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).

(1) The BadHashFunctor returns the ASCII value of the first character in the String. In this case, the other characters' values are not used. Besides, the ASCII value of one character just varies in a small range, which will result in many collisions.

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).

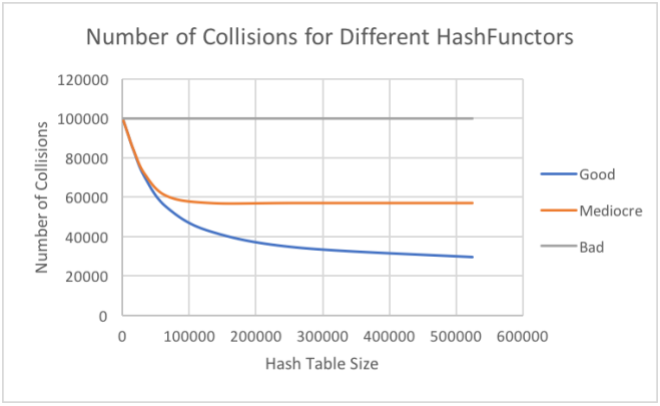
(2) The MediocreHashFunctor returns the sum of the ASCII values of each character's square in the string. In this case, the sum value varies in a bigger range compared with the bad one, which will still result in some collisions.

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).

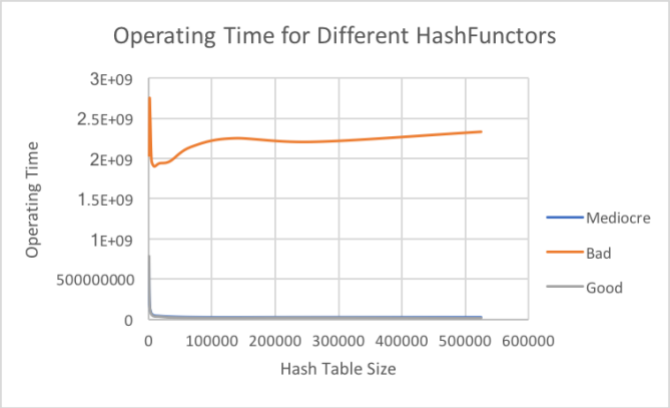
(3) The GoodHashFunctor first assign a prime number to the hash value, then loop through each character in the string, each time the product of the hash value and a prime number is added to the ASCII value of the character, then assign the sum to the hash value. In this case, it will result in few or no collisions.

4. Design and conduct an experiment to assess the quality and efficiency of each of your three hash functions. Briefly explain the design of your experiment. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plot(s), as well as, your interpretation of the plots is important. A recommendation for this experiment is to create two plots: one that shows the number of collisions incurred by each hash function for a variety of hash table sizes, and one that shows the actual running time required by various operations using each hash function for a variety of hash table sizes.

(4) I created an array of random strings used for testing. For different sizes of hash table, I calculated the total collisions for each hash functor by same string array. The result is shown below.



Then I tested the operation time like add method and remove method for each hash functor, the result is shown below.



5. What is the cost of each of your three hash functions (in Big-O 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)?

(5)
Bad: O(1)
Mediocre: O(N)
Good: O(N)

Yes they result in the expected number of collisions as the picture shown above.