Programming Assignment #1

Brendan Foley

21 February 2023

1 Description

Over the course of this project, I have moved from using lists (for example, with partitions) in my recursive 'strassen' function to using separate variables. This increased readability and promotes understanding of the underlying code. For many lists, I relied on list comprehensions to create instantiate them (often with 0s for padding) since they are faster than for loops.

In order to allow for my implementation of Strassen's Algorithm to work, I required two helper functions, 'add' and 'subtract' to handle matrix addition and subtraction. For driving the program, I defined a function, 'run', that handled input of data and calling 'get_results' with the proper 'a' and 'b' matrices for multiplication.

2 Performance

The expected performance of Naive Multiplication is $\Theta(n^3)$, while Strassen's algorithm is $O(n^{log_2 7})$. This is not apparent in the practical testing that I performed.

As shown by Figures 1 and 2 and expressed in Table 1 (as well as Table A1), my implementation of Strassen's algorithm takes much longer than even Naive Multiplication. This is most likely due to the fact that I included the timing of padding arrays to reach an order of 2^n so that Strassen's algorithm can take place. This is the cause of sudden jumps in run time just after powers of 2, i.e. 32 and 64. Since matrices can be multiplied even when not in a power of two order, I believe that it is important to include timing the prep work required of the algorithm in run time.

Runtime	Method	Size	Runtime	Method
0.0006469	naive	2	4.97E-05	strassen
0.00179	naive	3	8.03E-05	strassen
0.0011955	naive	4	8.10E-05	strassen
0.0080763	naive	5	8.69E-05	strassen
0.0089839	naive	6	0.0001411	strassen
4.56E-05	naive	10	0.0008566	strassen
8.57E-05	naive	11	0.0093866	strassen
0.0001455	naive	12	0.0065812	strassen
0.0393527	naive	42	2.3106942	strassen
0.0700723	naive	50	2.306723	strassen
0.5480216	naive	100	16.1304557	strassen

Table 1: A representative subset of tests of both algorithms. Full table in Appendix Tabla A1

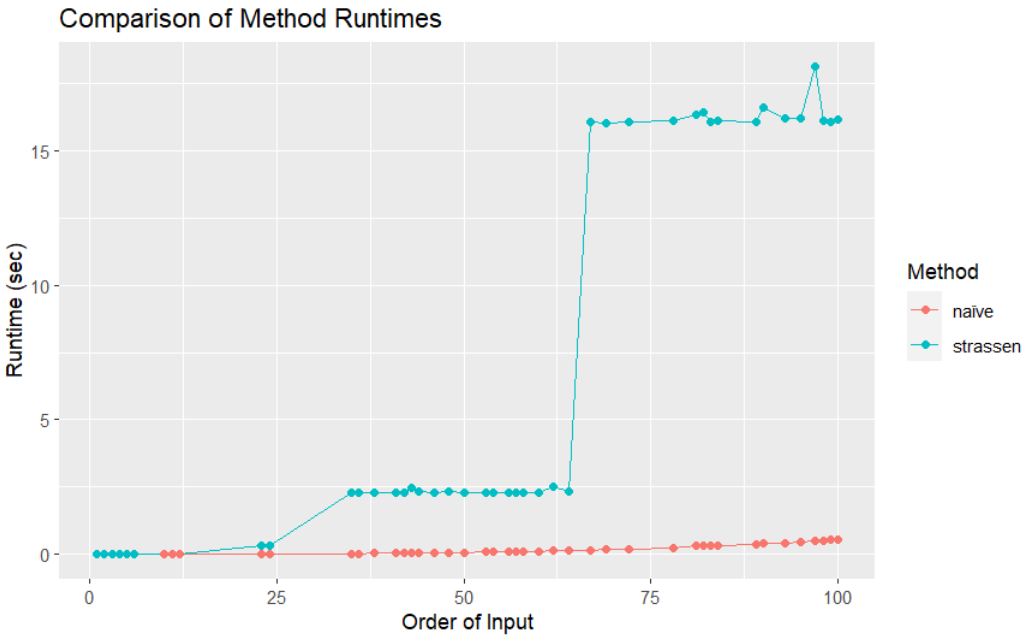

Figure 1: Average Runtime of Naive and Strassen's algorithm across datasets

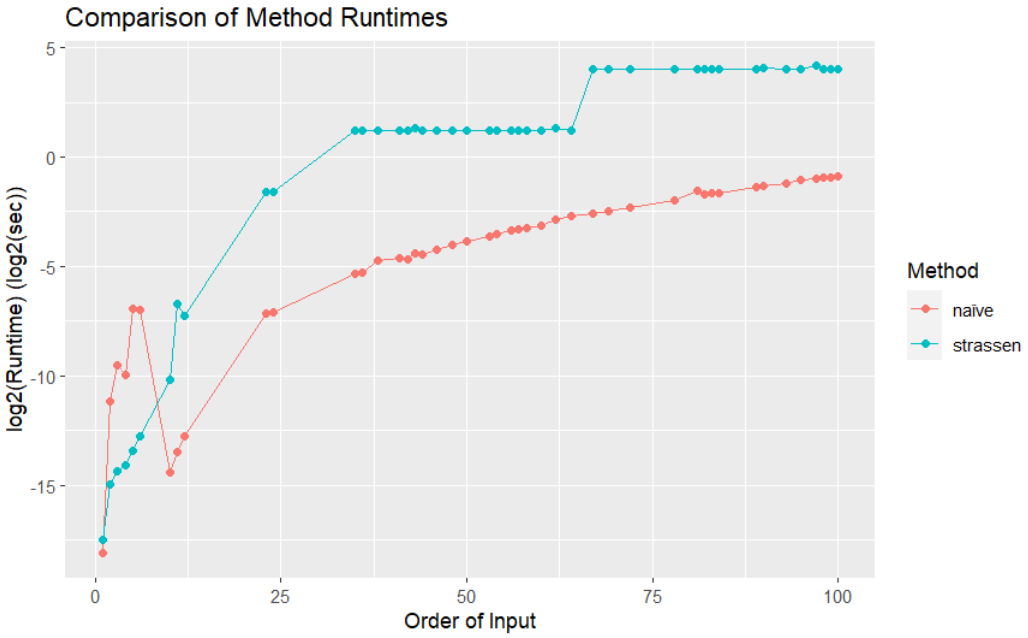

Figure 2: log_2(Average Runtime) of Naive and Strassen's algorithm across datasets

3 Application

As described by the assignment, matrix multiplication is prevalent throughout Bioinformatics. While we most likely won't be developing multiplication algorithms more efficient than ones already created (that can stay with Computer Scientists and Mathematicians), understanding proper matrix usage is crucial in our field.

One of the first things I learned about Bioinformatics was alignment matrices BLOSUM and PAM. While these are possibly the most famous matrices in Bioinformatics, matrix usage is a daily task since so much can be represented by them. While dataframes are becoming increasing popular (just as JSON and NoSQL are competing against CSV and SQL), understanding efficient matrix usage will cause an all around improvement in the quality and efficiency of any code written. Lastly, even though this assignment does not explicitly call for it, understanding how to vectorize array usage will take any code written to a step above in terms of raw speed.

4 Improvements

There are several things that I wish to improve upon next time from what I saw happen this time. First, I would like to vectorize more computations. This will speed up the program in pure runtime, even if the big-O remains the same. Although, 'good' vectorization can only happen with NumPy and Pandas, both of which have efficient matrix multiplication functions.

It was surprising to me to see that theoretical and actual runtime can be so vastly different. However, as stated in the previous section, this may be largely due to the fact that I included formatting of arrays for Strassen's algorithm in its runtime. I do believe this to be acceptable when comparing runtimes, as Naive multiplication does not require the data be in a power of 2, which is a large advantage. However, in the future I think that it would be best to measure the algorithm on its own and not include the padding of arrays.

5 Appendix

5.1 Table A1

Table of runtimes for each dataset I used, by method. The beginning 30 entries cover edge cases (i.e. all zeros, all negative, random 1s, etc.), while the later runs (30 and onward) are used to fill out runtime for graphing.

Runtime	Method	Size
0.0006469	naive	2
0.00179	naive	3
0.0011955	naive	4
0.0080763	naive	5
0.0089839	naive	6
0.0002114	naive	2
0.0014253	naive	3
0.0008969	naive	3
0.0009424	naive	4
0.0065921	naive	5
3.50E-06	naive	1

Continued on next page

Table 2 – continued from previous page

Runtime	Method	Size
0.0008566	naive	4
0.0093866	naive	5
0.0065812	naive	6
4.97E-05	strassen	2
8.03E-05	strassen	3
8.10E-05	strassen	4
8.69E-05	strassen	5
0.0001411	strassen	6
1.22E-05	strassen	2
3.97E-05	strassen	3
2.41E-05	strassen	3
4.65E-05	strassen	4
0.0001023	strassen	5
5.30E-06	strassen	1
4.56E-05	strassen	4
8.57E-05	strassen	5
0.0001455	strassen	6
0.0008566	strassen	10
0.0093866	strassen	11
0.0065812	strassen	12
4.56E-05	naive	10
8.57E-05	naive	11
0.0001455	naive	12
0.3469005	naive	81
0.0072799	naive	24
0.5480216	naive	100
0.5160927	naive	98
0.0393527	naive	42
0.0700723	naive	50
0.1008894	naive	57
0.0805409	naive	53
0.1286227	naive	62
0.0967567	naive	56
0.1155086	naive	60
0.5301643	naive	99
0.025945	naive	36
0.3867466	naive	89
0.0541721	naive	46
0.0878413	naive	54
0.2570404	naive	78
0.479197	naive	95
0.1022337	naive	57
0.0993857	naive	57
0.0069598	naive	23
0.0452074	naive	44
0.4365897	naive	93

Continued on next page

Table 2 – continued from previous page

Runtime	Method	Size
0.100985	naive	57
0.1067746	naive	58
0.1416816	naive	64
0.3948795	naive	90
0.0228817	naive	35
0.0623884	naive	48
0.1777033	naive	69
0.0263493	naive	35
0.3176175	naive	83
0.2021834	naive	72
0.0400708	naive	41
0.5010966	naive	97
0.1412419	naive	62
0.1717905	naive	64
0.3051213	naive	82
0.3232422	naive	84
0.4014923	naive	90
0.0480602	naive	43
0.0377103	naive	38
0.2059162	naive	72
0.141904	naive	64
0.097441	naive	57
0.1681909	naive	67
0.0830891	naive	53
0.0843632	naive	54
0.5252949	naive	98
0.0536395	naive	46
16.3289556	strassen	81
0.3272828	strassen	24
16.1304557	strassen	100
16.0422449	strassen	98
2.3106942	strassen	42
2.306723	strassen	50
2.3348955	strassen	57
2.2835982	strassen	53
2.379724	strassen	62
2.3140696	strassen	56
2.3134568	strassen	60
16.0719485	strassen	99
2.2722836	strassen	36
16.0666479	strassen	89
2.3059541	strassen	46
2.3012864	strassen	54
16.0852365	strassen	78
16.1964047	strassen	95
2.3273342	strassen	57

Continued on next page

Table 2 – continued from previous page

Runtime	Method	Size
2.2925598	strassen	57
0.3296772	strassen	23
2.3278334	strassen	44
16.1791138	strassen	93
2.2957221	strassen	57
2.3087256	strassen	58
2.2809429	strassen	64
16.1363507	strassen	90
2.2858065	strassen	35
2.3199178	strassen	48
15.9985957	strassen	69
2.2698485	strassen	35
16.0516996	strassen	83
16.0965185	strassen	72
2.2791098	strassen	41
18.113282	strassen	97
2.6220808	strassen	62
2.4232926	strassen	64
16.4330923	strassen	82
16.0907649	strassen	84
17.0803216	strassen	90
2.4585901	strassen	43
2.2986594	strassen	38
16.0496338	strassen	72
2.3182864	strassen	64
2.2980159	strassen	57
16.0794609	strassen	67
2.3054109	strassen	53
2.3039027	strassen	54
16.1372436	strassen	98
2.3087157	strassen	46