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).

Bad hash returns the length of the string, it will basically only use the first dozen or so places for words, maybe more for a full book

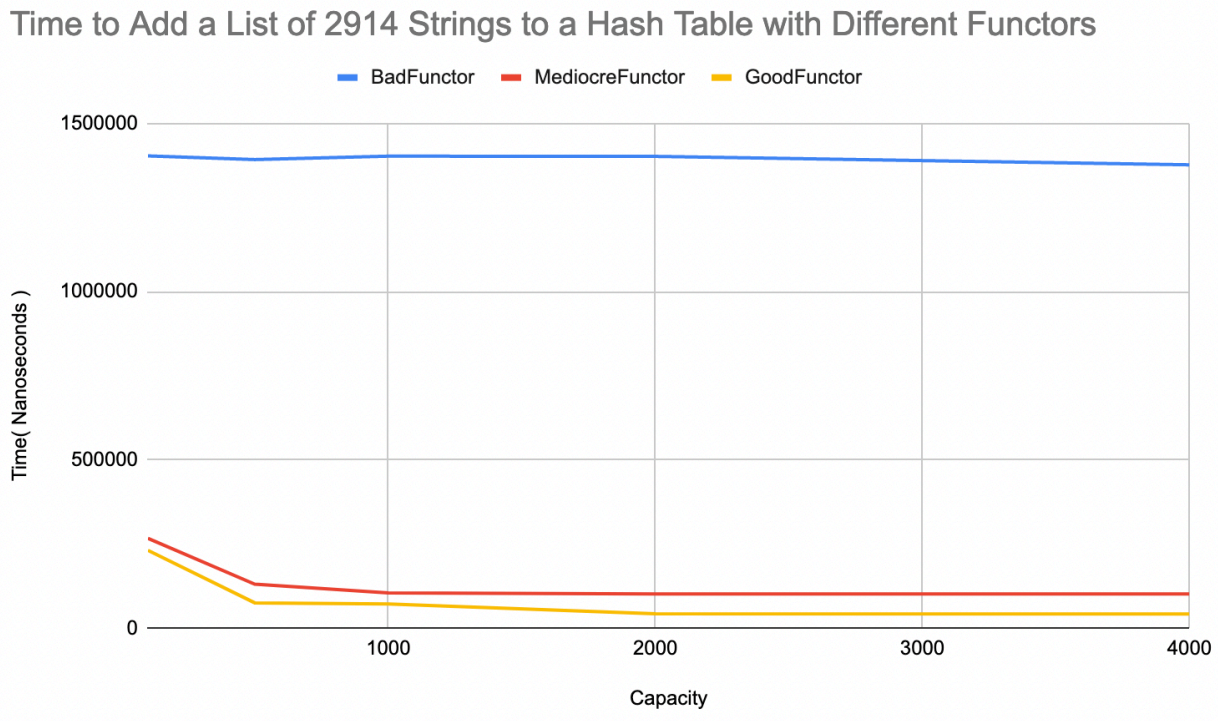
2. Explain the hashing function you used for MediocreHashFunctor. Be sure to discuss why you expected it to perform moderately (i.e., result in some collisions).

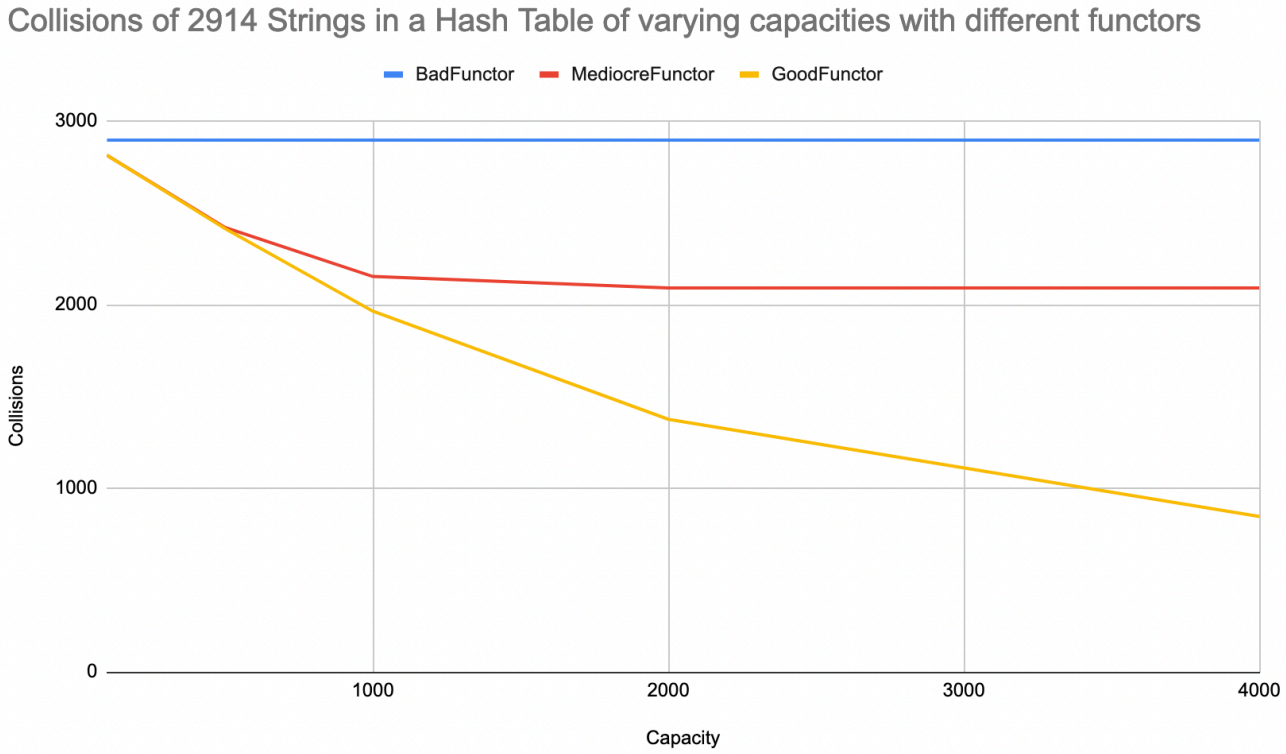
For mediocre hash functor, we take the ascii value of all the characters in the string and add them together, the value will likely not get too high, especially on short words

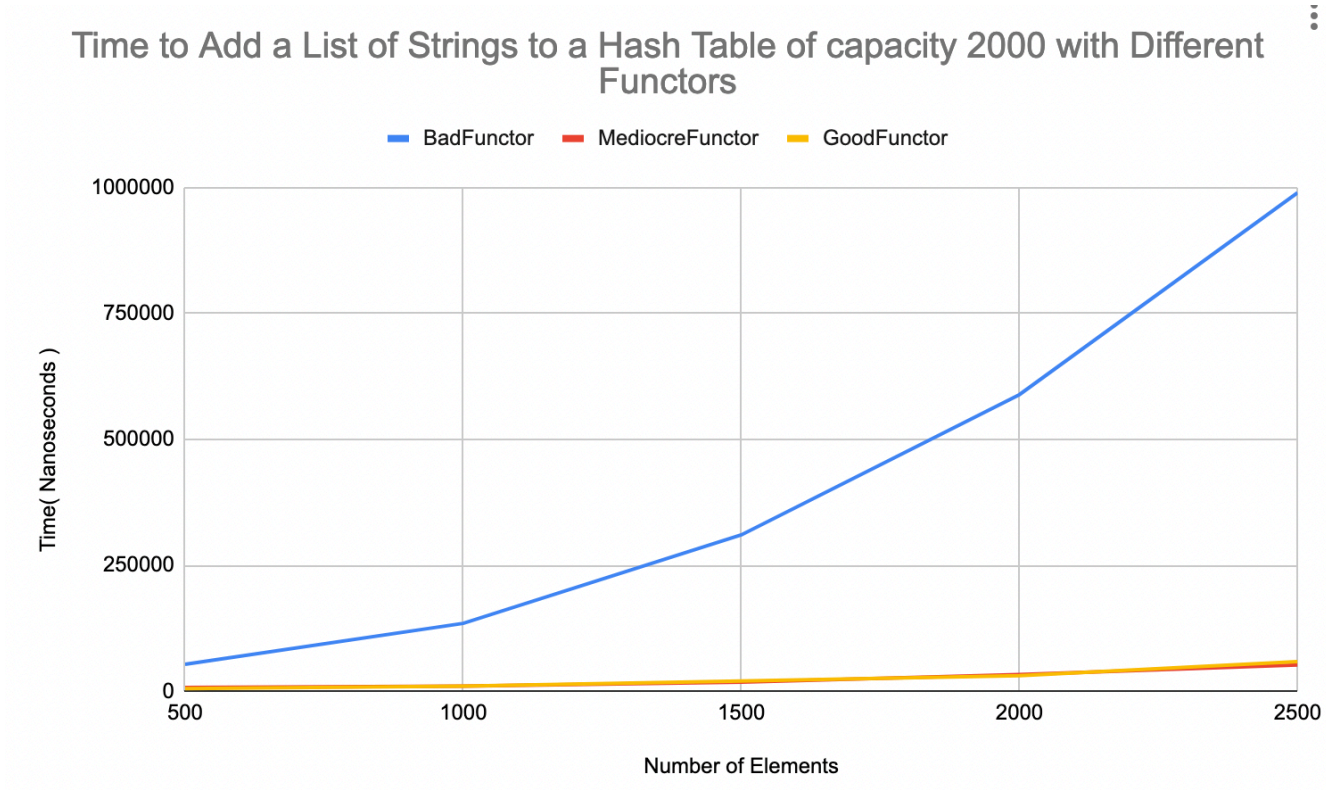
3. Explain the hashing function you used for GoodHashFunctor. Be sure to discuss why you expected it to perform well (i.e., result in few or no collisions).

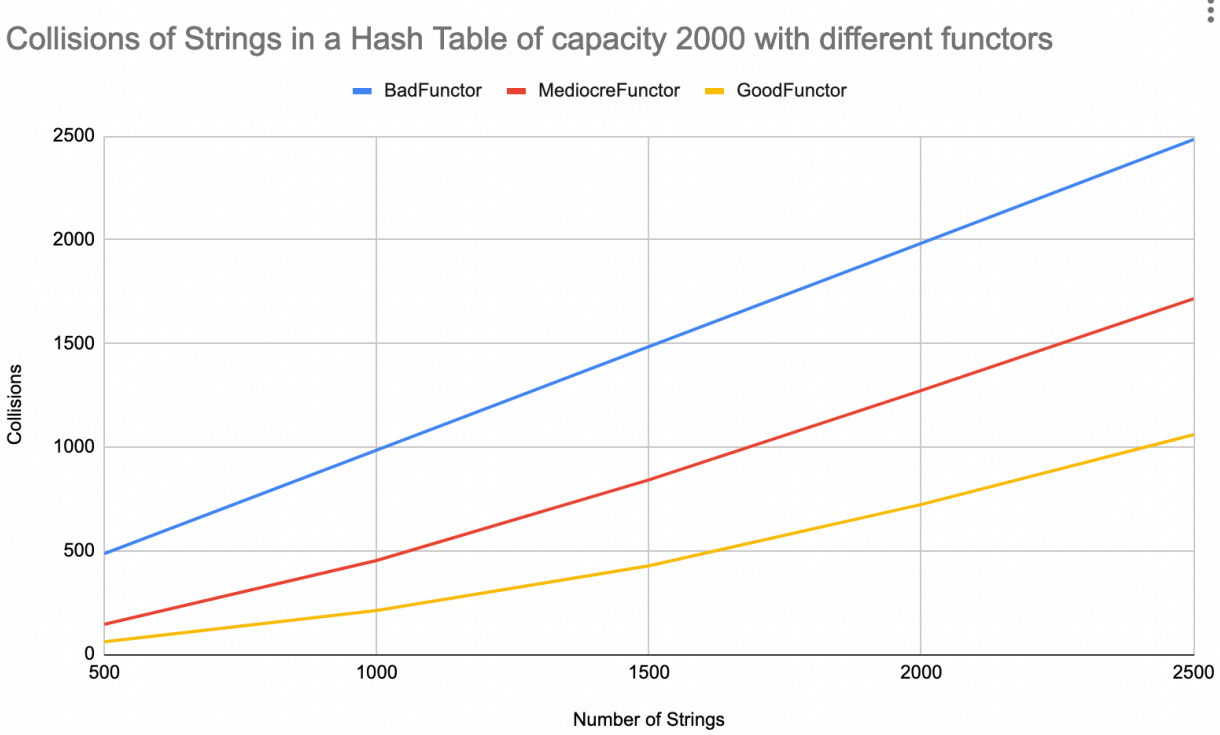
For good hash functor, we take the hash that starts at 5381, and for every character, multiply the hash by 33 and and the characters ascii value to it. This will make the value huge, and will likely even overflow, so we take the absolute value of it. The multiply by 33 makes every character change the value by a lot, and gives higher weight to earlier values, so words with different ordering of the same characters will still be very different

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.




Collisions of 2914 Strings in a Hash Table of varying capacities with different functors
BadFunctor
MediocreFunctor
GoodFunctor
3000
2000
1000
0
Collisions
1000
2000
3000
4000
Capacity


Time to Add a List of Strings to a Hash Table of capacity 2000 with Different Functors
BadFunctor
MediocreFunctor
GoodFunctor
1000000
750000
500000
250000
0
Time(Nanoseconds)
500
1000
1500
2000
2500
Number of Elements



Set up for loops for each functor and different capacities. Use 2000 loops to get an average time for adding a dictionary of words that we take different amounts of to get different number of elements to add

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)?

BadHash took what looks to be N^2 time, about what I'd expect due to the huge number of collisions, its basically going through the entire linked list checking for duplicates before adding it to the end, acting very much like expected of a bad hash

Mediocre and good hash looks like it increases much slower, but will get closer and closer to N^2 growth as the hash table fills up, about what we expect, but we could avoid this by rebuilding the table if it gets too filled up,