# Coding Test Follow-Up: Maclaurin Binomial Series Calculation Optimization

## Background:

Let's talk about Compound interest rates. In most banking system that handles compound interest, they will use a simple mathematical formula to calculate this value. See below:

$$ToRepayAmount = P * (1 + x)^{t}$$

Whereas:

- Compounding frequent is daily.
    - See explanation here: https://en.wikipedia.org/wiki/Compound_interest#Calculation
- P is the principal amount borrowed at Day 0. Let's make it $10000 for a simplified case.
- X is the simple interest rate, accrues on a daily basis. For example, for a daily interest rate of 0.05% loan, the x here should be 0.0005
- T is the length of time of such a loan being outstanding, measure in days. For example, for a 90 days loan, it should be 90.

In the above example, the total amount to be paid on due should be:

$$ToRepayAmount = \$10000 * (1 + 0.0005)^{90} = \$10460.16096082...$$

In the underlying floating number calculation procedure, most computers will use a Taylor series to expand this exponential calculation into an infinite sum of binomial series. The Taylor series of function $f(x) = (1 + x)^{\alpha}$ is:

$$\begin{aligned}(1+x)^{\alpha} &= \sum_{k=0}^{\infty} \binom{\alpha}{k} x^k \\ &= 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!}x^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!}x^3 + \cdots,\end{aligned}$$

In Solidity, since we are unable to handle any floating numbers, we have to make a minor transformation on the above equation:

$$ToRepayAmount = P * (1 + \frac{1}{x})^{\frac{a}{b}}$$

Whereas:

- Compounding frequent is continuously
- 1/x is the interest rate, that is compounded on a yearly basis. For example, if we want a 50% compounded annualized interest rate, we will set x here to 2 to make it ½ (50%)
- a/b is the fraction term of length of time. In the above example setting, if we need to calculate the 90th day accrued interest, we will set a = 90 and b = 365 to make it 90/365.

Using the above transformation, we can ensure that all variables here (x, a, and b) are all integers and can be represented as uint256 in Solidity.

## Material:

A simple contract MaclaurinBinomial.sol is implemented in the files you have received. The testing contract MBTest.sol is to call this library for exponential number calculation. Run the test.js in scripts to test the whole contract using this command: **yarn hardhat run scripts/test.js**

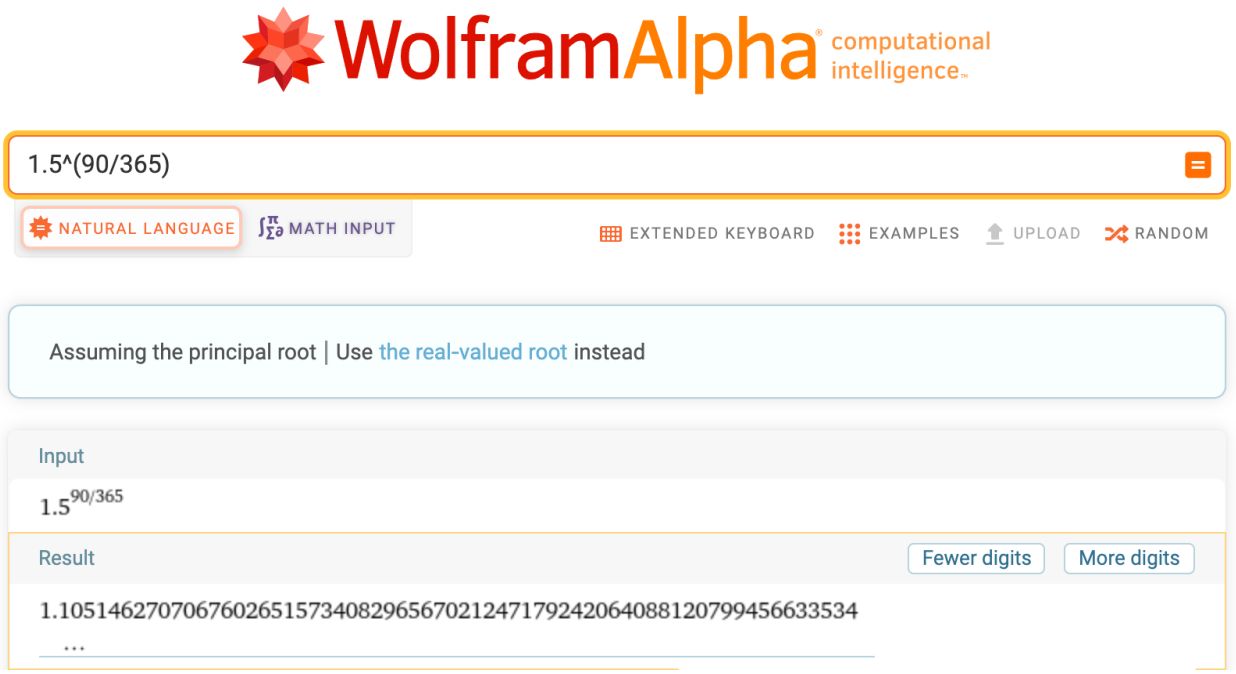
There are a few parameters for the calculation you need to take care about:

```
const decimal = "1000000000000000000";    // 18 decimals
const reciprocalOfInterestRate = 2;       // the x parameter
const a = 90;
const b = 365;
const precision = 18;                     // how precise the result is
                                          // higher precision will
                                          // cost more gas
```

By using the default setting you should be able to get this result:

```
➜  MaclaurinSeries yarn hardhat run scripts/test.js
yarn run v1.22.10
MBTest deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

`Using Precision=18 Calculation Result:` **`1105146285140337992`**`; Gas Used: 220613`




Compare it with the result from the floating number calculation. We can ensure the first 8 digits (**`11051462`**) are correct. The following digits started to diverge from the actual result, this is expected since it's integer calculation on Solidity so the precision loss.

## Assignment:

**** Download ZIP here:**
**https://www.dropbox.com/s/dcwrfdeov9uxz3m/CodingTest.zip?dl=0 ****

1. Please try to optimize the maclaurinBinomial() function to reduce gas cost. Given the same **`Precision,`** the gas cost should be equal for all possible inputs (x, a, and b)\
    a. If you can implement any code that actually reduces the gas cost, write it and test using the test.js
    b. If you don't intend to write code, give some thoughts about how to implement this in a viable way.
    c. All intellectual outcomes will be considered. Whatever code, docs, experiments, or just thoughts.
2. If the call reverted with error: SafeMath: multiplication overflow, please lower the **`Precision.`** It's good to use the 10 or 12, most inputs will pass.