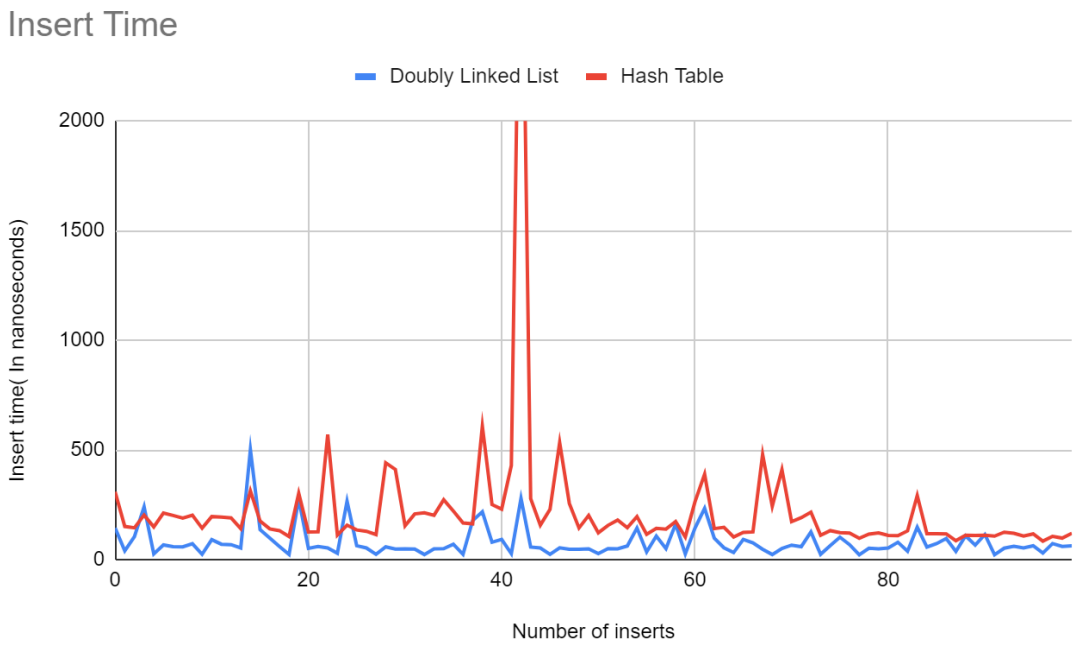
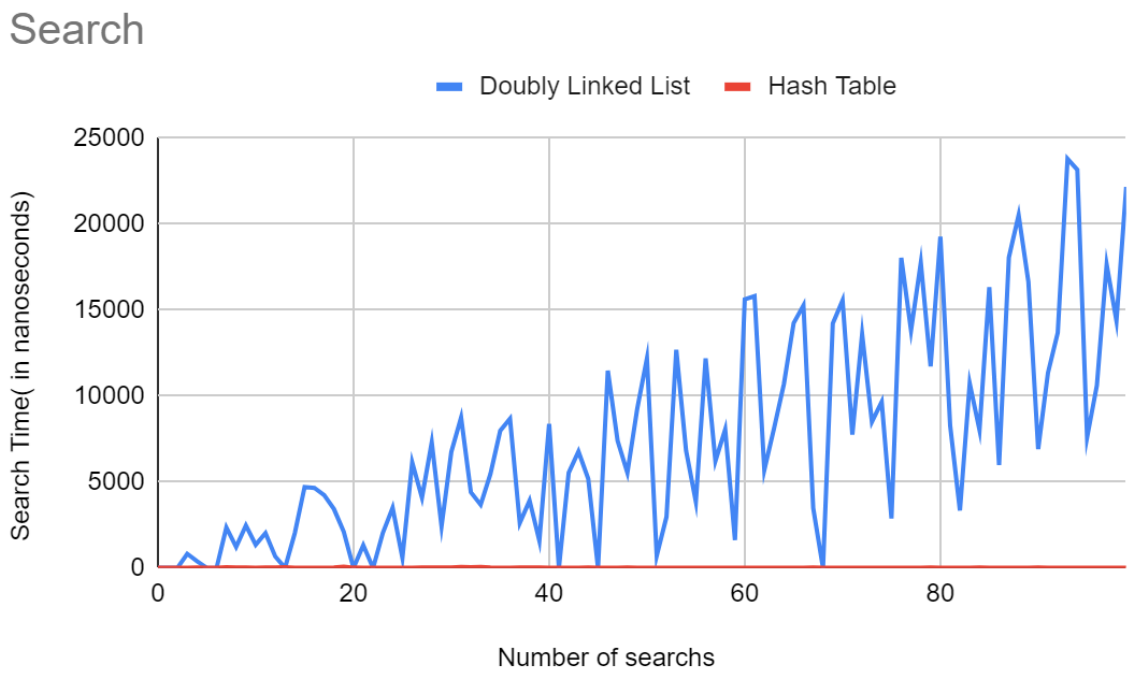
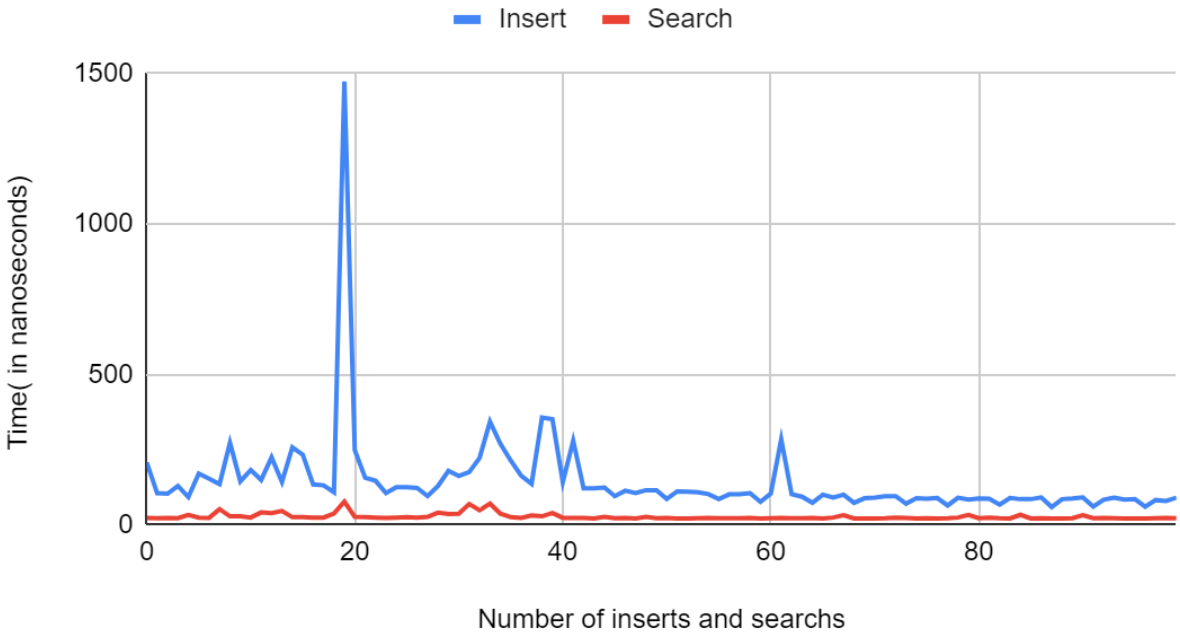
Niketh Gorla
CSCI 2270: Data Structures

Part A

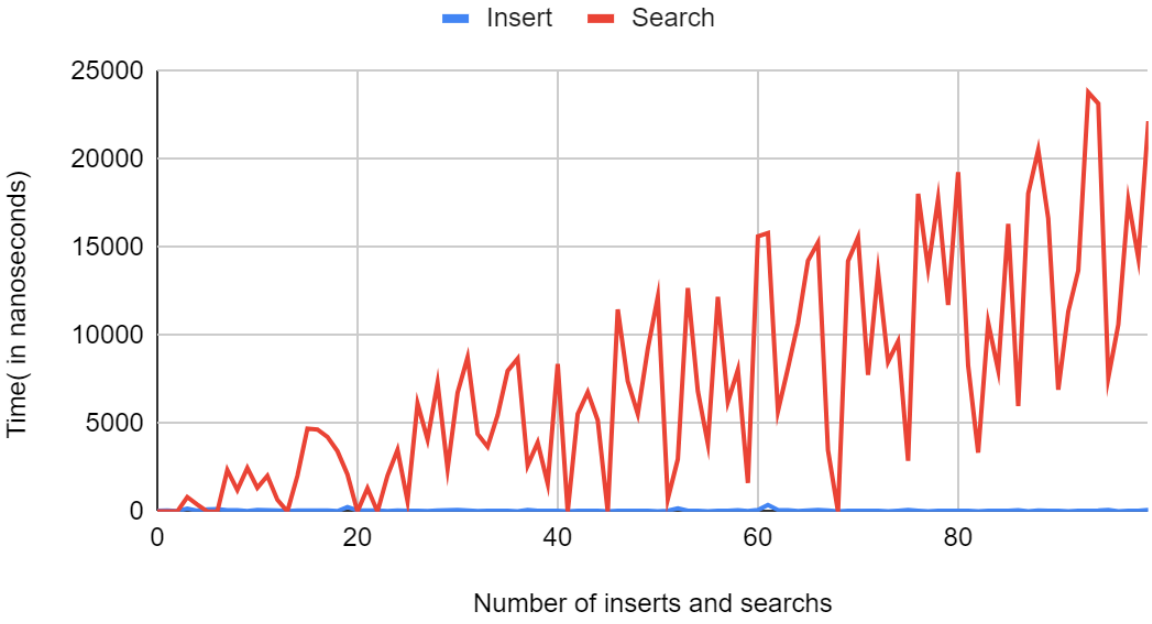




Hash Table

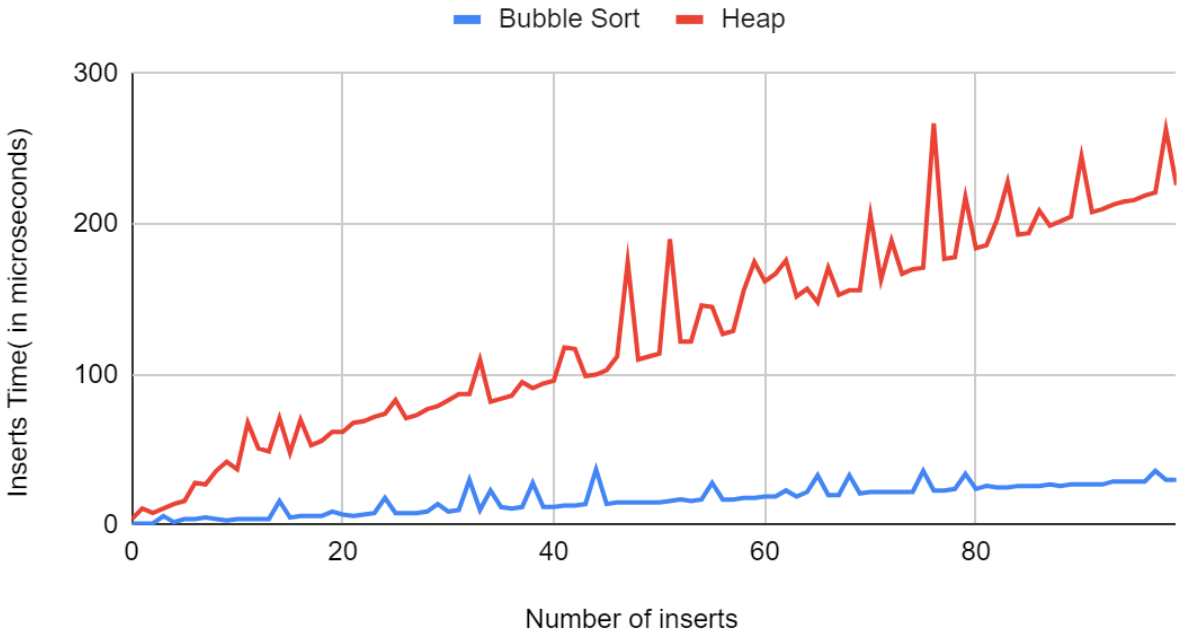


Doubly Liked List

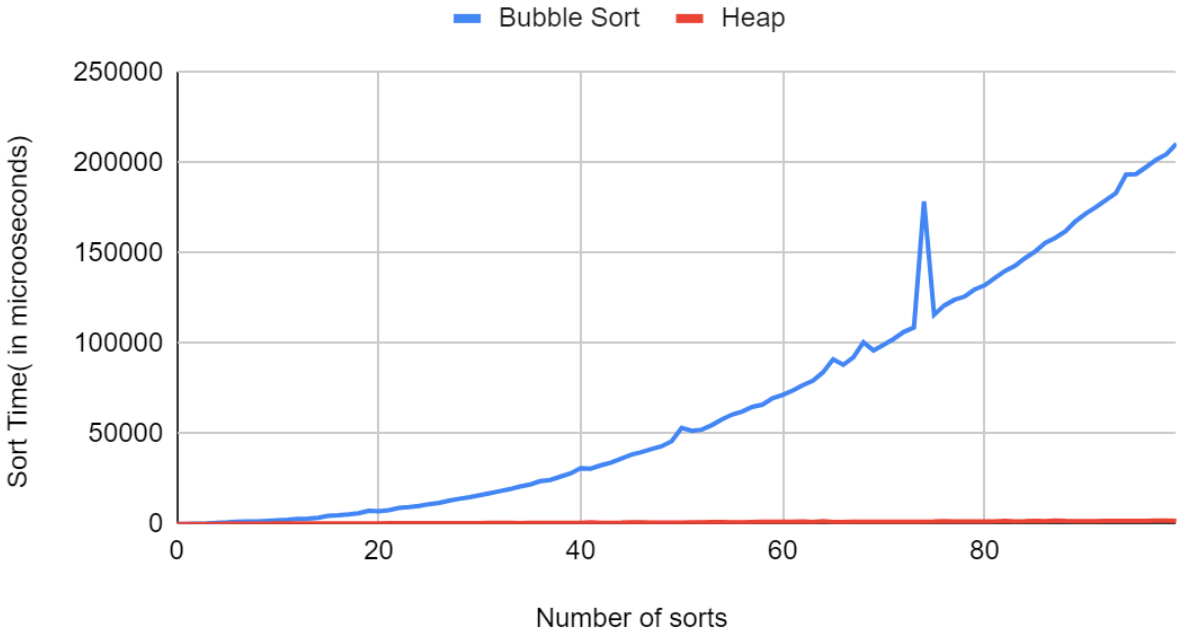


Part B

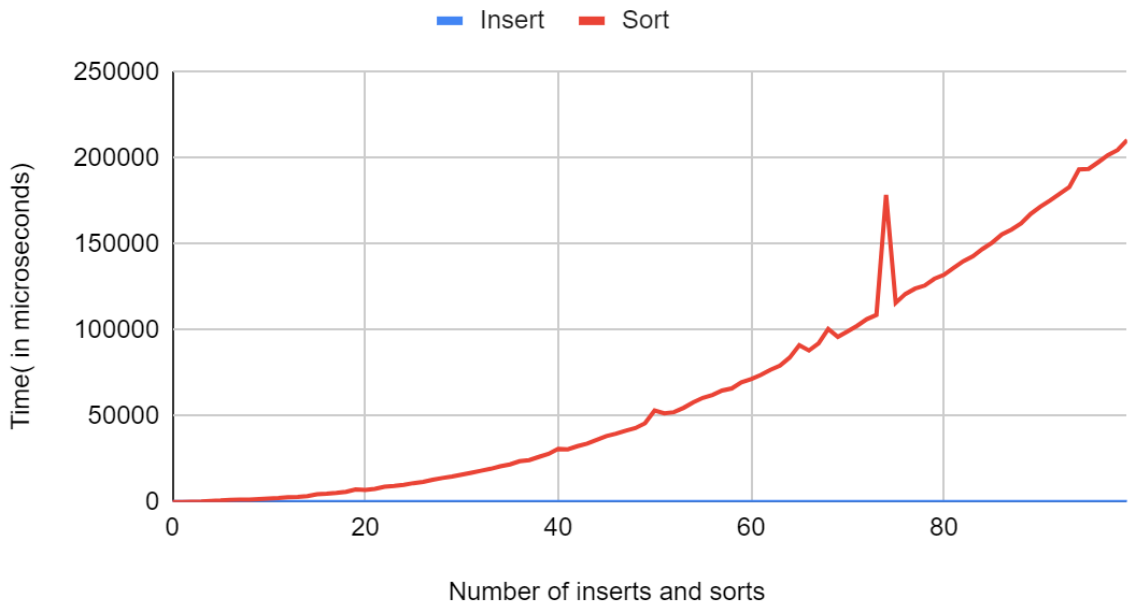
Insert


Bubble Sort
Heap
300
200
100
0
Inserts Time(in microseconds)
0
20
40
60
80
Number of inserts

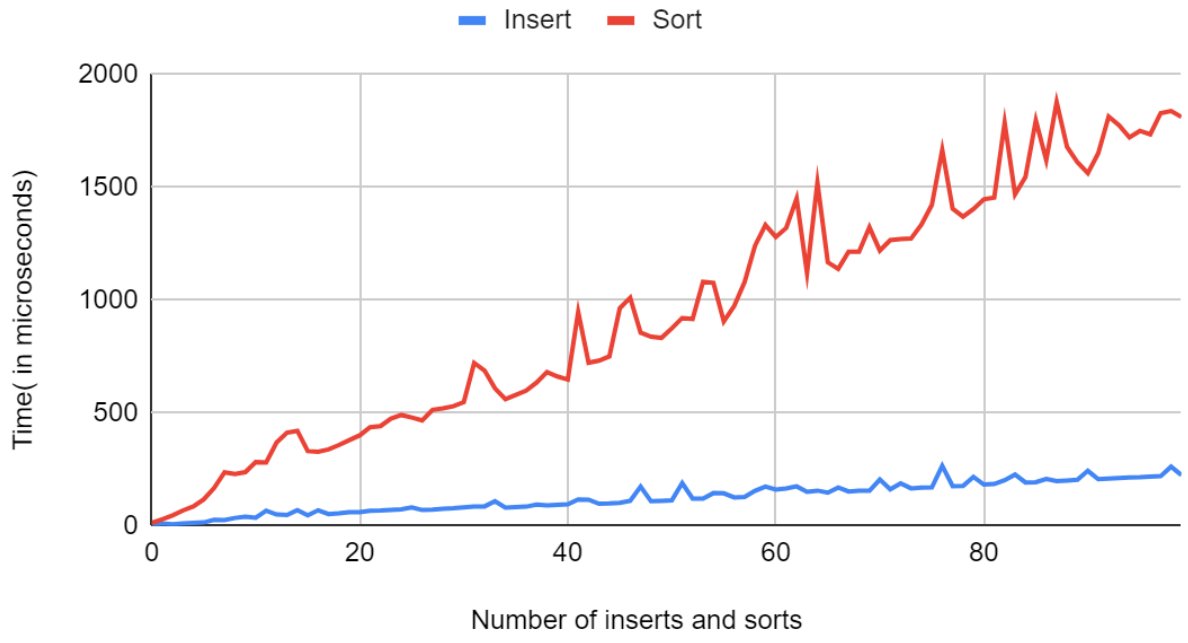
Sort


Bubble Sort
Heap
250000
200000
150000
100000
50000
0
Sort Time(in microoseconds)
0
20
40
60
80
Number of sorts

Bubble Sort



Heap



Overview:

Over the course of this project, I learned a lot about the different types of data structures and their implementations. One of these data structures is the doubly linked list. In the doubly linked list, the time complexity of the insertion into the list is O(1) due to the fact that I inserted at the end of the list every time. Because I had a tail pointer within the class of the list, it has a constant time complexity. The search for the doubly linked list has O(n) time complexity due to the fact that we need to traverse the entire list to find the item we are looking for, creating a linear time complexity. The next data structure is a Hash table. Insertion into a hash table has a higher time complexity than the doubly linked list due to the fact that the Hash function must be applied before insertion, and the fact that there are collisions which can cause a longer insertion. Despite this, the insertion into a hash table is still O(1) due to the fact that there are far more buckets in the table than there are pieces of data, therefore the number of collisions is low. The time complexity of the search in the hash table is also constant, due to the fact that we can look at the specific buckets which correspond to the hashed value. The next two types of data structures that I used are the Bubblesort and the Min Heap. The insertion into both of these is O(n) because we simply insert the data to the end of the array, which we have stored as the number of elements. Despite both of these insertions being constant, the time complexity of insertion into the heap is much greater than that of the bubble sort. This is due to the fact that after every insertion in the heap, we must heapify the array in order to maintain the validity of the heap as a data structure. The time complexity of the sort in both of these cases is different. In the best case scenario does the bubble sort, the time complexity is only O(n) if the array is already sorted. This is due to the fact that it must only be traversed once to confirm that it is already sorted. In the worst case scenario, it is O(n^2) because it must traverse the array n times, pushing the next highest to the end every time. In the heap, the sort is the time complexity of O(n). This is due to the fact that in the heap it must pop each element out of the array and then heapify the array. If there are n elements, then it must pop n number of times. The time complexity of sorting in the Min heap is much lower than that of the bubble sort due to the fact that the heap is already sorted at every insertion by heapify. The hepa is constantly maintained making the time complexity of sorting at the end much faster. Given the evidence that I have presented, I think that the best data structure is the heap due to its constant maintenance which allows it to be sorted very quickly. I think that this makes this data structure very useful for many functions in which every piece of data must be sorted as it is inserted.