Assignment 5 | Algorithms and Data Structures

Sakar Gopal Gurubacharya

Problem 5.1: *Fibonacci Numbers*

a. **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.**

   All four methods to computer Fibonacci numbers that were discussed in the lectures have been implemented in *a5_p1.cpp*.

b. **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 (selfwritten 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 ∈ {0, 1, 2, 3, 4, 5, 6, 8, 10, 13, 16, 20, 25, 32, 40, 50, 63, ...}.**

   In the main method of *a5_p1.cpp,* the library **chrono** has been used and with that, times of all four methods for increasing n has been measured. I have kept the time limit to 1.0 seconds, which is a lot of time for bottom up, closed form and matrix representation. ‘n’ increases by 5 in each step.

c. **For the same n, do all methods always return the same Fibonacci number? Explain your answer.**

   All methods return the same value of n, but in closed form, the higher the value gets, there is rounding error and it is the known drawback of this method.

d. **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.**

   No plot.

Problem 5.2: *Divide & Conquer and Solving Recurrences*

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 Θ(n).

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

Multiplying n-bit numbers like *123456789 x 987654321* using brute-force.

The brute-force implementation of multiplication expresses the product as the sum of multiple operations. Multiplying by powers of 10 shifts the result, so each of these multiplications is multiplying by a 1-digit number, which takes O(n) time. Adding two O(n) digit numbers takes O(n). We simply go through the digits and perform carries. We're doing one addition for each digit of the second number, so the total cost is **$O(n^2)$.**

b. 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.

A number can be broken down to two parts, one part that is multiplied with powers of 10, and the other part which fills the remaining difference by addition.

For e.g., a number like 2374 can be divided to:

$$(23 \times 10^{\frac{4}{2}}) + 74$$

which is like saying:

$$(a \times 10^{\frac{n}{2}}) + b$$

That is the general representation of any number. When we multiply numbers, we have two of them like:

$$\left(\left(a \times 10^{\frac{n}{2}}\right) + b\right) \times \left(\left(c \times 10^{\frac{n}{2}}\right) + d\right)$$

$$\left(a \times 10^{\frac{n}{2}}\right)\left(c \times 10^{\frac{n}{2}}\right) + d\left(a \times 10^{\frac{n}{2}}\right) + b\left(c \times 10^{\frac{n}{2}}\right) + bd$$

$$\left(ac \times 10^{\frac{n}{2}}\right) + (ad + bc) \times 10^{\frac{n}{2}} + bd$$

There are still 4 multiplications to be done and the time complexity remains $O(n^2)$. The recurrence formula is

$$T(n) = 4T(\frac{n}{2}) + \theta(n)$$

Multiplication takes more time to execute than additions and the $(ad + bc)$ from above can be calculated using additions

$$ad + bc = (a + b)(c + d) - ac - bd$$

$$\cancel{ac} + ad + bc + \cancel{bd} - \cancel{ac} - \cancel{bd}$$

$$\boldsymbol{ad + bc}$$

Implementing what we achieved above in the standard multiplication procedure to reduce multiplications from 4 down to 3.

$$\left(\left(a \times 10^{\frac{n}{2}}\right) + b\right) \times \left(\left(c \times 10^{\frac{n}{2}}\right) + d\right)$$

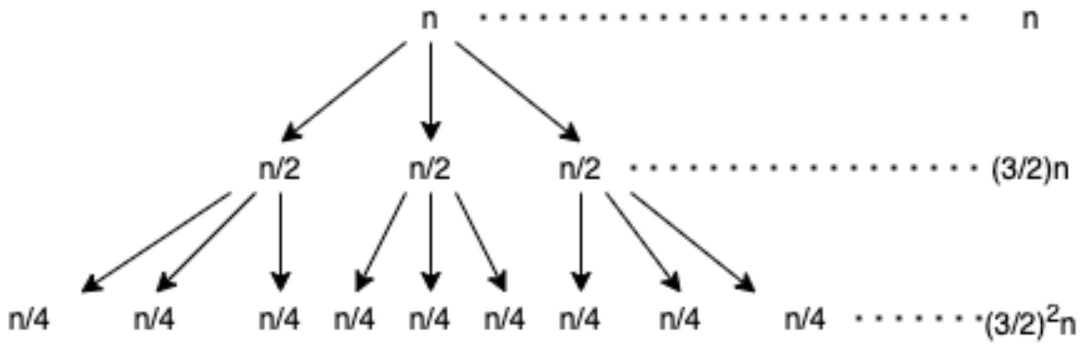
$$\boldsymbol{\left(ac \times 10^{\frac{n}{2}}\right) + \left((a + b)(c + d) - ac - bd\right) \times 10^{\frac{n}{2}} + bd}$$

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

We were able to reduce the number of multiplications from 4 to 3 so the new recurrence becomes

$$\boldsymbol{T(n) = 3T(\frac{n}{2}) + \theta(n)}$$

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



The height = $\log_2 n$. The next subtree is half of its parent.

If we were to add all the subproblems from all subtree, we would get,

$$\left(\frac{3}{2}\right)^0 n + \left(\frac{3}{2}\right)^1 n + \left(\frac{3}{2}\right)^2 n \left(\frac{3}{2}\right)^3 n + \cdots + \left(\frac{3}{2}\right)^n n$$

The above can be generalized as

$$\sum_{k=0}^{h} (\frac{3}{2})^k n$$

We know the height is $\log_2 n$, so

$$n \sum_{k=0}^{\log_2 n} (\frac{3}{2})^k$$

We know the sum of a geometric sequence is,

$$= n\left(\frac{1\left(1-\left(\frac{3}{2}\right)^{\log_2 n}\right)}{\left(1-\left(\frac{3}{2}\right)\right)}\right)$$

$$= n\left(\frac{1-n^{\log_2 \frac{3}{2}}}{-0.5}\right)$$

$$= n\left(\frac{n^{0.58\ldots}-1}{0.5}\right)$$

$$= \frac{n^{1.58\ldots}-n}{0.5}$$

$$= \frac{n^{1.58\ldots}}{0.5} - \frac{n}{0.5}$$

$$= 2n^{1.58\ldots} - 2n$$

Therefore,

$$T(n) = n^{\log_2 3}$$

$$\boldsymbol{\theta(n^{\log_2 3})}$$

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

$$T(n) = 3T(\frac{n}{2}) + \theta(n)$$

$$a = 3,$$
$$b = 2,$$
$$f(n) = n$$

$$T(n) = n^{\log_b a}\,[u(n)]$$

We know n, a and b but we don't know u(n)

Anyways,

$$\boldsymbol{T(n) = n^{\log_2 3}\,[u(n)]}$$

u(n) depends on h(n)

$$h(n) = \frac{f(n)}{n^{log_b a}}$$

$$h(n) = \frac{n}{n^{log_2 3}}$$

$$h(n) = n^{-0.5859625007\ldots}$$

Comparing with kn^r, r = -0.5859… which is less than 0

r < 0 == O (1)

Therefore,

$$[u(n)] = 1$$

So,

$$T(n) = n^{\log_2 3}\,[u(n)]$$

$$T(n) = n^{\log_2 3} \times 1$$

$$\boldsymbol{T(n) = n^{\log_2 3}}$$

Finally, the time complexity of this new Divide & Conquer algorithm of multiplication:

$$\boldsymbol{\theta(n^{\log_2 3})}$$