Handson 3 - Anna Francesca Montagnoli

Exercise 1

I implemented the requested method by using dynamic programming. I built a matrix (n+1)*(D+1), so row i refers to the city i and column j refers to the days of vacation left.

I also built a vector of size n*D containing the values of the cities: since the i-th day of a city can be visited iff all the (i-1) days were visited the vector has D values for each city which contain the sum of the values up to the i-th day of that city.

For every cell (i, j) in the matrix the program spends time D to compute the value, because I can decide to:

- **Select city i:** for x in 1..d+1 we check every (j - x) column in the previous row and add the value in the vector corrisponding to the sum of x days in city i.
  matrix[i][j] = maximum between all those values.

- **Not select city i:** matrix[i][j] = matrix[i-1][j]

Time complexity: $O(n*D^2)$ because the matrix has n*D dimension and for every cell we spend D time to compute the value.

Exercise 2

The first thing that the method I implemented does is sorting the vector containing the topics by increasing beauty. Then it is applied the Longest Increasing Subsequence on the difficulty of the topics.
I followed the solution explained here: https://www.geeksforgeeks.org/longest-monotonically-increasing-subsequence-size-n-log-n/

For LIS I used a vector ”res” to keep track of the results.

For each topic in the sorted vector, we perform the following steps:

- If the difficulty is greater than the one of the last element in res (the largest element in the current subsequence), we append the number to the end of the list. This indicates that we have found a longer subsequence.

- Otherwise, we perform a binary search on res to find the smallest element that is greater than or equal to the current number.

- Once we find the position to update, we replace that element with the current number. This ensures that we have the potential for a longer subsequence in the future.

Time complexity: O(nlogn).