National Research University “Higher School of Economics”

Big workshop on Algorithms and Data Structures:

Complexity of different multiplication algorithms.

Research is completed by

Gleb Zotov

DSBA 191-1

Supervisor: George Piatsky

28.04.2020

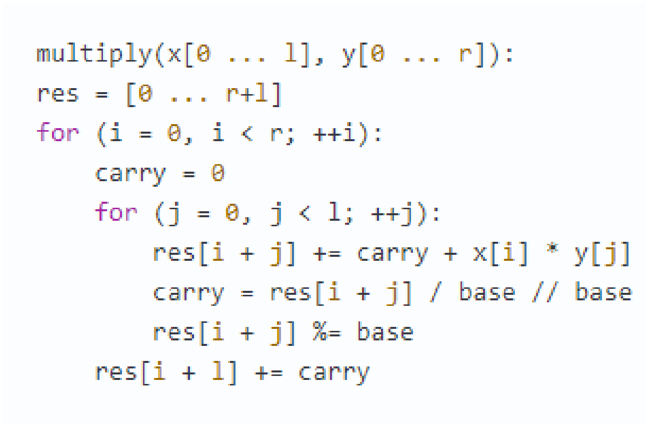
Problem statement

Introducing a problem

The aim of this research is to compare estimate the time complexity of different multiplication algorithms via experiments. There are 3 algorithms to compare: Grade School Multiplication, Divide and Conquer approach and Karatsuba algorithm. Let’s start from structure of these algorithms.

1. **Grade School Multiplication algorithm.**

It is the most simple and well-known way too multiply 2 numbers. The theoretical complexity of this algorithm is O(n^2) (according to A. Karatsuba).



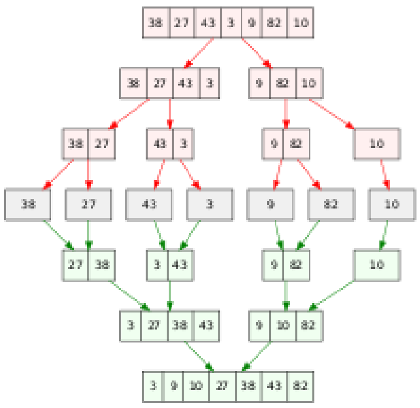
```
multiply(x[0 ... l], y[0 ... r]):
res = [0 ... r+l]
for (i = 0, i < r; ++i):
    carry = 0
    for (j = 0, j < l; ++j):
        res[i + j] += carry + x[i] * y[j]
        carry = res[i + j] / base // base
        res[i + j] %= base
    res[i + l] += carry
```

More information can be found here: https://en.wikipedia.org/wiki/Multiplication_algorithm .

2. **Divide and Conquer algorithm.**

The divide-and-conquer paradigm is often used to find an optimal solution of a problem. Its basic idea is to decompose a given problem into two or more similar, but simpler, subproblems, to solve them in turn, and to compose their solutions to solve the given problem. Problems of sufficient simplicity are solved directly.

For example, to sort a given list of *n* natural numbers, split it into two lists of about *n*/2 numbers each, sort each of them in turn, and interleave both results appropriately to obtain the sorted version of the given list.



The theoretical complexity of this algorithm is O(n^2), we discuss that on the lectures.

Here is the main idea of the algorithm:

```
if a or b has one digit, then:
    return a * b.
else:
    Let n be the number of digits in max{a, b}.
    Let $a_L$ and $a_R$ be left and right halves of a.
    Let $b_L$ and $b_R$ be left and right halves of b.
    Let $x_1$ hold Divide_and_Conquer($a_L$, $b_L$).
    Let $x_2$ hold Divide_and_Conquer($a_L$, $b_R$).
    Let $x_3$ hold Divide_and_Conquer($a_R$, $b_L$).
    Let $x_4$ holdDivide_and_Conquer($a_R$, $b_R$).
    return $x_1*10^n + (x_2 + x_3)*10^{n/2} + x_4$.
end of if
```

So, as it can be seen from the picture the algorithm consist of 4 n/2 operations with recursion calls.

3. Karatsuba Multiplication algorithm

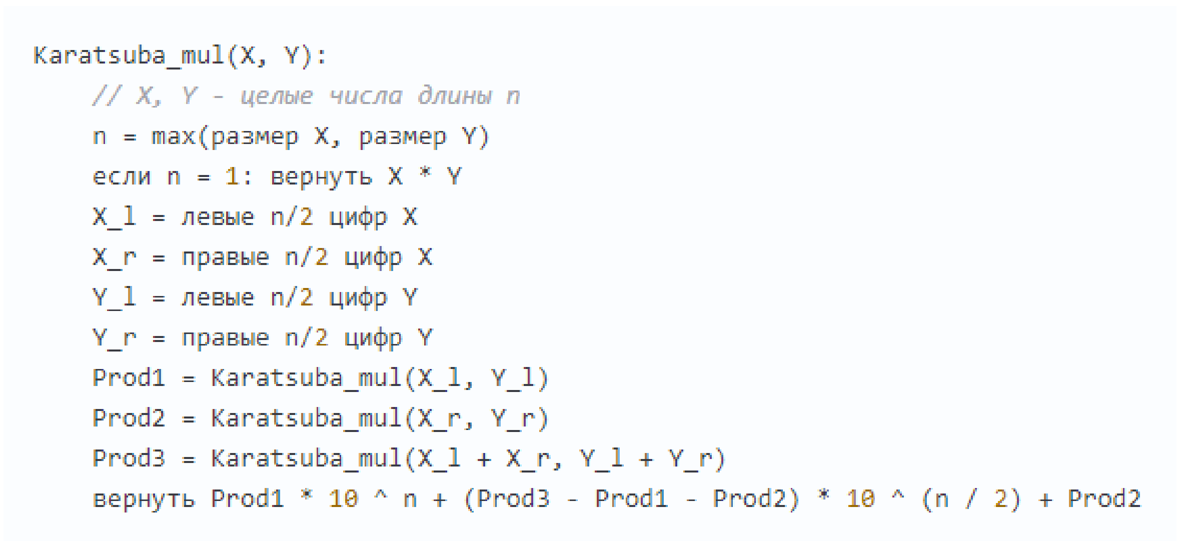
It is based on Divide and Conquer algorithm and was invented in 1962 by A. Karatsuba.

The main purpose of this algorithm is a fast multiplication of two sufficiently large numbers.

The theoretical complexity of this algorithm is approximately O(n^(1,58))

$$T(n) = O(n^{\log_2 3}), \quad \log_2 3 = 1{,}5849\ldots.$$

Implementation:



```
Karatsuba_mul(X, Y):
    // X, Y - целые числа длины n
    n = max(размер X, размер Y)
    если n = 1: вернуть X * Y
    X_l = левые n/2 цифр X
    X_r = правые n/2 цифр X
    Y_l = левые n/2 цифр Y
    Y_r = правые n/2 цифр Y
    Prod1 = Karatsuba_mul(X_l, Y_l)
    Prod2 = Karatsuba_mul(X_r, Y_r)
    Prod3 = Karatsuba_mul(X_l + X_r, Y_l + Y_r)
    вернуть Prod1 * 10 ^ n + (Prod3 - Prod1 - Prod2) * 10 ^ (n / 2) + Prod2
```

It is quite similar to the D&C algorithm but the main contrast here is the number of the recursive calls. Anatoly Karatsuba noticed that one recursive call is irrelevant and can be replaced by using the formula: ad + bc = (a + b)(c + d) – ac – bd.

Problem

Here I want to clarify all steps of the problem:

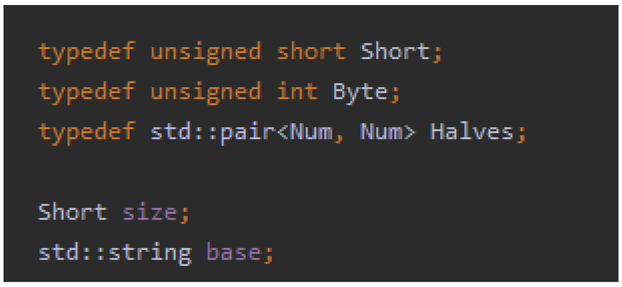
1. I have to create class of a large number **Num**, using vector or string, and implement functions and operators there.
2. I have to create a class **Multiplicator**. The class allows calculating a product of two integer numbers by applying different multiplication algorithms. We are allowed to use C++ built-in multiplication for doing one-digit multiplications (eg, 3 times 5) but not any other multiplication (eg, 23 times 55). We are allowed to use C++ built-in addition (even of large numbers)
3. Create a method in the **Multiplicator** that generates a random number of k-digits, where k is a parameter of the method.
4. In **Multiplicator** class I need create auxiliary methods needed to implement the Grade School Multiplication algorithm, the Divide-and-Conquer Multiplication algorithm and the Karatsuba algorithm.
5. Implement the Grade School Multiplication algorithm, the Divide-and-Conquer Multiplication algorithm and the Karatsuba algorithm as methods of **Multiplicator** class.
6. Create a method that performs calculation of a series of numbers containing from 1 to k digits by applying both algorithms.
7. Create a method that outputs the experimental results stored in the vector as a CSV file containing exactly 3 columns: size of input data and resulting time for all algorithms.
8. Based on the produced file, line charts are drawn by using Python's Matplotlib.

Implementation

1. The first step is a creation of the **Num** class. And I decide to realize the big number via string.

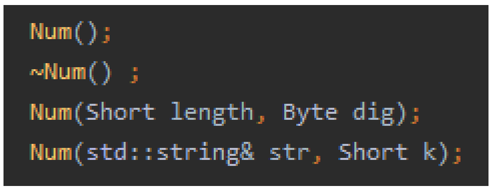
Class **Num:**

1) type of the variables



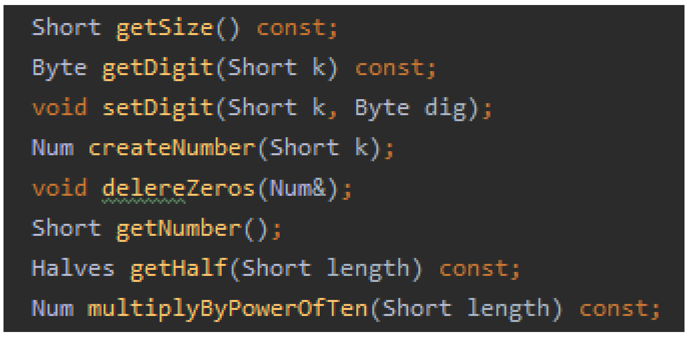
```
typedef unsigned short Short;
typedef unsigned int Byte;
typedef std::pair<Num, Num> Halves;

Short size;
std::string base;
```

2) constructors and destructor:



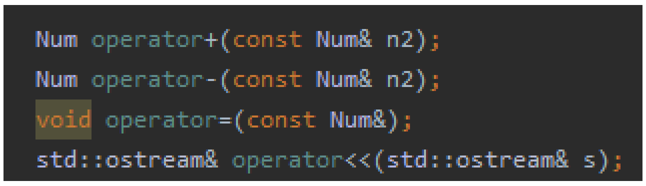
```
Num();
~Num() ;
Num(Short length, Byte dig);
Num(std::string& str, Short k);
```

3) functions:



```
Short getSize() const;
Byte getDigit(Short k) const;
void setDigit(Short k, Byte dig);
Num createNumber(Short k);
void delereZeros(Num&);
Short getNumber();
Halves getHalf(Short length) const;
Num multiplyByPowerOfTen(Short length) const;
```

4) operators:

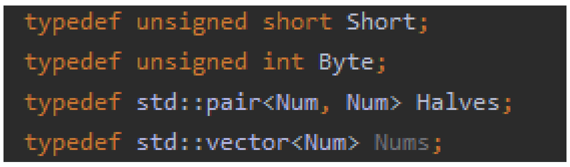


```
Num operator+(const Num& n2);
Num operator-(const Num& n2);
void operator=(const Num&);
std::ostream& operator<<(std::ostream& s);
```

2. Secondly, I have to create class **Multiplicator** with 3 subclasses for each algorithm. Also, virtual function **multiply** should be implemented here and each of the algorithm class.

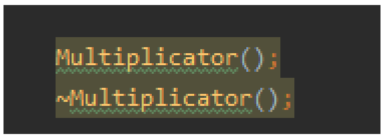
Class **Multiplicator:**

1) type of the variables



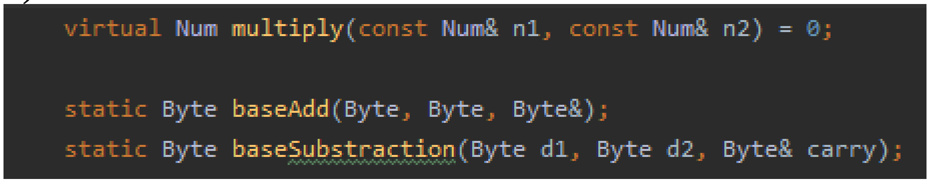
```
typedef unsigned short Short;
typedef unsigned int Byte;
typedef std::pair<Num, Num> Halves;
typedef std::vector<Num> Nums;
```

2) constructors and destructor:



```
    Multiplicator();
    ~Multiplicator();
```

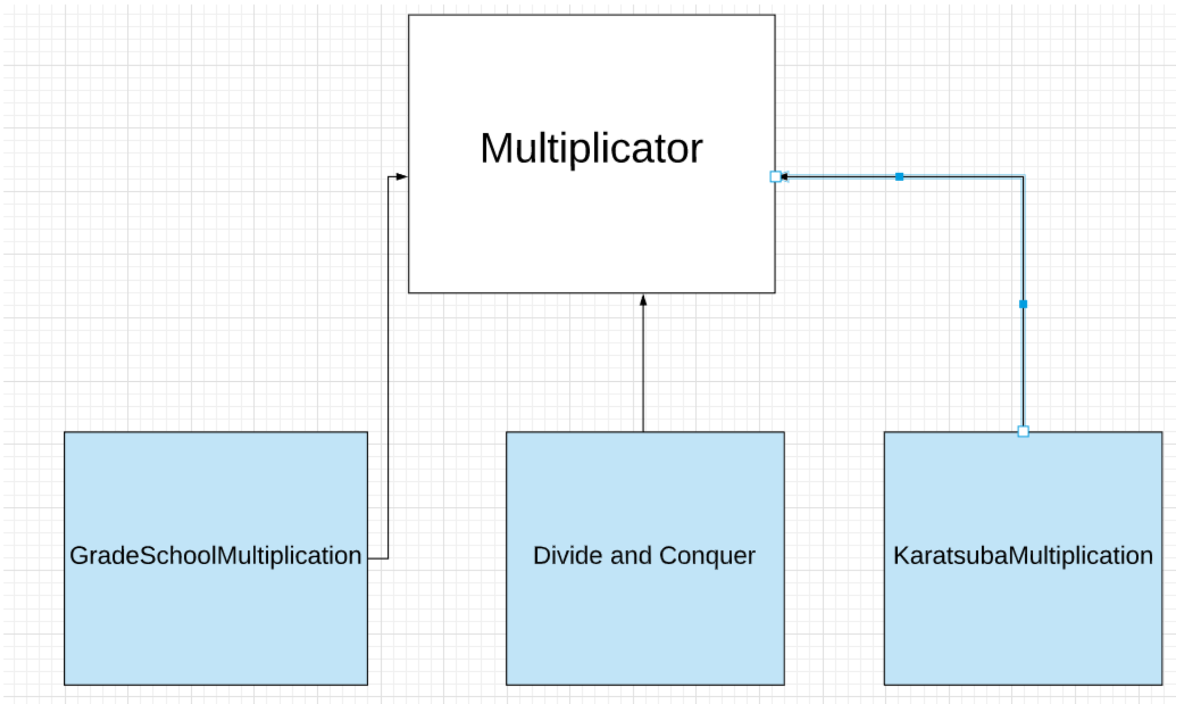
3) virtual function and function for add and substraction



```
    virtual Num multiply(const Num& n1, const Num& n2) = 0;

    static Byte baseAdd(Byte, Byte, Byte&);
    static Byte baseSubstraction(Byte d1, Byte d2, Byte& carry);
```

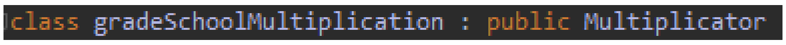
3. I have to create subclasses of **Multiplicator** which are the algorithms classes. Also virtual function **multiply** have to be implemented here.

Scheme:



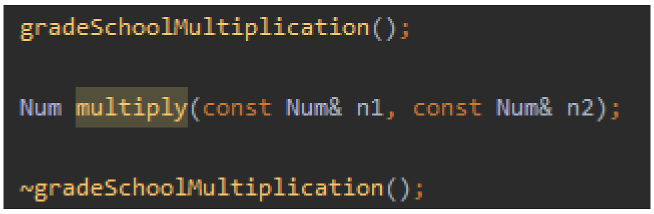
Class **gradeSchoolMultiplication**:

1) Subclass of **Multiplicator**



```
class gradeSchoolMultiplication : public Multiplicator
```

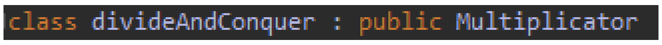
2) Constructor + multiply + destructor



```
gradeSchoolMultiplication();

Num multiply(const Num& n1, const Num& n2);

~gradeSchoolMultiplication();
```

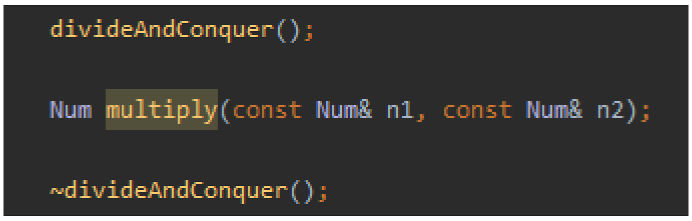
Class **divideAndConquer**:

1) Subclass of **Multiplicator**



```
class divideAndConquer : public Multiplicator
```

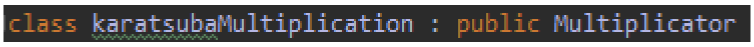
2) Constructor + multiply + destructor



```
divideAndConquer();

Num multiply(const Num& n1, const Num& n2);

~divideAndConquer();
```

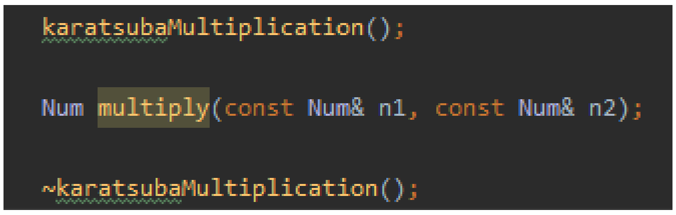
Class **karatsubaMultiplication**:

1) Subclass of **Multiplicator**



```
class karatsubaMultiplication : public Multiplicator
```

2) Constructor + multiply + destructor



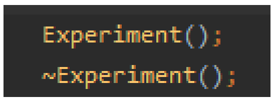
```
karatsubaMultiplication();

Num multiply(const Num& n1, const Num& n2);

~karatsubaMultiplication();
```

Obviously, implementation of the function **multiply** depends on the algorithm. And all functions, operators and conrtusctors/destructor are implemented in the .cpp files and I see no reason to add them here.

4. I need a class which combine all these together and provide an experiment itself.

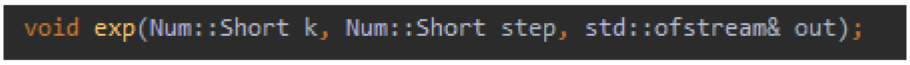
Class Experiment:

1) Constructor + destructor



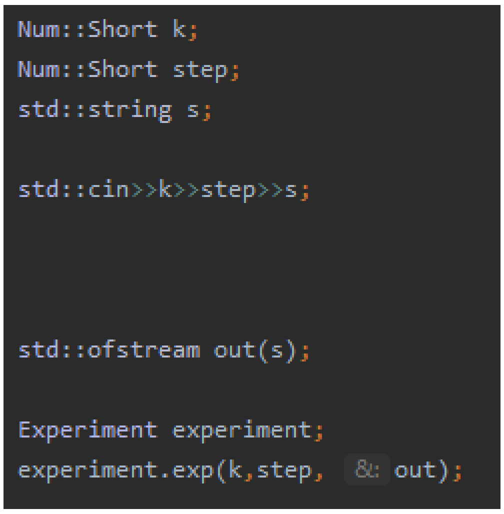
```
Experiment();
~Experiment();
```

2) Function exp which provide an experiment from the beginning (get the limit (k), step(step) and the CSV file to out), then it conduct an experiment with 3 algorithms, and finally output it into the file.



```
void exp(Num::Short k, Num::Short step, std::ofstream& out);
```

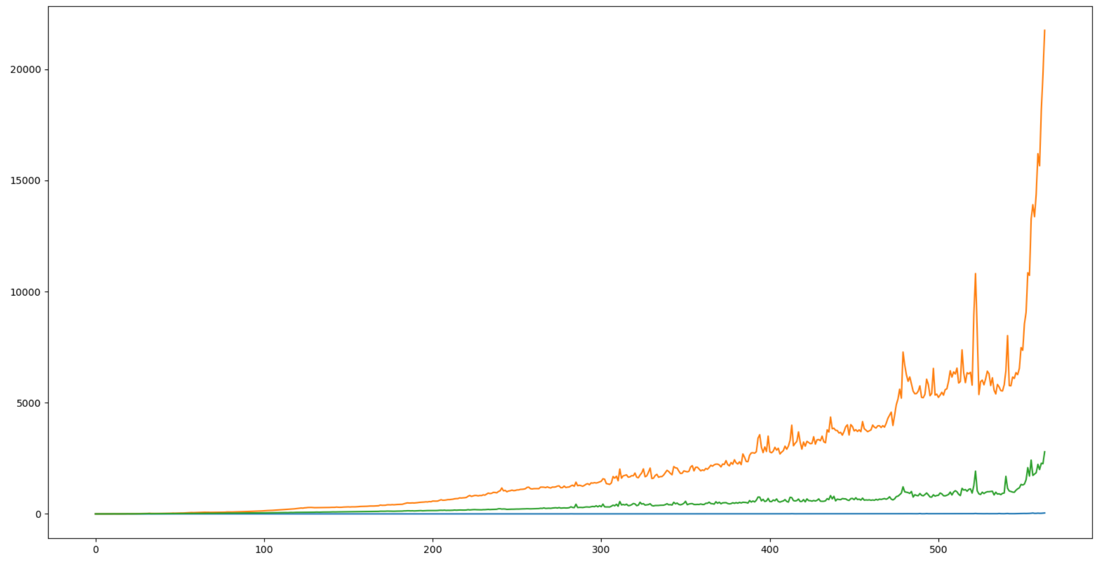
5. The main part. Here program ask for the **input** and then call the function **exp**.



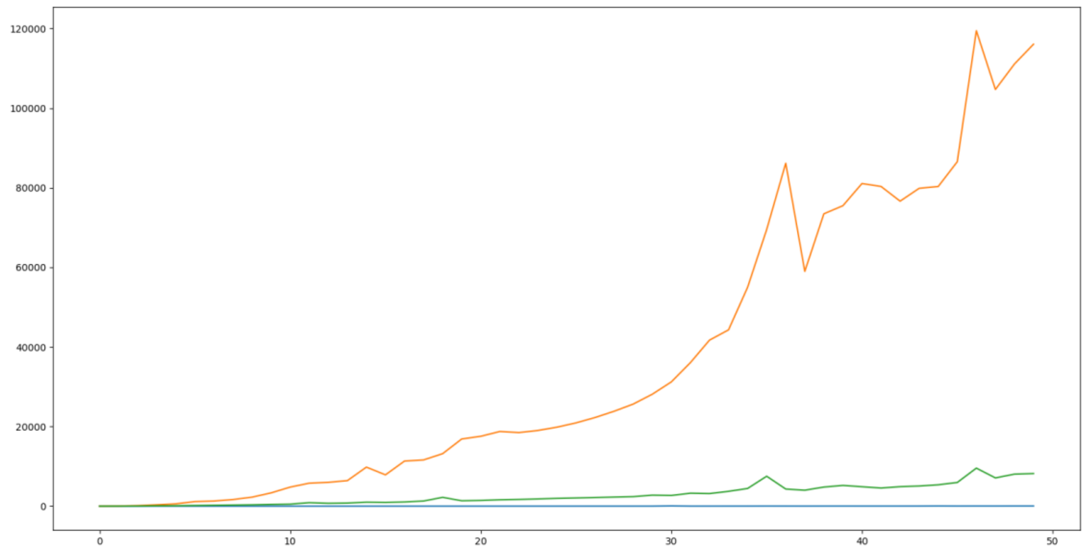
```
Num::Short k;
Num::Short step;
std::string s;

std::cin>>k>>step>>s;



std::ofstream out(s);

Experiment experiment;
experiment.exp(k,step, &: out);
```

Results

1. In fact my school algorithm is too fast and on the graph (blue line) to the 500 with the step 1 it is impossible to see the difference. But the difference between D&C (orange) and Karatsuba (green) algorithms is quite noticeable.

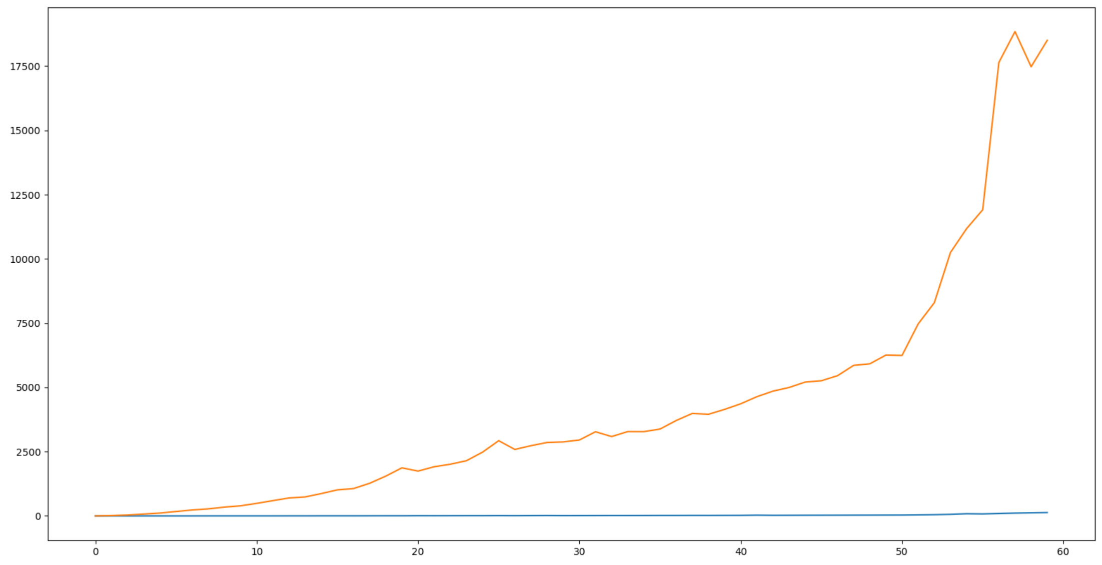


2. Now let’s look at the graph to the 5000 with step 100



As it can be seen Karatsuba algorithm is quite fast then D&C

3. But let’s check Grade School and Karatsuba algorithms. Here I give the input from 1 to 10000 with step 100.



So, in fact Karatsuba should overrun the Grade School on the considerably large numbers, but as I have mentioned before, my Grade School algorithm is too fast, I believe it's related with string type of a number.

I take some big numbers with 10.000, 20.00 , ..., 70. 000 and on such a big numbers Karatsuba become working faster, while Grade School Multiplication is working slowly. But I for some reason computing multiplication on such a big numbers take a lot of time and I can just guess that according to progression Karatsuba will overrun Grade School Multiplication algorithm approximately at 100.000 – 150.000 number of digits.

Sources used

- https://en.wikipedia.org/wiki/Multiplication_algorithm
- https://habr.com/ru/post/262705/
- https://drive.google.com/file/d/1xzKmigw4mwtjbV-vnZEUB_bHgVDKY_mh/view
- https://en.wikipedia.org/wiki/Karatsuba_algorithm