```
algorithm permu(longStr, shortStr, leftIndex)
      Input: longStr that can contain shortStr
      Input: shortStr where the permutations are going to be checked in longStr
      Output: printed whether permutation is found in longString and where it's
located

            if leftIndex == shortStr.length()

                  // print the permutation
                  Print(shortStr)

                  foundIndex <- longStr.indexOf(shortStr)

                  if foundIndex is not -1(if shortStr has been found in the
longStr)
                        Print("Found one match: " $shortStr " is in " $longStr " at
location " $foundIndex)

                  else
                        for i <- leftIndex to shortStr.length()
                              swapped <- swap(shortStr,leftIndex,i)
                              permu(longStr,swapped,leftIndex + 1)
end

algorithm swap(a,i,j)
      Input:  str - string that will have two of its characters swapped
      Input: I - first swapped index in the string
      Input: J - second swapped index in the string

            charArray <- str to an array

            temp <- charArray[i]

            charArray[i] = charArray[j]
            charArray[j] = temp
            return charArray

end
```

Explanation:
The algorithm has the acceptable complexity of O(n!).
That's the amount of calculations it takes to go through an n-lengthed string's permutations, so there isn't a lower complexity this can take.
The program is not scalable, as working on a small set at 10 characters or less works,
but once you go above 10, the time takes much too long for a computer to go through all of the permutations.