Assignment 5 Problem 2

A deranged instructor once posed the following question on a final exam:
Note: this is not the problem you are asked to solve.

Given a text T of length n and an array P of size ℓ of patterns where each $P[i]$, for $0 \leq i < \ell$, is a pattern, i.e. an array of characters, where:

- the length of $P[0]$ is m
- the length of $P[k]$ is $(k+1)m$ for $k \geq 0$
- $P[k-1]$ is a prefix of $P[k]$ for all $k > 0$
- n is larger than the length of the longest pattern $P[\ell - 1]$

State how to modify the Rabin-Karp algorithm so that it returns (i, k) where k is maximal such that $P[k]$ is found in T; i.e. $P[k+1]$ is not found in T. The returned i is the position in the text where pattern $P[k]$ is found. For simplicity, you may assume that $\Sigma = \{0, 1, 2, \ldots, 9\}$ and the hash function is the standard one described on Slide 16 of Module 9.

One possible solution is to do the following:

In the pre-processing step, create an array V of hash values for each pattern (of increasing length) where $V[i]$ is the hash value for $P[i]$.

The algorithm then uses a binary search approach on the pattern hash value array to find the longest pattern found in the text. For the initial step, let $b =$ the midpoint of $(0, \ell)$, then the Rabin-Karp algorithm is applied using hash value $V[b]$ and its corresponding pattern. If the pattern is found in the text, then a new pattern of V is chosen at the midpoint of $(b+1, \ell)$; otherwise the pattern of V at the midpoint of $(0, b-1)$ is chosen. The Rabin-Karp algorithm is then applied again with this new pattern. The algorithm continues in this way until the maximal length pattern found in T is determined. You may assume the appropriate values are returned (these details are not important for this question).

The questions for you to answer are below.

a) Give an algorithm to efficiently create array V and briefly justify the runtime.

For parts b) and c) below, do not include the pre-processing time from part a).

b) Analyze the algorithm given and give the following runtimes:

- Best-case runtime
- Worst-case runtime

- Best-case expected runtime
- Worst-case expected runtime

Explain how the runtimes were determined.
Note: The runtime function parameters are: n, m and ℓ.

c) Suppose the solution is modified so that if a pattern corresponding to $V[j]$ is found, Rabin-Karp is then executed using $V[j+1]$ and its corresponding pattern. If this pattern is not found, we can conclude that the pattern for $V[j]$ is maximal and terminate. Otherwise, the algorithm continues as originally stated.
Does this change either of the runtimes from part b)? Briefly justify your answer and give the new runtime(s) if they differ from the previous part.

solution:

a) Assume the value represented in P[i] is n_i. Then for V[0], we get it by $n_0 \mod p$ where p is a suitable prime number. Then we can get V[i+1] from V[i]. For V[i+1], we let temp = V[i], then we do $temp = temp \times 10 \mod p$ m times. Then, do
temp = (temp + the number that represented by the last m digits of V[i+1]) $\mod p$
For the running time, each time we get V[i+1] from V[i], we need to do O(m) operations (including "adding 0" to the end of temp, and translate the last m digits of V[i+1] into value). Also, for the V[0], we need O(m) operations to translate the characters into the value it represented. Therefore, the total running time should be O(ml)

b) Since we will stop the algorithm when we reach the maximal length pattern found, therefore, no matter the cases, we need to do $O(\log l)$ steps of "binary search"
For Best-case runtime, we assume that no hash collision occur, and we find the pattern at the first position of the text. In this case, since no hash collision occur, we can skip the character-by-character comparision. Therefore, the algorithm's best-case running time for each pattern is $O(1)$, therefore, the best-case running time for this alfothim is $O(\log l)$.
For Worst-case runtime, we assume that hash collision occurs each time, and we need to compare the entire text T each time. To go through the entire text, we need O(n) shifts. For each shifts, we need compare character-by-character due to the collision, which costs $O(ml)$ times. Therefore, totally, the worst-case running time for this algorithm is $O(nml \log l)$
For Best-case expected runtime, we assume that in the most cases, hashes do not collide, and we find the pattern at the first position of the text. In this case, since hash collision occurs hardly ever (O(1)), and each time we reach the pattern in O(1). Therefore, the best-case expected runtime should be $O(ml + \log l)$
For Worst-case expected runtime, we assume that in the most cases, hashes do not collide, and we need to go through the entire text T each time. In this case, Since we need to go through the entire text, O(n) number of shifts is required. And for these

O(n) shifts, there are only O(1) hash collision occurs, each of collision needs $O(ml)$ times to compare character-by-character. Therefore, the worst-case expected running time should be $O((n + ml) \log l)$

c) Since we are just doing another additional step in the "binary search" process, but overall, the running time of binary search in the worst-case are still $O(\log l)$, but in the best-case, the running time to find the maximal length pattern is just $O(1)$, therefore, the best-case runtime is now $O(1)$, and the best-case expected runtime is now $O(1)$ since in almost cases, the hash collision won't occur in O(1) number of comparision.