Name: An Le

Account Number: masc0369

Class: CS 310

Prof. Alan Riggins

Final Project Report

1. **Hash Table:**
2. **Insert method:**

Since hash tables are always array-based structures, the complexity for inserting one element is O(1). In particular, for this project, unordered linked list is employed for the hash table. Therefore, the complexity of the insert method is actually the complexity of method addFirst in LinkedListDS, which is always O(1) for the best case, average case and the worst case.

1. **Remove method:**

The remove method using chaining of the implemented hash table has the same complexity to the remove method of unordered linked list. In the worst case, it has O(n) complexity. However, the situation when many items are hashed to the same key rarely happens. And thus, in the average case and the best case, the complexity is O(1).

1. **Contains method:**

Contains method has the complexity O(1) for best and average cases. That is, when there is one element in each linked list, the time to find one key is instantaneous no matter how large the table might be. But in the worst case, when there are many values added to the same key (the same linked list), the method takes O(n) to loop through the linked list until the target value is found.

1. **Binary Search Tree:**
2. **Insert method:**

The insert method is an iterative method. It starts from the root and compares each node it goes through with the node that needs to be inserted. If the need-to-be-inserted node is less than the compared node, it goes down to the left child of the compared node. Or if it is greater than the compared node, it goes to the right child. The whole process is repeated until it reaches the leaf node and decides which side it stays. For each comparison, the number of nodes that need to be compared is cut into halves. Therefore, the complexity of this insert method in average is O(log n). When the input is in order, the worst case complexity is O(n).

1. **Remove method:**

The remove method calls an iterative method to find the node that needs to be removed and analyzes if that node has 1 child or 2 children in order to remove. Since it performs the same task as the insert method, it has O(log n) as its complexity. The removal task for 1 child or 2 children is less than lg n, so at the end the complexity is still O(log n).

1. **Contains method:**

Performing exactly the same task to insert and remove method, the contains method also has O(log n) for its runtime complexity. That is, each comparison gets rid of half of the nodes that need to compare. After finding the node, it merely returns true or false regards whether the node is found or not.

1. **Balanced Tree:**

**Insert, Remove and Contains method:**

Since the balanced tree is constructed to maintain O(log n) as the height of the tree, the best, average and the worst case for search, insertion and deletion are always O(log n). Starting from the root, the balanced tree performs comparison, as mentioned in binary tree, until the insertion place or the node that needs to be removed is found. After the node is inserted or deleted, the tree will rebalance using either color flip or rotation. The contains method does not need to perform rebalancing after the node found, but the search task is already O(log n).