COSC2123 Algorithm and Analysis

Assignment 1 Report

Japan Patel – s3595854
Danny Ho – s3603075

Report: Experimental Setup

Background:

We are using a modified version of Multiset Tester that captures Nano time between the multiset add/remove/search methods, adding the time for the operations to complete. We run each test for 3 times and using the average for comparison, this is done in each implementation. We are testing these under two scenarios as follows.

Scenario 1

Number of Additions: 1000
Number of Deletions: 250, 500, 750, 1000

We used 1000 additions for testing and not 100 or 10,000. So that we can accurately measure time in human understandable format when we test various additions and deletions. The reason we are using 25%, 50%, 75%, 100% deletions is that we can get an accurate representation of how the number of deletion affect the time in a large set of multisets. So as the multiset increases, the ratios of removals vary and the removals increase in size, up to the equal number of additions to removals. This we can see how the number of removals affects the time.

Scenario 2

Number of Additions: 1000
Number of Deletions: 1000
Number of Searches: 250, 500, 750, 1000

For this scenario, we are using the same number of additions and deletions and varying the number of searches to see which data structures can find the item in minimum time when all the data structures are given the same number of additions and deletions. And the reason of using 1000 addition is so we can get a time that is reasonable and understandable. We are generating 25%, 50%, 75%, 100% search entries and testing them so that we can get measurements of the search results from each data structures as the number of search entries increases in a growing multiset.

The way we are testing these scenarios is that we are generating files based on these scenarios from a Data generator class so that the data in each file is randomised that we created. But we are testing the same data set file in a modified multiset tester with different implementations which shows the time that it took to do each operation. The results we got from these scenarios are as follows.

Scenario 1 Results (varying ratio of additions and deletions)

Just so we know how much time it takes to do add operation in different data structures, Table 1.1 shows that the average time in millisecond(ms) that it took to do add operations in different implementations

Number of additions	Linked List	Sorted Linked List	Binary Search Tree	Hash Table	Balance Tree
100	5.90	16.32	1.71	0.32	0.61
500	8.45	12.98	2.55	1.11	2.10
1000	14.75	18.38	3.21	1.42	2.95

Table 1.1: Average time(ms) of additions

From the table 1.1 we can see that Hash table took the least amount of time to do add operations. While sorted linked list took the longest to do add operations as it was inserting values in ascending order using selection sort which has an average complexity of $O(n^2)$. From the Figure 1.1 we can see that even when the multiset is increasing the time it takes to do add operation is not that significant in Binary search tree, hash table, balance tree but it is very significant in linked list and sorted linked list this is due to the insertion complexity of linked list which is O(1).

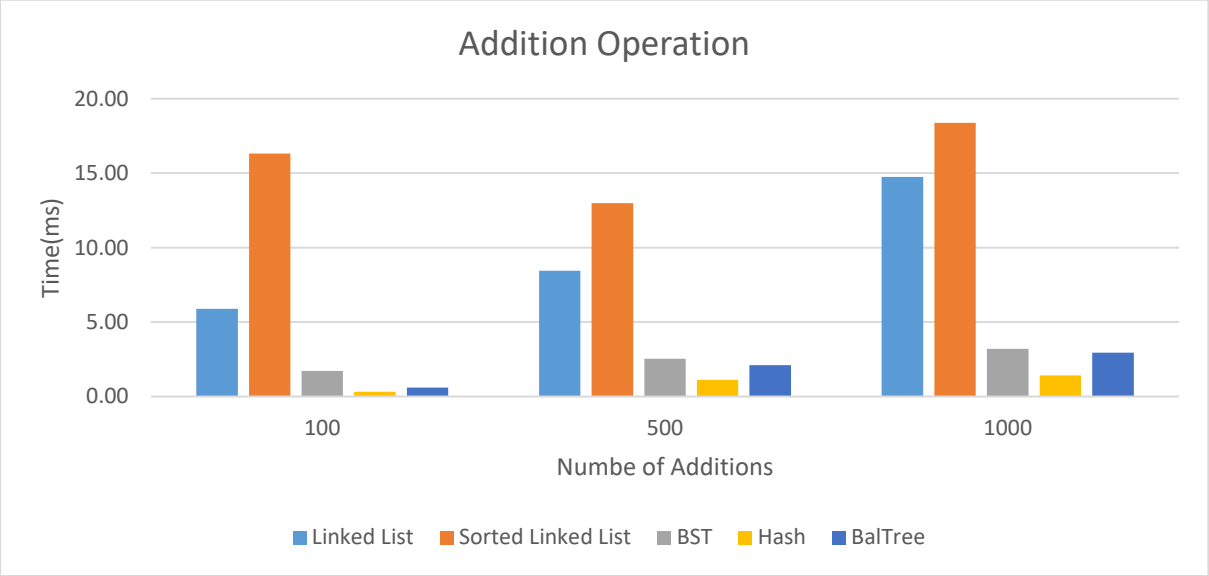

Figure 1.1: Average time(ms) of additions

For the deletions of values from different, we had a fixed amount of additions and varying number of deletions as the multiset grows. The results from this operation are found below in table 1.2 and Figure 1.2

Number of Deletions	Linked List	Sorted Linked List	Binary Search Tree	Hash table	Balance Tree
250	10.27	11.37	0.83	0.38	1.21
500	13.12	14.05	1.19	0.69	1.75
750	14.53	15.79	1.49	0.87	1.94
1000	14.79	16.20	1.58	1.02	2.18

Table 1.2: Average time(ms) of removals

From the table 1.2 we can see that the average time for deletion a value from a data structure. We observed that increasing multiset increases the overall time, However the interesting fact that we found was that. In all the implementations, the insertion time is very close to the deletion time but deletion is slightly faster than insertion. However, if we think about it practically the timings are very similar and the difference cannot be seen easily.

Even when the multiset grew and deletion grew, the difference between the 25% deletion and 100% deletions was not that significant. However, if we keep growing the size of multiset, the difference will be significant. So, we evaluate from this is that growing multiset and grow deletion operation, increases the time of operation.

The least amount of time for deletion operation was done by Hash Table with binary search and balance tree following very close. While the linked list and sorted linked list took the most amount of time to do deletion operation. The reason for this is that the complexity of linked list is O(n) which increases the time significantly compared to of hash table which has complexity of O(1). The difference of time can be seen significantly in figure 1.2

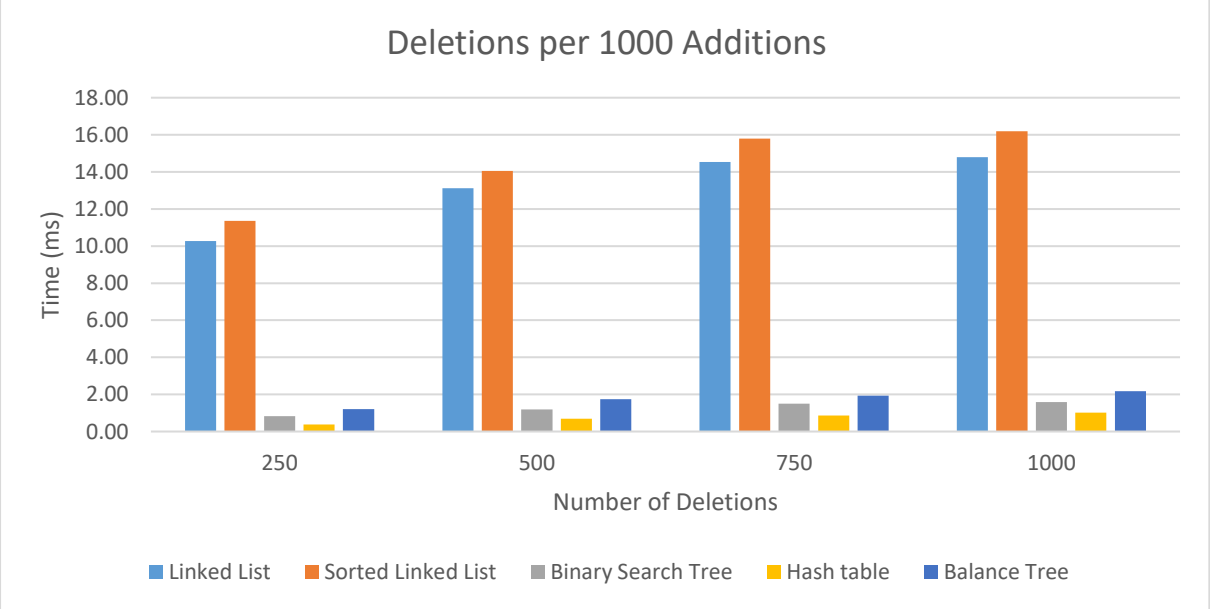

Figure 1.2: Average time(ms) of removals

Scenario 2 Results (varying ratio of addition/deletion to search)

For Comparison, we tested search operation two ways and the change in result was significant. So, to measure this difference we generated the with only add operations and search operations and the results from that tests are below in table 2.1

Number of searches	Linked List	Sorted Linked List	Binary Search Tree	Hash Table	Balance Tree
250	12.84	13.93	0.88	0.40	0.37
500	10.90	17.33	1.32	0.72	0.64
750	18.34	21.11	1.64	0.84	0.87
1000	21.27	26.66	2.22	0.88	1.08

Table 2.1: Average time(ms) of searches

From the table 2.1, we can see that the average time for searching a value from a data structure (without deleting any values). And table 2.1 shows average time of search result where the number of additions and deletions were the same and varying searches.

Number of searches	Linked List	Sorted Linked List	Binary Search Tree	Hash Table	Balanced Tree
250	10.26	10.54	0.79	0.26	0.28
500	11.68	12.37	1.06	0.48	0.46
750	13.53	13.62	1.26	0.54	0.60
1000	15.27	17.14	1.60	0.70	0.76

Table 2.2: Average time(ms) of searches with the same number of additions/deletions

We observed some interesting results for this operation. Comparing table 2.1 and table 2.2 we can clearly see that there is a significant time difference in both when we made a modification of randomly removing values as well. We observed that this is because when we do only search operation. It takes more time than we generate a data file that has random deletions placed inside it. So, for multiset, we tested the implementations with the same number of additions and deletions and a varying number of searches there might be a value that was added, deleted and then searched (didn't exist anymore) that is why the average time for the search result was different significantly.

From figure 2.1 and figure 2.2 (next page) we can see that hash table still takes the least amount of time searching value compared to sorted liked list or linked list. This is because the average complexity of hash table is O(1) where as the complexity of linked list is O(n). On the other hand, comparing hash table with balanced tree have very similar result as the complexity of balanced tree is $O(\log_n)$

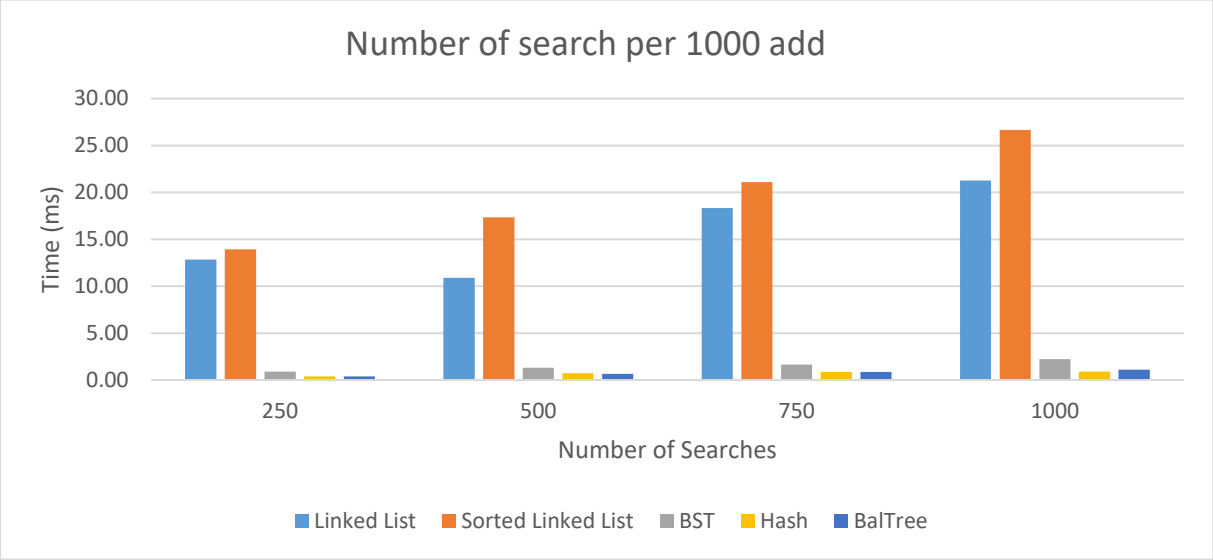

Figure 2.1: Average time(ms) of searches

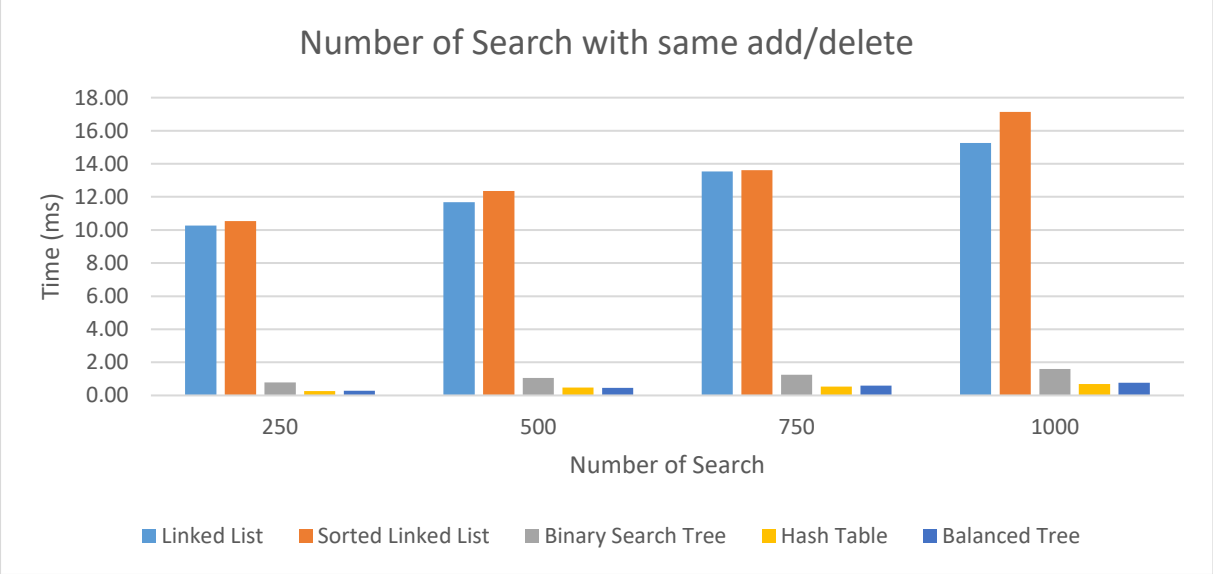

Figure 2.2: Average time(ms) of searches with the same number of additions/deletions

When we did only search operations (figure 2.1) we can see that increasing the multiset and increasing searches in a growing multiset increases time which we observed same in scenario 1 (figure 1.2). However, we also observed that Linked list, sorted linked list and Binary search tree takes more time in deletion than search operation whereas Hash table and balanced tree take less time searching than deletion operation.
And when we compare the search operation with same number of add and delete (figure 2.2) with deletion operation (figure 1.2) then time for search and delete are similar.

If we observe all the figure 1.1, 1.2, 2.1, 2.4 we see that the search operation takes the least amount of time whereas the add operation takes the most amount of time. We believe that all the measurements that we took had a lot of different factors that affected the result such as our way of programming things and the way we tested it for example, we tested our multiset on titan.csit.rmit.edu server so sometimes the time of operations were different at

certain point of day to another due to server lag or the other operations undertaken by server. So, the results we took are not perfect but do tell us an accurate representation of the algorithms and data structures.

Recommendation

For add operation we recommend using Hash table. However other data structures like binary search tree and balanced tree had very similar results to that of hash table so we also recommend that.

For delete and search operation we also recommend using Hash table. We also recommend binary search tree or balanced tree as these also had very similar results to that of hash table and are faster than linked list.

Table 3.1 shows the overall most recommend data structure to the least recommend data structure.

Most Recommended				**Least Recommended**
Hash Table	Balanced Tree	Binary search tree	Linked List	Sorted Linked List

Table 3.1: Recommended data structures

----- THE END -----