**Explanation of Pre-processing and Improvement in Search Time**

**Naive Approach:**

In the naive string matching algorithm, the pattern is compared to the text character by character. If a mismatch occurs, the pattern is shifted by one character to the right, and the comparison starts over. This results in a worst-case time complexity of O(MN), where M is the length of the pattern and N is the length of the text.

**KMP Algorithm:**

The KMP algorithm improves the search time by pre-processing the pattern to create an auxiliary array called the LPS (Longest Prefix Suffix) array. This array is used to skip characters in the text when a mismatch occurs, thereby reducing unnecessary comparisons.

**LPS Array:**

The LPS array for a given pattern is constructed in such a way that each entry at index i in the array contains the length of the longest proper prefix of the pattern that is also a suffix for the substring ending at index i.

**How Pre-processing Improves Search Time:**

**Avoiding Redundant Comparisons:**

During the search, when a mismatch occurs after j matches, instead of shifting the pattern by one character to the right and starting comparisons from scratch, the KMP algorithm uses the information from the LPS array to skip characters in the pattern and the text.

Specifically, it moves the pattern such that the next character in the text aligns with the longest prefix that is also a suffix. This ensures that the matched portion of the pattern does not need to be re-evaluated.

**Efficiency:**

The pre-processing step (computing the LPS array) takes O(M) time.

The actual search phase runs in O(N) time because the algorithm ensures that each character of the text is compared at most once, either directly or indirectly through the use of the LPS array.

Overall, the KMP algorithm improves the search time to O(M + N), which is much more efficient than the O(MN) time complexity of the naive approach, especially for larger texts and patterns.