**SUMMARY**

## USC ID/s:

5942406612 Atharva V Wagh

5990219480 Omkar P Shinde

5480473628 Chanatip Kriengkraipetch

9306851031 Sarkis Gafafyan

8868226878 Hsu Ting-Wei

## Datapoints

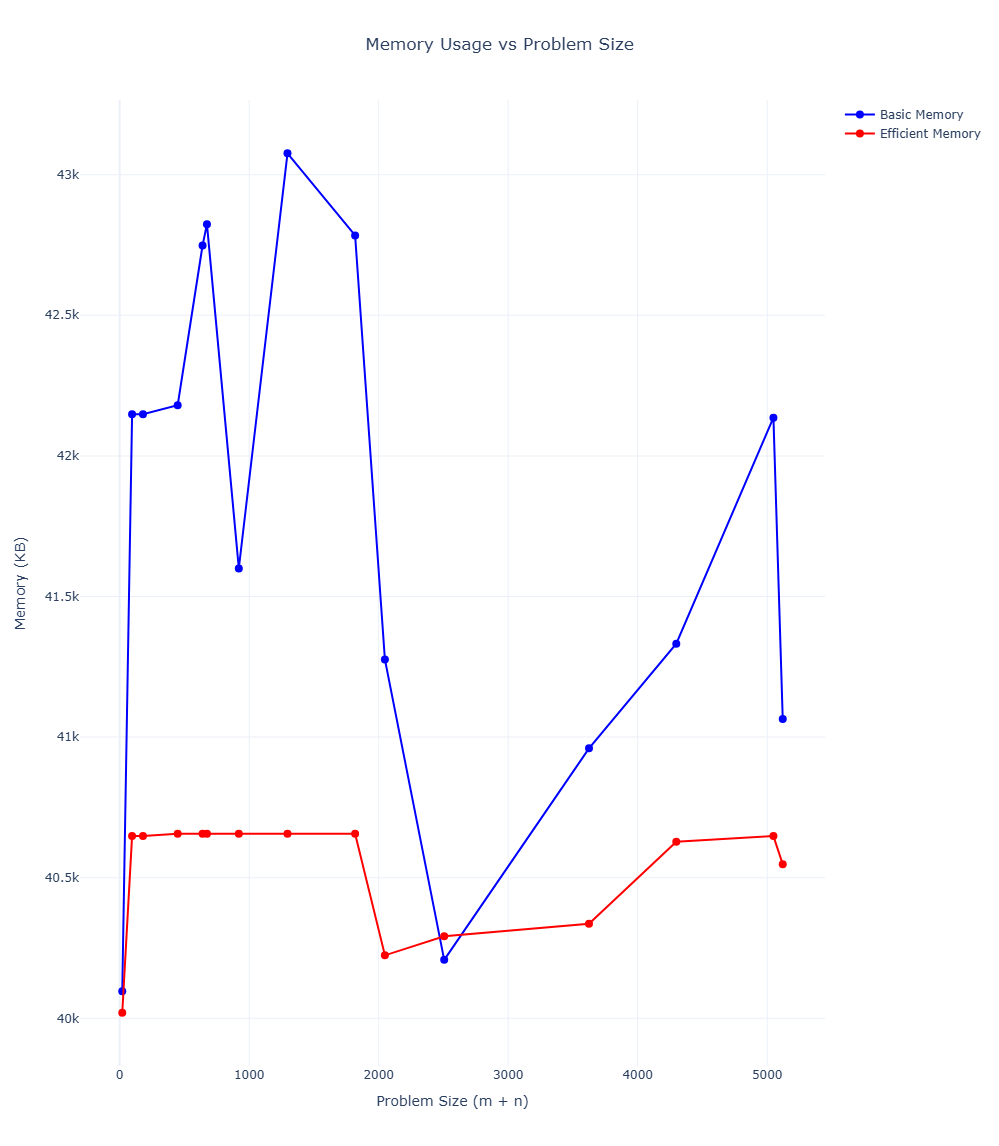
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M+N | Time in MS (Basic) | Time in MS (Efficient) | Memory in KB (Basic) | Memory in KB (Efficient) |
| 16 | 0.0 | 0.0 | 40548 | 40020 |
| 64 | 0.0 | 0.0 | 42148 | 40648 |
| 128 | 1.0 | 2.01 | 42148 | 40648 |
| 256 | 4.17 | 6.69 | 42180 | 40656 |
| 384 | 11.62 | 17.5 | 42748 | 40656 |
| 512 | 12.41 | 28.05 | 42824 | 40656 |
| 768 | 29.82 | 65.69 | 41600 | 40656 |
| 1024 | 56.13 | 122.78 | 43076 | 40656 |
| 1280 | 95.3 | 191.03 | 42784 | 40656 |
| 1536 | 137.17 | 277.9 | 41276 | 40224 |
| 2048 | 245.02 | 510.95 | 40208 | 40292 |
| 2560 | 386.87 | 806.9 | 40960 | 40336 |
| 3072 | 757.76 | 1503.77 | 41332 | 40628 |
| 3584 | 949.56 | 1973.8 | 42136 | 40648 |
| 3968 | 557.58 | 1114.02 | 41064 | 40548 |

## Insights:

As the combined length of increases, the running time of both algorithms grow quickly, which aligns with the expected polynomial time of , which is the time complexity of the sequence alignment. For minute inputs (which would be inputs that correspond to ) both algorithms finish instantly. For instance, at the basic algorithm takes ~387 ms while the efficient algorithm takes ~807ms, & at the times are about 950ms vs. 1974ms. As we can see, the efficient version is about 2x slower than the basic implementation on larger test cases, this is due to the fact that it performs multiple linear-space DP passes and multiple recursive calls, even though they both have the same asymptotic time.

The memory data shows a large fixed overhead of around 40,000 KB from the Python process, but the two algorithms scale much differently as the problem grows. The basic algorithm memory climbs above 42,000 KB for large inputs, while the efficient algorithm stays much flatter, around 40,800 - 41,000 KB. This matches the theory, that the basic method uses space, whereas the Hirschberg version uses only , so it does save memory but, at the cost of running extra time.

### Graph1 – Memory vs Problem Size (M+N)



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

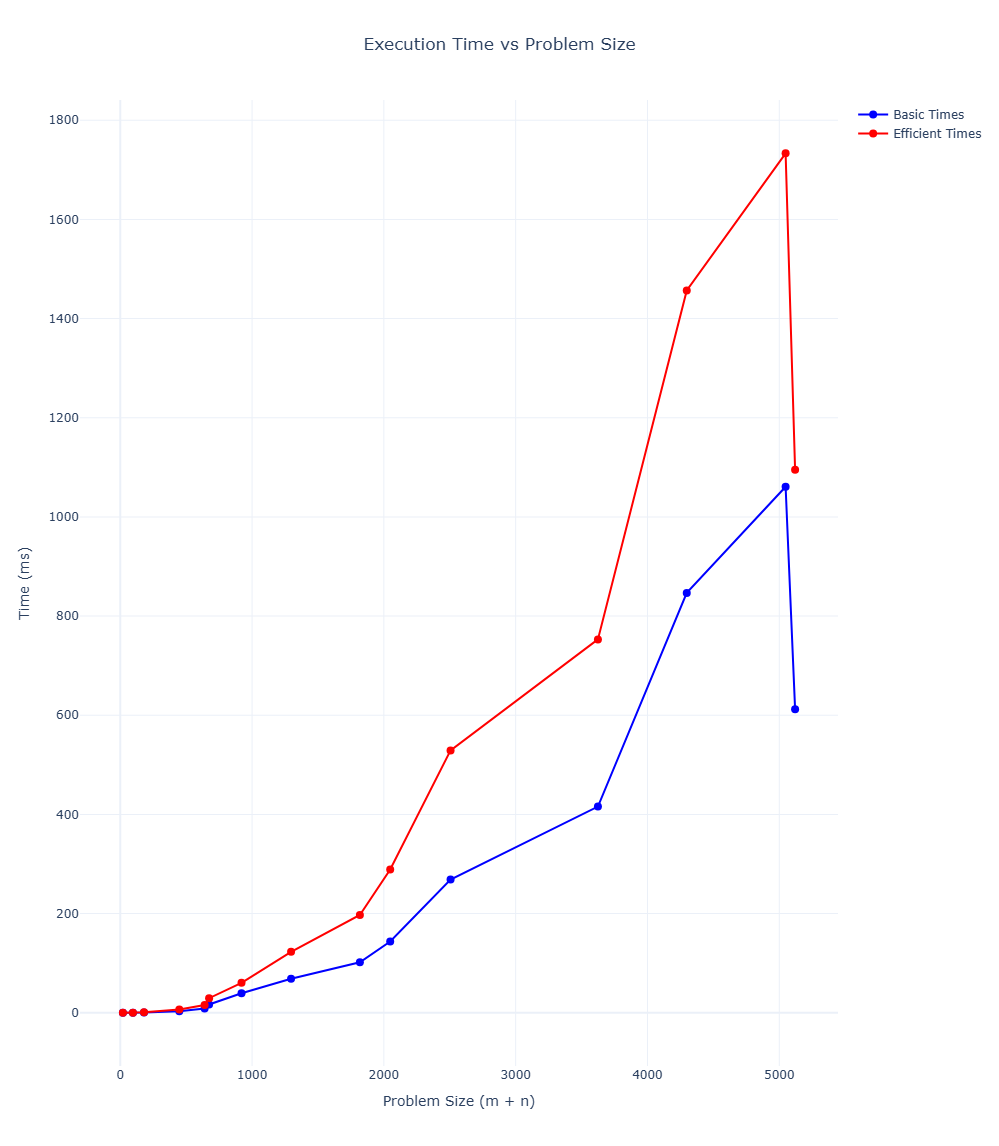
Basic: Polynomial

Efficient: Linear

#### Explanation:

### As the problem size of increases, the basic algorithm’s memory usage rises in a polynomial way because it stores the full DP table; as for the efficient algorithm, its memory stays almost constant with , since it only keeps a few DP rows at a time, corresponding with the Hirschberg method. The volatility of the curve also corresponds to the Python/OS memory allocation rather than the core algorithm itself.

### Graph2 – Time vs Problem Size (M+N)



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

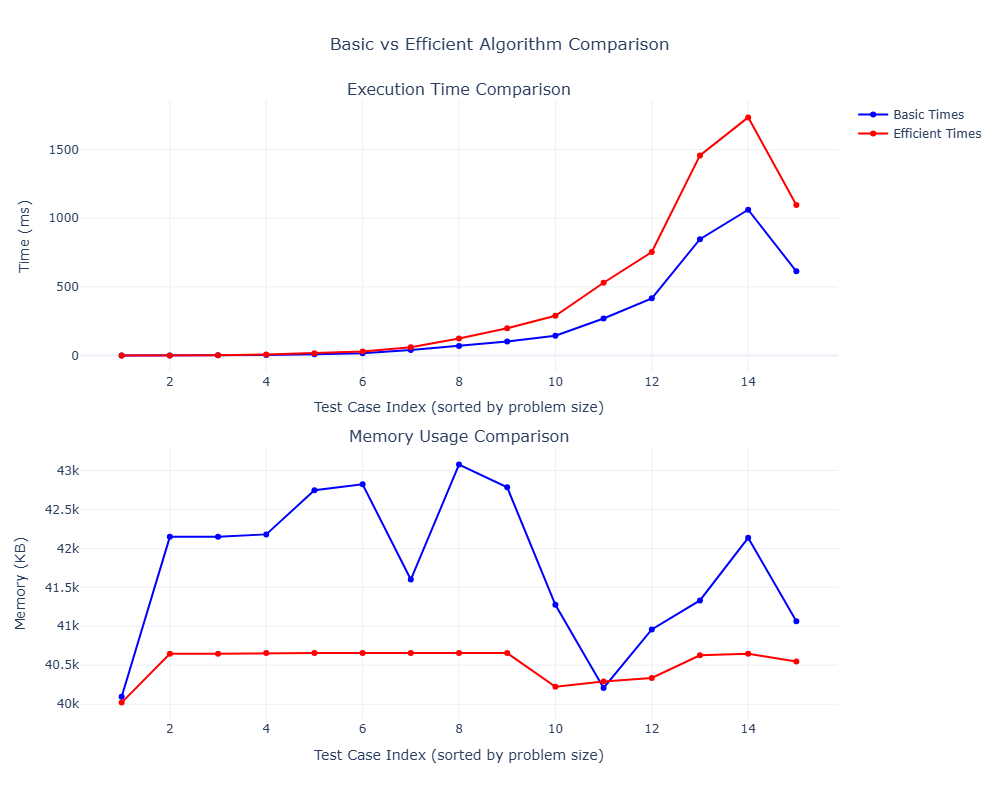
Basic: Polynomial

Efficient: Polynomial

#### Explanation:

As we see the problem size grow, this time we see both of the algorithms increase rapidly in polynomial time, which matches time complexity for sequence alignment. For the much larger inputs, the efficient algorithm is consistently above the basic implementation; it is ~1.5-2x slower dude to the Hirschberg methods multiple DP passes and recursive calls.

## Graph 3 - Head to Head Comparison Time & Memory, Basic Vs Efficient



This graph compares the two algorithms on each test case, which is sorted by problem size. In the top graph, the efficient algorithm is consistently slower than the basic algorithm and the gap grows for larger and larger test cases, which wholly confirms that Hirschberg's extra DP passes add a very noticeable runtime overhead.

For the bottom graph, the basic algorithm uses a bit more memory than the efficient one, this is even more noted for larger inputs, while the efficient memory stays almost flat. This is in part to how the efficient algorithm, (Hirschberg implementation) works with the table, as it never stores the full table as it recomputes DP rows multiple times and only keeps a couple of 1-D arrays and it has a recursion stack at depth; whereas in the basic algorithm, we build the entire DP table , and as m, n get better, the table increases quadratically so therefore, it has to reserve more and more memory.

## 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 | Contribution |
| 5942406612 | Designed and implemented boiler plate code for basic and efficient methods, Implemented the basic method and the respective unit tests. Plotted graphs of basic vs efficient against given parameters. |
| 5480473628 | Initiated unit testing workflow. Implemented part of the efficient method. |
| 5990219480 | Fixed bugs in the Basic implementation, performed extensive testing, created the shell scripts for automated execution, and contributed to writing and organizing the final report. |
| 9306851031 | Organized communication between our team, assisted in brainstorming ideas for implementation, performed complete work on writing the summary report and analyzing our datapoints, graphs, and codebase to accurately state our results. |
| 8868226878 | Implemented the majority of the efficient method and some of the respective unit tests. |