over and over again when it's not zero until it reaches the simple base point. This method is particularly accepted for linked or repetitive structures, such as tree walks or calculating mathematical sequences like Fibonacci numbers. For instance, by applying the rate of interest for one period can be used in compound interest for the future value of investment recursively and then if the interest function is not in the overflow mode the remainig period is applied until the final amount is calculated. This strategy is not only the most prominent way to solve but also frequently brings concise and readable code. Recursion plays the role of revealing the true nature of a problem and gathering all the needed details thus leading to easier and clearer troubleshooting.

Discuss the time complexity of your recursive algorithm.

The time complexity of the recursive method is O(n), where n is the number of years. This is because the method makes a recursive call for each year, resulting in n recursive calls.

Explain how to optimize the recursive solution to avoid excessive computation.

To avoid excessive computation and potential stack overflow errors due to deep recursion, memoization or iterative methods can be used. Memoization stores intermediate results to avoid redundant calculations, but in this case, an iterative approach may be more straightforward.