Q 3.1)

One of the possible $n = 3, k = 3, g = 4$ tomes is

```
A B C,
B A B,
A B D
```

This could not be in alphabetical order because when comparing the first 2 words, if we would like it to be in alphabetical order, 'A' should be before 'B', however, when comparing the last 2 words, 'B' should be before 'A'. If we reindex glyph 'A' to be before 'B' then the last 2 words wouldn't be in order, however, if we reindex glyph 'B' to be before 'A' then the first 2 words wouldn't be in order.

Thus, it is not possible to make both first and last 2 words be in order at the same time no matter how we reindex the glyphs.

Q 3.2)

Begin by initializing an adjacency list $graph$ having each glyph (1 to g) as vertices with no edges yet in $O(g)$ time.

Iterating through n names in S and comparing 2 adjacent words at a time ($S[i]$ with $S[i + 1]$). Then, for each of those comparing, iterating through k glyphs in those words and comparing their glyphs 1 by 1 ($S[i][j]$ with $S[i + 1][j]$). If both glyphs are the same then continue iterating to the next glyph ($j + 1$). But when both are different then assuming glyph $S[i][j]$ is before glyph $S[i + 1][j]$, adding an edge in $graph$ from glyph $S[i][j]$ (source) to glyph $S[i + 1][j]$ (destiny) then stop iterating to the next glyph and move on to comparing next 2 adjacent words ($S[i + 1]$ with $S[i + 2]$). Continue until iterating through all n names.

After iterating through n names, perform topological sorting of $graph$ in $O(n + g)$ time. However, if there are some edges remaining (according to lecture's topological sorting algorithm) meaning that $graph$ is a cyclic graph (two ways edge or cycle causing a similar situation in Q3.1 occurs, 'A' is supposed to be before 'B' but 'B' is also supposed to be before 'A') causing it impossible to find an alphabet that could make S be in alphabetical order. We then determine that no such alphabet exists.

Otherwise, L (according to lecture's topological sorting algorithm) is a topological ordering represents the order of the glyphs that the algorithm need to determine.

Since we add a new edge whenever we assume source is before destiny making the result from topological sorting be the order of glyphs that makes given names be in alphabetical order if the graph is acyclic as explained above.

Overall time complexity is; $O(g)$ for initializing $graph$. Iterating a n length array, for each element in the array, iterating an k length array and adding an edge all take $O(n(k * 1))$. And $O(n + g)$ for topological sorting. Which $O(g + nk + n + g) \cong O(nk + g)$ in total.