SUMMARY

USC ID/s:

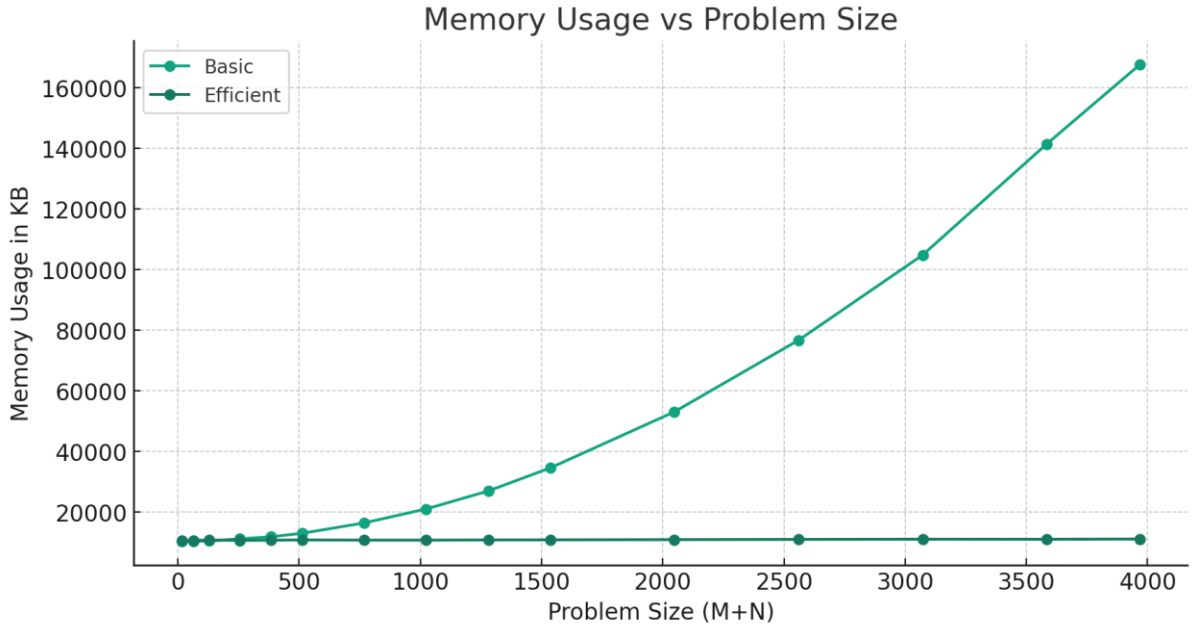
Xiangjie Yuan: 4040019986
Haoyue Xu: 8869966520
Chenchao Lin: 8495662924

Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.088	0.166	10360	10648
64	0.864	1.523	10332	10688
128	3.919	5.684	10484	10776
256	14.479	21.620	11184	10664
384	31.005	48.810	11852	10728
512	54.213	86.191	13008	10812
768	126.101	285.441	16424	10760
1024	222.848	355.337	21032	10744
1280	358.636	580.146	26964	10820
1536	553.691	827.774	34584	10840
2048	1022.417	1554.703	53092	10920
2560	1408.274	2407.146	76716	10996
3072	2192.303	3733.849	104820	11060
3584	2837.150	4529.528	141448	11036
3968	3982.372	5501.081	167628	11116

Insights

Graph1 – Memory vs Problem Size (M+N)



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

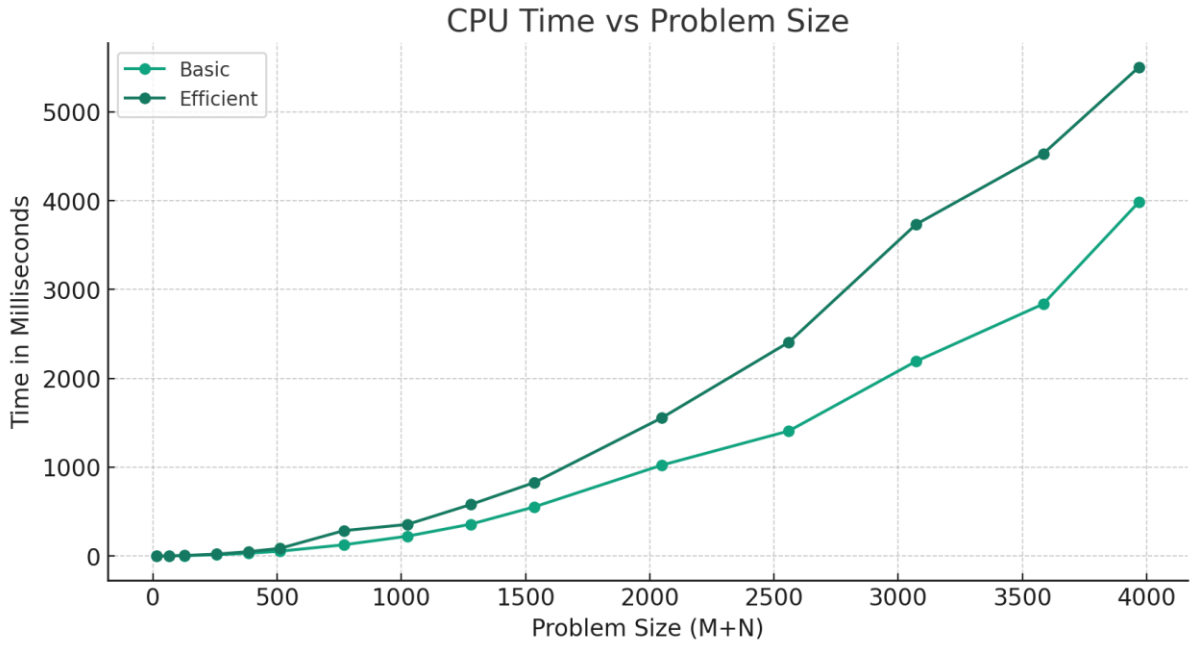
Explanation:
The Basic implementation shows a significant increase in memory usage as the problem size grows, indicating a higher space complexity. This is due to the use of a 2D memorization table in the Basic implementation, which stores the intermediate results for all subproblems.

The memory usage of the Basic implementation appears to grow quadratically with the problem size, suggesting an O(M*N) space complexity. The size of the memorization table is directly proportional to the product of the lengths of the input strings.

In contrast, the Efficient implementation maintains a linear memory usage across all problem sizes, indicating a more efficient memory utilization. The divide-and-conquer approach used in the Efficient implementation allows for a more space-efficient solution by recursively solving smaller subproblems and discarding intermediate results that are no longer needed.

The memory usage of the Efficient implementation is consistently lower than the Basic implementation, suggesting a better space complexity. The Efficient implementation achieves a space complexity of O(min(M, N)) by storing only the necessary information for the current subproblem and the recursive calls.

Graph2 – Time vs Problem Size (M+N)



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

Explanation:

Both basic and efficient solution suggests a polynomial trend in CPU time as the problem size increases. This aligns with the expected behaviour of a dynamic programming solution to the sequence alignment problem, where the time complexity is generally O(M*N) for two sequences of lengths M and N. The growth of CPU time is due to the fact that each cell in the dynamic programming matrix represents a subproblem whose solution depends on the solutions to other subproblems.

The Efficient implementation consistently takes more CPU time compared to the Basic implementation for all problem sizes. This is likely due to the divide-and-conquer approach used in the Efficient implementation. The recursive calls and the merging of subproblems contribute to the increased CPU time.

Contribution

4040019986: Equal Contribution
8869966520: Equal Contribution
8495662924: Equal Contribution