SUMMARY

USC ID/s:

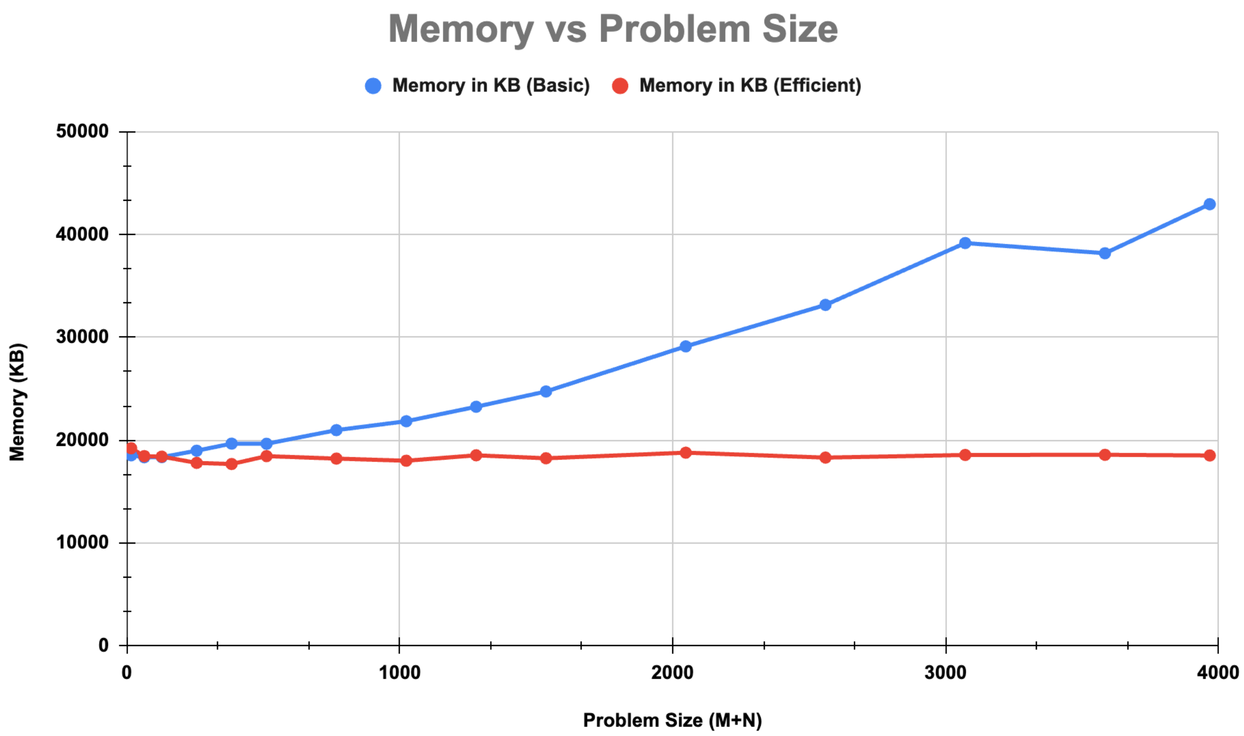
1248363133 – Anjali Mehta
3908543660 – Siqi Dong
7184608860 – Dean Woods
2919603029 – Varun Kapoor
8142329307 – Tianming Ji

Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.06689453125	0.07605552673	18528	19216
64	0.2451171875	0.538110733	18336	18448
128	0.8930664063	1.437902451	18352	18400
256	3.135009766	6.530761719	18976	17792
384	8.321044922	11.82174683	19664	17680
512	14.91210938	25.30264854	19648	18448
768	36.37011719	46.23293877	20976	18208
1024	52.17822266	86.52091026	21840	18000
1280	85.66113281	135.3108883	23248	18528
1536	112.0031738	183.2647324	24736	18240
2048	215.2919922	350.6522179	29120	18784
2560	340.5209961	530.8549404	33152	18304
3072	493.7492676	783.6890221	39168	18560
3584	673.5979004	1067.532063	38176	18576
3968	828.5388184	1338.90605	42944	18512

Insights

Graph1 – Memory vs Problem Size (M+N)



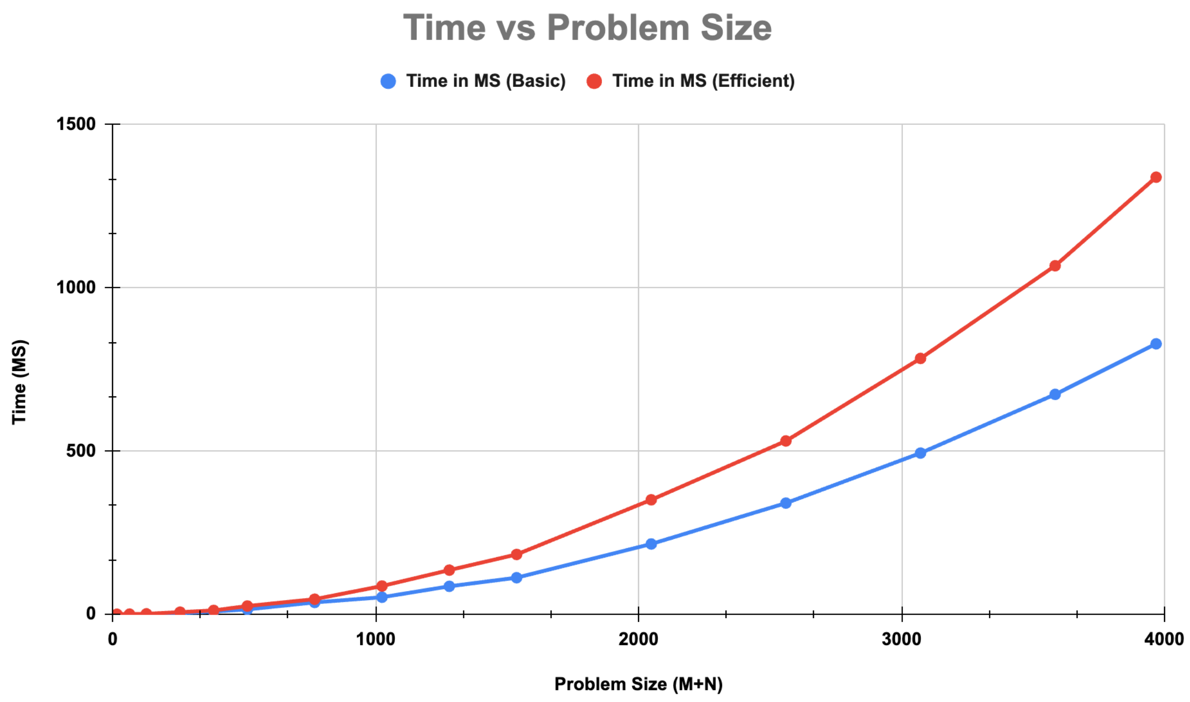
Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)
Basic: Polynomial
Efficient: Linear

Explanation:
The memory usage of the basic algorithm increases polynomially because it constructs a full dynamic programming table of size m*n. As the expanded input sequences grow, the DP matrix grows quadratically, which leads to a steady and increasingly steep rise in memory consumption. In the graph, this appears as the upward-curving blue line, which starts around 18,000 KB and grows to roughly 43,000 KB for the largest problem sizes.

In contrast, the memory-efficient algorithm drastically reduces memory requirements by storing only a small portion of the DP table, typically one or two rows at a time, while recomputing subproblems as needed. This reduces the space complexity to O(m + n). As a result, its memory usage remains almost constant, fluctuating only slightly around 18,000 KB regardless of input size. This behavior is reflected in the flat red line on the graph.

Overall, the graph illustrates the clear scalability advantage of the efficient algorithm, while the basic version's memory usage grows rapidly with problem size, the efficient version maintains stable and predictable memory requirements, making it far more suitable for large-scale problems.

Graph2 – Time vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)
Basic: Polynomial
Efficient: Polynomial

Explanation:
Both the basic and efficient algorithms exhibit polynomial growth in runtime as the problem size increases, which is expected because dynamic programming for sequence alignment requires evaluating an m * n grid of subproblems, giving it a time complexity of O(mn). The basic algorithm computes the dynamic programming table once, filling all entries in a direct and systematic manner, so its runtime increases steadily with input size but stays lower than that of the efficient algorithm.

In contrast, the efficient algorithm, while significantly better in memory usage, incurs additional time overhead due to its divide-and-conquer strategy, which requires repeatedly recomputing portions of the DP table and executing multiple recursive calls. This overhead becomes increasingly impactful for larger inputs, causing the red curve to rise more sharply and eventually surpass the blue curve.

Overall, the graph illustrates a clear trade-off between time and space: the basic algorithm runs faster but consumes substantially more memory, whereas the efficient algorithm dramatically reduces memory usage at the cost of increased runtime.

Contribution
Equal Contribution