SUMMARY

USC ID/s:

4156882528,
1231409457,
5048266342.

Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	1.035451889038086	1.039743423461914	13108	13156
64	2.9990673065185547	9.598255157470703	13152	13204
128	9.58704948425293	25.055885314941406	13196	13188
256	37.83845901489258	97.12815284729004	13380	13248
384	43.72048377990723	142.39811897277832	13708	13196
512	184.72790718078613	305.31978607177734	13700	13192
768	216.92156791687012	718.9590930938721	13820	13256
1024	764.0266418457031	1259.092092514038	14052	13244
1280	851.6218662261963	1749.077320098877	14116	13380
1536	1303.1256198883057	1902.9974937438965	14448	13476
2048	2401.360034942627	4092.467784881592	14640	13500
2560	2610.7864379882812	5871.683597564697	14584	13544
3072	4089.21480178833	9173.839092254639	14732	13656
3584	6426.533460617065	11464.92600440979	14720	13720
3968	6224.087953567505	14884.938716888428	14532	13808

Insights

- Efficient algorithm takes time twice as much as the basic algorithm.
- Efficient algorithm takes n times lesser memory than basic algorithm.
- Therefore, it is a trade-off between time & memory.
- Basic algorithm uses a lot of memory compared to the efficient algorithm’s time consumption.
- In reality, memory is very expensive while computing time can be reduced using efficient machines.
- In sequence alignment, the amount of memory required is in billions. Thus, basic algorithm requires a huge amount of memory.

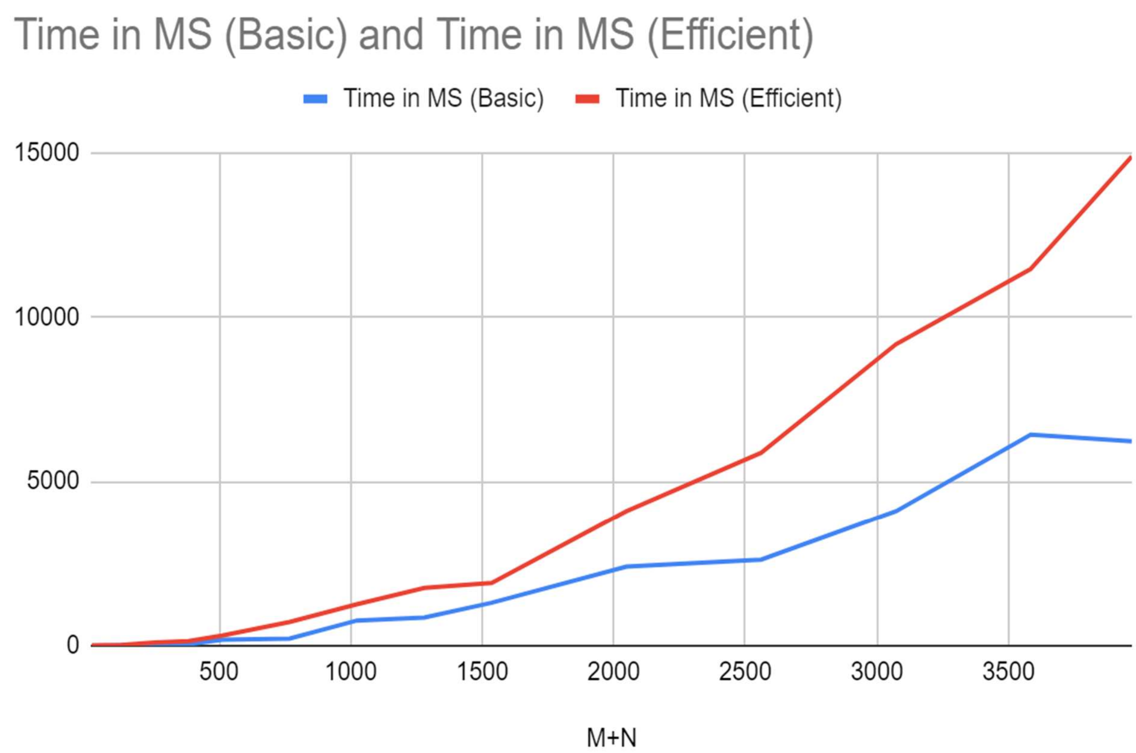
Graph1 – Memory vs Problem Size (M+N)

Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)

Basic: The memory increases in polynomial time.
Efficient: The memory is almost constant up to a point and then there is a slight increase and overall runs in polynomial time.

Explanation:
In the Basic algorithm, m represents the size of word X and n represents the size of word Y. The memory required to compute in this algorithm grows quadratically with respect to the problem size, which is O(nm). We use divide and conquer in the Efficient algorithm to make the solution memory efficient. We divide the string Y in two equal halves. We compute the optimal point for string X by first running the DP recurrence method on X and Y's first half. Then we reverse X and Y's last half and use the DP's recurrence method to find the point for X. Thus, memory usage is greatly reduced in this manner.

Graph2 – Time vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)

Basic: As problem size increases, there's a polynomial increase in the runtime which is proportional to O(m*n).
Efficient: As the problem size increases, there's a polynomial increase in the runtime which is proportional to O(2*m*n).

Explanation:

In the basic algorithm, we create a 2D table for the Dynamic approach with m rows and n columns using Dynamic programming. First, both X and Y strings are computed from the input, and then we simply execute the DP recurrence formula recursively on the two strings. In the recurrence formula, we take the minimum of the gap in front of X and Y-1, the gap in front of Y and X-1 and Y-1, and X-1 while taking alpha values into account. After that, perform a top-down pass to find the optimal solution. If there is a match with the diagonal element, we have paired the two values; if we move diagonally, there is a gap in front of both strings. There is a gap in front of Y-1 if we move vertically; otherwise, we move horizontally. As a result, there is a space in front of X-1. To get the actual alignment, reverse both alignment values because we came from the top down.

Time complexity: O(m*n)
In the Efficient approach, we divide the string Y into two halves using Divide and Conquer methods and perform more comparisons in a 2D table for instance of Y to instance of X. As a result, it takes longer than the basic approach but is still limited to the size of X and Y time complexity: O(2*m*n).

Contribution:

4156882528: Equal Contribution
1231409457: Equal Contribution
5048266342: Equal Contribution