ex1 - Lior Ziv

Q1.a

I added another property which is not mentioned in your description, suffTree{i}{4} will hold the leaf index or and empty cell if it's an internal node.

Q1.c.2

The degree is 2, The time complexity is polynomial O(n^2)

Q1.c.3

The actual time was = 43.9 seconds, the predicted time was 48.85 seconds.

Q3.a

- I would suggest an algorithm that allows at most K mismatches.

  For a given read(R) and a sequence it builds a joint suffix tree and than work as follows:

  Find the largest common part(LCA) - lowest common ancestor, for each suffix of the sequence and R.

  After finding the LCA the strings split which means there is one mismatch(the next letter most be different)

  The next move will be taking R{splitPoint + 1:end} ,suffix{{splitPoint + 1:end} and start again running on the suffix tree from the root.

  For every split we add one mismatch and start again until r is empty. At the end we check if the amount of mismatches < K and only those suffixes are saved.

pseudo code

```
• k-mismatches(r,sequence,k):
  ST = initialize(sequence,r) one tree which contains both words
  result = {}
    – for each suffix in sequence:
        ∗ res = k-mismatchHelper(r,sequence,counter,k)
        ∗ if(~isempty(res)) , add res to results
• k-mismatchHelper(r,sequence,counter)
    – if(isempty(r)) - return
    – splitPoint = find LCA(suffix,r)
  – r=r{splitPoint + 1:end} ,s=suffix{{splitPoint + 1:end}
  – raise mismatch counter by 1
  – counter = k-mismatchHelper(r,sequence,counter)
  – if(counter < k)
      ∗ return (index ,index + length(r) -1)
  – return empty cell
```

Q3.b

- The algorithm build a suffix tree for the sequence and the read r → O(length(sequence)+ length(r)), now the algorithm runs on each suffix with the read and might have k(up to r length) mismatches → O(length(r))

  combined the total **time complexity** is O(length(sequence)*length(r))

- The algorithm build a suffix tree for the sequence and the read r → O(length(sequence)+ length(r)) and for each suffix has to save at most an array with two indexes to save in results.

  combined the total **space complexity** is O(length(sequence) + length(r))