Philippe Carrier 40153985

Assignment 1 – Programming questions

Version 1:

```
Algorithm permu(shortStr, longStr, permutation)
    Input shortStr: random number of random character smaller in size then longStr
    Input longStr: random character with size bigger than shortStr
    Input permutation: final version of shortStr

    if shortStr.isEmpty() then
        S.O.P(permutation)
        findOccurrences(permutation, longStr)

    for i = 0 to shortStr.length – 1 do
        permu(shortStr.substring(0, i) + shortStr.substring(i+1),
            longStr,
            permutation + shortStr.charAt(i))


Algorithm findOccurrences(permutation, longStr)
    Input permutation: set of random character
    Input longStr: random character with size bigger than shortStr

    len = permutation.length
    limit = longStr.length – len

    for i = 0 to limit do
        if permutation.equals(longStr.substring(i, i + len)) then
            S.O.P("Found one match: " + permutation + " is in " + longStr + " at
            location " + i)
            Break
```

The permu algorithm is called n! to create all the permutation of the shortStr. Inside the for loop, there is a string concatenation that time complexity is O(n). Hence, without considering the findOccurrences, the time complexity of permu is O(n*n!). The algorithm findOccurrences time complexity is $O(n^2)$ since the for loop runs for n – b where n is the length of the longStr and b is the length of the permutation and b < n. Also, the substring time complexity is O(n) which make the whole process $O(n^2)$. The whole algorithm is then $O(n^3 * n!)$. This complexity is not acceptable since it's not scalable as the longer the shortStr is the greater the time it will take.

Note: The number of iteration made in Version 1 was limited because the time it was taking to iterate was getting out of hand.

Version 2:

```
Algorithm permuNew(shortStr, longStr)
        Input shortStr: random number of random character smaller in size then longStr
        Input longStr: random character with size bigger than shortStr

        len = shortStr.length()
        limit = longStr.length() – len
        for i  = 0 to limit do
                perms = StringBuilder(shortStr)
                isPresent = true
                for j = i to i + len do
                        letter = longStr.charAt(j);
                        index = perms.indexOf(letter);
                        if index < 0 then
                                isPresent = false
                                break;
                        perms.deleteCharAt(index);
                if isPresent then
                        S.O.P("Found one match: " + longStr.substring(i, i + len) + " is in " + longStr
                        + " at location " + i;
```

The algorithm takes a segment of the longStr with length equal to the length of the shortStr and verify one letter at a time that the segment could be form by the permutation of the shortStr. Time Complexity: the outer for loop runs (n – b) times where n is the length of the longStr and b is the length of shortStr and n > b. The inner for loop runs b times. Without considering the operation that happens at a constant time, the function for this algorithm runs (n-b)*b = bn – b^2 <= bn <= n^2. Therefore, the time complexity of my algorithm is O(n^2).