Programming Assignment 2 Report

1) Brute Force Implementation

As we know, brute force is an exhaustive search method by checking all possible cases in the problem. I have implemented a characterCount function, which finds the frequency of character repeated in the string.

```
#counting the times that letter present in input
def characterCount(characterList, key):
    count = 0
    for character in characterList:
        if (character == key):
            count = count + 1
    return count
```

If the character count is more than 1 and if it is not existing in the repeated characters list, then the character is added to the repeated list. This repeated characters list holds the sequence of the longest repeated subsequence of the given string.

```
#Finding the repeated characters in the sequence through iteration
for i in range(length):
    characterFrequency = characterCount(inputString,inputString[i])
    if(characterFrequency > 1):
        if inputString[i] not in repeatedCharacters:
            repeatedCharacters.append(inputString[i])
```

Dynamic Programming Implementation

We implement the optimal substructure. Here we build a L[length+1][length+1] array to hold the length of all possible subsequence of the string. We perform the checking in bottom up approach. We compare each character of the string with entire string in each iteration. If the characters match, the diagonal cell value is incremented by 1, otherwise the maximum of top and left cell value is counted. Here, we do not allow overlapping indexes of characters.

```
for i in range(1, length + 1):
    flag = 0
    for j in range(1, length + 1):
        #When the last characters match,the diagonal cell value is incremented by 1
        if (inputString[i - 1] == inputString[j - 1] and i != j and flag == 0):
            L[i][j] = L[i - 1][j - 1] + 1
            flag=1
        #When the last characters does not match,maximum value among the top and left cell is taken
        else:
            L[i][j] = max(L[i - 1][j], L[i][j - 1])
return L
```

Now L contains the length of LRS. Using this array we find the subsequence by traversing from the bottom of array and checking the length obtained is equal to the L[i-1][j-1]+1 or max(L[i-1][j],L[i][j-1].

2) The worst case **time complexity of LRSbrute.py is O(n^2)** as inside the main for loop,a function character count is called, which has one more for loop. **Space complexity is O(n)** as only 2 single dimension arrays are used to store the frequency count of character and the repeated characters.
The worst case **time complexity of LRSdynamic.py is O(n^2)** as we iterate through two loops. **Space complexity is O(n^2)** as we use a 2 dimension array L[i][j]

Dynamic programming is typically **faster** than Brute Force. It checks for all possible solutions to choose the best one. Dynamic programming **reuses computation**, whereas Brute Force doesn't reuse. The former uses the previously solved solution to work on the current solution, hence it is faster. Even in our program, the values are obtained from the previous rows and column computation. Thereby, dynamic programming is much better than brute force method.a

3) References

The following links were used as references.

- Class Slides:http://crystal.uta.edu/~gonzalez/alg_spring_2019-05.html
- https://stackoverflow.com/questions/1065433/what-is-dynamic-programming
- https://www.geeksforgeeks.org/
- https://www.youtube.com/watch?v=sSno9rV8Rhg