Report for Big Homework № 1

Made by Mark Lukyanov (193 – 1)
Supervisor: Aleksey Varenikov
26.04.2020

Intro:

Operations on huge numbers has always been an issue, not only in arithmetic, but in the whole of mathematics. However, with the appearance of computers, it has become much more easy to implement computations on those numbers.

Nevertheless, probably the most interesting among all of the operations is multiplication. There are different approaches, how one can carry out these particular calculations. The only matter is time complexity and how a “developer” will be able to optimize his code.

Goal:

There exist different ways of multiplication, the most famous of which are “Divide and Conquer Multiplication Algorithm” or, simply, “DAC”, “Grade School Multiplication algorithm”(“GSM”) and “Karatsuba Algorithm for fast Multiplication”(“Karatsuba”). Each of the above – mentioned methods has its own running time measurements. DAC – O(n^2), GSM - O(n^2), Karatsuba – O(n^1.6). However, this is only in theory. And my aim is to determine, whether it is so or not.

I am going to do it by writing a C++ program, that will carry out all the different multiplications and measure their runtimes. Then I will plot the graphs in order to be able to estimate time complexities of those algorithms.

How I did it:

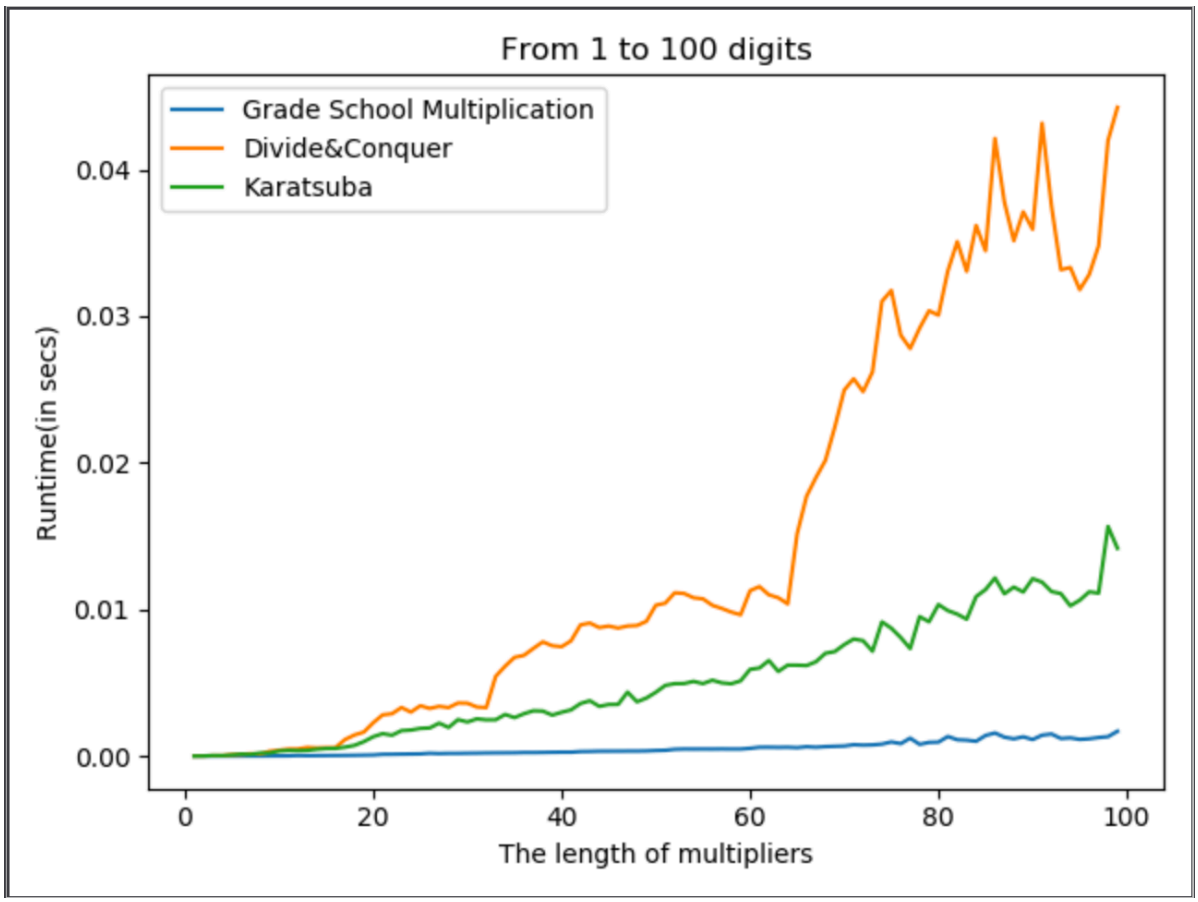
First of all, it was necessary to create my own class of numbers, which I initially presented via std::string, due to the fact that my program needs to deal with sufficiently large numbers, and the longest already implemented type (long long) is unable to deal even with 100 – digit numbers. In order to have the ability to execute all the previously mentioned algorithms, I also needed to overload operators “+” and “-“, because the original “+” does not work the way I need (“10” + “10” = “1010”, not “20”) and there is no original – for strings. Moreover, one of the major parts of DAC and Karatsuba is splitting initial numbers into pieces, that is why it was mandatory to write a function that could split the numbers. In order to be able to output my numbers and the results of operations, performed on them, to an output stream I also had to

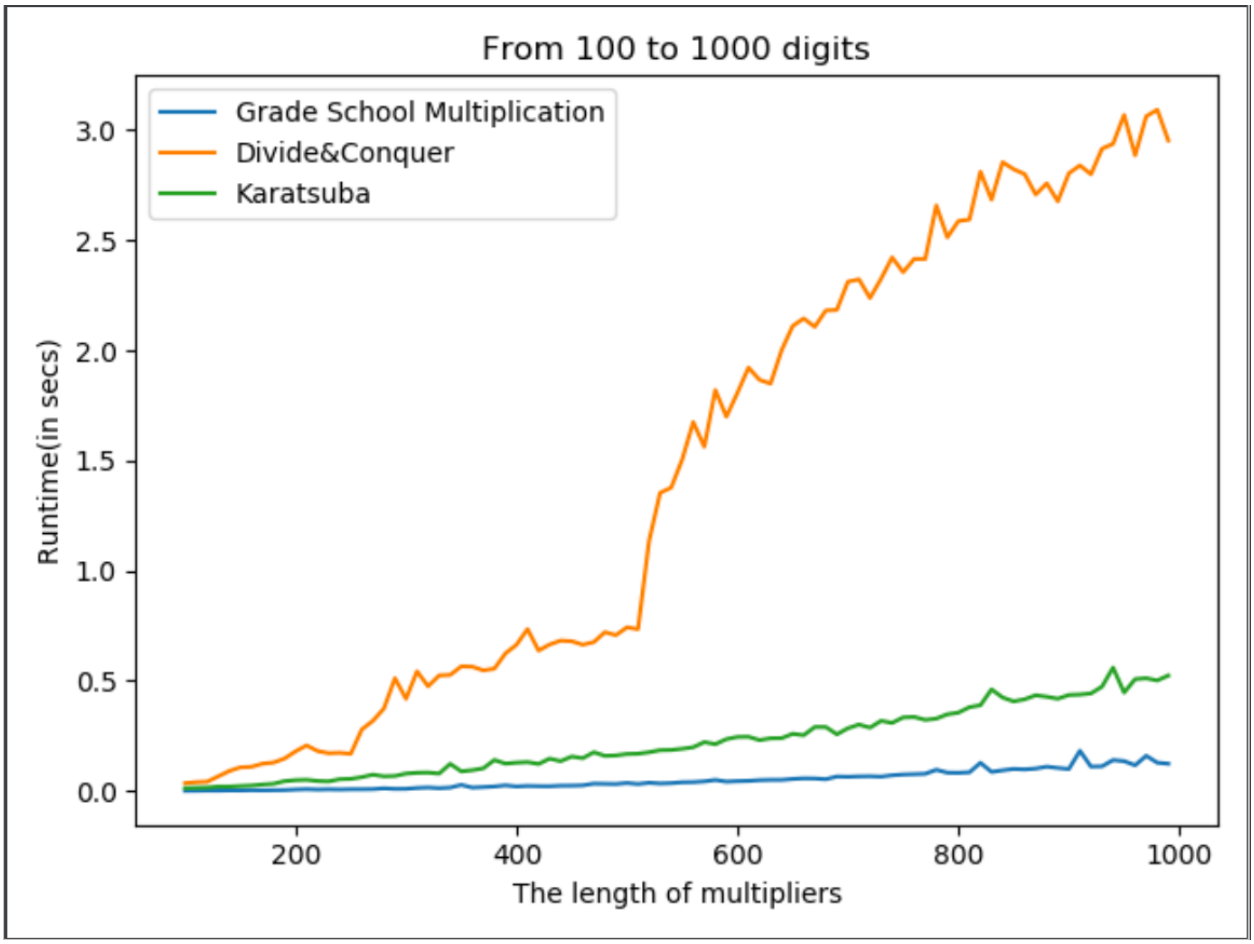
overload an output operator “<<”.One more thing is that my class needed to be equipped with all the necessary semantics that encapsulate referencing to individual components of a number, for example, a digit. And last, but not least, it was necessary to create a function that could provide me with the opportunity to get the private field of an object of the class outside this particular class.

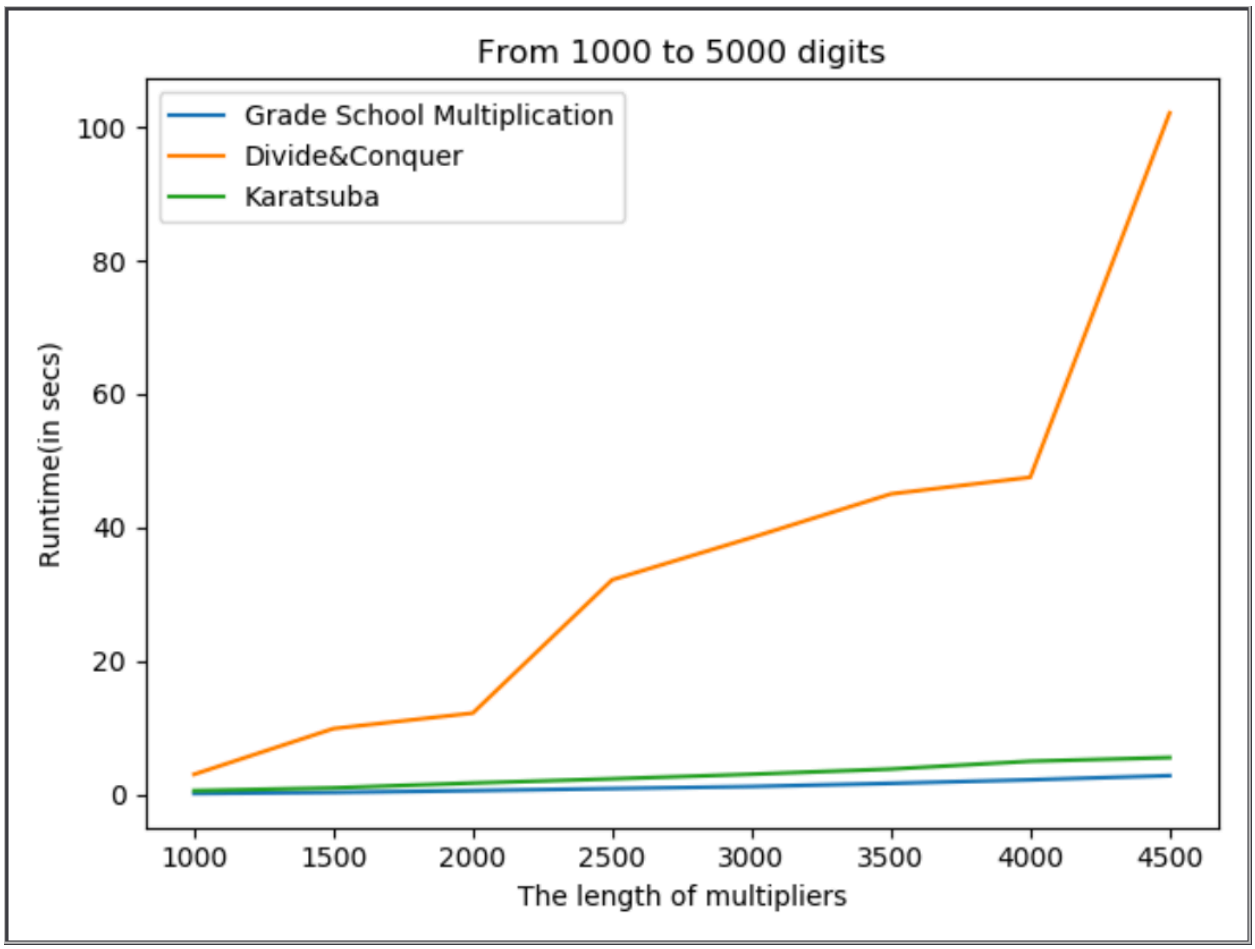
In the other class I implemented all the algorithms as its static methods. Furthermore, in order not to input objects of the class “Number” manually all the time(it would be significantly inconvenient to do so for 10000 – digit numbers), it was very helpful to create a method that would generate pseudo – random numbers each time I needed to deal with them.

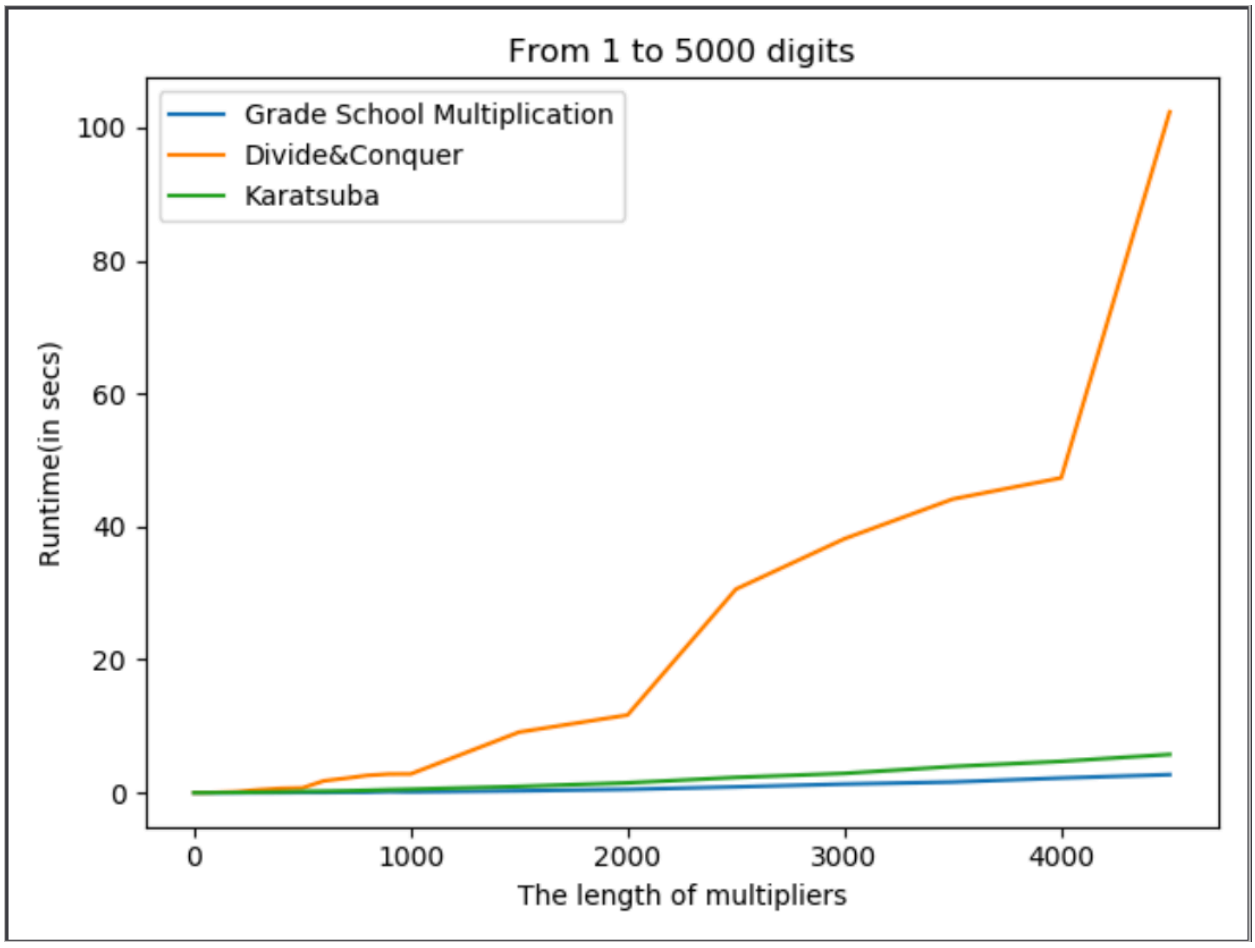
My “main” function outputs the results of applying different methods of the whole of my project to different numbers. It also outputs the runtimes of all the multiplication algorithms into a “.csv” file. All the data was visually presented by using Python’s “matplotlib”.
All the results of what I’ve done can be found here:
https://github.com/MarkLukyanov/dsba-ads2020-hw1

Results:




From 100 to 1000 digits
Grade School Multiplication
Divide&Conquer
Karatsuba
Runtime(in secs)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
200
400
600
800
1000
The length of multipliers


From 1000 to 5000 digits
Grade School Multiplication
Divide&Conquer
Karatsuba
Runtime(in secs)
0
20
40
60
80
100
1000
1500
2000
2500
3000
3500
4000
4500
The length of multipliers


From 1 to 5000 digits
Grade School Multiplication
Divide&Conquer
Karatsuba
Runtime(in secs)
0
20
40
60
80
100
0
1000
2000
3000
4000
The length of multipliers

From all these graphs it becomes obvious that the experimentally obtained results may differ from the theoretical propositions, depending on a person's implementation.

Conclusion:

Taking into consideration the fact that my results are far from the expected ones, I can state for sure that there is space for the work of upgrading the efficiency of my implementation of the initial algorithms.

Moreover, it would be better to somehow improve my code style by declaring prototypes in header files and making implementations in .cpp. Unfortunately, I completely forgot about it while writing the program.