Implementation of Fixed Growth Simulation

D'Andre Williams (2301128)

School of Computing and Information Technology, University of Technology, Jamaica

CIT 3003: Analysis of Algorithms

Mr. Oral Robinson

November 26, 2025

Table of Contents

Problem Identification and Description

Retirement planning is a complex process wherein individuals must predict the future growth of their savings over a very long period. Most people find it rather challenging to decide whether their current investments would be sufficient to meet their future needs, particularly in calculating the power of compound interest over several decades. Investors who fail to make appropriate forecasts may save less than what is needed, miscalculate the power of interest on long-term growth, or even retire without meeting their objectives.

This is where the difficulty heightens, since manual calculations of compound interest over many years are tiresome, cumbersome, and prone to errors. The fixedInvestor() function was developed as a solution for the problem. It acts as a robust computational model for predicting the future value of an investment at a fixed rate through annual compounding. Thus, it enables the users to rapidly assess various long-term financial situations and get an understanding of how their money is going to grow under consistent conditions.

The function emulates annual compounding, where interest is added to the balance every year and subsequent interest is earned on the updated amount, until the total number of years has passed. Mathematically, the future value of such an investment is given by:

$FV=P(1+r)^n$

Where:

- P = principal
- r = annual rate
- n = years

- FV = Future Value (how much the money will grow to)

Assumptions

- Interest compounds occur once per year.

- There are no deposits or withdrawals during the investment period.

- The growth rate is constant.

- All input values should be positive or zero.

Constraints

- All inputs provided to the function must be numeric values, since the calculation cannot be performed on text, symbols, or empty inputs.

- The compounding loop is required to run exactly for the number of years specified, ensuring that the calculation is neither repeated too many times nor stopped prematurely, which would result in an incorrect future value.

- The annual interest rate must be entered in decimal form, because the function does not convert percentages automatically, and providing the rate incorrectly would lead to inaccurate output.

Algorithmic Classification

The fixedInvestor() algorithm is an iterative, polynomial-time algorithm. It belongs to the class P, which contains all problems solvable in polynomial time using deterministic computation.

The algorithm performs a loop that iterates once per year and applies constant-time updates to the balance. There is no recursion, no branching beyond the loop counter, and no exponential computations.

Why it is in P

- The input size is the number of years n
- The loop executes exactly n times.
- Each iteration performs a constant number of operations (multiplication + addition).
- Therefore, total work is proportional to n.

$T(n)=O(n)$

This confirms that the problem is **polynomial-time solvable**.

The algorithm is one of the simplest forms of polynomial-time financial computation.

Space Complexity

The function stores only:

- the principal
- rate
- years
- and a single balance variable

Thus:S(n)=O(1)

The algorithm is efficient in both time and memory.

Algorithmic Design and Representation

Pseudocode

```
FUNCTION fixedInvestor(principal, rate, years):
   balance = principal
   FOR i =1 TO years DO
      balance = balance × (1 + rate)
   END FOR
   RETURN balance
END FUNCTION
```

Explanation of Pseudocode

Set the initial balance:

The line balance = principal stores the principal as the starting amount. This ensures the calculation begins with the correct initial value.

Begin a loop to simulate each year:

FOR i = 1 TO years DO

This loop runs exactly one time per year, meaning if the investment grows for 10 years, the loop executes 10 times.

Apply annual compounding inside the loop:

The line

balance = balance × (1 + rate)

updates the balance for the current year by multiplying it by (1 + rate).

- The 1 keeps the original balance.

- The rate adds the interest for that year.

  This means the interest earned becomes part of the balance and will itself earn interest in future years—this is compound interest.

Repeat this update for every year:

Each pass through the loop represents one year of growth. The balance increases a little more each year because the interest is always calculated on the updated (and larger) amount.

Return the final accumulated amount:

After the loop finishes running all the years,

RETURN balance

sends back the final value of the investment after all compounding has been applied.

Proof of Correctness (Using Mathematical Induction)

We prove the algorithm computes the correct future value:

$FV(n) = P(1+r)^n$

Base Case: n = 1

After one iteration of the loop, the algorithm performs:

balance=P x (1+r)

The mathematical formula for n=1

$FV(1)= P(1+r)^1 = P(1+r)$

Both results match, so the base case holds for n = 1.

Inductive Hypothesis

Assume that for some integer k=1 true:

$balance=P(1+r)^k$

Inductive Step

Must show that the formula remains correct for iteration n= k+1

On iteration k+1, the algorithm updates the value using

balance=balance x (1+r)

Substituting the inductive hypothesis:

$balance=P(1+r)^k$ x (1+r)

Combing the exponents:

$=P(1+r)^{k+1}$

This matches the mathematical formula for year k+1.

Conclusion

By the principle of mathematical induction, fixedInvestor(principal, rate, years) correctly computes:

$FV=P(1+r)^n$

for all integers $n \geq 1$.

Asymptotic Analysis of FixedInvestor

```
Algorithm FixedInvestor(P, r, n)
    balance = P
    for year =1 to n do
        balance =balance * (1 + r)
    end for
    return balance
End Algorithm
```

Cost Table

Line / Operation	**Cost**	**Times**
balance = P	C_1	1
for year = 1 to n	C_2	n + 1
balance =balance * (1 + r)	C_3	n
return balance	C_4	1

Total Running Time Expression

T(n)= C1(1) + C2(n+1)+C3(n) +C4(1)

Expand:

T(n)=C1+C4+C2(n+1)+C3n

Group like terms:

T(n)=(C1+C4+C2) + (C2+C3)n

Rename constants:

T(n)=a+bn

Asymptotic Complexity

As n grows large, constant terms disappear:

$T(n)=O(n)$

Space Complexity

The algorithm uses:

- P
- r
- n
- balance
- year

All are scalar variables (constant memory), so:

$S(n)=O(1)$

Time Complexity:
$T(n)=C1+C4+C2(n+1)+C3(n)= O(n)$

Space Complexity:

$S(n)=O(1)$

Trade-Offs and Discussion

Although the iterative version of the FixedInvestor(P, r, n) algorithm has a linear running time of O(n), it provides several practical advantages over the closed-form approach. A mathematically equivalent formula,

FV=P(1+r)n,

can be computed using Math.pow() in O(\log n) time due to fast exponentiation. While this is asymptotically faster, it lacks flexibility. Real-world financial modelling often needs year-by-year updates in the form of added features such as annual deposits, withdrawals, inflation adjustments, fees, changing interest rates, and irregular contribution schedules. It would be difficult to incorporate these variations naturally in the closed-form formula, whereas they can easily be added into the iterative version by simply adjusting the loop body.

This can result in floating-point inaccuracies for exponentiation over long periods of time, especially with large n or for high-precision financial calculations. The iterative solution, updating the balance incrementally at each year, will have more stable numerical behavior and be less prone to rounding errors. This iterative solution is also more maintainable from a software engineering perspective: easier to read, test, debug, and validate, since it provides an intuitive implementation of the process-one period at a time.

In all, while the mathematical shortcut has a better asymptotic bound of O(log n), the iterative O(n) version provides clearer logic, better extensibility, and more accurate long-term simulation. Thus, linear-time versions are also practical and appropriate for real-world financial computation.