1. [3+3+3+3 points] Each of the following pairs of concepts are somehow connected. Describe the main connecting factors or commonalities as well as the main separating factors or differences.
   (a) (Knuth–)Morris–Pratt algorithm and Shift-and algorithm.
   (b) String quicksort and ternary tree.
   (c) String binary search and backward search.
   (d) Compact trie and suffix tree.

A few lines for each part is sufficient.

2. [6+6 points] Let \( S[0..n] \) be a string over an alphabet \( \Sigma \) of size \( \sigma \). A \( q \)-gram is a string of length \( q \). Let \( \text{occ}[0..\sigma^q] \) be an array that for each \( q \)-gram \( Q \), in lexicographic order, tells how many times \( Q \) occurs in \( S \). For example, if \( \Sigma = \{a, b, c\} \), \( q = 2 \) and \( S = \text{acbbbbc} \), then \( \text{occ} = (0, 0, 2, 1, 3, 0, 0, 1, 0) \). Describe algorithms for computing \( \text{occ} \) given \( q \) and \( S \):
   (a) \( O(n + \sigma^q) \) time algorithm for integer alphabet \( [0..\sigma) \).
   (b) \( O(n \log \sigma + \sigma^q) \) time algorithm for an ordered alphabet.

In both cases, prove the time complexity.

3. [6+6 points]
   (a) Compute the edit distance between strings \( \text{tukholma} \) and \( \text{stockholm} \) using the dynamic programming algorithm described on the course.
   (b) Give all optimal alignments between \( \text{tukholma} \) and \( \text{stockholm} \), i.e., alignments with the same cost as the edit distance.

4. [6+6 points] Let \( A, B, B' \) and \( C \) be strings such that \( A \leq B \leq C \) and \( A \leq B' \leq C \).
   (a) Prove that \( \text{lcp}(B, B') \geq \text{lcp}(A, C) \). You may assume only basic definitions from the course to be known, i.e., do not use any lemmas or theorems from the course.
   (b) Describe how the above result can be used to speed up string binary searching.

5. [12 points] The reverse of the string \( A = a_1a_2 \ldots a_m \) is the string \( A^R = a_m \ldots a_1 \). Describe an algorithm that, given two strings \( S \) and \( T \), finds the shortest string \( X \) such that \( X \) occurs in \( S \) but neither \( X \) nor \( X^R \) occurs in \( T \). The time complexity should be linear on a constant size alphabet.