Analysis of Algorithm

(CIT3003)

Retirement Investment Optimisation Using Algorithmic Design

Julia Williams (2305618)

Mr. Oral Robinson

November 21, 2025

Table of Contents

Problem Identification and Description

The **variableInvestor()** function models a more realistic and dynamic retirement investment scenario where interest rates fluctuate over time. Unlike fixedInvestor(), which assumes a constant growth rate, variableInvestor() acknowledges that real-world financial markets do not grow steadily. Instead, investment performance is influenced by multiple macroeconomic and microeconomic factors such as inflation, recessions, changing government policies, geopolitical instability, corporate earnings, currency fluctuations, and investor sentiment.

In this problem, the investor begins with an initial principal and experiences different annual interest rates each year, represented as a list called rateList. Each value in this list corresponds to the growth (or decline) in a specific year. Rates may be positive (indicating investment gains) or negative (representing market losses), giving the function the ability to simulate periods of economic expansion as well as downturns such as recessions or stock market crashes.

The fundamental operation of the function is iterative compounding based on yearly rates:

Formula = Balancei+1=Balancei×(1+rateList[i])

This sequential update reflects the way retirement portfolios actually behave—returns in one year form the basis for the next year's growth. For example, a negative return early in retirement can significantly reduce long-term wealth due to the loss of compounding potential, a phenomenon known as sequence-of-returns risk. Similarly, strong early gains can substantially increase future balance even if later years perform poorly.

Relevance to Retirement Planning

This function is important because:

- Financial markets rarely grow at a fixed rate
- Retirement savings depend heavily on unpredictable yearly changes
- It allows simulation of historical market data
- It supports stress-testing retirement portfolios under different economic conditions
- It prepares the ground for later tasks such as depletion simulation and binary-search-based optimization

Inputs

- **principal:** Initial amount invested
- **rateList:** A list/array of annual return rates (e.g., {0.12, -0.08, 0.05, 0.03})

Output

- The final accumulated balance after applying each year's rate in sequence.

Assumptions

- The length of rateList equals the number of years being simulated.

- Values in rateList may be positive, zero, or negative.

- Compounding occurs once per year, consistent with standard retirement models.

- All inputs are valid numeric entries.

- No deposits or withdrawals occur during the growth period unless later extended.

- The same rateList produces the same result each time.

Constraints

- The algorithm must handle negative growth safely.

- principal must be non-negative.

- rateList cannot be null or empty.

Algorithmic Classification

The algorithm is:

- **Iterative**: processes rateList sequentially

- **Polynomial-time**: runtime depends on the number of years

If n = number of rates:

- One loop over n elements

- Constant-time operations per iteration

Thus:

$T(n) = O(n)$

Space Complexity:

Only a single balance variable is used:

$S(n) = O(1)$

Class:

variableInvestor()∈PvariableInvestor() \in PvariableInvestor()∈P

It is not NP-hard, not exponential, and contains no branching complexity.

Algorithmic Design and Representation

Pseudocode

```
FUNCTION variableInvestor(principal, rateList):
    balance = principal
    FOR each rate in rateList DO
        balance = balance × (1 + rate)
    END FOR
    RETURN balance
END FUNCTION
```

Design Justification

Step 1 - Initialization of the investment balance

balance = principal

Step 2 - Sequential iteration through the rate list

FOR i = 0 TO length(rateList) - 1 DO

Step 3 - Apply yearly compounding

balance = balance × (1 + rate)

Step 4 - Final output

RETURN balance

Proof of Correctness

We must show that after applying all rates, the balance equals:

$P \times \Pi (1 + r_i)$ from $i=0$ to $n-1$

Base Case (n=0):

If the rate list is empty:

Balance = P

Which matches the mathematical definition: an investment with zero years growth remains the same.

Assumption:

Assume after k years:

Balance$_k$ = $P \times \Pi (1 + r_i)$ from $i=0$ to $k-1$

Inductive Step:

For year k+1, the algorithm applies:

Balance$_{k+1}$ = Balance$_k$ $\times (1 + r_k)$

Substitute the hypothesis:

Balance$_{□+1}$ = P × Π (1 + r_i) from i=0 to k

Thus the formula holds for k+1.

Therefore, by induction, the algorithm is correct.

Asymptotic Efficiency Analysis

Time Complexity

Each rateList element is processed once:

$T(n)=O(n)$

Where n = number of years.

Space Complexity

$S(n)=O(1)$

Trade-off Discussion

While a closed-form product approach is possible, processing year-by-year:

- allows for negative rates
- avoids floating-point overflow
- aligns with simulation-based modeling used in finance
- is extensible (fees, inflation, taxes, contributions can be added)

Thus, the O(n) simulation is optimal for algorithmic accuracy and clarity.