Department of Computer Engineering

BLG 336E
Analysis of Algorithms II
Project 2 Report

Spring 2019

Muhammed Raşit EROL

150150023

1 INTRODUCTION

Multiplying two polynomials efficiently is an important subject for computer science community because signal processing, cryptography and coding theory are popular areas which use polynomial multiplication. The multiplication operations are very costly in terms of time complexity. Hence, in order to reduce time complexity of multiplication of polynomials, there have been developed several algorithms using divide and conquer approach. In this project, it is required that comparing two polynomial multiplication algorithms which are Algorithm1 also called Karatsuba Algorithm and classical method which uses shift and add approach.

2 IMPLEMENTATION

In this project, it is used one class called Multiplier. With that class, two algorithms which are Karatsuba Algorithm and classical method have been implemented.

In the program workflow, it is asked bit size of the binary string at the start of the program. After that, two random binary string is created with given size and respectively, two algorithms are executed. Also, it is possible two see all bit sizes which is mentioned homework pdf with pressing "1". The created binary strings and results for both Karatsuba Algorithm and classical method in type of decimal and binary format can be seen both terminal and the output file with the execution time for both algorithms. The file name can be given in start of execution of the program.

In this project, in order to reduce time complexity, divide and conquer approach is used. When we apply the classical method for multiplication of polynomials, time complexity becomes $O(n^2)$. However, if we used use Karatsuba Algorithm, because of the divide and conquer method, time complexity has been reduced $O(n^{1.59})$.

However, in this project, this reduction cannot be seen. The reason of that is there are a lot of string operation for Karatsuba Algorithm, which increases time complexity of the Karatsuba Algorithm. It can be seen in the code, classical method works with two loops, which causes time complexity $O(n^2)$. However, in the Karatsuba Algorithm, divide and conquer algorithm is used and intermediate values P1, P2 and P3 is calculated. These causes reduction of time complexity because, it has been decreased number of calculation. Normally, it causes time complexity of $O(n^{1.59})$; however, instead of using integer values for P1, P2 and P3 values, it is used string type. The operation of these variables require equal length and all operations about like these problems prevent reduction of time complexity. Hence, classical method has been work much faster than Karatsuba Algorithm.

Shorty, in the classical method, first string has been added the result if the item of each iteration on string 2 is one. That is simplified also as shift and add approach. The Karatsuba Algorithm is given homework pdf. In this homework, it cannot be seen expected result for both algorithms in terms of execution times. However, the theoretical explanation of these algorithms can be seen in the following parts.

The code compiled at ITU SSH server successfully like that:

```
g++ *.cpp –o algo -std=c++11
```

```
./algo output.txt
```

3 ANALYSIS

The problem of the polynomial multiplication is large number of addition and shifting operations. That causes $O(n^2)$ time complexity and it can be reduced by using divide and conquer method. The Karatsuba Algorithm is one of the algorithm which uses divide and conquer approach and it reduces time complexity $O(n^{1.59})$. The Karatsuba Algorithm is explained in detail in the following part.

Using divide and conquer approach we can divide the binary string in two parts and we can multiply these parts (see Figure 1).

$$\begin{aligned} x &= 2^{n/2} \times x_1 + x_0 \\ y &= 2^{n/2} \times y_1 + y_0 \\ xy &= \left(2^{n/2} \times x_1 + x_0\right)\left(2^{n/2} \times y_1 + y_0\right) = 2^{n} \times x_1 y_1 + 2^{n/2} \times \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 \end{aligned}$$

Figure 1: Divide and conquer approach for binary strings

The recurrence equation become like that (see Figure 2):

$$\mathrm{T}(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{cn}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) = \Theta(n^2)$$

Figure 2: The recurrence equation of Figure 1

From that, it can be seen that time complexity remains same. However, Karatsuba Algorithms prevents unnecessary computation (see Figure 3).

$$\begin{aligned} x &= 2^{n/2} \times x_1 + x_0 \\ y &= 2^{n/2} \times y_1 + y_0 \\ xy &= 2^{n} \times x_1 y_1 + 2^{n/2} \times \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 \\ &= 2^{n} \times x_1 y_1 + 2^{n/2} \times \left((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0\right) + x_0 y_0 \end{aligned}$$

Figure 3: Karatsuba Algorithm divide and conquer approach

From that, the recurrence equation become like that (see Figure 4):

$$T(n) = \underbrace{3T(n/2)}_{\text{recursive calls}} + \underbrace{cn}_{\text{add, subtract,shift}} \Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

Figure 4: The recurrence equation for Figure 3

As a result, Karatsuba Algorithm reduces time complexity of polynomial multiplication (see Figure 4). The pseudo code of Karatsuba Algorithm can be seen below.

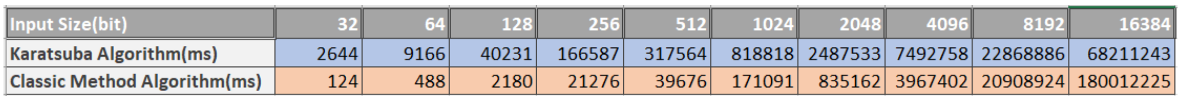
```
string karatsuba_algorithm(string str1, string str2):

       Input: Positive integers x and y, in binary
       Output: Their product

       n = max(size of x, size of y)
       if n = 1: return str1*str2

       string xL, xR = leftmost [n/2], rightmost [n/2] bits of x
       string yL, yR = leftmost [n/2], rightmost [n/2] bits of y

       string P1 = karatsuba_algorithm (xL, yL)
       string P2 = karatsuba_algorithm (xR, yR)
       string P3 = karatsuba_algorithm (xL + xR, yL + yR)

       return P1 × $2^n$  + (P3 – P1 – P2) × $2^{n/2}$+ P2
```

The line by line time complexity is not given in the pseudo code because it is already given in the figure 4. However, line by line time complexity of classical method is given below.

```
classical_method ():

O(1)   int str1_size = size(str1)
O(1)   int str2_size = size(str2)

O(1)   string result
O(1)   string temp_str = str1

O(n)   For each element i on range of str1_size
              result += “0”
       Endfor


O(n)   If the str2[str2_size - 1] is ‘1’
              result = add_strings(result, str1)
       Endif

$O(n^2)$ For each element i on the range of str2(back iteration)
              temp_str += “0”
              If the str2[i] is ‘1’
O(n)                 result = add_strings(temp_str1, result);
              Endif
       Endfor

       return result
```

The time complexity of *add_strings* function is $O(n)$. Hence, due to the loop in the classical method total time complexity is $O(n^2)$.

The execution time of both algorithms can be seen in the following part.



Input Size(bit)	32	64	128	256	512	1024	2048	4096	8192	16384
Karatsuba Algorithm(ms)	2644	9166	40231	166587	317564	818818	2487533	7492758	22868886	68211243
Classic Method Algorithm(ms)	124	488	2180	21276	39676	171091	835162	3967402	20908924	180012225

Figure 5: The execution time of algorithms

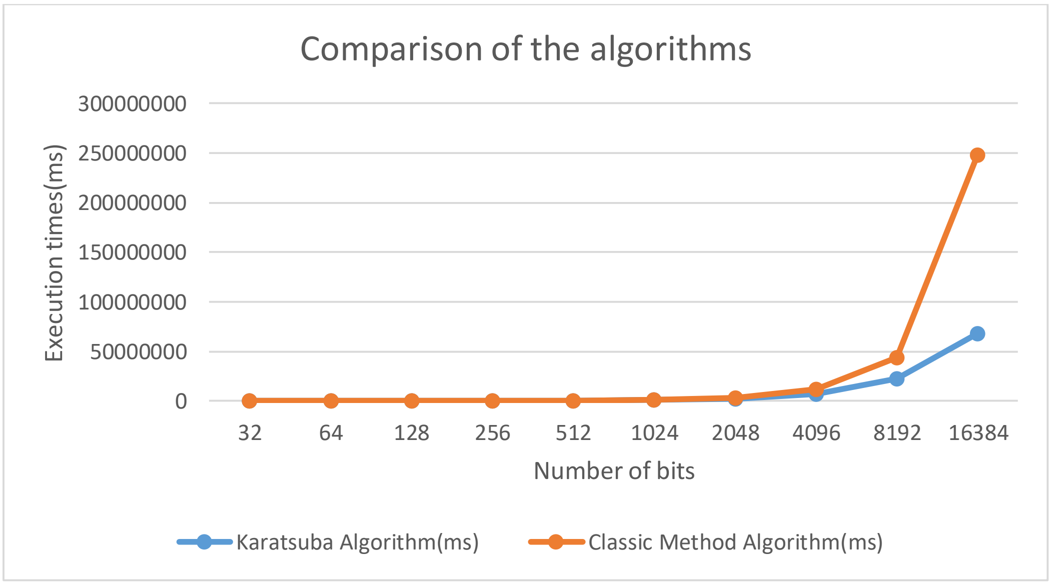


Figure 6: Comparison of the algorithms

As a result of this project, divide and conquer method reduces time complexity of problems. It can be seen for classical method and Karatsuba Algorithm in order to multiply two binary strings. In the Figure 6, Karatsuba Algorithm work more efficiently than classical method because time complexity for Karatsuba Algorithm is $O(n^{1.59})$ but for classical method it is $O(n^2)$.