Algorithms and Data Structures

Sheet 5

Valdrin Smakaj

Problem 5.1 *Fibonacci Numbers* (12 points)

(a) (5 points) Implement all four methods to compute Fibonacci numbers that were discussed in the lecture: (1) naive recursive, (2) bottom up, (3) closed form, and (4) using the matrix representation.

(b) (3 points) Sample and measure the running times of all four methods for increasing n. For each method, stop the sampling when the running time exceeds some fixed amount of time (same for all methods). If needed, you may use classes or structs for large numbers (self-written or library components). Create a table with your results (max. 1 page).
Hint: The "gap" between samples should increase the larger n gets,
e.g. $n \in \{0, 1, 2, 3, 4, 5, 6, 8, 10, 13, 16, 20, 25, 32, 40, 50, 63, ...\}$.

(c) (2 points) For the same n, do all methods always return the same Fibonacci number? Explain your answer.

(d) (2 points) Plot your results in a line plot, such that the four approaches can be easily compared. Briefly interpret your results.
Hint: Use logarithmic scales for your plot.

Answers to sub-problems a,b and d are attached inside the folder I submitted

(c) In the beginning, for smaller values of n, all the methods are going to yield same values and correct but afterwards, by increasing and further more calculations of bigger and different values of N, one of the 4 methods will differ with a slightly different outcome for the N calculated value due to its floating point precision error. This method is the Closed Form method since also mentioned in the slides it has a margin of error for different and bigger values of N as a cause of the floating point precision.

Problem 5.2 *Divide & Conquer and Solving Recurrences* (10 points)

Consider the problem of multiplying two large integers a and b with n bits each (they are so large in terms of digits that you cannot store them in any basic data type like `long long int` or similar). You can assume that addition, substraction, and bit shifting can be done in linear time, i.e., in $\Theta(n)$.

(a) (2 points) Derive the asymptotic time complexity depending on the number of bits n for a brute-force implementation of the multiplication.

(b) (4 points) Derive a Divide & Conquer algorithm for the given problem by splitting the problem into two subproblems. For simplicity you can assume n to be a power of 2.

(c) (1 point) Derive a recurrence for the time complexity of the Divide & Conquer algorithm you developed for subpoint (b).

(d) (2 points) Solve the recurrence in subpoint (c) using the recursion tree method.

(e) (1 point) Validate the result you got in subpoint (d) by using the master theorem to solve the recurrence again.

Solution

(a) Since the overall time complexity of multiplication is $\Theta(n^2)$, considering the case with more N-bits such that it cannot be held by even larger data types such as long long int or similar ones, would still yield the same time complexity since the bit shifting and the addition is still going to be linear and the multiplication of each bit of number one by each bit of the other number will still give us the same time complexity. Therefore, in a more formal and mathematical way we can formulate this as:

$$\begin{aligned} T(n) &= T_{\text{multiplication}}(n) + T_{\text{addition}}(n) = \Theta(n^2) + \Theta(n^2) \\ &= 2\Theta(n^2) \\ &= \Theta(n^2) \end{aligned}$$

(b) In this part I used reference from https://www.geeksforgeeks.org/karatsuba-algorithm-for-fast-multiplication-using-divide-and-conquer-algorithm/ page in order to assist me. Although I got the help, I genuinely understood the concept and know how to approach to the solution of this question.

Since the basic Idea is to split the number into two sub-parts, we approach in the following:

```
X =  Xl*2^(n/2) + Xr     [Xl and Xr contain leftmost and rightmost n/2 bits of X
Y =  Yl*2^(n/2) + Yr     [Yl and Yr contain leftmost and rightmost n/2 bits of Y
```

The product XY can be written as following.

```
XY = (Xl*2^(n/2) + Xr)(Yl*2^(n/2) + Yr)
   = 2^n XlYl + 2^(n/2)(XlYr + XrYl) + XrYr
```

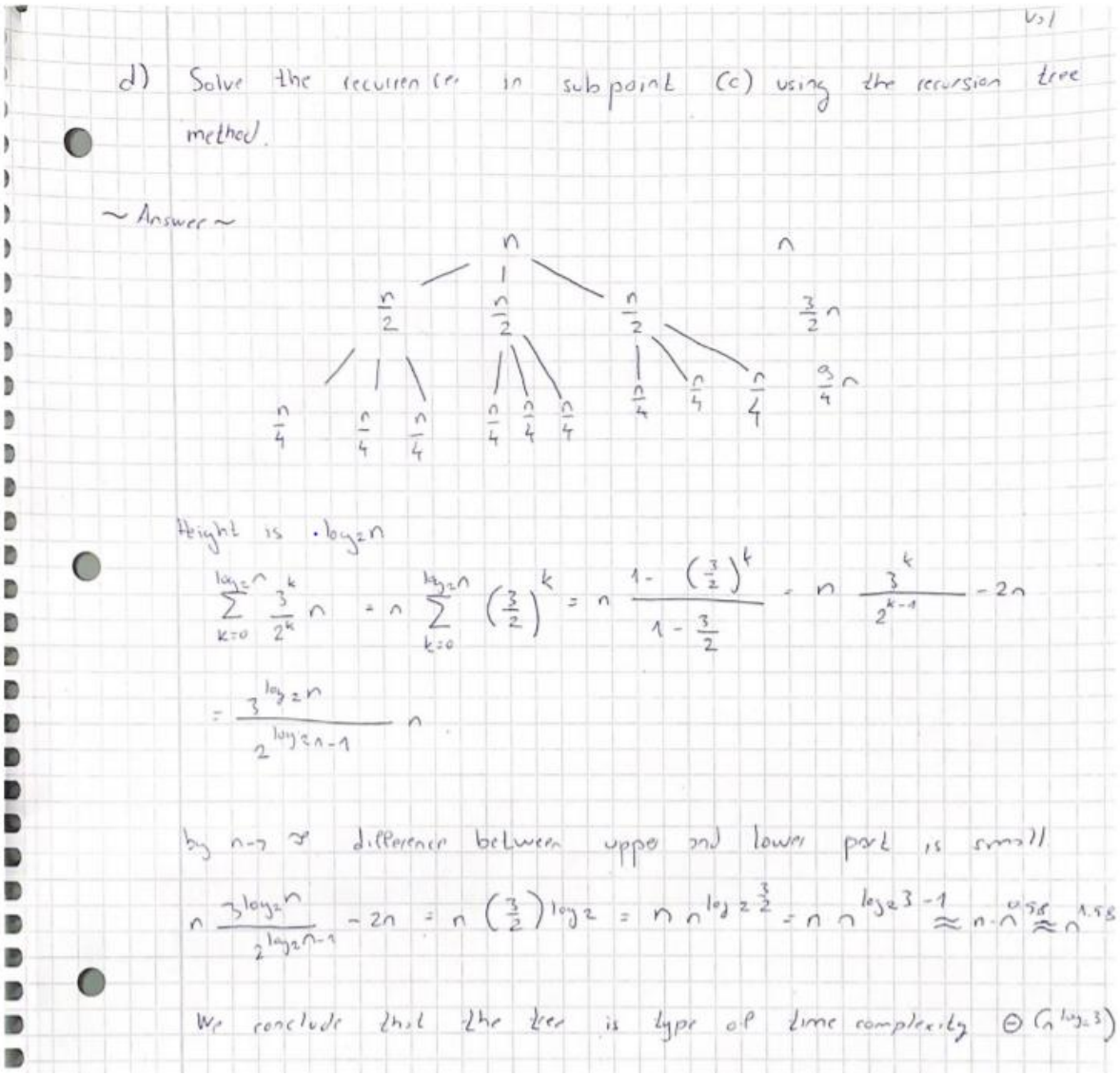
Where as we can see that this is not really a big help since we split the problem of size n to 4-sub-problems of size n/2 which when summed up still lead to the result of $\Theta(n^2)$. But if we observe the middle part in a slightly different way, we can transform it on some other way such that we get one less multiplication of n/2. Therefore we can transform the middle part into:

```
XY = 2^n XlYl + 2^(n/2) * [(Xl + Xr)(Yl + Yr) - XlYl - XrYr] + XrYr
```

With above trick, the recurrence becomes T(n) = 3T(n/2) + O(n) and solution of this recurrence is $O(n^{1.58})$.

(c) For the time complexity as derived from the point b answer, we got $T(n) = 3T(n/2) + O(n)$ which is part of $O(n^{1.58})$

(d)



(e) Now we already got the time complexity $T(n) = 3T(n/2) + O(n)$. Solving it by using the Master's theorem we proceed:

$a = 3$
$b = 2$

$$n^{\log_2 3} = n^{1.58}$$
$$f(n) = n$$
$$f(n) = O(n^{\log_2 3 - \epsilon}) = O(n^{1.58-\epsilon}) \text{ where } \epsilon = \log_2 3 - 1 \approx 0.58$$

Case 1: $f(n)$ is polynomially smaller than $n^{\log_2 3}$
Result: $T(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.58})$