Details about Time Complexity

**Best Case:**

In the best case scenario, the Rabin-Karp algorithm efficiently identifies a match between the pattern and a substring of the text by quickly detecting a hash match without performing unnecessary comparisons. This occurs when the hash function and the chosen rolling hash algorithm minimize collisions and efficiently distribute hash values.

**Time Complexity: O(n+m)**

In the best case, where the pattern occurs exactly once in the text, and there are no hash collisions, the time complexity approaches linear time, O(n + m), where n is the length of the text and m is the length of the pattern.

**Worst Case**:

The worst-case scenario for Rabin-Karp arises when multiple spurious hash collisions occur, leading to the need for exhaustive comparison of the pattern with potential matches in the text. This can happen when the hash function produces many collisions or when the rolling hash algorithm performs poorly, resulting in frequent rehashing.

**Time Complexity: O((n-m+1) \* m)**

In the worst case, where every substring of the text matches the pattern's hash value, the time complexity can approach quadratic time, O((n - m + 1) \* m), where n is the length of the text and m is the length of the pattern.

**Average Case:**

The average case performance of the Rabin-Karp algorithm depends on the quality of the hash function and the rolling hash algorithm. With a well-designed hash function and rolling hash, the algorithm tends to perform efficiently by quickly identifying matches with minimal comparisons.

**Time Complexity: O(n+m)**

On average, the time complexity of the Rabin-Karp algorithm remains linear, O(n + m), where n is the length of the text and m is the length of the pattern. This assumes a reasonably distributed hash function and effective rolling hash algorithm, leading to a balanced distribution of hash values and minimized collisions.

**Summary:**

**Best Case:**

O(n + m) when the hash function and rolling hash algorithm efficiently detect matches without excessive comparisons.

**Worst Case:**

O((n - m + 1) \* m) when multiple spurious hash collisions occur, necessitating exhaustive comparison of the pattern with potential matches.

**Average Case:**

O(n + m) when the hash function and rolling hash algorithm effectively distribute hash values and minimize collisions, resulting in efficient matching.