1. Explain the hashing function you used for BadHashFunctor.
Be sure to discuss why you expected it to perform badly (i.e.,
result in many collisions).

The hash function uses the ASCII code of the last character in the string. Strings with the same last character, they will collide

2. Explain the hashing function you used for MediocreHashFunctor. Be sure to discuss why you expected it to perform moderately (i.e., result in some collisions).

hash = (sum of all ASCII codes) × ASCII(first character) × ASCII(last character)
The function reduces collisions, such as those caused by identical character sums (e.g., “az” and “by”), by incorporating the first and last characters.However, switched characters (e.g., “ac” and “ca”) still result in collisions because the ASCII sum and the multiplication terms remain the same:

3. Explain the hashing function you used for GoodHashFunctor. Be sure to discuss why you expected it to perform well (i.e., result in few or no collisions).

For GoodHashFunctor, I implemented the djb2 algorithm. The algorithm uses a special constant (5381) derived experimentally and a “magic” calculation, combining bit-shifting and addition, to create a highly varied and well-distributed set of hash values.

5. What is the cost of each of your three hash functions (in BigO notation)? Note that the problem size (N) for your hash functions is the length of the String, and has nothing to do with the hash table itself. Did each of your hash functions perform as you expected (i.e., do they result in the expected number of collisions)?

For BadHash, it’s O(1) For Mediocre Hash, it’s O(N) For Good hash, it’s O(N) Expectedly, good hash has the better performance along with the increase of the data size.