Multiplying Large Integers Using FFT and the Schoenhage-Strassen Algorithm

1. Multiplying Integers via FFT

To multiply two large integers efficiently, we can represent them as polynomials and use the Fast Fourier Transform (FFT).

Step-by-step:

1. Convert integers to polynomials:
   Represent integers in base B, so that an integer becomes a polynomial in B. For example:
   $1234 = 1*B^3 + 2*B^2 + 3*B + 4$

2. Use polynomial multiplication:
   Multiplying two integers becomes equivalent to multiplying two polynomials.

3. FFT-based multiplication:
   - Evaluate both polynomials at n-th complex roots of unity using FFT.
   - Pointwise multiply the results.
   - Use Inverse FFT to reconstruct the result polynomial.
   - Convert the result polynomial back to an integer.

The time complexity becomes $O(n \log n)$, which is much faster than the traditional $O(n^2)$ method for large n.

2. Problems With Using FFT for Integers

While the FFT method works well in theory, practical implementation with complex numbers introduces problems:

- Floating-point precision errors
- Carry handling after inverse FFT
- Numerical instability for very large numbers

These issues make purely complex FFT unsuitable for very large integer multiplication.

3. How the Schoenhage-Strassen Algorithm Solves This

The Schoenhage-Strassen algorithm avoids floating-point issues by working entirely within modular arithmetic.

Key ideas:

- Replace complex roots of unity with modular roots of unity.
- Work in rings like $Z/(2^k + 1)Z$, where arithmetic is exact and efficient.
- Use negacyclic convolution instead of regular cyclic convolution.

This ensures safe and efficient FFT-like operations with full precision.

4. The Role of Roots of Unity and Modular Arithmetic

Complex Roots of Unity:

A complex number w is a primitive n-th root of unity if $w^n = 1$ and $w^k \neq 1$ for $0 < k < n$.

Modular Analogs:

In Z/mZ, we want w such that:

$w^n \equiv 1 \mod m$ and $w^k \not\equiv 1 \mod m$ for $0 < k < n$

These enable the Number Theoretic Transform (NTT), a modular analog of the FFT.

Why It Works:

- $(Z/mZ)^*$ is cyclic if m is prime.
- If n divides $(m - 1)$, then an element of order n exists.
- Modular arithmetic mimics root-of-unity structure algebraically and exactly.

5. Choosing the Right Modulo and Its Benefits

To use NTT efficiently:

- Choose m such that n divides $(m - 1)$
- Use $m = 2^k + 1$ (Fermat-type moduli) for speed

Schoenhage-Strassen specifics:

- Use $m = 2^{2^k} + 1$
- Use negacyclic convolution with $w^n = -1$

This allows multiplication of integers with millions of digits in $O(n \log n \log \log n)$ time.

Summary

- FFT speeds up large integer multiplication, but suffers from floating-point issues.
- Schoenhage-Strassen uses modular arithmetic to fix this.
- Modular roots of unity provide safe and efficient FFT analogs.
- Moduli like $2^k + 1$ allow fast binary operations.

This method underlies modern cryptographic and big-number libraries.