IN.3028 : Algorithmics - Assignment 3

Antoine Tissot, Marc Akuatse, Samim Umut Ökten

October 10, 2023

1 Chapter 1 : Foundation

1.1 *O(lg n)* exercise search, Throwing eggs

Determine F with lg N throws (Worst case)

1. Throw the first egg from the halfway point of the building → N/2.
   (a) If it break, it means F is below N/2. So go halfway between the ground and N/2 → N/4
   (b) If it doesn't break, it means F is above N/2. So go halfway between N/2 and N for the next throw → 3N/4
2. Repeat this process, narrowing down the range in which F lies by half with each throw.

This algorithm is essentially based on the binary search. Since the search range is divided by half with each throw, it will take Lg N throws to determine F.

Reduce the cost to 2lg F

1. Throw an egg from the 1st floor, if it doesn't break move to the 2nd floor
2. Throw an egg from the 2nd floor, if it doesn't break move to the 4th floor
3. Throw an egg from the 4th floor, if it doesn't break move to the 8th floor
4. ...

During this phase, we go up floor exponentially until the egg break. When it does, we have found an upper bound for F. If the egg broke at floor 2^k, this means F is somewhere between 2^{k-1} and 2^k. In the worst case, the number of throws here is then lg F.

To find F between 2^{k-1} and 2^k, we can again use the binary search as presented in the first strategy.We saw that in the worst case, the number of throws is also lg F. Therefore, this strategy reduce the cost of throws to 2lg F.

1.2 Fibonacci: the first log-log scale plot (Implementation)

Implementation provided in the src and tests files.

We can notice on the graph that for the first 26 Fibonacci numbers, there is no significant differences. However, after those simple iterations, we notice an exponential growth with the logarithmic time complexity. This means that we should always consider the efficiency of our loop/method to make the right decision and take the less time consuming way.

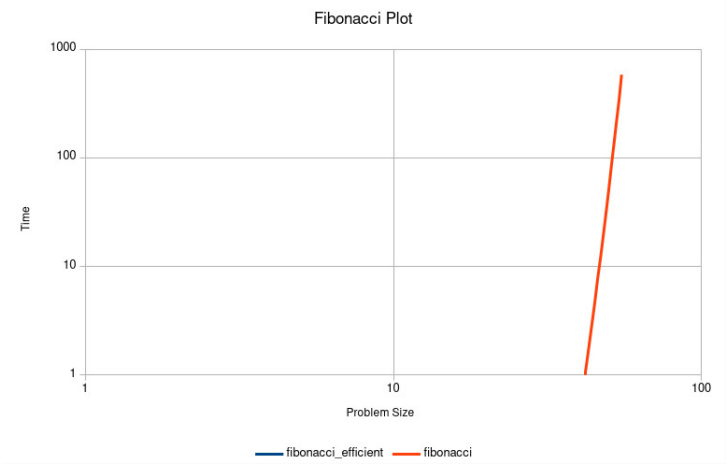

Figure 1: Fibonacci Normal Plot

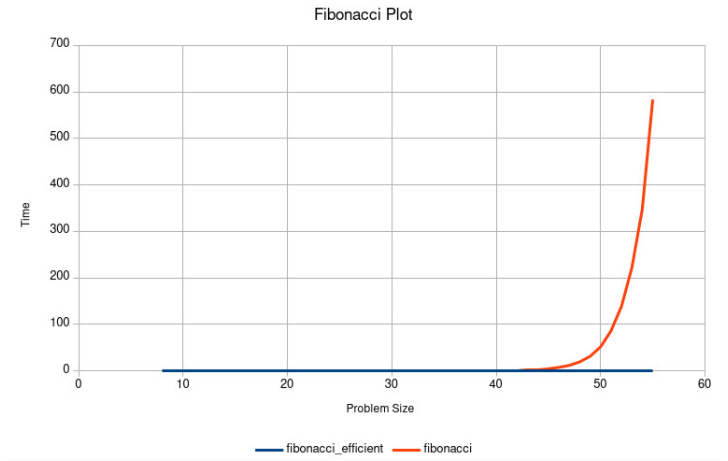

Figure 2: Fibonacci Log Plot

1.3 Binary search algorithm (Implementation)

Implementation provided in the src and tests files.

To find the number of 0 and 1 in a sorted array using the binary search algorithm

```
function findNumberOfZero(array):
    low = 0
    high = length(array) - 1

    while low <= high:
        mid = (low + high) / 2
        if array[mid] == 1 and array[mid - 1] == 0:
            return mid
        else if array[mid] == 0:
            low = mid + 1
        else:
```

```
            high = mid - 1

    return -1

TP = findNumberOfZero(array)
number_of_zeros = TP
number_of_ones = length(arr) - TP
```

Order of growth is $O(\log(n+m))$

1.4 ThreeSum (Implementation)

From the slides we know that our slower Three Sum Zero implementation has a cubic (N^3) order of growth. If we didn't know this, we would definitely guess that it had a linearithmic one. This is probably because we don't have enough data to plot.

On the other hand if we only consider the plots for our fast Three Sum Zero implementation, it, again, looks like a linearithmic growth. But we know that because it has a nested array accesses, N^2, and one $\log N$ access in the inner loop, it should be a quadratic growth.

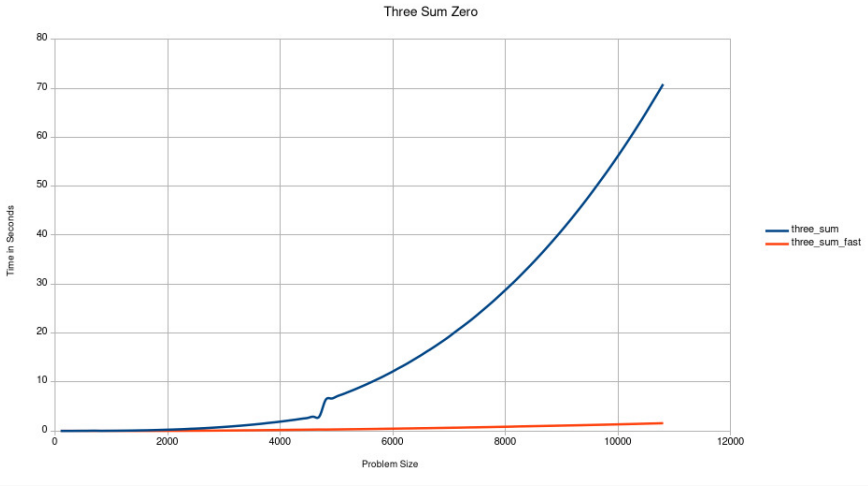

Figure 3: ThreeSum Normal Plot

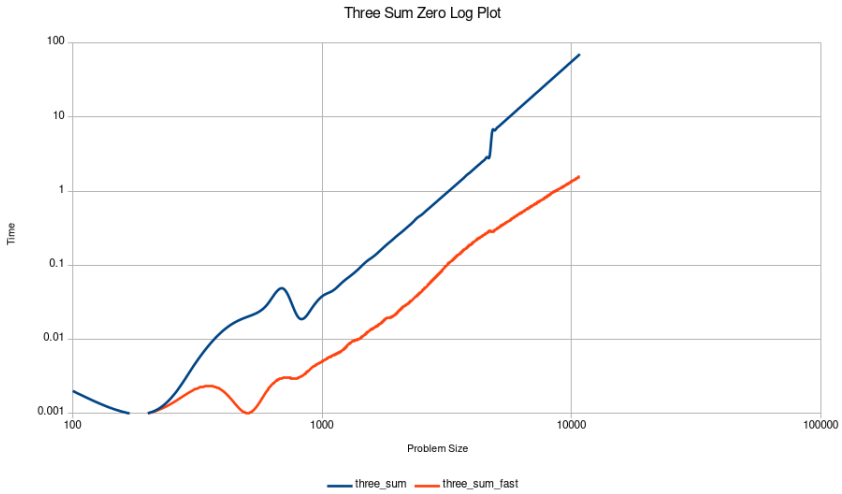

Figure 4: ThreeSum Log Plot