Assignment 10 Analysis Paper

Chris Grayston u0906710

Q1 - Quadratic Probing)

The load factor for quadratic probing means the amount of items divided by the size of the array we are placing the items in. Load size is important for Quadratic Probing because once the load size starts approaching .5 there is a much higher probability for collisions, the time it takes to create a new bigger array and copy the new information over to is much less work in the long run than attempting to enter things into an array that is overcrowded which will result in unnecessary collisions which waste time.

Q1 - Separate Chaining)

The load factor for Separate Chaining doesn't matter as much, for me in this assignment I decided just to go with a bigger initial array size which basically nullified a reason to have to refactor and rehash our entire array. Because we have LinkedListed associated with each bucket in our array we don't worry nearly as much about load factor with separate chaining.

Q2 - BadHashFunctor)

My idea for BadHashFunctor added 1 on to the return HashValue every 4 letters, this way there would be a higher likelyhood to be collisions since there is a lower chance of variability with the values created. There are sure to be a lot more collisions which will make this set perform slower because the hash functio has to look at more spacves for an appropriate landing space for the data.

Q3 - MediocreHashFunctor)

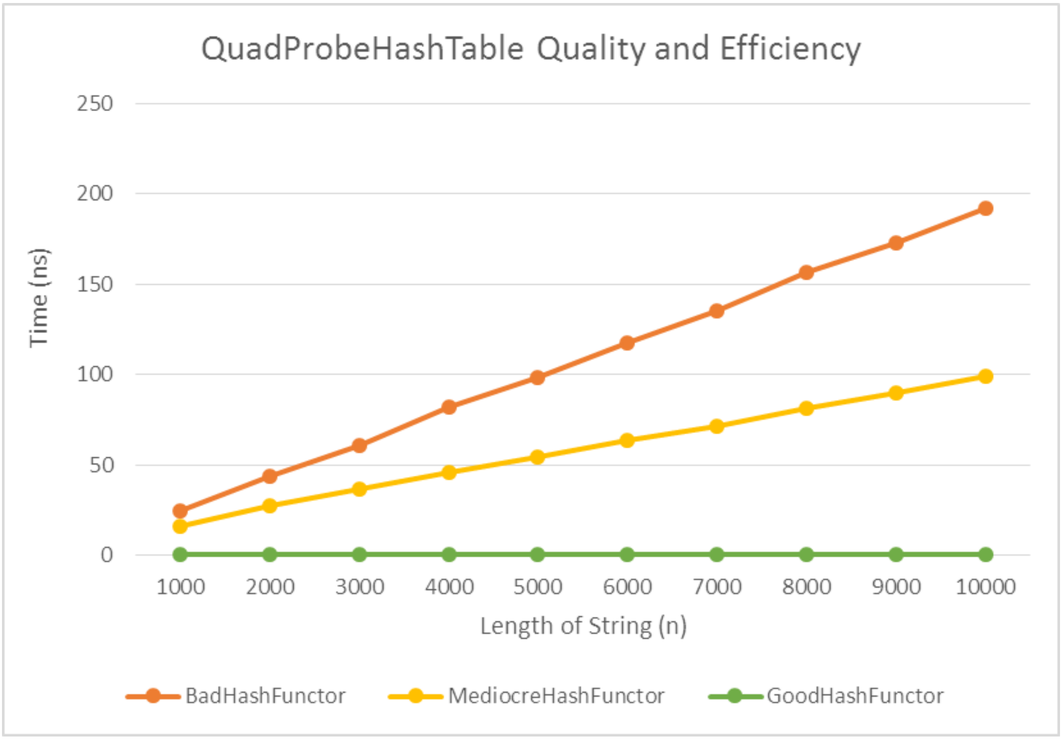
On my MediocreHashFunctor I did it so it would calculate the hash based on the values of the chars in the String. This is better because it allows for more variability when it comes to returning hash values. It is still not perfect because there is still a decently good chance they we will ahev repeat values that will cause collisions, this will result in an average speed.

Q4 - GoodHashFunctor)

My GoodHashFunctor I basically had the hash multiply by a large number and add the value of the chars in it to add two more levels of variability when returning our hash value. Using this method adds another variable to the equation making it very unlikely for any repeats on words with similar values. Even words that are spelled differently will get massively different values before they are hashed. I expect this to perform the best cause there should almost never be collisions if we choose a good size for our array.

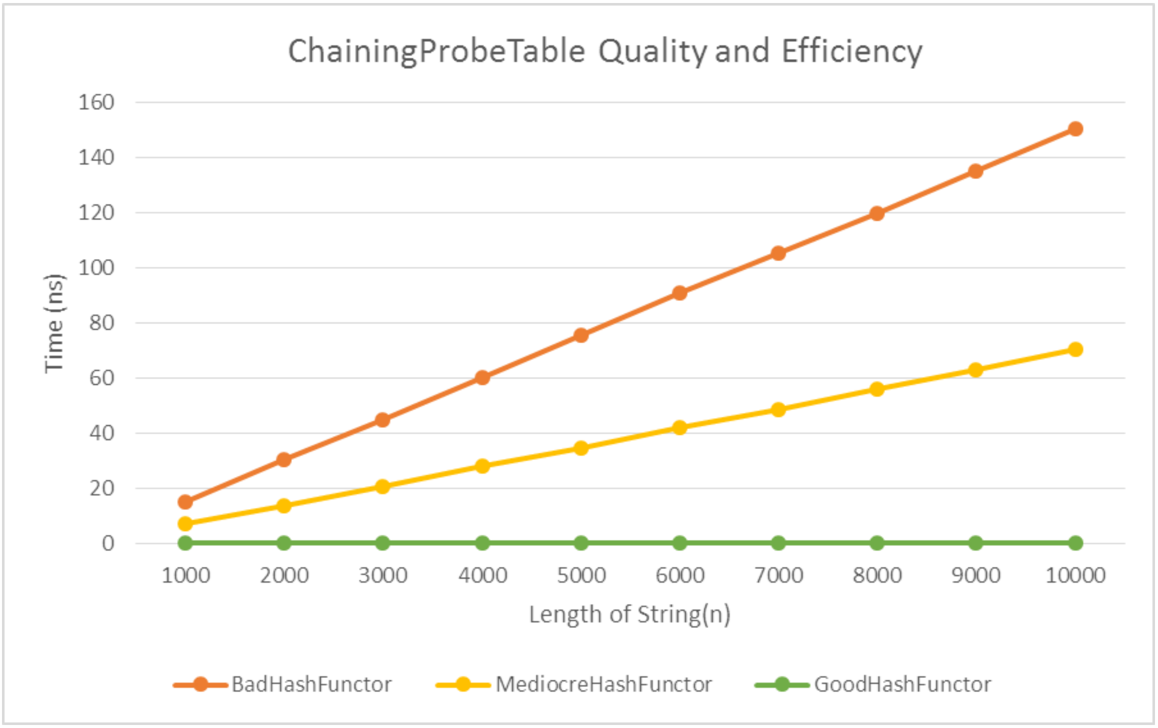
Q5 - Quad Probe)

The quality and efficiency of my three HashFunctors for Quad Probing add() function was to be expected. The Bad and Mediocre functions look to be following that of a O(n) line as the sample size gets bigger they get slower, this can be attributed not just to collisions but more to having to resize which is making the line resemble that of O(n). However with the GoodHashFunctor we seem to have met our goal which was to make it perfume as close to O(c) as possible. Though not every single add was exactly O(c), it was overall close enough that I think it is fair to call Quadratic Probing with a good Hash Function to be O(c).



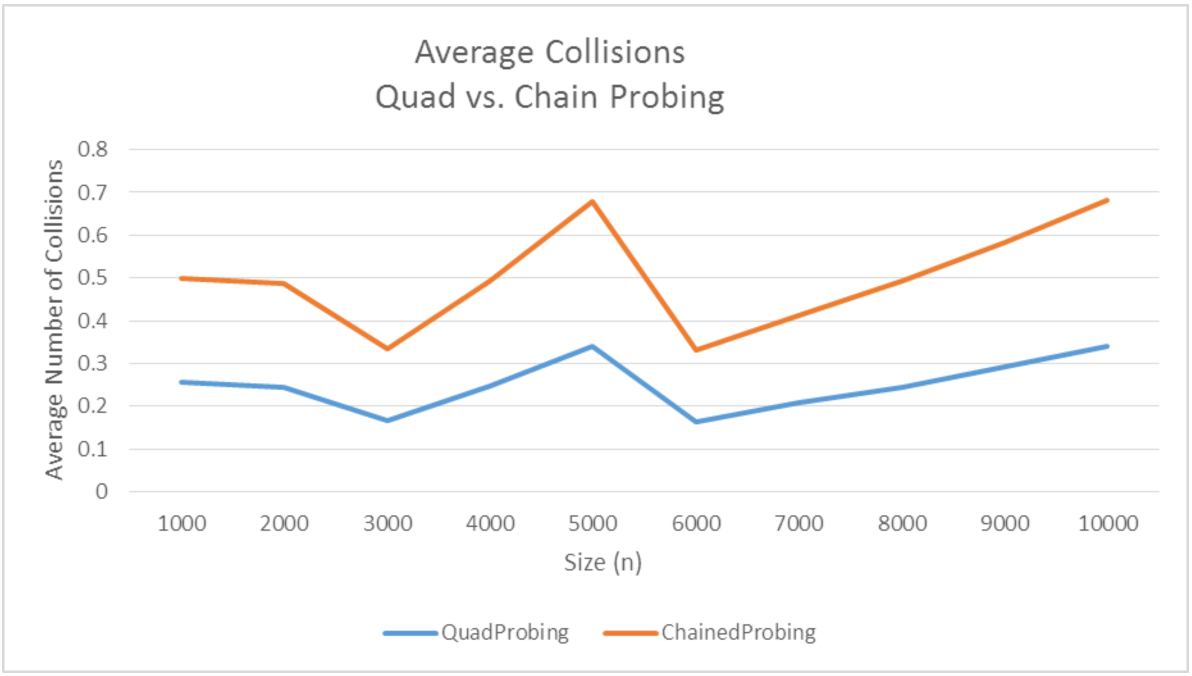
Q5 - Chained Probe)

The quality and efficiency of my three HashFunctors for a Chained Probe add() function was to be expected. The Bad and Mediocre functions look to be following that of a O(n) line as the sample size gets bigger they get slower, this can be attributed not just to collisions but more to having to resize which is making the line resemble that of O(n). However with the GoodHashFunctor we seem to have met our goal which was to make it perfume as close to O(c) as possible. Though not every single add was exactly O(c), it was overall close enough that I think it is fair to call Chained Probing with a good Hash Function to be O(c).



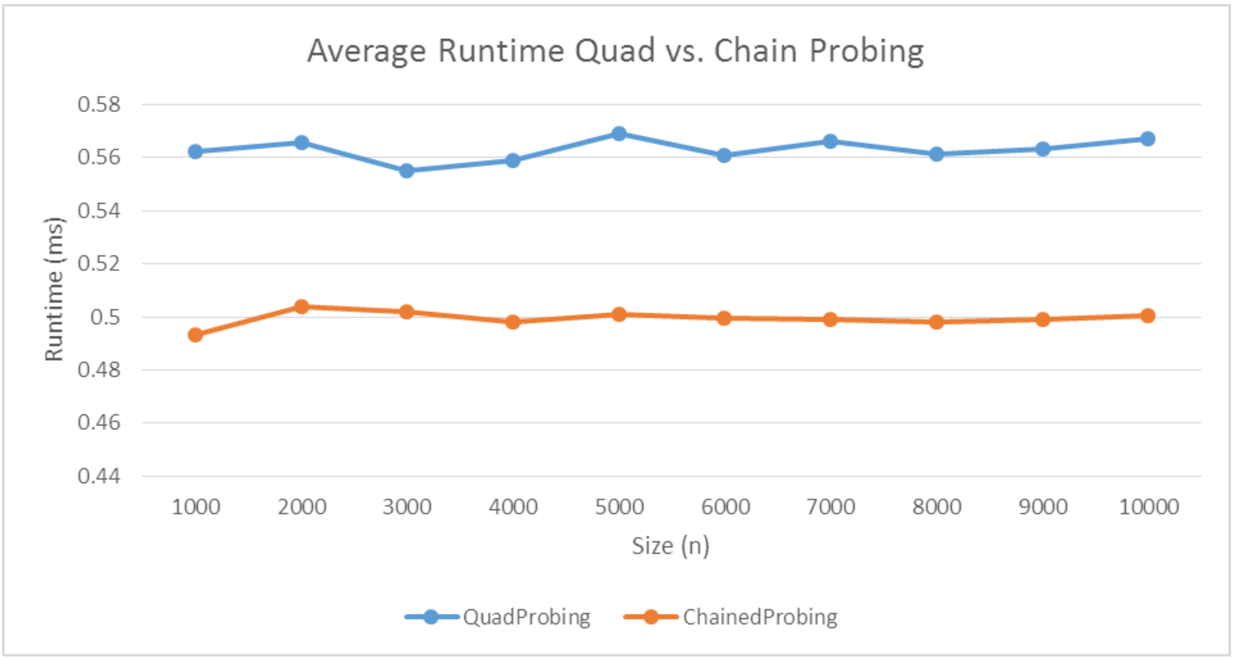
Q6 - Average Collisions, Quad vs. Chain Probing)

In the graph below we show the average number of collisions that occur on sample sizes ranging from 1,000 to 10,000 by steps of 1,000. Though marginally, QuadProbing seemed to cause fewer collisions compared to ChainedProbing. I think the reason this occurs is because in QuadProbing we are creating bigger and bigger arrays to hold our data depending on if the load factor has surpassed .5. So it makes sense that there will be fewer collisions because we will just make our array much bigger when the probability of collisions becomes too high. This looks good but it does mean there is a lot of resizing and copying data over and over, this I believe will show when we examine the Runtime of QuadProbing to ChainedProbing.



Q6 - Average Runtime Quad vs. Chain Probing)

Though average collisions are fewer and farer in-between for QuadProbing as seen in the graph above, that doesn’t tell the full story about runtime. Though the time it takes to have a collision and to find another spot is significant, in the graph below we see that ChainedProbing runs faster than QuadProbing. I think that even though we might be getting more collisions with ChainedProbing we don’t have to resize it which makes a big difference and is the reason ChainedProbing ran faster for me in this experiment



Q7)

The cost of my Bad and Mediocre Hash Functions both ended up resembling O(n). While my Good Hash Function ended up resembling around O(c) or our desired goal. I expected a bigger difference between my Bad and Mediocre Hash Functions but since they were similar in how they chose a Hash value it makes sense why they both seemed to follow the same line as the length of the string got longer and the result was more collisions as the String got longer. But with the Good Hash Function had around the same amount of collisions consistently even as the length of the string increased.

Q8)

The load factor is a tricky variable because if it is too low then there will be less collisions but much more resizing of the array. But if you have your array reset when the load factor is equal to .7 then there will likely be tons of collisions but not very much resizing. In this program we had the array resize when the load factor exceeded .5, this was a very good number for performance since lots of collisions is just as costly as resizing the array so having load factor at .5 strikes a nice balance on collisions and resizing.

Q9)

I would implement a remove function by using the getPosition function I have which gets the position of the item you pass into it. Once you get the position, it will be null if the item is not present, once you know the position you go to the spot and you set the Boolean isAlive to false and decrement the size of the array. So technically you wouldn’t actually remove it but now that spot in memory can be used again if we add another item in.

Q10)

Yes, it is possible. We would have to implement a generic Set that way whatever the user passed in as the type would become what is accepted throughout the program. Also the way I store items I would have to make able to store generics instead of only String objects as I have it now, but that would be simple as instead of saving String item it would just be E item.

Q11)

I spent 11 hours on this assignment including coding and the analysis portion.