# CSC 110 – Lab 11
# Bioinformatics Algorithms

Names: ________________________________

***Introduction***
The purpose of this lab is to understand how certain bioinformatics algorithms work and to practice implementing parts of these algorithms in Python.

***Exercises***

1) In class we looked at the simple dot plot algorithm. We discussed how it can be difficult to see patterns in the plot when we are matching each character individually. The sliding windows version of the dot plot allows us to visualize patterns in groups of characters.

   In this lab you will implement the sliding windows algorithm for computing a dot plot.

   Let's start by writing a function that will compute the percentage of matches found between a pair of given sequences. Note, you cannot assume that the sequences are of equal length. The function skeleton will look like this:

```
def computePct(seq1, seq2):

    # Fill in the code to compute the percentage match
    # between the sequences

    return pct
```

Instructor Initials: ______________________

2) In order to implement the sliding windows, we will have to access substrings from a Python string. Python has a way to do this. If you want to take a substring from position 2 to up to but not including position 5 in a string, you can use:

```
myString[2:5]
```

Try this out. In the Python shell, enter the following:

```
myString = "UNIVERSITY"
```

Now enter:

```
myString[2:5]
```

What does it return?

What command would you use if you wanted to return the match percentage between the first five characters of two given strings?

Try this on these two strings:

```
str1 = "AACTCGTGAGTCT"
str2 = "ACTTGCGGGCTA"
```

What is the result of this command for these strings?

Instructor Initials: ______________________________

3) Now let’s write a function to compute the dot plot. We will use a small example to demonstrate how this will work:

```
        0  1  2  3  4  5  6
        A  G  G  T  A  A  G
   0 A
   1 A
   2 G
   3 T
   4 A
```

If we assume a window size of 3 and a threshold of 0.5, then we put a dot in the dot plot in position (0,0) if we have better than a 50% match between the first three characters in the first sequence and the first three characters in the second sequence. In this example, we would be comparing:

```
AGG
AAG
```

Since two of the three characters match, we would have a dot in cell (0,0).

When we consider cell (0,1), we slide the window on the top sequence over by one so are comparing:

```
GGT
AAG
```

In this comparison, we have no matches, so there is no dot placed in cell (0,1).

Here is a start to the Python function that computes the dot plot:

```
def computeDotPlot(str1, str2, winsize, threshold):
    # Initialize a list that will hold all of the rows
    # of the dot plot

    dot_plot = []
    # Initialize the first row of the dot plot
    row = []

    # Loop through all of the rows (characters in the
    # sequence on the y-axis)
    for i in range(len(str2)):

        # Loop through all of the colums (characters in the
        # sequence on the x-axis)
        for j in range(len(str1)):
          # Fill in the code to determine if we should add
          # a "*" to the plot row, or add a " "

     # Add the row to the dot plot
     dot_plot = dot_plot + [row]
     row = []
 return dot_plot
```

You can also find the skeleton code for this program here:

http://www.cs.uri.edu/~cingiser/csc110/labs/dot_plot_sliding_skeleton.py

Fill in the code for the two functions you have written, and then test the program using the following sequences with several different window sizes and different threshold values:

```
AAGGTAGCCTAACGTCCACTTTACCC
AGTAAGGTACCTACCTCAACTTCA
```

What do you observe about how the plot changes when you make the window larger?

What do you observer about how the plot changes when you make the threshold higher?

Instructor Initials: ______________________

***Challenge Problem***

In class we briefly discussed the Needleman_Wunsch dynamic programming algorithm to find the optimal alignment for a pair of sequences. Although we did not consider the details of the algorithm, you can find python code to implement it here:

http://www.cs.uri.edu/~cingiser/csc110/labs/dynamic_programming.py

For this challenge problem, look at this code and see if you can analyze how much work (order of magnitude) is done to find this optimal alignment.

You can run the program to see what it does. You can put print statements in the code if it helps to analyze it.

If you would like to see an animation of how the algorithm works, you can find one here:

http://baba.sourceforge.net/

Choose Simple DP for the easiest algorithm to follow.

Instructor Initials: ______________________________