Title: Hashing
Author : Mustafa Mert Gülhan
ID: 22201895
Section : 1
Homework : 3
Description : Answers to Q6,Q7,Q8

**Question 6:**

When there is no restriction on pattern lengths the rolling hash becomes inefficient. Assume for each pattern length L, rolling the original text takes O(n) time. If pattern lengths can differ widely between 1 and N which means there are closely n different pattern lengths, it will increase the rolling hash loop from O(5*n) to O(n^2) which will increase the given time complexity. I believe when there's no restriction on pattern lengths, the solving strategy might need an algorithm that can check and match multiple lengths of patterns in single rolling.

**Question 7:**

Maximum number of insertion orders is N! and to obtain this result if the hash table has every value in its original position, which means there's no linear probing. For minimum it can be 1, to obtain this result we can put every 3 elements dependent on each other continuously which causes the result to be divided continuously to 6 can turn

the result into 1 because of dependencies we need to put every element in a specific order to obtain a valid hash table.

**Question 8:**

Analysis of Question 1:

For hashing strategy, the solution uses double hashing with two different base numbers and two different prime modulos. This significantly reduces collision probability compared to single hashing. Because of this reduction this solution avoids char by char comparison for each pattern which helps reduce the complexity. Rolling hash allows O(1) window updates for the main text. Also, binary search on sorted pattern hashes reduces the search from O(m) to O(log m). For worst case scenario analysis, if m is very large, sorting the hashes for every pattern might take longer time and if the maximum pattern length is close to size of the main text it will reduce the rolling hash step however it can also increase the time because of very large hash computations for each pattern.

Analysis of Question 2:

Similar to question 1, this solution also uses double hashing which significantly reduces the collisions. Also it uses a similar concept of rolling hash but this time it is used for calculating the hash value of cyclic shifts for

each pattern. Normally cyclic shifting a string takes O(len) but rolling hash reduces it to O(1) instead. Also normally comparing two strings takes O(len) because we check every char matches. However, hashing strategy helps it to reduce to O(1). Main loop runs for every patterns which is O(n), inside the main loop it checks every cyclic shift of the current pattern and it's reverse which is O(2*len) and searching algorithm utilizes binary search which is O(log n). Thus total complexity is since we run an inner loop for length of the current pattern which happens for every pattern it takes O(totalcharacters) and from binary search O(log n) comes so total complexity becomes O(totalcharacters*log n).

Analysis of Question 4:

This solution utilizes finding dependency between each element in the hash table and counting the amount of elements that should be inserted before inserting the current element. By continuously picking the smallest element with no dependencies. It constructs the smallest order. This solution utilizes hashing (taking modulo with size) for finding the original position of each element and uses linear probing strategy for calculating the difference between its original position. The main loop runs for the filled element size of the table which can maximum O(n) and it compares with every other element which also takes O(n). Because of

these two nested loops, total time complexity becomes O(n^2).

Analysis of Question 5:

The solution strategy in this question is first it calculates the total possible insertion order count for taking the factorial of valid element count. Then it calculates how far is it from its original position by hashing (taking modulo with table size) similar to question 4. However because of the constraints, the difference from the original position can be 0,1,2. Then it loops through the hash table 3 by 3 trying to catch some sequences that can limit the insertion sequence which will reduce the factorial result because of dependencies caused by probing. Also this solution utilizes calculating modular inverses for dividing the factorial result and continuously taking modulus of result to eliminate the risk of overflowing. Thus for the time complexity, finding the difference from the original position for each element takes O(n) and searching patterns is also O(n) which makes total complexity O(n) + O(n) = O(2n) = O(n).