SUMMARY

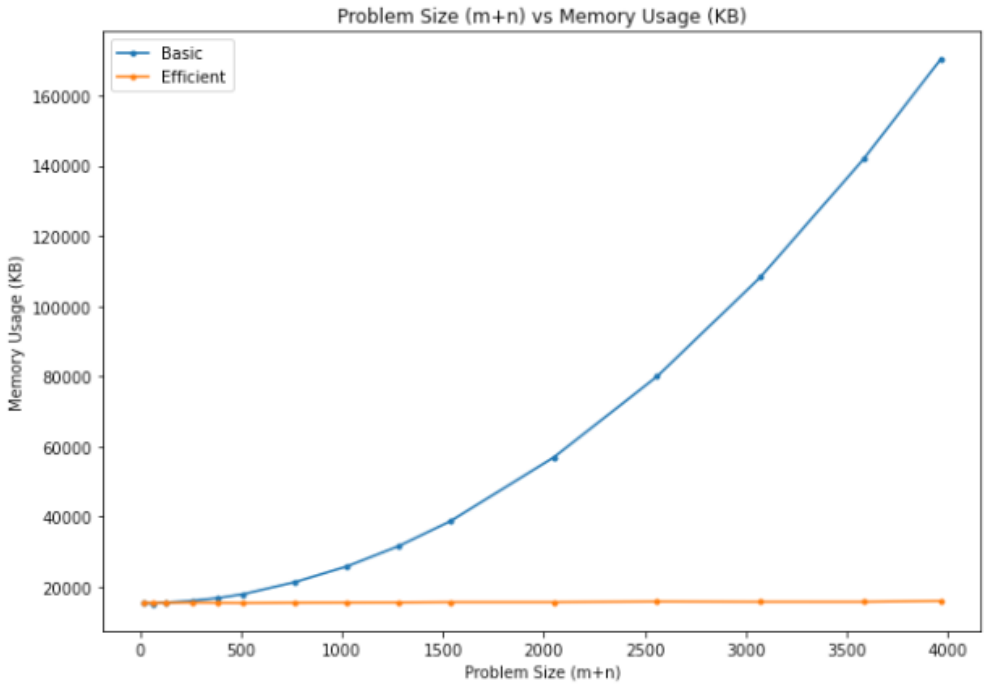
USC ID/s:
1757489699, Preston Fong
3698334829, Cameron Schultz

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.10824203491210938	0.24700164794921875	15280	15504
64	0.7059574127197266	1.680135726928711	15268	15392
128	0.9965896606445312	2.9900074005126953	15544	15476
256	2.9904842376708984	6.976604461669922	16056	15520
384	7.972955703735352	14.950275421142578	16820	15464
512	12.95614242553711	27.90665626525879	17952	15380
768	30.896902084350586	62.790393829 3457	21388	15492
1024	55.81307411193848	106.6434383392334	25836	15552
1280	86.70997619628906	170.49765586853027	31616	15588
1536	124.58372116088867	235.47053337097168	38700	15684
2048	224.24983978271484	426.4991283416748	56880	15684
2560	352.81968116760254	674.7837066650391	79936	15816
3072	525.2430438995361	946.4926719665527	108212	15752
3584	712.6164436340332	1349.3218421936035	141916	15748
3968	881.0622692108154	1652.9595851898193	170520	16008

Datapoints

Insights

Graph1 – Memory vs Problem Size (M+N)



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

Basic: Polynomial [O(mn)]
Efficient: Linear [O(m)]

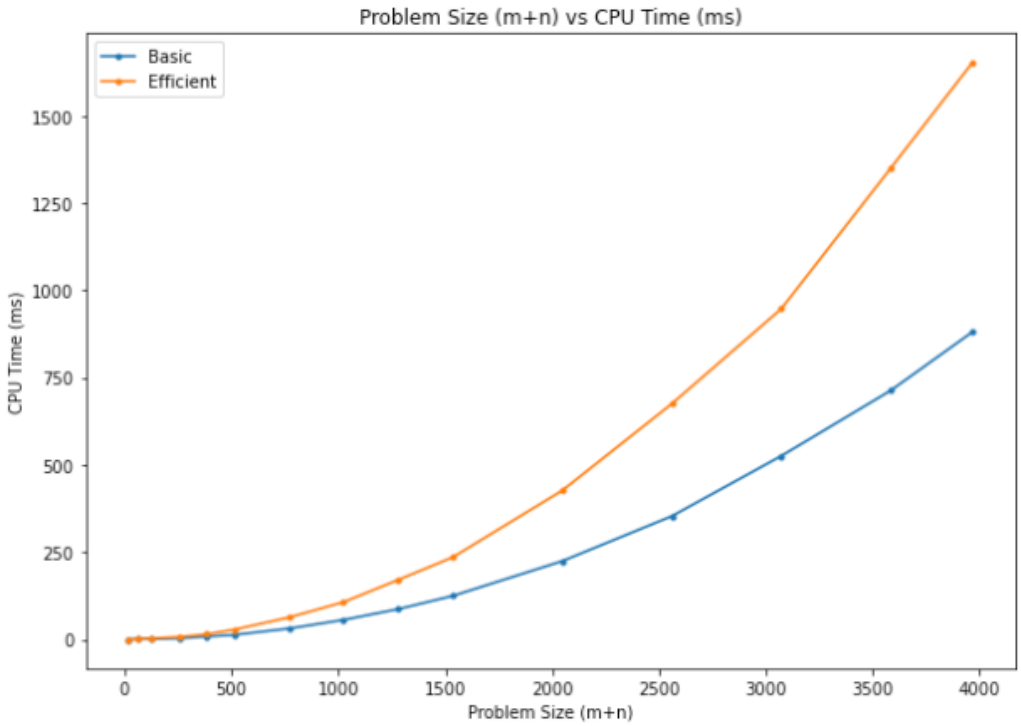
Explanation:

For the Basic algorithm, we need a dynamic programming matrix of size m*n. Without loss of ambiguity, if m>n, we can bound this function by n^2 < basic < m^2. Therefore, the memory usage is polynomial with respect to input size, m and n.

For the efficient algorithm, in order to calculate the split, we only require the storage of two matrix rows at any given time. This gives us a matrix of at most size 2m, which is a linear relationship with respect to the input size m.

These two relationships can be shown in the graph above, where the basic function appears as a polynomial line, and the efficient algorithm appears as a constant linear function.

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial [O(mn)]
Efficient: Polynomial [O(mn)]

Explanation:

For the basic algorithm, the resultant dynamic programming matrix requires a calculation of m*n values. In order to calculate the minimum cost sequence alignment, we need to iterate to the last entry of the matrix, which stores the optimal alignment cost. This will require a total of m*n calculations, each of which are completed in constant time.

For the efficient algorithm, we need to calculate two matrices, each of size (m/2)(n). The combined total number of calculations for this first split is m*n, and for every split following, we reduce m by a factor of 2 and n by the split factor. The total calculations required for the completion of this algorithm is therefore 2[(mn/2)+(mn/4)+(mn/8)+ …] = 2mn. These calculations will also be completed in constant time.

As we can see from the above graph, the efficient algorithm requires twice as many constant time calculations, and therefore takes about twice as long to run as the basic algorithm.

Contribution

Equal Contribution.