SUMMARY

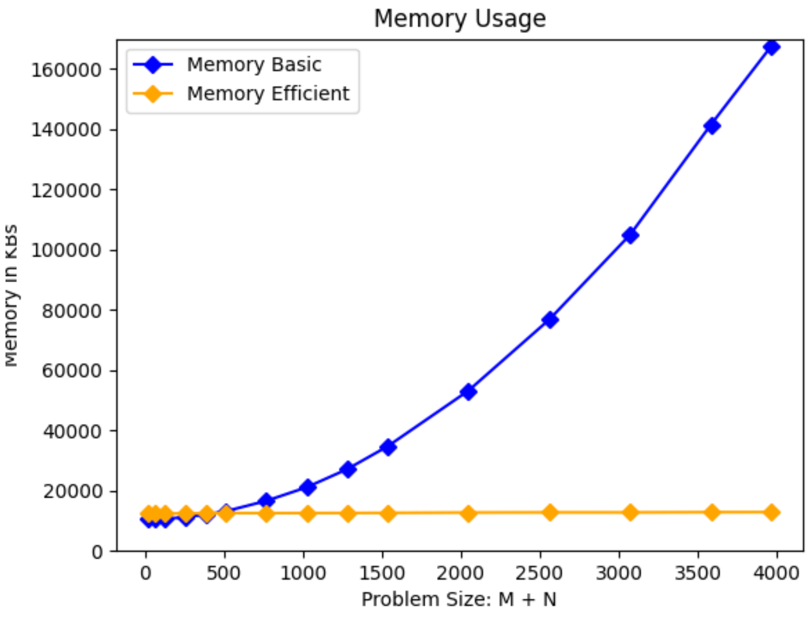
USC ID/s: 9974490095, 4636227623, 5150891285

Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.624179840087891	0.675201416015625	10456	12476
64	1.32393836975098	2.90274620056152	10496	12432
128	4.53877449035645	15.1779651641846	10636	12364
256	19.683837890625	34.9090099334717	11172	12468
384	26.7570018768311	55.8409690856934	12040	12448
512	53.5650253295898	104.321002960205	13156	12564
768	125.284671783447	222.446918487549	16584	12536
1024	211.942195892334	383.392095565796	21104	12552
1280	346.046924591064	629.167795181274	27100	12568
1536	460.262060165405	887.938022613525	34676	12604
2048	865.328311920166	1546.4608669281	53168	12724
2560	1316.11514091492	2438.58599662781	76788	12776
3072	1845.92223167419	3417.96684265137	104964	12764
3584	2643.03684234619	4643.72825622559	141528	12848
3968	3332.80682563782	5817.87300109863	167724	12864

Insights

Graph1 – Memory vs Problem Size (M+N)



Nature of the Graph

Basic: Polynomial

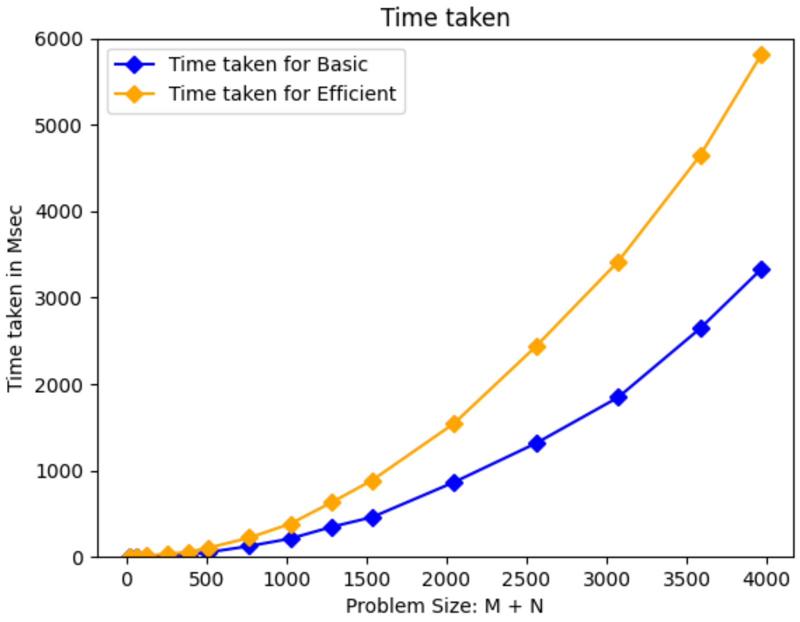
Efficient: Linear

Explanation:

The Basic Algorithm has an exponential space complexity, because we create a data structure of 2D 'table' with a size of m*n. The space needed to store and process data is directly proportional to the input size of the two sequences. The longer the input, the more memory space, and it can be a severe problem when the input size becomes tremendous.

However, the Efficient Algorithm only stores values (total alignment cost) of the optimal alignment which saves a lot of space. For this algorithm, we care about the values instead of the alignment itself that needs a table-like structure. To find the minimum alignment cost, it only uses 2 'arrays' with a total size of 2*M. We have one array of length M holding the value (cost) of the previous column, and based on that, we can then find out the value of the current array. Therefore, for each iteration, we only take up a constant memory space.

Graph2 – Time vs Problem Size (M+N)



Nature of the Graph

Basic: Polynomial

Efficient: Polynomial

Explanation:

The running time of both the basic and the memory-efficient algorithms is O(m*n), because it takes constant time to determine the value in each of the m*n cells of the cost array. Also, the running time of the memory-efficient algorithm is around twice of the basic one, and this is because in the memory-efficient algorithm, some of the cells are visited multiple times during the divide and conquer stage to determine the alignment, and this blows up the running time of the memory-efficient algorithm by a factor of two, compared to the basic one.

Contribution

9974490095, 4636227623, 5150891285: Equal Contribution