SUMMARY

USC ID/s: 7474225277

Datapoints

* linecount refers to the variable dictating the length of the strings, shown in string_gen.py

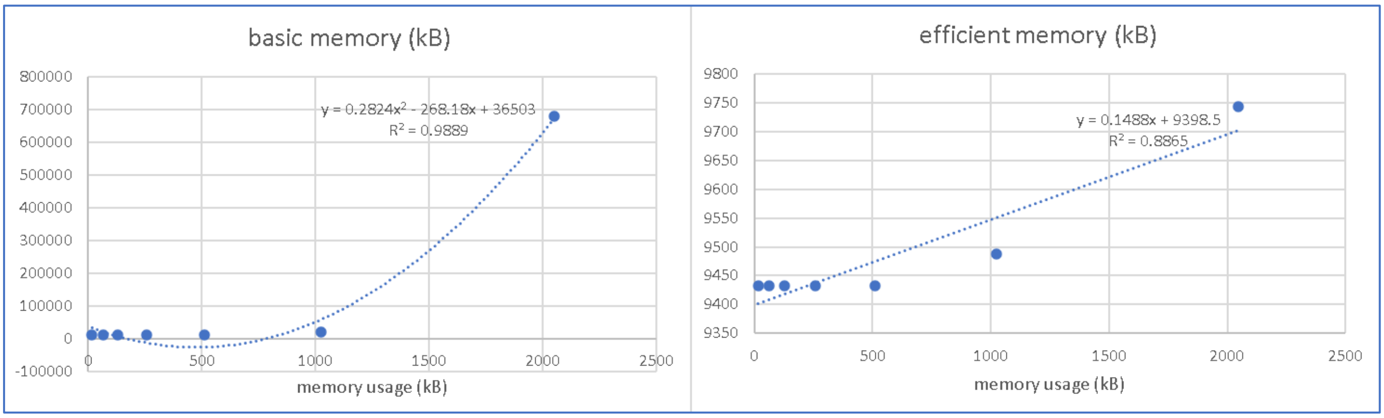
M+N	LineCount*	Time in MS (Basic)	Time in MS (Efficient)	Memory in KB (Basic)	Memory in KB (Efficient)
16	1	0.0923	0.8650	9400	9432**
64	2	1.3983	4.9827	9444	9432
128	3	5.4660	9.8674	9564	9432
256	4	23.4706	30.2048	10076	9432
512	5	74.4240	100.9598	12048	9432
1024	6	354.5356	417.1081	19920	9488
2048	7	1366.3640	1617.3174	51860	9500
2048	input5.txt	22247.8902	26943.1818	678504	9744

** I believe the reason the memory efficient numbers are identical in all cases I doubled the string each time without adding any additional patterns

Insights

Graph1 – Memory vs Problem Size (M+N)

(note the scale difference on the vertical axis between the graphs)



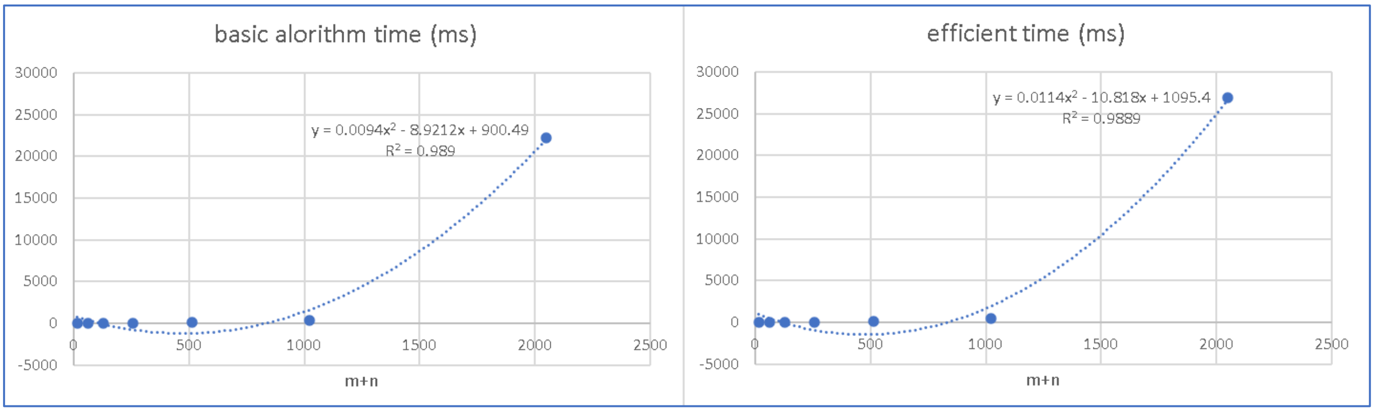
Nature of the Graph (Logarithmic/ Linear/ Exponential) – assume m = n

Basic: quadratic

Efficient: linear

*Explanation: The basic Needleman Wunsch DNA sequence alignment algorithm has a memory requirement of O(m*n). If we consider m = n, then the overall memory required is O(n^2), in order to have an mxn array which is filled (and then retraced) using dynamic programming. The memory efficient version of the algorithm, as discussed in lecture, will use O(m) (or O(min(m,n)) memory, as it uses a 'sliding window' approach to compute the same sequence.*

Graph2 – Time vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Exponential)

Basic: quadratic

Efficient: quadratic

Explanation: Both the 'basic' and 'memory efficient' versions use a similar dynamic programming approach, but the 'memory efficient' version uses a divide and conquer mechanism that requires each subproblem to be computed again. The effect is that there are approximately 2x as many calculations in the efficient version.

Contribution

USC ID/s: 7474225277 - Equal Contribution

Appendix: string_gen.py – how I generated random string of the length in the table

```
import numpy as np
import sys

def main():
    f = open("input.txt","w")
    np.random.seed(5)

    linecount = 7
    arr = np.random.rand(linecount)
    x_base = "TGCAGACT"
    y_base = "CCATTAGC"
    s = x_base
    t = y_base
    for i in range(linecount):
        s = s + "\n" + str(int(1024*arr[i]) % len(s))
        t = t + "\n" + str(int(1024*arr[i]) % len(s))
    both = s + "\n" + t
    f.write(both)
    #print(both)
    f.close()


if __name__ == '__main__':
    main()
```