NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

"IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE"

INSTITUTE OF PHYSICS AND TECHNOLOGY

### DEPARTMENT OF MATHEMATICAL METHODS OF INFORMATION SECURITY

**Lab report №1**

Bignum arithmetic (multiple-precision) implementation over finite fields and groups

**Done by:**

Fifth-year students

FI-73m

Oleksandr Bohutskyi

Maryna Soloviova

Olesia Stolova

**Checked by:**

Fesenko A.V.

**КYIV-2018**

**Purpose:** The study of algorithms for implementing arithmetic operations over large (multi-digit) numbers over finite fields and groups in terms of their efficiency over time and memory for various software and hardware environments.

**Tasks**

**Group 1.** To study the time and memory efficiency of many bignum arithmetic algorithms. Perform a comparative analysis of the complexity of multiplication algorithms.

**Subgroup 1V.** Computation model - processors with 32-bit architecture and the amount of RAM up to 4 GB. Example - workstations.

**Group 2.** Develop a library of long-range arithmetic for Intel-compatible PCs. The dimensionality of the numbers - 768, 1024 bits, and for elliptic curves - 163 bits.

**Subgroup 2A.** Orientation to the implementation of high-speed algorithms for multiplication calculation. Mandatory implementation of the Karatsuba-Ophman method.

**Group 3.** Explore standard libraries of long-range arithmetic. Subgroup 3A. PGP 2.3 (or PGP 6.5.2.).

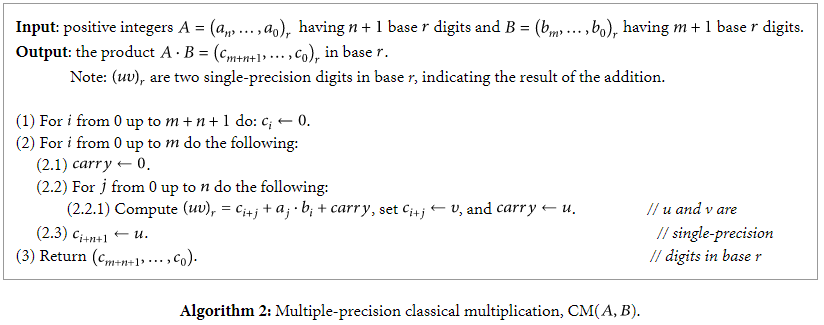
**Subgroup 3B**. Built-in arithmetic of the Java system.

1. **Multiplication Algorithms**

The most well-known algorithms for multiplication of two large integers or two polynomials are classical, Karatsuba-Ofman’s, Toom-Cook’s, and fast Fourier transform (in particular Schonhage-Strassen) multiplication algorithms. In spite of all the differences in these methods, which sometimes make them apparently unrelated to each other, these methods have been founded based on the same idea, that is, how to represent a polynomial to behave efficiently in calculations.

**Classical Multiplication**

In positional numeral system, the natural way of multiplying numbers, known as classical multiplication algorithm, is by multiplying each digit of the multiplicand by each digit of the multiplier and then adding up all the properly shifted results. This method requires a multiplication table for single digits available to the algorithm.



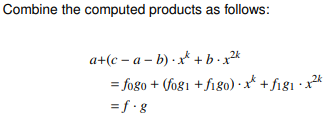
The complexity of the classical multiplication algorithm is O(n2) . Therefore, the number representation that has fewer digits theoretically should run faster than the number representation that has more digits in its representation. In addition, the density of nonzero digits in the numbers influences the number of addition that has to be carried out by the classical multiplication algorithm as well.

##### **Karatsuba Multiplication**

Karatsuba’s algorithm is an efficient scheme for multiplying two large numbers or two polynomials. It was introduced by Karatsuba and Ofman in 1960 and published in 1962. This algorithm is a remarkable example of the divide and conquer paradigm, specifically for its binary splitting. This method requires three multiplications and four additions in each iteration.







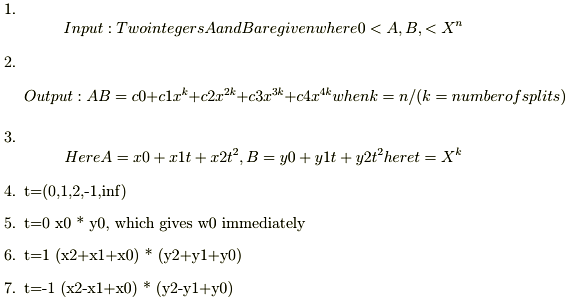
The Karatsuba algorithm was the first multiplication algorithm asymptotically faster than the quadratic "grade school" algorithm. The [Toom–Cook algorithm](https://en.wikipedia.org/wiki/Toom%E2%80%93Cook_multiplication" \o "Toom–Cook multiplication) is a faster generalization of Karatsuba's method, and the [Schönhage–Strassen algorithm](https://en.wikipedia.org/wiki/Sch%C3%B6nhage%E2%80%93Strassen_algorithm" \o "Schönhage–Strassen algorithm) is even faster, for sufficiently large *n*.

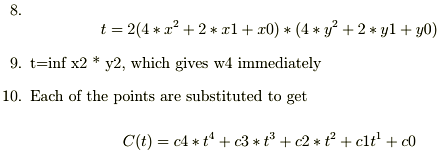
**Toom-Cook**

Toom Cook algorithm is developed by Andrei Toom in 1963 and is later improved and published by Stephen Cook in his Phd thesis. Toom Cook Algorithm is also referred as Toom 3 which is the collective name for all Toom Cook based algorithms. Toom Cook is the faster generalisation of the Karatsuba method.

A three-way Toom–Cook can do a size-3N multiplication for the cost of five size-N multiplications, improvement by a factor of 9/5 compared to the Karatsuba method's improvement by a factor of 4/3.

Unllike Karatsuba it deals with 3 parts rather than 2 parts which makes it even more complex.





Although using more and more parts can reduce the time spent on recursive multiplications further, the overhead from additions and digit management also grows. For this reason, the method of Fourier transforms is typically faster for numbers with several thousand digits, and asymptotically faster for even larger numbers.

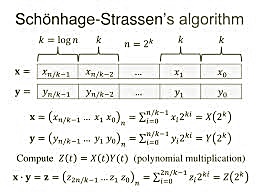
**Schonhage-Strassen Algorithm**

The Schönhage–Strassen algorithm is an asymptotically fast multiplication algorithm for large integers. It was developed by Arnold Schönhage and Volker Strassen in 1971. The algorithm uses recursive Fast Fourier transforms in rings with 2n+1 elements, a specific type of number theoretic transform.

The Schönhage–Strassen algorithm was the asymptotically fastest multiplication method known from 1971 until 2007, when a new method, Fürer's algorithm, was announced with lower asymptotic complexity; however, Fürer's algorithm currently only achieves an advantage for astronomically large values and is not used in practice.

In practice the Schönhage–Strassen algorithm starts to outperform older methods such as Karatsuba and Toom–Cook multiplication for numbers beyond  (10,000 to 40,000 decimal digits).The GNU Multi-Precision Library uses it for values of at least 1728 to 7808 64-bit words (33,000 to 150,000 decimal digits), depending on architecture.

There is a Java implementation of Schönhage–Strassen which uses it above 74,000 decimal digits.



**Fürer's algorithm**

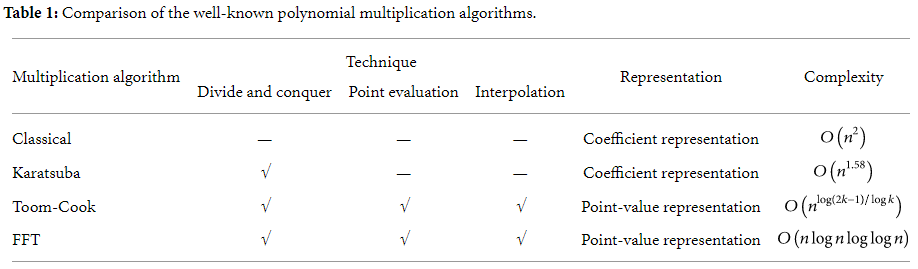
Fürer's algorithm is an [integer multiplication algorithm](https://en.wikipedia.org/wiki/Integer_multiplication_algorithm) for extremely large integers with very low [asymptotic complexity](https://en.wikipedia.org/wiki/Asymptotic_complexity). It was published in 2007 by the Swiss mathematician [Martin Fürer](https://en.wikipedia.org/w/index.php?title=Martin_F%C3%BCrer&action=edit&redlink=1) of Pennsylvania State Universityas an asymptotically faster algorithm when analysed on a multitape Turing machine than its predecessor, the [Schönhage–Strassen algorithm](https://en.wikipedia.org/wiki/Sch%C3%B6nhage%E2%80%93Strassen_algorithm" \o "Schönhage–Strassen algorithm).

The Schönhage–Strassen algorithm uses the [fast Fourier transform](https://en.wikipedia.org/wiki/Fast_Fourier_transform) (FFT) to compute integer products in time  {\displaystyle O(n\log n\cdot \log \log n)} and its authors, [Arnold Schönhage](https://en.wikipedia.org/wiki/Arnold_Sch%C3%B6nhage) and [Volker Strassen](https://en.wikipedia.org/wiki/Volker_Strassen), conjecture a lower bound of  . {\displaystyle \Omega (n\log n)}..ююю..Fürer's algorithm reduces the gap between these two bounds. It can be used to multiply integers of length {\displaystyle n}n in time {\displaystyle O\left(n\log n\ \cdot 2^{O(\log ^{\*}n)}\right)}  where [log\*](https://en.wikipedia.org/wiki/Iterated_logarithm)n is the [iterated logarithm](https://en.wikipedia.org/wiki/Iterated_logarithm). The difference between the {\displaystyle \log \log n}  {\displaystyle 2^{\log ^{\*}n}} terms, from a complexity point of view, is asymptotically in the advantage of Fürer's algorithm for integers greater than {\displaystyle 2^{2^{64}}} . However the difference between these terms for realistic values of {\displaystyle n}  n is very small.

**Differences among the multiplication algorithms**

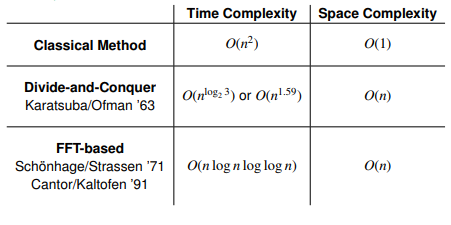
The classical method uses coefficient representation, while the other three methods use point-value representation. This representation conversion enables us to reduce the cost of convolution from of classical method to a lower cost for point-to-point multiplication. The process of finding point-value representation from its coefficient representation is called “evaluation” or “point evaluation” and the reverse process is known as “interpolation.”

Table 1 summarizes the differences among the multiplication algorithms by their complexity, technique, and representation used.



Algorithms such as FFT and Toom-Cook have lower algorithm complexity. However, because of the preprocessing overheads such as the divide and conquer, evaluation, and interpolation, the operating cost of these algorithms is actually much higher, making them useful only when the integers are extremely large. Consequently, only classical and Karatsuba multiplication algorithms and their combination are being used in current cryptosystem. This is especially true after considering circumstances such as memory constraints and the practical finite field size.

**Resume:**



1. **Program code**

Program code you can find following the link:

<https://github.com/Bohutskyi/CryptographicMechanisms.Labs>

1. **Built-in arithmetic of the Java system**

In the table below you can view measurement of how long time it takes for the each tested method to be executed, including max, min time etc. Count of iterations for each method with different random values for selected size is 20.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Used type (class) | Method name | Input size, bits | Score, us/op\* | Error |
| LongNumber (developed) | LongMul() | 1024 | 96600,064 | ± 2601,612 |
| Karatsuba() | 82997,412 | ± 8448,563 |
| SchonhageStrassenMul() | 167231,615 | ± 1635,109 |
| LongMul() | 768 | 45800,926 | ± 1458,895 |
| Karatsuba() | 36717,714 | ± 1797,139 |
| SchonhageStrassenMul() | 79792,990 | ± 694,283 |
| BigInteger | multiply() | 1024 | 0,047 | ± 0,004 |
| multiplyKaratsuba() | 0,602 | ± 0,125 |
| multiplyToomCook3() | 0,710 | ± 0,070 |
| multiply() | 768 | 0,047 | ± 0,004 |
| multiplyKaratsuba() | 0,706 | ± 0,240 |
| multiplyToomCook3() | 0,559 | ± 0,010 |
| BigDecimal | multiply() | 1024 | 0,054 | ± 0,004 |
| 768 | 0,048 | ± 0,003 |
| LargeInteger | times() | 1024 | 0,055 | ± 0,006 |
| 768 | 0,063 | ± 0,005 |

\* All results are printed in **microseconds/operation**

As you can see from results in the table, our written LongNumber multiplication methods take much more time their execution.

To summarize we have created one more table in which each method was compared (LongNumber vs. Built in classes):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input size, bits | Class / method, us/op | LongNumber | BigInteger | BigDecimal | LargeInteger |
| 678 | Simple multiplication in row / with modifications for built-in class | 45800,926 ±1458,895 | 0,047 ±0,004 | 0,048 ±0,004 | 0,063 ±0,005 |
| Karatsuba | 36717,714 ±1797,139 | 0,706 ±0,240 | - | - |
| Schönhage–Strassen | 79792,990 ±694,283 | - | - | - |
| 3-way Toom-Cook | - | 0,559 ±0,010 | - | - |
| 1024 | Simple multiplication in row / with modifications for built-in class | 96600,064 ±2601,612 | 0,047 ±0,004 | 0,054 ±0,004 | 0,055 ± 0,006 |
| Karatsuba | 82997,412 ±8448,563 | 0,602 ±0,125 | - | - |
| Schönhage–Strassen | 167231,62 ±1635,109 | - | - | - |
| 3-way Toom-Cook | - | 0,710 ±0,070 | - | - |

Want to highlight that implemented in this lab Schönhage–Strassen algorithm haven’t been found in built-in libs, so another — 3-way Toom Cook algorithm was added for comparison.

So, BigInteger multiply() method has the best performance results considering different size of inputs because it’s oriented actually on this size as threshold during making the decision which algorithm to use internally (to square OR to multiply on one-digit operand OR to use Karatsuba OR 3-way Toom Cook algorithm). That’s why, on own opinion, other methods are slower: they just have fewer options to do multiplication depends on the inputs size, even some of them implement only one algorithm to multiply its arguments.