Jonathan Nguyen, jnguye36@jhu.edu
Module 7 Lab Analysis Dynamic Programming

This implementation uses two matrices and dynamic programming to find the longest common substring (LCS), and is based off of the pseudocode found in CLRS Chapter 15.4. The overall logic is the same, and I've made some modifications to fit as needed, for example replacing the symbolic arrows with a String representation of the direction, in order to find the LCS. For data structures, I've created three separate classes, one as the main driver, another to hold the two matrices for organizational purposes, and one for an enhancement to the lab.

Because we're doing a pairwise comparison, I thought that the best way to handle this is to first read in all the sequences into an ArrayList, and then loop through all possible combinations. To be space efficient, I created the LCS_Holder class to hold a given test case, and also have that object be able to handle all output, that way the main driver only needs to worry about which sequences to handle at a given time, the Align function will do the dynamic programming portion, and the LCS_Holder will be a convenient way to store and output data. This ended up working well, as the main driver is just formatting and no difficult logic is put in place there, this also opens up the LCS_Holder for use in other applications that might need to work with multiple matrices. The main function, DynamicProgramming.Align takes a LCS_Holder to keep a matrix by reference, IUPAC object to use special matching logic, and two strings to compare. From there it builds the matrix from the top down, incrementing the matches by one whenever it finds one, and also maintains a history of the path taken at each location. This process uses two for loops, and takes O(n*m) time, where n,m are the length of the two strings. If they are roughly the same size, we can generalize to O(n^2). The second major function is the LCS_Holder.printLCS method that recursively calls from the end to the start, to follow the paths history, and find the LCS, the output is printed when the recursion closes, so this method climbs down and back up. The recursion will continue to call until both counters, i and j, are at 0, so it ends up taking O(n+m) time, where n,m are the lengths of the two strings. We can multiply that value by 2 for when it comes back out of the recursion, but it still reduces down to O(n+m).

This lab was a good learning experience in that I used two forms of breaking a problem down into smaller problems, one was through dynamic programming, and building a table up to find a difficult value, and the other was finding the LCS after a table is built, through the use of recursion. In terms of planning, I think I did well compared to previous labs for the given requirements, but for what I'd like to do differently is make the classes more flexible for use in other scenarios, and having the code be more reusable.

As an enhancement I added two features. The first one is through finding the LCS, I also display the location of the gaps. For example, when comparing AC to AGCT, the correct LCS is AC, but I show A-C- as well to indicate where the gaps were located. In terms of bioinformatics, this is where we could introduce a gap penalty for a match, and potentially an extension penalty, where longer gaps are considered worse matches. Another enhancement I added was the use of IUPAC codes for special matching. By default, G will match G, but if the 'S' flag for strong bonds is turned on, then G can match C, allowing more flexibility for bioinformatics if we were searching for sequences that had particular characteristics.

Another potential bioinformatics application is to extend this beyond sequence matching. For example, I've demonstrated that we can perform special matches, but these matching algorithms can also be used for comparing proteins given the correct set of codes, or more logic can be build into the matching logic, to account for long repeats, such as long stretches of CG, which has been associated with high gene activity. It could also be used to search for common motifs, and used for comparisons across different genes, and different species.