LCPOfSuffixes(S, i, j, equal)

```
lcp ← max(0, equal)
while i + lcp < |S| and j + lcp < |S|:
    if S[i + lcp] == S[j + lcp]:
        lcp ← lcp + 1
    else:
        break
return lcp
```

InvertSuffixArray(order)

```
pos ← array of size |order|
for i from 0 to |pos| − 1:
    pos[order[i]] ← i
return pos
```

ComputeLCPArray(S, order)

```
lcpArray ← array of size |S| − 1
lcp ← 0
posInOrder ← InvertSuffixArray(order)
suffix ← order[0]
for i from 0 to |S| − 1:
    orderIndex ← posInOrder[suffix]
    if orderIndex == |S| − 1:
        lcp ← 0
        suffix ← (suffix + 1) mod |S|
        continue
    nextSuffix ← order[orderIndex + 1]
    lcp ← LCPOfSuffixes(S, suffix, nextSuffix, lcp − 1)
    lcpArray[orderIndex] ← lcp
    suffix ← (suffix + 1) mod |S|
return lcpArray
```

Analysis

Lemma

This algorithm computes LCP array in $O(|S|)$

Proof

- Each comparison increases lcp
- $\text{lcp} \leq |S|$
- Each iteration lcp decreases by at most 1
- Number of comparisons is $O(|S|)$ □