SUMMARY

USC ID/s: 6421273823, 8185460719

Datapoints

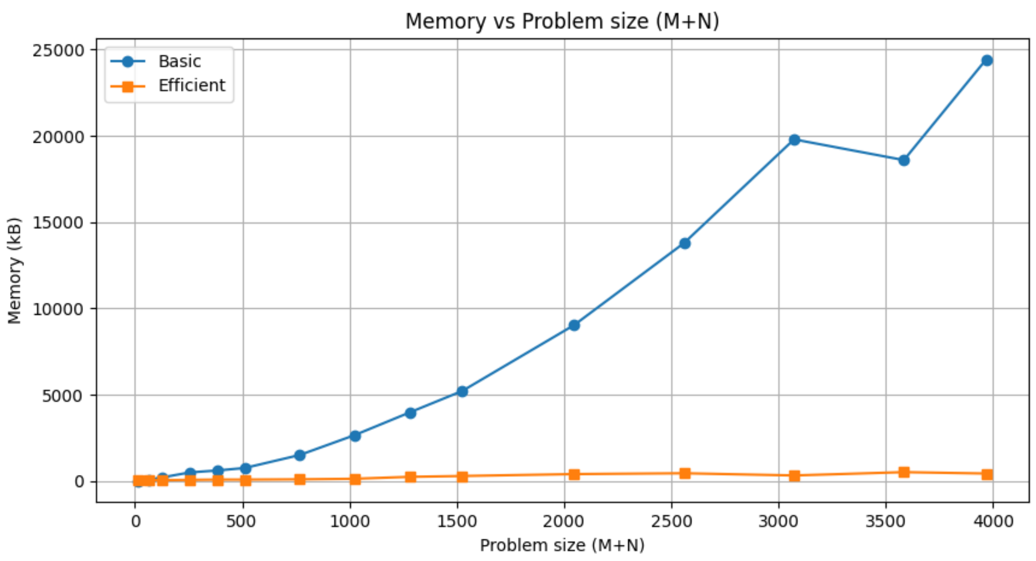
M+N	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	0.0391006469726563	0.0741481781005859	16	48
64	0.355958938598633	0.624895095825195	64	48
128	1.27124786376953	2.18486785888672	208	48
256	5.00679016113281	7.78698921203613	496	64
384	10.2739334106445	17.3630714416504	608	80
512	18.6717510223389	32.3729515075684	752	80
768	44.8658466339111	70.9590911865234	1504	96
1024	78.9182186126709	131.012916564941	2656	128
1280	123.322010040283	199.260234832764	3968	240
1536	184.705972671509	283.928155899048	5216	288
2048	327.716112136841	524.778127670288	9040	400
2560	524.760961532593	816.53618812561	13792	448
3072	744.256973266602	1142.14086532593	19792	320
3584	1051.81288719177	1594.21420097351	18592	512
3968	1252.10690498352	1958.82797241211	24432	432

Insights

There are two methods that have been used here to solve the sequence alignment problem. First, basic method (only DP), and second, memory efficient method (DP + Divide & Conquer).

1. The basic approach for sequence alignment involves dynamic programming to figure out the gaps, mismatches and matches and outputs the minimum cost of alignment between the given two strings. The time and space complexity for this approach are both O(m*n) where m and n are the sizes of the given input strings.
2. The space complexity poses a challenge due the nature of sequence alignment problem where the input sizes of DNA sequences can be unimaginably big. The memory efficient approach overcomes this challenge by utilizing divide and conquer, in addition to dynamic programming.
3. Memory efficient approach: There are 2 input strings X and Y. X is split into 2 equal halves - Xr and Xl. We then calculate the alignment cost for Xl with each substring of Y and similarly, for Xr with each substring of Y (reversed). In our code, the function returns 2 1-D arrays from which we calculate the minimum alignment cost for the sub problem. This procedure is recursively repeated for the subsequent sub problems.
4. The memory efficient approach for sequence alignment has a time complexity same as the basic approach. But, the space complexity is significantly reduced to linear space, i.e. O(m+n), which can also be seen below.

Graph1 – Memory vs Problem Size (M+N)



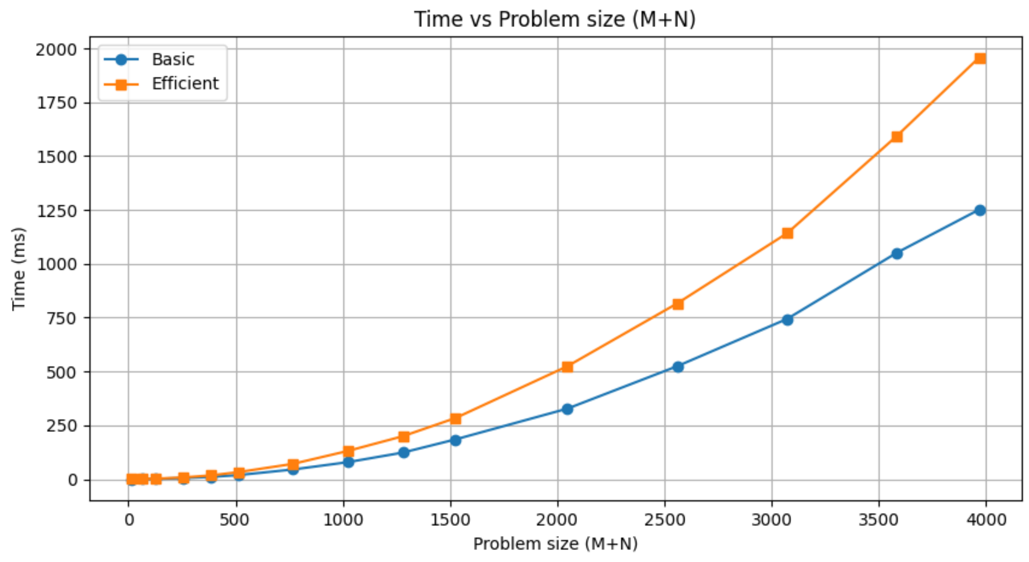
Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)

Basic: Polynomial
Efficient: Linear

Explanation:

The above graph depicts the plot of memory against the problem size. As we can see, the basic approach has a polynomial curve whereas, the memory efficient approach has almost linear curve. This shows the relation between the increasing problem size and the memory requirements. We can infer from the graph that the memory required basic approach increases at a very fast pace for increasing problem size as compared to the efficient approach.

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial

Efficient: Polynomial

Explanation:

The above graph depicts the plot of running time against the problem size. As we can see, both the basic and efficient approaches have polynomial curves. This is expected because both the approaches have O(m*n) time complexity. It can also be seen that the efficient approach takes more time as compared to the basic approach because we are performing double the amount of operations with he divide and conquer strategy in the efficient approach. This trade off is beneficial, given the amount of reduction in the memory requirements.

Contribution

Equal Contribution