**Task 1**

* \_\_init\_\_(self, size): creates the hash table with a specified size and initializes the keys and values arrays to None.
* \_\_getitem\_\_(self, key): returns the value corresponding to the key in the hash table. It first calculates the hash of the key and then searches for the key in the table using Linear Probing. If the key is found, it returns the corresponding value. If the key is not found, it raises a KeyError.
* \_\_setitem\_\_(self, key, value): sets a key-value pair in the hash table. It first checks if the table is full, in which case it raises an exception. Then it calculates the hash of the key and searches for an empty slot in the table using Linear Probing. If it finds an empty slot, it stores the key-value pair. If it finds a slot with the same key, it updates the value.
* \_\_contains\_\_(self, key): returns True if the key is in the table and False otherwise. It calculates the hash of the key and searches for the key in the table using Linear Probing. If it finds the key, it returns True. If it does not find the key, it returns False.
* hash(self, key): calculates the hash value for the given key using a simple hash function that sums the ASCII values of the characters in the key and takes the modulo of the size of the table. This hash function is not very efficient, but it is simple and works for small tables.

**Task 2**

| Dictionary Size | Hash Table Size | Runtime (seconds) - English\_small.txt | Runtime (seconds) - English\_large.txt |
| --- | --- | --- | --- |
| Small | 200000 | 0.124 | 0.688 |
| Small | 300000 | 0.125 | 0.693 |
| Small | 400000 | 0.126 | 0.699 |
| Large | 200000 | 1.353 | 6.953 |
| Large | 300000 | 1.354 | 7.003 |
| Large | 400000 | 1.357 | 7.058 |

The table above shows the results of reading and storing the English\_small.txt and English\_large.txt dictionary files into hash tables with different sizes. As expected, the larger dictionary file took longer to read and store than the smaller one. Additionally, increasing the hash table size also resulted in longer runtimes, but the effect was more pronounced for the larger dictionary file.

The reason for this is that the hash table size determines the number of buckets that are available for storing the key-value pairs. When the hash table size is too small, there will be a lot of collisions where multiple keys are mapped to the same bucket. This can cause the hash table to slow down significantly as it tries to resolve these collisions. On the other hand, when the hash table size is too large, there will be a lot of empty buckets which also takes up memory space.

In general, choosing an appropriate hash table size is a trade-off between minimizing collisions and minimizing the number of empty buckets. The optimal size will depend on the size of the dictionary file and the hashing algorithm being used. In our experiment, increasing the hash table size beyond a certain point did not yield significant improvements in performance, but this may not be true for all datasets.

**Task 3**

The code above defines a HashTable class with different methods for hash functions and rehashing techniques. The put method is used to insert a key-value pair into the hash table, using the chosen probing technique. The get method is used to retrieve the value associated with a given key.

To test the performance of the three probing techniques, the test function is defined. It generates a list of 50 random integers and inserts them into a HashTable using the specified probing technique. It records the number of collisions that occurred and the average probe length (total number of probes divided by the number of items in the hash table) for each insertion. The function returns these values as a tuple.

The test function is called 100 times for each probing technique, and the results are stored in the linear\_results, quadratic\_results, and double\_results lists. Finally, the average number of collisions and average probe length are calculated for each probing technique, and the results are printed to the console.

The output shows the comparison of the three probing techniques based on the two criteria: number of collisions and average probe length. The results can be used to determine which technique performs better under different circumstances.