Project 5 Report:

I have the following functions in my code and their Big O:

1) Align
    a. This function runs the code for the unrestricted and banded implementation by calling the appropriate functions listed below
    b. Unrestricted Big-O = O(m*n) + O(m*n) + O(C) + O(log(m*n) == O(m*n)
    c. Banded Big-O = O(k*n) + O(k*n) + O(C) + O(log(k*n) == O(k*n)
    d. For details about how I got the previous Big-O calculations, look at the following functions with their Big-O's
2) initialize_matrix_unrestricted
    a. This function creates a list inside a list and initializes each row and column to have a tuple with two values representing the cost and the direction it took to get to that point in the matrix. The lengths of the rows and columns were determined by the length of two strings (m & n) plus 1 to add a buffer. I then initialized every point to be (0.0, 0). I then initialized the first row and column with the appropriate values based on the indel penalties.
    b. The big O = 2*O(m*n) == O(m*n) where "m" is the length of one string and "n" is the length of the other. This is because we are iterating over every data point twice in our matrix.
3) initialize_matrix_banded
    a. In this function, I used a dictionary with the key as a tuple of coordinates and the values were a tuple of the cost and direction. I then would use the banded width value to determine how far I would need to fill out a table. I did this by taking the absolute value of the difference for the row and column index and compared that with the banded width. If it was smaller than or equal, I would add that point to my table. I then initialized the first row and column that were within the banded width based on the indel penalties.
    b. The big O = O(k*n) where k represents the number of points restricted by the banded width and n represented by the data points visited
4) NeedleMan_Wunsch_unrestricted
    a. This takes the full matrix and goes point by point and calls the compare_diagonal_top_left_unrestricted function to calculate the cells cost and directions for the unrestricted, uninitialized cells.
    b. The big O = 2*O(m*n) == O(m*n) where "m" is the length of one string and "n" is the length of the other. This is because we are iterating over every data point twice in our matrix.

5) NeedleMan_Wunsch_banded
    a. This takes the full matrix and goes point by point and calls the compare_diagonal_top_left_banded function to calculate the cells cost and directions for the banded, uninitialized cells.
    b. **The big O = O(k*n) where k represents the number of points restricted by the banded width and n represented by the data points visited**
6) compare_diagonal_top_left_unrestricted
    a. This function does the actual arithmetic for calculating the cost for each cell, it looks at the cell to the left, up, and diagonal of each cell. It will assign the appropriate cost and direction for each cell.
    b. Big O = constant because it will always look at three cells, a constant number of data points every time
7) compare_diagonal_top_left_banded
    a. This function does the actual arithmetic for calculating the cost for each cell, it looks at the cell to the left, up, and diagonal of each cell. This differs from the unrestricted version by checking the cell index to see if it is in the table before performing the calculation.
    b. Big O = constant. At worst, it is the same Big O as the unrestricted version
8) traceback_unrestricted
    a. This function takes the cost from the last point in the matrix and it follows the direction values to recreate the appropriate aligned sequence for each of the string sequences.
    b. Big O = O (log(m*n)). I don't have a great reason for why it's log, but I know its less than O(m*n) but it is not constant time, so it's pigeonholed into log time
9) traceback_banded
    a. This function takes the cost from the last point in the table and it follows the direction values to recreate the appropriate aligned sequence for each of the string sequences. This also has a check whether or a solution was found or not. The unrestricted makes the assumption that a solution was found.
    b. Big O = O (log(k*n)). I don't have a great reason for why it's log, but I know its less than O(k*n) but it is not constant time, so it's pigeonholed into log time