SUMMARY

USC ID/s: 4361009421

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.008843899	0.017263198	0	0
64	0.0410981	0.1370078	0	0
128	0.1202128	0.2904779	0	0
256	1.0252639	1.430603	0	0
384	1.0973561	1.4060497	0	68.1568
512	2.5047598	2.1851351	0	68.1568
768	1.8189248	3.4040625	68.1568	204.4704
1024	2.795803	12.364924	136.3136	340.784
1280	3.6886468	6.3825455	340.784	545.2584
1536	4.228153	6.1195254	408.9408	817.9856
2048	4.4664116	12.053415	613.4112	1295.1904
2560	6.814395	14.587878	1363.136	1976.8145
3072	6.635735	24.806742	2659.9736	2726.6768
3584	6.1024027	17.69344	1908.3904	279.332
3968	7.822385	16.684355	2613.9648	1028.5024
6144	25.314775	47.79627	1771.704	210.764
8192	33.233894	67.55033	4155.738	1030.1312
10240	109.67325	127.40256	9511.294	4025.839
12288	69.32502	163.8314	2961.236	2201.232
16384	98.44889	238.72444	15954.476	18455.766
20480	196.12692	276.90234	14210.672	2569.7449

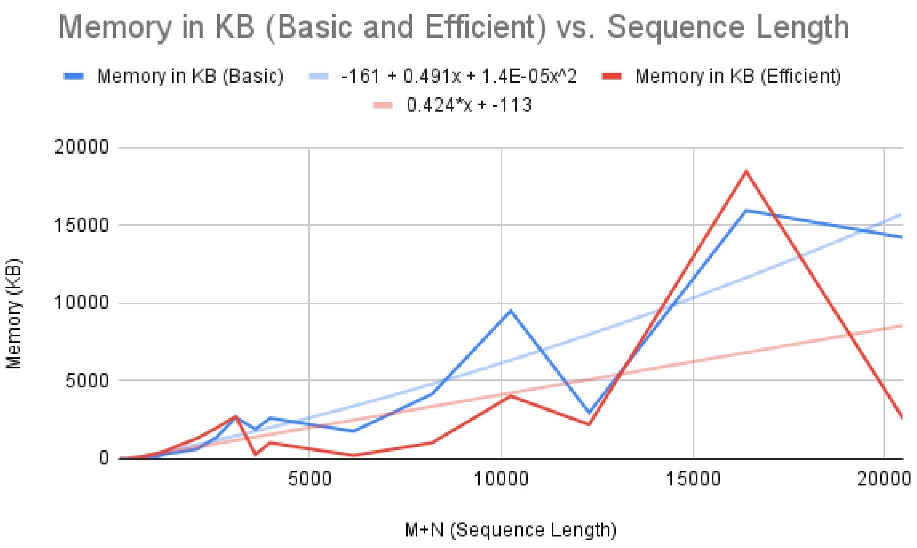
Datapoints

Insights

In this project, we will compare the memory efficient solution and the basic solution of the sequence alignment problem. We will obtain data on each solution's performance in time and memory and generate line graphs regarding time and memory vs. sequence length. The graphs indicate that the memory efficient solution has an overall slower rate of increase in memory compared to the basic solution. Additionally, the memory efficient solution has an overall faster rate of increase in runtime compared to the basic solution. The data points and graphs show that the memory efficient solution requires less memory but more runtime while the basic solution requires more memory but less runtime.

To display a clearer trend, I have included 6 additional data points (m+n is 6144, 8192, 10240, 12288, 16384, 20480) and added System.gc() before and after the time and memory calls in both solutions. I have also tested extremely large inputs and noticed that the basic solution is unable to generate the output due to Java heap error while the memory efficient solution correctly produces the output.

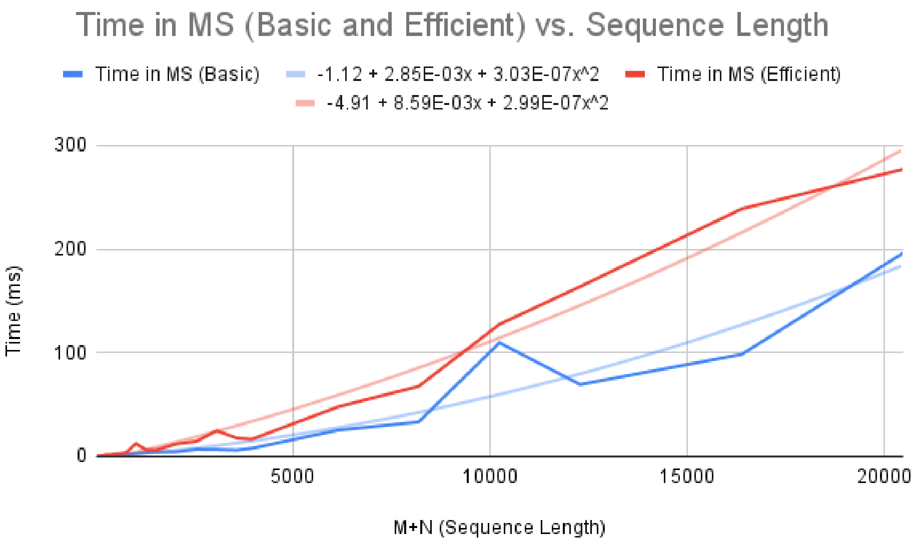
Graph1 – Memory vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Exponential)

Basic: Polynomial
Efficient: Linear
Explanation: The basic solution graph is polynomial because the memory needed to store the 2D m by n array of optimal costs would be mn. The efficient solution graph is linear because it only requires two columns, each with at most m+n length stored at a time; the amount of storage for the efficient solution would be 2(m+n).

Graph2 – Time vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Exponential)

Basic: Polynomial
Efficient: Polynomial
Explanation: The basic graph is polynomial because it takes O(mn) time to fill all values of the 2D m by n array holding the optimal costs. The efficient graph is polynomial because the efficient solution would take twice the time of the basic solution. After combining all the recursive steps, the time complexity of the efficient solution is 2*c*mn, so the efficient solution also works in O(mn) time.

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>: <Equal Contribution>

4361009421: Equal Contribution