Tutorial Week 3

Definition 1. *A string is a* **border** *of another string, if the two are different and the former occurs both as a prefix and as a suffix for the later. For a string w, an integer p with $0 < p \leq |w|$ is a* **period** *of w if w has a border of length $|w| - p$.*

Exercise 1. *Consider the following algorithm calculating the length of the longest borders of all prefixes of a string x of length m.*

Algorithm 1 Compute borders(string x; integer m)

```
1: MP_next[0] = −1
2: for i = 0 to m − 1 do
3:     j = MP_next[i].
4:     while j ≥ 0 and x[i] ≠ x[j] do
5:         j = MP_next[j]
6:     end while
7:     MP_next[i + 1] = j + 1
8: end for
9: return MP_next
```

Fill up the values in the following table of borders of prefixes, for each of the strings.

strings	0	1	2	3	4	5	6	7	8	9	10
ababa											
abcacabb											
abcacababc											
abacabacab											

What is the complexity of the algorithm?

Exercise 2. *Consider the Morris-Pratt search algorithm, that finds a pattern x of length m in a text y of length n. Consider for this, the pattern $x = aba$ and the text $y = abcacacabac$.*

Algorithm 2 MP(string x, y; integer m, n)

```
1: i = 0, j = 0
2: while j < n do
3:    while (i == m) or (i ≥ 0 and x[i] ≠ y[j]) do
4:       i = MP_next[i]
5:    end while
6:    i = i + 1
7:    j = j + 1
8:    if i == m then
9:       output: x 'occurs in' y 'at position' j − i
10:   end if
11: end while
```

a) *Complete the following table concerning the Preprocessing phase.*

strings	0	1	2	3
$x[i]$	a	b	a	
MP_{next}				

b) *Complete the following table concerning the Searching phase.*

j	0	1	2	3	4	5	6	7	8	9	10
i											

c) *What is the actual value that the algorithm will return?*

d) *What is the search phase complexity?*

Exercise 3. *Consider the following algorithm calculating the length of the longest borders of all prefixes of a string x of length m, followed by a character different from the one following the prefix, and -1 otherwise.*

Algorithm 3 Compute KMP_next(string x; integer m)

```
1: k = 0
2: j = KMP_next[0] = −1
3: for i = 0 to m − 1 do
4:    if x[i] == x[k] then
5:       KMP_next[i] = KMP_next[k]
6:    else
7:       KMP_next[i] = k
8:       do k = KMP_next[k]
9:       while k ≥ 0 and x[i] ≠= x[k]
10:   end if
11:   k = k + 1
12: end for
13: KMP_next[m] = k
14: return KMP_next
```

Fill up the values in the following KMP_next table, for each string.

strings	0	1	2	3	4	5	6	7	8	9	10
ababa											
abcacabb											
abcacababc											
abacabacab											

What is the complexity of the algorithm?