Homework II - Analysis of Algorithms

Edwing Alexis Casillas Valencia

September 21, 2025

Abstract

In this homework we analyze the Gauss's trick and the use of a XOR into a classic multiplication program to reduce the execution time, measuring it with the "sys/time.h" library. Also, we practice the complexity calculation with three recursion exercises, calculating the upper and lower bounds of the asymptotic.

Index Terms

Gauss's Trick, Time complexity, Recursion, Mater Method

PROBLEM 1

(50 pts) We have the following code "Classic_Bit_Multiplication.c" which implements the old version of bit multiplication. Your task are two for this:

(a) Implement the Gauss Trick, as you can see we have something quite simple when reaching the base case:

```
if (n == 1){
return a*b ;
}
```

This uses the ALU for integer multiplication, but it can be substituted for something more efficient using xor because. For example, we can use xor for unsigned integer addition:

```
unsigned i n t bitwise_add ( unsigned i n t x , unsigned i n t y ) {
unsigned int carry ;
while ( y != 0) {
carry = x & y ;
x = x ^ y ;
y = carry << 1 ;
}
return x ;
}
```

(b) Please compare the time complexity between both implementations by using the "sys/-time.h" using gettimeofday() for higher precision.

Answer:

We can observe that temp3 and temp4 are always in addition, and we can calculate that term before entering the return of the function, but is not as simple of doing that addition. Also, we can see that the calculation for the four terms is similar to a second grade equation with the following form.

$$(a+b)(a+b) = a^2 + 2ab + b^2 \tag{1}$$

Where a^2 is a_l * b_l, b^2 is a_r * b_r and $2ab$ is the addition of temp3 and temp4. It's a summation of 4 terms that we can reduce to 3, passing the addition of term3 and term4 in one term calculation, which I'm going to call new_temp and has the following form:

$$newtemp = multiplication(a_l + a_r, b_l + b_r, size, mask) \tag{2}$$

This new temp gives me the complete result from the multiplication of a with b, but due we have bit-to-bit operations with each temp at the moment of return the result of each function call, we now have to obtain only the addition of the middle term (2ab). To do this, we just need to subtract temp1 and temp2 of it.

$$op = newtemp - temp1 - temp2 \tag{3}$$

We this, now we can change the return of the function with the following

$$temp2 + ((op) << size) + (temp1 << n) \tag{4}$$

If we count the number of calls from the function to itself before doing this modification, we can see that it does eight times; after the modification it does six times. This matches with the Gauss's trick because Gauss use a summation of the natural numbers, and we use a summation of products to obtain our result.

To substitute the a*b section with a XOR, we can use the left and right shifts on our number and use the bitwise_add on them to simulate the AND behavior. Why?, because we want that the bit-to-bit operation returns us 1 when our tow terms are 1.

```
unsigned long long XOR(unsigned long long a, unsigned long long b)
{
    unsigned long long carry;
    while (b != 0){
            carry = a & b;
            a = a ^ b;
            b = carry << 1;
        }
    return a;
}

unsigned long long multiplication(unsigned int a, unsigned int b, int n, unsigned int in_mask)
{
    if (n == 1) {
        unsigned long long result = 0;
        while (b > 0)
        {
            if (b & 1)
            {
                result = XOR(result, a);
            }
            a = a<<1;
            b = b>>1;
        }
        //return a*b;
        return result;
    }
```

Fig. 1. XOR multiplication

The execution time using both implementations where approx. 275 microseconds, without the implementations where approx. 320 microseconds. In some runs, the program without the implementations where quicker than the one with the implementations, but the results fluctuate too much compared with the one with both implementations.

PROBLEM 2

(50 Points) Give asymptotic upper and lower bounds for T(n) in:

(a) $T(n) = T(n-1) + n^c$, where $c \geq 1$ is a constant.
(b) $T(n) = T(\sqrt{n}) + 1$
(c) $T(n) = 5T\left(\frac{n}{2}\right) + \frac{n}{logn}$

Answer:

- $T(n) = T(n-1) + n^c$, where $c \geq 1$ is a constant.

We can calculate the first terms to see the behavior of the equation. For n=2 we have

$$T(2) = T(1) + 1^c \tag{5}$$

for n=3 we have

$$T(3) = T(2) + 2^c = T(1) + 1^c + 2^c \tag{6}$$

for n=4 we have

$$T(4) = T(3) + 3^c = T(1) + 1^c + 2^c + 3^c \tag{7}$$

We can notice that we have a sum of powers, and $j \leq n^c$, for all $j \leq n$, so we'll have

$$T(n) \leq T(0) + n(n^c) \leq n^{c+1} \tag{8}$$

This give us the following complexity for the upper bound

$$T(n) = O(n^{c+1}) \tag{9}$$

For the lower bound we have

$$\sum_{i=1}^{n} j^c \geq \int_0^n x^c = \frac{n^{c+1}}{c+1} \tag{10}$$

Due x^c is increasing and a convex function in $[0, \infty)$, we'll have

$$T(n) = \Omega(n^{c+1}) \tag{11}$$

We can observe that the upper and lower bounds are equals, so we can say that we have a tight

$$T(n) = \Theta(n^{c+1})$$

- $T(n) = T(\sqrt{n}) + 1$.

Let N0 > 1 a constant, and suppose that t(n) satisfies

$$t(n) = t(\sqrt{n}) + 1 \qquad (n > N0) \tag{12}$$

with base condition $t(n) = \Theta(1)$ for $n \leq N_0$. Then, for all n $> N_0$,

$$t(N0) + \frac{lnlnn - lnlnN0}{ln2} \leq t(n) \leq t(N0) + 1 + \frac{lnlnn - lnlnN0}{ln2} \tag{13}$$

In particular, $t(n) = \Theta(loglogn)$

Proof

Let $x = ln\, n$ and $x_0 = lnN_0$, define $s(x) = t(e^x)$. The recurrence is as follow

$$s(x) = s(\frac{x}{2}) + 1 \tag{14}$$

After process k times, we have:

$$s(x) = s(\frac{x}{2^k}) + k \tag{15}$$

the process stops when $\frac{x}{2^k} \leq x_0$, that is to say, when $2^k \geq \frac{x}{x_0}$. Consider $g(u) = \frac{1}{uln2}$, which is decreasing in $[0, \infty)$. Each step reduce x to x/2 it "consumes" exactly

$$\int_{x_0}^{x} g(u)du = \int_{x/2}^{x} \frac{du}{uln2} = \frac{lnx - ln(x/2)}{ln2} = 1 \tag{16}$$

Applying the integral test (with a possible step overshoot in the last interval) we have

$$\int_{x_0}^{x} g(u)du \leq k \leq 1 + \int_{x_0}^{x} g(u)du \tag{17}$$

As

$$\int_{x_0}^{x} g(u)du = \frac{lnx - lnx_0}{ln2} = \frac{lnlnn - lnlnN_0}{ln2} \tag{18}$$

returning at t(n)=s(x) we obtain

$$t(N_0) = \frac{lnlnn - lnlnN_0}{ln2} \leq t(N_0) + 1 + \frac{lnlnn - lnlnN_0}{ln2} \tag{19}$$

The additive constants shows that $t(n) = \Theta(loglogn)$

- $T(n) = 5T(\frac{n}{2}) + \frac{n}{logn}$

We can observe that it has the form of the Master Method, so we have a=5 and b=2, with this, our watershed function is calculated as

$$n^{log_b a} = n^{log_2 5} \approx n^{2.321928095} \tag{20}$$

Evaluating with n=2 the watershed function and the driver function, we observe that the driver function is slower than the watershed, so it corresponds to the first case of the Master Method

$$T(n) = \Theta(n^{log_b a}) = \Theta(n^{log_2 5}) \approx n^{2.321928095} \tag{21}$$