The KMP algorithm preprocesses the pattern to construct a Longest Prefix Suffix (LPS) array, which helps in skipping unnecessary comparisons during the search process.

```
import java.util.*;
public class KMPAlgorithm {
    public static int[] computeLPSArray(String pattern) {
        int[] lps = new int[pattern.length()];
        int len = 0; // Length of the previous longest prefix suffix
        int i = 1;
        while (i < pattern.length()) {
            if (pattern.charAt(i) == pattern.charAt(len)) {
                len++;
                lps[i] = len;
                i++;
            } else {
                if (len != 0) {
                    len = lps[len - 1];
                } else {
                    lps[i] = 0;
                    i++;
                }
            }
        }
        return lps;
    }
    public static List<Integer> search(String text, String pattern) {
        List<Integer> indices = new ArrayList<>();
        if (text == null || pattern == null || text.length() == 0 || pattern.length() == 0)
            return indices;
        int[] lps = computeLPSArray(pattern);
        int i = 0; // Index for text[]
```

```
    int j = 0; // Index for pattern[]
    while (i < text.length()) {
        if (pattern.charAt(j) == text.charAt(i)) {
            i++;
            j++;
        }
        if (j == pattern.length()) {
            indices.add(i - j);
            j = lps[j - 1];
        } else if (i < text.length() && pattern.charAt(j) != text.charAt(i)) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
    return indices;
}
public static void main(String[] args) {
    String text = "ABABDABACDABABCABAB";
    String pattern = "ABABCABAB";

    List<Integer> indices = search(text, pattern);
    if (indices.isEmpty()) {
        System.out.println("Pattern not found in the text.");
    } else {
        System.out.println("Pattern found at indices: " + indices);
    }
}
```

}

Explanation:

computeLPSArray: This method calculates the Longest Prefix Suffix (LPS) array for the given pattern. The LPS array at index i stores the length of the longest proper prefix of the pattern that is also a suffix ending at index i. This information helps in avoiding unnecessary comparisons during the search.

search: This method performs the actual pattern searching using the computed LPS array. It iterates through the text and pattern, using the LPS array to efficiently skip unnecessary comparisons. If a mismatch occurs, it adjusts the indices based on the values stored in the LPS array.

By pre-processing the pattern to construct the LPS array, the KMP algorithm avoids redundant comparisons that the naive approach would perform. This leads to a significant improvement in search time, especially for large texts and patterns, as it eliminates the need to backtrack and recheck previously matched characters. As a result, the KMP algorithm has a time complexity of O(n + m), where n is the length of the text and m is the length of the pattern, compared to the naive approach with O(n * m) time complexity.