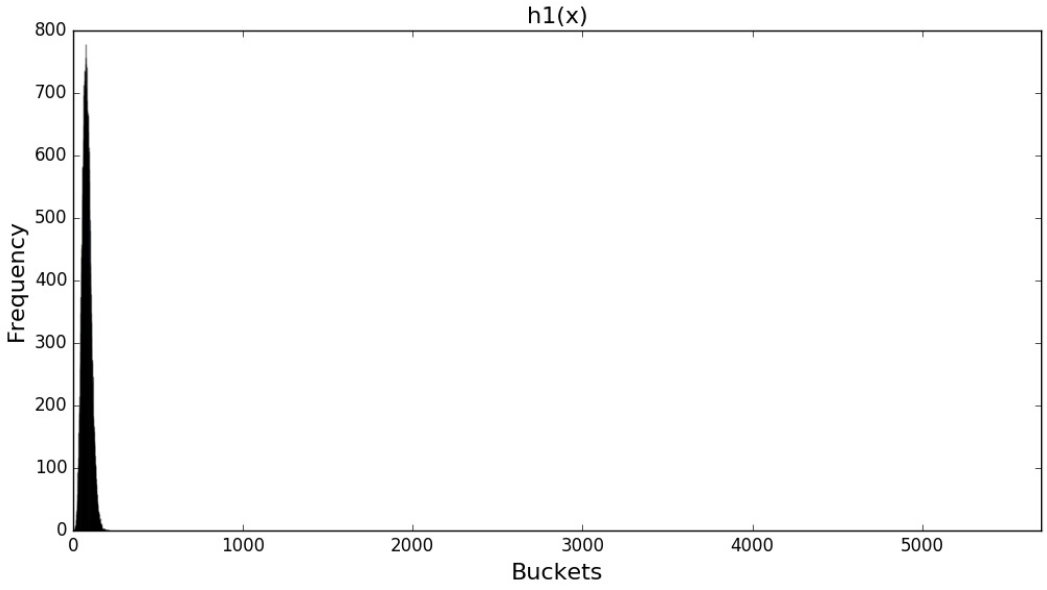
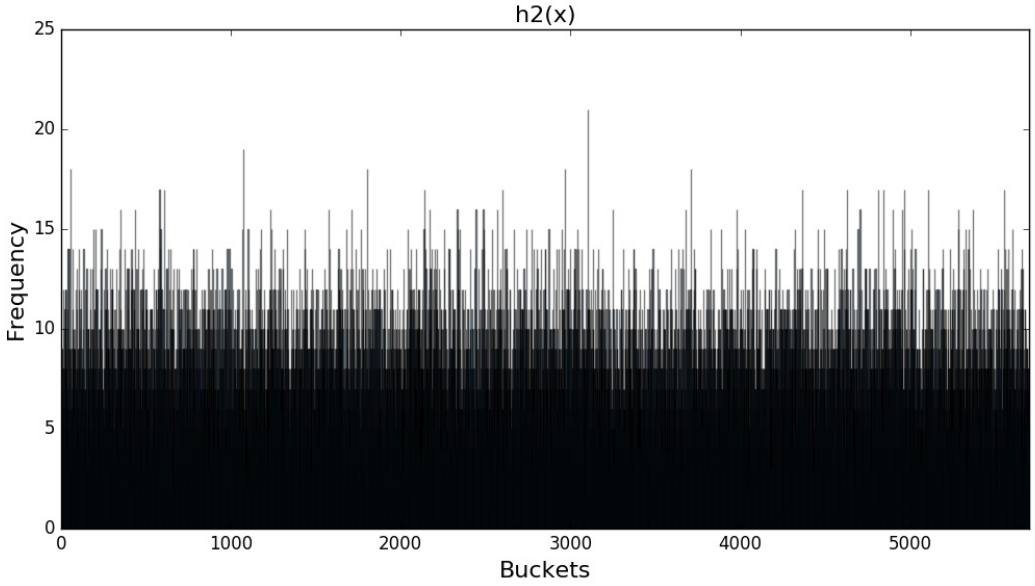
1.a. $\Theta(m \log n)$. This algorithm would have to traverse through T_1 in its entirety, each time comparing its current key to T_2's keys. The runtime on searching for T_1's current key in T_2 is $\Theta(\log n)$ because T_1 and T_2 are binary trees, so if the current key is less that T_2's first node half of T_2's keys can be disregarded for the moment and the search continues like so.

b. $\Theta(n \log m)$. This algorithm first traverses through T_1 filling in H_1 which operates with a running time of $\Theta(m)$. Then the algorithm traverses every element of T_2 with a running time of $\Theta(n)$ and checks if each node is in H_1 which operates with running time $\Theta(\log m)$. Each search through H_1 takes the current key from T_2 and maps it with the hash function for H_1. Then it checks that specific bucket to see if any keys have already been mapped there. So at each node traversed in T_1 a search through H_1 occurs giving a total running time of $\Theta(n \log m)$.

2.b. The histogram for the first hash function shows that all of the keys were hashed into the first 250 or so buckets. A few of those buckets contain above 700 keys which does not make for the best hash function. The remaining 5400 buckets or so seem to be empty. Compared to the behavior of h(x), h1(x) does not seem to perform close to a uniform hashing function like h(x) does.

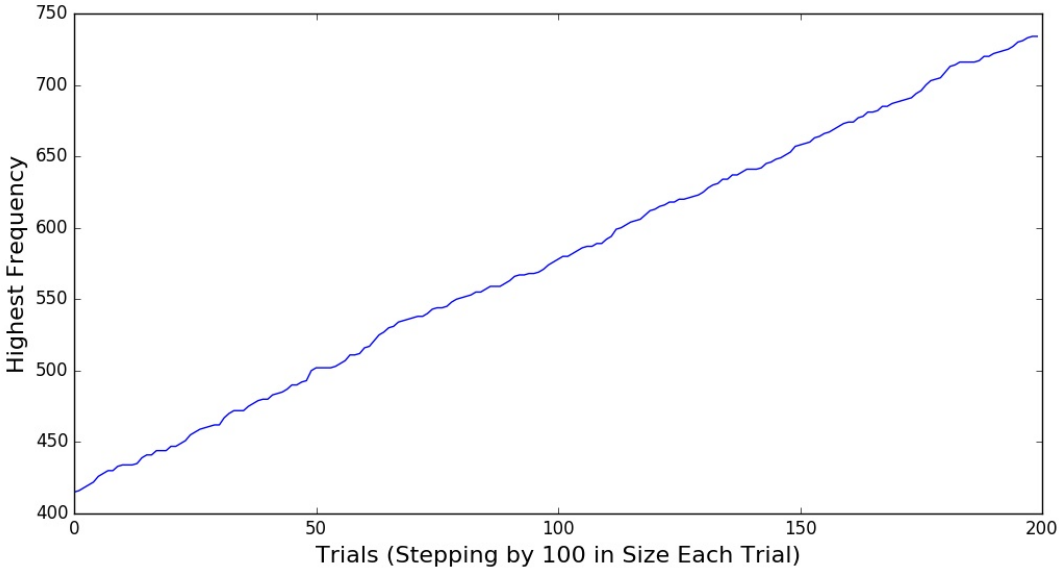


The histogram for the second hash function shows a uniform distribution of the keys between buckets 0 and 5701. The average amount of keys in each bucket seems to be a little over 10. This shows a huge improvement compared to the first hash function. This hash function operates in a similar manner that h(x) executes because h2(x) appears to be close to a uniform hashing function as h(x) is intended to be.

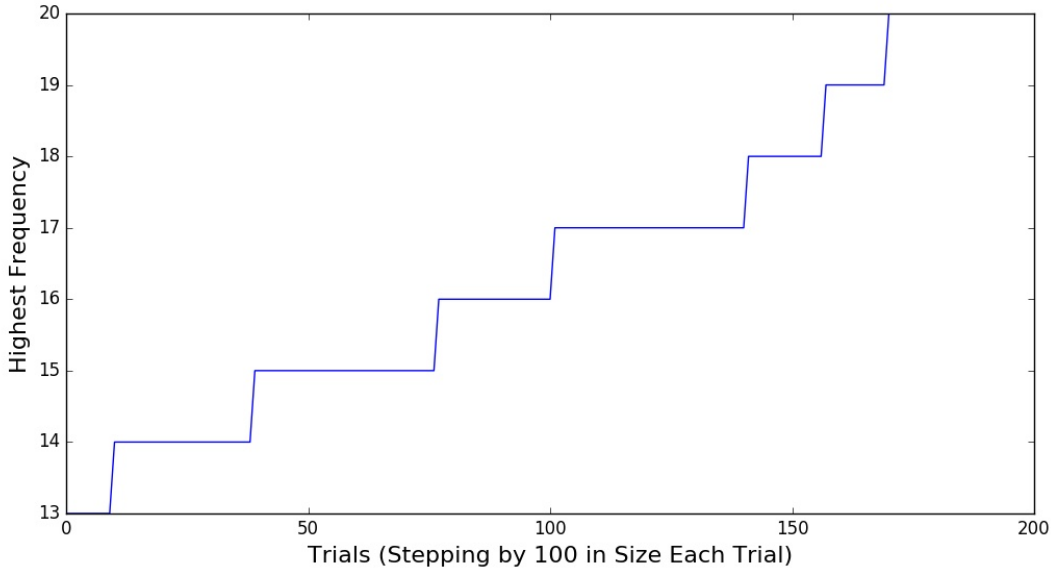


c. Reasons why h1(x) is a bad function relative to the ideal behavior of uniform hashing:
-All of the keys were hashed into roughly 5% of the total number of buckets, so about 95% of the buckets are empty.
-In roughly half of the buckets that actually contain values, the frequency is over 300. A few buckets even have a frequency of over 700.
-Searching for a value within a hash table like this would have worst case and best case methods that differ severely where a uniform hashing function's worst case and best case searching are fairly close.
-Deleting a value has the same issue as searching with h1(x) because it still has to use search to find that value.

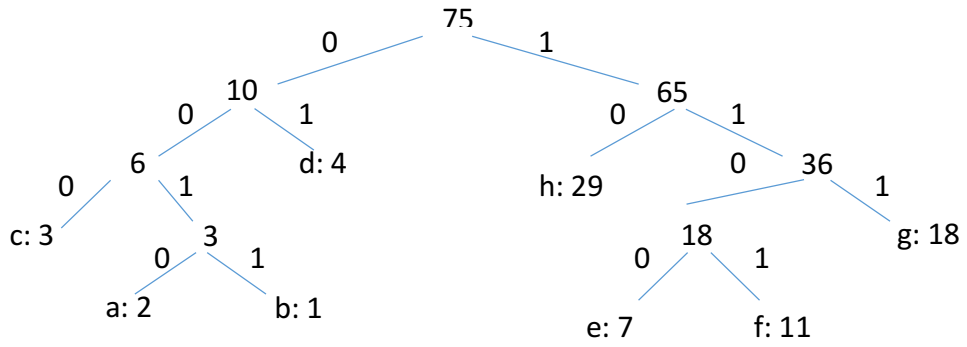
d. The first plot shows a linear relationship between longest chain (highest frequency) and increasing number of strings (each trial). As we step through and increment the number of strings used in each trial we see that the longest chain grows steadily with the increase of strings.



This plot shows the longest chain increasing as the number of strings in each trial increases. The difference here is that the longest chain increases by one, after several trials have ran. It does not seem to increase each trial. It only changes at certain lengths of the input strings. In all it does show an increase in longest chain as the size of n strings increases with each trial.



3.a.



a = 0010 b = 0011 c = 000 d = 01 e = 1100 f = 1101 g = 111 h = 10

b. When the frequencies are the first n Lucas numbers, the depth of the encoding tree will be the largest number of bits that one frequency will be encoded as. Every other frequency will be encoded with depth-1 or less bits. In the case above the depth of the tree was 4 and a, b, e, and f were encoded with 4 bits each. Every other frequency was encoded with 3 or less bits.