Assignment 2WF90, Integer Arithmetic

Julian Dziegielewski 1300814, Oana Radu 1325973,
Ramon Iancu 1299794, Lyuben Petrov 1306081

September 21, 2019

Contents

1 User Guide

For the aims of this software assignment we decided to use C++ as a programming language. The code can be opened and run with any C++ - friendly environment such as CodeBlocks or SublimeText. To work with our program one should use console application for C++. Manual for console application: 1. The program starts with choosing which arithmetic user wants to use in a current calculation, typing "integer" or "modular"
2. User should choose the radix of the numbers
3. User can choose the operation: addition, subtraction or multiplication by naive method typing "add", "subtract" or "multiply" for integer arithmetic and "add" and "subtract" for modular arithmetic.
4. User should input numbers x and y in a correct radix (that he/she has chosen before)
If user chose modular arithmetic he will need to give modulo as well
5. Answer outputs on the screen
6. User can repeat the calculation

2 General Information

In all of our algorithms we take the input numbers as strings and then translate them into arrays, such that every character in the string is a separate element of the array. For bases from 11 to 16 the cycle replaces the letter with its respective value in decimal and puts that value in the array (for instance instead of $A_{[16]}$ in that position of the array we put $10_{[10]}$). Also, it translates the Ascii characters such as '9', for example, from strings into integers by subtracting 48.

3 Integer Arithmetic

3.1 Addition

To implement this simple addition algorithm we took inspiration from Algorithm 1.1 from the lecture notes[1]. The algorithm adds two b - base numbers of length n and m, respectively. The number of atomic operations (addition, subtraction) is at most 2*min(m,n) - 1 which means the algorithm runs in O(n) time for n, m $\in \mathbb{Z}$.

3.2 Subtraction

The subtraction algorithm is also inspired from the lecture notes[1] (Algorithm 1.2). In a fashion identical to the addition algorithm from the previous section, this procedure subtracts 2 base - b numbers of length n and m. Here, the number of atomic operations again falls in O(n), hence linear time complexity, if n, m $\in \mathbb{Z}$.

3.3 Multiplication by primary school method

For the multiplication by primary school method we used Algorithm 1.3 from the lecture notes[1].For 2 words of length n, the algorithm runs in $O(n^2)$ time.

Algorithm 1.3 (Naive Multiplication Algorithm)

Input:		the radix $b \geq 2$, and the radix-b-representations $[x]_b = [x_{m-1}, \ldots, x_0]_b$, $[y]_b = [y_{n-1}, \ldots, y_0]_b$ (without leading zeroes) of numbers $x, y \in \mathbb{N}$
Output:		the radix-b-representation $[z]_b = [z_{k-1}, \ldots, z_0]_b$ (without leading zeroes) of $z = xy$
Step 1:	1.1	$z_i \leftarrow 0$ for $0 \leq i < m+n-1$
Step 2:	2.1	for $i = 0, 1, \ldots, m-1$ do
	2.2	$\quad c \leftarrow 0$
	2.3	$\quad$ for $j = 0, 1, \ldots, n-1$ do
	2.4	$\quad\quad t \leftarrow z_{i+j} + x_i y_j + c$
	2.5	$\quad\quad c \leftarrow \left\lfloor \frac{t}{b} \right\rfloor$
	2.6	$\quad\quad z_{i+j} \leftarrow t - cb$
	2.7	$\quad z_{i+n} \leftarrow c$
Step 3:	3.1	if $z_{m+n-1} = 0$ then $k \leftarrow m+n-2$ else $k \leftarrow m+n-1$
	3.2	output $[z_{k-1}, \ldots, z_0]_b$

4 Modular Arithmetic

4.1 Multiplication (mod m)

Our algorithm is based on Algorithm 2.9 from the lecture notes[2]. It uses the algorithms we created for integer multiplication and integer subtraction. The procedure returns a correct result most of the time as illustrated below. Its limitations are discussed in Section 5.1

```
Input the radix: 10
Input the operation: multiplication
Input the 1st number: 5
Input the 2nd number: 7
Input the modulus: 23
12
Process returned 0 (0x0)   execution time : 15.998 s
Press any key to continue.
```

```
Input the radix: 16
Input the operation: multiplication
Input the 1st number: 2B
Input the 2nd number: 1
Input the modulus: 1A
11
Process returned 0 (0x0)   execution time : 80.093 s
Press any key to continue.
```

4.2 Addition (mod m)

In order to implement the modular addition algorithm we used the lecture notes[2] from section 2.2.2. Our program is computing the sum of two numbers and then calculates the reminder in the following way: if the sum is smaller than m the code outputs the number. Otherwise it outputs $s - m$.

4.3 Subtraction (mod m)

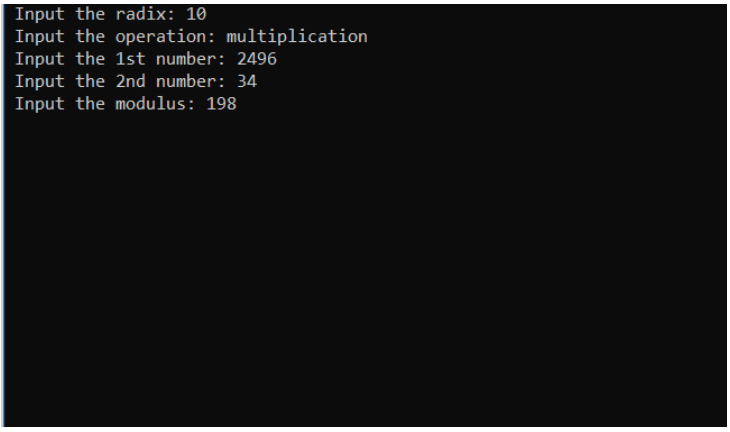
As the addition, to solve modular subtraction we used the lecture notes [2] and the program works alike for these 2. The number of atomic operations for modular subtraction and addition is O(n), so the complexity is linear.

5 Limitations

We will submit the code for the algorithms that are not fully working in separate files

5.1 Modular Multiplication (mod m)

Our modular multiplication algorithm is limited by its incapability to process large numbers as illustrated below.



5.2 Karatsuba multiplication

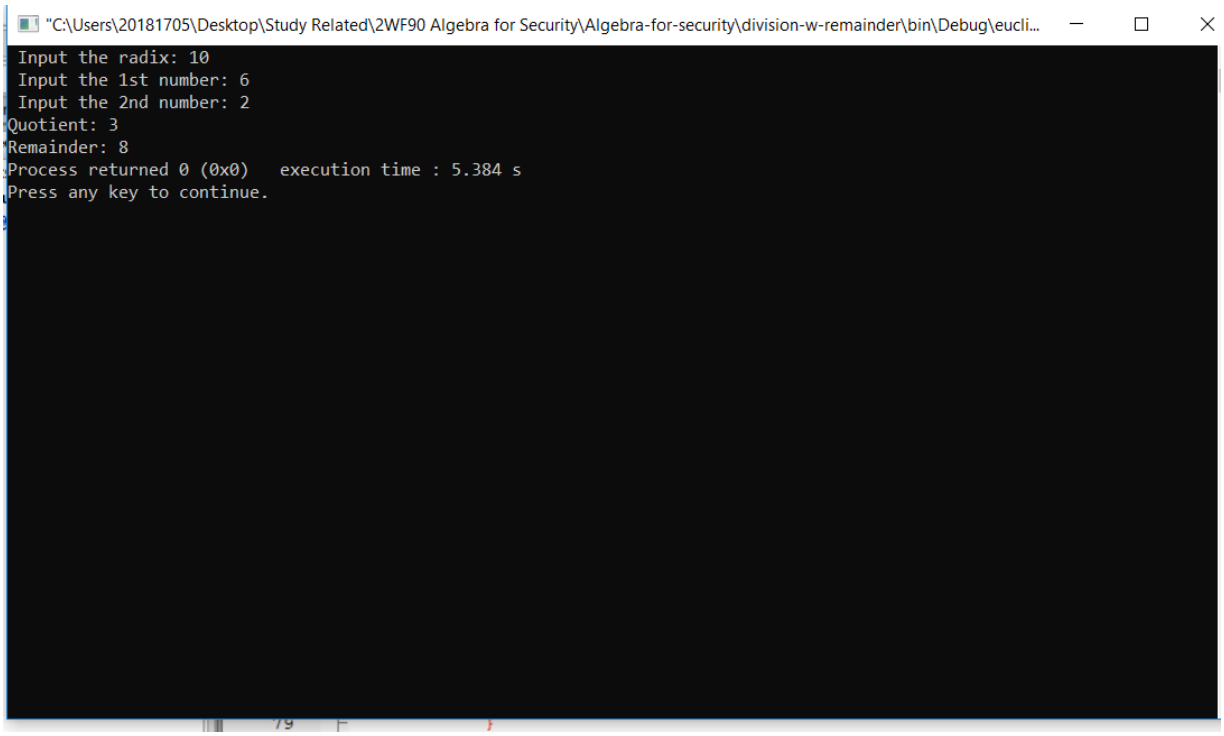
For the Karatsuba multiplication we tried to use both the lecture notes and some additional information that we found on the internet in order to implement the algorithm. However, at the moment the code only works for one digit numbers in base 10. We tried to improve it in order cover more examples but unfortunately we could not find the right approach in order to solve this inconvenience.

5.3 Eucledian Algorithm

We decided to implement the Eucledian Algorithm by using our previous algorithms for integer addition, multiplication and division with remainder and use them as functions in the implementation of Euclid's algorithm. However, we were not able to implement the algorithm for division with remainder, which hindered our work on the Eucledian algorithm. Due to the lack of a fully working division procedure our efforts to proceed with Euclid's algorithm were in vain.

5.3.1 Division with remainder

At first we tried implementing the division with remainder algorithm as given in Section 1.3.3 of the the lecture notes[1]. Since we were not able to figure out the required implementation details on our own, we looked further in the literature. We tried to implement the division with remainder for large integers procedure given in Section 3.3.4 of the book 'A Computational Introduction to Number Theory and Algebra'[3]. After exhaustive work on it, however, we only managed to make the algorithm work for radix 10 and division by 2 and even then, the algorithm still would not calculate the remainder correctly as shown in the example below.



As an alternative we were willing to implement an algorithm that subtracts the divisor y from the dividend x until the remainder r is smaller than y and count the number of subtractions n (x = n*y + r). However, due to time constraints the implementation of this algorithm was beyond our capabilities.

5.4 Modular reduction

The modular reduction works only for base 10. We could not find an error in our code. We were using algorithm 2.5 from "Algorithmic Number Theory"

5.5 Input and Output files

We were unable to read input from .txt files and output it onto .txt files. However, we created a user guide (Section 1) which contains instructions on how to run our algorithms.

References

[1] Benne de Weger. 2wf90 - algebra for security, part 1 - algorithmic number theory, chapter 1.

[2] Benne de Weger. 2wf90 - algebra for security, part 1 - algorithmic number theory, chapter 2.

[3] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.