SUMMARY

USC ID/s:

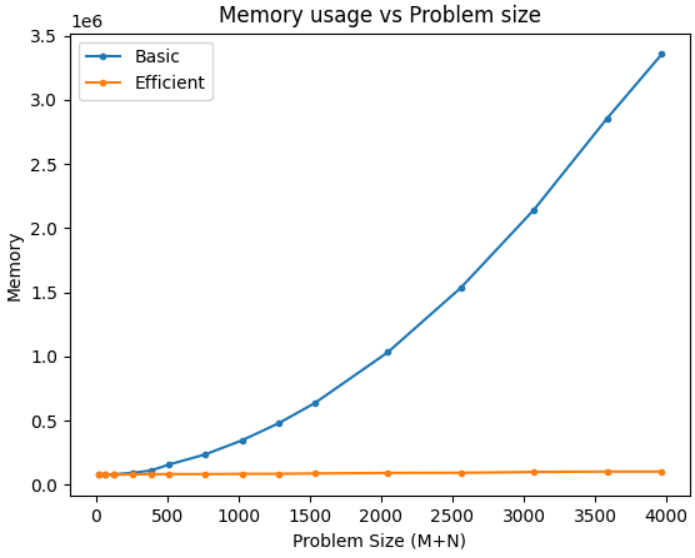
8138053893
1626160744

Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.092	0.210	77414	77004
64	0.249	0.797	76595	77824
128	0.707	2.191	80691	77414
256	2.807	6.514	92160	81100
384	7.238	12.697	109772	81510
512	10.196	24.156	156467	82739
768	22.140	48.510	236339	82739
1024	40.399	84.123	345292	84377
1280	75.973	131.283	478003	84787
1536	93.670	249.415	636928	87244
2048	217.612	377.600	1033420	91750
2560	259.211	515.170	1536410	92569
3072	381.130	908.852	2144260	99123
3584	438.149	1123.290	2855320	101990
3968	619.946	1438.690	3356670	101990

Insights

Graph1 – Memory vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Exponential)

Basic: Polynomial

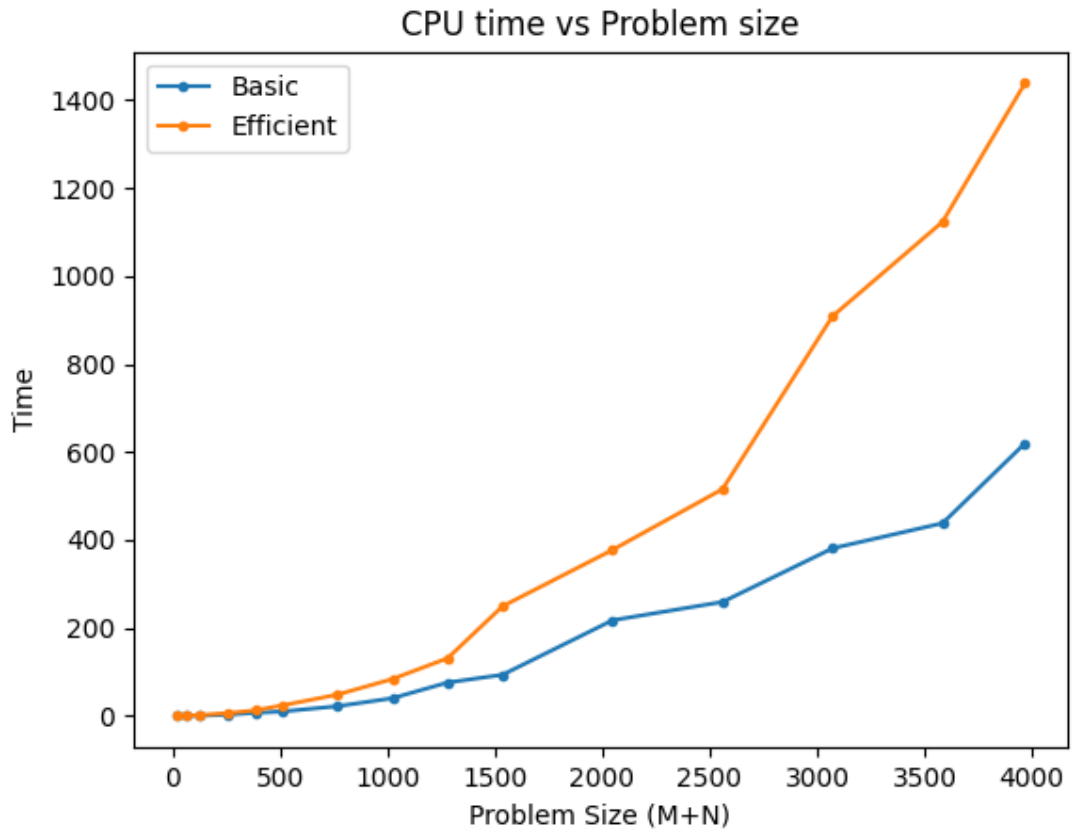
Efficient: Linear

Explanation:

For the basic algorithm, we implemented a dynamic programming solution with a 2 dimensional memory array. The memory space is of O(M*N) with M and N being the size of the generated string sequences.

With the efficient algorithm, however, we reduced the 2 dimensional array to two rows only as any value of dynamic programming array only depends on its previous three values. For instance, dp[i][j] would only needs values from dp[i-1][j], dp[i][j-1], dp[i-1][j-1]. Thus, size of 2*min(M,N) memory space is required to store all the necessary information.

Graph2 – Time vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Exponential)

Basic: Polynomial

Efficient: Polynomial

Explanation:

The time complexity for the basic algorithm is O(M*N). It is the time needed to fill the 2 dimensional dynamic programming array.

For the efficient version of this solution, we implement using divide and conquer. We treat the sequence alignment problem as a finding a path from (0,0) to (m,n) in our array. To apply divide and conquer, we could divide the path with a mid-pointin x and finding a divide point in y called j. The problem became finding a path from(0,0) to (mid,j) and from(mid,j) to (m,n). From there we could solve the problem recursively. This would give us we have M*N + M*N/2 + M*N/4 + ... < 2*M*N the total number of calculations, hence O(M*N)

Contribution

8138053893: Equal Contribution
1626160744 : Equal Contribution