Recap of this week’s lectures:

- Suffix Arrays
- Parallel Algorithms (perhaps)

Define \( LCP(i, j), i < j \) as the length of the longest common prefix between \( SA[i] \) and \( SA[j] \). Consider an \( LCP \) array defined as \( LCP[i] = LCP(i, i + 1) \). From lecture, we can construct such an array, given the suffix array, in linear time.

**Theorem.** For all \( i < j \), \( LCP(i, j) = \min_{k=i}^{j-1} LCP(k, k + 1) \).

Thus with the \( LCP \) array, we can efficiently calculate any \( LCP(i, j) \) in \( O(\log n) \) or \( O(1) \) using segment trees and sparse tables respectively, which unfortunately are not in this semester’s material.

**Corollary.** For all valid \( i \) and \( j \), \( LCP(i, j) \geq LCP(i, j + 1) \) and \( LCP(i, j) \geq LCP(i - 1, j) \).
We are given two strings $S$ and $T$. Let $N$ be their combined length. Give an $O(N \log N)$ algorithm to compute the longest common substring of the two strings. (In lecture we showed how to do this with a suffix tree. This time do it using a suffix array and the LCP array.)
We are given a string $S$ of length $N$. Give an $O(N \log N)$ time algorithm that computes the number of distinct substrings of $S$. For instance, the string “aa” has 2 unique substrings.

Solution: The answer is

$$n - 1 \sum_{i=0}^{\lceil n/2 \rceil} \left( n - 1 - \text{SA}[i] - \text{LCP}[i-1] \right)$$

assuming $\text{LCP}[i-1] = 0$ when $i = 0$.

For each $i$, this calculates the number of unique substrings that are a prefix of $\text{SA}[i]$ that have not already been counted previously in the sum (i.e. is not a prefix of any suffix $\text{SA}[i']$ where $i' < i$). Again this relies on the facts from the first page.