SUMMARY

USC ID/s:

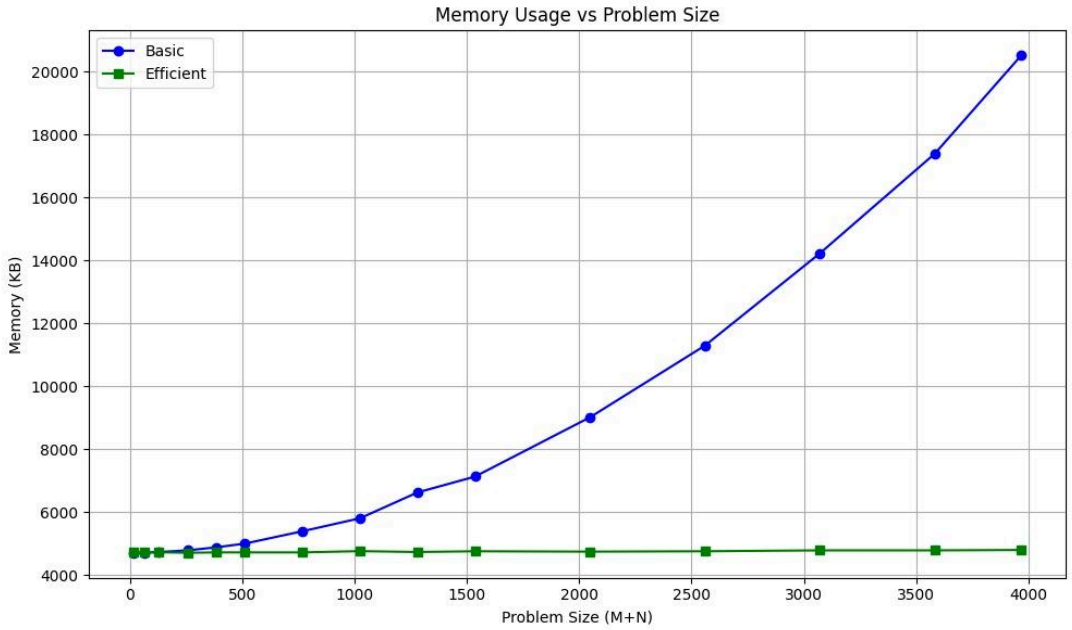
3703126779, 7585775949, 4394172627

Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.0298	0.0917	4712	4728
64	0.0972	0.3392	4716	4724
128	0.3564	0.6733	4740	4732
256	1.0028	2.7269	4792	4716
384	2.056	5.1612	4888	4728
512	2.8937	12.421	5008	4728
768	7.7227	21.755	5400	4728
1024	5.0536	15.6226	5812	4768
1280	12.4675	21.9471	6632	4740
1536	14.8021	33.6758	7136	4764
2048	20.5473	56.4763	9020	4752
2560	40.0483	83.7788	11300	4764
3072	47.6296	126.627	14233	4792
3584	66.0652	157.36	17404	4792
3968	77.5113	192.217	20520	4804

Insights

Graph1 – Memory vs Problem Size (M+N)



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

Basic: Polynomial

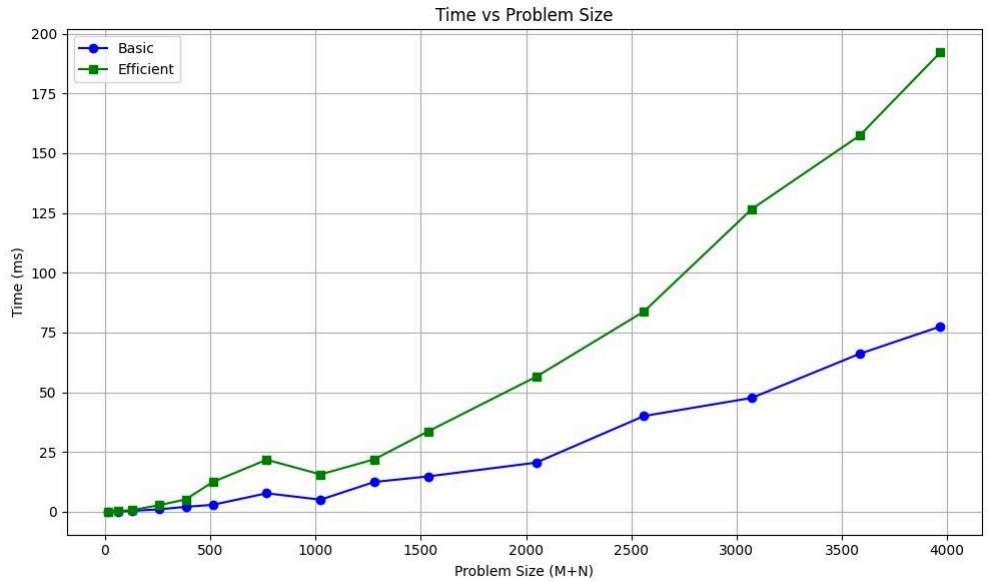
Efficient: Linear

Explanation:

The memory usage of the Basic implementation grows quadratically (polynomially) with the problem size (m + n), because it uses a 2D dynamic programming table of size (m + 1) x (n + 1)—which leads to O(mn) space complexity. As the string lengths double, the memory requirement increases steeply, and this is clearly visible in the upward curve of the basic plot.

On the other hand, the Efficient implementation (Hirschberg's algorithm) maintains a nearly flat memory footprint, because it only stores one row or one column of the DP table at a time, giving it O(min(m, n)) space usage. This keeps memory usage almost constant, even when input sizes grow large.

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial

Efficient: Polynomial (but steeper due to recursion overhead and extra calls)

Explanation:

The Basic implementation follows a classical dynamic programming approach with time complexity O(m × n). This results in a polynomial growth of execution time as the input strings grow. The time increases smoothly and predictably with increasing problem size.

The Efficient implementation also has O(m × n) time complexity in theory, but in practice it incurs additional recursive overhead due to the divide-and-conquer strategy and recomputation of cost vectors. This causes its runtime to be consistently higher than the basic version in your experiment, especially at larger input sizes.

Although the memory-efficient version uses less memory, it comes at the cost of greater runtime due to repeated calculations and lack of memoization in its recursive design. That's why the green line grows faster and surpasses the basic one.

Contribution

3703126779: Equal Contribution

7585775949: Equal Contribution

4394172627: Equal Contribution