SUMMARY

USC ID/s: 6281993930

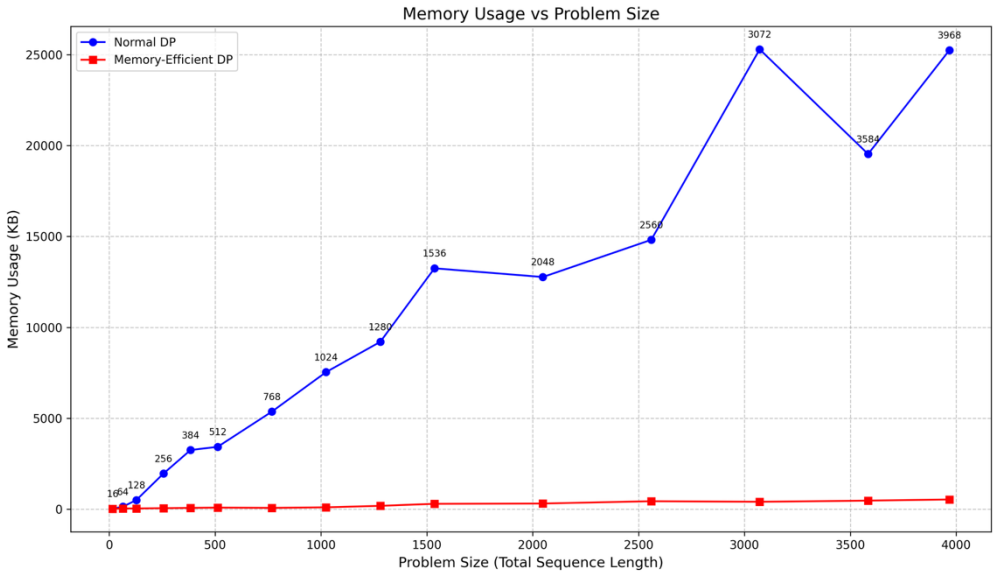
Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memroy in KB (Basic)	Memory in KB (Efficient)
16	0.02	0.06	16	16
64	0.16	0.36	128	32
128	0.65	1.24	496	32
256	3.15	4.48	1952	48
384	6.47	10.65	3248	64
512	10.99	18.4	3424	80
768	26.59	43.39	5360	64
1024	52.12	80.43	7536	96
1280	83.6	127.47	9200	176
1536	118.25	174.36	13248	288
2048	221.68	344.16	12768	304
2560	336.96	534.12	14816	432
3072	477.52	714.05	25296	400
3584	677.3	1074.85	19536	464
3968	839.51	1349.32	25248	528

Insight:

As problem size gets bigger, the time grows at a fairly even rate with the efficient algorithm being slower. This is due to some computation overhead when backtracking, but both still run in a O(m*n) rate. Memory complexity on the other hand grows much faster in the basic algorithm and memory in the efficient algorithm doesn't grow much with bigger inputs.

Graph1 – Memory vs Problem Size (M+N)



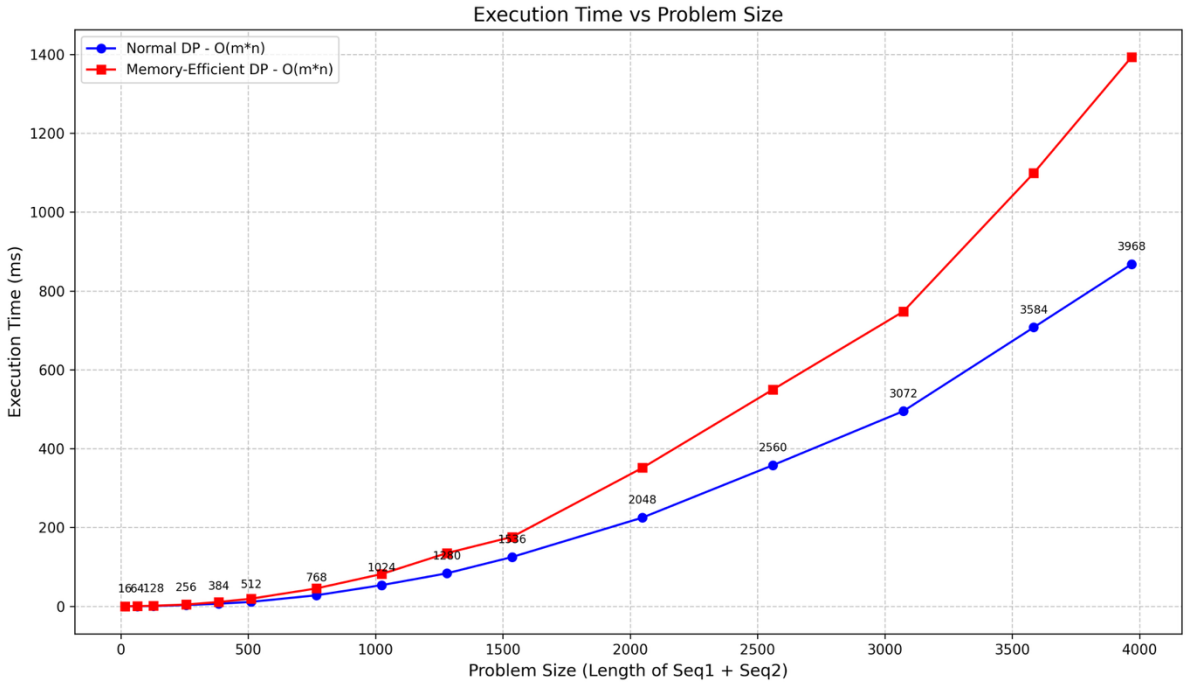
Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)

Basic: Polynomial

Efficient: Linear

*Explanation: The basic algorithm has a memory complexity of O(m*n), which is from the dp array having to fill all cells. The efficient algorithm has a complexity of O(m), which only requires two rows to (current and previous) to get the final answer.*

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial

Efficient: Polynomial

*Explanation: Both algorithms need to finish the whole dp array, which contains m*n elements. Every value is calculated with simple addition and comparison. The efficient algorithm requires some more calculating because when calculating the min cost, we only save the previous row and current row, which makes backtracking only able to track one row in front, we would need to calculate the previous row every iteration which would cause more time. Nonetheless, we only need to calculate at most m*n times.*

Contribution

Individual work.