**Understanding Recursive Algorithms**

**Recursion is a concept when a function/method calls itself to solve a smaller sub-problem of the original task, and build up the final output.**

**Ex: Solving factorial: n! \* (n-1)!, Fibonacci: F(n)=F(n-1)+F(n-2); Generating subsets, etc.**

Recursion can simplify code for problems that are:

* Naturally repetitive (patterns, trees)
* Output is based on previous steps (growth, compounding)

We can predict future values using a fixed growth rate(r):

FutureValue(n)=InitialValue\*(1+r)n

Formula for recursion: FutureValue(n)=(1+r)\*FutureValue(n-1)

**Time Complexity of this algorithm:**

Each recursive call computes 1 multiplication and goes down by 1 year.

Recursive Depth=n where n is number of years

Which results in O(n) time complexity.

**Optimization of the solution**

Generally, we can use memoization for recursion if there are repeating subproblems.

In this problem, there are no chances of repetition because values vary for different years.

So, we can simply iterate for n years and calculate the futureValue to avoid StackOverFlow and improve performance.