Comparison of the running time of three multiplication algorithms (C++)

(Grade School, Divide and Conquer and Karatsuba)

Fayzulina Arina
author name

192-2
group number

28.04.2020
submission date

Rudakov Kirill Aleksandrovich
supervisor name

Problem statement

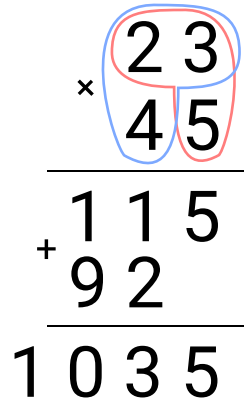
The goal of this project is to compare the operating speed of three different multiplication algorithms for two large n-digit integers. The most appropriate way of solving this task is to measure the time spent on multiplying two integers by each algorithm.

First of all, for comparing the efficiency of different algorithms, it is essential to understand their complexity, which is usually estimated by considering the memory or the time used during the implementation of the algorithm. However, regardless the way of obtaining the complexity, it depends on the size of the input data (for example, the smaller the number of elements is, the faster they will be processed).

For describing the complexity, the capital letter O is used. O(f(n)) means that the spent time or the used memory is increasing not faster than a constant multiplied by f(n).

Firstly, let's consider the most popular way of multiplication - column method (Grade School Algorithm). Assume we are multiplying two n-digit integers: X and Y. So, we are choosing one digit (element of an array) from Y, and multiply it by every digit from X. We repeat this algorithm for every digit from Y (the variable i changes from 1 to N; each time i changes, the variable j also changes from 1 to N). Thus, the number of operations depend on the size of the array as N*N = N^2.

For example:



So, the complexity of Grade School algorithm is $O(n^2)$. For a long time it was the most efficient way of multiplying two n-digit integers, until Anatoly Alexeyevich Karatsuba developed a new method with complexity $O(n^{\log_2 3})$. This method is based on the Divide and Conquer algorithm.

Assume we want multiply X and Y. The idea is to divide these integers into two parts:

$$X = ab = a*10^{(n/2)} + b$$
$$Y = cd = c*10^{(n/2)} + d$$

Notice that if we multiply the decomposed versions of X, Y:

$$X*Y = (a*10^{(n/2)} + b)*(c*10^{(n/2)} + d) =$$
$$= a*c*10^n + (a*d + b*c)*10^{(n/2)} + b*d$$

... we need to make 4 multiplications, each of them is obtained with complexity N^2 and the total complexity is 4N^2, so it doesn't simplify the problem (if we stop here, the algorithm described above is Divide and Conquer algorithm). However, Karatsuba noticed that this algorithm can be transformed so that we multiply only 3 integers:

$$X*Y = (a*10^{(n/2)} + b)*(c*10^{(n/2)} + d) =$$
$$= a*c*10^n + ((a+b)*(c+d) - ac - bd)*10^{(n/2)} + b*d$$

So, instead of computing a*c, a*d, b*c and b*d, we can compute a*c, b*d and (a+b)*(c+d).

We divide each integer into two parts, repeat the same for each half, and so on. Thus, we can apply the method recursively and reach 1-digit integers. This method reduces the complexity from N^2 to ~N^1,585, i.e. to $n^{\log_2 3}$.

Thus, theoretically, Karatsuba algorithm is faster for multiplying large integers than the Grade School algorithm. In order check this statement practically, the program was written using C++, that computes n-digit integers (where n is from 1 to k) using Grade School, Divide and Conquer and Karatsuba algorithms. In the end, time spent by three of them was compared and displayed on the graph.

Implementation details

The code contains the class “Multiplicator” that allows to multiply two n-digit integers (class instances) using methods that implement three different algorithms (that were discussed before). Vector was chosen as an instance field, so each integer is represented as a vector, and its elements are 1-digit integers.

In the further explanations “n” is a number of digits in an integer. Build-in multiplication (*) is used only when n = 1. The program randomly generates two integers of the same length.

Structure of the class

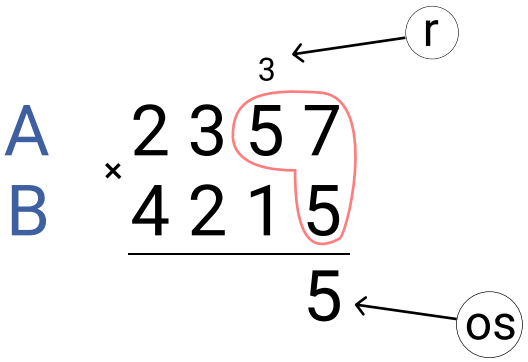
Field	**data** *// vector with n elements*
Constructor	**by default**
Methods	**RunExperiment()** **ComputeTime(n)** time4n(int) randomizer(int) GradeSchool(vect, vect) DaC(vect, vect) Karatsuba(vect, vect) digits2int(vector) int2vec(int) addzeros(vect, int) mult10n(int, int) changesize(vect, vect) divide2(vect, vect, vect) summ(vect, vect) **outputfile(vector)**

(+Python Matplotlib)

To understand the whole algorithm, let's describe the code for n=4 (as an example). For the simplicity, we will start "from the inside", applying the simplest methods first.

Step 1. Using **randomizer(4)**, generate two 4-digit integers A and B.

Step 2. Multiply A and B using **Grade School** algorithm and compute the spent time in milliseconds.

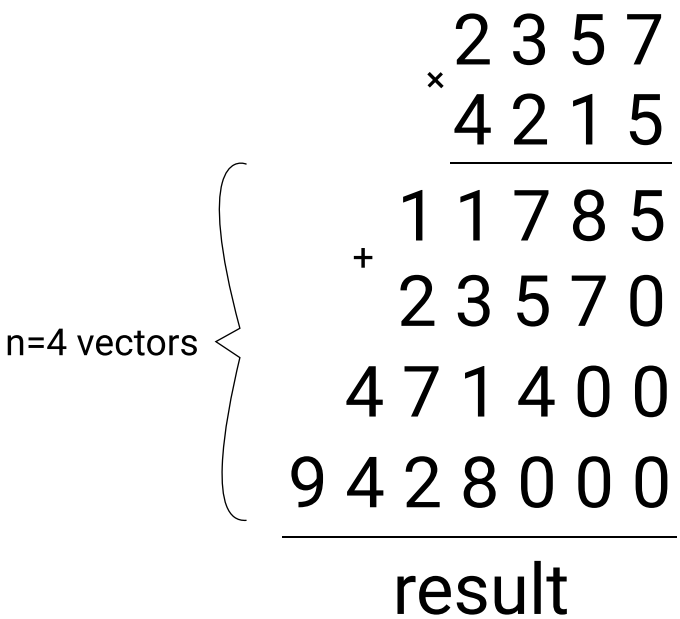


Each result of multiplying one digit from B by A is stored in a vector.

temp1 = { r , os1 , os2 , os 3 ...}

temp1 is added to the two-dimensional vector.

Each vector temp1, temp2... is transformed to int by the method **digits2int(temp)** and added to the result.



Step 3. Multiply A and B using **Divide and Conquer** algorithm and compute the spent time in milliseconds.

Case 1: n = 1

Multiply two integers

Case 2: n = 2

Result = a*c*10^2 + (a*d+b*c)*10 + b*d

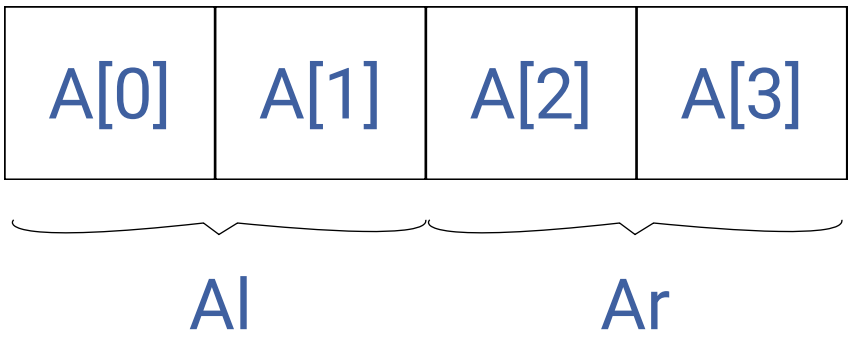
*// multiplication by 10^n is replaced by the function **mult10n(int, n)** to avoid using build-in multiplication for long integers*

Otherwise:

Make A and B have the same length that is a multiple of 2 using **changesize(A, B).**

// for the experiment we consider two integers of the same length, but the program is written considering all the cases for its universality

Divide A and B into two parts - vectors of the same length n/2 using **divide2(A, Al, Ar).**



Compute a*c, a*d, b*c and b*d using recursion.

Compute result = a*c*10^n + (a*d+b*c)*10^(n/2) + b*d.

*// again, multiply by 10^n using **mult10n(int, n)***

Step 3. Multiply A and B using **Karatsuba** algorithm and compute the spent time in milliseconds.

Case 1: n = 1

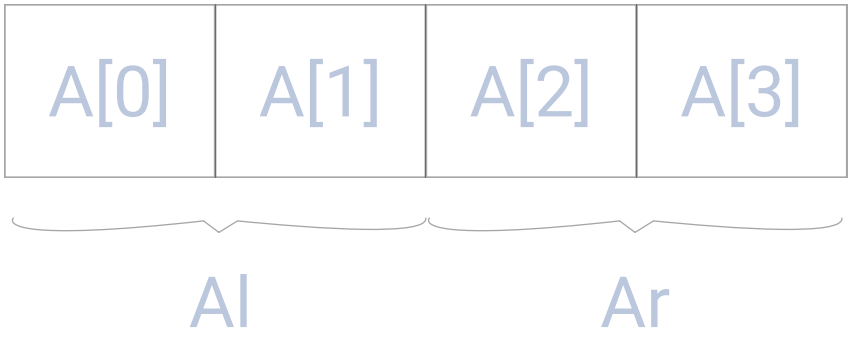
Multiply two integers

Otherwise:

Make A and B have the same length that is a multiple of 2 using **changesize(A, B).**

// for the experiment we consider two integers of the same length, but the program is written considering all the cases for its universality

Divide A and B into two parts - vectors of the same length n/2 using **divide2(A, Al, Ar).**



Compute a+b, c+d using **summ(Al, Ar).**

Compute a*c, (a+b)*(c+d) and b*d using recursion.

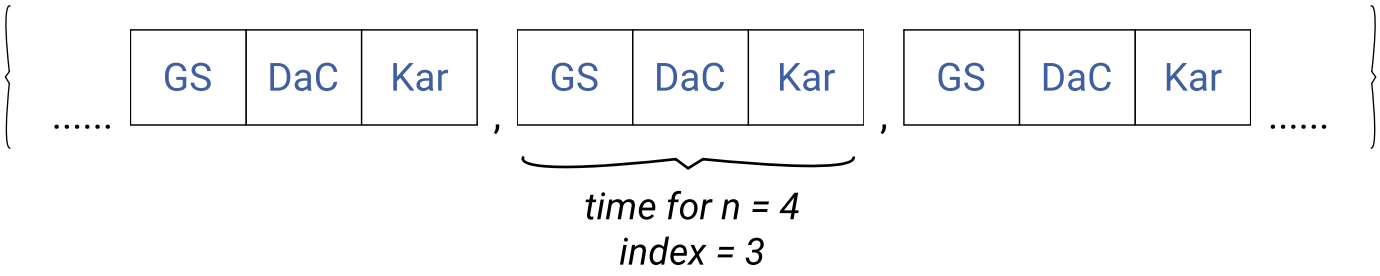
Result = a*c*10^n + ((a+b)*(c+d) - ac - bd)*10^(n/2) + b*d.

*// again, multiply by 10^n using **mult10n(int, n)***

The same as in Divide and Conquer

Step 4. Put time in milliseconds spent by these three algorithms in a vector using **time4n(4).**

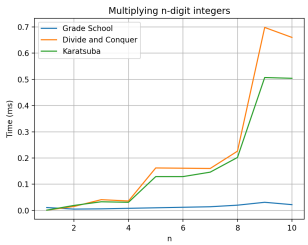
Step 5. Put the created vector in a two-dimensional vector using **ComputeTime(1000).**



Step 6. Save data in TimeTest.csv using **outputfile(vect).**

n	Grade School	Divide and Conquer	Karatsuba
1	0.011	0.002	0.002
2	0.005	0.018	0.022
3	0.009	0.05	0.041
4	0.011	0.047	0.042
5	0.014	0.209	0.187
6	0.015	0.202	0.171

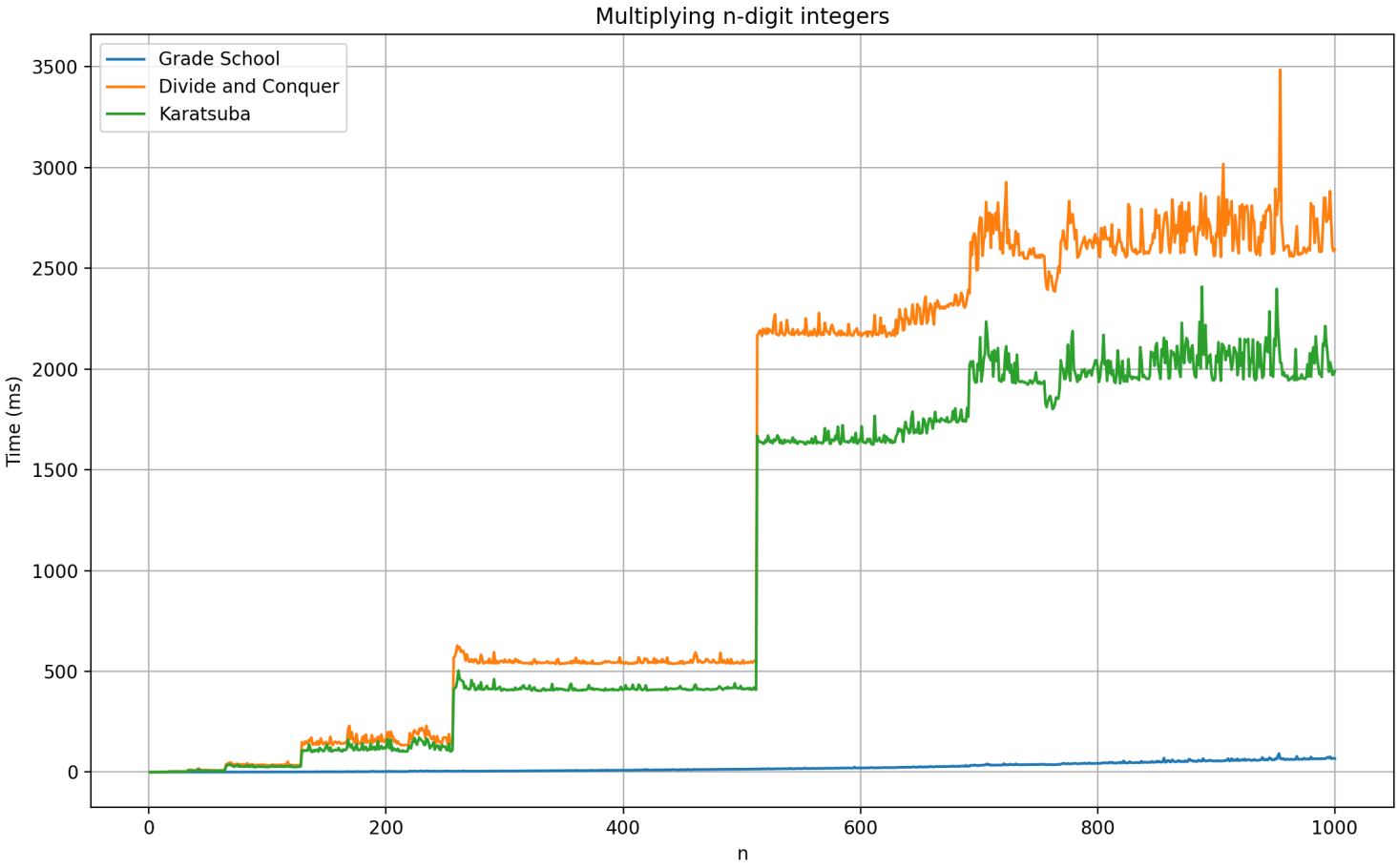
Step 7. Plot a graph of the results with **Python Matplotlib.**



Link to the repository

Results

The program was run for multiplying n-digit integers when n is from 1 to 1000. The results are shown on the graph were drawn with Python Matplotlib.



From the graph we can see that the developed code for n=1000 showed that the Grade School algorithm, contrary to theoretical calculations, is faster for mutiplying large integers, when other two algorithms (Divide and Conquer and Karatsuba) show a similar pattern. The most time consuming algorithm is Divide and Conquer.

To conclude, the results of the experiment using the written code didn’t match the theoretical position. It can be assumed that one of the key factors that affects the results is using built-in multiplication only for performing one-digit multiplications. So, modifying the code, other results can be obtained.