SUMMARY

USC ID/s:

3003578247
6103512846
9428726172
8376583523

Datapoints

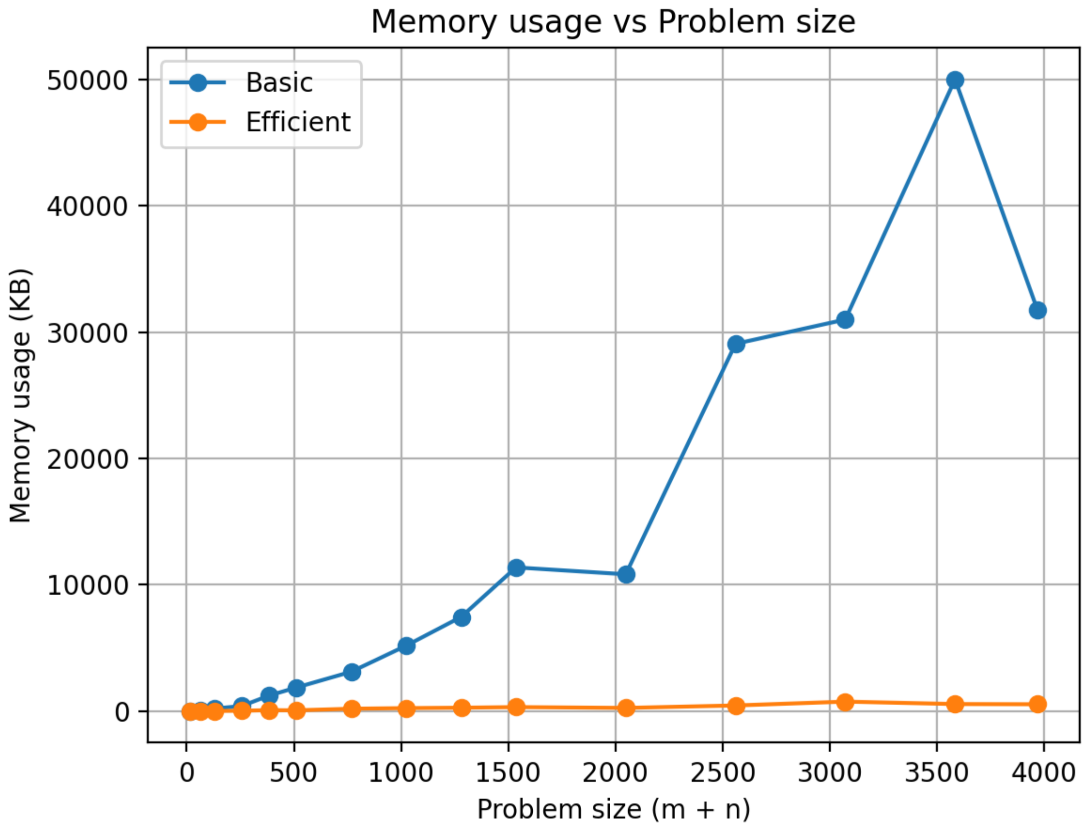
size	time_basic	mem_basic	time_efficient	mem_efficient
16	0.05698204040527340	0.0	0.11014938354492200	0.0
64	0.5211830139160160	64.0	0.9119510650634770	16.0
128	2.1572113037109400	208.0	3.6628246307373000	16.0
256	8.336782455444340	400.0	13.397932052612300	32.0
384	18.435955047607400	1232.0	31.261920928955100	80.0
512	32.67097473144530	1872.0	54.221153259277300	64.0
768	74.66912269592290	3120.0	164.54076766967800	192.0
1024	137.14194297790500	5184.0	215.51895141601600	240.0
1280	218.33395957946800	7456.0	333.5998058319090	272.0
1536	290.65918922424300	11376.0	480.81493377685500	320.0
2048	556.4789772033690	10832.0	876.8582344055180	256.0
2560	845.1242446899410	29072.0	1333.1551551818800	448.0
3072	1210.434913635250	30992.0	1891.4589881897000	752.0
3584	1697.9060173034700	50000.0	2658.862829208370	560.0
3968	2160.665988922120	31776.0	3250.148296356200	544.0

Insights

- Memory Usage:
    - The basic algorithm using dynamic programming requires a much larger memory size since it is creating an n*m table. By filling this entire table, the size needed for larger cases balloons fast: The quadratic growth goes from 1,232 KB when the size is n=384 and ends up costing 31,776 KB on the largest test case of n=3968.
    - Our efficient algorithm (Hirschberg's) chooses avoid filling out an entire table since the only requirements to compute alignment are the current and previous rows. By doing this, large portions of memory are saved at the cost of needing to recompute certain parts of the table later on. As a result, when n=384, the algorithm uses 80 KB and when n = 3968, the algorithm uses 544 KB.
    - As we can see, the memory gains from Hirschberg's are significant.
- Time Performance:
    - Time-wise, both algorithms are in the same ballpark. At the smallest size (n = 16), the basic algorithm takes 0.057ms and the efficient algorithm takes 0.110ms. At the largest size (n = 3968), basic takes 2.16 seconds and efficient takes 3.25 seconds. So efficient is about 1.5x slower, which makes sense given the recursion overhead.
- Other Notes:

- For very small inputs (size 16), both show 0KB memory. This is most likely measurement noise due to how small the problem size is.
- The memory measurements can be a bit inconsistent (basic dropping at size 2048), but the cost values remain similar throughout, meaning both algorithms are finding the correct optimal solution.

Graph1 – Memory vs Problem Size (M+N)



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

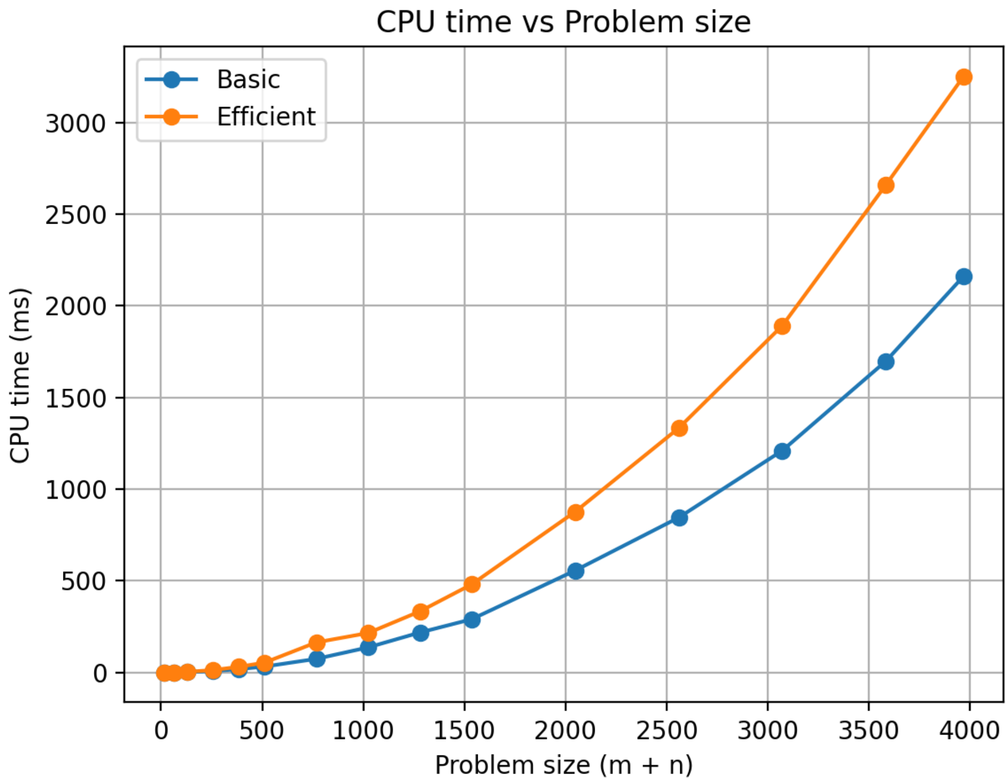
Basic: Polynomial (O(m*n))

Efficient: Linear (O(m+n))

Explanation:

- The basic algorithm uses a standard DP approach of filling out the whole n*m table. The efficient algorithm instead chooses to only keep the current and previous row during each step, while forgetting the rest at the cost of needing to recalculate it later.
  - By keeping the full table, memory usage for our basic algorithm becomes O(n*m) ~ O(n^2)
  - Our efficient algorithm will only keep two rows, meaning n+m instead of n*m. This efficiency results in a memory complexity of O(n+m) ~ O(n)
- Both algorithms start showing memory usage at size 64. Basic uses 64KB while efficient only needs 16KB, showing a 4x improvement. As the problem size grows, the size of the gap increases as well. At size 3968, basic is using 31,776KB while efficient only needs 544KB. Showing a difference factor of almost 58x.
- The key takeaway is that basic's memory grows quadratically with problem size, while efficient grows linearly. The graph supports that efficient is better for memory, especially as the problem size increases.

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial (O(mn))

Efficient: Polynomial (O(mn))

Explanation:

- From the graph, both algorithms appear to have a similar pattern as the problem size increases. Since both algorithms are comparing all values in both strings, we see a quadratic time complexity $O(m*n) \sim O(n^2)$
- In addition to the time complexity, we can see that in practice the efficient algorithm takes longer than the basic algorithm as the problem sizes increase. This is due to the amount of overhead being done in the Hirschberg memory efficient algorithm. As stated above, the algorithm chooses to forget prior rows at the cost of needing to potentially recalculate it later. Since the algorithm performs a forward and backward pass, it ultimately does take more time than simply tabulating the results for recall. This divide and conquer strategy, while saving lots of space, will on average take the same time as the basic algorithm.

Contribution

(Please mention what each member did if you think everyone in the group does not have an equal contribution, otherwise, write “Equal Contribution”)

8376583523: Equal Contribution
6103512846: Equal Contribution
9428726172: Equal Contribution
3003578247: Equal Contribution