DAA Assignment No. : 2
Implement 1-D look-up table to get single digit multiplication and using it to multiply two positive integers.

Lovely Digra
IIT2017090

Abhishek Vishwakarma
IIT2017091

Anuj Raj
IIT2017092

***Abstract*—Our required aim is to design and analyze an optimal algorithm to perform single digit multiplication without using multiplication,division or modulo operator. A 1-D lookup table is created to get the result by the multilpication of two single digit numbers. While designing our problem we calculated the time complexity for the algorithm and plotted graphs of no. of digits of nos. vs time elapsed.**

I. INTRODUCTION

To design and analyse aglorithm for single digit multiplication without using multiplication, division or modulo operator and use this application to multiply two positive integers.

With the advancement of technology certain basic computations is taken for granted by the programmers.One such example is arithmetic multiplication of two integers. Suppose the system hardware does not have architecture required for multiplication using '$*$' operator.

Now the question arises how to perform single digit multiplication without using an asterisk?

For this, A 1-D look-up table is created to get the result by the multiplication of two single digit numbers.

Now what is a look-up table?

A Look-Up table is an array that replaces run time computation with a simpler array indexing operation. It can be 2D,3D, single input single output, multi input multi output.

II. ALGORITHM DESIGN AND EXPLANATION

For the problem, we followed the following approach to design an algorithm:

Approach 1: 2-D array
For multiplication of two integers without using arithmetic ”*” operation, take a digit from the multiplier and check its product with every digit of the multiplicand using the look-Up table.

Since for multiplying two integers using the look-Up table main problem lies in accessing digits of a number separately.

The key idea is to use string operations and string-Streams for accessing and manipulating digits separately.

Firstly, store the product of all single digit numbers in 1-D look-Up Table.Product of a and b can be found in the look-up table in location ”9*(a-1)+(b-1)”. For calculating 9*(a-1) repeated addition comes handy. Use the look-Up table for the product of num1[i] with num2[j].

Store the intermediate result of product of num1 with each num2[j] in a 2-D Matrix, like the way a primary school students does .

Finally add the intermediate results in column fashion to get the final result.

Approach 2: Optimization
Instead of storing intermediate values in 2-D matrix concatenate the result into a string and parse it to get the intermediate result.

Now addition becomes very simple . Just add it to the variable storing the final result.

Repeat the process for every digit of num2 to get the final result. The pseudo-code for this algorithm is:-
Given: both the numbers are non negative integers.

Algorithm 1 PseudoCode for Approach 2:

```
procedure $mul(inta)$
    for all $i = 1$ to a do
        //a times repeated addn of c where is 9
    return c
procedure $multiply(inta, intb)$
    return look-Up[$mul(a-1)$+$(b-1)$]
$number1 \leftarrow$ input
$number2 \leftarrow$ input
$num1String \leftarrow$ parsed num1 to string
$num2String \leftarrow$ parsed num2 to string
$reverse(num1Str)$
$reverse(num2Str)$
for all $i = 1$ to no of digits num1 do
    $carry \leftarrow 0$
    for all $j = 1$ to no of digits num2 do
        $product \leftarrow multiply$(num1Str[i], num2Str[j])
        $product \leftarrow product$ + $carry$
        $result \leftarrow$ unit_place_of_product
        $carry \leftarrow$ tens_place_of_product
        $intermediate\_result \leftarrow concatenate(result)$
        procedure $reverse(intermediate_result)$
        $finalResult \leftarrow finalResult$ + $intermediateRes$
    $print(finalResult)$
```

III. Experimental Study

For the experimental analysis of this algorithm, we will take same value for both num1 and num2 for easy calculations and iterate a loop for assigning num1 and num2 with the loop variable. A .dat file is generated containing duration for each corresponding num value in a tabluated manner.

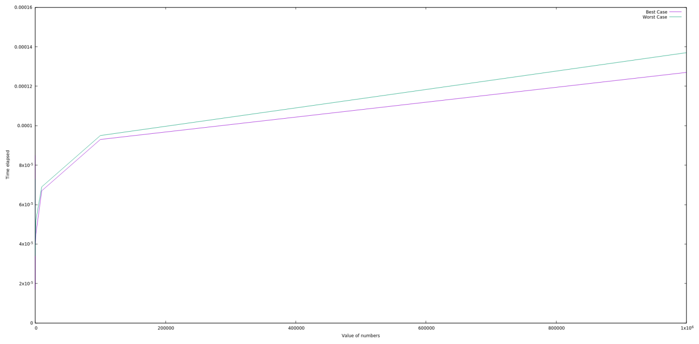

Fig. 1. Graphical Explanation

IV. Time Complexity and Discussion

The process of accessing product from the look-Up table at max 9 iterations in the repeated addition loop of mul(9) for function and 0 iterations for mul(0). This happens for every single digit pair of the multiplicant and the multiplier. Thus, the complexity depends in the number of digits of multiplier , number of digits in the multiplicant and value of the single digit of each pair of the two integers. For simplicity take same value for both the integers. That is num1 = num2 = n.

So worst case and best case are defined for total digits in the given integer.
Worst Case : each single digit pair of integers is 9.
growth rate : $O(9 * d * d)$ (see fig.1, green line)
Best Case : each single digit pair contains atleast one 0.
growth rate : $\omega(1 * d * d)$ (see fig.1 voilet line)
where d is the total digits in the given integer.

Nature of the graph: The graph Time vs Value of Number will be logarithmic of base 10 (See Fig.1) .

This is because total digits in a number is equal to $[log_1 0(num)] + 1$

V. Conclusion

We designed an algorithm to perform multiplication of two positive integers with the help of a 1-D lookup table. We also analyzed the algorithm and plotted the graph for Value of nos vs time elapsed.

One of the possible application of the project can be in designing Compiler for systems (say, embedded systems) with native processors without the architecture for multiplication operations.