REPORT

dsba-ads2020-hw1

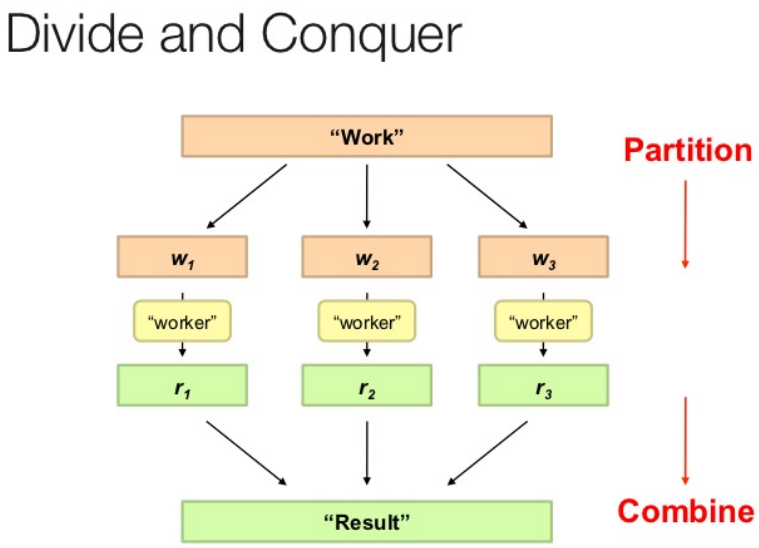
Author: Dzhkha Anika
Group DSBA193-2
Date of submission: 28.04.2020
Supervisor: Sergey Shershakov

The research problem.

The main issue I have learned is **the complexity of different integer multiplication algorithms**, such as Grade School Multiplication, Divide and Conquer Multiplication and Karatsuba Multiplication.

The Grade School Multiplication is the primitive algorithm, when me multiply digit per digit from the end of one number with shifting from right to the left, and then we add all. And it is the simplest way, but not the fastest (for computer). As we can have **n** digits in each number, we have to perform at most 2n operations to form one partial product (to get one component of adding/ per row) and we have exactly n rows, so the number of operations in the GSM grows to some constant * n^2. Therefore, the running time of this algorithm scales like $O(n^2)$. So, our goal was to find an algorithm that works faster.

We found Divide and Conquer algorithm, that works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. This approach is the basis of many algorithms, such as sorting (for example, quicksort, merge sort), multiplying large numbers etc. The principle of his work:



Divide and Conquer Multiplication example:

1234×5678

$= (12 \times 100 + 34) (56 \times 100 + 78)$

$= (12 \times 56)10000 + (34 \times 56 + 12 \times 78)100 + (34 \times 78)$

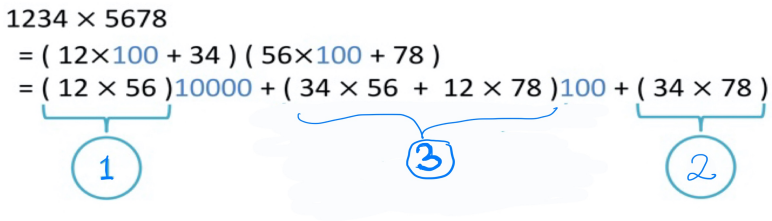
1 2 3 4

We recursively break numbers from n-digit to $\frac{n}{2}$ -digit and find 1, 2, 3 and 4 until we got 1x1 digit multiplication and then return in the formula:

$$xy = \left(a * 10^{\frac{n}{2}} + b\right)\left(c * 10^{\frac{n}{2}} + d\right) = ac * 10^n + (ad + bc) * 10^{\frac{n}{2}} + bd$$

Where x and y are two n-digits numbers, a – first half of number x, b – second, c – first half of number y, d – second. But is it works faster than GSM? The answer is not always, it can work in the same time, but not faster. This is because firstly, we separate numbers, then multiply every pair, and after that we combine results, so the running time is at least n^2.

Finally, we come to the Karatsuba algorithm, that was discovered by Anatoly Karatsuba. Initially, the algorithm based on Divide and Conquer approach, but it works faster than GSM and D&C. If we back to the example in D&C, in the Karatsuba algorithm we need to compute three multiplication:



The 3rd thing here we can get from $(a + b)(c + d) - 1 - 2$. So, the new formula looks like :

$$xy = \left(a * 10^{\frac{n}{2}} + b\right)\left(c * 10^{\frac{n}{2}} + d\right) = ac * 10^n + ((a + b)(c + d) - ad - bc) * 10^{\frac{n}{2}} + bd$$

Returning to runtime of algorithm, we may see, that we break numbers by half $\log_2(n)$ times down to problem of size 1. So, the algorithm multiplies two n-digit numbers in $O(n^{\log_2(3)}) \approx n^{1,6}$.

The plan of my research:

1) to implement a class Number, that can compute integers with a large count of digits with addition, subtraction, splitting by half, getting individual digit;

2) to learn how algorithms work and implement all three (GSM, D&C, Karatsuba) to check and estimate the time complexity experimentally;

3) to get results, and then to represent them as line charts.

Useful sources:

https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

https://en.wikipedia.org/wiki/Karatsuba_algorithm

https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm

http://www.cburch.com/csbsju/cs/160/notes/31/1.html

Implementation details.

The way to implement the Number class I chose is represent a number like a string. This approach seemed the most appropriate tool for me to work with. I put declarations and definitions in the separate files (“number.h” and “number.cpp”, “multiplicator.h” and “multiplicator.cpp”), also I have “main.cpp”, where implemented an output to CSV-file method and the call of this method.

Number class has constructors, inline methods of getting length and overloaded operator[] for getting an individual digit from number; overloaded operators+:

for adding string to Number (like “str1” + “str2” = “str1str2”), Number to string (the similar), and Number1 + Number2 = Number3 (usual addition “in a column”).

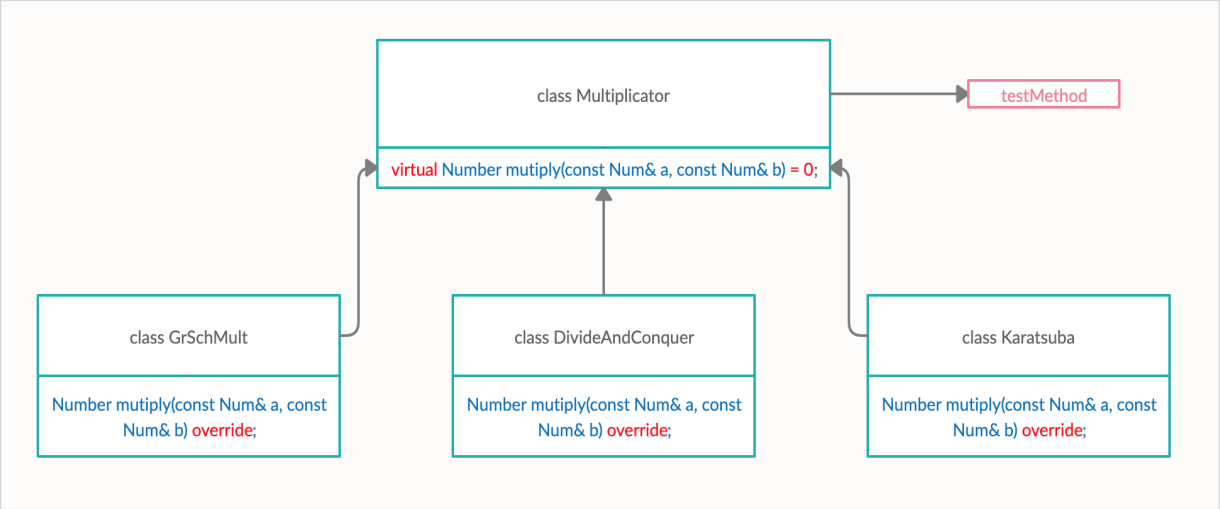
There is also an overloaded subtraction for Karatsuba algorithm and split method for D&C and Karatsuba.

Split method takes as a parameter the maximum length of two multiplied numbers (they should have equal lengths, because D&C and Karatsuba compute problems of equal size), separate a number to two halves (if number has odd number of digits, then the second part is bigger) and returns a pair of two Numbers. I haven’t found it necessary to overload an output operator or some other functions.

To the class Multiplicator I applied class inheritance. Class Multiplicator is the Base class and Derived classes are GrSchMult, DivideAndConquer and Karatsuba. This approach allowed me to redefine virtual method “multiply”.

In the class Multiplicator there are also static method that create a random Number (randomValue), static method testMethod, which performs calculation of a series of numbers by applying every algorithm 3 times and store the size of problem and average time of each algorithm straightforward in a file.

And the main part here is a pure virtual function “multiply” that was overridden in the Derived classes I mentioned earlier. The structure looks like that:



TestMethod takes as parameters the number of digits (the size of problem) and output stream, where are the size and average results of running time of each algorithm stored. Here it is created two random numbers of a given size and counted the running time of algorithm that called 3 times.

The realization of multiplication algorithms is better to see in my code, but in short:

- Grade School Multiplication – goes from end to start of one number and multiply it on each digit of second number and adds all, with adding zeroes as shifting method;
- Divide and Conquer – splits and multiplies recursively until reached base case (n = 1) and uses the giving formula;
- Karatsuba– almost the same like D&C, but uses subtraction.

Then, we can go to the main.cpp file. There is a method “out” that inputs the length of digits from we start, end; a step, that we would like to perform each time; and a file, where all the data stores. This method calls testMethod inside him. Outputting directly in CSV seemed better than storing time results firstly in vector, in order not to overcomplicate my code. The last thing here I can add, that file is opened before calling an output function and closed after.
In my code I also have some comments that explained some things and actions.

I used PyCharm for creating diagrams that takes data from “outputFile”. The file is importing pandas and mathplotlib.

URL: https://github.com/AnikaJha/dsba-ads2020-hw1

Results and discussion.

As we know for theory, the complexity of GSM is about $O(n^2)$, but thinking about different operations like shifting components of adding or some other it may take constant* n^2. According to my data, algorithm works like $(0.5 * n^2)$ and I think it is a good result, it is close to the theory. It is a middle result between three algorithms.

```
101,0.00485233,
201,0.020671,
301,0.0495557,
401,0.0878337,
501,0.142198,
```

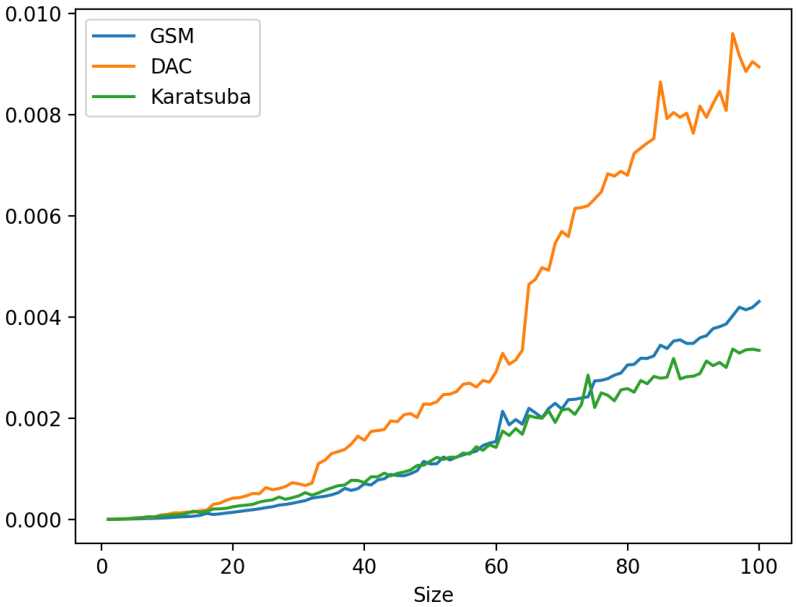
D&C algorithm in multiplication turned out like the slowest among others, and comparing to the GSM it takes almost twice more time. So, it proved that this algorithm about n^2. Only in a few moments the runtime is close to school multiplication time. But it is also may be greater than n^2, e. g. when n greater than 5000:

```
Size,GSM,DAC,
5001,15.4724,26.9006,
```

Realization of Karatsuba algorithm proved that it is really fast and smart algorithm for calculating long numbers. The most considerable results we can see, when we perform operations in huge numbers. As we can see, my algorithm in big numbers not more than $n^{1,6}$.

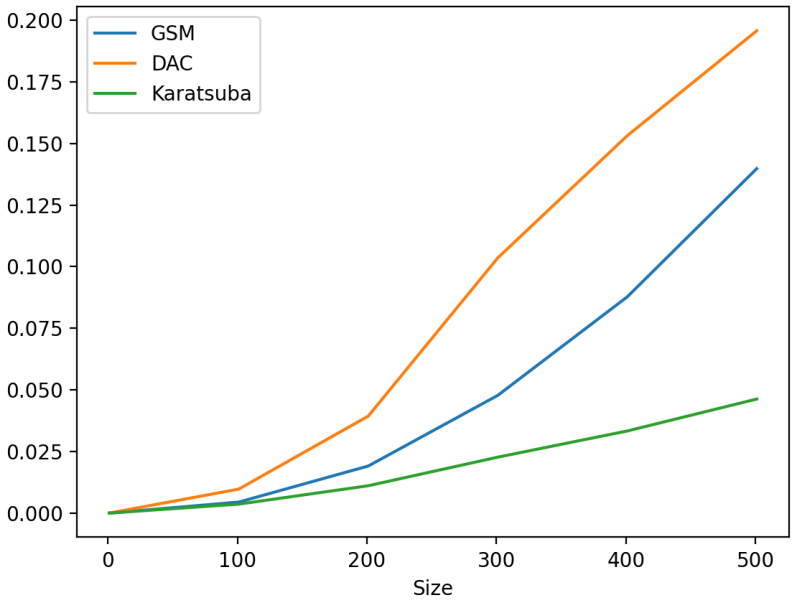
```
Size,GSM,DAC,Karatsuba
1000,0.549608,0.749231,0.130647
2000,2.17905,3.12387,0.416541
3000,5.16123,8.91813,0.812478
4000,9.20859,12.4727,1.20898
5000,14.6125,25.6391,1.77884
```

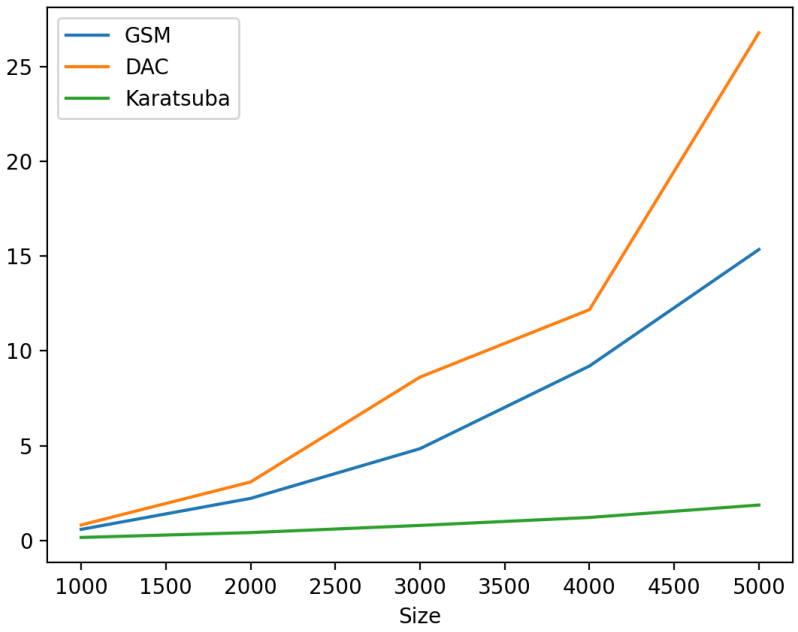
Let’s observe diagrams:



The y-axe is performed runtime of algorithms. First diagram is the result of numbers with size from 1 to 100 with step 1. We can see that GSM works faster when size of numbers is small. Starting from the size 60-70 Karatsuba algorithm overruns GSM. Interestingly that before the size 16 all algorithms work quite similar.

Then, we can consider sizes from 1 to 500 with step 100. It shows us an overall picture to understand differences between runtimes of algorithms. It is clear to see that Karatsuba works faster than others.





The last diagram I wanted to show is size 1000-5000 with step 1000. We can see here that in some points (2000 and 4000) D&C works faster than it can and it is close to the GSM. But it is still the slowest algorithm. Karatsuba performs great results and it seems like a best approach for calculating a multiplication of long numbers.

Conclusion.

To sum up, I have done a researching work, learnt some new features, for ex. how to count the running time and how to plot graphs. I would improve my algorithms in order to check it on different exceptions or something like that and try to create better realization of GSM and D&C. The result of my research is that the complexity of Karatsuba is better, than GSM in big-sized numbers, especially, and D&C works really slow, despite of recursion.