Problem 1 (Recurrences) **YASIN AYDIN**

(a)

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

(b)

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

(c)

$$T(n) = \Theta(n^{\frac{1}{2}} \log n)$$

(d)

$$T(n) = \Theta(n^2)$$

Problem 2 (Longest Common Subsequence - Python) **YASIN AYDIN**

(a) i) The best asymptotic worst-case running time of naive recursive algorithm is:

$$O(2^n) \quad \text{where m = n.}$$

Reason: The worst-case it that at each recurrence first two statements fail because of the inequalities and the function end up with the two "lcs" recursive functions. So, we can say that for each worst-case scenario function has to calculate two more recursive calls which increases the number of calculations to be made by two times.

When $i = 1$ the number of recurrent calls is 2.

When $i = 2$ the number of recurrent calls is 4.

When $i = 3$ the number of recurrent calls is 8.

So, we can see the 2^i pattern here. This can be converted into:

$$O(2^n) \text{ where i = n}$$

In that case, we have n = m because the worst-case scenario is when we have a complete binary tree. The reason for this bad run time is that the recursive function solves same problems repeatedly, which increases the asymptotic bound.

ii) The best asymptotic worst-case running time of naïve recursive algorithm is:

$$O(n * m). \quad \text{where n and m can be anything.}$$

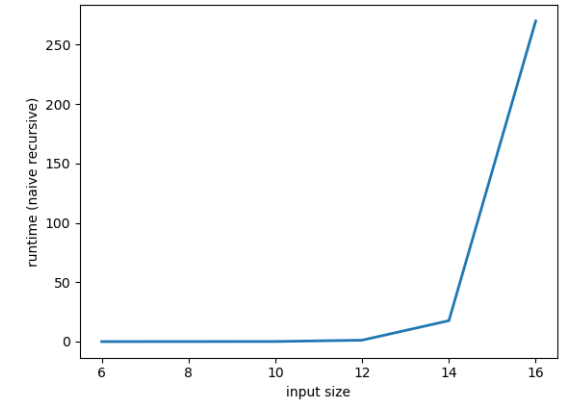
Reason: With the memoization method we basically use the previous computations in a matrix and use them for future computations so that the run-time is much better. Even in the worst-case were we do not have any common subsequences the algorithm just fills the matrix with 0's and return the max value that's been found at that time (or "location" if we consider the matrix). So, the complexity is related to the amount of time spent to fill the matrix, which is the multiplication of both lengths of strings. This gives us the $O(n * m)$ complexity which is much better with the longer strings compared to naïve approach.

(b) i) *Table 1*, Running time table in seconds.

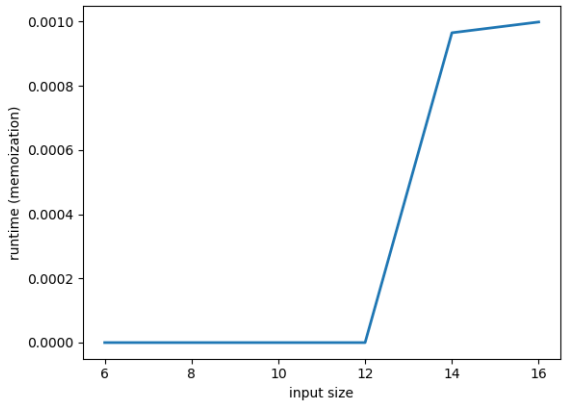
Algorithms	n=m=6	n=m=10	n=m=12	n=m=14	n=m=16
Naive	0.0	8.17e-02	1.18e-00	1.76e+01	2.69e+02
Memoization	0.0	0.0	0.0	9.65e-04	0.0

SPECS: Intel(R) Core(TM) i7-8750H CPU @ Base speed: 2.20GHz (Average 3.90 GHz), 16.00GB RAM, Windows OS

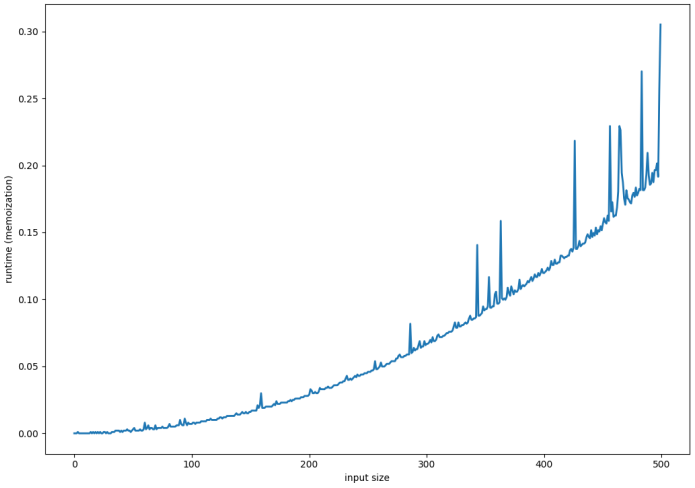
ii) *Graph 1* for Naive recursion:



Graph 2 for Recursion with Memoization:

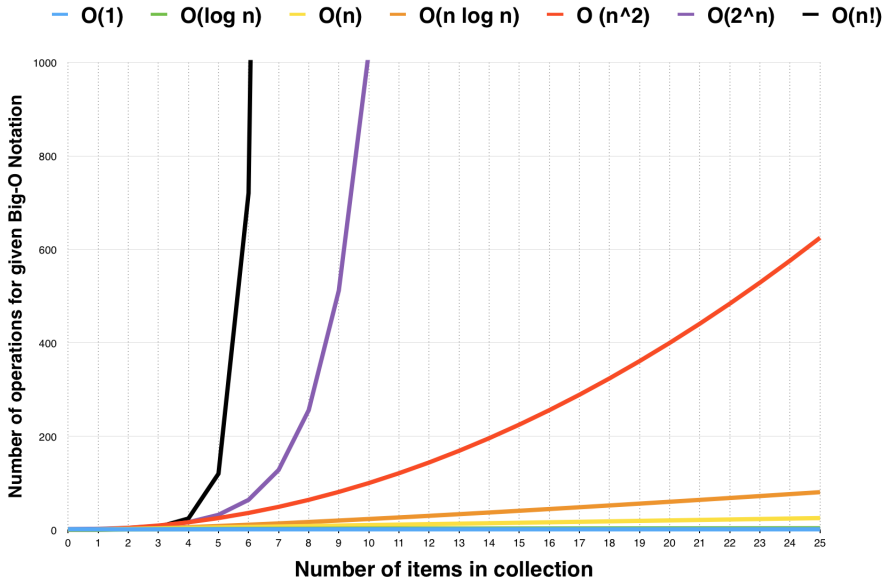


Graph 3 for extend Memoization: (To see the gradual increase with more input lengths)



iii) When we consider the naïve recursive approach, we can conclude that it is not scalable algorithm. Even with pairs of string length of 20, algorithm took considerable amount of time and in the biology, we have very long DNA sequences, so this is not a logical and scalable algorithm for this problem. We also have a recursive function with memoization technique where the graphs with run-time values and the complexity analysis shows that it is more scalable algorithm than the naive approach. We can tell that by looking at the complexities $O(2^n)$ and $O(n * m)$. But, again the DNA sequences in real life can me so long (million-billions of nucleotide pairs) that this algorithm will not be sufficient (For instance the using 2 strings with 100k lengths would require 2 Terabytes of space for integer matrix). In the case of n=m and no common substring, (worst case of this algorithm) we have $O(n^2)$ complexity which indicates that it is not scalable algorithm, but in the cases where one of the string length is very small we can consider this algorithm in scalable category since the growth rate will get closer to the linear line. Also, the memoization helps the algorithm to be sufficiently good for much longer string since the repetitiveness is gone. Another thing to point out, at some point Python will give an error about the recursion limit (RecursionError: maximum recursion depth exceeded in comparison). In my case, it was about string length of 570 when I got the error (n=m). So, overall I would say both algorithms are not scalable.
As the lines $O(2^n)$ and $O(n^2)$ from the *graph 4* aligns with the generated graphs and by looking at the run-times we can conclude that the experimental results confirm the theoretical results[1].

Graph 4 Big-O complexity lines:

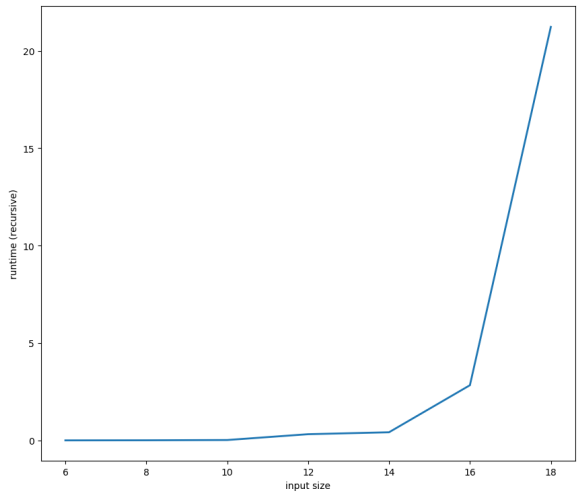


[1]Graph from https://medium.com/free-code-camp/my-first-foray-into-technology-c5b6e83fe8f1

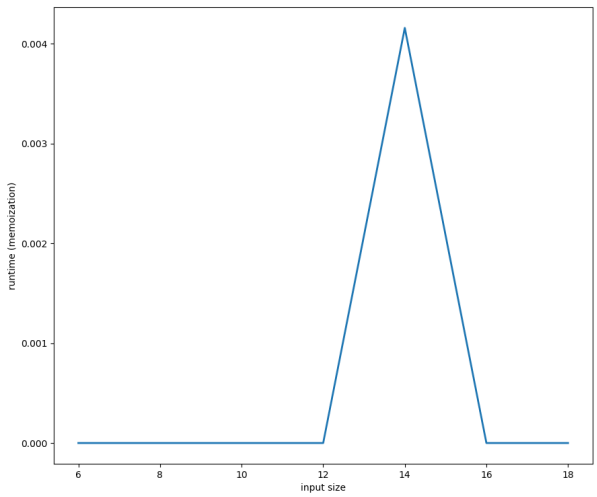
(c) i) *Table 2*, Running time table in seconds.

Algorithms	n=m=6	n=m=8	n=m=10	n=m=12	n=m=14	n=m=16	n=m=18
Naive μ	0.0	4.27e-03	1.70e-02	3.16e-01	4.14e-01	2.82e-00	2.12e+01
Naive σ	0.0	7.32e-03	2.93e-02	5.42e-01	7.10e-01	4.85e-00	3.63e+01
Memoization μ	0.0	0.0	0.0	4.15e-03	0.0	0.0	0.0
Memoization σ	0.0	0.0	0.0	4.15e-03	0.0	0.0	0.0

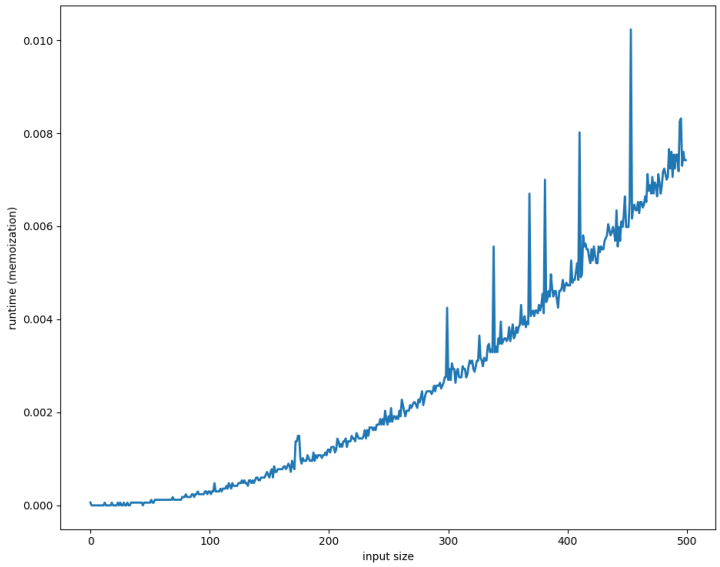
ii) *Graph 5*, Naive recursion with mean values:



Graph 6, Recursion with memoization (with mean values):



Graph 7, Recursion with memoization (mean values from n=m=0 to n=m=500):



iii) The growth line of both worst-case and experimental cases look similar but as expected the worst-case results are much higher (generally 2 t0 15 times) than the average results. As results of this, we can conclude that in the worst-case scenario the acceleration of the line is more than the average case.We can compare few values to see the difference:
In the $n = m = 12$ case, the worst-case result was $1.18e + 00$ but the average results was $3.16e - 01$. In the $n = m = 14$ case, the worst-case result was $1.76e + 01$ but the average results was $4.14e - 01$.

$$\text{Worst-case growth rate was } \frac{n = m = 14 => 1.76e + 01}{n = m = 12 => 1.18e + 00} = 14.91$$

$$\text{Average-case growth rate was } \frac{n = m = 14 => 4.14e - 01}{n = m = 12 => 3.16e - 01} = 1.31$$

In the $n = m = 14$ case, the worst-case result was $1.76e + 01$ but the average results was $4.14e - 01$. In the $n = m = 16$ case, the worst-case result was $2.69e + 02$ but the average results was $2.82e - 00$.

$$\text{Worst-case growth rate was } \frac{n = m = 16 => 2.69e + 02}{n = m = 14 => 1.76e + 01} = 15.28$$

$$\text{Average-case growth rate was } \frac{n = m = 16 => 2.82e - 00}{n = m = 14 => 4.14e - 01} = 6.81$$

As seen from the calculations, in the worst-case scenarios the growth rate (slope) is much higher than the average-case scenario. To conclude, with the values and the generated graphs we can say that the average-case values grow much slower than the worst-case values.