When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 30% of your Assignment 10 grade. Ensure that your analysis document addresses the following.

1. What does the load factor Î» mean for each of the two collision-resolving strategies (quadratic probing and separate chaining) and for what value of Î» does each strategy have good performance?

The load factor allows the Separate Chaining and Quadratic Probing to resolve collisions in different ways. For quadratic probing if the index we hashed for is occupied we will consider H + i^2. The variable I will increment by one with each detected collision and will wrap around to the beginning of the array if necessary. To avoid clustering if our loadfactor is >= 0.5 we will rehash the array rather than just copying over the items over. For Separate Chaining we utilize and array of linked lists and a hash function to select the index into an array. Our loadfactor is determined by the average length of the linkedlists. Therefore, this method is efficient for short lists. So instead of rehashing the table when it's half full we rehash when the average length of linkedLists becomes too large.

2. Give and 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).

For a bad hashFunctor, I used the the class that was given. I expected it to perform badly because it only returns one value which we can map to a valid array. This, in turn, will cause a collision in almost every instance. Givin that it only returns to the same position every time.

3. Give and 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 a MediocreHashFunctor I chose to use assign the hasValue to the length of the character multiplied by a prime number. I expect this to perform moderately because it does not provide uniform distribution of values greater than 3 which in turn increases the number of collisions and the cost of resolving them.

4. Give and 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).

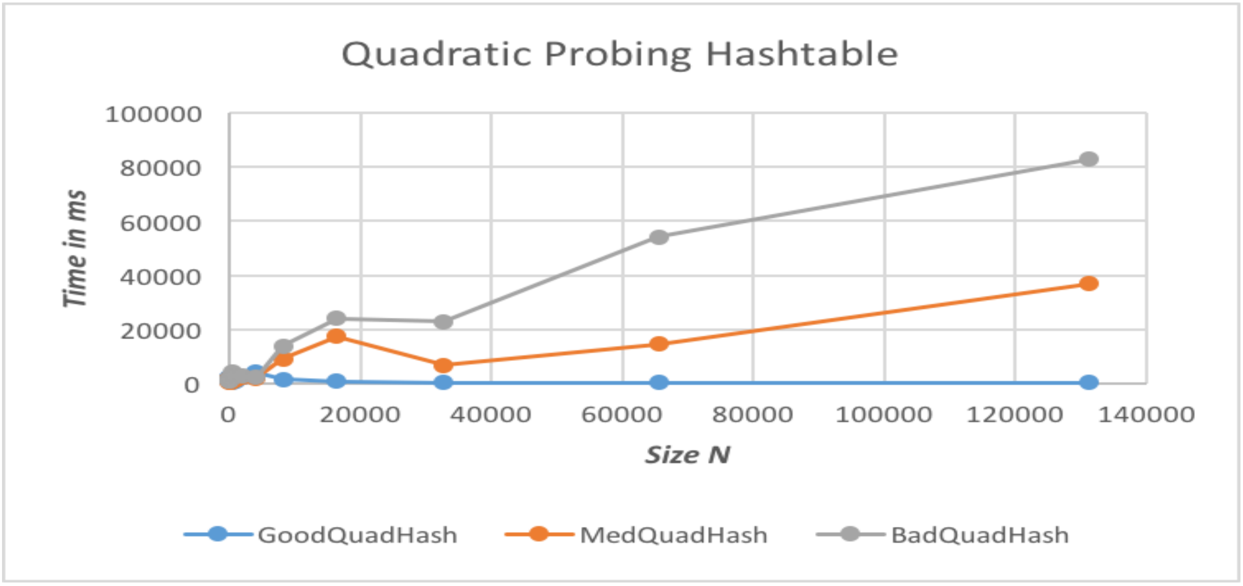
For a GoodHashFunctor I chose to use assign the hasValue to the length of the string. I expect this to perform well because it does provide uniform distribution and consistency which in turn reduces the number of collisions and the cost of resolving them. There is still some clustering but given the Separate chaining and Quadratic Probing this will still allow for quick access.

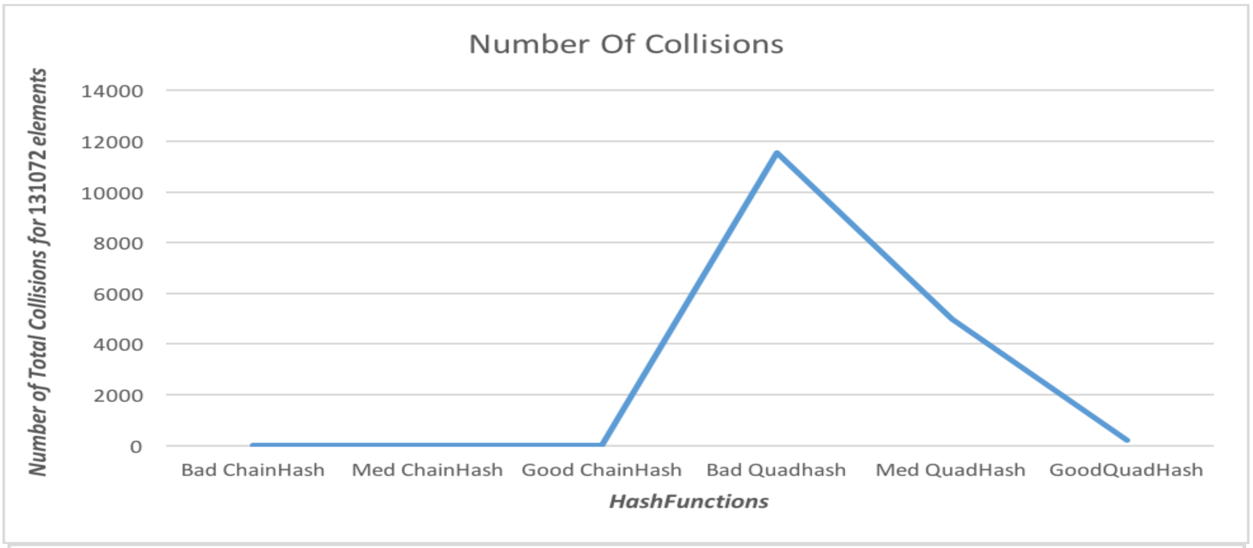
5. Design and conduct an experiment to assess the quality and efficiency of each of your three hash functions. Carefully describe your experiment, so that anyone reading this document could replicate your results. Plot the results of your experiment. Since the organization of your

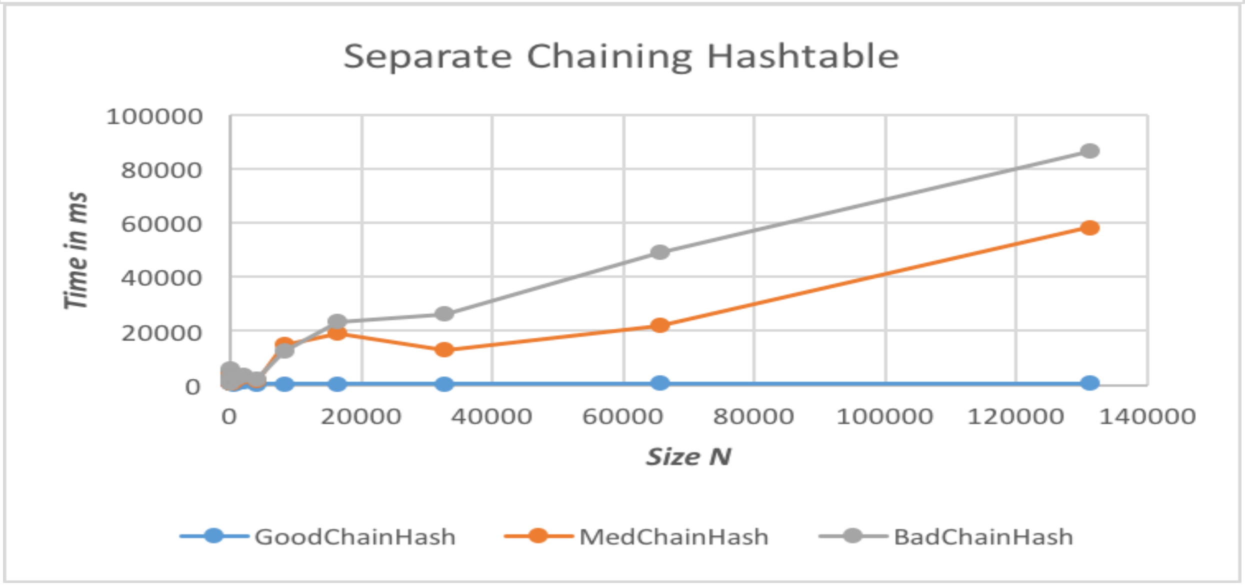
plot(s) is not specified here, the labels and titles of your plot(s), as well as, your interpretation of the plots is critical. â ¨A recommendation for this experiment is to create two plots: one that shows

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 each hash function for a variety of hash table sizes. You may use either type of table for this experiment.

So I set up a basic timing class with a random string generator. From there I implemented some variables to keep track of the plot times and number of collisions occurring while the method was running. From there I would print the result to a file.



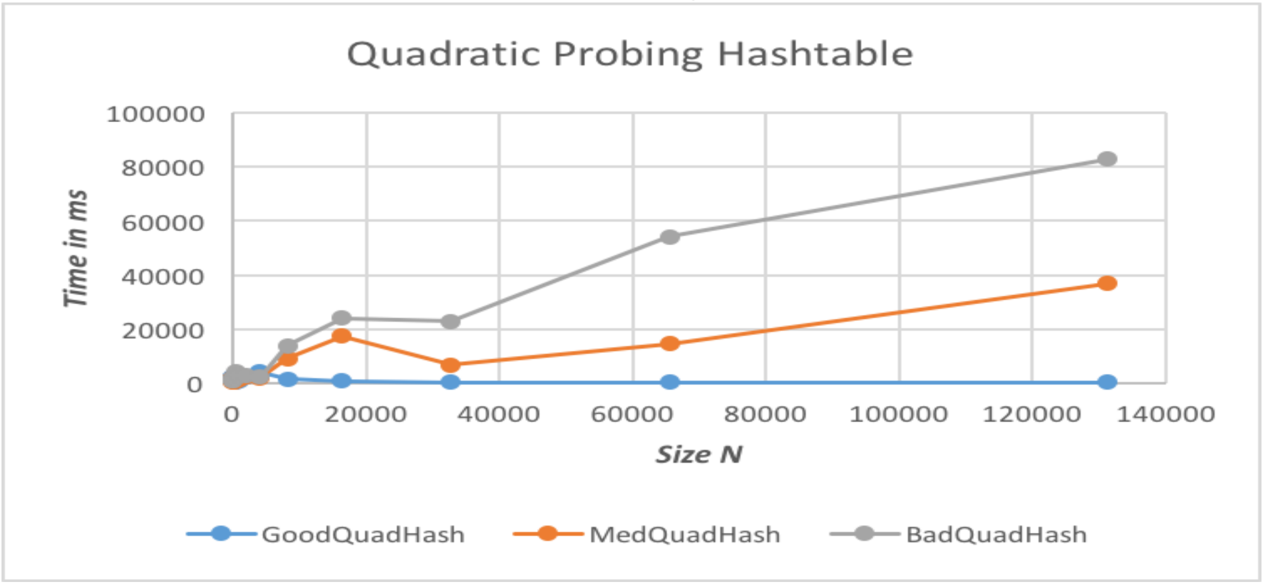


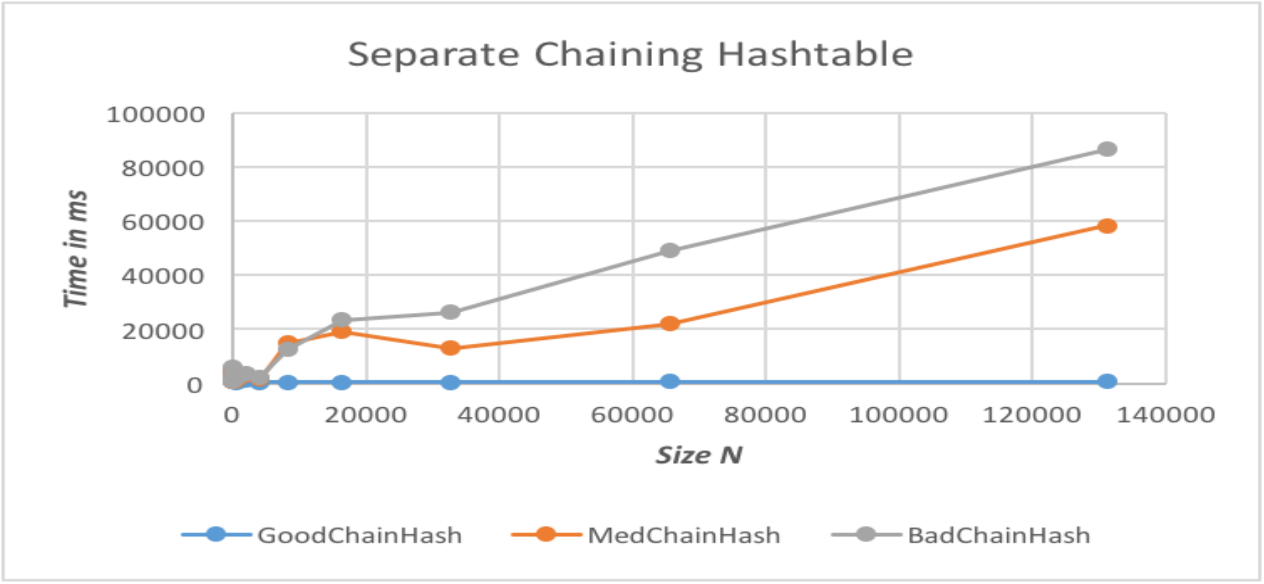


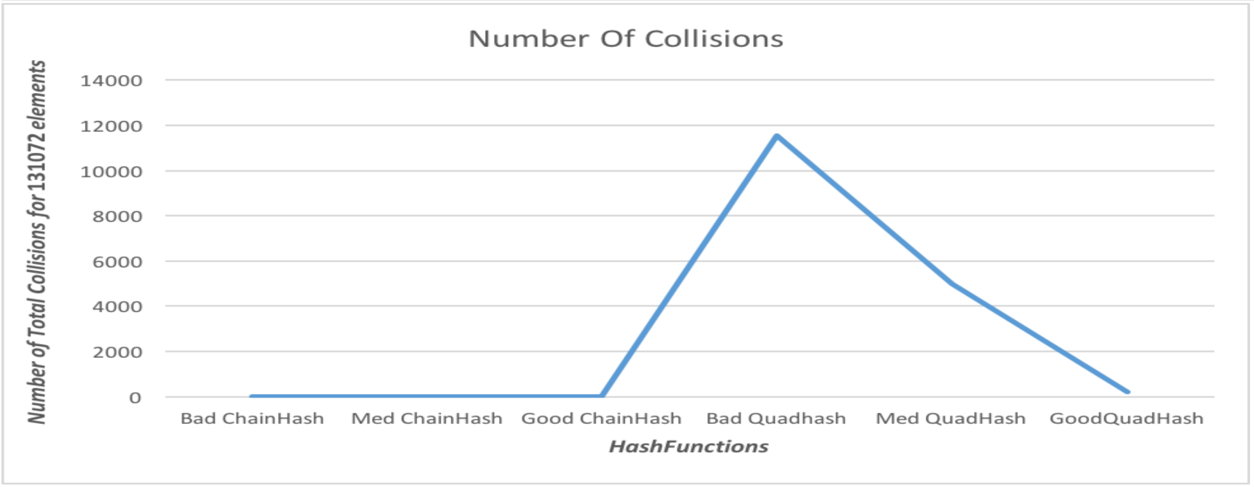
6. Design and conduct an experiment to assess the quality and efficiency of each of your two hash tables. Carefully describe your experiment, so that anyone reading this document could replicate your results. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plot(s), as well as, your interpretation of the plots is critical. â ¨A recommendation for this experiment is to create two plots: one that shows

the number of collisions incurred by each hash table using the hash function in GoodHashFunctor, and one that shows the actual running time required by each hash table using the hash function in GoodHashFunctor.

So to test the efficiency of the hash tables again I set up a basic timing class with a random string generator. From there I switched some of the variables around from 5-25 character strings and kept track of the plot times and a number of collisions occurring while the method was running.


Quadratic Probing Hashtable
Time in ms
100000
80000
60000
40000
20000
0
0
20000
40000
60000
80000
100000
120000
140000
Size N
GoodQuadHash
MedQuadHash
BadQuadHash


Separate Chaining Hashtable
Time in ms
100000
80000
60000
40000
20000
0
0
20000
40000
60000
80000
100000
120000
140000
Size N
GoodChainHash
MedChainHash
BadChainHash


Number Of Collisions
Number of Total Collisions for 131072 elements
14000
12000
10000
8000
6000
4000
2000
0
Bad ChainHash
Med ChainHash
Good ChainHash
Bad Quadhash
Med QuadHash
GoodQuadHash
HashFunctions

7. 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)? (Be sure to explain how you made these determinations.)

The cost of my three hashFuntion in Big-O notation was O(1). Though the graphs may look otherwise the plotted time were very consistent for the length of the string I was using. In separate tests, I used strings from 25 characters to 5 characters and besides the time to read in the strings the timing was very similar across the boards.

8. How does the load factor Î» affect the performance of your hash tables?

The load factor is a measure of how full the hash table is allowed to get before its capacity is increased. When the number of elements in the hash table exceeds the load factor the hash table is rehashed so that the hash table has approximately twice the number of buckets and more space to work with for additional elements.

9. Describe how you would implement a remove method for your hash tables.

Likely I would create a method to find the element I was looking for then assign the position it to null. From there I would reduce the size of items in the array as well as the load factor. To reduce the possibility of errors later.

10. As specified, your hash table must hold String items. Is it possible to make your implementation generic (i.e., to work for items of AnyType)? If so, what changes would you make?

I believe it could be possible to make a generic version of a hash table. I think the only implementation you would have to create is the hasher function to make sure it can accept a wide variety of data such as integers, strings, and objects in general.

11. How many hours did you spend on this assignment?
I spent 15 hours on this assignment.