**Questions:      Submit answers to these questions in addition to your source code:**

**1) Explain the algorithm used in each case of a, b, c above.**

**2) Explain the big-O running time in each case of a, b, c above.**

a) The dictionary words are loaded into the linked list by the "add" method. Every new word read from the dictionary has been inserted at the end of the linked list. All the inserts takes place in constant time.

Once the complete list is being inserted, the searching of the list is done by checking each element in the linked list from the start till either it finds a match or till the end of the list. The worst performance can be O(n). The linked list performs worse than the avl tree or the hash map I have implemented.

b) The dictionary words are loaded in to the avl tree using the "insert" method. The new word that is to be inserted is compared with the existing words till the correct position is found and inserts at the node based on its value. This occurs recursively until it finds the right position to insert. The expected performance of avl tree is O(N).

Whereas the searching with avl tree is done by checking all the nodes of the existing nodes until a match is found or else the end of the list is reached where both the children are null.

c) The dictionary words are loaded in to the avl tree using the "add" method. For Every word the hash value is calculated using the hash function (used the Horner’s method of calculation here) and mapped into the correct columns of the table. Whenever there is a collision multiplied by n times and next location value is calculated.

When searching for the word in the dictionary the hash value is calculated using the same hash function and the value at that location is checked if its null or not and if not its value is checked if it is same as the word we are checking for. If it keeps using the probing function that has been written until it finds the value or there is a null value.

2

1. In case of a linked list the value that has to be checked by traversing the node in the sequence one by one and so it can be worst case equivalent to O (N).
2. In case of the avl tree, as it is a balanced tree, the worst case search needs to traverse the depth of O(logN) as not all nodes are compared with each other. There should be very good performance as compared to the linkedlist.
3. In case of the hashtable each search can be located using the hash function and ideally if there is a collision it has to traverse through the constant number of items and hence on average access times should be O(1).