«National Research University

«Higher School of Economics»

**FCS DSBA**

# “Algorithms and Data Structures. First Big Homework Report”

*Accomplished by Oleg M. Malchenko*

*The student of DSBA 193-1*

Workshop presenter:

Aleksey B. Varenikov

**Moscow 2020**

## Problem Statement

We were given a task to implement three algorithms, namely:

- Grade School Multiplication Algorithm (further be named as GSM)
- Divide and Conquer Multiplication Algorithm (further be named as DNC)
- Karatsuba Multiplication Algorithm (further be named as KMA)

We knew that the following should hold for given three algorithms:

The time complexity of GSM is given by: $T(n) = O(n^2)$, as far as we have to sift through each digit of the first number given and multiply it (digit-wise) by the second number given (which is a cycle with an inner cycle (for(for())).

The time complexity of DMC is given by: $\mathrm{T(n)} = 4\mathrm{T}(\frac{n}{2}) + \mathrm{O(n)}$, (as far as we substantially decompose the task by splitting it into four sub-problems of size $\frac{n}{2}$, and continue to split the problem recursively until getting the problem of base size) which by thorough estimation which would not be provided here is to be $O(n^2)$.

The time complexity of KMA is given by $T(n) = 3T\left(\frac{n}{2}\right) + O(n)$, as we now decompose the problem into three sub-problems by performing a smart trick ( AD + BC ) == ( (A + B)(C + D) – AC – BD). So, again, by thorough estimation time complexity is to be $O(n^{\log_2 3})$.

Thereof we have had to understand whether the realization of these algorithms in reality proves the given time complexities are precisely determined.

## Implementation Details

Realization of these algorithms in our case was done in the Microsoft Visual Studio application. The attempt was to create two different classes:

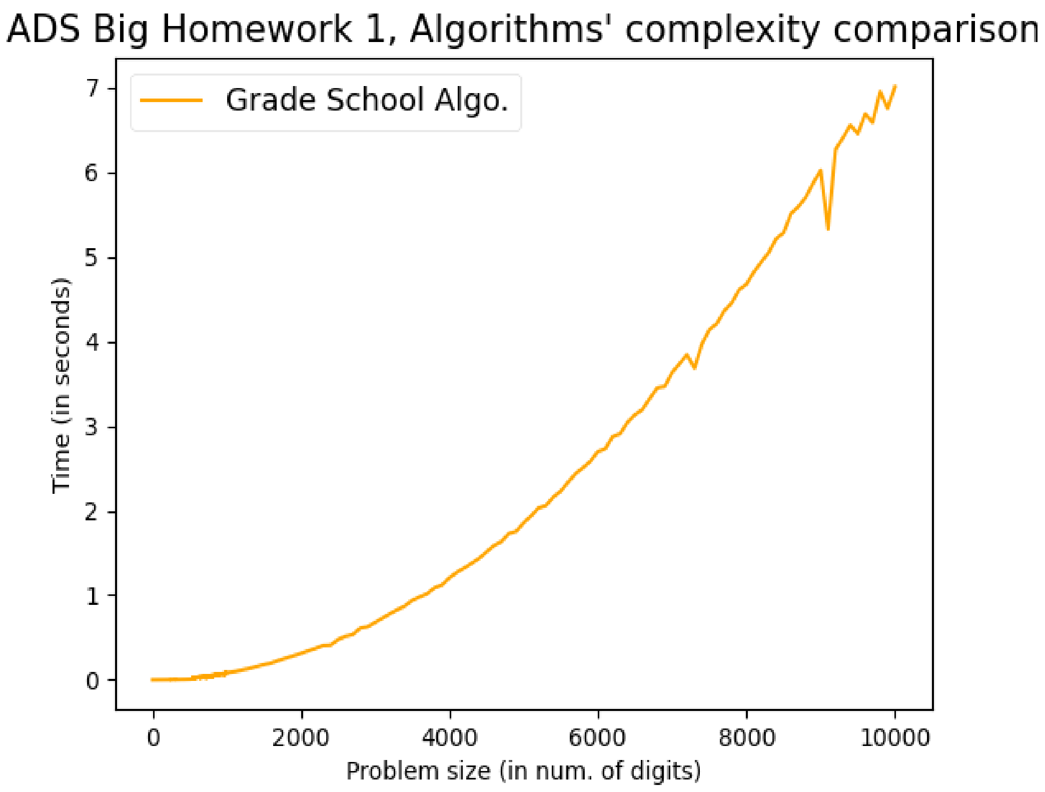
- `class Number`, in order to represent the numbers, consisting of a any number of digits. The private member of `Number` is the std::string element: _data, containing the number of arbitrary length, represented as a string. In `Number` class, several methods were implemented, the most important ones of which were: `static void makeEqual(Number& num, size_t n),` function, which inserts $n$ insignificant zeros to the left part of a number, in order to make its size equal to the size of the “longer” number, and `Number& zeroPowerInserter(size_t i, size_t n)`, which inserts $n$ zeros to the right part of the number, what literally means multiplying the number by $10^n$, and `std::pair<Number, Number> split(int n) const`, in order to be able to split the number into two numbers of (almost) equal size. Also, several operators were overloaded, namely: operator+, operator-, operator+= and operator<<, so that it became easier to implement the given algorithms.
- `class Multiplicator`, the abstract class, with the only pure virtual function: `virtual Number product(Number num1,Number num2) = 0,` which was further overloaded in three derived classes:
    1) `class GradeSchoolMult : public Multiplicator`
    2) `class KaratsubaMult : public Multiplicator`

3) `class DNCMult : public Multiplicator`

- `GradeSchoolMult`

  The approach to implement the GSM algorithm was quite straightforward. Creating if-statement, returning zero for cases, when the length of one of two given numbers is zero, or if one number of two is exactly zero, and else-statement, in which the main body of the algorithm was to be situated. The algorithm has two cycles in the main body, each iterating through the reverse one of two numbers given to be multiplied. Each digit of the first of two numbers is multiplied by the second number digit-wise, and the remainder of the multiplication (the first digit of number $\overline{ab} = n_1 * n_2$, where $n_1$ and $n_2$ are some digits of first and second numbers respectively) is stored so that to further be added to the second digit of the next digit-product. For each step `i` zeros are added to the end of the number (where `i` is the step-value), as far as at each step the decimal part of each product increases. The resulting number, which is being returned by the overloaded product function of GSM class, consists of the sum of all the intermediate products (roughly speaking, intermediate product is the product of one digit of the first number and the whole second number).

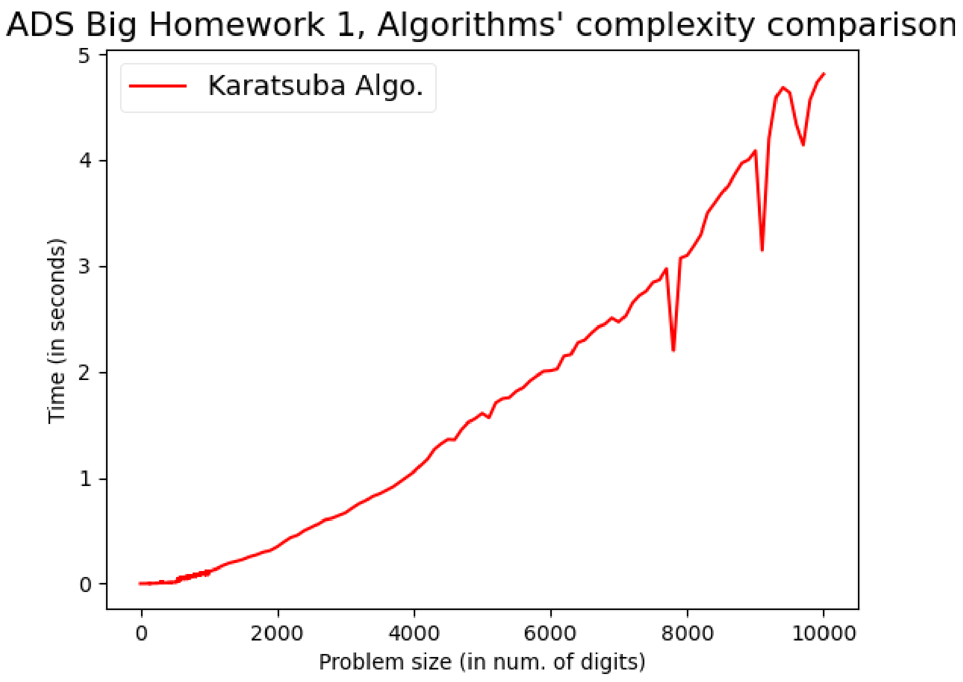
  The graph, which was received after processing the data, obtained from the `void process()` function shows us that generally the time complexity of the GSM algorithm is clearly $O(n^2)$. Thus, we have proved empirically that the time complexity of GSM is $O(n^2)$.

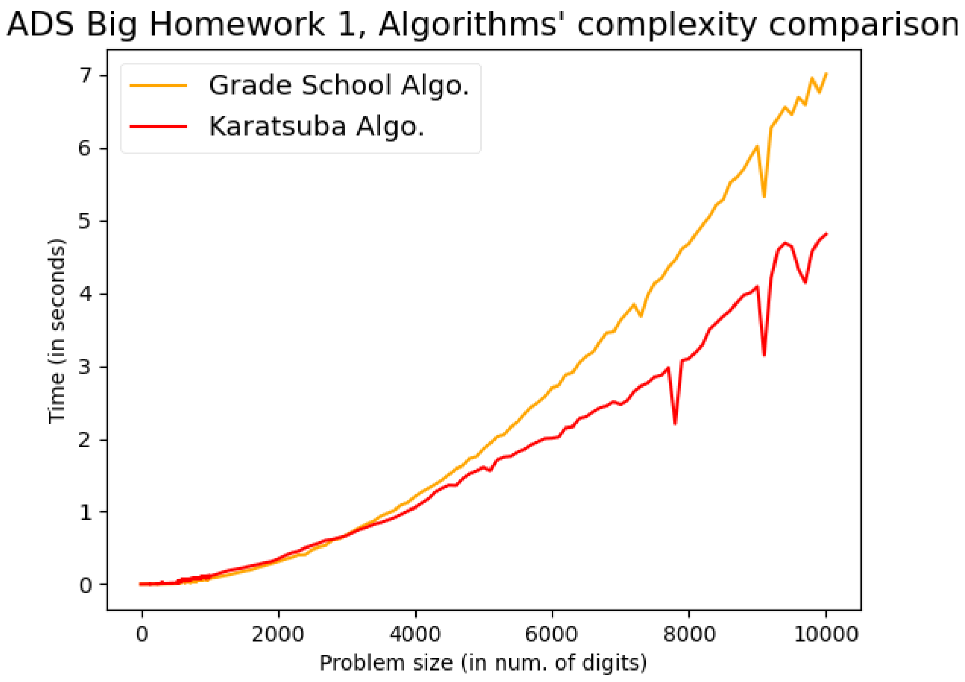


- `KaratsubaMult`

  The attempt was to implement the algorithm, suggested by A. Karatsuba to multiply long integers. The case was just to realize the recursion of multiplying the numbers repeatedly by applying Karatsuba's algorithm, and splitting the problem into three sub-problems of equal size at each recursive step, until we reach the base case. The base was chosen to be the one-digit case (and “null case”, so that is one multiplies two numbers of zero-size, the output would be zero).

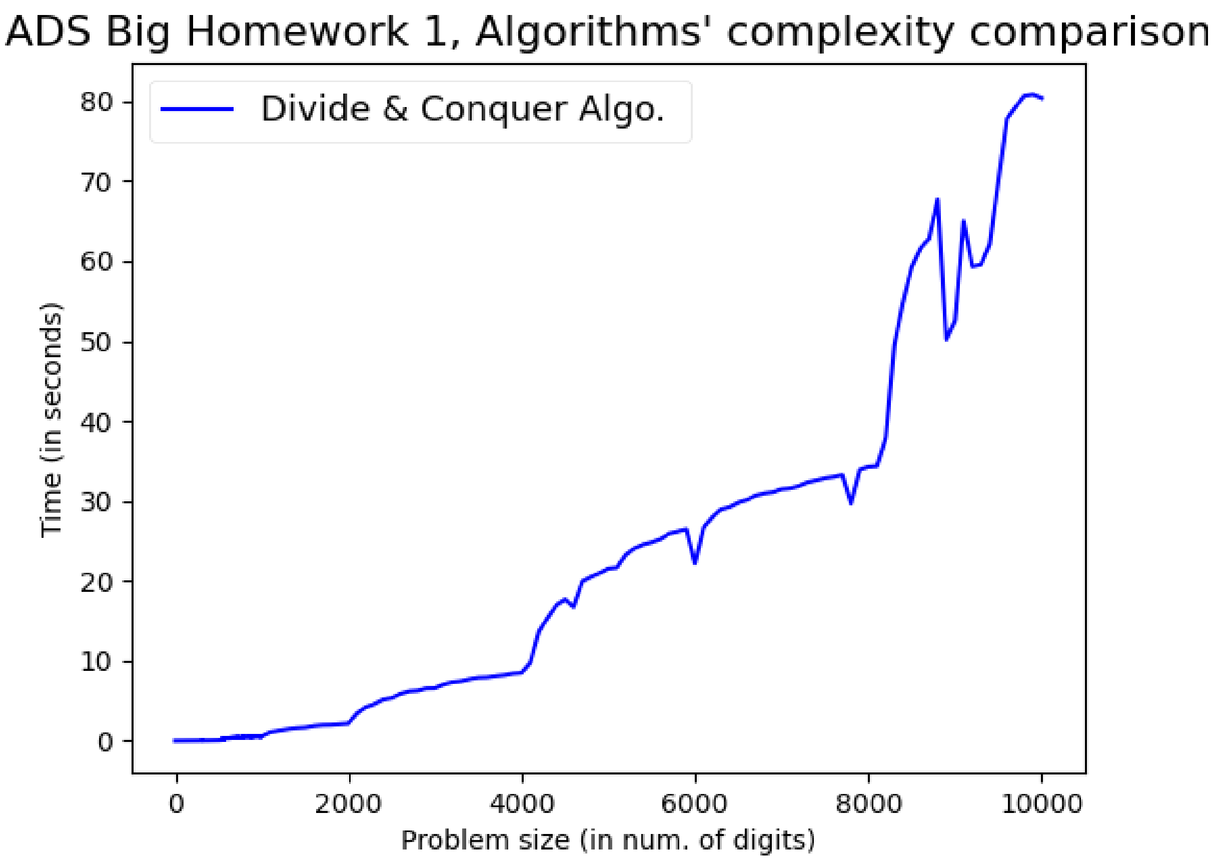
We have just directly followed the algorithm to obtain the result, and though succeeded, as far as the the graph, which was received after processing the data, obtained from the `void process()` function shows us that the time complexity of the KMA is clearly approaching $O(n^{\log_2 3})$, and clearly works much faster than the GSM algorithm. We have obtained that KMA’s growth rate is definitely much lower than the growth rate of GSM, and following the obtained results, have proved empirically that the time complexity of KMA is to be $O(n^{\log_2 3})$.





- `DNCMult`

The exact realization of Divide and Conquer algorithm did not have many differences from the implementation of the KMA, the only difference was just that the "smart trick", applied in Karatsuba algorithm had not been used there, and so the problem at each step of the recursion applied, was decomposed not to three, but to four different sub-problems of the same length, what has resulted into the time complexity of O($n^2$) with far more worse constant factor, than in GSM algorithm. The DNC algorithm appeared to be the slowest one of all three, but nevertheless, the time complexity of O($n^2$) directly follows from the results, obtained after processing the data, gotten from the `void process()` function, so that we have empirically proved that the time complexity of GSM is to be O($n^2$).



Despite the obtained results, there is a huge field for improvement of DNC algorithm, so that our current realization of DNC algorithm might be improved in the future.

- Several words about `void process()` function. This function was realized to write the data, obtained by measuring the runtime of each algorithm, when applied to multiply two arbitrary numbers of length n. The function counts the runtime of each algorithm at each step (there step means proceeding to a number of greater length, the step has increased to 100 after counting first 1000-digit product, so that not to overload the central processor of the computer) three times, and then outputs the mean-value to the .csv file, storing the data about the algorithms and their runtime in the following order:

| Problem size #digits | GSM runtime | KMA runtime | DNC runtime |
|---|---|---|---|

In order to construct the graphs, shown previously we made a short program on Python, using "matplotlib" and "pandas" libraries.

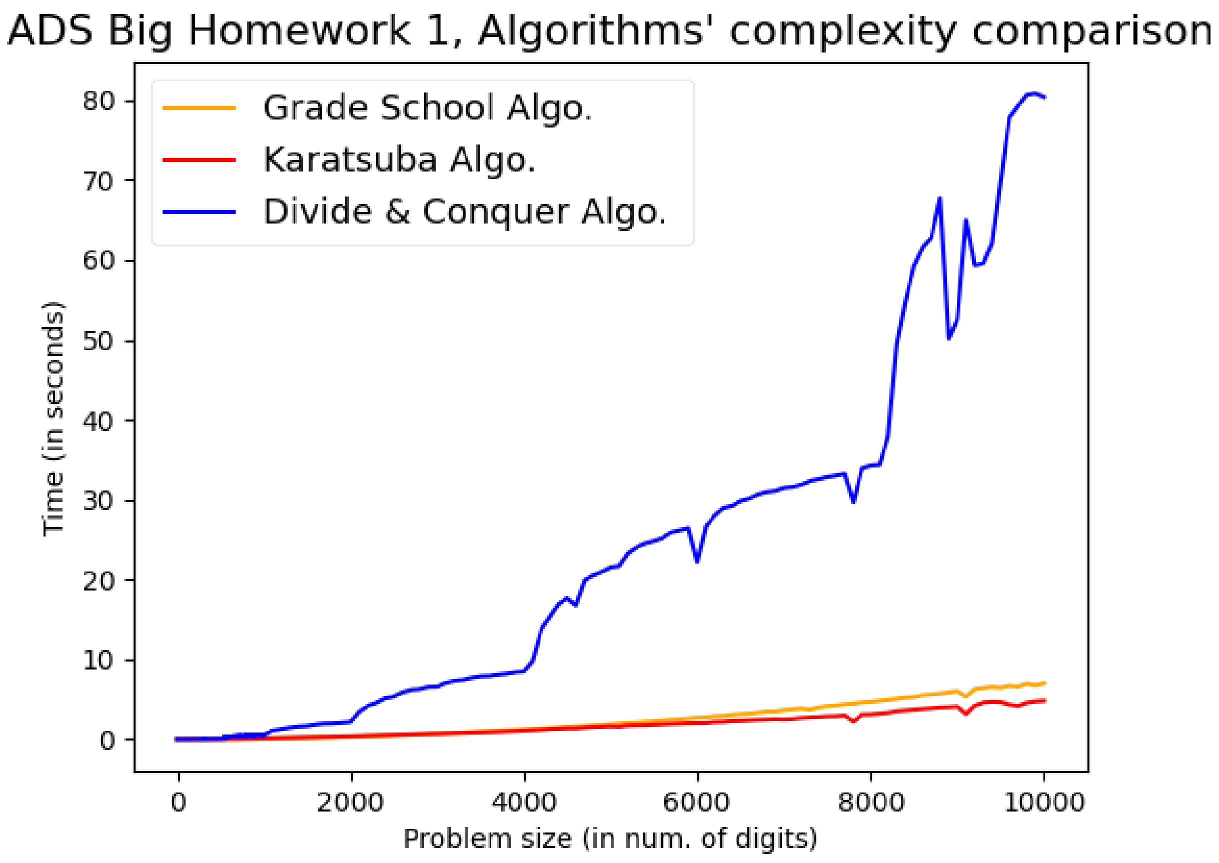
- Providing the GitHub link to the repository with the project: https://github.com/OlegMalchenko/dsba-ads2020-hw1

## Results

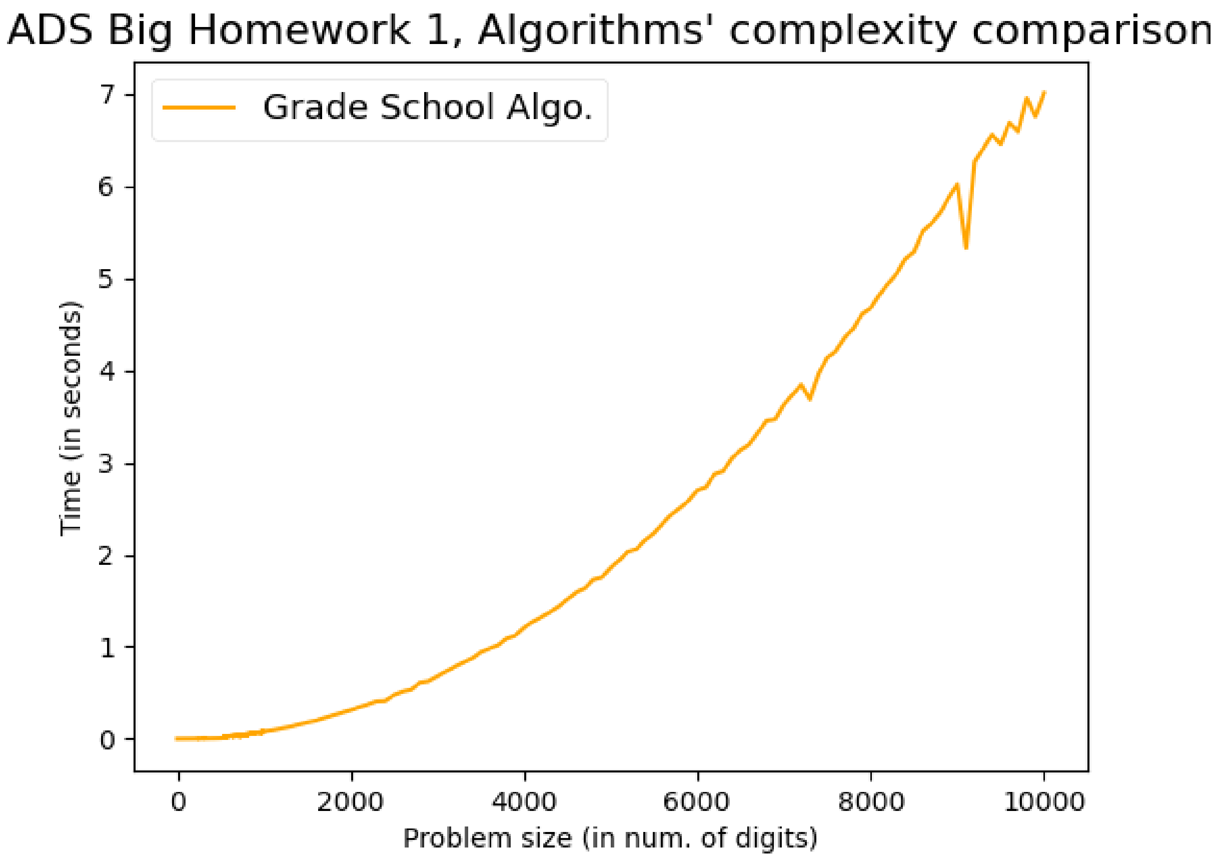
We have obtained that that the theoretical time complexity defined for each algorithm have been determined precisely, O($n^2$) for GSM and DNC algorithms, O($n^{\log_2 3}$) fir KMA. Both of the algorithms worked correctly for any problem size, and obtained the trustful results. During the local tests, no memory leakages were found and no additional computer memory was used.

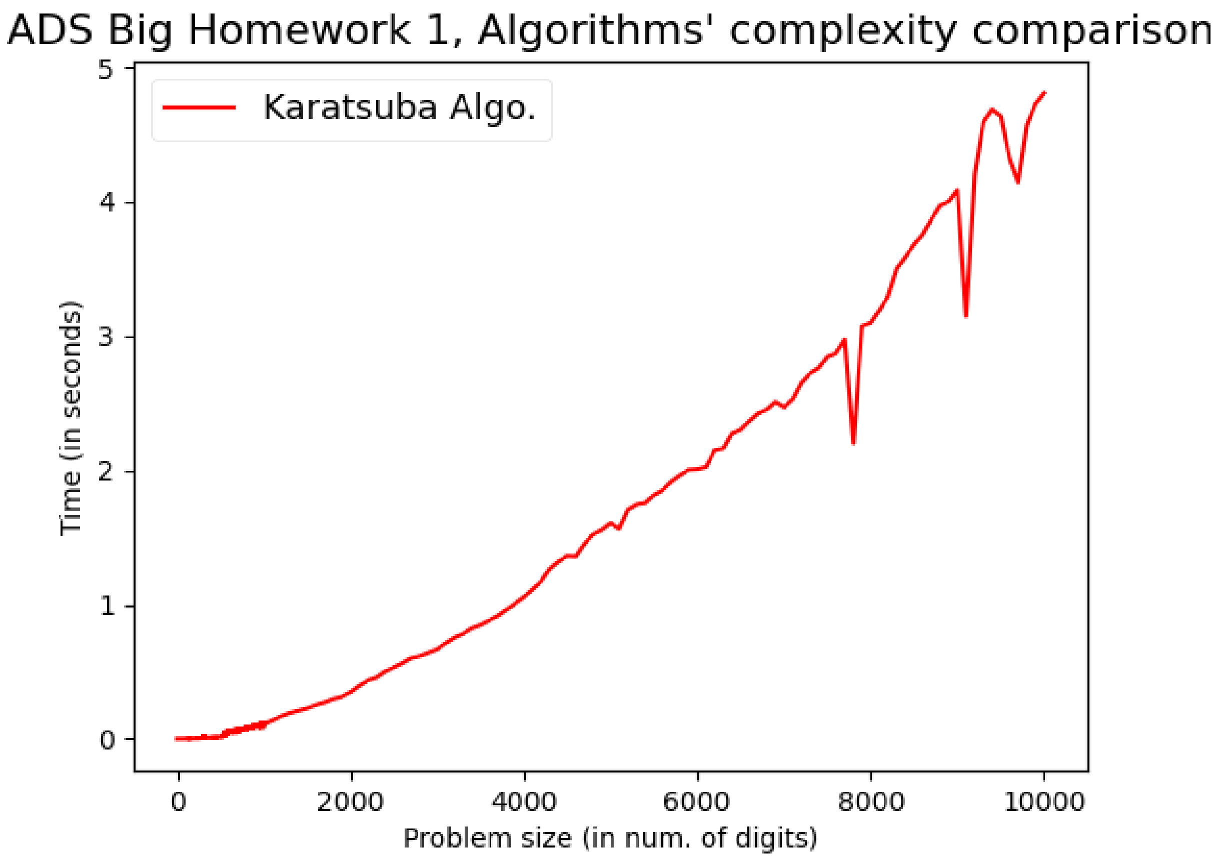
All the algorithms, which were implemented, have lived up to the expectation, with the exception of DNC algorithm, which appeared to work even slower than GSM algorithm. We see there the big field for improvement, and so that, our realization of the DNC algorithm might be improved further in the future.

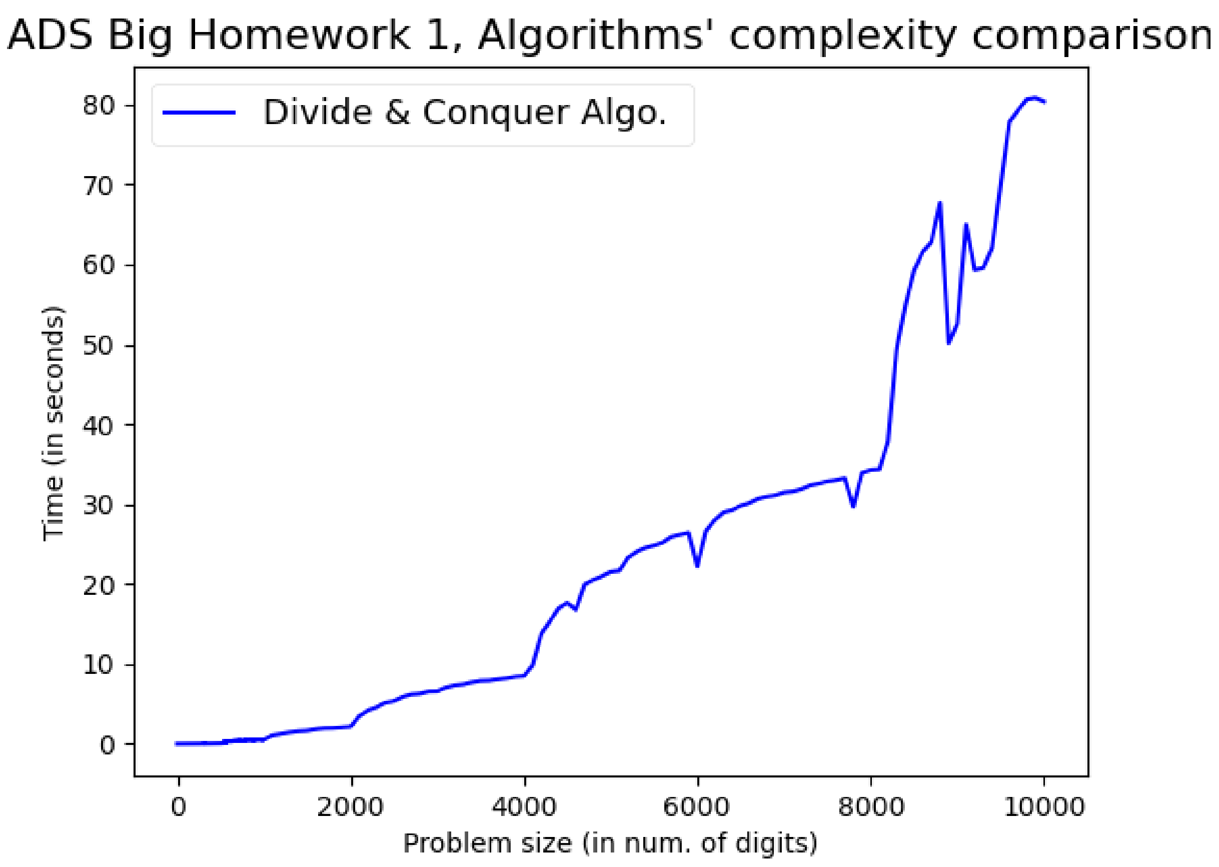
Providing the graph, comparing the time complexity of both algorithms:



Again, the graphs, describing time complexity of GSM, KMA and DNC respectively:


ADS Big Homework 1, Algorithms' complexity comparison
Grade School Algo.
Time (in seconds)
0
1
2
3
4
5
6
7
0
2000
4000
6000
8000
10000
Problem size (in num. of digits)


ADS Big Homework 1, Algorithms' complexity comparison
Karatsuba Algo.
Time (in seconds)
0
1
2
3
4
5
0
2000
4000
6000
8000
10000
Problem size (in num. of digits)



## Conclusion

Our research have definitely shown that multiplication of large integers is a very interesting and important problem in the field of the Algorithm Analysis and Computer Science, the obtained results clearly show that theoretically estimated time complexities for given algorithms are proved to be truthful, however the algorithms implemented by us are still imperfect and definitely can be improved furtheron.