National Research University Higher School of Economics Double-Degree Bachelor's Program of HSE and University of London “Data Science and Business Analytics”

“Estimating the time complexity of different multiplication algorithms”

Author: Ivanova Daria

Group: 191-2

Supervisor: Shershakov S.A.

Submission date: 28.04

Moscow, 2020

Problem statement

In this research we are consider three different multiplication algorithms: Grade School Algorithm, Divide and Conquer and Karatsuba.

Complexity of these algorithms:

Grade school: $O(n^2)$

Divide and Conquer: $O(n^2)$

Karatsuba: $O(n^{\log_2 3})$

Description of these algorithms:

Grade school:

1) Write the numbers one under the other, and the larger number (consisting of a large number of digits) should go first — this is more convenient.

2) Then we consistently multiply all the numbers of the first (upper) number by the digit of the second number and write below the line — we multiplied the first number by the units of the second number.

3) Then multiply all the digits of the first number by the second digit of the second number — we have already multiplied this by tens, so we write the result below and one digit to the left.

4) Act similarly until we multiply all the digits of the first by all the digits of the second. Then we add the received products and get the answer.

Divide and Conquer:

It is an algorithm design paradigm based on multi-branched recursion. A divide-and-conquer algorithm 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. The solutions to the sub-problems are then combined to give a solution to the original 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.

$x = a*10^{(n/2)} + b$

$y = c*10^{(n/2)} + d$

// a and b are halves of x, c and d are halves of y //

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

Karatsuba:

x = a·10^(n/2) + b

y = c·10^(n/2) + d

// a and b are halves of x, c and d are halves of y //

Compute recursively ac, (a + b) · (c + d), bd

$x \cdot y = a \cdot c10^{n} + ((a + b) \cdot (c + d) - a \cdot c - b \cdot d) \cdot 10^{n/2} + b \cdot d$

Plan of the research:

1) Implement the algorithms using classes and their methods

2) Calculate the time of each for random numbers

3) Save the result to a csv file

4) Construct graphs using Python's Matplotlib

5) Compare data

6) Make a conclusion

Literature:

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

https://spadilo.ru/umnozhenie-v-stolbik/

Implementation details

Code structure:

Class Multiplicator

Basic methods:

```
class Multiplicator {
public:
    static std::string generateRandomDigits(std::uint32_t);
    static std::string integerToString(std::uint8_t);
    static std::uint8_t integerToChar(std::uint8_t);
    static std::uint8_t charToInteger(std::uint8_t);
    static std::string addition(std::string, std::string);
    static std::string substraction(std::string, std::string);
    static std::string shiftLeft(std::string, std::size_t);
    static std::string shiftRight(std::string, std::size_t);
    static std::uint32_t nextPowerOf2(std::uint32_t);
    static std::string multiplyByOneDigit(std::string, std::uint8_t)
```

These methods will be used to implement algorithms.

Algorithms:

```
static std::string gradeSchoolMultiply(std::string, std::string);
static std::string karatsubaMultiply(std::string, std::string);
static std::string divideAndConquerMultiply(std::string, std::string);
```

1) gradeSchoolMultiply is implemented with the help of methods: multiplyByOneDigit, charToInteger, ahiftLeft and addition. Also, I used one cycle “for”

2) Divide and Conquer algorithm is implemented with the help of methods: shiftLeft, charToInteger, nextPowerOf2, addition and substraction.
   Also, it is important to mention that it is a recursive algorithm ⇒ almost everything is done recursively and using the formula, which I wrote in the first part

3) Karatsuba algorithm is implemented with the help of methods: charToInteger, addition, substraction, shiftLeft
   It is also a recursive algorithm ⇒ almost everything is done recursively and using the formula, which I wrote in the first part

Measuring time:

These are methods which help me to estimate the running time of an algorithm and save the data to a csv file.

```
static std::chrono::steady_clock::duration measureGradeSchoolMultiply(std::string,
std::string);
static std::chrono::steady_clock::duration measureKaratsubaMultiply(std::string,
std::string);
static std::chrono::steady_clock::duration
measureDivideAndConquerMultiply(std::string, std::string);

static void csvSave(
        std::vector<std::uint32_t>, std::vector<std::chrono::steady_clock::duration>,
        std::vector<std::chrono::steady_clock::duration>,
std::vector<std::chrono::steady_clock::duration>);

static void doTests(std::uint32_t);
```

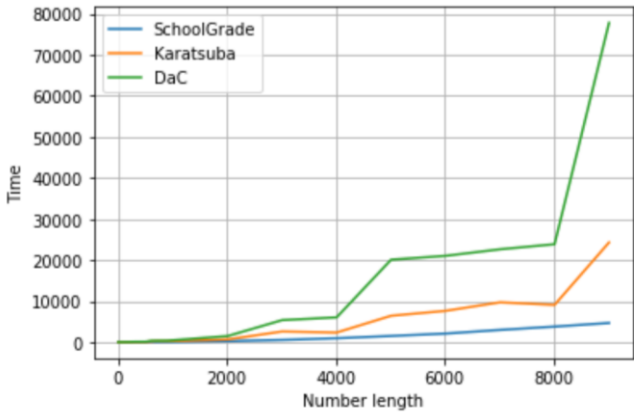
Method “doTests” uses almost all methods of the class to test the operating time of the algorithms.

Graphs:

For construction of the graph I used Python library matplotlib.pyplot and did it in Jupyter Notebook

```
import matplotlib.pyplot as plt
import csv
import numpy as np

y1 = []
y2 = []
y3 = []
x = []

with open('results.csv','r') as csvfile:
    plots = csv.reader(csvfile, delimiter=',')
    for row in plots:
        x.append(int(row[0]))
        y1.append(float(row[1]))
        y2.append(float(row[2]))
        y3.append(float(row[3]))

x = np.array(x)

plt.plot(x, y1, label='SchoolGrade')
plt.plot(x, y2, label='Karatsuba')
plt.plot(x, y3, label='DaC')
plt.grid()
plt.xlabel('Number length')
plt.ylabel('Time in mls')
plt.legend()
plt.show()
```

Results and Discussion



The Karatsuba multiplication algorithm was supposed to asymptotically faster than the quadratic "grade school" algorithm. Grade school and Divide and Conquer algorithms are supposed have approximately the same execution time. But in my implementation, it turned out that Grade school multiplication works faster for smaller numbers.

Comparing recursive algorithms, Karatsuba is faster because it performs less multiplication operation. I think, for results to be correct and Karatsuba to be the fastest we should work with extremely long numbers (over 100k digits), but unfortunately it is almost impossible to do such calculations on my laptop even with the logarithmic step.

Maybe in my program Karatsuba and DaC work slower because they are implemented recursively and if they are implemented in non-recursive structure, they will work correctly => faster than grade school algorithm.

Conclusion

It is clear that my implementation of the given algorithms is not perfect. I think that it would be better to divide my code and create more classes, for example, for every algorithm create its own class. Also, for now I cannot answer the question why my grade school algorithm works faster then other do, but I think it could be improved by reorganizing the whole code.

To conclude, I think that despite the fact that the results of my program are not accurate, all my algorithms are implemented well and work ok. It is easy to read the code and to understand methods and algorithms I implemented.