CS301 - Assignment 1

Ceren Arkac - 28269

October 2022

Note1: In this homework, I couldn't write properly after drawing the recursion tree in Latex, so I created another Latex file for problem 2 a. And a google document for the rest. Note2: The 2 between the dashes in the recursion tree is the page number.

1 Problem 1

a Use master theorem since it is in the correct format to apply master theorem.
$a = 2 \geq 1, b = 2 \geq 1,\ f(n) = n^3$ (asymptotically positive)
$n^{\log_b a} = n$
$f(n) = n = \Omega(n^{1+\varepsilon})$ for some $\varepsilon > 0$ (Actually any $\varepsilon > 0$). Then case 3 may apply.
Check if $af(n/b) \leq cf(n)$
Check if $2f(n/2) \leq n^3/4\ n^3/4 \leq c(n^3)$, for $1/4 \leq c < 1$
Case 3 applies.

$$T(n) = \Theta(f(n)) = \Theta(n^3)$$

b Correct format to apply master theorem. By master theorem: $a = 7 \geq 1$, $b = 2 \geq 1,\ f(n) = n^2$(asymptotically positive)
$f(n) = n^2 = O(n^{\log_2 7-\varepsilon})$ for some $\varepsilon > 0$. Then case 1 applies. $T(n) = \Theta(f(n)) = \Theta(n^{\log_2 7})$

c I can use master theorem since $a = 21, b = 4 > 1$ and $f(n) = \sqrt{n}$ (asymptotically positive) $f(n) = \sqrt{n} = \Theta(n^{\log_4 2})$ Then the second case applies. $T(n) = \Theta(n^{\log_4 2} \log n) = \Theta(\sqrt{n} \log n)$

d $T(n) = T(n-1) + n$
$T(n-1) = T(n-2) + (n-1)$
$T(n) = [T(n-2) + (n-1)] + n$
$T(n-2) = T(n-3) + (n-2)$
$T(n) = [T(n-3) + (n-2)] + (n-1) + n$
$T(n) = T(n-3) + (n-2) + (n-1) + n$
I can observe the below pattern if I move for k steps further.

$T(n) = T(n-k) + (n-(k-1)) + (n-(k-2)) + ... + (n-1) + n$
Assume $n - k = 0$. Then $n = k$.
$T(n) = T(n-n) + (n-n+1) + (n-n+2) + ... + (n-1) + n$
$T(n) = T(0) + 1 + 2 + 3 + ... + (n-1) + n$
$T(n) = T(0) + (n(n+1)/2)$
$T(n) = \Theta(n^2)$

2 Problem 2 - a - i

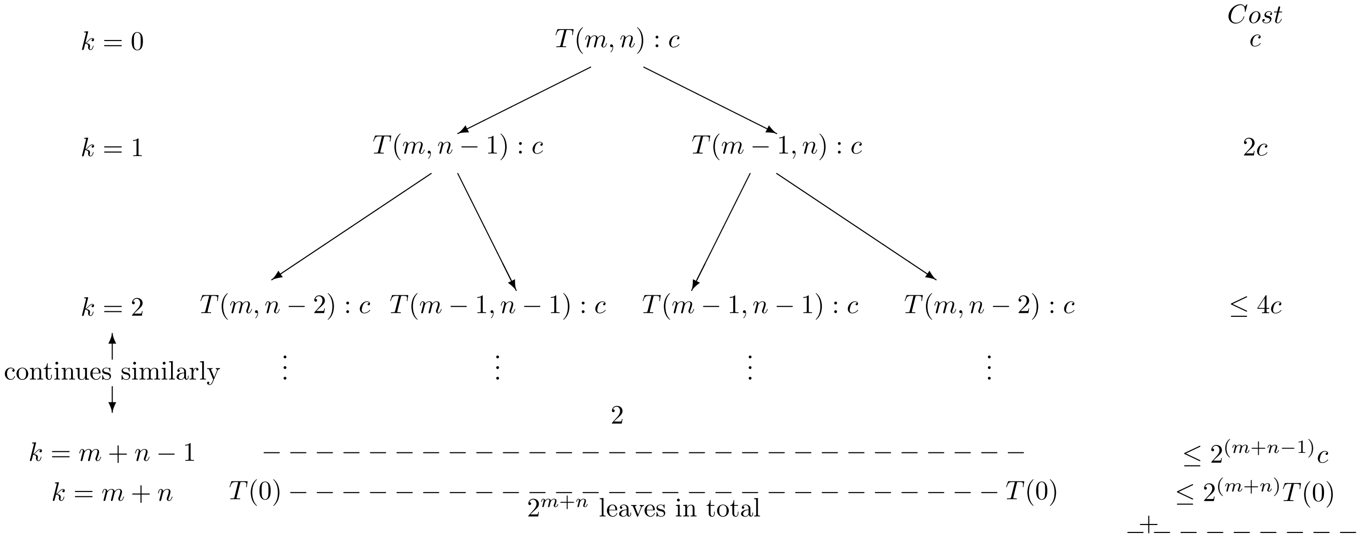
The worst scenario would occur when the input strings don't have any common character. Because, when it is the case, "else:" condition will be executed always which includes two recursive calls in contrast to the "else if:" condition.

Then running time of the algorithm will be determined according to the length of strings.

$T(m,n) = T(m, n-1) + T(m-1, n) + c_1 + c_2 + O(2)$
c_1, c_2 and $O(2)$ can be written as $O(1)$ since the total of c_1, c_2 and $O(2)$ represents constant time complexity.
c_1 =the cost to check the if condition
c_2 =the cost to check the elif condition
$O(2)$ =the cost to find the maximum of 2 values with max() function.

```
def lcs(X,Y,i,j):---------------------------------------->T(m,n)
    if (i == 0 or j == 0): ----------------------------->c1
        return 0
    elif X[i-1] == Y[j-1]:------------------------------>c2
        return 1 + lcs(X,Y,i-1,j-1)
    else:
        return max(lcs(X,Y,i,j-1),lcs(X,Y,i-1,j))------->T(m,n-1) +
             T(m-1,n) + O(2)
```

$T(m,n) = T(m, n-1) + T(m-1, n) + O(1)$
The above can be written as $T(m,n) = T(m, n-1) + T(m-1, n) + c$



By geometric series formula: $T(m,n) = c + 2c + 4c + ... + 2^{m+n-1}c + 2^{m+n}c = (2^{m+n} - 1) = 2^{m+n} - 1$

$\leq 2^{m+n} - 1$

$= O(2^{m+n})$

1 Problem 2 - i continues

With the recursion tree method, I got an upper bound. But, to find the best asymptotic worst-case running time, I should also find a lower bound to ensure that there is a tight bound for this algorithm as the best asymptotic worst-case running time complexity. I will claim that the running time of the algorithm is $\Omega(2^{m+n})$ To show that, I will use substitution method which is based on mathematical induction.
Step 1
Guess: $T(m,n) = \Omega(2^{m+n})$
Step 2
Use induction to prove $T(m,n) = \Omega(2^{m+n})$
$T(m,n) = \Omega(2^{m+n}) \iff \exists$c,n$_0 \geq 0$ such that $\forall n \geq n_0$: $T(m,n) \geq c.2^{m+n}$

1) Induction base step: $T(1,1) \geq c.2^{1+1} \geq c$
There is always some c's to satisfy this inequality.

2) Induction proof
Assume that for all $a, b < n$, $T(a,b) \geq c.2^{a+b}$
Show that $T(m,n) \geq c.2^{m+n}$

$T(m,n) = T(m,n-1) + T(m-1,n) + O(1)$
$T(m,n) \geq c.2^{m+(n-1)} + c.2^{(m-1)+n} + c_1$
$c.2^{m+(n-1)} + c.2^{(m-1)+n} + c_1 = c.2^{m+n} + c_1$
Try to write: $T(m,n) \geq c.2^{m+n}$+(something nonnegative)
I know that c_1 is a nonnegative value since it is actually $O(1)$.
Then I could achieve to write the inequality as I want. $T(m,n) \geq c.2^{m+n}$

Therefore, $T(m,n) = \Omega(2^{m+n})$ for all $n > n_0$ and for some $c, n_0 \geq 0$

Because I found the upper bound and lower bound for $T(m,n)$ with the same function (2^{m+n}), I can say that this function represents also a tight bound for $T(m,n)$.

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

2 Problem 2 - ii

The worst case happens when there is no common characters among the strings since, if it is the case, two recursions will be ran always instead of 1. Because the aim is to compute a subproblem only once, it should be stored in a way that taking it back will take O(1) time. In this algorithm c is created for this purpose and it is a $(m+1).(n+1)$ sized matrix because there will be $(m+1).(n+1)$ number of subproblems (while m and n are the lengths of the input strings). Since taking the results of the subproblems back will takes constant time, and max function takes 0(2), the running time of the algorithm will be $\Theta(m.n)$.

Problem 2 - b]

i)

I ran the algorithms with the worst cases, in which the input strings consist of a type of character and the strings do not include any common characters.
Here are my input strings for different lengths.

```
X_worst_5="AAAAA"
Y_worst_5="TTTTT"
```

```
X_worst_10="AAAAAAAAAA"
Y_worst_10="TTTTTTTTTT"
```

```
X_worst_15="AAAAAAAAAAAAAAA"
Y_worst_15="TTTTTTTTTTTTTTT"
```

```
X_worst_20="AAAAAAAAAAAAAAAAAAAA"
Y_worst_20="TTTTTTTTTTTTTTTTTTTT"
```

Because I couldn't measure the time for m=n=20 and m=n=25, I tried to measure time for 15<m=n<20. I managed to measure m=n=16, but not m=n=17. I stopped running in the m=n=17 situation when time exceeded 21 minutes.

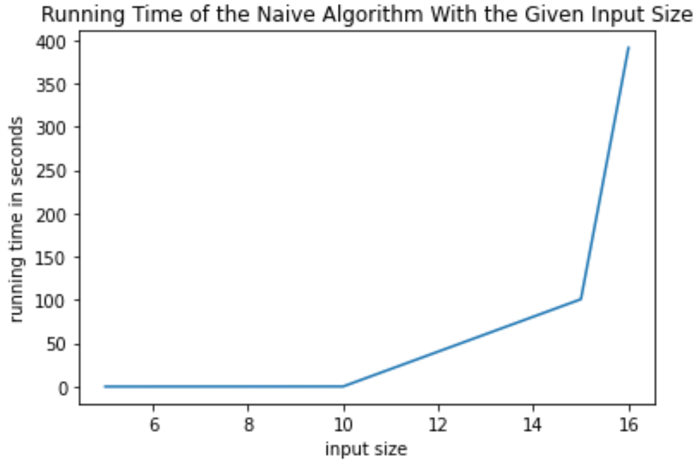
```
properties = "CPU: İşlemci  Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 2904 Mhz, 2 Çekirdek, 4 Mantıksal İşlemci, 8GB RAM"
```

Algorithm	m = n = 5	m = n = 10	m = n = 15	m = n = 16	m = n = 20	m = n = 25
Naive	0.00026607513427734375	0.12606358528137207	100.87317323684692	391.254864692688	Unable to measure	Unable to measure
Memoization	0.00013303756713867188	0.00023651123046875	0.00032734870910064453		0.000606536865234375	0.0009329319000244141

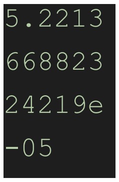
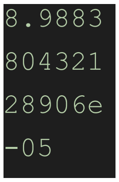
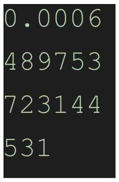
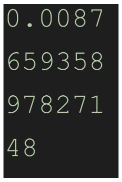
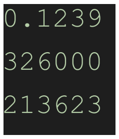
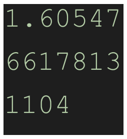
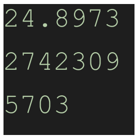
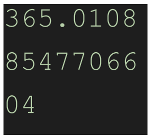
ii)

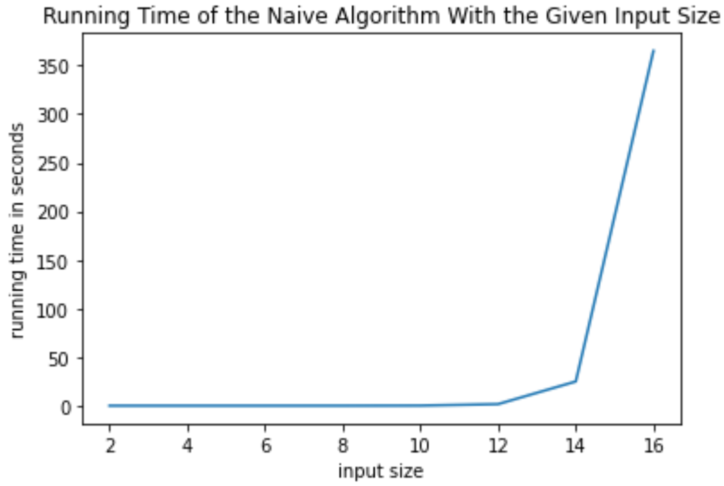
- ***For the naive algorithm,***

Below graph shows the corresponding running times to the input sizes m=n=5,10,15,16 for the naive algorithm.



But, to be more precise, to use equal intervals, I created another graph with the input size m=n=2,4,6,8,10,12,14,16. I will be taking into account this graph for the rest of this question.

Algortihm	m=n=2	m=n=4	m=n=6	m=n=8	m=n=10	m=n=12	m=n=14	m=n=16
Naive	 5.221366882324219e-05	 8.988380432128906e-05	 0.0006489753723144531	 0.008765935897827148	 0.1239326000213623	 1.6054766178131104	 24.897327423095703	 365.01088547706604



iii)

When it comes to the scalability of the algorithm,

I got an exponentially increasing graph in the size of the input as I expected (T(m,n) = 2^(m+n) is an exponential function.)

This algorithm is poorly scalable since the amount of computations increase exponentially with respect to input size (according to the theoretical results: sum of lengths of the input strings).

To compare the experimental results and the theoretical results, I claimed that

When m=n=12, the experimental result for the running time is:

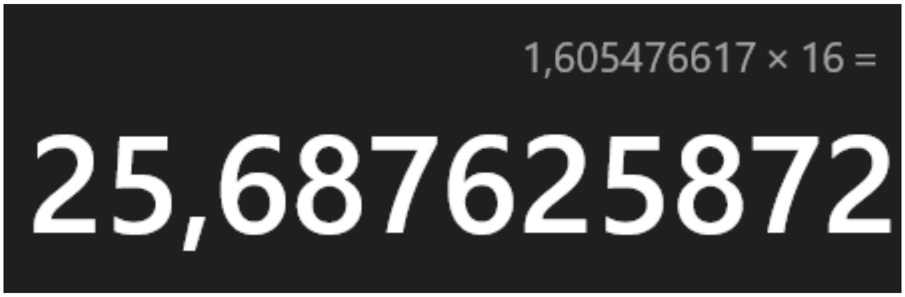
```
1.6054766178131104
```

While m=n=14, the experimental result for the running time is:

```
24.897327423095703
```

If the theoretical result (O(2^(m+n) is a good asymptotic bound for this algorithm, then the experimental result for the running time when m=n=14 should be about 16 times the experimental result for the running time when m=n=12.

Indeed, it is a very close approximation as it has been shown below.



1,605476617 × 16 =

25,687625872

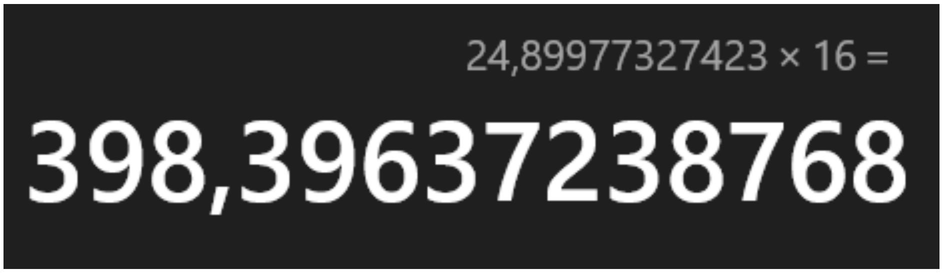
A similar observation can be made with m=n=14 and m=n=16.

When m=n=14, the experimental result for the running time is:

```
24.897327423095703
```

When m=n=14, the experimental result for the running time is:

```
365.01088547706604
```



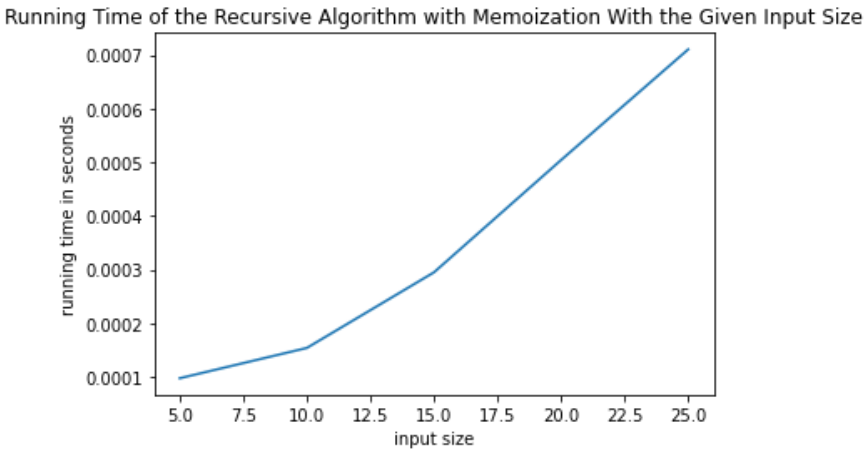
24,89977327423 × 16 =

398,39637238768

ii)

- ***For the recursive algorithm with memoization,***

This graph shows the corresponding running times to the input sizes m=n=5,10,15,20,25 for the recursive algorithm with memoization.



iii)

When it comes to the scalability of the algorithm,

I got an polynomially increasing graph in the size of the input as I expected (T(m,n) = m x n is a polynomial function. It is not a linear function because m and n are not constants. Here, the found time complexity is a function of m and n which are the lengths of the input strings.) Because I computed for m=n, in this case, time complexity is like n^2 (or m^2). In this case T(m,n) is a polynomial function with degree 2 and this means it is a quadratic function.

This algorithm is highly scalable since the amount of computations shows polynomial ratios within the equal intervals with respect to the input size (according to the theoretical results: multiplications of lengths of the input strings).

To compare the experimental results and the theoretical results, I claimed that

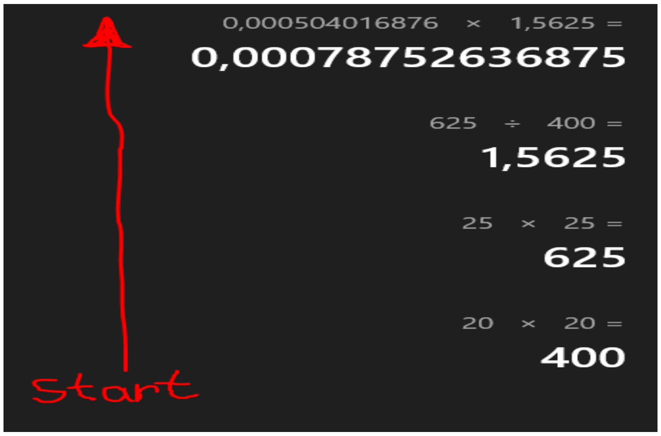
When m=n=20, the experimental result for the running time is:

```
0.0005040168762207031
```

While m=n=25, the experimental result for the running time is:

```
0.0007102489471435547
```

If the theoretical result O(m+n) is a good asymptotic bound for this algorithm, then the experimental result for the running time when m=n=25 should be about 1,5625 times the experimental result for the running time when m=n=20. Indeed, it is a very close approximation as it has been shown below.



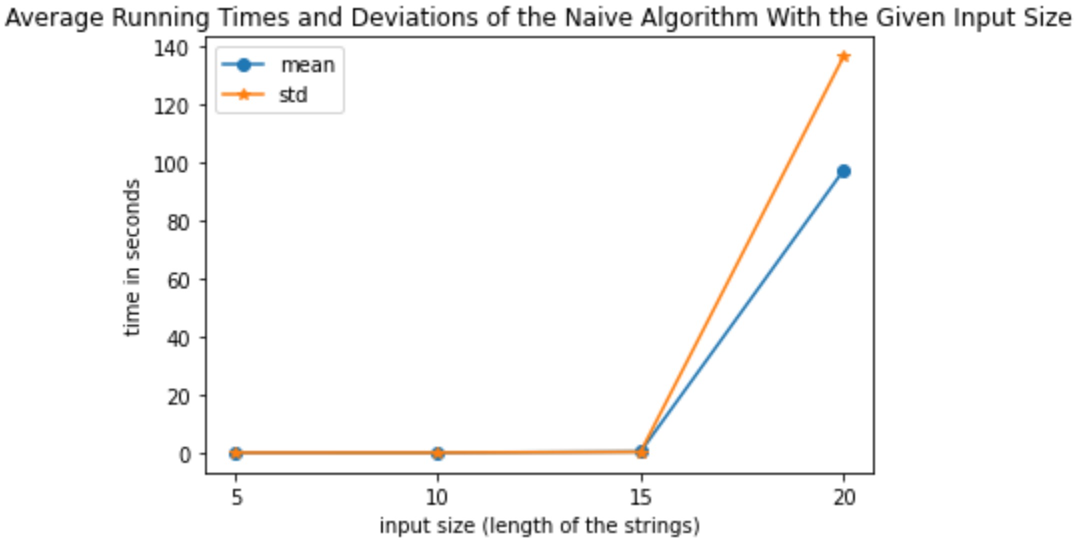
```
0,000504016876  ×  1,5625 =
0,00078752636875

625  ÷  400 =
1,5625

25  ×  25 =
625

20  ×  20 =
400
```

Problem 2 - c)

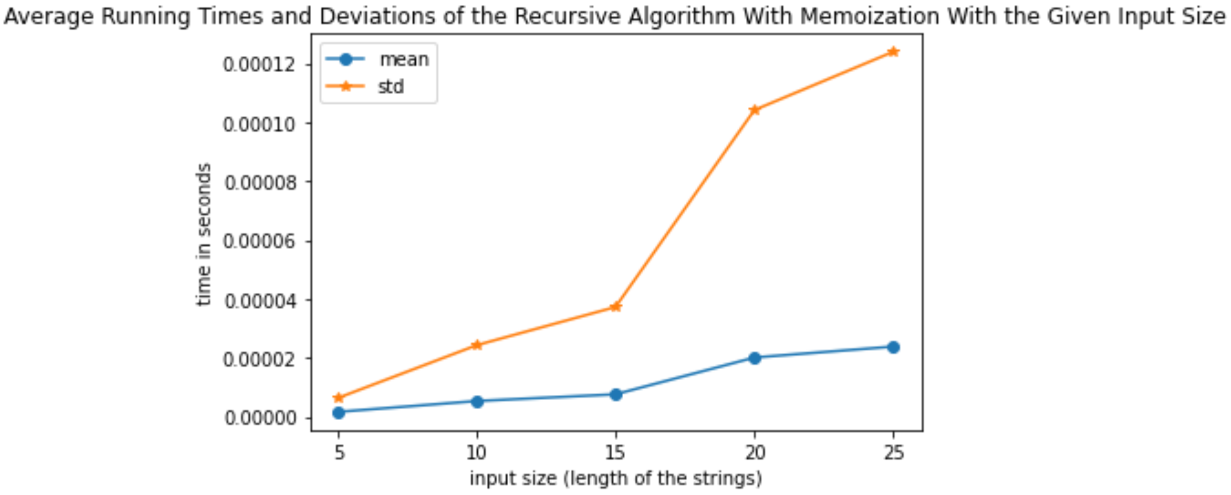
i)

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	0.0001676638921101888	0.000350407442905240 8	0.005364489555358886 5	0.004770979833190523	0.344398927688598 64	0.26484650130098114	97.13865716457367	136.27752328780218	Unable to measure	Unable to measure
Memoization	1.6450881958007813e-06	6.3374361181971 38e-06	5.3326288859049 48e-06	2.4290277305748794e-05	7.613499959309896e-06	3.728137496519869e-05	2.011458079020182e-05	0.000104158807474973 96	2.3825963338216147e-05	0.0001239680771505982 6

ii)



Average Running Times and Deviations of the Naive Algorithm With the Given Input Size



Average Running Times and Deviations of the Recursive Algorithm With Memoization With the Given Input Size

iii)

- For the naive algorithm:
  Again the running time grows rapidly but it differs from the worst case scenario because I can run m=n=20 case. Moreover, I couldn't achieve even one time of m=n=20 case while I ran here for 30 times at the end of 49 minutes. This is the case because having no common character is not a common thing among the randomly generated 30 pairs of DNA sequences. Here, we don't call two recursive calls every time. However, again, the graph showing the average case looks like an exponentially growing function. The standard deviations are close to each other because in each pair, the same processes apply. There is no utilized way of solving the subproblems.

- For the recursive algorithm with memoization:
  In the graph above, the means for different lengths are pretty close to each other and look like growing slower than the worst case scenario. This is again, there is never or almost no worst case scenario among the input strings. The reason for the standard deviations are higher in this algorithm compared to the naive algorithm is that the number of already computed subproblems are diverse among the different pairs.