2. For lower values of n (2 and 5), the reconstructed value of x after multiplying x by itself n times matches the original value of x (100) approximately. As n increases (10, 20), the reconstructed value starts to deviate slightly from the original value. This deviation is likely due to the limited precision of single-precision floating-point arithmetic. For larger values of n (30, 40), the reconstructed value of x becomes significantly different from the original value. It approaches 1, indicating that the precision loss and rounding errors accumulated during the recursive square root calculations cause the loss of significant digits. The reconstructed value of x becoming 1 after 30 and 40 iterations indicates that the loss of precision and rounding errors eventually lead to a value close to 1, effectively losing information about the original value of x.

3. Calculating z1 as (e^x-1)/x; I observed x approaches 0 and z1 approaches 1 but may deviate slightly due to numerical precision.L'Hôpital's rule states that for functions f(x) and g(x) such that lim x→a f(x) = lim x→a g(x) = 0 or ±∞, if f'(x) and g'(x) exist and f'(a) ≠ 0, g'(a) ≠ 0, then:
lim x→a f(x)/g(x) = lim x→a f'(x)/g'(x).
As x approaches 0, both (e^x - 1)/(x) and log(x)/log(e^x) tend to 0. Applying L'Hôpital's rule, you can differentiate the numerator and the denominator, which simplifies the limit to 1.
Both approaches should converge to 1 as x approaches 0, but due to numerical precision, there are some deviations in the output. When comparing approaches: the second approach (e^x - 1)/(log(x)/log(e^x)), tends to give results closer to 1 for smaller values of x compared to the first approach. As x becomes smaller, the logarithmic function log(x)/log(e^x) approaches 0 more slowly than x, which contributes to a more stable computation of Approach 2.

4. The findings indicate that as the number of terms N in the series increases, both the forward and backward summations approach the analytical solution of log(2) / log(e) (2).For the forward summation, as N increases from 10^3 to 10^9, the forward sum converges to approximately 0.69313753. The absolute error relative to the analytical solution decreases rapidly as N increases, with the error ranging from about 9.65 × 10^(-6) for N = 10^6 to 1.90 × 10^(-9) for N = 10^9. The backward sum also converges to approximately 0.69314718 as N increases. The absolute error decreases with increasing N, showing a similar trend to the forward summation, but the errors are slightly smaller for each N. Overall, changing the order of summation (from left to right or from right to left) does not significantly affect the numerical convergence of the series. Both forward and backward summations converge to the analytical solution of log(2) / log(e) (2) as the number of terms increases. The errors decrease rapidly as N increases, indicating that the series converges to the analytical solution accurately for large N.

5. The estimateExponent function interprets the double-precision floating-point number f as a uint64_t using reinterpret casting. It extracts the biased exponent stored in bits 52 to 62 of the IEEE 754 double-precision format and converts it to a real exponent by multiplying by log(2) / log(10). The main function,tests the estimation method for different numbers, and calculates the actual exponent using the logarithm base 10 of the absolute value of the number. Finally it prints the number, the estimated exponent, the actual exponent, and their absolute difference.

This method provides a computationally cheaper way to estimate the exponent compared to directly computing log10(|f|). The results will show how well the estimation method performs compared to the more expensive computation.