Chandler Phillips
Instructor: Maciej Zagrodzki
TA: Jay Luther
CSCI 2270-100, Recitation 101
12/6/20
**Please Note Part B is on a different PDF

Part A Figures:

Figure 1:

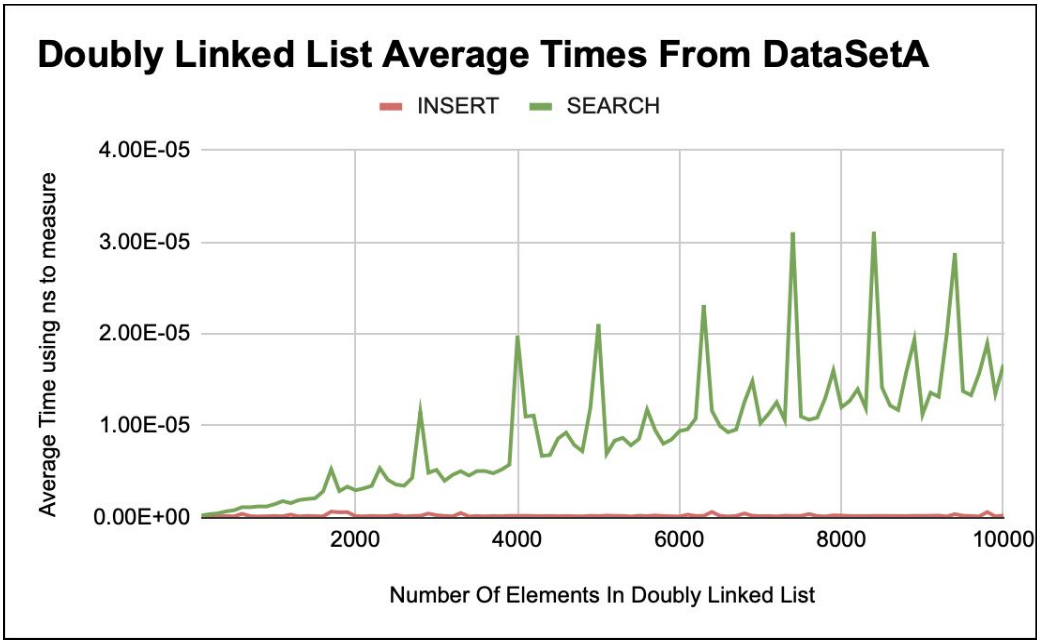


Figure 2:

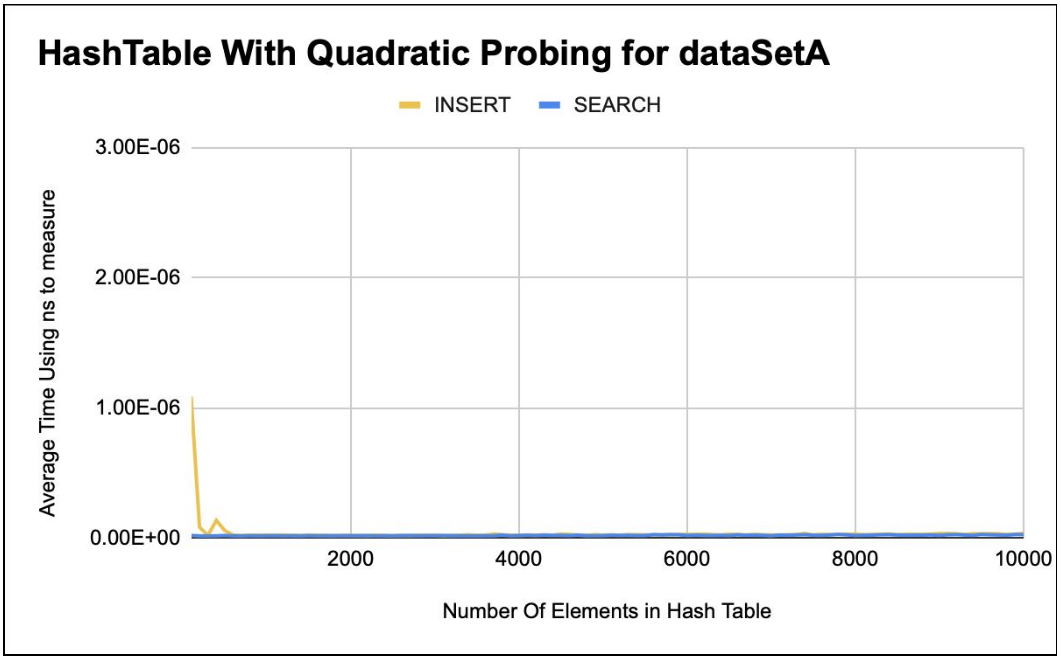


Figure 3:

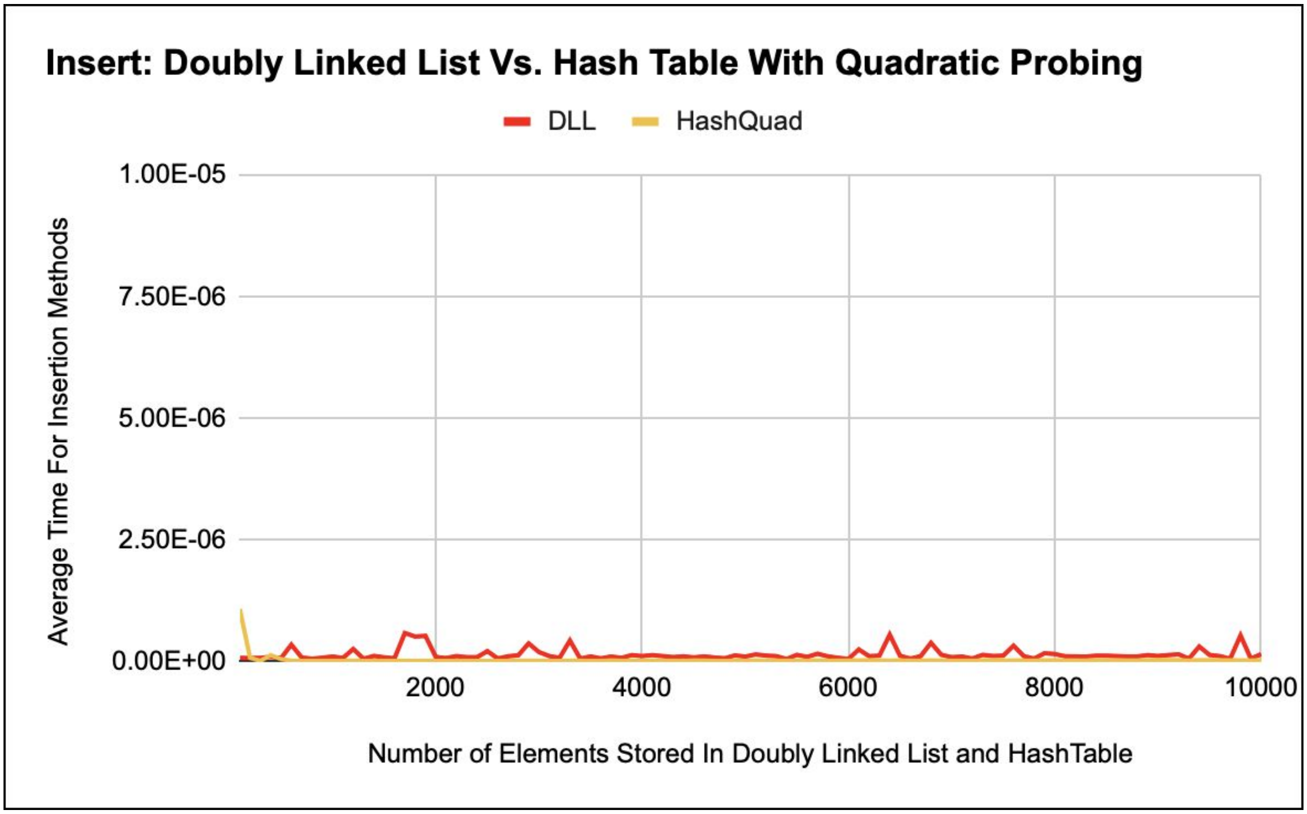


Figure 4:

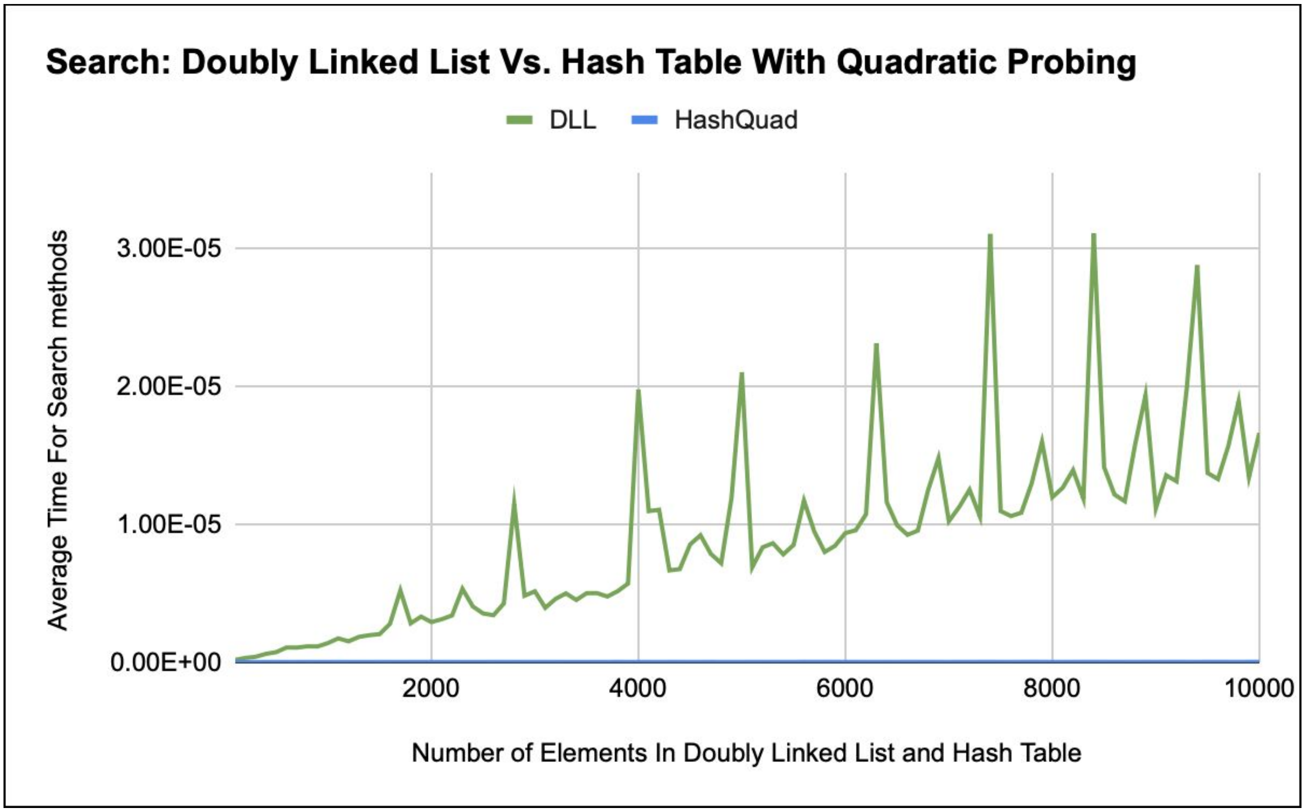
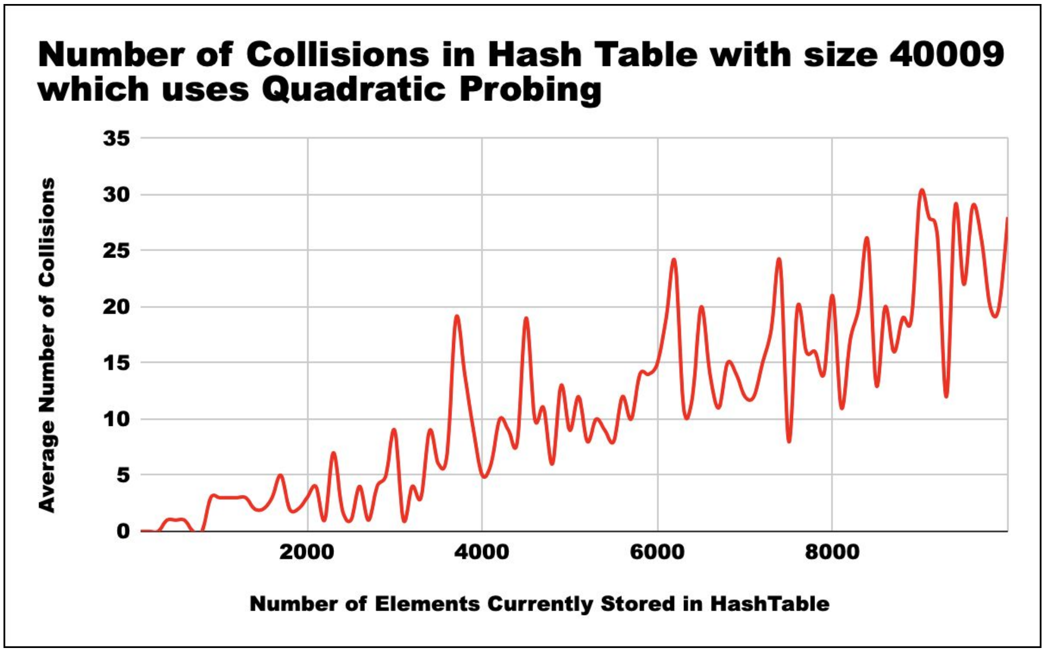


Figure 5:



Analysis Of Part A

In part A of this project, we created two different data structures, a doubly linked list and a Hash Table, which could be used for a Medical Tracker Company in order to retrieve the patient's medical history. The whole point of the software we made would use the patient's ID number in order to add it to our data structure efficiently, but also search for the patient's ID efficiently in order to retrieve the patient's medical History efficiently.

First lets begin with talking about the doubly linked list. When inserting data into the the linked list, in order to make the data structure more efficient I decided to add all the Data to the head of the linked list, this way we would be able to avoid any traversals when adding to the data structure making insertion as fast as possible, giving us Big O(1) for the inserts. However when it came searching in the linked list, there was nothing we could do in order to help the efficiency of the data structure, so in order to retrieve data we would have to traverse through the list until we were able to find the data we were looking for, so this gives our search method for a doubly linked list an average Big O(n) unfortunately. In our project in order to graph the time complexity for our Linked list we had to generate 100 random IDs already stored in the linked list and then search for them all, if more values were towards the end of the linked list, it would take a longer time to search for all of them compared to if they were all towards the head of the linked list, which is why we get the sudden spikes in our graphs of the time complexity up above (Figure 1).

Now let’s discuss the findings of the Hash Table data structure, which uses Quadratic probing when collisions occur. A Hash Table data structure is very nice because it uses the key value in order to Store and retrieve the data that it is holding and the key is unique. Setting up this data structure we decided to use a table size of 40009, which is a lot bigger than the number of data that we had to store into the data structure (10,000). When inserting into our data structure, because we use the key in order to store into this data structure, we take the modulo of that key value using the table size and store it at that index that results. This is an excellent way to store data, using the actual data that is stored with it, but if two key values have the same modulo value, collisions can occur, but this is what we are using quadratic probing for, which also prevents clustering and has a higher computational time. The more data we add to the hashtable the more likely it is to have a collision (as seen in figure 5), but because our hash table is so big compared to the data the collisions barely do anything to our time complexity. This means we have a really good insert time complexity which is really close to Big O (1) which we can see in figure 2. As for the search method for the Hash Table, it has a time complexity of Big O (1) too because the same way we insert is the same way we search basically, but instead of inserting data we just return true stating we found it.

So when we compare the two data structures, we see that the hash table would be the better choice. Although the hash table and the Doubly Linked list have the same insert time complexity, it is clear that the Hash Table is a better data structure to use because when searching in the data structures, the linked list has a time complexity of Big O(N) versus the Hashtable having a time complexity of Big O(1) seen in figure 3 and 4. Because figure 3 and figure 4 compare the times for the doubly linked list and the hash table, we are able to conclude that the hash table would be the better data structure for this software, if we didn't have the data on the same graph we would be able to look at the time complexity of each individual graph and see that the average time complexity for the search and insert methods highest value is way smaller in the Hashtable compared to the doubly linked list.