Project 4 Report

Explain the time and space complexity of your algorithm by showing and summing up the complexity of each subsection of your code.
There are a few subsections of code that the unrestricted and banded algorithms have in common. I grouped those subsections of code into the Method abstract class.

calc_min_prev
Time O(1), Space O(1)
This function loops through the plausible previous cells for a the given cell (i, j). The function is dependent on the calc_prevs function, which is implemented differently by the unrestricted and banded algorithms. Regardless of the implementation of calc_prevs, calc_min_prev loops through an array of two or three plausible previous cells. The number of plausible previous cells does not scale up with the size of m and n. Therefore, this function has a time complexity of O(1) and a space complexity O(1).

get_alignments
Time O(n + m), Space O(n + m)
An in-depth description for how this algorithm works is found below under the label "Alignment extraction algorithm". In its worst case, the algorithm will loop either along the bottom of the table and up the left side or up the right side and along the top of the table. In other words, it will cycle through n + m cells and store alignment strings that are n + m characters long. Even in the banded implementation, the algorithm does not assume how wide the band is. In the worst case, the band would occupy the whole table. In any case, the time complexity is O(n + m) and the space complexity is O(n + m)

[10 points] Your analysis should show that your unrestricted algorithm is at most O(nm) time and space.
align
Time O(nm), Space O(nm)
The dominant operation in this algorithm is a double for loop that cycles through every cell in the table, except for the top and leftmost cells. The body of the double for loop runs calc_min_prev, whose time and space complexities are O(1) (as described above). The body of the double for loop also assigns the values of the cell in the scores and prevs dictionaries, which also have time and space complexities of O(1).

[10 points] Your analysis should show that your banded algorithm is at most O(kn) time and space.
align
Time O(kn), Space O(kn)
Below is a table that highlights the different sections of cells that my banded align algorithm addresses.

Grey: Initial section

Populating the initial section runs in O(k) time and space. About half of k cells are populated on the top row, while the other half is populated in the leftmost column. In this case k is 7, and the initial section has 7 cells in it.

Green: Section 1

Populating section 1 runs in O(k^2) time and space. Section 1 is at most k cells wide and is about k/2 cells deep. Since k is expected to be a relatively low constant compared to n and m, populating this section of the table in practice is a subordinate operation. I acknowledge that this section could be viewed as the dominant operation, since k has the potential to be much larger than n and m. For the scope of this project though, where k is set to 7, I will treat this section of the algorithm as if it scaled with constant time and space being unbound to the size of n and m.

Blue: Section 2

Populating section two runs in O(kn) time and space. It is k cells wide and O(n) cells deep. Each cell is populated in constant time.

Red: Section 3

Like section 1, populating section 3 technically runs in O(k^2) time and space. However, k is expected to be a low value constant, so this is not nearly a dominant operation.

	-	e	x	p	o	n	e	n	t	i	a	l
-	0	5	10	15	-	-	-	-	-	-	-	-
p	5	1	6	7	12	-	-	-	-	-	-	-
o	10	6	2	7	4	9	-	-	-	-	-	-
l	15	11	7	3	8	5	10	-	-	-	-	-
y	-	16	12	8	4	9	6	11	-	-	-	-
n	-	-	17	13	9	1	6	3	8	-	-	-
o	-	-	-	18	10	6	2	7	4	9	-	-
m	-	-	-	-	15	11	7	3	8	5	10	-
i	-	-	-	-	-	16	12	8	4	5	6	11
a	-	-	-	-	-	-	17	13	9	5	2	7
l	-	-	-	-	-	-	-	18	14	10	6	-1

[10 points] Write a paragraph that explains how your alignment extraction algorithm works, including the backtrace

Alignment extraction algorithm:

1. *Get the indexes (i, j) for the lower right cell, the same cell where the final score should be extracted from.*
2. *Get the prev character from the prevs dictionary under the (i, j) key.*
3. *Evaluate the prev character:*
    a. *If the prev character is 'D', then the previous cell to (i, j) is (i - 1, j - 1). The ith character of the first sequence and the jth character of the second sequence are prepended to alignment string 1 and alignment string 2, respectively. These additions to the alignment strings represent substitution or, if the additions are*

identical, matching. After prepending to the alignment strings, i and j should be decremented.

   b. *If the prev character is 'U', then the previous cell to (i, j) is (i - 1, j). The ith character of the first sequence and a dash '-' character are prepended to alignment string 1 and alignment string 2, respectively. These additions to the alignment strings represent deletion. After prepending to the alignment strings, i should be decremented.*
   c. *If the prev character is 'L', then the previous cell to (i, j) is (i, j - 1). A dash '-' character and the jth character of the second sequence are prepended to alignment string 1 and alignment string 2, respectively. These additions to the alignment strings represent insertion. After prepending to the alignment strings, j should be decremented.*

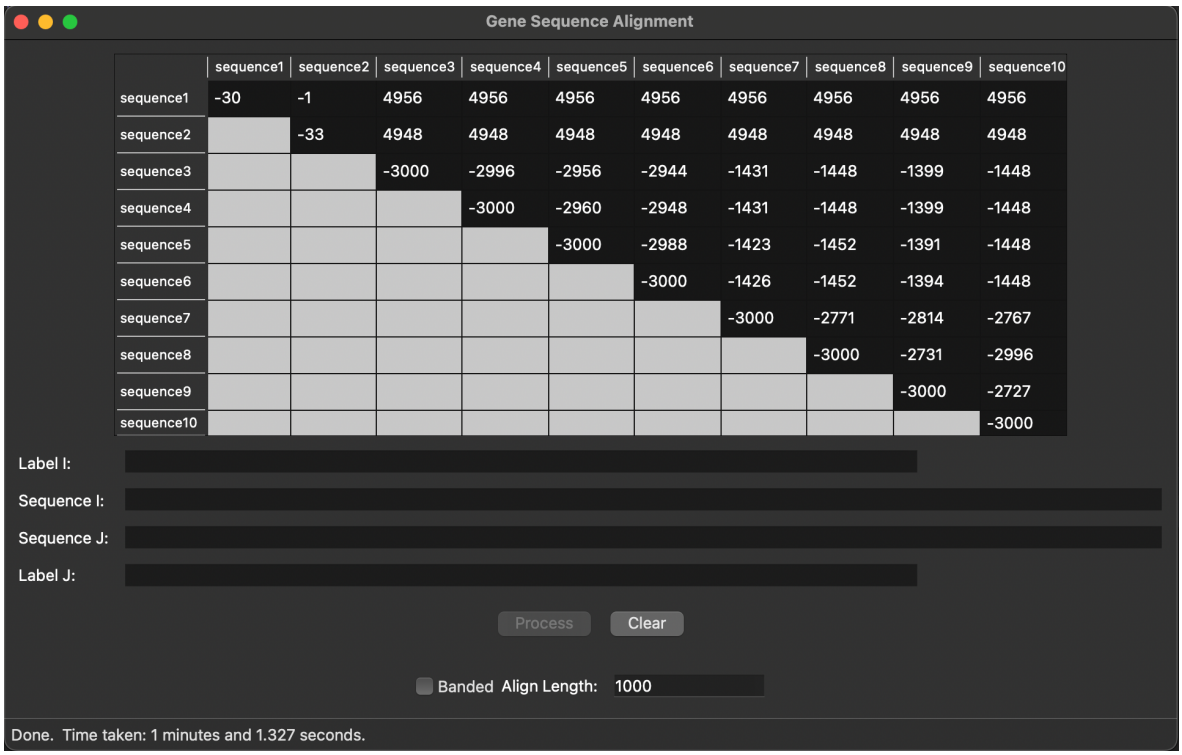
4. *Repeat steps 2 and 3 until the prev character from the prevs dictionary under the key (i, j) is None. This means that the alignment strings have been fully built and should be returned.*

[20 points] Include a "results" section showing both a screen-shot of your 10x10 score matrix for the unrestricted algorithm with align length n = 1000 and a screen-shot of your 10x10 score matrix for the banded algorithm with align length n = 3000.
See Results section

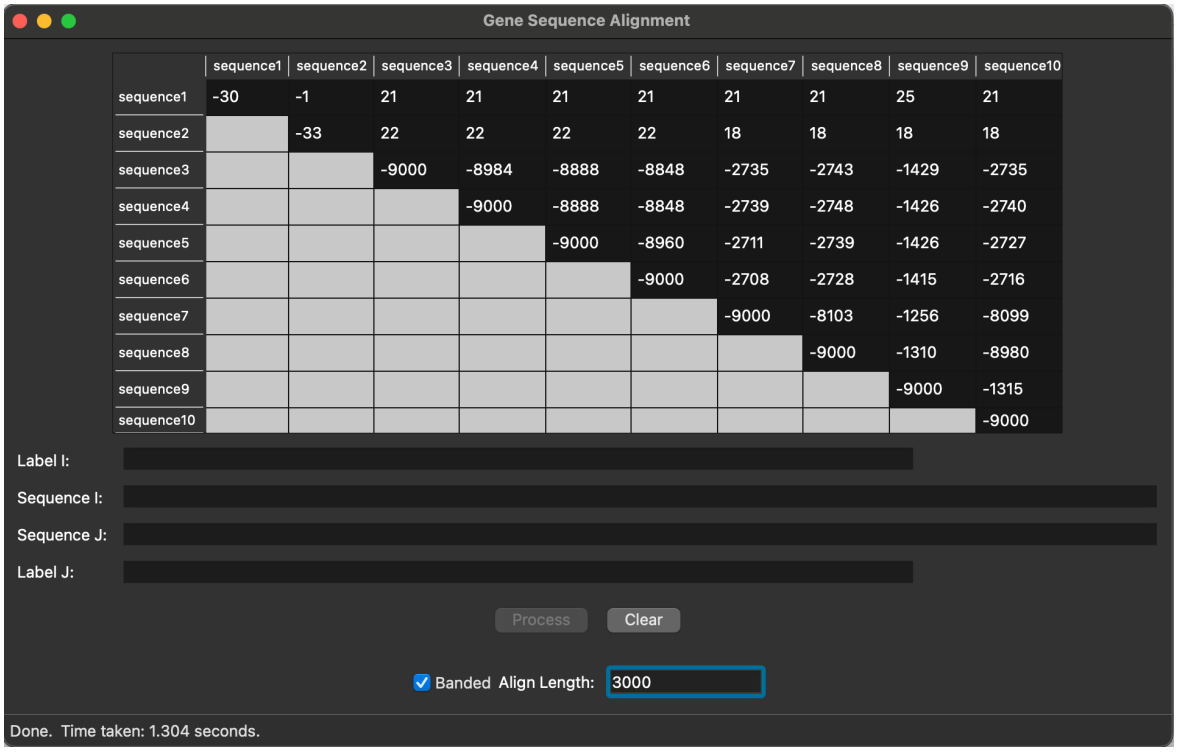
[10 points] Include in the "results" section the extracted alignment for the first 100 characters of sequences #3 and #10 (counting from 1), computed using the unrestricted algorithm with n = 1000. Display the sequences in a side-by-side fashion in such a way that matches, substitutions, and insertions/deletions 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 n = 3000.
See Results section

[30 points] Attach your commented source code for both your unrestricted and banded algorithms.
See Code Snippets section

Results



Performance of my unrestricted algorithm with an alignment length of 1000



Performance of my banded algorithm with an alignment length of 3000

Unrestricted algorithm alignments, aligning sequence #3 with sequence #10

```
gattgcgagcgatttgcgtgcgtgcatcccgcttc-actg--at-ctcttgttagatcttttcataatctaaactttataaaaacatccactccctgta-
-ataa-gagtgattggcgtccgtacgtacccttttctactctcaaactcttgttagtttaaatc-taatctaaactttataaa--cggc-acttcctgtgt
```

Banded algorithm alignments, aligning sequence #3 with sequence #10

```
gattgcgagcgatttgcgtgcgtgcatcccgcttc-actg--at-ctcttgttagatcttttcataatctaaactttataaaaacatccactccctgta-
-ataa-gagtgattggcgtccgtacgtacccttttctactctcaaactcttgttagtttaaatc-taatctaaactttataaa--cggc-acttcctgtgt
```

Code Snippets

Gene_Sequencing.py

```python
#!/usr/bin/python3
from method.Banded import Banded
from method.Unrestricted import Unrestricted
from which_pyqt import PYQT_VER

if PYQT_VER == 'PYQT5':
    pass
else:
    raise Exception('Unsupported Version of PyQt: {}'.format(PYQT_VER))


class GeneSequencing:
    # Unrestricted: Time O(nm), Space O(nm)
    # Banded: Time O(kn), Space O(kn)
    def align(self, seq1, seq2, banded, align_length):
        # Time O(1), Space O(1)
        if banded:
            method = Banded(seq1, seq2, align_length)
        else:
            method = Unrestricted(seq1, seq2, align_length)

        # Unrestricted: Time O(nm), Space O(nm)
        # Banded: Time O(kn), Space O(kn)
        method.align()

        # Time O(1), Space O(1)
        score = method.get_score()

        # Time O(n + m), Space O(n + m)
        alignment1, alignment2 = method.get_alignments()

        return {'align_cost': score,
                'seqi_first100': alignment1,
                'seqj_first100': alignment2}
```

Method.py

```
from abc import ABC, abstractmethod

MATCH = -3
INS_DEL = 5
SUB = 1


class Method(ABC):
    # Time O(1), Space O(1)
    def __init__(self, seq1, seq2, align_length):
        self.seq1 = '-' + seq1
        self.seq2 = '-' + seq2
        self.seq1_length = len(seq1)
        self.seq2_length = len(seq2)
        self.align_length1 = min(self.seq1_length + 1, align_length + 1)
        self.align_length2 = self.calc_align_length2(align_length)
        self.scores = {}
        self.prevs = {}

    # ------------------------------------------------------------------------#
    #                                                                         #
    #                                  Align                                  #
    #                                                                         #
    # ------------------------------------------------------------------------#
    @abstractmethod
    def calc_align_length2(self, align_length):
        pass

    # Time O(1), Space O(1)
    def calc_diagonal_score(self, i, j):
        diagonal_score = self.scores[i - 1, j - 1]
        align1 = self.seq1[i]
        align2 = self.seq2[j]

        return diagonal_score + MATCH if align1 == align2 else diagonal_score + SUB

    @abstractmethod
    def calc_prevs(self, i, j):
        pass

    # Time O(1), Space O(1)
    def calc_min_prev(self, i, j):
        score = 0
        prev_char = 1
        priority = 2
        prevs = self.calc_prevs(i, j)
```

```
        min_prev = prevs.pop()
        for prev in prevs:
            if prev[score] < min_prev[score] or \
                    (prev[score] == min_prev[score] and prev[priority] <
min_prev[priority]):
                min_prev = prev

        return min_prev[score], min_prev[prev_char]

    # Time: 0(1), Space: 0(1)
    def align_initial(self):
        self.scores[0, 0] = 0
        self.prevs[0, 0] = None

    @abstractmethod
    def align(self):
        pass

    # ------------------------------------------------------------------------#
    #                                                                         #
    #                                Deliverables                             #
    #                                                                         #
    # ------------------------------------------------------------------------#
    # Time O(1), Space O(1)
    def get_score(self):
        return self.scores[self.align_length1 - 1, self.align_length2 - 1]

    # Time O(n + m), Space O(n + /[-[m)
    def get_alignments(self):
        alignment1 = ''
        alignment2 = ''
        i = self.align_length1 - 1
        j = self.align_length2 - 1
        curr = self.prevs[i, j]

        while curr is not None:
            if curr == 'D':
                alignment1 = self.seq1[i] + alignment1
                alignment2 = self.seq2[j] + alignment2
                i -= 1
                j -= 1

            elif curr == 'U':
                alignment1 = self.seq1[i] + alignment1
                alignment2 = '-' + alignment2
                i -= 1
```

```
            elif curr == 'L':
                alignment1 = '-' + alignment1
                alignment2 = self.seq2[j] + alignment2
                j -= 1

            curr = self.prevs[i, j]

        return alignment1, alignment2
```

Unrestricted.py

```
from Method import Method

MATCH = -3
INS_DEL = 5
SUB = 1


class Unrestricted(Method):
    # Time: O(1), Space: O(1)
    def calc_align_length2(self, align_length):
        return min(self.seq2_length + 1, align_length + 1)

    # ------------------------------------------------------------------------------#
    #                                                                               #
    #                                    Align                                      #
    #                                                                               #
    # ------------------------------------------------------------------------------#
    # Time: O(1), Space: O(1)
    def calc_prevs(self, i, j):
        return [[self.calc_diagonal_score(i, j), 'D', 2],
                [self.scores[i - 1, j] + INS_DEL, 'U', 1],
                [self.scores[i, j - 1] + INS_DEL, 'L', 0]]

    # Time O(nm), Space O(nm)
    def align(self):
        self.align_initial()

        # ----------------------------- Align edges -----------------------------#
        # Time O(n), Space O(n)
        for i in range(1, self.align_length1):
            self.scores[i, 0] = i * INS_DEL
            self.prevs[i, 0] = 'U'

        # Time O(m), Space O(m)
        for j in range(1, self.align_length2):
```

```
            self.scores[0, j] = j * INS_DEL
            self.prevs[0, j] = 'L'

        # ------------------------------ Align rest -------------------------------#
        # Time O(nm), Space O(nm)
        for i in range(1, self.align_length1):
            for j in range(1, self.align_length2):
                prev_score, prev_char = self.calc_min_prev(i, j)

                self.scores[i, j] = prev_score
                self.prevs[i, j] = prev_char
```

Banded.py

```
from method.Method import Method

MATCH = -3
INS_DEL = 5
SUB = 1

MAX_INS_DEL = 3
BAND_RADIUS = MAX_INS_DEL + 1
BAND_WIDTH = 2 * MAX_INS_DEL + 1


class Banded(Method):
    # Time: O(1), Space: O(1)
    def calc_align_length2(self, align_length):
        return min(self.seq2_length + 1, align_length + 1, self.align_length1 +
MAX_INS_DEL)

    # ---------------------------------------------------------------------------#
    #                                                                            #
    #                                  Align                                     #
    #                                                                            #
    # ---------------------------------------------------------------------------#
    # Time: O(1), Space: O(1)
    def calc_prevs(self, i, j):
        prevs = [[self.calc_diagonal_score(i, j), 'D', 2]]

        if (i + MAX_INS_DEL) != j:
            prevs.append([self.scores[i - 1, j] + INS_DEL, 'U', 1])

        if (i - MAX_INS_DEL) != j:
            prevs.append([self.scores[i, j - 1] + INS_DEL, 'L', 0])

        return prevs
```

```
    # Time O(1), Space O(1)
    def align_cell(self, i, j):
        prev_score, prev_char = self.calc_min_prev(i, j)

        self.scores[i, j] = prev_score
        self.prevs[i, j] = prev_char

    # Time: O(kn), Space: O(kn)
    def align(self):
        self.align_initial()

        # ------------------------------ Align edges ------------------------------#
        # Time: O(k), Space: O(k)
        for i in range(1, BAND_RADIUS):
            self.scores[i, 0] = i * INS_DEL
            self.prevs[i, 0] = 'U'

        # Time: O(k), Space: O(k)
        for j in range(1, BAND_RADIUS):
            self.scores[0, j] = j * INS_DEL
            self.prevs[0, j] = 'L'

        # ------------------------ Align upper left section ------------------------#
        # Time: O(k^2), Space: O(k^2)
        for i in range(1, BAND_RADIUS):
            for j in range(1, BAND_RADIUS + i):
                self.align_cell(i, j)

        # ------------------------- Align center section -------------------------#
        # Time: O(kn), Space: O(kn)
        for i in range(BAND_RADIUS, self.align_length1 - MAX_INS_DEL):
            for j in range(i - MAX_INS_DEL, i + BAND_RADIUS):
                self.align_cell(i, j)

        # ------------------------ Align lower right section ------------------------#
        # Time: O(k^2), Space: O(k^2)
        for i in range(self.align_length1 - MAX_INS_DEL, self.align_length1):
            for j in range(i - MAX_INS_DEL, min(i + BAND_RADIUS,
self.align_length2)):
                self.align_cell(i, j)
```