Assignment 1

Görkem Güzeler 26841

March 2021

1 Problem 1

1.1 Part a:

$\Theta(n^3)$

1.2 Part b:

$\Theta(n^{\log_2 7)}$

1.3 Part c:

$\Theta(n^{\log_4 2 * \log_2 n)}$

1.4 Part d:

$O(n^2)$

2 Problem 2

2.1 Part a:

2.1.1

Naive recursive algorithm: Worst case is $O(2^n)$.

Worst case turns out when the LCS is 0 which means each character of the strings mismatch. Because, recursive algorithm will be solving same sub-problems again and again every time that mismatch occurs since algorithm does not store the computed values.

We use substitution method in order to prove: Guess T(n) = $O(2^n)$
Assume T(n) $\leq c_1(2^k) - c_2(2^{k-1})$ for all k < n
We know our recurrence is: T(n) = 2T(n-1)+ a
T(n) $\leq 2(c_1 2^{n-1} - c_2 2^{n-2})$+ a
T(n) $\leq c_1 2^n - (c_2 2^{n-1}$ - a)
We obtained desired - residual when $c_2 \geq 1$
Our proof is completed.

2.1.2

Recursive algorithm with memoization: Worst case is $O(m * n)$.

There is a matrix mxn (mxn sub-problems) in the given algorithm. Finding the value of each cell (solving the subproblem) takes constant time. Therefore the growth rate will be bounded by $O(m * n)$. The difference with naive algorithm is that we solve each subproblem only once. Then, we check the result of the subproblem whether we found it before, we use it without recomputing. Otherwise we compute and store the result in the matrix so that we do not need to recompute again.

2.2 Part b:

2.2.1

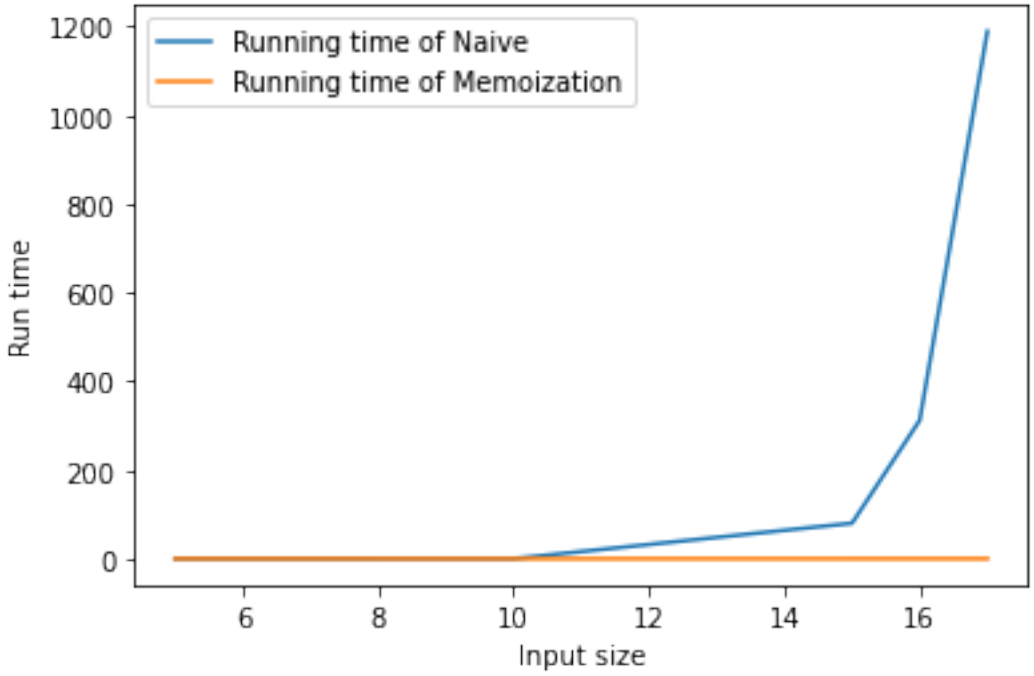
Machine information: Macbook Pro i5 2019,CPU: 1.4 GHz ,RAM: 8 GB, OS: Mac-OS Bigsur 11.2.2

Algorithm	m=n=5	m=n=10	m=n=15	m=n=16	m=n=17
Naive	9.52e-4	0.101	80	312	1189
Memoization	9.68e-4	2.63e-03	1.16e-03	5.22e-3	6.56e-3

2.2.2

Plot:

Examples are taken as LCS = 0 in order to satisfy the worst case.



2.2.3

As it is shown in the graph above, Naive recursive algorithm has an exponential growth whereas the recursive with memoization algorithm has a linear growth rate. If we compare the experimental results with theoretical results found in part a: We see that naive algorithm has a $O(2^n)$ which confirms the exponential growth rate found in the graph. Recursive algorithm with memoization has a $O(m * n)$ growth rate which confirms the linear growth rate found in the graph. In terms of scalability, there is no difference between algorithms for small length (until 10 length) strings. However, when the input length increases, naive algorithm takes much time to execute and becomes inefficient. Thus, for the larger string sequences, it is efficient to use memoization algorithm.

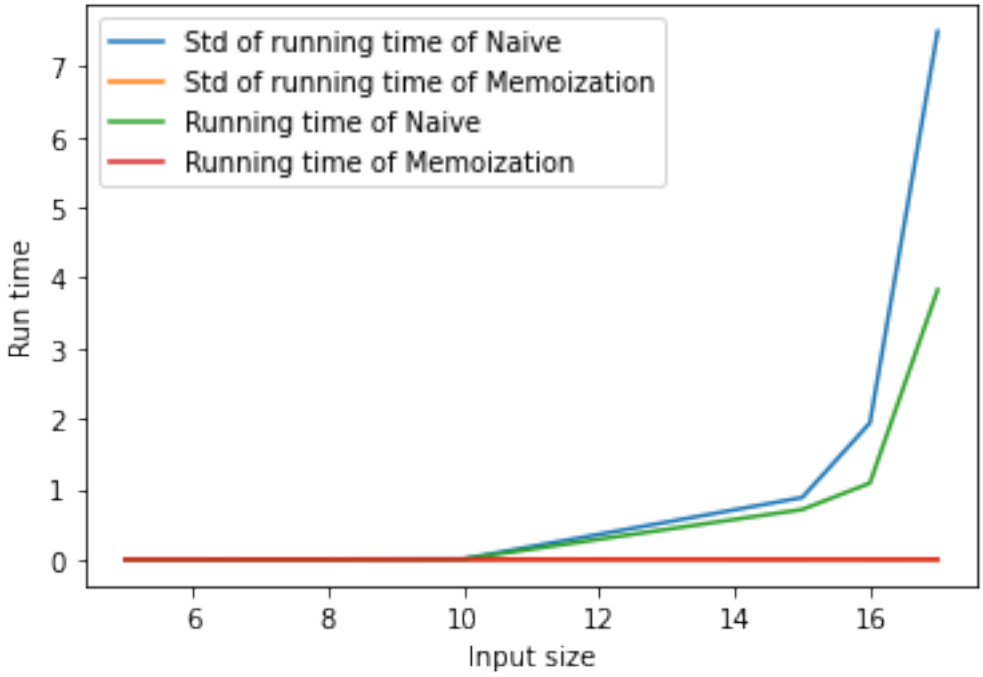
2.3 Part c:

2.3.1

Algorithm	m=n=5 mean - std	m=n=10 mean - std	m=n=15 mean - std	m=n=16 mean - std	m=n=17 mean - std
Naive	0.00054 - 0.0017	0.0058 - 0.0090	0.71 - 0.88	1.085 - 1.94	3.83 - 7.5
Memoization	0.00024 - 0.00047	0.00037 - 0.001	0.0004 - 0.0005	0.0008 - 0.00025	0.001 - 0.0001

2.3.2

Plot:



2.3.3

For the naive algorithm: Clearly, the average run time has decreased drastically when 30 pairs sample were executed for large length strings. The main reason is that part b example has 0 LCS for 17 length strings whereas in 30 pairs sample we have much more common LCS in average. Since the worst case turns out when the LCS is 0, decline in LCS led to increase in execution time. Because same sub-problems were solved again and again every time that mismatch occurs. In both parts growth rate is obviously exponential although values changed.

For the Memoization algorithm: There is no difference at all in terms of running time of algorithms between part b and part c. The reason is that: Although the LCS is decrease, there is no huge consequences since the algorithm growth rate is linear.