3) I structured the code with error handling like if the string contains numbers and or any characters but alphabets it throws an error message saying, Invalid input and if the empty string is returned , It says, “Given string has no repeated characters”.

4)

Brute-force:

I implemented the LRS problem with optimal substructure which is as follows:

$$T[i,j] = \begin{cases} 0 \text{ if } i=0 \text{ or } j=0, \\ T[i-1, j-1]+1 \text{ if } x_i = x_j \text{ and } i \mathrel{!}= j, \\ \max(T[i-1, j], T[i, j-1]) \text{ otherwise.} \end{cases}$$

The answer is T[n,n].

Dynamic:

So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again. In our case, when the length of string ‘str’ is 5, it will have a recursion tree of depth 5 considering it has all distinct characters. Since we have the overlapping allowed we might get the same subproblem computed again and again. To optimize we can save the solution in memory for reuse (memoization).

5)

The worst case time complexity of brute.py program is **$O(n^2)$** and the auxiliary space used by the program brute.py is $O(n^2)$ since we used a 2D array of size twice the length of the string ‘str’.

The worst case time complexity of dynamic.py program is **$O(2^n)$** and the auxiliary space used by the program dynamic.py is $O(n^2)$ since two recursive calls happens for N times. The worst case happens when there is no repeated character present in string str(when length becomes 0) and each recursive call will end up in two recursive calls.

Dynamic Programming Vs Brute-force:

A dynamic programming algorithm will examine all possible ways to solve the problem and will pick the best solution. Therefore, we can roughly think of dynamic programming as an **intelligent, brute-force method that enables us to**

go through all possible solutions to pick the best one. If the scope of the problem is such that going through all possible solutions is possible and fast enough, dynamic programming guarantees finding the optimal solution.

6) References:

https://cs.stackexchange.com/questions/51065/finding-the-longest-repeating-subsequence