In class, I described the rough idea on how to compute LCP array in linear time. I did not give the exact algorithm. To help you understand, I now describe in greater details on how the algorithm works.

Consider the input is the string $T$, and assume the suffix array $SA_T$ (simply denoted as $SA$) is given. Recall our goal is to construct an array $LCP$, where $LCP[i] =$ length of the longest common prefix of suffix $SA[i]$ and suffix $SA[i - 1]$. For example, for $T=$tartar$, $LCP = 0 0 2 0 1 0 3$, and $SA = 7 5 2 6 3 4 1$. Before proceeding, verify the above example to make sure you understand the definitions.

We initialize a variable $h = 0$, where $h$ is the LCP length computed for the previous suffix. The algorithm process suffix $suff$, starting from 1 to $n$ (where $n = |T|$). Note that $suff$ is not the index of $SA$, but rather the real suffix position in $T$. In our example, it starts from $suff = 1$. We need to assume that we can easily find where $suff$ appears in $SA$. This is easy to do by some linear-time preprocessing. We find $suff = 1$ has a predecessor 4 (that is, $\text{Pred}(1) = 4$). Since $h=0$, we simply compare suffix 4 with suffix 1. It is easy to see these two suffixes match 3 positions, and we know $LCP[7] = 3$, and also set $h=3$. Now we move on to the next suffix, which is $suff = suff + 1 = 2$. Again check where suffix 2 is. We find $\text{Pred}(2) = 5$. Now since $h = 3$, by what I described in class, we can skip the first two positions of suffix 2 and suffix 5, and start with the third position in our comparison. In this case, it is a mismatch at position 3. And we know $LCP[3] = 2$, and we set $h=2$. Now we move on to $suff = suff + 1 = 3$, and so on. I hope the algorithm becomes clear to you. If so, continue to read Gusfield’s notes on how exactly the running time of this LCP method is analyzed to make sure you understand what we discussed in class.