Q.8

```
// let X[1...k] and Y[1...n] where k≤n be two global arrays
// def length() : returns the length of the longest subsequence
                  of Y that is not a super-sequence of X.

def length() :
    ~~temp = ""~~        ~~// temp is null or empty string~~
    result = longestSubsequence (1, Y) ~~temp~~)
    return result. length

// def longestSubsequence () : returns the longest subsequence of Y[1...n]
                               which is not a supersequence of X[1..k]

def longestSubsequence (index, Y, ~~temp~~) :
    if (index > Y. length) :
        return ""        // returning empty string
    t = Y[index] + longestSubsequence (index+1, Y, ~~temp~~)
    nt = longestSubsequence (index + 1, Y, ~~temp~~)
    subSeqA = ""    } // initializing two variables as
    subSeqB = ""    }    empty strings
    if (chkSubSeq (X, t, X.length, t.length) == FALSE)
        subSeqA = t
    if (chkSubSeq (X, nt, X. length, nt. length) == FALSE)
        subSeqB = nt
    k = max (subSeqA, subSeqB)
    return k

// def chkSubSeq () : returns True if X is subsequence of str,
                      else False.

def chkSubSeq ( X, str, m, p):
    if (m ==0): return TRUE
    if (p ==0): return FALSE
    if (X[m-1] == str[p-1]) :
        return chkSubSeq (X, str, m-1, p-1)
    return chkSubSeq (X, str, m, p-1)
```

Analysis

// let $T(n)$ be the time complexity of length()

// let $L(n)$ be the time complexity of longestSubsequence()

// let ~~$C(n)$~~ $C(i,j)$ be the time complexity of chkSubSeq()

$$C(i,j) = \max\{T(i-1, j-1), T(i, j-1)\} + c$$

$C(1,1) = d$ // base case

claim $C(i,j) = c(i+j) + d$

Thus, complexity of chkSubSeq() is $C(m,p) = c(m+p) + d$

here $m < p$, so $C(m,p) \cong C(p,p) = 2cp + d = O(p)$

(here p is length of str)

$$L(i) = 2L(i+1) + O(1)$$

let, $n - i = i' \Rightarrow i = n - i'$

$$L(n-i') = 2L(n-i'+1) + c$$

let, $S(i') = 2S(i'-1) + c$

$$= 2\{2S(i'-2) + c\} + c$$

$$= 2^2 S(i'-2) + c + 2c$$

$$= 2^3 S(i'-3) + c + 2c + 2^2c$$

$$\vdots$$

$$= 2^k S(i'-k) + (c + 2c + 2^2c + \cdots + 2^{k-1}c)$$

let $i' - k = 0$

$$S(i') = 2^k S(0) + c * \frac{2^k - 1}{2-1}$$

$$= 2^k * 1 + c * (2^k - 1)$$

$$S(i') \cong O(2^{i'})$$

$$L(1) = S(n-1) \le S(n)$$

$S(n) \cong O(2^n)$ (here n is length of y)

Now, $T(n) = L(n) + c$

$T(n) = O(2^n)$ (here n is length of y)

Explaination :

in the chkSubSeq() method:

we start matching the characters of two strings X and str from the last character and approach towards the begining recursively. If the last two characters of X and str matches then we reduce both the string lengths by 1 unit and recursively call chkSubSeq(). If they don't match, then only reduce the length of str by 1 and recursively call chkSubSeq(), (as shown in line 5), because we need to find out if all characters of X is present in str as a subsequence or not.

in longestSubsequence () method :

Base case: if index > Y. length, i.e., There exist no subsequence of Y, we return a null string. (shown in line 2)

For each index of Y we take two cases :

(1) it is a part of the subsequence

(2) it is not a part of the subsequence

case 1: if Y[index] is part of the subsequence then we add ~~to~~ it to string 't' and recursively call longestSubsequence (index+1, Y) (as shown in line 3), where the call returns the longest subsequence of Y[index+1 ... n] which is not a supersequence of X[1...k].

case 2 : if Y[index] is not a part of the subsequence then we simply recursively call longestSubsequence (index+1, Y) (as shown in line 4), and store it in "nt", where the call returns the longest ~~sed~~ subsequence of Y[index+1 ... n] which is not a supersequence of X[1...k].

After the two possibilities have been explored and ~~sta~~ subsequences of Y gets stared in 't' and "nt", we have to return the longest between these two and which is not a supersequence of X.

Thus we first check whether X is subsequence of 't' and "rt" in line 7 and 8 respectively, and then we store these two strings in subSeqA and subSeqB only if X is not a subsequence. Then return maximum of subSeqA and subSeqB, ensuring that neither of them are supersequence of X, in line 10.

in length() method :

From this method we make the initial call to longestSubsequence (1, Y), which returns the longest subsequence of $Y[1 \ldots n]$ which is not a supersequence of $X[1 \ldots \underline{k}]$, and store that resultant string in variable "result". Then we simply return the length of string result which is desired.