**Hash Table Analysis Report**

**Introduction**

This report contrasts and analyzes two hash table implementations based on the same set of test cases: a Linear Probing Hash Table and a Chaining Hash Table. Both tables are designed to handle string-type keys and are tested with a small capacity to induce collisions.

**Hash Functions**

The Linear Probing Hash Table uses a hash function that multiplies the length of the string key by a prime number (31) and takes the modulus with the current capacity. This approach is simple and fast but may lead to clustering, where consecutive entries become filled, leading to a higher chance of collisions.

The Chaining Hash Table employs a different strategy, combining the character values and their positions in the string to calculate the hash. This method spreads out the keys more evenly across the table, reducing the likelihood of collisions.

**Contrast Analysis:**

* The Linear Probing hash function is more prone to primary clustering.
* The Chaining hash function provides a better distribution, minimizing collisions.

**Collision Resolution Strategies**

**Linear Probing:**

* **Pros:**
  + Simple to implement.
  + Ensures that all entries in the table are used, which can be more space-efficient.
* **Cons:**
  + Clustering can occur, leading to longer search times.
  + Performance degrades significantly when the table is near capacity.

**Separate Chaining:**

* **Pros:**
  + Less sensitive to the hash function, as collisions do not lead to a significant increase in search time for other keys.
  + Performance remains more consistent as the table fills up.
* **Cons:**
  + Requires additional memory for pointers in the linked lists.
  + Overhead of managing linked lists.

**Rationale and Detailed Analysis:** Linear probing is generally faster for tables with low load factors, where the chance of collision is minimal. However, as the load factor increases, the cost of probing for an empty slot increases. Separate chaining, while requiring more memory, handles high load factors better because the linked lists can grow independently of the table size.

In our tests, the Linear Probing Hash Table required resizing more frequently as it approached its capacity, which is a costly operation. The Chaining Hash Table, on the other hand, handled collisions gracefully without the need for immediate resizing, as the linked lists absorbed the additional entries.

**Conclusion:** The Chaining Hash Table is more robust and maintains performance across a wider range of scenarios, especially when the table size is small, and collisions are frequent. The Linear Probing Hash Table can be more efficient with a good hash function and low load factors but requires careful management of the table size to avoid performance degradation.

This report provides a high-level overview of the differences and effects of the two hash table implementations and their collision resolution strategies. For a more comprehensive analysis, one would need to consider additional factors such as the expected load factor, key distribution, and memory constraints.