Code:

# O(100n\*m) = O(n\*m)  
 def align\_all( self, sequences, banded, align\_length ):  
 results = []  
 #This loop runs 10 times  
 for i in range(len(sequences)):  
 jresults = []  
 sequ\_i = sequences[i]  
 sequ\_i\_len = len(sequ\_i)  
 #This loop runs 10 times as well  
 for j in range(0, len(sequences)):  
 sequ\_j = sequences[j]  
 sequ\_j\_len = len(sequ\_j)  
 if(i == j):  
 s = {'align\_cost': max(-3\*align\_length,-3\*sequ\_i\_len),  
 'seqi\_first100': 'Self Comparison',  
 'seqj\_first100': 'Self Comparison'}  
 elif((i == 0 and j != 1) or (i == 1 and j != 0)):  
 s = {'align\_cost': float('inf'),  
 'seqi\_first100': 'No Alignment Possible',  
 'seqj\_first100': 'No Alignment Possible'}  
 elif(i > j):  
 s = {'align\_cost': 0,  
 'seqi\_first100': 'Dummy',  
 'seqj\_first100': 'Dummy' }  
 else:  
 # 2 Matricies of size n\*m are made here O(2n\*m) = O(n\*m)  
 if(align\_length > sequ\_i\_len):  
 if(align\_length > sequ\_j\_len):  
 dist\_matrix = [[0 for col in range(sequ\_i\_len + 1)]

for row in range(sequ\_j\_len + 1)]  
 path\_matrix = [[("") for col in range(sequ\_i\_len + 1)]

for row in range(sequ\_j\_len + 1)]  
 else:  
 dist\_matrix = [[0 for col in range(align\_length + 1)]

for row in range(align\_length + 1)]  
 path\_matrix = [[("") for col in range(align\_length + 1)]

for row in range(align\_length + 1)]  
  
 for m in range(len(dist\_matrix[0])):  
 if(m > 3 and banded):  
 dist\_matrix[0][m] = float('inf')  
 else:  
 dist\_matrix[0][m] = m \* 5  
 path\_matrix[0][m] = 'r'  
  
 for m in range(len(dist\_matrix)):  
 if (m > 3 and banded):  
 dist\_matrix[m][0] = float('inf')  
 else:  
 dist\_matrix[m][0] = m \* 5  
 path\_matrix[m][0] = 'a'  
  
 path\_matrix[0][0] = ""

#For loop loops through each of the sequences for each of the sequences. At most O(n\*m)  
 for k in range(1, len(dist\_matrix)):  
 # This code puts the bound on the rows for the constraint 3  
 if (banded):  
 if ((k - 3) > 0):  
 l\_start = k - 3  
 else:  
 l\_start = 0  
 if ((k + 4) < len(dist\_matrix[0])):  
 l\_end = k + 4  
 else:  
 l\_end = len(dist\_matrix[0])  
 else:  
 l\_start = 1  
 l\_end = len(dist\_matrix[0])  
  
 for l in range(l\_start, l\_end):  
 right\_of = dist\_matrix[k - 1][l] + 5  
 above = dist\_matrix[k][l - 1] + 5  
 diagonal = dist\_matrix[k - 1][l - 1]

+ self.diff(sequ\_i, sequ\_j, k-1, l-1)

mininum = min(right\_of, above, diagonal)

dist\_matrix[k][l] = mininum  
 if(mininum == right\_of):  
 path\_matrix[k][l] = ("r")  
 elif(mininum == above):  
 path\_matrix[k][l] = ( "a")  
 elif(mininum == diagonal):  
 path\_matrix[k][l] = ("d")  
  
 # Function is O(n\*m)  
 alignment = self.extract\_align(path\_matrix, sequ\_i, sequ\_j)  
  
 s = {'align\_cost': dist\_matrix[-1][-1],  
 'seqi\_first100': alignment[0],  
 'seqj\_first100': alignment[1] }  
  
 jresults.append(s)  
  
 results.append(jresults)  
 return results  
  
 #O(3)  
 def diff(self, sequ\_i, sequ\_j, k, l):  
 if(sequ\_i[l] == sequ\_j[k]):  
 return -3  
 else:  
 return 1  
  
 #O(n\*m)  
 def extract\_align(self, path\_matrix, sequ\_i, sequ\_j):  
 return\_i = ""  
 return\_j = ""  
 path = ""  
 j = -1  
 k = -1

#get path from the bottom corner at most repeated the longer sequence (n or m)  
 while(path\_matrix[j][k] != ''):  
 path += path\_matrix[j][k]  
 if(path\_matrix[j][k] == 'd'):  
 j = j - 1  
 k = k - 1  
 elif(path\_matrix[j][k] == 'a'):  
 j = j - 1  
 elif(path\_matrix[j][k] == 'r'):  
 k = k - 1  
  
 path = path[::-1] # Reverse the path  
 i\_spot = 0  
 j\_spot = 0  
  
 #Gets the correct alignment for the two sequences at most longest sequence (n or m)  
 for i in range(len(path)):  
 if (path[i] == 'a'):  
 return\_i += "-"  
 else:  
 return\_i += sequ\_i[i\_spot]  
 i\_spot += 1  
  
 if (path[i] == 'r'):  
 return\_j += "-"  
 else:  
 return\_j += sequ\_j[j\_spot]  
 j\_spot += 1  
  
 return (return\_i, return\_j)

Complexity:

Time:

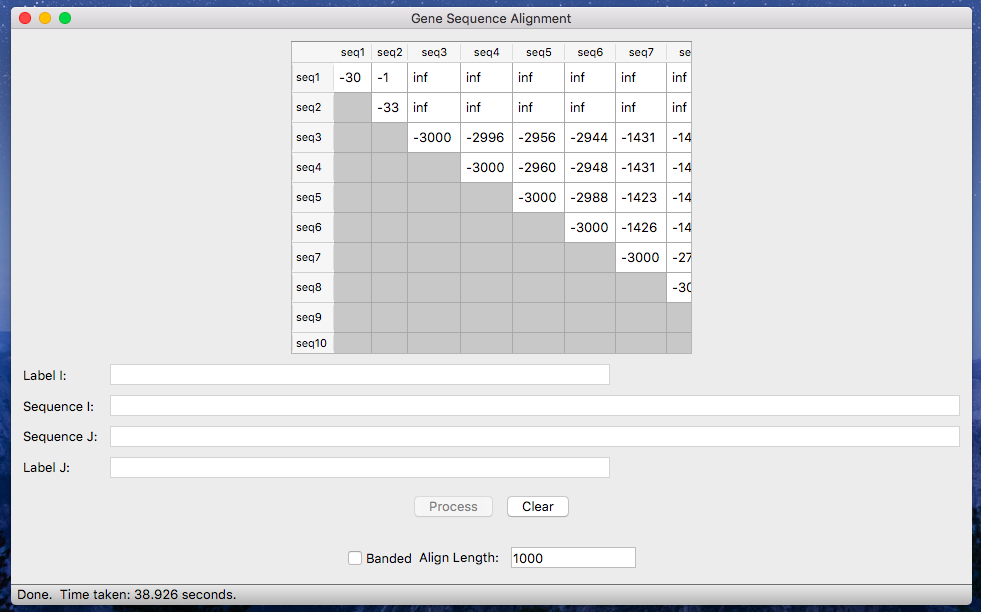
Time complexity is O(n\*m) for each sequence. There are 10 sequences run at 10 times each, so it comes out to O(100\*n\*m) = O(n\*m)

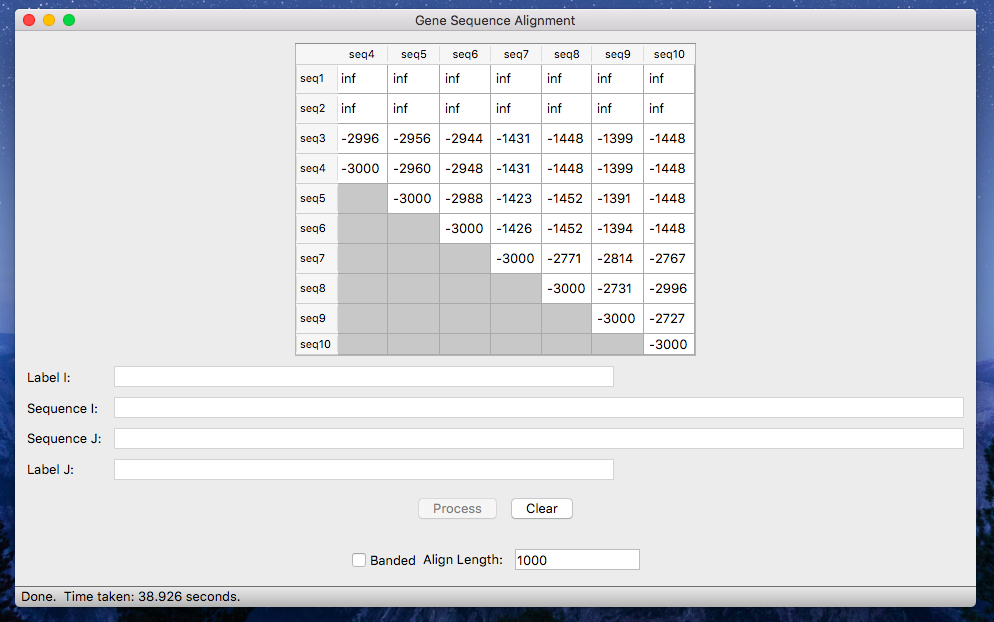
Space:

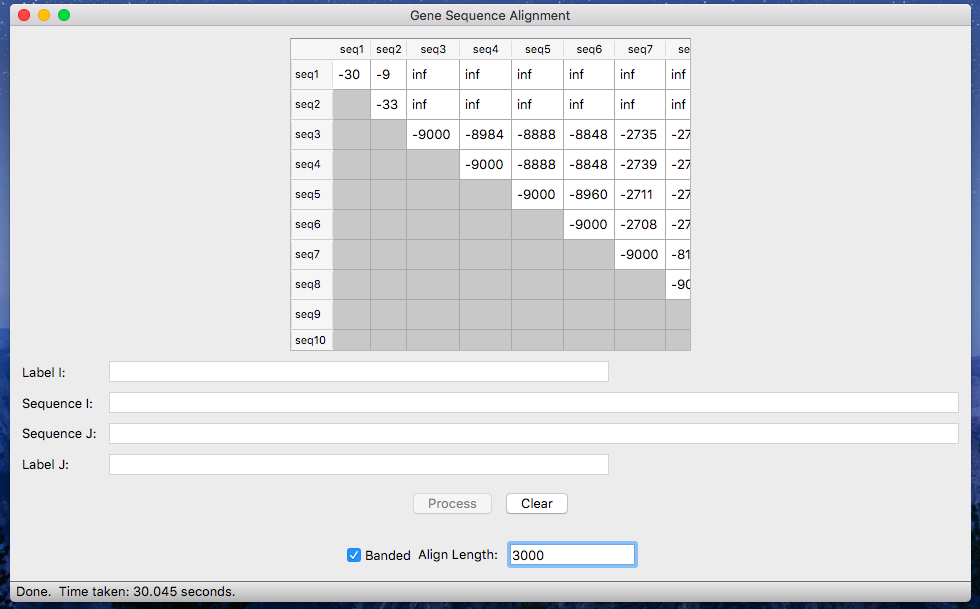
There are 2 matrices of size n\*m made making the space O(2\*n\*m) = O(n\*m)

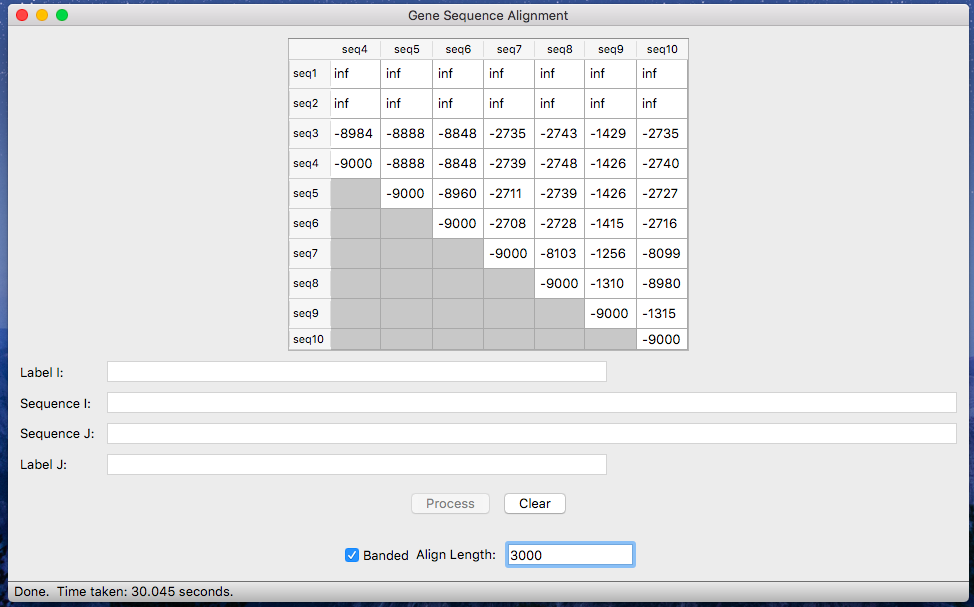
Alignment Extraction:

For the alignment extraction I store a matrix the same size as the 2 sequences require. I store a string letter in each spot depending on if the lowest cost came from the diagonal, the right, or from above. I then go back through the matrix starting at the bottom corner (-1, -1) and moving diagonal, right, or up depending on the letter and add that letter to the path. I move up the matrix until I hit the top left corner and then stop. The path is then reversed, and the alignments are made. To make the alignments I put a dash in the output string if it comes from above and if not just add the letter of the sequence to the output string and then increment the spot in the sequence. I do the same for the second sequence but if it comes from the left (moves right) then I put a dash in the output string.

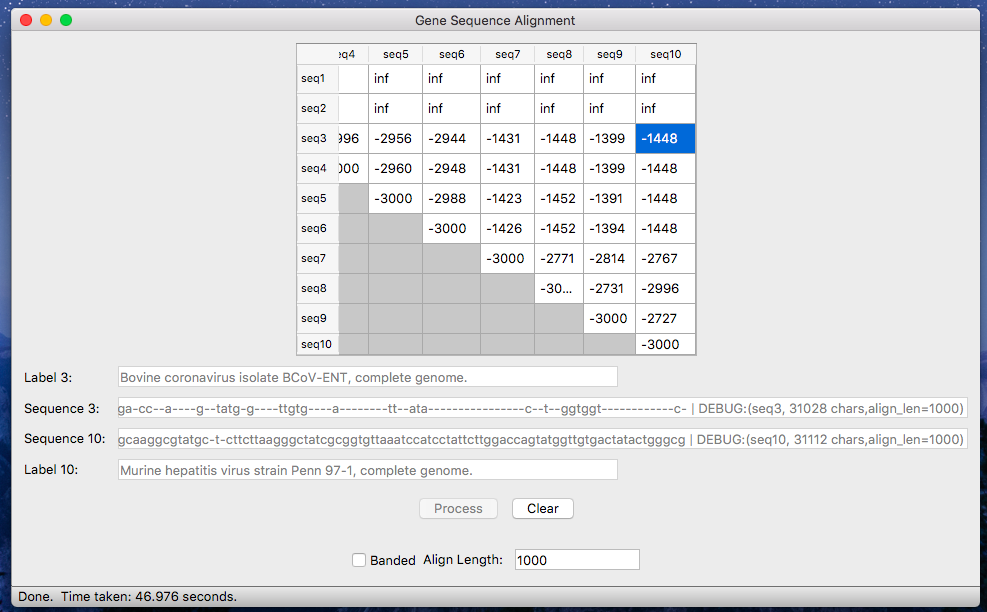
Sequencing for un-banded at 1000:



Sequencing for banded at 3000:



Sequences 3 and 10, un-banded:



Sequences 3 and 10, banded:

