COT 4400 Project 2

Reubin George

Yuan Chieh Lou

Malek Khaled Zaben

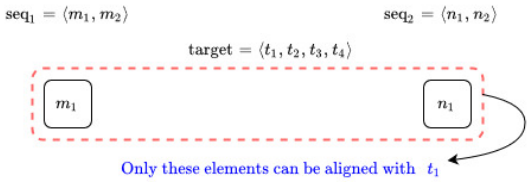
Group 8

November 12, 2021

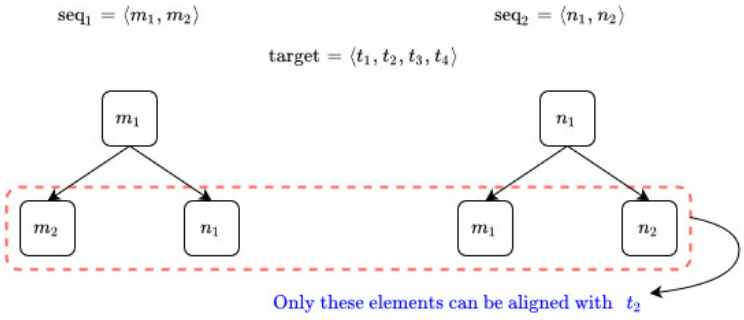
1. ***How would you break down the problem of calculating the maximum alignment score for a combination of two sequences seq_1 and seq_2 (length n and m, respectively) with the sequence target $(length\ n + m)$ into one or more smaller instances of this problem? Your answer should include what subproblems you are evaluating, as well as how you are using the answers to these subproblems to solve the original problem.***

   We look at all possible subsequence combinations of seq_1 and seq_2 and create a recursion tree. We then look at all possible branches to figure out how to create a look up table to store recurring values. Let's look at the steps that will be used to solve this problem. We are given two input sequences, $n = \langle n_1\ ,\ n_2\rangle$, $m = \langle m_1\ ,\ m_2\rangle$ and a target sequence $t = \langle t_1, t_2, t_3, t_4\rangle$. Keep in mind that the size of target sequence must be equal to the sum of size of a and size of b.

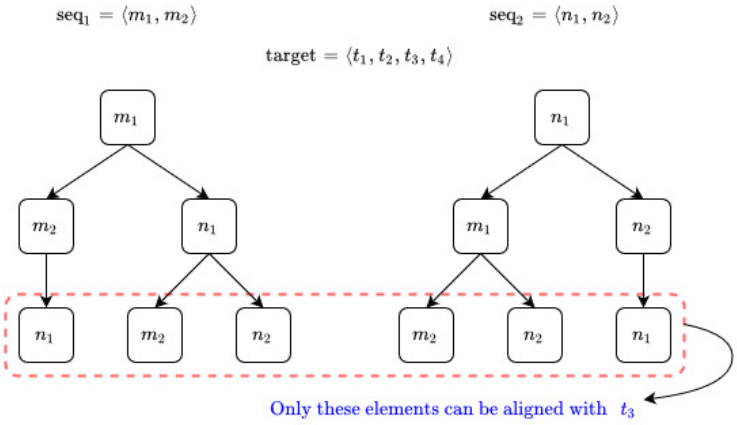
   a. Start with first element of the target sequence. To get the best alignment score the first target element must be paired with either the first element of $\vec{n}$ or $\vec{m}$.



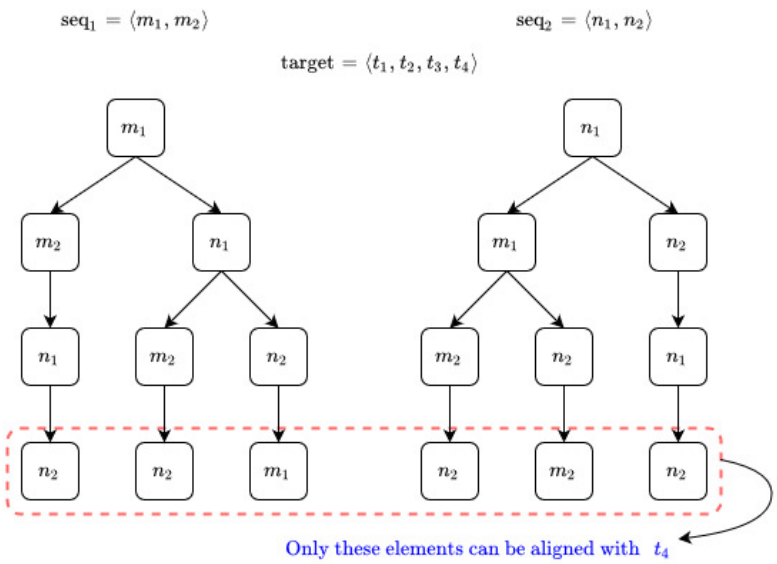
   b. Look at all the remaining sequences that best aligns with the next element in the target sequence.



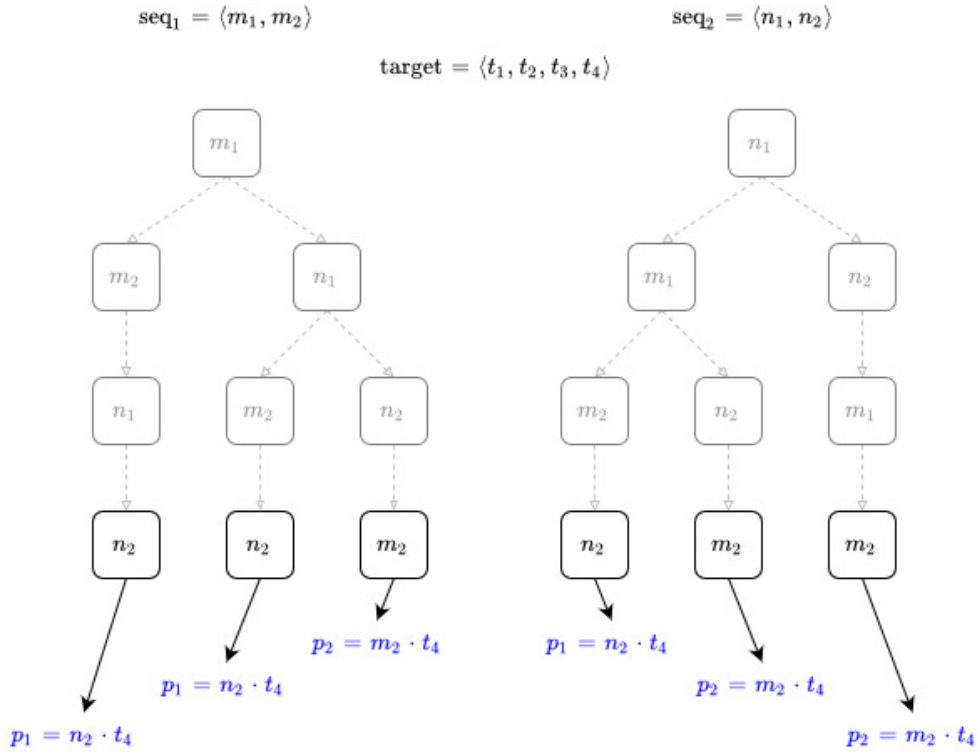
   c. Follow each branch of the recursion tree to find the next element.



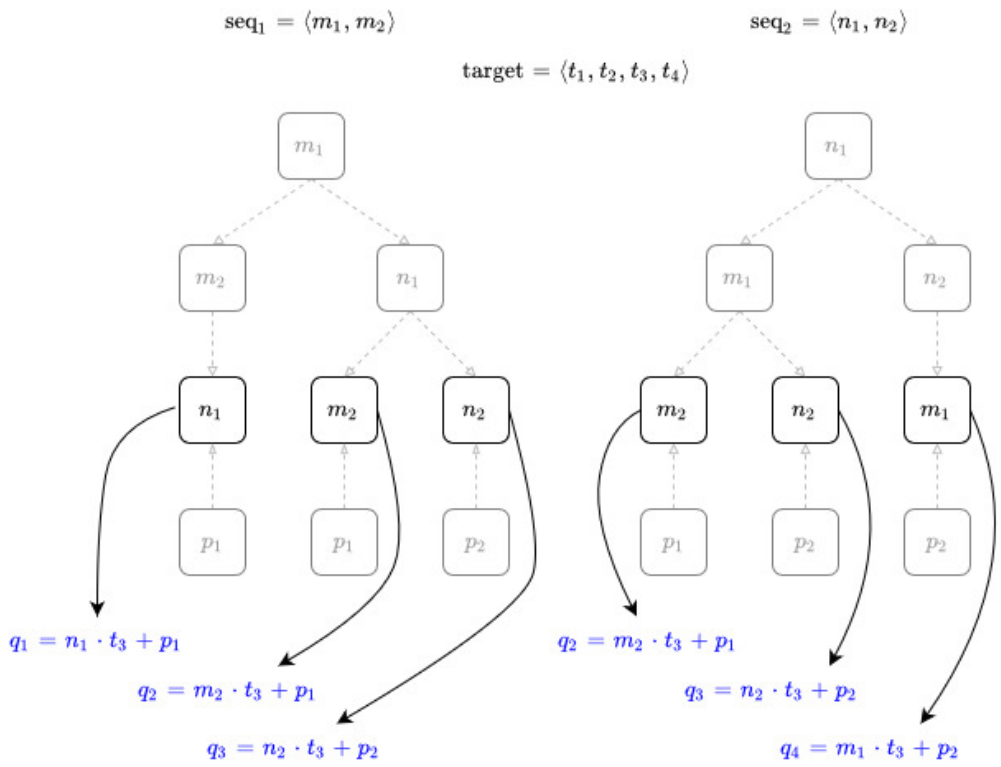
   d. Finally, because there are no other elements left in any of the three sequences. This must be our base case.



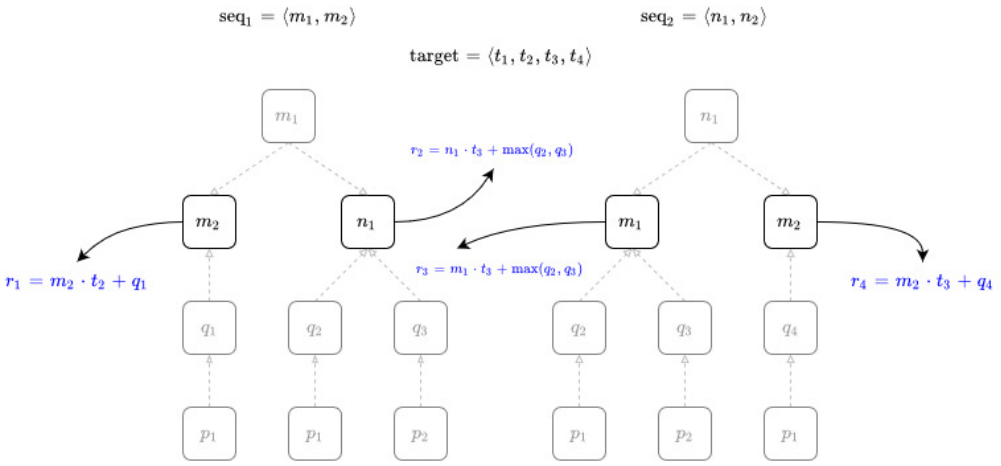
e. Once the base case has been identified travel up the recursion tree to gain the necessary results, starting with the base case.



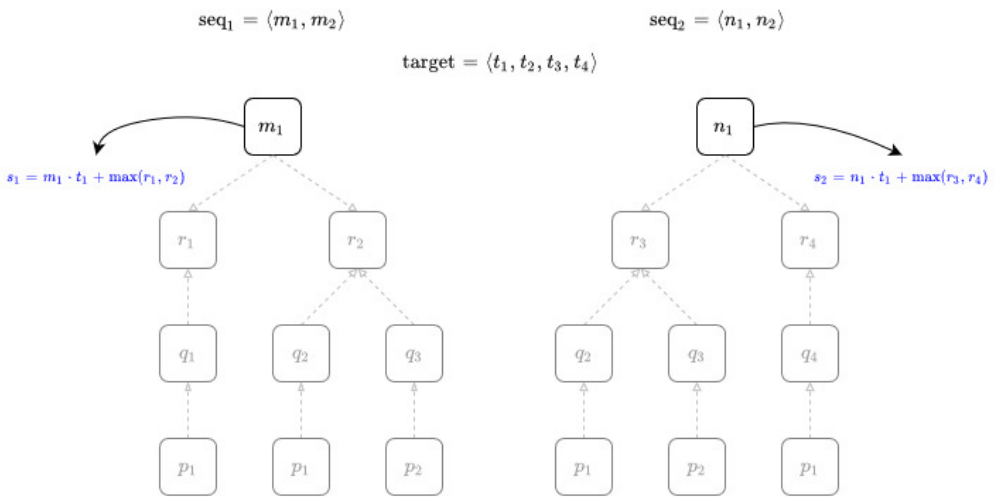
f. From the base case, we can go up the tree to the parent of each node to calculate the value of the parent node.



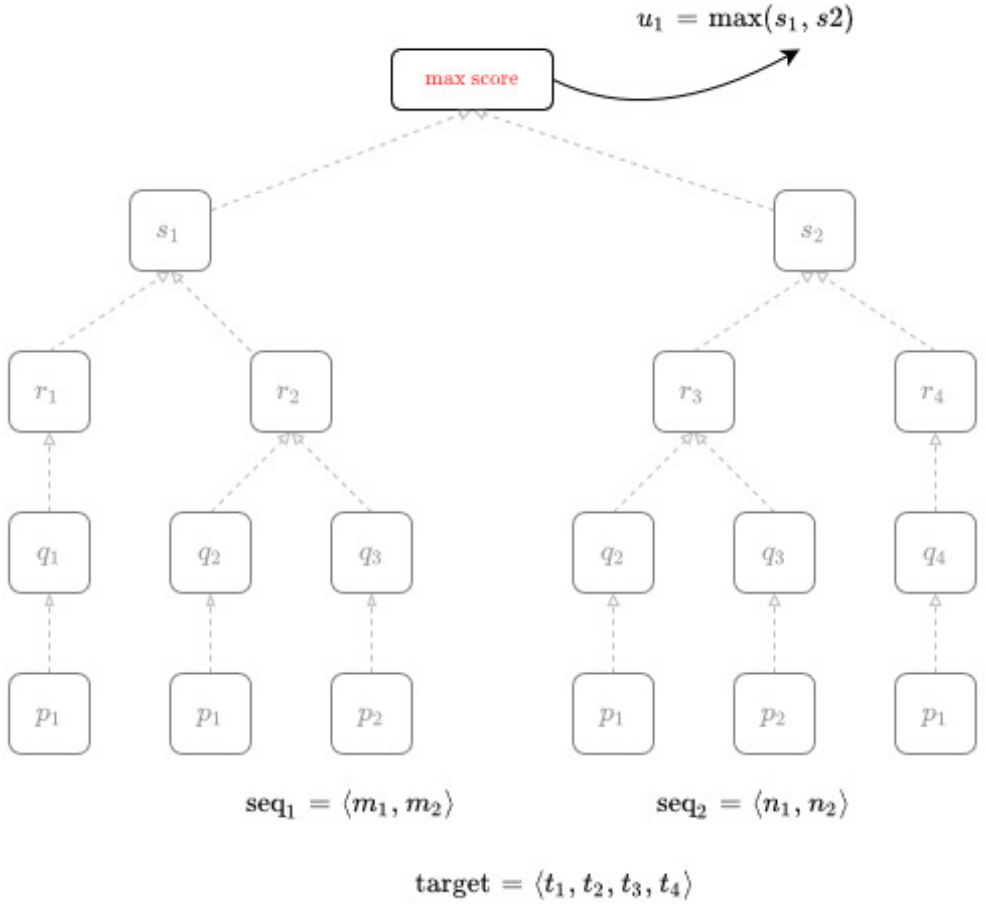
g. We travel up the tree to find the parent of the previous node.



h. We finally travel up to the root and figure out the value of each corresponding root.



i. Once the root has been calculated, we can find the maximum alignment score.



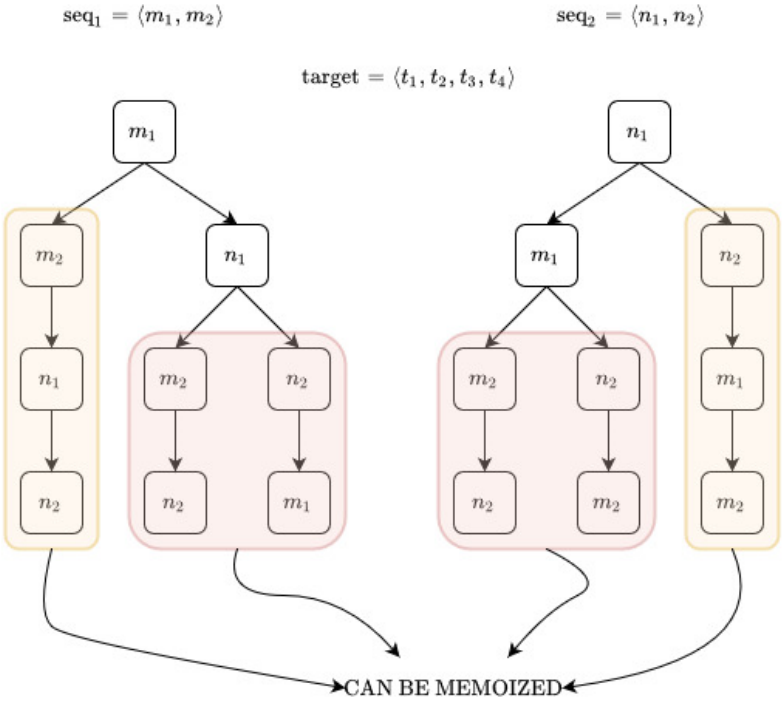
j. Once all the recursion path has been calculated, we can develop the pseudocode need to develop this recursion.

Algorithm 1 Find highest alignment score using recursion

Input: p , q input sequences
Input: t target sequence
Input: m size of p sequence
Input: n size of q sequence
Output: Best Alignment Score
Algorithm: RecursiveAS

```
1: i ← (m − 1);
2: j ← (n − 1);
3: k ← (m + n − 1);
4: if m = 0 and n = 1 then
5:     return q[0] · t[0];
6: else if m = 1 and n = 0 then
7:     return p[0] · t[0];
8: else if m = 0 and n > 1 then
9:     return RecursiveAS(p, q, t, m, n − 1) + t[k] · q[j];
10: else if n = 0 and m > 1 then ;
11:     return RecursiveAS(p, q, t, m − 1, n) + t[k] · p[i];
12: else
13:     f ← RecursiveAS(p, q, t, m, n − 1) + t[k] · q[j];
14:     g ← RecursiveAS(p, q, t, m − 1, n) + t[k] · p[i];
15:     return max(f, g);
```

k. After successfully developing the pseudocode, we can think about memoizing the data for faster recursion.



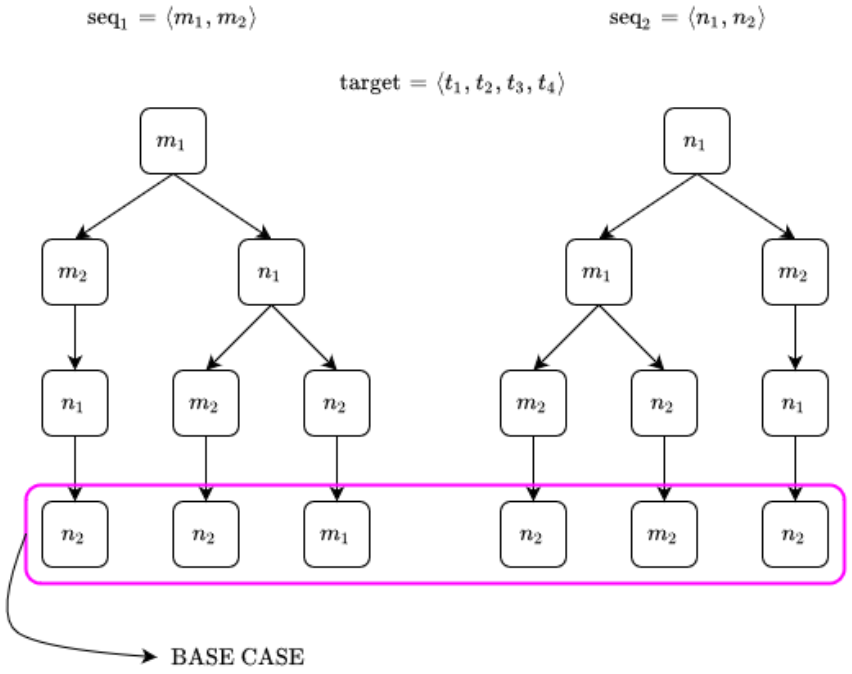
2. *What are the base cases of this recurrence?*

Given an input sequence: $a = \langle a_1, \ldots, a_m \rangle$

Given an input sequence: $b = \langle b_1, \ldots, b_n \rangle$

Given a target sequence: $t = \langle t_1, \ldots, t_{m+n} \rangle$

$$AS(a_i, b_j, t_k, m, n) = \begin{cases} a_0 \cdot t_0 & m = 0 \text{ and } n = 1 \\ b_0 \cdot t_0 & m = 1 \text{ and } n = 0 \\ f = AS(a_i, b_j, t_k, m, n-1) + t_{m+n} \cdot b_n & m = 0 \text{ and } n > 1 \\ g = AS(a_i, b_j, t_k, m-1, n) + t_{m+n} \cdot a_m & n = 0 \text{ and } m > 1 \\ \max(f, g) \end{cases}$$



3. *What data structure would you use to recognize repeated problems? You should describe both the abstract data structure as well as its implementation.*

a) To store the first input sequence, we will use a 1-D array-based map of floats, size of which will depend on the sequence.

b) To store the second input sequence, we will use a 1-D array-based map of floats, size of which will depend on the sequence.

c) To store the target sequence, we will use a 1-D array-based map of floats, size of which will depend on the sequence.

d) To store the look up table, we will use a 2-D array-based map of floats, the number of rows is the size of the first sequence and the number of columns is the size of the second sequence.

4. ***Give pseudocode for a memoized dynamic programming algorithm to find the maximum alignment score when combining *seq1* and *seq2* to align with *target*.***

Algorithm 2 Find highest alignment score using memoized recursion

Input: p , q : input sequences
Input: t : target sequence
Input: m : size of p sequence
Input: n : size of q sequence
Input: $memo$: 2D Array initialized to -2
Output: Best Alignment Score
Algorithm: MemoizedAS

```
1: i ← m − 1;
2: j ← n − 1;
3: k ← m + n − 1;
4: if memo[m, n] = −2 then
5:     if m = 0 and n = 1 then
6:         memo[m, n] ← q[0] · t[0];
7:     else if m = 1 and n = 0 then
8:         memo[m, n] ← p[0] · t[0];
9:     else if m = 0 and n > 1 then
10:        memo[m, n] ← MemoizedAS(p, q, t, m, n − 1) + t[k] · q[j];
11:    else if n = 0 and m > 1 then ;
12:        memo[m, n] ←MemoizedAS(p, q, t, m − 1, n) + t[k] · p[i];
13:    else
14:        f ← MemoizedAS(p, q, t, m, n − 1) + t[k] · q[j];
15:        g ← MemoizedAS(p, q, t, m − 1, n) + t[k] · p[i];
16:        memo[m, n] ← max(f, g)
17: return memo[m, n];
```

5. ***Give pseudocode for an iterative algorithm to find the maximum alignment score when combining seq1 and seq2 to align with target. This algorithm does not need to have a reduced space complexity relative to the memoized solution.***

Algorithm 3 Find highest alignment score using iterative dynamic programming

Input: p , q : input sequences
Input: t : target sequence
Input: m : size of p sequence
Input: n : size of q sequence
Input: $memo$: 2D Array initialized to -2
Output: Best Alignment Score
Algorithm: IterativeAS

```
1: i ← m − 1;
2: j ← n − 1;
3: k ← m + n − 1;
4: for i ← 1 to m do
5:     memo[i, 0] ← memo[i − 1, 0] + t[i − 1] · p[i − 1];
6:     for j ← 1 to n do
7:         if i = 0 then
8:             memo[0, j] ← memo[0, j − 1] + t[j − 1] · q[j − 1];
9:         f ← memo[i − 1, j] + t[i + j − 1] · p[i − 1]
10:        g ← memo[i, j − 1] + t[i + j − 1] · q[j − 1]
11:        memo[i, j] ← max(f, g)
12: return memo[m, n];
```

6. ***Can the space complexity of the iterative algorithm be improved relative to the memoized algorithm? Justify your answer.***

Currently, our iterative and memoized code have the same space complexity as they both use every cell in a 2D array of floats to calculate the maximum alignment score. If we want to reduce the space complexity, we can use C++ map STL implementation to store information of the cells that are only necessary to calculate the alignment score. However, while this can save on space complexity, our time complexity will be increased because additional lines of codes will be deployed to remove the unnecessary information.

7. ***Give pseudocode for an algorithm that identifies the sequence which will achieve the maximum alignment score. If there is more than one sequence that yields the maximum score, you may return any such sequence. Your algorithm may be iterative or recursive.***

Algorithm 4 Find sub-sequence with the highest alignment score

Input: p , q : input sequences
Input: t : target sequence
Input: m : size of p sequence
Input: n : size of q sequence
Input: $memo$: 2D Array initialized to -2
Output: Array of sub-sequence with highest alignment score
Algorithm: BestSeq

```
1: i ← m;
2: j ← n;
3: k ← m + n − 1;
4: bestArr = Array[m + n];
5: while i ≥ 0 do
6:     f ← memo[i, j] − p[i − 1] · t[k]
7:     g ← memo[i, j] − q[j − 1] · t[k]
8:     if (i = 0 and j > 0) or (g = memo[i, j − 1]) then
9:         j ← j − 1;
10:        bestArr[k] ← q[j];
11:        k ← k − 1;
12:    else if (j = 0 and i > 0) or (f = memo[i − 1, j]) then
13:        i ← i − 1;
14:        bestArr[k] ← p[i];
15:        k ← k − 1;
16:    else
17:        throw Error;
18: return bestArr;
```