COMP3121 Homework Q1

Arth Sanskar Patel
z5228942

June 29, 2020

1 Answer

Here n can be written as

$$n = 2^{k_1} + 2^{k_2} + \ldots + 2^{k_m}$$

. Since any number can be written in terms of power of 2, it will take $logn$ number of multiplications to get the answer. For example we can calculate M^8 in the following way

$$M^2 = M * M$$

$$M^4 = M^2 * M^2$$

$$M^8 = M^4 * M^4$$

As you can see this will take $log_2 8 = 3$ multiplication. Hence rewriting n as shown above we get:

$$M^n = M^{2^{k_1}} * M^{2^{k_2}} \ldots * M^{2^{k_m}}$$

Since all these multiplications will take time $O(logn)$, we will get the overall time complexity to be $O(logn)$. For any largen umber, we can repeat the same process of squaring again and again to eventually get to it. Even when n is a odd number, we can still calculate it using the same method. For example let n = 15, it can be written as

$$n = 2^3 + 2^2 + 2^1 + 2^0$$

And all of these multiplications will be of $O(logn)$ time and hence the total will always be $O(logn)$. Psuedo code for the algorithm is given below.

```
def multiplyFunction(int number, int power):
    if power = 0:
        return 1
    if power = 1:
        return number
    if power is even:
        return multiplyFunction(number, power/2)*multiplyFunction(number, power/2)
    if power is odd:
        return number*multiplyFunction(number, (power-1)/2)*multiplyFunction(number, (power-1)/2)
```