605.420 Foundations of Algorithms

Anam Ahsan

Project 3 Analysis

Due Date: November 30th, 2022

Dated Turned In: November 30th, 2022

Description and justification of your data structures, implementation choices, and design decisions

For this project, an array was used as the primary data structure. The arrays held strings and substrings that were manipulated into characters or integers.

The strings were extracted from the input files and parsed to identify the sequences; this occurs in 'main'. The substrings were manipulated into characters or integers to allows the methods 'findLCS' and 'printLCS' to iterate through, identify the lowest common substring (LCS), then print it. An array was the best data structure for this task, as it is easy to iterate through, and allows for random access. A unique array was made for each method, for the 'main' method, an array called 'seq[ ][ ]' was made to hold sequences and their respective lengths to allow the appropriate comparisons to take place of all the unique pairs in the input. In the 'findLCS' method, the 'LCS[ ][ ]' array identified the LCS and its length. Finally, although present in 'findLCS' the 'bLCS[ ][ ]' array is applied in the 'printLCS' method. This array identifies the LCS and matches it to the array that holds one of the sequences in a pair, to allow the LCS to be printed.

This programmed functioned through passing specific rows of the array or the entire array into the next method. Only the 'printLCS' method uses recursion to call itself with modified parameters. The use of arrays and subdividing the task into three methods allowed the 'main' method to remain uncrowded.

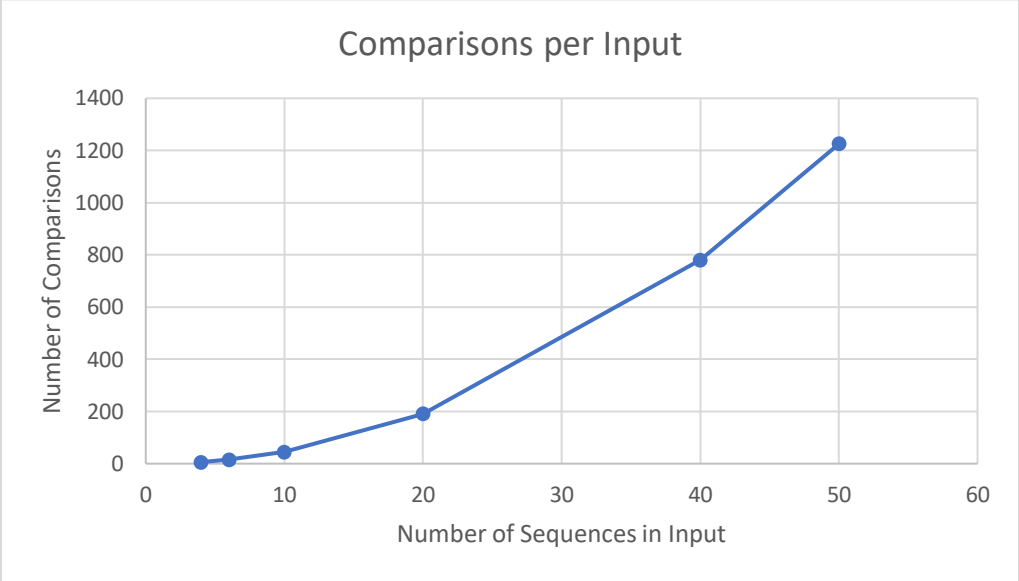
Efficiency with respect to both time and space

In the 'main' method, there are two while-statements. The first is to take the input file and store sequences and sequence length into an array from each line 's'. The second while-statement is to iterate through the array of sequences 's' to push unique pairs 'r', to the 'findLCS' method. To get unique pairs with all sequences, 's' factorial will need to be applied, the following equations gives the total number of unique pair combinations $\frac{s!}{r!*(s-r)!}$. If we look at Table 1, we can see if we apply the equation to the number of sequences 's', where r = 2 since we are looking for pairs, we will get the same number of comparisons per input as the table. In example, in input 2 where s=6 and r=2, using equation $\frac{6!}{2!*(6-2)!}$, we get 15 which can be observed in Table 1. Together, the cost of the 'main' method is $\Theta(s + \frac{s!}{r!*(s-r)!})$.

Input	Number of Sequences	Comparisons per Input
1	4	6
2	6	15
3	10	45
4	20	190
5	40	780
6	50	1225

Table 1 – Shows the number of sequences in an input file and the number of unique pair comparisons

We can see in Graph 1 that as the number of sequences increases, the number of comparisons per input increases, due to it being a factorial time complexity, the efficiency will decrease significantly as the input size grows larger.



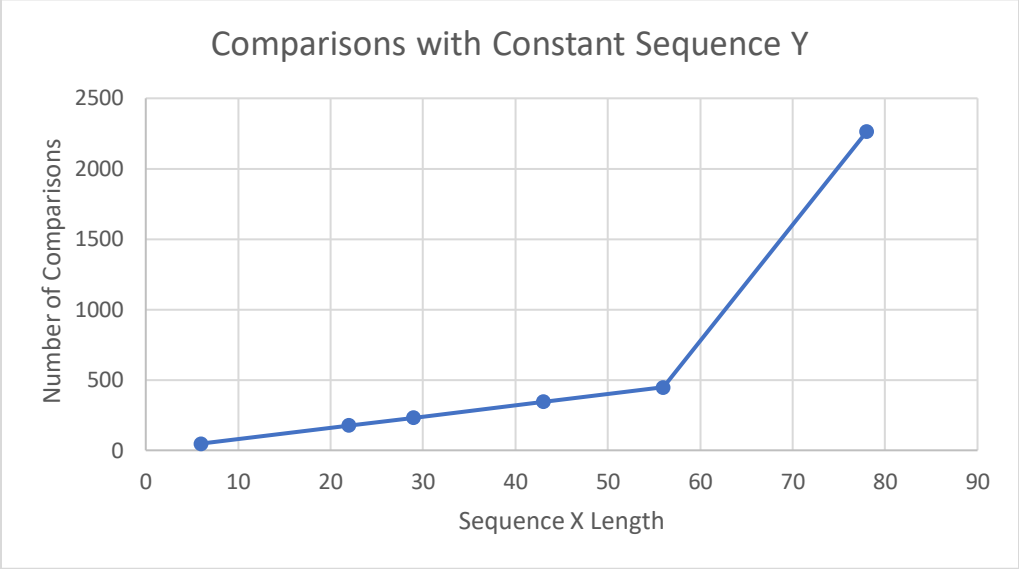
Graph 1 – Shows the number of sequences in an input file and the number of unique pair comparisons

The theoretical efficiency of the 'findLCS' method is Θ(mn) as it iterates through both sequences to fill the LCS[ ][ ] array. This is observed as well when we look at the numbers of comparisons in relation to the lengths of the two sequences in a pair. In Table 2, for pair 1, sequence 'X' length equals 6 and sequence 'Y' length equals 8, the number of comparisons is 48, which is 6 * 8. This can be observed for all pairs 1-9.

	Pair	Sequence X Length	Sequence Y Length	Comparisons
1	S29 and S11	6	8	48
2	S5 and S11	22	8	176
3	S1 and S11	29	8	232
4	S14 and S11	43	8	344
5	S34 and S11	56	8	448
6	S44 and S11	78	8	2262
7	S34 and S14	56	43	2408
8	S44 and S14	78	43	3354
9	S34 and S44	56	78	4368

Table 2 – Shows the number of individual comparisons between a pair, extracted from RESULT_6

Using pairs 1-6 of Table 2, Graph 2 was created to demonstrate asymptotic cost, we can see that if one sequence is kept constant, and compare it against sequences of increasing size, that the number of comparisons increases dramatically. Between S5 and S14, there is a difference of 21 base pairs, and the difference between the number of comparisons (when each compared to S11), the number of comparisons doubles. In comparison between S34 and S44, there is a difference of 22 base pairs, and the difference between the number of comparisons (when each compared to S11), the number of comparisons more than quadruples. If we compare the larger sequences with each other, we get even larger number of comparisons (see Table 2). From our observations we can conclude that as sequence size increases the cost of the function will dramatically increase as well.



Graph 2 – Shows number of individual comparisons between pairs 1-6, extracted from RESULT_6

The theoretical efficiency of 'printLCS' is Θ(m + n). This is observed as well as we iterate through both the bLCS arrays and the array of one sequence using recursive calls. Therefore, it is also be impacted when sequence lengths become larger.

What you learned / What you might do differently next time

This was an interesting project, and I was able to learn a lot when running test cases where some of the drawbacks and limitations of the program became apparent. One of the first points to come up is that a pair may have more than one LCS. If we wanted to output all the LCS of a given pair, the efficiency of this program would change significantly to accommodate all the extra runs to find those LCSs and print them. This is especially interesting to consider if there is a particularly large number of sequences to evaluate where all the sequences are very long. Such a scenario would be very time consuming and would require a much higher-powered computer to run the program. I was able to observe through this project one of our module 6 practice homework problems which asked us to show that a polynomial number of calls to a polynomial-time subroutine may result in exponential-time algorithm.

One thing that I might do differently next time is have the 'printLCS' method not be a recursive call. It was difficult to print the LCS when there were multiple LCS to print. As it is now, we call 'printLCS' for every pair and then it recursively calls itself too. As the input size and sequences grow larger, this would be very time consuming.

Applicability to Bioinformatics

The obvious applicability to bioinformatics is that we compare sequences to identify homology between species. Being able to find large amounts of similarity between two species can help us understand how closely related they may be in the evolutionary tree. Another helpful application is in understanding gene function. If we have a sequence of a gene, but we do not know its function, but we find a homologous sequence of it in a gene of a different species that we do know the function of, we can infer the potential function of the unknown gene. One can also use the LCS to build a genome if one has only partial DNA sequence.