SUMMARY

USC ID/s:

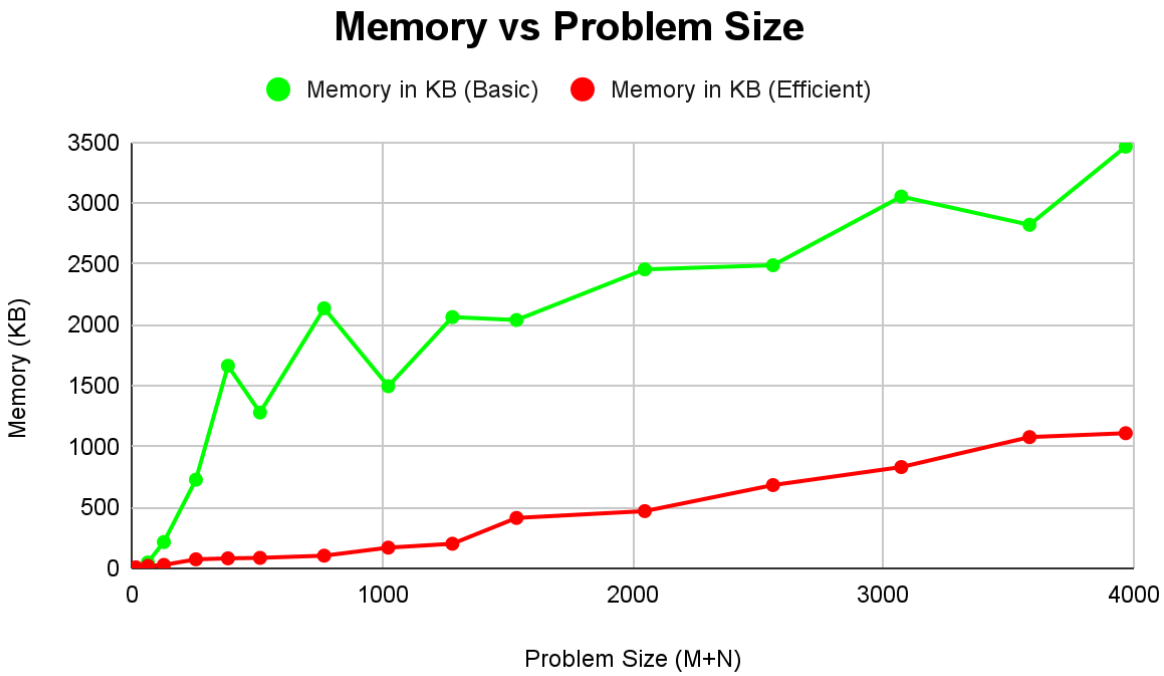
2963653580 (Nicholas Weber)
8940791833 (Joshua Maranan)

Datapoints

Input	Number of Tests	M+N	Average Time in MS (Basic)	Average Time in MS (Efficient)	Average Memory in KB (Basic)	Average Memory in KB (Efficient)
in1.txt	3	16	0.0	0.0	4	8
in2.txt	3	64	0.64	0.43	48	20
in3.txt	3	128	0.99	1.04	216	28
in4.txt	3	256	3.39	5.34	728	74
in5.txt	3	384	7.09	12.53	1664	82
in6.txt	3	512	12.67	21.83	1280	86
in7.txt	3	768	29.97	50.24	2136	104
in8.txt	3	1024	54.98	87.83	1496	170
in9.txt	3	1280	85.87	139.35	2064	202
in10.txt	3	1536	123.03	195.03	2040	414
in11.txt	3	2048	226.42	358.60	2456	470
in12.txt	3	2560	351.80	559.61	2490	684
in13.txt	6	3072	512.73	803.04	3054	832
in14.txt	6	3584	703.95	1110.02	2822	1078
in15.txt	6	3968	859.92	1391.77	3464	1110

Insights

Graph1 – Memory vs Problem Size (M+N)



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

Basic: Linear

Efficient: Linear

Explanation:

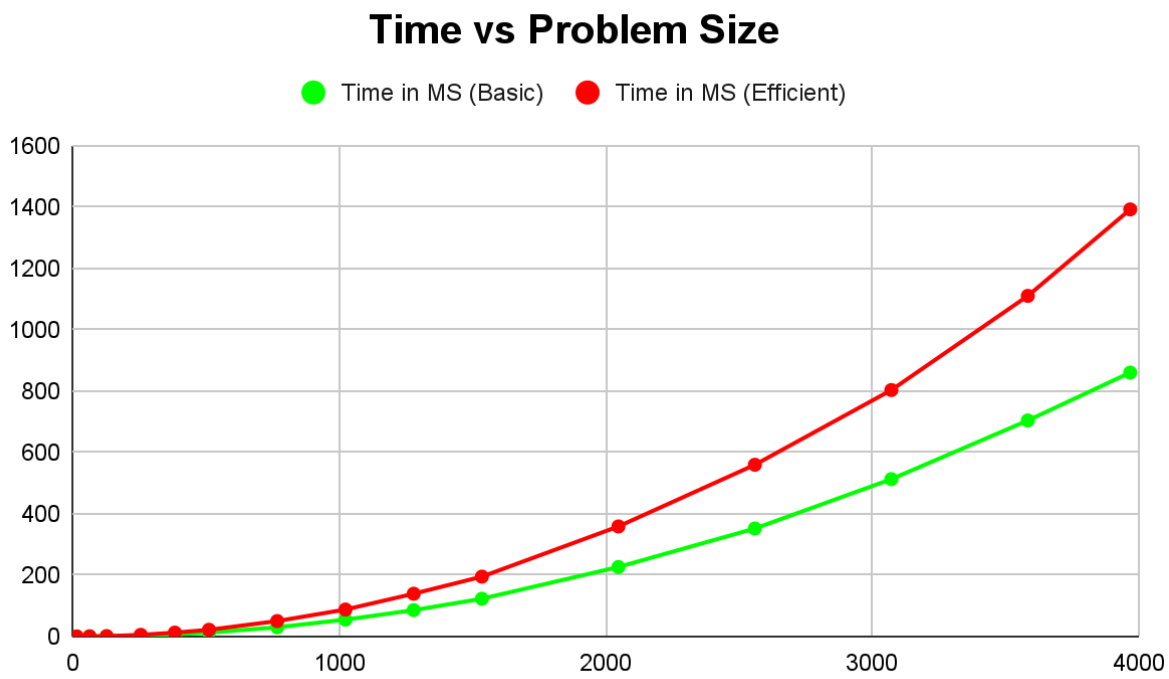
In the graphed data, both the basic and efficient algorithms are increasing linearly with respect to the size of the input. However, we expected the basic algorithm to be of a polynomial nature because it requires O(N*M) memory to compute the dynamic programming table, where N and M are the lengths of the input strings. Behind the scenes this may be influenced and optimized by Python at runtime. Additionally for the efficient algorithm, the overhead cost of additional memory stacks for recursive function calls in Python is much higher than in some other languages, such as C++, which affects this data.

We see that the memory used from the basic algorithm consistently remains higher than that of the efficient algorithm. This is to be expected, as we discussed in class, we should see that basic would consume more memory but take less time than efficient.

One last note is that our y-axis does not include the baseline memory consumption of the runtime. There were two factors which caused considerable noise with measuring our memory consumption in Python:

1. The garbage collection is unpredictable and we cannot be certain that some of the memory is not being cleaned up while the algorithm is running. Therefore, we disabled garbage collection at the start of main and then re-enabled it at the end for both the basic and efficient algorithms.
2. The baseline memory consumption of Python varies by Python version, the machine it is run on, the operating system, and other factors. On Nicholas' machine, it was averaging about 15.5 MB, while on Josh's machine it was averaging nearly 20 MB. It was also not a consistent amount and varied by nearly 1 MB each time we would run the script. To adjust for this, we checked the memory used before and after the algorithms were run and subtracted the former from the latter. This then accounts for the differences in machines and the variability of Python and operating systems.

Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial

Efficient: Polynomial

Explanation:

Both the basic and efficient are increasing polynomially on the size of the input. Compared to the basic algorithm, the efficient algorithm increases in time much more than basic. As discussed in Lecture 8, the tradeoff for this was a decrease in memory used for the efficient algorithm, as we saw in Graph1.

Conclusion:

The data trends we see in the two graphs are in-line with the theoretical complexities we discussed in class, as the efficient algorithm uses less memory but more time than 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”)

<USC ID/s>: <Equal Contribution>

2963653580: Equal Contribution

8940791833: Equal Contribution