Anna Glass
CSCI 2270 Final Project
December 6th 2020

Medical Tracker Data Structure Comparison Report

Part A: Doubly Linked List vs. Hash Table Methods

Data Figures for DataSetA.csv

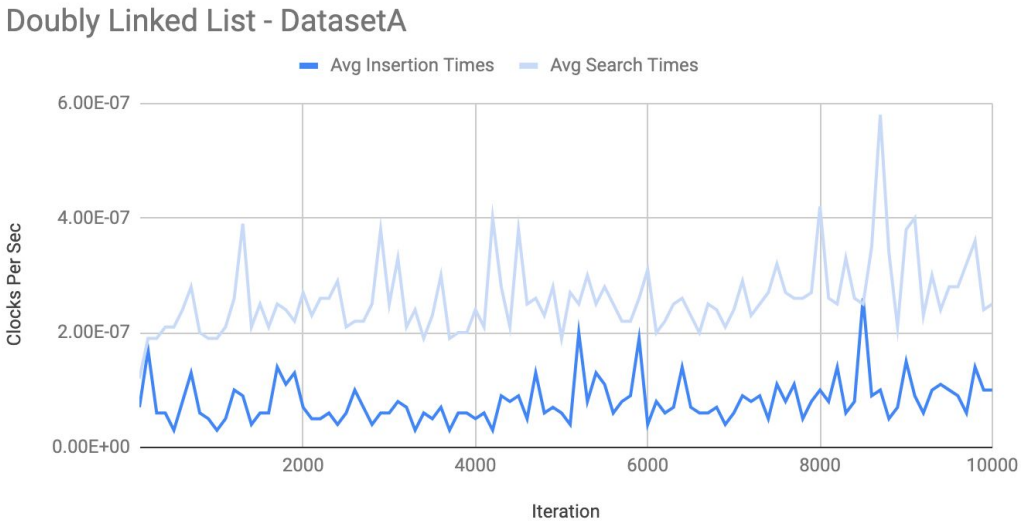

Figure 1: Average insertion and search times for doubly linked list using Data Set A, units of clocks per second with data points at iteration intervals of 100.

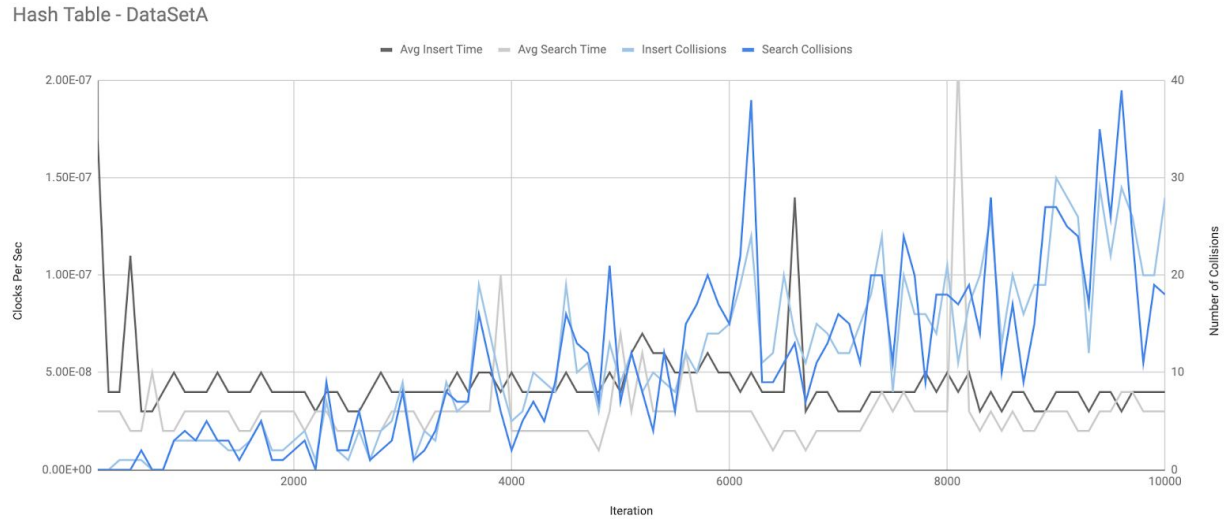

Figure 2: Average insertion and search times (left axis, in clocks per sec), and number of collisions (right axis) for hash table using Data Set A. Data points are at iteration intervals of 100.

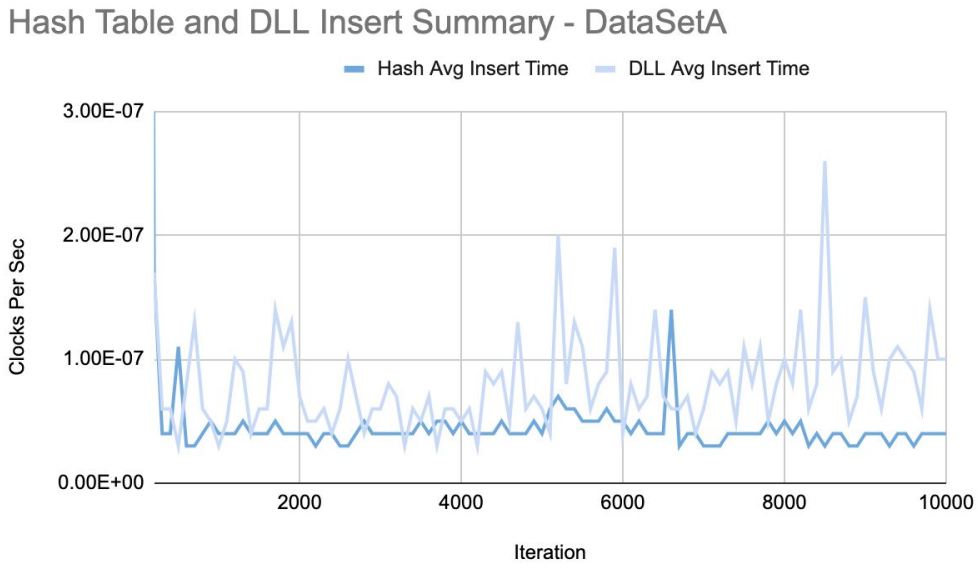

Figure 3: Summary of average insertion time for doubly linked list and hash table using Data Set A, units are in clocks per sec and data points plotted at iteration intervals of 100.

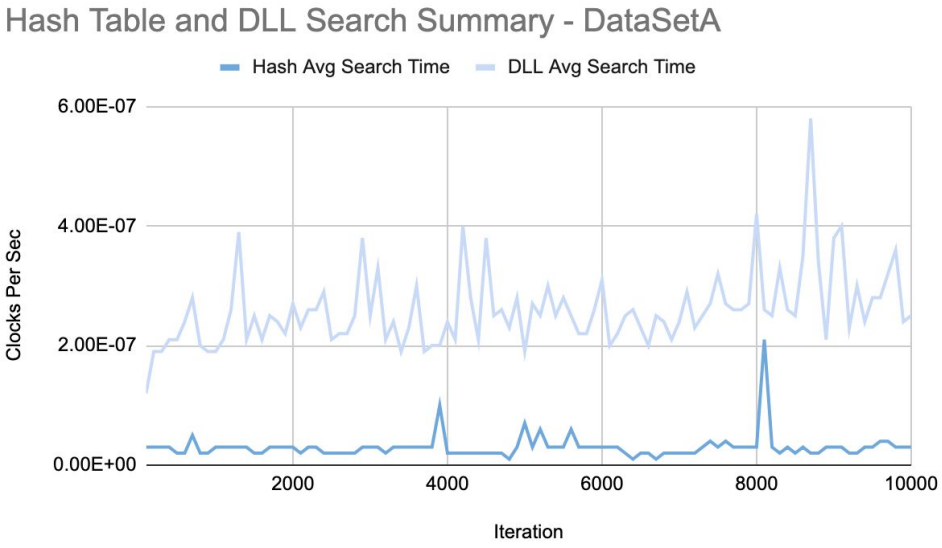

Figure 4: Summary of average search time for doubly linked list and hash table using Data Set A, units are in clocks per sec and data points plotted at iteration intervals of 100.

Analysis

The hash table consistently outperformed the doubly linked list in search times by approximately 1.5E-7 clocks per sec (Figure 4), and more frequently outperformed in insertion times, though by less consistent amounts (Figure 3). Because the insertion and search times for the hash table stayed consistently low over the interval, and according to the graph seem to be more predictable than the doubly linked list (in terms of variation in insertion and search times), the hash table has presented itself as not only the more efficient data structure for medical tracking, but also the more reliable and predictable structure if there were higher volumes of patient IDs in the future that required consistency in a data structure.

The hash function is more efficient because it doesn't store these patient IDs in any particular order, nor does it require traversal to search for an ID. It has an average search time complexity of O(1) while the average for DLL is O(n), hence the sizable difference seen in Figure 4. To find a particular ID in a hash table, we simply use the hash function and if the ID isn't found there, we use quadratic probing to find the next location it could be. For a linked list, you must start from the head of the list and check every node until the ID is found. The search function for a hash table checks a lot less locations than the doubly linked list search function, so it will always run more efficiently against time.

The average insert time complexity is O(1) for both structures, hence we see the data are more similar in Figure 3. I think the slight difference between these is because a doubly linked list requires adjusting pointers, while the hash function is a formula that instantly gives the location (with the chance of the necessitation of quadratic probing, which is also a simple formula). Figure 2 shows there is little correlation between the number of collisions and average insertion and search time, demonstrating the efficiency of quadratic probing.

Part B: Bubble Sort vs. Heap Methods

Data Figures for DataSetA.csv

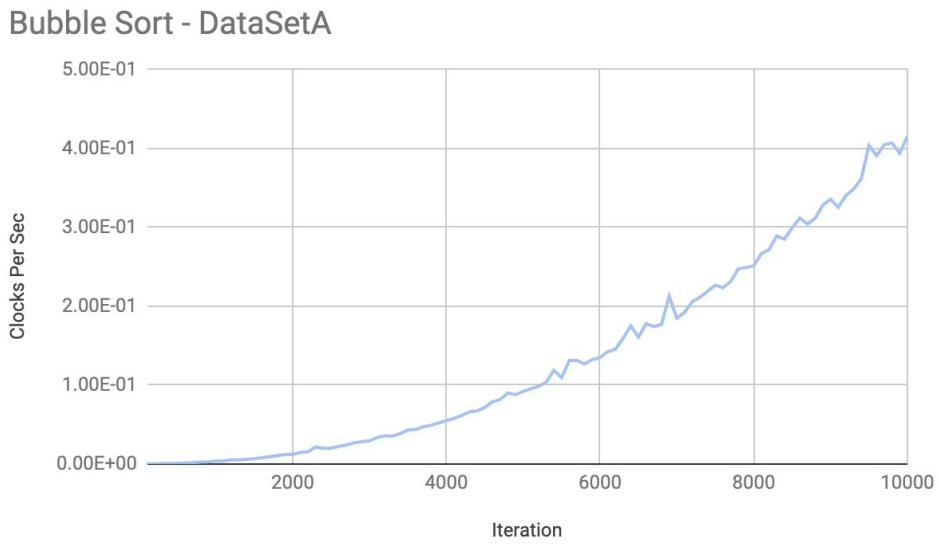

Figure 5: Average sort times for bubble sort using Data Set A, units of clocks per second with data points plotted at sorting of 100, 200, 300, … , 10000 medical IDs.

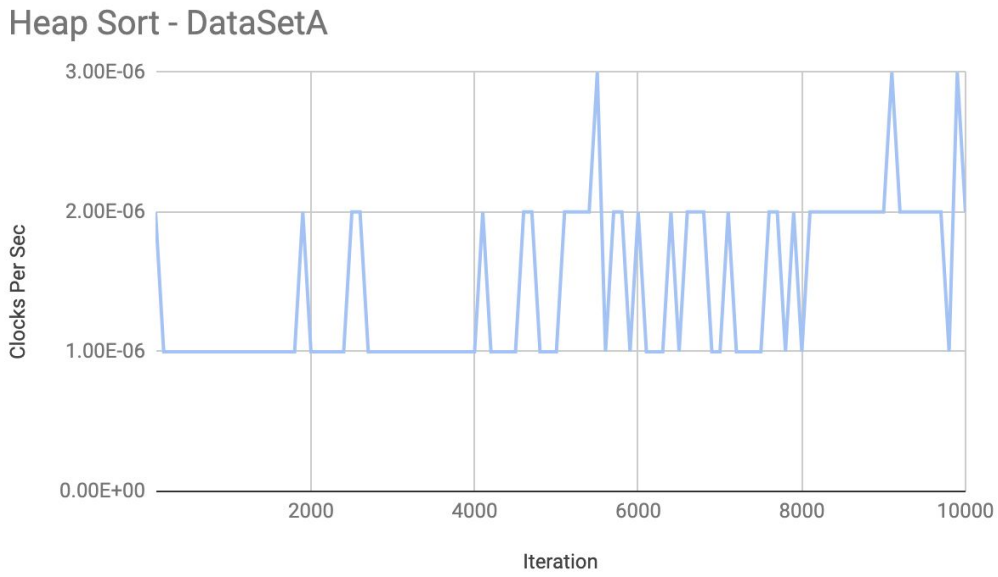

Figure 6: Average sort times for heap using Data Set A, units of clocks per second with data points plotted at sorting of 100, 200, 300, … , 10000 medical IDs.

Analysis

The heap sorting very clearly outperformed bubble sort, ranging sort times between 1E-6 and 3E-6 clocks per second (Figure 6) while bubble sort had sort times up to approximately 0.4 clocks per second (Figure 5), and was steadily increasing while heap sort maintained a constant range over the interval.

The heap sorting algorithm has a worst case time complexity of O(nlogn), where n is the number of elements, while bubble sort has an average time complexity of $O(n^2)$, which is extremely inefficient in comparison to heap sort. This is because bubble sort works by two for loops, comparing adjacent elements and repeatedly swapping elements that are out of order. It requires three "passes" of the array to correctly swap all of the elements, where the third pass has no swaps and purely tells the algorithm it is done. On the other hand, heap sorting using min heap divides it's input into sorted and unsorted sections, then finds the minimum of the unsorted section and moves it to the sorted section. It is more efficient than quicksort in a sense that using heap allows the sorting algorithm to find the minimum element quicker.

So, it is clear that heap sorting is much more efficient in storing medical IDs than bubble sort, and based on Figures 5 and 6 heap sort would be more suitable for future use in case the data size were to increase above 10000 IDs. Figure 6 shows that the sort time for heap ranges between two fixed values (at least on this interval), while bubble sort is a strictly increasing curve in Figure 6. This means that if we were to extrapolate time data beyond the scope of the graph, it is likely that bubble sort would have a higher sorting time while heap sort would remain relatively the same, hence heap is the more predictable, reliable, and efficient data structure for Medical Tracker Company.