# Longest Repeated Subsequence Report

By Avinash Shanker 1001668570

## Brute-Force Algorithm:

Brute-Force algorithm is implemented as a simple solution to generate all subsequences in a recursive loop fashion. The below recursive approach is called brute force as we compute all possible combinations then choose the maximum from it.

*Below is the code snippet which is Heart of my LRS brute-force Code:*

```
def Longest_Repeated_SubSeq_Brute_Force(xlist, ylist):
    if not xlist or not ylist:
        return []
    x, xs, y, ys = xlist[0], xlist[1:], ylist[0], ylist[1:]
    if x == y and xs != ys:
        return [x] + Longest_Repeated_SubSeq_Brute_Force(xs, ys)
    else:
        return max(Longest_Repeated_SubSeq_Brute_Force(xlist, ys), Longest_Repeated_SubSeq_Brute_Force(xs,ylist), key=len)
```

In brief the function *Longest_Repeated_SubSeq_Brute_Force(Seq_A, Seq_B)* takes 2 inputs of the **same** string as *Seq_A* and *Seq_B* and then recursively splits them compares each element in Seq_A with that in Seq_B. If both the elements match and their indexes don't overlap, then it is stored in a list in the same order. This is done for all possible subsequence and then the max of this returned.

This solution is **exponential** in terms of time complexity, as a string of **n** characters has $\mathbf{2^n}$ subsequences. Post which testing if a sequence whether or not is a subsequence takes O(m) time. Thus, the brute force algorithm would take $O(m2^n)$ time.

## Dynamic Algorithm:

In the brute-force approach, we can easily notice that we recursively repeat multiple subsequences which increases the time complexity exponentially by computing the same again and again.

This can be avoided in dynamic programming approach my storing these values in the form of matrix which is called memoization or tabulation. We also note that LRS has an optimal substructure.

Let Longest_Rep_Seq(Input_Seq[0..n-1], Input_Seq[0..n-1]) be the length two same sequences given to our function.

*Below is the code snippet which is Heart of my LRS dynamic Code:*

```
for x_axis in range(1, String_Length + 1):
    for y_axis in range(1, String_Length + 1):
        if (Input_Sequence[x_axis-1] == Input_Sequence[y_axis-1] and x_axis != y_axis):
            Sequence_Array[x_axis][y_axis] = 1 + Sequence_Array[x_axis-1][y_axis-1]
        else:
            Sequence_Array[x_axis][y_axis] = max(Sequence_Array[x_axis][y_axis-1], Sequence_Array[x_axis-1][y_axis])
```

If last characters of both sequences match (Input_Seq[n-1] == Input_Seq[n-1]) then,

Longest_Rep_Seq(Input_Seq[0..n-1], Input_Seq[0..n-1]) = 1 + Longest_Rep_Seq(Input_Seq[0..n-2], Input_Seq[0..n-2])

If last characters of both sequences do not match and the indexes do not overlap,

(Input_Seq[n-1] != Input_Seq[m-1]) then,

Longest_Rep_Seq(Input_Seq[0..n-1], Input_Seq[0..n-1]) = MAX ( Longest_Rep_Seq(Input_Seq[0..n-2], Input_Seq[0..n-1]), Longest_Rep_Seq(Input_Seq[0..n-1], Input_Seq[0..n-2]) )

## Complexity and Contrast Analysis

| Brute-Force LRS Algorithm | Dynamic LRS Algorithm |
|---|---|
| Sample subsequence is solved multiple times in recursion | Memoization approach is used i.e. storing in table which reduces time complexity |
| Time Complexity of $O(m*2^n)$ | Time Complexity of O(m*n) |
| Space Complexity O(n*m) as the depth of tree $2^n$ | Space Complexity of O(m+n) |

## Conclusions:

1. It can be clearly noted that Dynamic approach of Longest Repeated subsequence (LRS) has much lower time complexity when compared to brute force which has exponential complexity.
2. Brute force algorithm may work fine for lower inputs but, the performance of the algorithm will take a very bad hit for huge input sequence.
3. Space complexity of both the algorithms are comparable, but brute force takes more space as compared.
4. When compared to coding of iteration complexity to implement, brute force is much easier and quicker than Dynamic approach.

Test Output Of My Code:

```
PS D:\UTA\Semester 1\DAA\Code> python dynamic.py ATACTCGGA
Longest Repeated Subsequence Using Dynamic Approach: ATCG
PS D:\UTA\Semester 1\DAA\Code> python brute.py ATACTCGGA
Longest Repeated Subsequence using Brute-Force Approach: ATCG
```

## References:


[1]https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/

[2]http://www.mathcs.emory.edu/~cheung/Courses/323/Syllabus/DynProg/LCS-2.html

[3]http://www.techiedelight.com/longest-repeated-subsequence-problem/

[4] http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node4.html