Problem 1 (LCS with k strings) **YASIN AYDIN**

(a) In the problem of LCS with 2 string, we were using 2D matrix to store the values. 2 dimension was enough for that problem because putting character of one string as rows and other one's as columns were sufficient. In the case of k strings, we will be needing k-dimension to store these values.

We also have to change the algorithm's recursive part. So, if we have k strings as $s_0, s_1, s_2, ...s_k$ and the longest common sub-string is $L(i_0, i_1, i_2, ..., s_k)$

Then we can change the recursive part as following:

$$\begin{aligned} L(i_0, i_1, i_2, ..., s_k) = max\{&L(i_0 - 1, i_1, i_2, ..., s_k), \\ &L(i_0, i_1 - 1, i_2, ..., s_k) \\ &L(i_0, i_1, i_2 - 1, ..., s_k) \\ &... \\ &L(i_0, i_1, i_2, ..., s_k - 1)\} \end{aligned}$$

As we did in the 2 string LCS, we will check if $L(i_0 - 1, i_1 - 1, i_2 - 1, ..., s_k - 1)$ exists and if it exist we will add 1 to that. If it doesn't exist, we calculate the recursive formulation above and store it. As we see, we will be checking k longest common sequences for each sub-problem. These checks cost constant time because of Memoization.

(b) Let N be the longest string out of k strings.

1. The complexity for solving a subproblem is $O(k)$ because for each cell there are k checks to be made and they cost constant time.
2. We will be also checking if the current letters of k dimension match which is $O(k)$.
3. The complexity for filling the k dimension will be $O(N^k)$ since we have k-dimension with N characters.

So, the total asymptotic time complexity is $O(N^k) * (O(k) * O(k)) = O(N^k * k)$
This was expected because the number of inputs are increased and the number of checks are increased. The problem is exponential in terms of k.

Problem 2 (BONUS) **YASIN AYDIN**

(a) **Input:** A sequence of cities $C = c_0, c_1, ...c_n$
A sequence of cities on the way to the destination city $S_c = s_0, s_1, s_2, ...$ where c is a city from the C.
Output: One I_c such that the total itinerary time for visiting all cities in C and popular touristic places on the way to the cities in C starting from Istanbul.

(b) Let our problem be T.

1. T is NP:
In this problem at each city (vertex) we have polynomially number of options to choose from so this is an non-deterministic polynomial problem. It's solutions can be verified in polynomial time.
2. We will be using Hamiltonian Cycle Problem as H which is known to be NP-Complete. We choose this problem because it is similar to our problem in a way that we also have to visit all vertices(cities) once and we also have to do this in minimum cost way. We will reduce H to T. H is NP-complete so it is NP-Hard and every problem y in NP can be reduced to H in polynomial time. If we reduce H to T in polynomial time, then we will have shown that every y in NP reduces to T is polynomial time.
3. Showing that H can be transformed/reduced into T in polynomial time:
We will use the decision version of T which is "is there a T of cost at most k?". We reduce Hamiltonian cycle to H to T. Given G(V, E) with $|V| = n$ we define c(e) = 1 for all $e \in E$. Then we add edges E' to G to make G a complete graph and assign c(e) = 2 for all $e \in E'$. We can do this in polynomial time. Now given the cost, if the answer to the question is "yes", then we know there is a cycle that visits all cities exactly once. The edges it uses from E, hence we can say that there is H in G. Given that H is NP-Complete, T is NP-Complete too. Our problem asks for the shortest itinerary from Istanbul to C so we can the decision version of T to find the shortest itinerary by lowering the k. So, the problem T is NP-Complete.

   Since we know that H is NP-Complete, for any problem $H' \in NP$, we must have $H' \rightsquigarrow H$. We also know that $H \rightsquigarrow T$, hence $H' \rightsquigarrow H \rightsquigarrow T$ implies $H' \rightsquigarrow T$. That is, for any problem $H' \in NP$, we have $H' \rightsquigarrow T$. Since also $T \in NP$, T is an NP-Complete problem.