SUMMARY

USC ID/s:

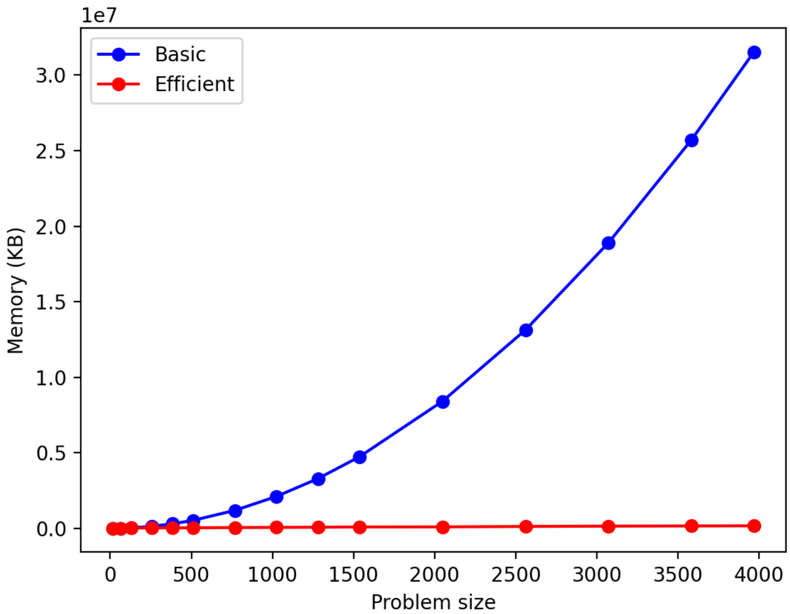
4965781366, 5920401239, 5857424113

Datapoints

M+N	Time in M (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	1.752853394	1.99484825	1784	8523
64	23.37598801	24.3937969	9848	13105
128	93.2841301	100.686789	34944	18355
256	408.6949825	379.305124	134666	28139
384	884.9947453	791.834116	299818	38095
512	1797.211885	1406.60191	530269	36537
768	4365.257978	3161.56507	1188170	53583
1024	8133.978844	6174.69382	2108274	64398
1280	11393.62788	8940.58013	3290753	74551
1536	14671.66686	15391.0298	4734938	92053
2048	32566.16998	24599.575	8409737	96327
2560	48204.05984	36080.982	13134102	125217
3072	69435.50301	53270.9701	18907610	145439
3584	68718.79911	81970.6123	25726217	158873
3968	100089.5708	97679.2312	31530350	168878

Insights

Graph1 – Memory vs Problem Size (M+N)

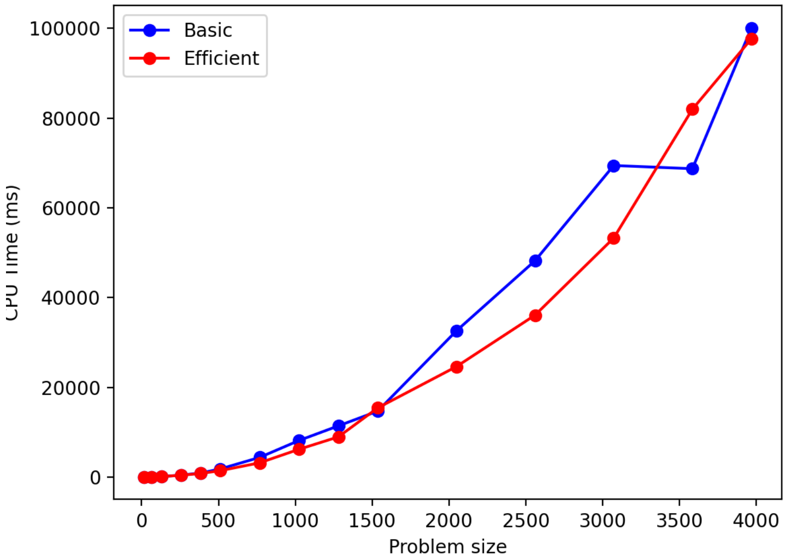


Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)
Basic: Polynomial
Efficient: Linear

Explanation:

The basic algorithm has a space complexity of O(mn) such that m and n are the lengths of the two strings X and Y, as it uses a m by n matrix to store all the subproblems in the DP algorithm. As seen in the graph, when the problem size m+n increases, the memory also increases in a O(mn) fashion. The efficient algorithm has a linear time complexity of O(min(m, n)) based on Hirschberg's algorithm. The algorithm takes advantage of the fact that the optimal solution must take a path going through some value in the middle column of the m by n matrix, making it easily applicable to divide and conquer. It improves upon the basic algorithm by only utilizing values from the previous column and continuously overwriting the values in the array of size min(m, n) without requiring any extra space.

Graph2 – Time vs Problem Size (M+N)



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

Explanation:

The basic algorithm has a time complexity of O(mn), since initialization and setting the values of the m by n matrix requires a double for loop through m and n, filling in every value in order for the DP algorithm to work. The efficient algorithm manages to make space improvements while still maintaining the same O(mn) runtime. This can be generally shown through induction on the recurrence $T(m, n) \leq mn + T(m/2,\ i) + T(m/2,\ n - i)$ $for\ i = 0,.., n$. With the induction hypothesis $T(m, n) \leq kmn$, we can see that $T(m, n) \leq mn + ki * m/2 + k * m/2(n - i) = mn + kn * m/2 = (1 + k/2)mn$ Thus, $T(m, n) \leq kmn$, and the divide and conquer method provides a space-efficient solution without major detriments to time.

Contribution

Equal contribution