Benjamin Bowser

CSE 274 UA

Homework #5

Speed Analysis of Linked List and Sorted Linked List Implementations in Java

During my development of both the Linked List and Sorted Linked List class implementations for storing objects (Cars used in this test), I designed both classes to work in almost identical ways, with the only exception being that the sorted version of the linked list would check the compareTo method written in the Car class before adding the object to the linked list. This results in the objects being sorted, first by year, then by string comparisons based on the other fields of data in the Car class. To run comparisons, I checked how long it took in milliseconds to complete the task I requested each class perform. Each class utilized enums of color, make, and model, then used random numbers to generate a year between 1990 and 2018. Although models of cars would be assigned to makes that do not manufacture that model of car, having a wide set of data was the most important feature to have occur, to ensure a wide set of hashes being used inside of the hash table. When checking over the data visually, multiple collisions occurred resulting in objects being put into either linked lists or sorted linked lists respectively. This was the goal of adding 20,000 entries to each list to ensure that a wide set of data was used for testing. I then ran this process for each implementation 200 times. After the process completed, the time in milliseconds was printed for each implementation to the console. In the testing, I found that the sorted linked list generally took a shorter period time. The task requested in each iteration was to create a car object from the enums and a random year between 1990 and 2018. The car was then added to the set, followed by calling contains on the object.

The object was then removed, checked if the set was empty, then added the object back to the set. Once this was done, the set was converted into an array (but not printed).

In this testing, I found that in most cases, the sorted linked list implementation resulted in a faster completion time. Although it completed faster, it was never significantly faster than the Java linked list implementation. For example, when I ran the above test (adding 20,000 objects) 5 times, the Java Linked List took 10,831 MS, while the Sorted Linked List took 10,019 MS. In this test, the Max Load Factor was set to 1. I then changed the load factors of both classes to 0.1, which resulted in the Java Linked List taking 16,343 MS and the Sorted Linked List taking 15,510 MS. These tests show that the lower the Max Load Factor, the more times the hash table had to be resized, resulting in a longer runtime.

In another test, I observed that adding clear at the start of each iteration of the 20,000 iteration for loop, resulted in the times of the processes dropping all the way to 72 MS for Java Linked List, and 39 MS for the Sorted Linked List. This discovery has a direct correlation with the findings of lists taking longer if they have more items added when resizing.

Although the testing showed that both versions of implementations resulted in similar results, I can conclude that the Sorted Linked List implementation took a shorter period of time due to the lookups for contains and remove. Because the list was sorted, it took a shorter period of time to find objects that had lower years, than if they had higher years, because they occurred at an earlier point in the Sorted Linked List. Additional consideration must be taken of the data structure types of each of these structures. Although technically both are of linked list types, the Java Linked List stores it's data inside of an array, while the Sorted Linked List uses a Node class to store data. Calling next and iterating through a Linked List can take more time than using a for loop to iterate through an array of data. In addition, linked lists in both implementations that

had only one entry in their hash table field would have had the best efficiency of lookup time, due to the O(1) time of searching the hash table for the linked list, and having the object being searched for being the first item to show up in the list. While the efficiency of iterating through these would be similar to O(n), structures that contain fewer entries will naturally result in shorter search times.