Jeffrey Lansford

Lab2

CS 372

09-04-2019

## Introduction:

For this Lab we are examining the difference of multiplying binary numbers two different ways. One way is how we use to doing it with an iterative method, the other a divide and conquer algorithm. We have developed a program that incorporates both and shows the runtime of each method along side our theorical runtime we calculate in class.

## Methods:

First, we start with the iterative method as it is easier to program. The input of the method requires two Boolean vectors, x and y, that represent our binary numbers. We need to make the sizes of the two vectors the same, so we do not run into trouble with our loops. We go through the y byte first then take each true value, or 1, and create a vector for holding temporary values and copy x to it. Then we need to add zeros in front like how we space the bytes when do multiplying by hand. Then add vectors together! Logic is:

```
Vector z, intermediate

z.size = x.size + y.size

for ( y ) {

        if ( y[i] is true )

                intermediate = x

                pad intermediate with zeros depending on what y-bit we are on

                z + intermediate

}
```

Then we can do the divide and conquer strategy. We based the program around this pseudocode given by our book:

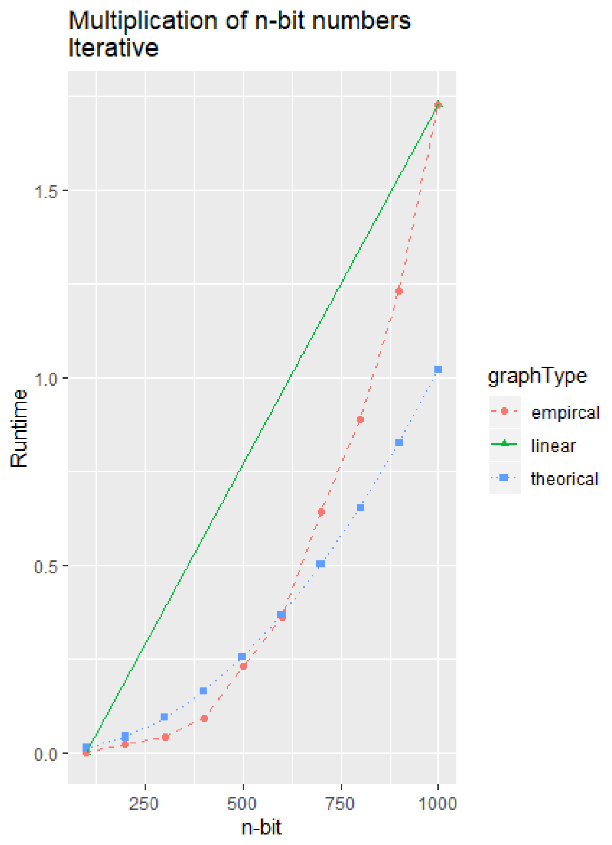
```
If n == 1: return xy

$X_L$ , $X_R$ = leftmost and rightmost bits of x
```

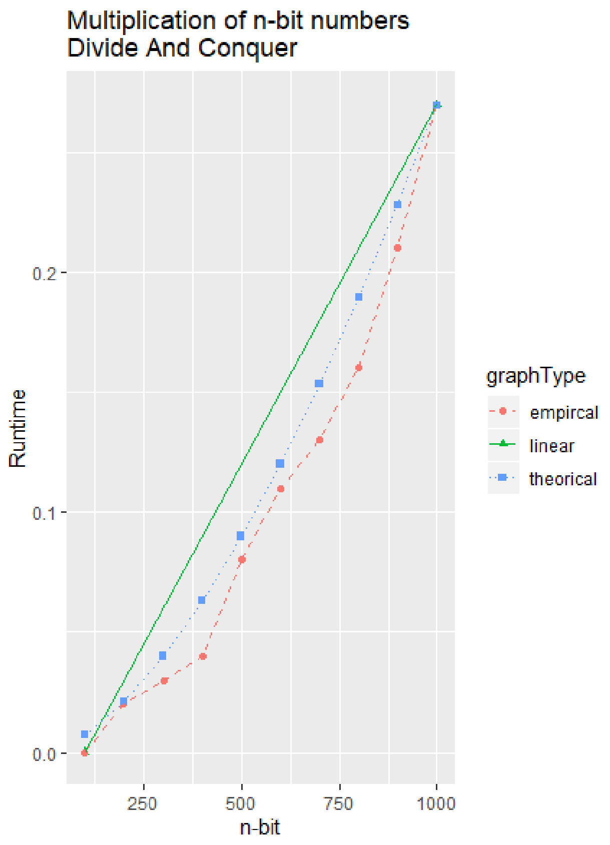
```
$Y_L$ , $Y_R$ = leftmost and rightmost bits of y

P1= multiply($x_L$,$y_L$)
P2= multiply($x_R$,$y_R$)
P3 = multiply($x_L$+$x_R$, $y_L$ + $y_R$)
Return P1 x $2^n$ + (P3 - P1 - P2) x $2^{n/2}$ + P2
```

With this, we can create the function with some work arounds for C++ and vectors. We check for size of 1 for our two parameters, x and y as before, and return xy. Then I make the two vectors even sizes because the algorithm does not correctly without it splitting it evenly, though it can be fixed easily with padding zeros to the vectors. Then we get our $x_L$, $x_R$, $y_L$, $y_R$ vectors to assign the left and right sides of the bytes. Then simply do the recursive calls like the pseudocode. There are some draw backs as you can not pass temporary vectors in C++, so we had to create temporary vectors for it to hold intermediate values. Then we do our test cases and it works! Then we develop our R code for generating our graphs of the runtime to see the difference. I had to use some libraries to accomplish combining graphs together to get it the way we are supposed to do it. The code that generates the theorical graph gets its coefficient from subtracting .0000001 from the coefficient till the difference of the actual runtime and the produced number are less than .1. Not the best way of doing this, but I could not figure out how else to do this as doing regression lines in R is not very good on non-linear models.

## Results:

When we run the program, we get our graph that shows both functions runtime on the same output:





The theorical functions are $c_1x^2$and $c_2x^{log_2 3}$ where $c_1$ = 6.600001e-07 and $c_2$ = 4.741e-06

On my machine, the two graphs take about the same time around an input size of 400. Even though the divide and conquer method is faster, it can be a mess to deal with. My implementation of the divide and conquer creates a lot of vectors that could take up a lot of memory doing the tasks. Also, the divide and conquer solution is messier to look at then that of the iterative function. The iterative function is cleaner but takes more time. So, if you are working with big numbers, then I would recommend the divide and conquer method, but normally aim for the iterative.  My machine is a that ran the program is a Intel i7-7700HQ @ 2.80GHz with 16GB of memory.

## Discussions:

This lab is considerably hard than the last and had more frustrating bugs to deal with. The iterative function came prettier fast once you know how to do it, but the divide and conquer function gave me many headaches. I made some helper functions, so I did not have to repeat code through this program, but later finding out, that what hurt me in the end. I was subtracting wrong the whole time after looking into why my output is always off. The R code also gave my trouble and lots of research into graphing came about. I would wish there was some help with graphing as I had to go through many libraries to find one that did what I wanted.

## Conclusions:

We created a program that multiples two binary numbers two different ways at two different runtimes. Taking what we learned from class with the divide and conquer theorem and apply it the program and analyze about how different it is to the normal way of multiplying. We visualize the data we collected to see these differences and make our own conclusions to whether the divide and conquer method is worth it. Looking towards the future, we know to examine some usual tasks that we might program and think it is the best solution. We can now take theorical runtimes and apply to actual runtimes to get close results.