ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 336E

ANALYSIS OF ALGORITHMS II
PROJECT 3 REPORT

DATE : 17.05.2021

STUDENT:

NAME : ABDULKADİR

SURNAME : PAZAR

NUMBER : 150180028

SPRING 2021

Question 1

```
SW(s1,s2)
    max index array
    result set
    max score = 0
    n = s1.size
    m = s2.size
    matrix[n+1][m+1] (stores index value and i,j for traceback)
    for(i from 0 to n + 1)
        for(j from 0 to m + 1)
            if i or j = 0
                matrix(i,j) = 0
            else
                similarity(a(i-1),b(j-1))
                matrix(i,j) = max(matrix(i-1,j-1) + similarity,
                matrix(i-1,j) + gap, matrix(i,j-1) + gap, 0)
    for(i from 0 to n + 1)
        for(j from 0 to m + 1)
            if(matrix(i,j) > max score)
                max score = matrix(i,j)
    for(i from 0 to n + 1)
        for(j from 0 to m + 1)
            if(matrix(i,j) = max score and max score is not 0)
                add i,j to max index array
    for(indexes in max index array)
        s = ""
        while(matrix(index) != 0)
        s += s1(index - 1)
        index = index[traceback]
        reverse s
        add s to result set
```

Complexity is equal to $O(mn)$ where m and and n are lengths of strings since the longest operations in the pseudo code are the for loops for filling the matrix, calculating the max score, finding max indexes and it has $(n+1)(m+1)$ steps and that multiplication is equivalent to $O(mn)$.

Question 2

a)

To calculate the number of calculations when assessing all possible alignments one by one, we first calculate find the number of subsequences with lengths ranging from 1,2,..n-1. This takes $C(n,0) + C(n,1) + C(n,2) + \ldots C(n,n) = 2^n$ calculations. To check if a subsequence is common in both strings takes around n calculations. Compared to my algorithm which has 3 for loops with $(n+1)(m+1)$ steps brute force has too many calculations.

b)

In the dynamic programming approach because of memoization we keep nm results in memory stored in the matrix for future use. In the brute force approach, since there is no memoization the calculation results aren't kept in the memory but the program itself consumes a lot of memory because of number of recursion steps.

c)

To find out the complexity of brute force approach, we need to first know the number of possible different subsequences of a string with length n, (find the number of subsequences with lengths ranging from 1,2,..n-1.). Since the number of combinations with 1 element are $C(n,1)$. Number of combinations with 2 elements are $C(n,2)$ and so on. We know that $C(n,0) + C(n,1) + C(n,2) + \ldots C(n,n) = 2^n$. So a string of length n has $2^n - 1$ different possible subsequences since we do not consider the subsequence with length 0. So time complexity of the brute force is $O(n2^n)$. It takes $O(n)$ time to check if a subsequence is common to both the strings. Running time of my solution is $O(mn)$ which is a much better result compared to assessing all possible alignments one by one.