**Hash Table 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

* **Linear Probing Hash Table**: The hash function used here multiplies each character’s ASCII value by 31 (a prime number) and accumulates the result, modulo the table’s capacity.
* **Chaining Hash Table**: Uses a prime number (1031) as a multiplier in its hash function. This is a common practice to reduce collisions because a prime number will not introduce common factors other than 1 when multiplied with character codes, which are also mostly prime or odd numbers.

**Collision Resolution Strategies**

**Linear Probing:**

* **Pros:**
  + Simple to implement.
  + Ensures that all buckets in the table are used, which can be more space-efficient.
* **Cons:**
  + May lead to clustering, where a group of consecutive slots gets filled, leading to longer search times.

**Chaining Probing:**

* **Pros:**
  + Performance remains more consistent as the table fills up because it simply adds the colliding entry to the linked list.
* **Cons:**
  + Requires additional memory for pointers in the linked lists.

**Rationale and Detailed Analysis:**

ChainingHashTable: Implements separate chaining, where each hash index points to a doubly linked list that can store multiple buckets. LinearProbingHashTable: Uses linear probing, where it places the colliding buckets in the next available slot in the array.

### Effects of Collision Resolution Strategies

Memory Usage: Separate chaining typically requires more memory due to the overhead of linked list pointers. Linear probing can be more memory-efficient but may suffer from clustering.

Performance: Separate chaining can handle a higher load factor without a significant performance penalty, as the linked lists can grow as needed. Linear probing, however, can see a performance drop as the load factor approaches the threshold, due to increased collisions and clustering.

Load Factor: ChainingHashTable specify a load factor 75% for resizing, while LinearProbingHashTable has a set threshold of 0.7. This aims to keep the number of collisions low.

In summary, separate chaining is generally more robust to high load factors and can maintain performance better as the number of entries grows. Linear probing is more space-efficient but can suffer from performance issues due to clustering, especially as the load factor increases. The choice between the two methods depends on the specific requirements of the application, such as memory constraints and expected load factor.