COMP2007 Designs of Algorithms, Semester 2 2016
Adrian Hendrawan Putra
ID: 743206
Assignment 2 Report

Q1.) insertion: O(1), search: O(*n*).
Assuming that hash collisions is resolved with a linked list, an item will be inserted to the head of the linked list, which is doable in O(1) time complexity.
Searching for an item in a linked list requires traversing through the list of n items, and since all item is hashed into the same bucket, it would take O(*n*) time complexity to search for an item.

Q2.) insertion: O(1), search: O(*n*/*size*).
Insertion is still O(1) due to insertion to the head of a linked list.
If the hash function distributes *n* items evenly over *size* buckets, each bucket would have at least *n*/*size* items. This is due to the "pigeonhole-principle", meaning that if there are more items than there are buckets, some of the buckets would have to hold more than 1 items. This means that searching for an item in a bucket will take at most O(*n*/*size*) time complexity.

Q3.) insertion: O(1), search: O(1).
Insertion is still O(1) due to insertion to the head of a linked list.
If the hash function never hashes two different inputs to the same bucket, each bucket will only hold one items. Therefore search is doable in O(1) time complexity.

Q4.) The reason that this is a bad hash function is that for any strings that starts with the same letter will get hashed into the same bucket. If the probability of the first letter occurring is not evenly distributed (i.e. the character 'A' is more likely than the other characters), some buckets will receive more items than the others.

Q5.) insertion: O(1), search: O(*n*/*size*).
Insertion is still O(1) due to insertion to the head of a linked list.
Each character has 256 (2^8) possible values, and assuming that the random integer is distributed evenly, this means that for a string with length *L* will get distributed more or less evenly through the bucket. Therefore each bucket will contain at least O(*n*/*size*) items.

Q6.) *collide_dumb()*:
This function works by generating strings and testing it against the hashing function to see whether the string is hashed to zero. The length of the string is then increased by one, and the character for each position in the string is set starting from one (This is to comply with the requirement of no string padded with zero).

Q7.)unfinished.