SUMMARY

USC ID/s: 3679585849

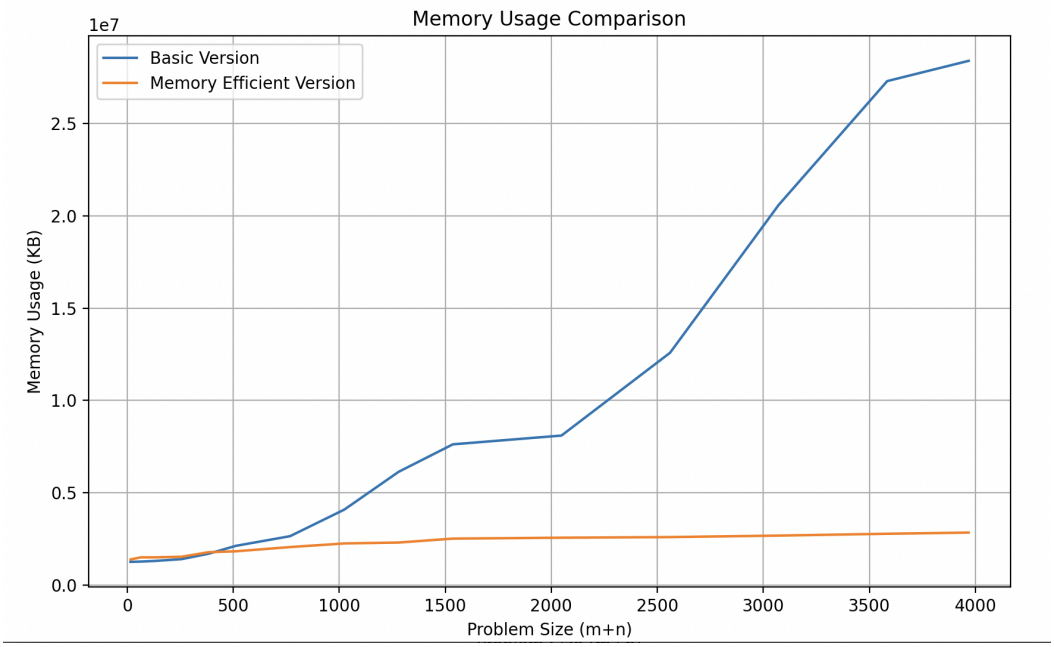
Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.024(ms)	0.095(ms)	1245184(KB)	1376256(KB)
64	0.195(ms)	0.582(ms)	1261568(KB)	1490944(KB)
128	0.729(ms)	1.907(ms)	1294336(KB)	1490944(KB)
256	2.786(ms)	6.582(ms)	1392640(KB)	1523712(KB)
384	6.209(ms)	16.053(ms)	1687552(KB)	1769472(KB)
512	11.151(ms)	28.482(ms)	2113536(KB)	1818624(KB)
768	24.912(ms)	50.135(ms)	2637824(KB)	2048000(KB)
1024	44.941(ms)	86.267(ms)	4079616(KB)	2244608(KB)
1280	68.843(ms)	130.610(ms)	6127616(KB)	2293760(KB)
1536	96.822(ms)	183.312(ms)	7618560(KB)	2506752(KB)
2048	177.071(ms)	335.720(ms)	8093696(KB)	2555904(KB)
2560	274.003(ms)	518.107(ms)	12582912(KB)	2588672(KB)
3072	383.559(ms)	719.758(ms)	20594688(KB)	2670592(KB)
3584	539.232(ms)	1011.669(ms)	27312128(KB)	2768896(KB)
3968	662.058(ms)	1236.576(ms)	28409856(KB)	2834432(KB)

Insights

As the size increases, both the basic version algorithm and the memory-efficient version algorithm experience an increase in time cost. However, the memory-efficient version increases more quickly because it does not utilize the dynamic programming method. Furthermore, while the memory usage increases polynomially as the size increases in the basic version algorithm, the memory-efficient version increases nearly linearly.

Graph1 – Memory vs Problem Size (M+N)



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

Basic: Polynomial

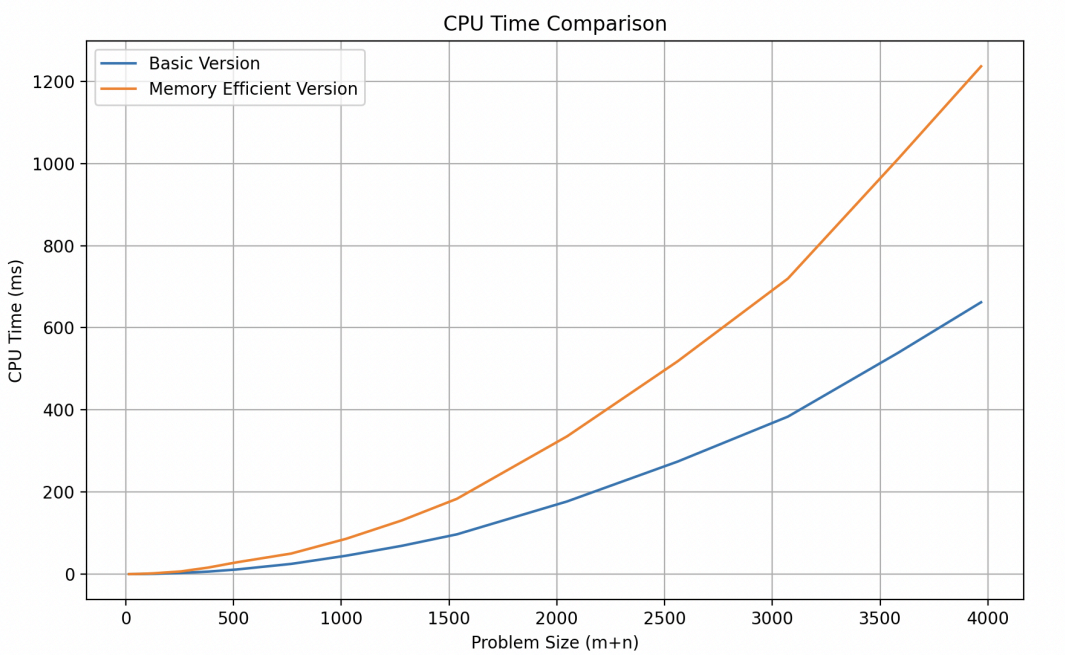
Efficient: Linear

Explanation:

In the basic method, we build a 2-D table (matrix) to store all the values of the dynamic programming matrix. The size of this matrix is typically O(m * n), where m and n are the lengths of the input sequences. Therefore, the memory cost is polynomial in terms of the input size.

In the memory-efficient method, we use a divide-and-conquer approach with linear space optimization. Instead of storing the entire 2-D matrix, we only store the current row and the previous row of the matrix at any given time. This optimization reduces the memory required to O(n), where n is the length of the longer sequence. Therefore, the memory cost is linear in terms of the input size.

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial

Efficient: Polynomial

Explanation:

In the basic method, we fill up the entire 2-D table (matrix) using dynamic programming to compute the alignment scores for all possible pairs of elements in the input sequences. The time complexity of this process is typically O(m * n), where m and n are the lengths of the input sequences. Therefore, the time cost is polynomial in terms of the input size.

The memory-efficient method uses a divide-and-conquer approach, which can indeed be applied to sequence alignment problems in polynomial time using techniques such as the master theorem. The time complexity of the memory-efficient method can still be polynomial.

Contribution

(Please mention what each member did if you think everyone in the group does not have an equal contribution, otherwise, write “Equal Contribution”)

<USC ID/s>: Completed by myself.