Loretta Chan
CS170-VA
HW9

Main idea: My algorithm uses 3 ideas we've discussed in class - greedy algorithm, dynamic programming and memoization. We start with the longest substring and store in the array *usedStrings*, then at each step, we look for a substring that that has the biggest overlap with any of the strings in usedStrings, then concatenate with these strings. We use LCS (longest common substring) algorithm to look for the biggest overlap dynamically, and memoize the result that it returns for later use. If the string is a substring of the previous strings (or has been previously looked at), then we do nothing. Repeat until all relevant strings are concatenated.

Runtime: Each call of LCS takes $O(k^2)$ time and and we call it on each pair of substrings exactly once. Greedy algorithm iterates through all possible pairs, and the total number of pairs is approximately n^2 so overall runtime is $O(n^2k^2)$

Pseudocode:

```
procedure longestCommonSubstring (s1, s2): @memoized
LCSuff = [[0 for x in xrange[len(s2)] for x in xrange(s1)]
z = 0
for i in range (1, len(s1)):
    for j in range (1, len(s2)):
      if s[i] == t[j]:
        LCSuff[i][j] = LCSuff[i-1][j-1] + 1
        if LCSuff[i][j] > z:
          z = LCSuff[i][j]
          indexS1 = i
           indexS2 = j
     else:
       LCSuff[i][j] = 0
if z != 0:
  result = z+1
  if indexS2 > indexS2:
     resultString = t[:indexS2 - z]
  else:
     resultString = t[indexS2+1:]
return (result, resultString)

procedure DNASequencing (substrings):
currString = s with max len(s) for all s in substrings
biggestOverlapLength = 0
usedStrings = [currString]
for all s in substrings:
          DNAvisited[s] = False
i, j = 0
while i < len(substrings):
  while j < len(substrings):
    if DNAvisited[substrings[j]] == False:
```

```
      for s in usedStrings:
        result = longestCommonSubstring(s, substrings[j])
        currOverlapLength= result[0]
        stringToAppend = result[1]
        if currOverlapLength > biggestOverlapLength | ((currOverLapLength == biggestOverlapLength) &
(len(substrings[j] > len(finalStringToAppend))):
          finalStringToAppend = stringToAppend
          biggestOverlapLength = currOverlapLength
    j+=1
    concatenate currString, finalStringToAppend
  i+=1
return currString
```