Advanced Algorithms (CS 5512)
Project #4
Submitted by: Shishir Khanal
Submitted to: Dr Paul Bodily

I. Explain the time and space complexity of your algorithm by showing and summing up the complexity of each subsection of your code.

A. Your analysis should show that your unrestricted algorithm is at most *O*(*nm*) time and space.

In the unrestricted algorithm, the largest time complexity occurs at the third nested for loop. The two loop iterates for altogether for n*m times where n is the length of first subseaquence and m is the length of second subsequence. Hence the time complexity is **O(nm)**

However, the space complexity for this algorithm is not exactly O(nm). Two arrays: one storing the cost and another storing the backpointers are created. This makes the space complexity 2O(nm). The equivalent space complexity still becomes **O(nm)**.

B. Your analysis should show that your banded algorithm is at most *O*(*kn*) time and space.

In the banded algorithm, the nested loop runs the whole length of sequence 1. However, the second loop only runs at most 7 times when i = 3. Hence, the time complexity of the banded algorithm is **O(kn)**.

The costmatrix and backpointer matrix have dimensions 7xn . Hence, the equivalent space complexity still becomes **O(kn)**.

II. Write a paragraph that explains how your alignment extraction algorithm works, including the backtrace

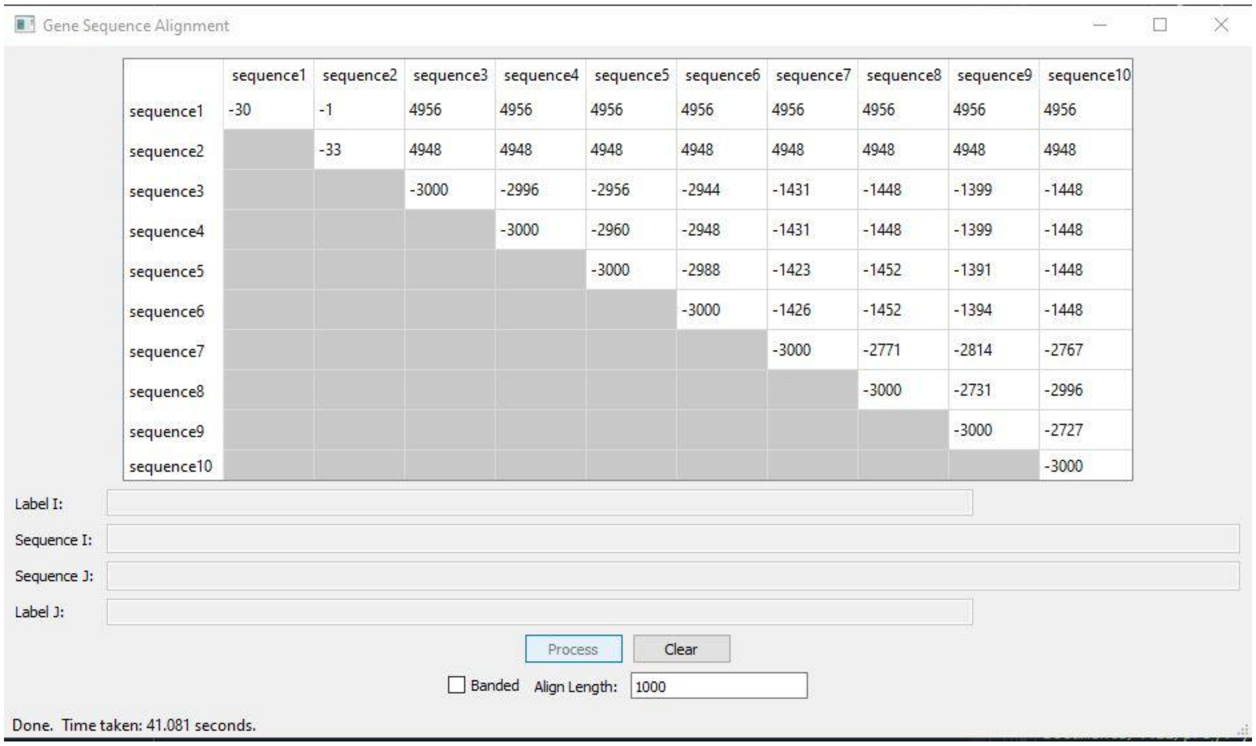
The alignment cost evaluation algorithm starts by creating a costmatrix that is 1 row and 1 column size bigger than the required text size for the strings A and B to allow the insertions and deletions at the left side of the string. This loop iterates through the row and column element to compare the similarity of the two texts by evaluating a costmatrix. The final element of this cost matrix is the cost of alignment of the particular sequences of texts in comparison. If the two characters are the same, they will be matched with the cost of -3, if they are not the same, then they need to be substituted with the cost of 1. Also if the texts need to be inserted or deleted they will be done at a cost of 5 plus the cost till the particular block. Now, for a current step, the algorithm looks at the left upward

diagonal, left and up block and evaluates the characters corresponding into these elements in the respective string and iteratively evaluates the cost.
The text alignment algorithm uses the backpointer matrix previously computed during the evaluation of the alignment cost. It starts at the final element of the backpointer array matrix. If the backpointer matrix consists of the string “match”, then it traverses upward diagonally and copies the corresponding elements for the strings A and B to new array newA and newB. Similarly, if the element has the string “insert”, then the pointer traverses upward and the newA is appended a character ‘-’ whereas the newB is fed the current element of B. Finally, if the element has string “delete”, the newA is appended the current value of A and B is appended a character ‘-’ representing a deletion of the element.

III. **Include a “results” section showing both a screen-shot of your 10x10 score matrix for the unrestricted algorithm with align length *k* = 1000 and a screen-shot of your 10x10 score matrix for the banded algorithm with align length *k* = 3000.**

Results:

a. **unrestricted algorithm with align length *k* = 1000**



	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-1	4956	4956	4956	4956	4956	4956	4956	4956
sequence2		-33	4948	4948	4948	4948	4948	4948	4948	4948
sequence3			-3000	-2996	-2956	-2944	-1431	-1448	-1399	-1448
sequence4				-3000	-2960	-2948	-1431	-1448	-1399	-1448
sequence5					-3000	-2988	-1423	-1452	-1391	-1448
sequence6						-3000	-1426	-1452	-1394	-1448
sequence7							-3000	-2771	-2814	-2767
sequence8								-3000	-2731	-2996
sequence9									-3000	-2727
sequence10										-3000

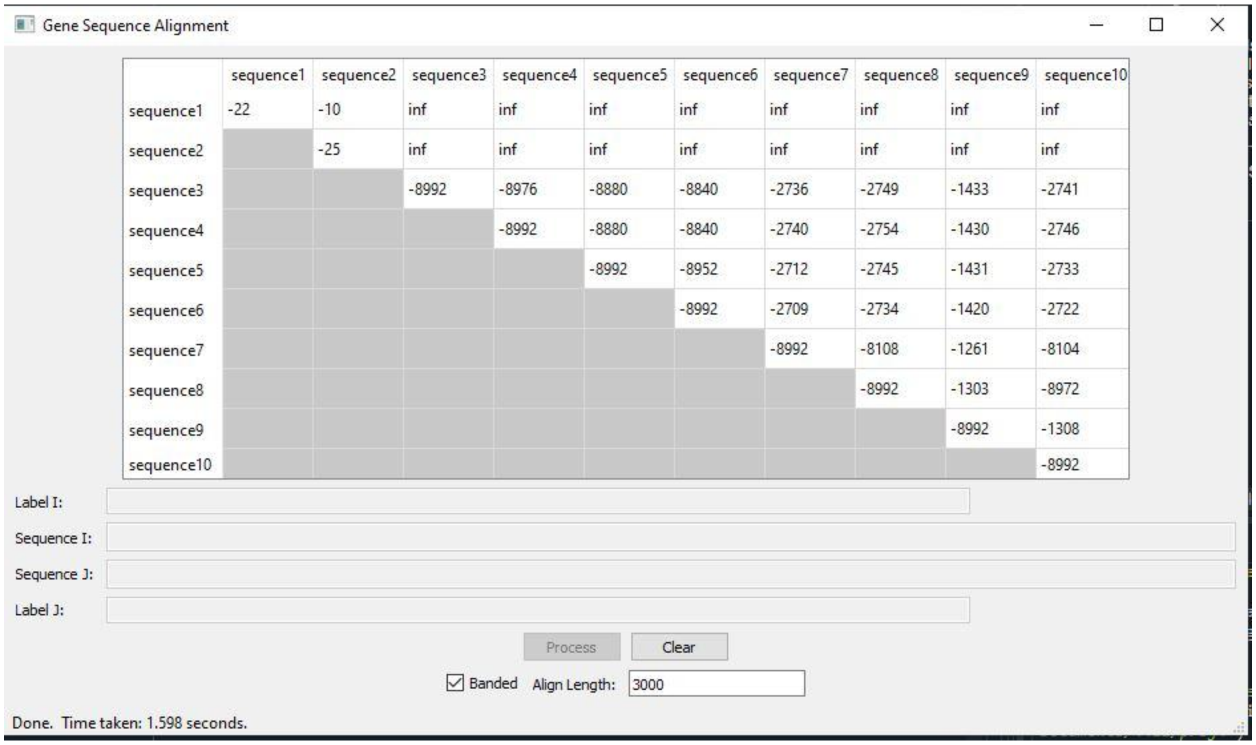
Label I:
Sequence I:
Sequence J:
Label J:

☐ Banded Align Length: 1000

Done. Time taken: 41.081 seconds.

Alignment does not display in the text bars (reason unknown)

b. **banded algorithm with align length *k* = 3000**



Gene Sequence Alignment

	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-22	-10	inf	inf	inf	inf	inf	inf	inf	inf
sequence2		-25	inf	inf	inf	inf	inf	inf	inf	inf
sequence3			-8992	-8976	-8880	-8840	-2736	-2749	-1433	-2741
sequence4				-8992	-8880	-8840	-2740	-2754	-1430	-2746
sequence5					-8992	-8952	-2712	-2745	-1431	-2733
sequence6						-8992	-2709	-2734	-1420	-2722
sequence7							-8992	-8108	-1261	-8104
sequence8								-8992	-1303	-8972
sequence9									-8992	-1308
sequence10										-8992

Label I:
Sequence I:
Sequence J:
Label J:

☑ Banded Align Length: 3000

Done. Time taken: 1.598 seconds.

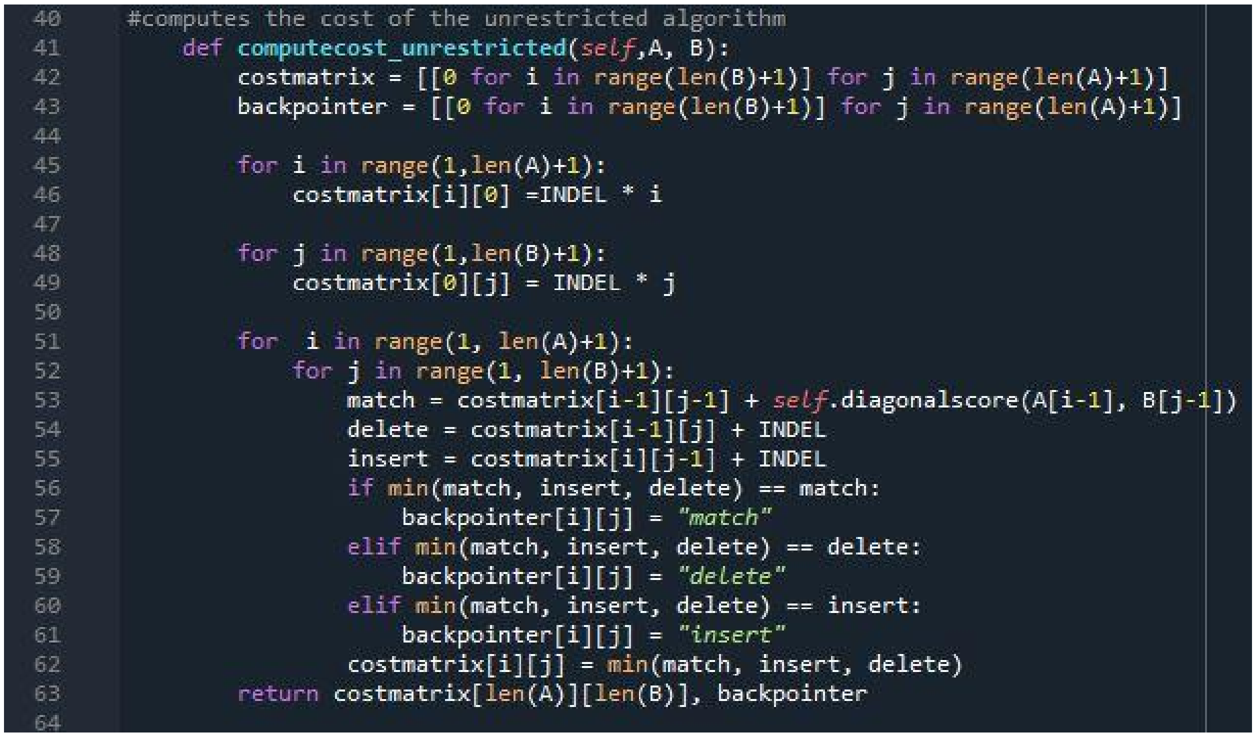
Alignment does not display in the text bars. This is due to a bug in text alignment algorithm due to incorrect indexing.

IV. [10 points] Include in the "results" section the extracted alignment for the first 100 characters of sequences #3 and #10 (counting from 1), computed using the unrestricted algorithm with *k* = 1000. Display the sequences in a side-by-side fashion in such a way that matches, substitutions, and insertions/deletions are clearly discernible as shown above in the To Do section. Also include the extracted alignment for the same pair of sequences when computed using the banded algorithm and *k* = 3000.

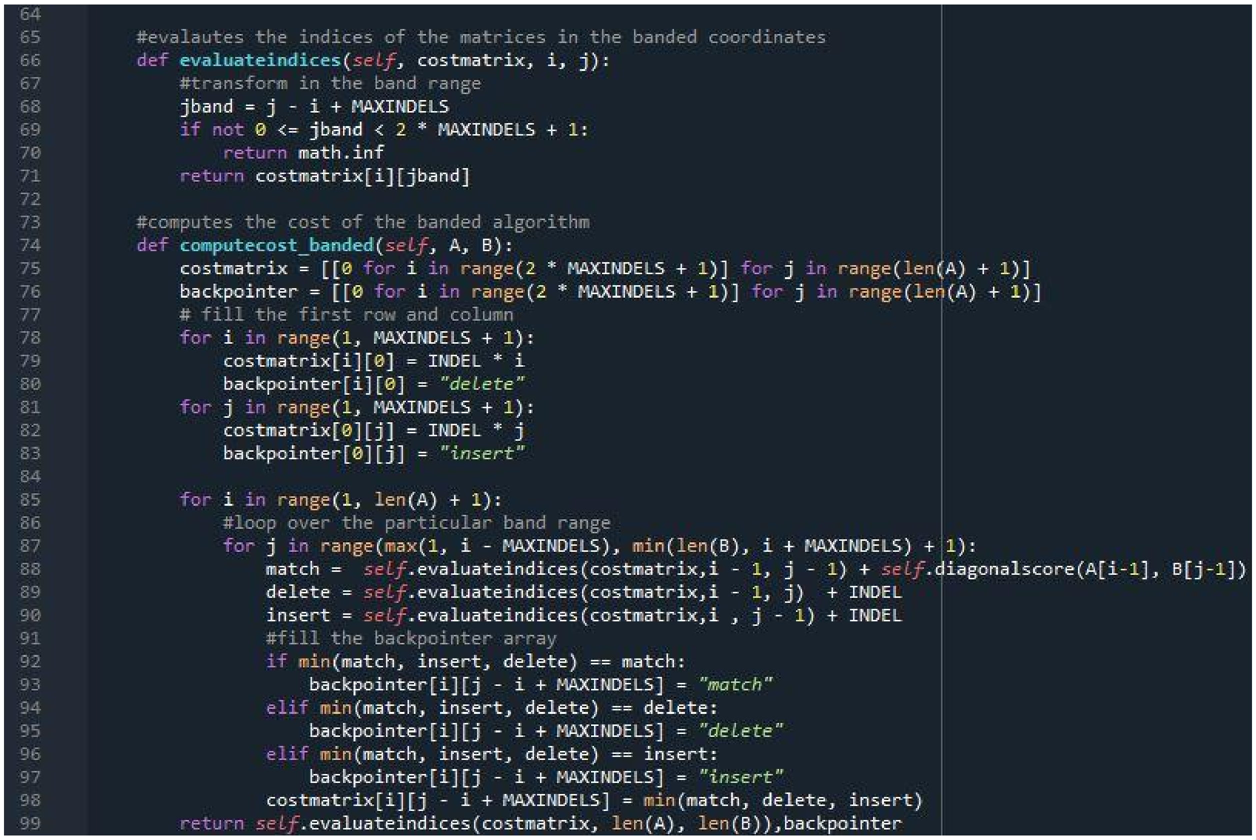
For sequence 3 and 10, the expected result would be:

----gat-g-ga--ggcgatttgcgtgcgtgcatcccgcttcactgatctcttgttagatcttttcataatctaaactttataaaaacatccactccctgtagt
ataagagtgattggcgtccgtacgtaccctttctactctcaaactcttgttagtttaaatctaatctaaactttataaacggcacttcctgtgtgtccat

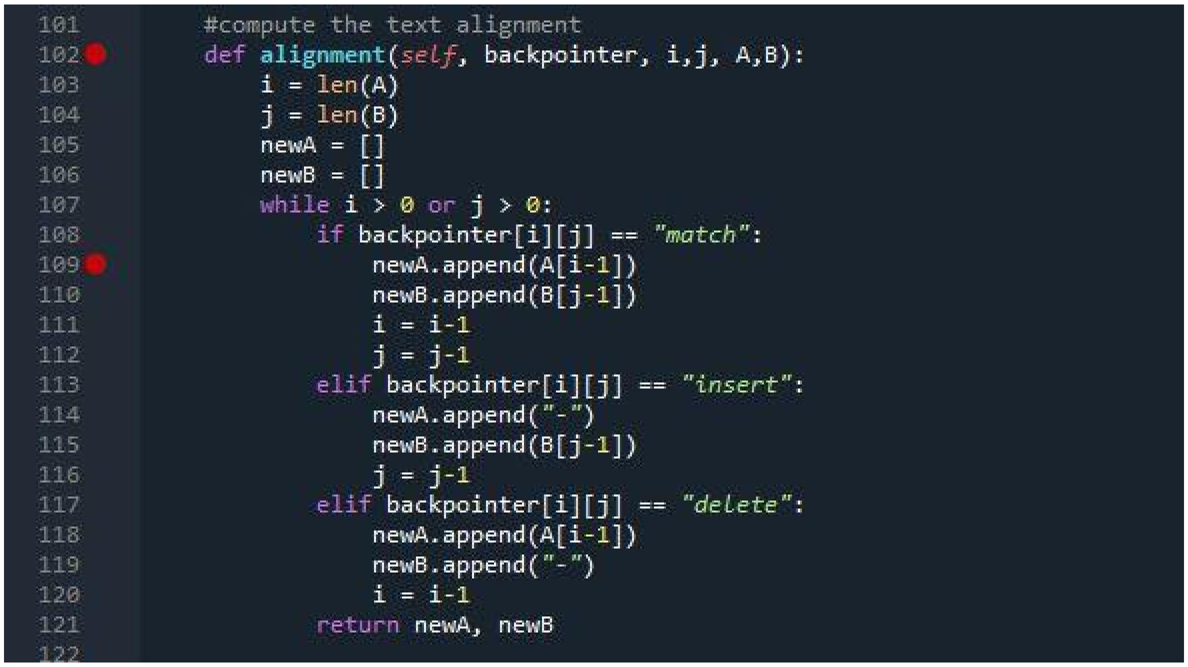
V. **[30 points] Attach your commented source code for both your unrestricted and banded algorithms.**



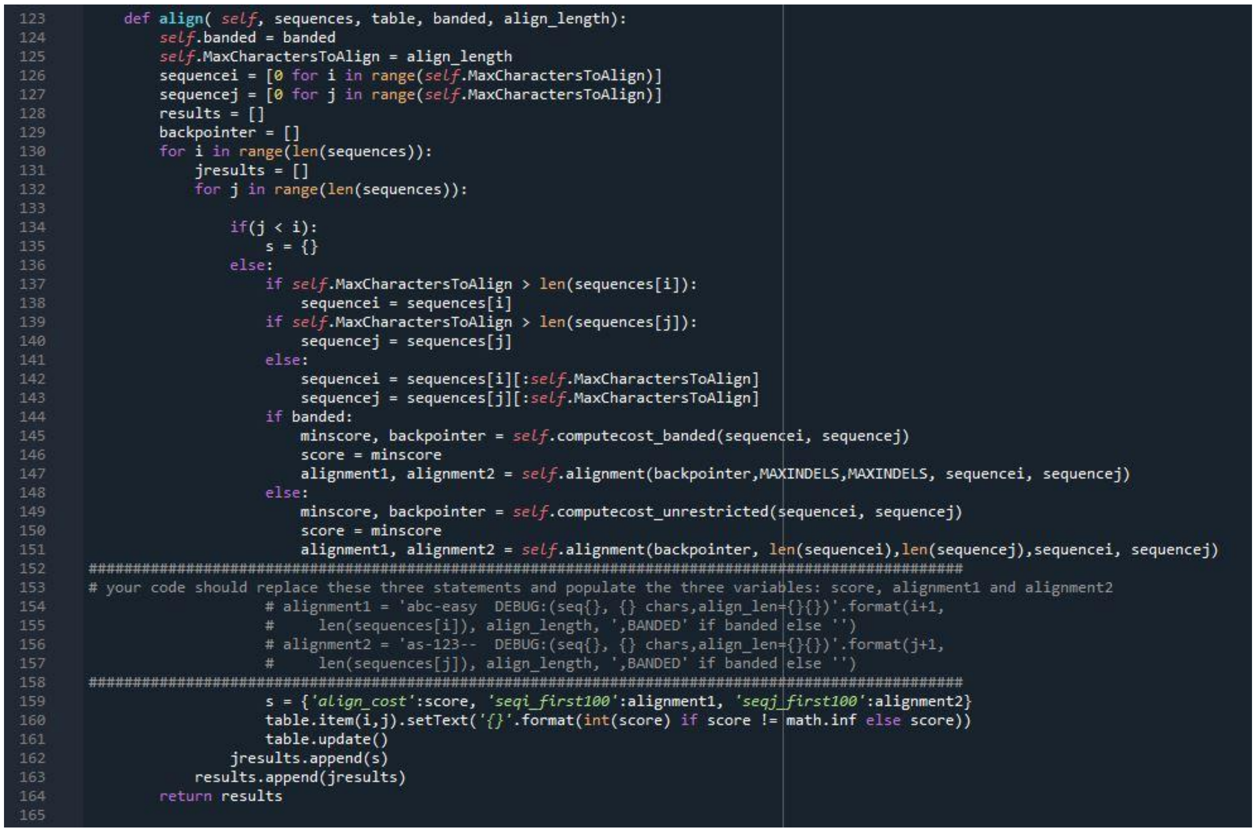
```python
#computes the cost of the unrestricted algorithm
    def computecost_unrestricted(self,A, B):
        costmatrix = [[0 for i in range(len(B)+1)] for j in range(len(A)+1)]
        backpointer = [[0 for i in range(len(B)+1)] for j in range(len(A)+1)]

        for i in range(1,len(A)+1):
            costmatrix[i][0] =INDEL * i

        for j in range(1,len(B)+1):
            costmatrix[0][j] = INDEL * j

        for  i in range(1, len(A)+1):
            for j in range(1, len(B)+1):
                match = costmatrix[i-1][j-1] + self.diagonalscore(A[i-1], B[j-1])
                delete = costmatrix[i-1][j] + INDEL
                insert = costmatrix[i][j-1] + INDEL
                if min(match, insert, delete) == match:
                    backpointer[i][j] = "match"
                elif min(match, insert, delete) == delete:
                    backpointer[i][j] = "delete"
                elif min(match, insert, delete) == insert:
                    backpointer[i][j] = "insert"
                costmatrix[i][j] = min(match, insert, delete)
        return costmatrix[len(A)][len(B)], backpointer
```



```python
    #evalautes the indices of the matrices in the banded coordinates
    def evaluateindices(self, costmatrix, i, j):
        #transform in the band range
        jband = j - i + MAXINDELS
        if not 0 <= jband < 2 * MAXINDELS + 1:
            return math.inf
        return costmatrix[i][jband]

    #computes the cost of the banded algorithm
    def computecost_banded(self, A, B):
        costmatrix = [[0 for i in range(2 * MAXINDELS + 1)] for j in range(len(A) + 1)]
        backpointer = [[0 for i in range(2 * MAXINDELS + 1)] for j in range(len(A) + 1)]
        # fill the first row and column
        for i in range(1, MAXINDELS + 1):
            costmatrix[i][0] = INDEL * i
            backpointer[i][0] = "delete"
        for j in range(1, MAXINDELS + 1):
            costmatrix[0][j] = INDEL * j
            backpointer[0][j] = "insert"

        for i in range(1, len(A) + 1):
            #loop over the particular band range
            for j in range(max(1, i - MAXINDELS), min(len(B), i + MAXINDELS) + 1):
                match =  self.evaluateindices(costmatrix,i - 1, j - 1) + self.diagonalscore(A[i-1], B[j-1])
                delete = self.evaluateindices(costmatrix,i - 1, j)  + INDEL
                insert = self.evaluateindices(costmatrix,i , j - 1) + INDEL
                #fill the backpointer array
                if min(match, insert, delete) == match:
                    backpointer[i][j - i + MAXINDELS] = "match"
                elif min(match, insert, delete) == delete:
                    backpointer[i][j - i + MAXINDELS] = "delete"
                elif min(match, insert, delete) == insert:
                    backpointer[i][j - i + MAXINDELS] = "insert"
                costmatrix[i][j - i + MAXINDELS] = min(match, delete, insert)
        return self.evaluateindices(costmatrix, len(A), len(B)),backpointer
```



```python
    #compute the text alignment
    def alignment(self, backpointer, i,j, A,B):
        i = len(A)
        j = len(B)
        newA = []
        newB = []
        while i > 0 or j > 0:
            if backpointer[i][j] == "match":
                newA.append(A[i-1])
                newB.append(B[j-1])
                i = i-1
                j = j-1
            elif backpointer[i][j] == "insert":
                newA.append("-")
                newB.append(B[j-1])
                j = j-1
            elif backpointer[i][j] == "delete":
                newA.append(A[i-1])
                newB.append("-")
                i = i-1
            return newA, newB

```



```python
    def align( self, sequences, table, banded, align_length):
        self.banded = banded
        self.MaxCharactersToAlign = align_length
        sequencei = [0 for i in range(self.MaxCharactersToAlign)]
        sequencej = [0 for j in range(self.MaxCharactersToAlign)]
        results = []
        backpointer = []
        for i in range(len(sequences)):
            jresults = []
            for j in range(len(sequences)):

                if(j < i):
                    s = {}
                else:
                    if self.MaxCharactersToAlign > len(sequences[i]):
                        sequencei = sequences[i]
                    if self.MaxCharactersToAlign > len(sequences[j]):
                        sequencej = sequences[j]
                    else:
                        sequencei = sequences[i][:self.MaxCharactersToAlign]
                        sequencej = sequences[j][:self.MaxCharactersToAlign]
                    if banded:
                        minscore, backpointer = self.computecost_banded(sequencei, sequencej)
                        score = minscore
                        alignment1, alignment2 = self.alignment(backpointer,MAXINDELS,MAXINDELS, sequencei, sequencej)
                    else:
                        minscore, backpointer = self.computecost_unrestricted(sequencei, sequencej)
                        score = minscore
                        alignment1, alignment2 = self.alignment(backpointer, len(sequencei),len(sequencej),sequencei, sequencej)
##################################################################################################
# your code should replace these three statements and populate the three variables: score, alignment1 and alignment2
                    # alignment1 = 'abc-easy  DEBUG:(seq{}, {} chars,align_len={}{})'.format(i+1,
                    #     len(sequences[i]), align_length, ',BANDED' if banded else '')
                    # alignment2 = 'as-123--  DEBUG:(seq{}, {} chars,align_len={}{})'.format(j+1,
                    #     len(sequences[j]), align_length, ',BANDED' if banded else '')
##################################################################################################
                    s = {'align_cost':score, 'seqi_first100':alignment1, 'seqj_first100':alignment2}
                    table.item(i,j).setText('{}'.format(int(score) if score != math.inf else score))
                    table.update()
                jresults.append(s)
            results.append(jresults)
        return results

```