Unrestricted

```
def alignCode(self, sequence1, sequence2):
    # Initializes A and B arrays
    self.A = [[math.inf for row in range(len(sequence2) + 1)] for col in range(len(sequence1) + 1)]
    self.B = [[math.inf for row in range(len(sequence2) + 1)] for col in range(len(sequence1) + 1)]

    # Initializes row 0 and col 0
    for row in range(len(sequence1) + 1):
        self.A[row][0] = row * 5
        self.B[row][0] = UP
    for col in range(len(sequence2) + 1):
        self.A[0][col] = col * 5
        self.B[0][col] = LEFT

    # Runs the alignment algorithm on the rest of the Array
    for row in range(len(sequence1)):
        row += 1
        for col in range(len(sequence2)):
            col += 1
            upVal = self.A[row - 1][col] + INDEL
            leftVal = self.A[row][col - 1] + INDEL
            diagVal = self.diagonal(row, col, sequence1, sequence2)

            self.A[row][col] = min(upVal, leftVal, diagVal)  # Taking minimum of the three values

            # Sets the B array value with either UP, LEFT, or DIAG
            if self.A[row][col] == diagVal:
                self.B[row][col] = DIAG
            if self.A[row][col] == upVal:
                self.B[row][col] = UP
            if self.A[row][col] == leftVal:
                self.B[row][col] = LEFT
            self.score = self.A[row][col]

    # Extracts the solution from the B array
    self.extract(sequence1, sequence2)
```

```python
def extract(self, sequence1, sequence2):
    row = len(sequence1)
    col = len(sequence2)
    self.align1 = ""
    self.align2 = ""
    while row != 0 and col != 0:
        if self.B[row][col] == LEFT:
            self.align1 += sequence2[col - 1]
            self.align2 += "-"
            col = col - 1
        elif self.B[row][col] == UP:
            self.align2 += sequence1[row - 1]
            self.align1 += "-"
            row = row - 1
        else:
            self.align1 += sequence2[col - 1]
            self.align2 += sequence1[row - 1]
            row = row - 1
            col = col - 1
```

Time Complexity:

- Overall it is O(nm) time
- O(nm) time from initializing the A array to size n * m
- O(nm) time from initializing the B array to size n * m
- O(n) time from initializing row 0 to 0, 5, 10, 15...
- O(m) time from initializing col 0 to 0, 5, 10, 15…
- O(nm) time from filling in the A array with INDEL, SUB, and MATCH values
- O(nm) time worst case from extract algorithm (if visiting every value in the array during the backtrace).
- O(nm + nm + n + n + m + nm + nm) overall — which becomes **O(nm)**

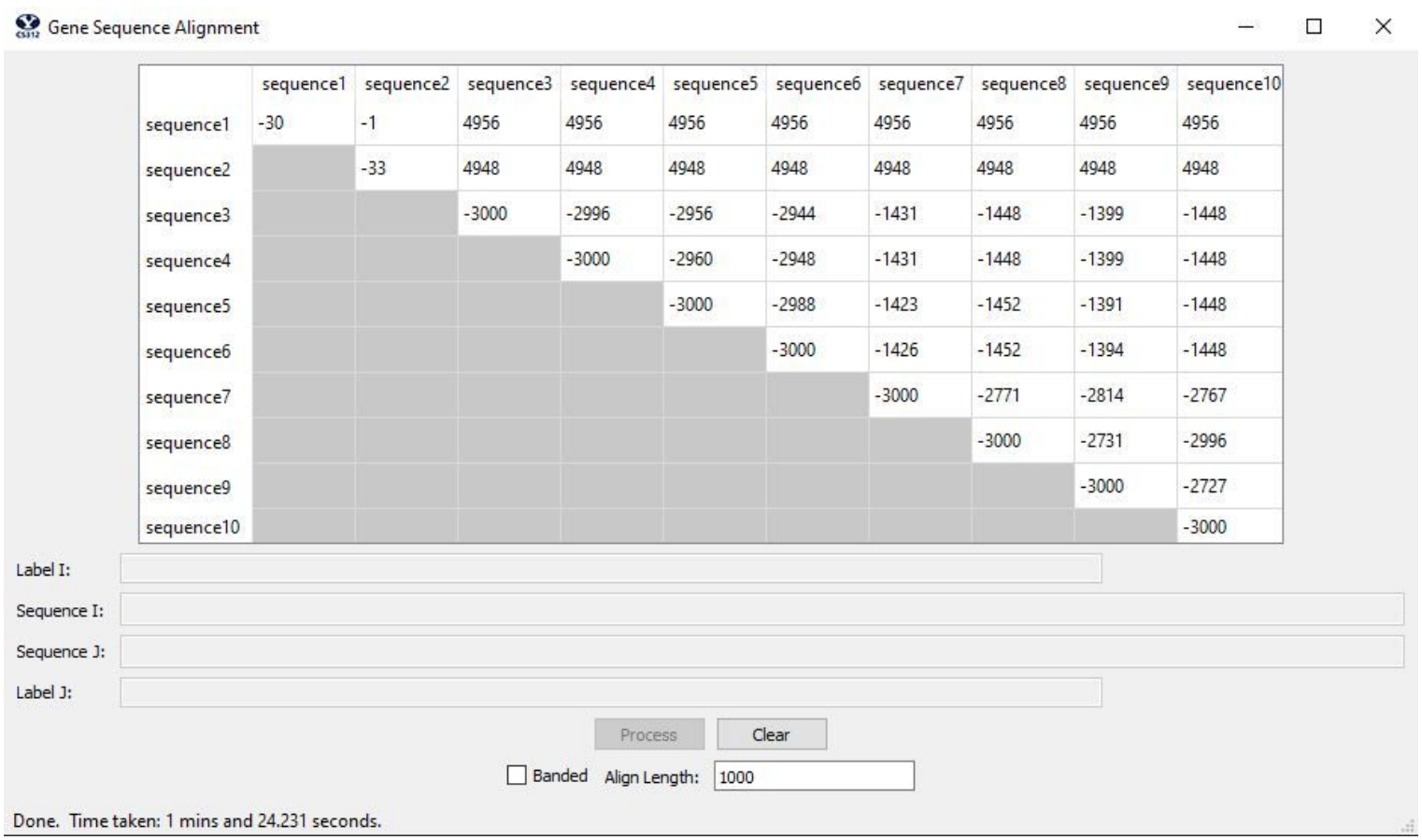
Space Complexity:

- Overall it is O(nm) space
- O(nm) space from storing the A array.
- O(nm) space from storing the B array.
- O(nm + nm) overall — which becomes **O(nm)**

Alignment Extraction Algorithm:

- As we went through the A array to fill in the values for INDEL, DEL, and MATCH, we would fill in the B array in the same spot with a previous pointer pointing at the place where we came from (either LEFT, UP, or DIAG).
- After the A and B array were completely filled, we started at the far corner of the B array ([n][m]) and did a backtrace using the previous pointers. In order to the backtrace, we would look at the previous pointer in the spot that we were in and move to the previous position.
- As we did the backtrace, we extracted the alignment by adding the characters based on this logic:
    - If the previous pointer is LEFT, add sequence2[col-1] to alignment2, add “-” to alignment1.
    - If the previous pointer is UP, add sequence1[row-1] to alignment1, add “-” to alignment1.
    - If the previous pointer is DIAG, add sequence1[row-1] to alignment1, add sequence2[col-1] to alignment2.

Results for Unrestricted:



	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

gattgcgagcgatttgcgtgcgtgcatcccgcttc-actg--at-ctcttgttagatcttttcataatc
t

ataa-gagtgattggcgtccgtacgtaccctttctactctcaaactcttgttagtttaaatc-taatct
a

aactttataaaaacatccactccctgta-g

aactttataaa--cggc-acttcctgtgtg

Banded

```
def alignCodeBanded(self, sequence1, sequence2):
    # Initializes A and B arrays
    self.A = [[math.inf for row in range(7)] for col in range(len(sequence1) + 1)]
    self.B = [[math.inf for row in range(7)] for col in range(len(sequence1) + 1)]

    # Initializes row 0 and col 0
    self.setValues();

    # Runs the banded alignment algorithm on the rest of the array
    for row in range(len(sequence1)):
        row += 1
        for col in range(7):
            self.skipNeededCheck(row, col)

            if not self.skipNeeded:
                upVal = self.newDiagonal(row, col, sequence1, sequence2)

                if col == 0:
                    leftVal = math.inf
                else:
                    leftVal = self.A[row][col - 1] + INDEL

                if col == 6:
                    diagVal = math.inf
                else:
                    diagVal = self.A[row - 1][col + 1] + INDEL

                self.A[row][col] = min(upVal, leftVal, diagVal)  # Taking minimum of the three values

                # Sets the B array value with either UP, LEFT, or DIAG
                if self.A[row][col] == diagVal:
                    self.B[row][col] = DIAG
                if self.A[row][col] == upVal:
                    self.B[row][col] = UP
                if self.A[row][col] == leftVal:
                    self.B[row][col] = LEFT

    # Sets the score for the alignment
    self.score = self.A[len(sequence1)][3]

    # Extracts the solution from the B array
    self.extractBanded(sequence1, sequence2)
```

```python
def extractBanded(self, sequence1, sequence2):
    self.B[0][3] = None
    row = len(sequence1)
    col = 3
    len1 = len(sequence1)
    len2 = len(sequence2)
    self.align1 = ""
    self.align2 = ""
    while self.B[row][col] is not None:
        if self.B[row][col] == LEFT:
            self.align2 += sequence2[col + row - 4]
            self.align1 += "-"
            col = col - 1
        elif self.B[row][col] == DIAG:
            self.align1 += sequence1[row - 1]
            self.align2 += "-"
            col = col + 1
            row = row - 1
        else:
            self.align2 += sequence2[col + row - 4]
            self.align1 += sequence1[row - 1]
            row = row - 1
```

Time Complexity:

- Overall it is O(kn) time (k = 7)
- O(kn) time from initializing the A array to size k * n
- O(kn) time from initializing the B array to size k * n
- O(kn) time from filling in the A array with INDEL, SUB, and MATCH values
- O(kn) time worst case from extract algorithm (if visiting every value in the array during the backtrace).
- O(kn + kn + kn + kn) overall — which becomes **O(kn)**

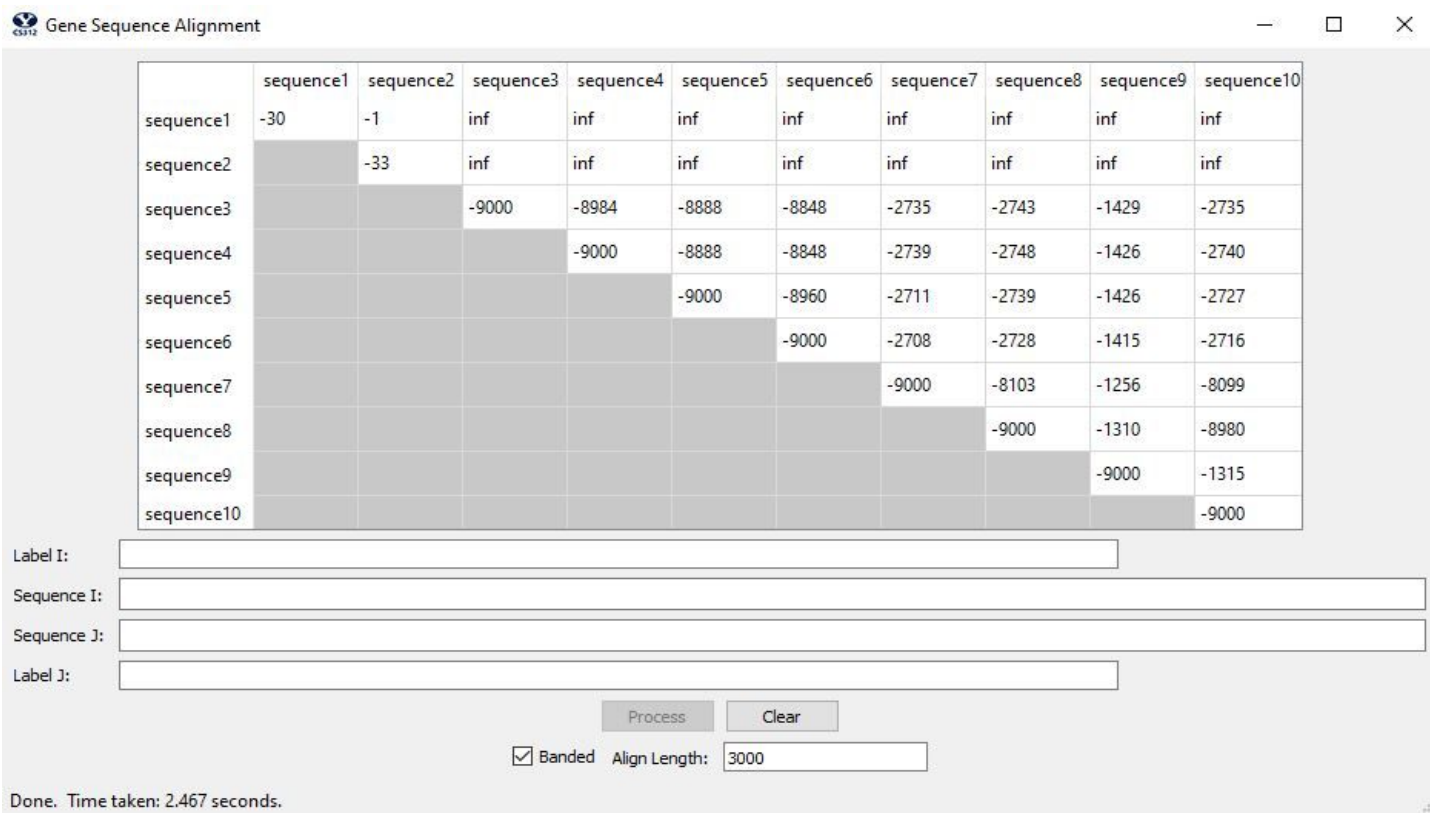
Space Complexity:

- Overall it is O(kn) space
- O(kn) space from storing the A array.
- O(kn) space from storing the B array.
- O(kn + kn) overall — which becomes **O(nm)**

Alignment Extraction Algorithm:

- As we went through the A array to fill in the values for INDEL, DEL, and MATCH, we would fill in the B array in the same spot with a previous pointer pointing at the place where we came from (either LEFT, UP, or DIAG).
- After the A and B array were completely filled, we started at the last value at the bottom of the A array that was filled in and did a backtrace using the previous pointers. In order to the backtrace, we would look at the previous pointer in the spot that we were in and move to the previous position.
- As we did the backtrace, we extracted the alignment by adding the characters based on this logic:
    - If the previous pointer is LEFT, add sequence2[col + row - 4] to alignment2, add "-" to alignment1.
    - If the previous pointer is DIAG, add sequence1[row-1] to alignment1, add "-" to alignment1.
    - If the previous pointer is UP, add sequence1[row-1] to alignment1, add sequence2[col + row - 4] to alignment2.

Results for Banded:



	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-1	inf	inf	inf	inf	inf	inf	inf	inf
sequence2		-33	inf	inf	inf	inf	inf	inf	inf	inf
sequence3			-9000	-8984	-8888	-8848	-2735	-2743	-1429	-2735
sequence4				-9000	-8888	-8848	-2739	-2748	-1426	-2740
sequence5					-9000	-8960	-2711	-2739	-1426	-2727
sequence6						-9000	-2708	-2728	-1415	-2716
sequence7							-9000	-8103	-1256	-8099
sequence8								-9000	-1310	-8980
sequence9									-9000	-1315
sequence10										-9000

-gatt-gcgagcgatttgcgtgcgtgcatcccgcttcactgatctcttgttagatcttttcat-aatct
a

ataagagtgattggcgtccgtacgtac-cct--ttctactc-tcaaactcttgttagtttaaatctaat
c

aactttataaaa--acat-ccactccctgt

taaactttataaacggcacttcctgtgt-g