CS 301 - Assignment 1

Rebah Özkoç
29207

March 20, 2022

Problem 1

I used the master method for solving a, b, and c parts.

a) $T(n) = 2 * T(n/2) + n^3$
$f(n) = n^3 = \Omega(n^{\log_2 2+\epsilon})$ for some $\epsilon > 0$
$=> T(n) = \Theta(f(n)) = \Theta(n^3)$

b) $T(n) = 7 * T(n/2) + n^2$
$f(n) = n^2 = O(n^{\log_2 7-\epsilon})$ for some $\epsilon > 0$
$=> T(n) = \Theta(n^{\log_2 7})$

c) $T(n) = 2 * T(n/4) + n^{1/2}$
$f(n) = n^{1/2} = \Theta(n^{\log_4 2-\epsilon})$ for some $\epsilon > 0$
$=> T(n) = \Theta(n^{1/2} * \log_2 n)$

d) $T(n) = T(n-1) + n$
$= T(n-2) + n - 1 + n$
$= T(n-3) + n - 2 + n - 1 + n$
$= T(n-4) + n - 3 + n - 2 + n - 1 + n$
.
.
.
$= T(1) + n^2 - (n * (n+1))/2$
$T(n) = \Theta(n^2)$

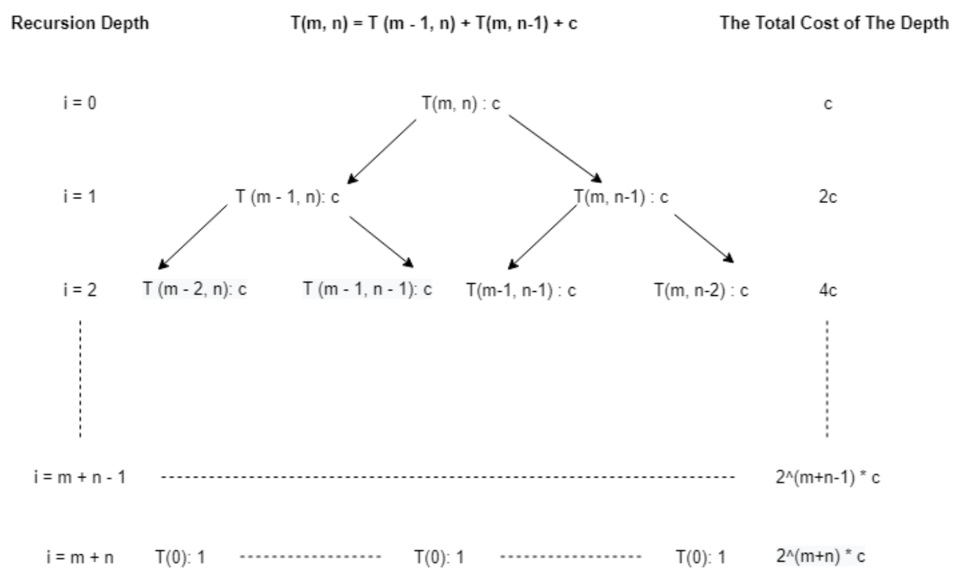
Problem 2

a)

(i)

The worst case happens when there are no common letters in n and m. In this case the algorithm always makes two recursive calls in every step instead of one recursive call. For the worst case we can describe complexity of the algorithm like this:

$$T(n,m) = \left\{ \begin{array}{cc} 1 & \text{if n = 0 or m = 0} \\ T(n-1,m) + T(n,m-1) + O(1) & \text{otherwise} \end{array} \right\}$$



Number of Leaves = 2^i = 2^(m+n)

Sum of the cost of the levels:

$$T(m,n) = c + 2c + 4c + \ldots + 2^{m+n-1} * c + 2^{m+n} * T(0)$$

$$\sum_{k=0}^{i} 2^k = 2^{i+1} - 1 = 2^{m+n+1} - 1$$

$$T(m,n) = \Theta(2^{m+n})$$

Thus, the best asymptotic worst-case running time of the naive recursive algorithm is $\Theta(2^{m+n})$

(ii)

For the memoization algorithm the worst case happens when two strings do not have any common letters. In the worst case, the algorithm will again take make two recursive calls like the naive algorithm. However it will not calculate the steps that are calculated before due to the dynamic programming technique that arises from memoization list. We need to find the maximum number of subproblems that are distinct to calculate the worst case. We can have (n+1)*(m+1) distinct subproblems. Each time the memoized function is called, if the current subproblem is in the memoization matrix, it returns the result in constant time. Also if the i or j is equal to 0 it will return in constant time. In the worst case recursion will happen as many times as the number of distinct subproblems which is (n+1)*(m+1). Furthermore taking the maximum will take O(2) times. And this makes the algorithm polynomial.

Thus, the best asymptotic worst-case running time of the naive recursive algorithm is
$\Theta((m+1)*(n+1)) = \Theta(m*n)$

b)

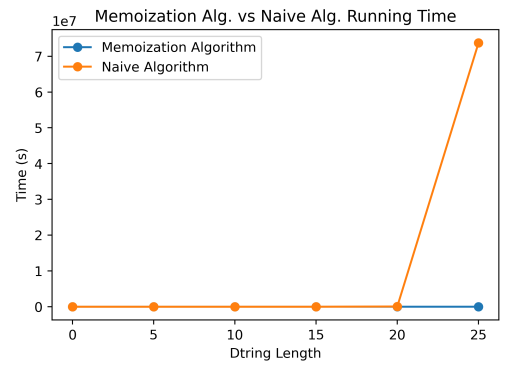
(i)

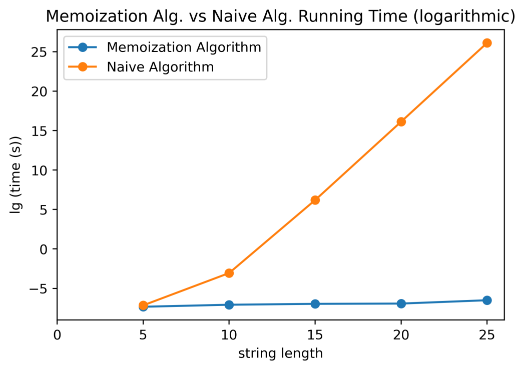
Algorithm	$m = n = 5$	$m = n = 10$	$m = n = 15$	$m = n = 20$	$m = n = 25$
Naive	0.007	0.12	73.17	72012.8	73741107
Memoziation	0.0062	0.0074	0.0080	0.0082	0.010

For the naive algorithm with input size 20 and 25, I used estimated values with using value of n = 18 and and multiplied with 4 for each step, because my computer was unable to run the program with these inputs.
Intel Core i7 - 8750H
16,0 GB RAM
Windows 10

Python 3.8.2 Shell

(ii)





(iii)

Naive Algorithm

The naive algorithm is not scalable because it takes so much time to calculate the results after n=m=15. It takes hours to calculate the result for an input with length 20. The experimental results confirms the theoretical results that I found in the part a. It is visible from the graph that the algorithm grows exponentially.

Memoization Algorithm

The memoization algorithm is scalable because it takes reasonable time to calculate for longer inputs. The experimental results confirms the theoretical results that I found in the part a. It is visible from the graph that the algorithm does not grow exponential but grows polynomial. However, it is not that easy to deduce the degree of the polynomial from the graph which is 2.

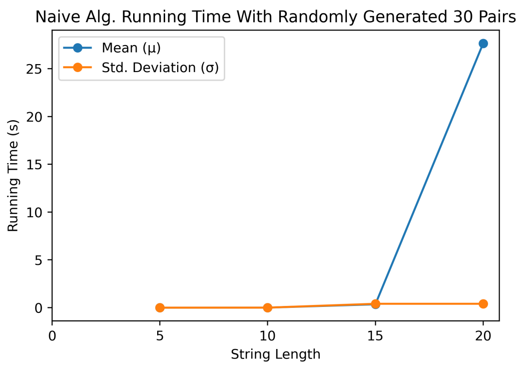
c)

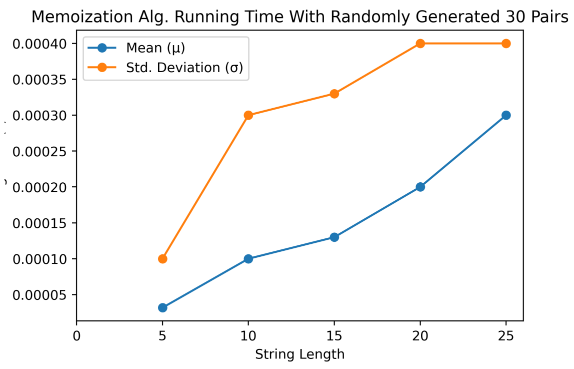
(i)

Algorithm	m =	n = 5		m = n = 10		m = n = 15		m = n = 20		m = n = 25
	µ	σ	µ	σ	µ	σ	µ	σ	µ	σ
Naive	3.48e-05	0.0001	0.0035	0.0031	0.3526	0.4064	27.647	131.402	-	-
Memoization	3.18e-05	0.0001	0.0001	0.0003	0.00013	0.00033	0.0002	0.0004	0.0003	0.0004

The last experimental result for the naive algorithm is not available due to the exponential growth of the algorithm.

(ii)





(iii)

Naive Algorithm

The naive algorithm is still not scalable and the average running times observed in the experiments grow very fast according to the table. From the graph, it looks like it is growing exponentially but without calculations we can not firmly decide it is growing exponential in average case. If we compare the results with the worst case from part a, we can see that we were able to calculate the results for n=m=20. This means that we did not have any DNA sequence pairs that are completely different in the 30 samples.

Memoization Algorithm

The memoization algorithm is still scalable because it takes reasonable time to calculate for longer inputs.From the graph, it looks like it is growing linear but without calculations we can not firmly decide it is growing linear in average case. If we compare the results with the worst case from part a, we can see that we are faster than the worst case.