SUMMARY

Nicolas Moy, Harshawn Singh, Abhi Surendra

USC ID/s: 4155139297 4689235825 3468053994

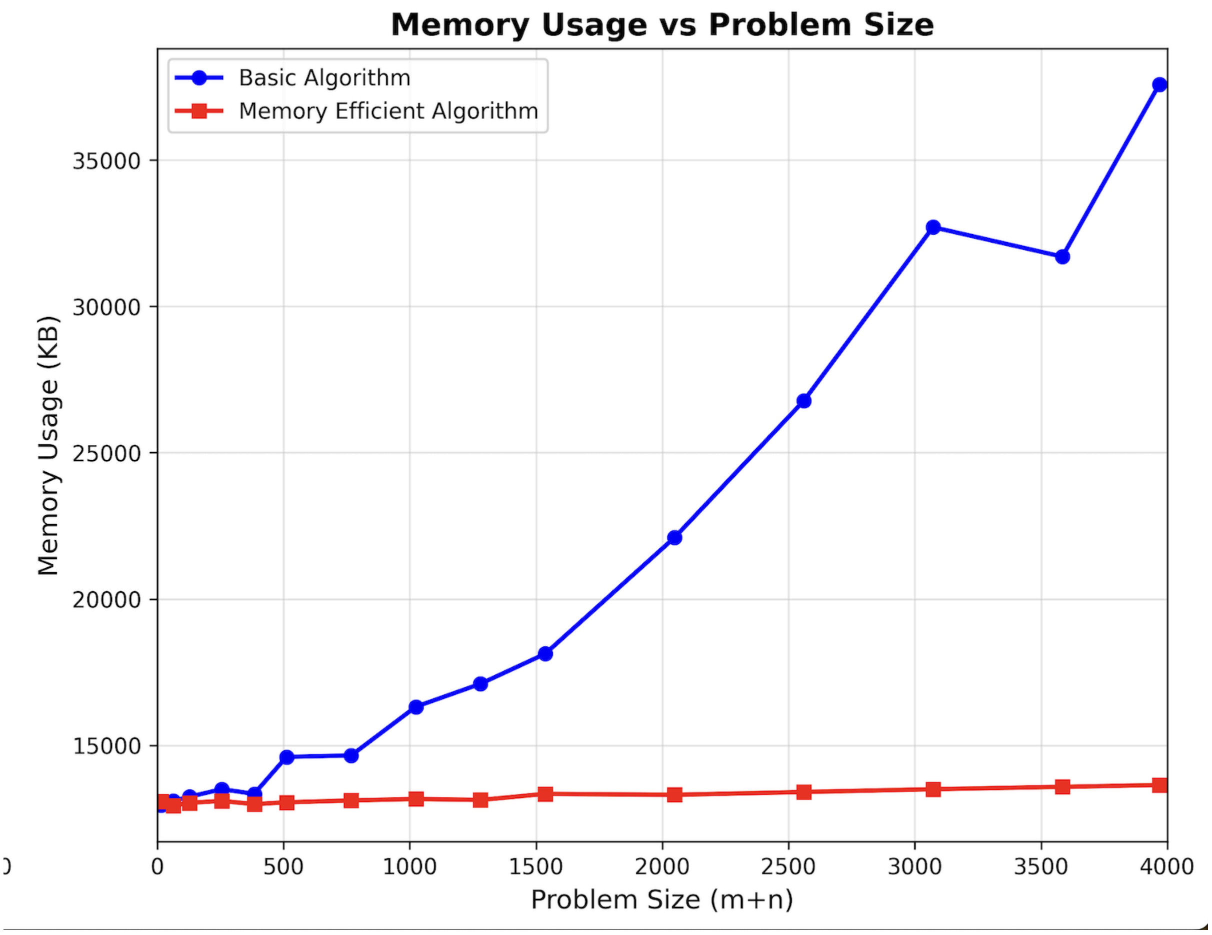
Datapoints

M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.03933	0.062	12960	13088
64	0.25701	0.463	13104	12944
128	0.89502	1.709	13248	13040
256	3.34787	5.911	13504	13104
384	7.39789	14.094	13344	12992
512	13.14998	23.966	14608	13056
768	33.00881	60.189	14656	13120
1024	59.66306	102.471	16320	13168
1280	94.12908	160.032	17104	13136
1536	135.9539	224.924	18128	13344
2048	249.6631	426.818	22096	13312
2560	400.0709	655.887	26768	13408
3072	570.7142	954.119	32704	13504
3584	772.0508	1307.074	31696	13584
3968	950.09	1646.218	37584	13648

Insights

For both algorithms, the execution time increases dramatically with problem size - the basic algorithm grows from 0.039 ms to 950 ms (a 24,000x increase), while the efficient algorithm increases from 0.062 ms to 1,646 ms (a 26,500x increase). This slightly higher growth rate for the efficient algorithm is shown on the CPU time graph and is due to the divide and conquer approach. The memory measurements show even bigger differences: the basic algorithm's memory usage increases by 190% (from 12,960 KB to 37,584 KB), while the efficient algorithm's memory consumption increases by only 4.3% (from 13,088 KB to 13,648 KB). This confirms that the memory-efficient algorithm successfully reduces space needed while keeping CPU time decently similar.

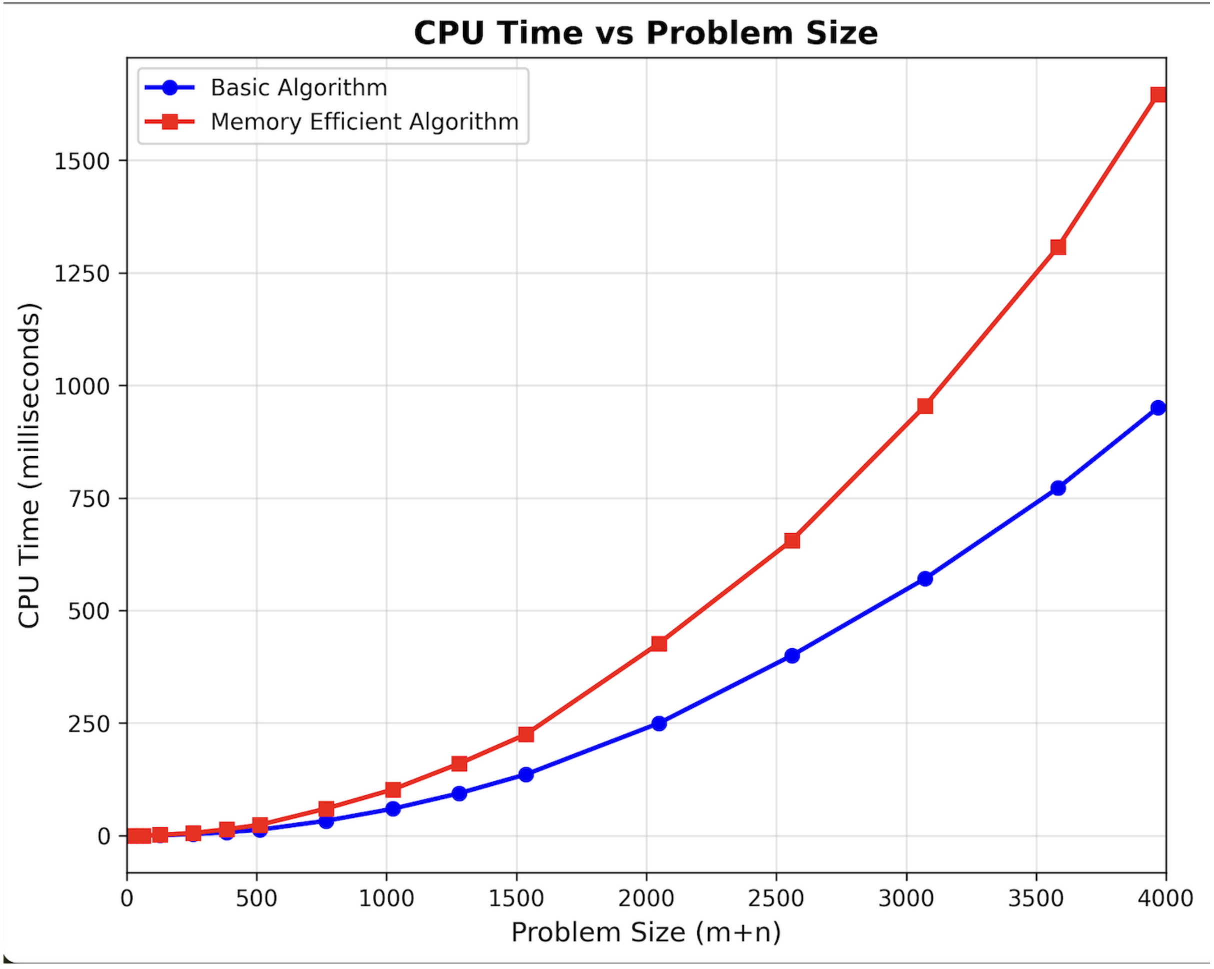
Graph1 – Memory vs Problem Size (M+N)



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

The memory usage graph really shows the key advantage of the memory-efficient algorithm. While the basic algorithm's memory consumption grows from approximately 13 MB to 37.5 MB as the problem size increases, the efficient algorithm memory usage stays around 13 MB throughout all test cases. This difference shows the space complexity reduction from O(mn) for the basic algorithm to O(m+n) for the efficient version. The horizontal line for the efficient algorithm graph shows its success in achieving constant space complexity relative to the product of sequence lengths. At the largest problem size tested, the memory-efficient algorithm uses only about 36% of the memory required by the basic algorithm, representing a significant 64% reduction in memory consumption while solving the same problem.

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial

Efficient: Polynomial

Explanation:

The CPU time graph reveals that both algorithms exhibit quadratic growth patterns, confirming the O(mn) time complexity expected for our sequence alignment problems. The basic algorithm consistently outperforms the memory-efficient version. The memory-efficient algorithm shows similar characteristics to the basic but runs approximately 1.5-2 times slower across all problem sizes. This is from the divide and conquer approach which requires additional function calls and splitting operations at each level. As problem size increases, the relative performance ratio remains fairly constant showing that both algorithms scale similarly with input size but the divide and conquer adds an extra constant increase to the efficient algorithm's CPU 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”)
4155139297: Equal Contribution
4689235825: Equal Contribution
3468053994: Equal Contribution