Exercise 1 was an exercise in evaluating the time complexity and performance of dictionary operations when implementing sets through three different types of data structure containers. These three containers are the Linked List, the Binary Tree, and the Hash Table.

Thus, Hash Tables should give us the best performance when it comes to both search and insert dictionary operations at both O(1) average case. Binary Search trees are better at searching O(log(n)) on the average case than Linked Lists, while Linked Lists are better at inserting (O(1)) than searching O(n). Here is a summary of best case, average case, and worst case complexity of the three data structures:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data Structures** | **Best Case** | | **Average Case** | | **Worst Case** | |
| **Search** | **Insert** | **Search** | **Insert** | **Search** | **Insert** |
|
| **Linked List** | O(1) | O(1) | O(n) | O(1) | O(n) | O(1) |
| **Binary Search Trees** | O(1) | O(1) | O(log n) | O(log n) | O(n) | O(n) |
| **Hash table** | O(1) | O(1) | O(1) | O(1) | O(n) | O(n) |

The results anticipated by our theoretical analysis is exactly what we observed in our actual implementation as can be seen by the graphs shown below.

**For Searching:**

We expect a linear search time O(n) for Linked List, which is we got. We expected binary trees to be better performing during searching because the average case is better than Linked List, and for Hash Tables it’s constant O(1) as expected. So during runtime, Hash Table << Binary Tree << Linked List.

**For Insert**

The graph below shows the runtime performance of our three data structures during insert dictionary operation.

The graph above is consistent with our expected results. Average case Insert for Hash Tables and Linked Lists should be faster than Binary Trees because insert for binary trees perform at O(n) time worst case and O(log(n)) average case, while Hash Tables and Linked Lists insert in O(1) time. Hash Tables are generally faster than Linked Lists during runtime because of the nature of hashing and indexing.