Name-Moin Khan

Student Id – 22101287

CS202-1

Question 6

Yes, we can still solve Q1 in the required time complexity using hashing, even without restrictions on patterns to a certain extent. We will have to increase the hashing from double to triple or more depending on the size of the array so we can increase the possible hash combination to increase the range of hash. We will still follow same algorithm and same use of array and binary search. But to modify our code accordingly we will need to know rough idea of the size of the string, it can be much larger than what we have now, but during programming we will still need to be aware of it. Because no possible way of coding to handle infinitely large sizes of strings.

Question 7

The minimum number of insertion orders is 1, occurring when no collisions happen, and each element is placed directly in its hashed position. On the other hand, maximum number of Insertion order possible based on question 5 stating each element can be placed in up to three positions (its hashed position or the next two) would be is $3^{k} mod\ 10^{9} + 9$, here k is the number of elements in the hash table.

Question 8

In the Q1 of this assignment I first find 2 hash Values to avoid collisions of the patterns given and store them in an array. I use a hash function with a base of 26 and one with a base of 37. I precalculated all the required mod-based power to avoid repeated calculation which would waste resources. Once I had the hash of patterns, I took one pattern at a time, took the window of that patterns size, calculates all possible hashes using rolling hash which only take O (1) time complexity making it very efficient. Once I had all the hashes of the string size that I am currently processing, I sorted them, and I used binary search first to find the first occurrence of the hash with same hash1 and hash2 value, then I did the same to find last such occurrence and subtracted both from one another to get that strings count, I then repeated this for all strings. By this approach I was very efficiently able to find pattern occurrence in a very large string.

In the Q2, I used the same hash functions, in this time I found all the possible variation hashes for the first string I get, I search in the list of these hashes for all patterns, if one is found I mark that as visited or you can say processed so that I don't have to look it up unnecessarily again. I would also keep a rcount and lcount and at the end of the one string hash variation list processing, I would choose rcount

if lower and add it into reversecount. I would also increase set count once a list was processed as long as one of the patterns matched in that hash variation list.

This is a very efficient way to process this question. My time complexity is around what was requested in both questions.

In Q4, I used the algorithm suggested by the TA on Moodle to find the lexicographically smallest order to get the hash table. I kept track of all the dependencies during checking if it followed the hash table linear probing rules given in question. I also used a separate function to check it to have a redundancy in detecting any Impossible hash tables for given rules. Then I processed the table based on the dependency, smallest size as well as having being processed or not.

Each time a number is processed based on this criterion its removed as dependency from other numbers and added to the list that's going to be our answer, we keep doing this until all are processed. By following TA's suggested algorithm, I was successfully able to keep O(N^2) time complexity as requested.

In Q5, I wasn't able to fully solve the problem as I am failing in one of the tests given and my time complexity is very high in factorial form, While I am able to successfully run the code for 4 of the inputs, 5^{th} one is taking too long to be called successful and had to be cancelled, but 4 able to run in around a few second of time in 3 of them but one of them gives wrong output and takes around a minute. This is due to me using permutations, which take very long to calculates. While my code works to a certain extent, its not completely up to the standard due to using somewhat brute force method.