Ringo Yen
Compsci 201

Markov Assignment Analysis

**Brute Markov Hypothesis (BH):**

After running the code, and looking at the Benchmark output, it is clear that the hypothesis is approximately supported by the data. It does seem as though the time to generate the random text is proportional to NT, and this is independent of the k value. This can be demonstrated with the following plots:

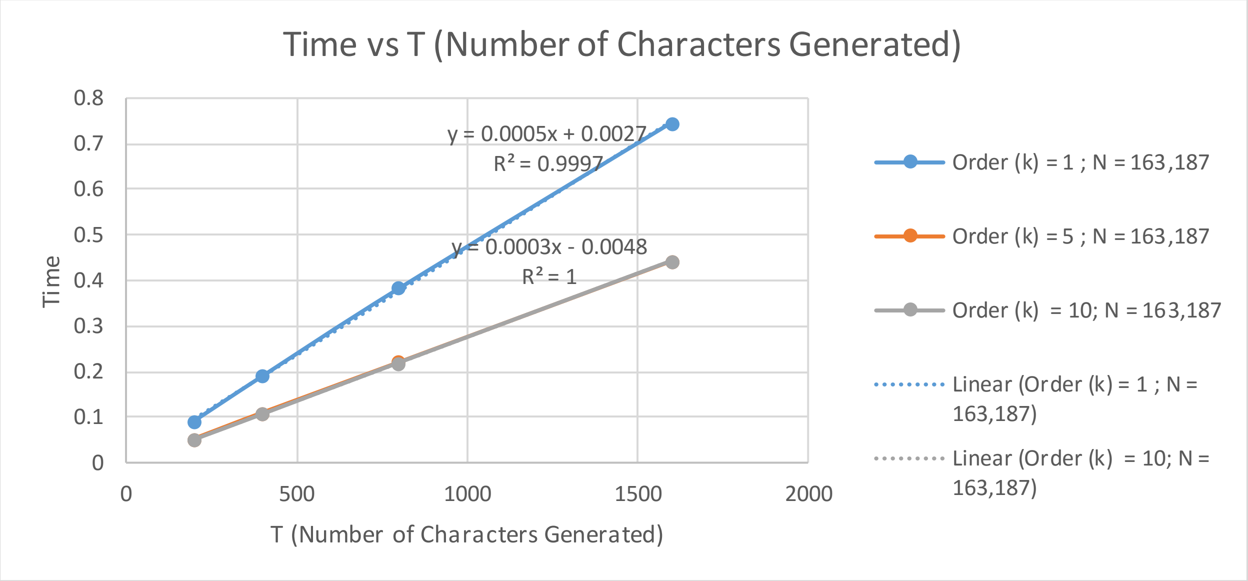

Figure #1: Plot of Time vs T (Number of Characters Generated – Random Text).

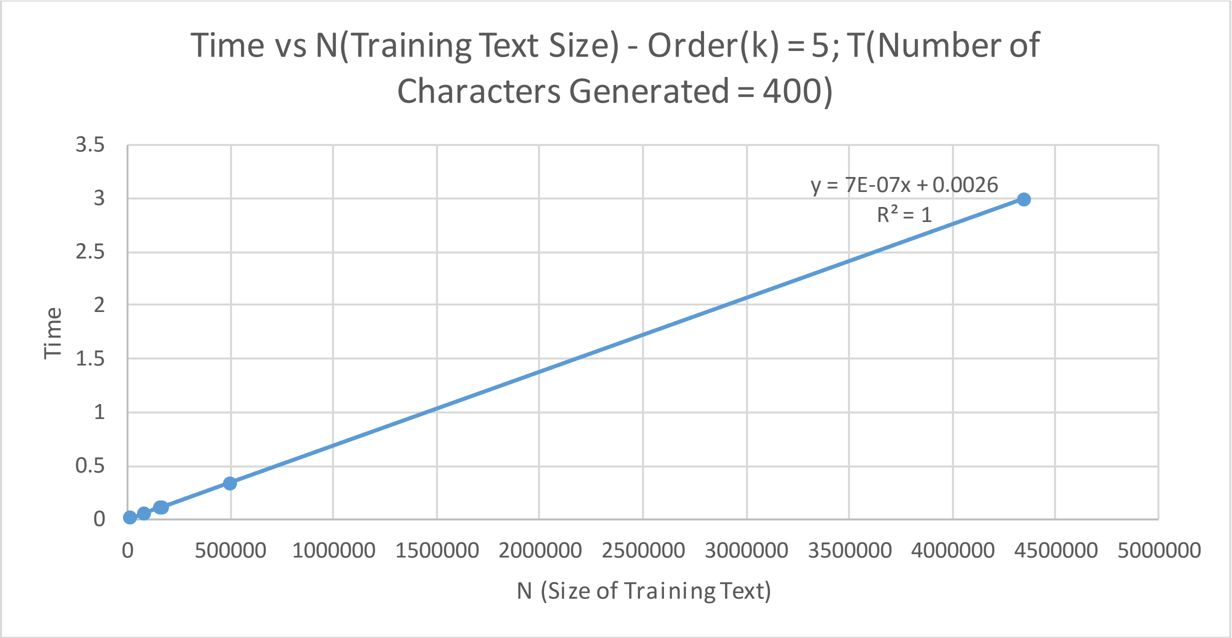

Figure #2: Plot of Time vs N (Size of Training Text).

As seen in the above plots, time seems to increase linearly as T (random text – number of characters generated) (figure 1) increases, and as N (size of training text) (figure 2) increases. In figure 1, the N value (training text) is fixed at 163,187 for all cases.
In figure 2, the order is fixed at 5, and the T value is fixed at 400 characters. Figure 2 was generated by using the different training text data provided at the end of the program (for Hawthorne, kjv10, etc).
The excellent linear fits in both plots also provides additional support that the time is proportional to the T and N values. Thus, it seems that time is proportional to NT.

In addition to this, it appears that the other half of the BH is correct – the time is independent of k. This is proved in the graph below:

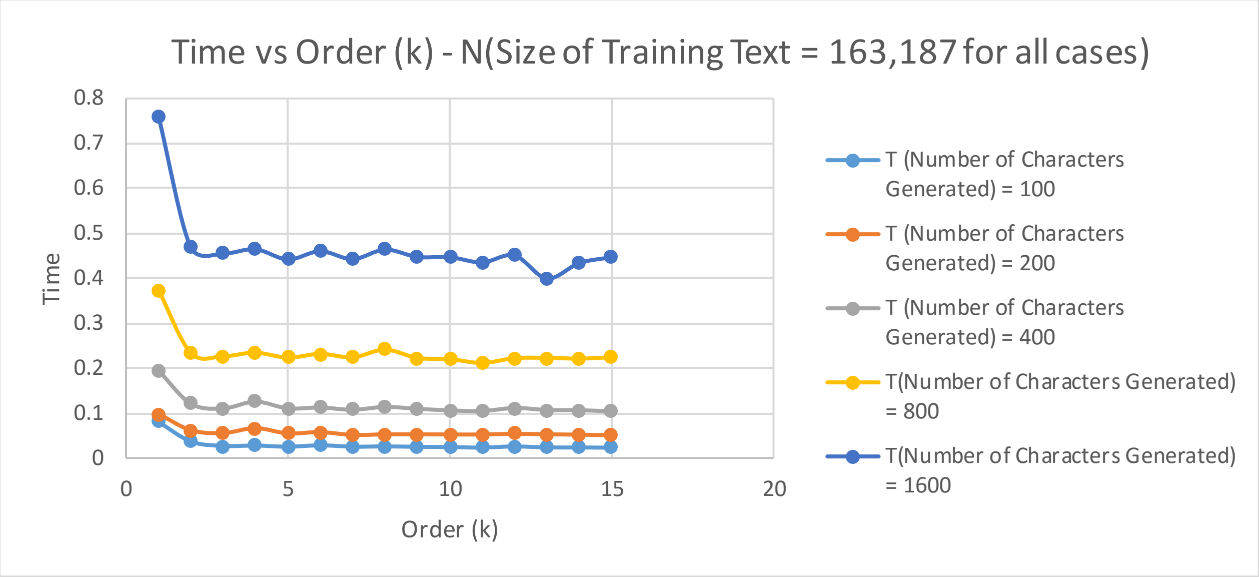

Figure #3: Plot of time vs order (k)

As you can see in the plot above, as order (k) increases, the time pretty much stays constant – there are some small perturbations as k increases, but time generally stays constant, and the curves are approximately flat. There does seem to be a drop in time from order (k) = 1 to order (k) = 2, but this may be due to the fact that the k-gram size is too small at k = 1. Generally, as k increases, the curves of time vs k seem to remain stable.

**Efficient Hypothesis (EH):**

After running the benchmark program, it appears that the hypothesis is actually incorrect, based on the data and results. It seems that the process is independent of T and actually dependent on the order(k) size. This is opposite of the hypothesis, which states that the Efficient Markov Process is dependent on the size of T and independent of order (k). The plots below show this:

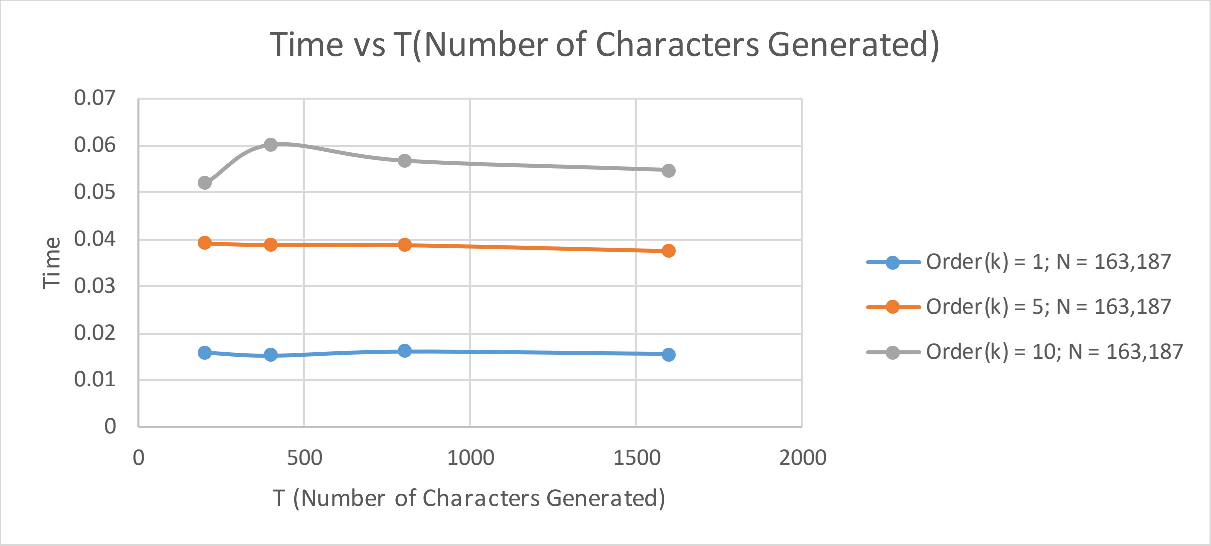

Figure #4: Time vs T (Number of Characters Generated – Random Text)

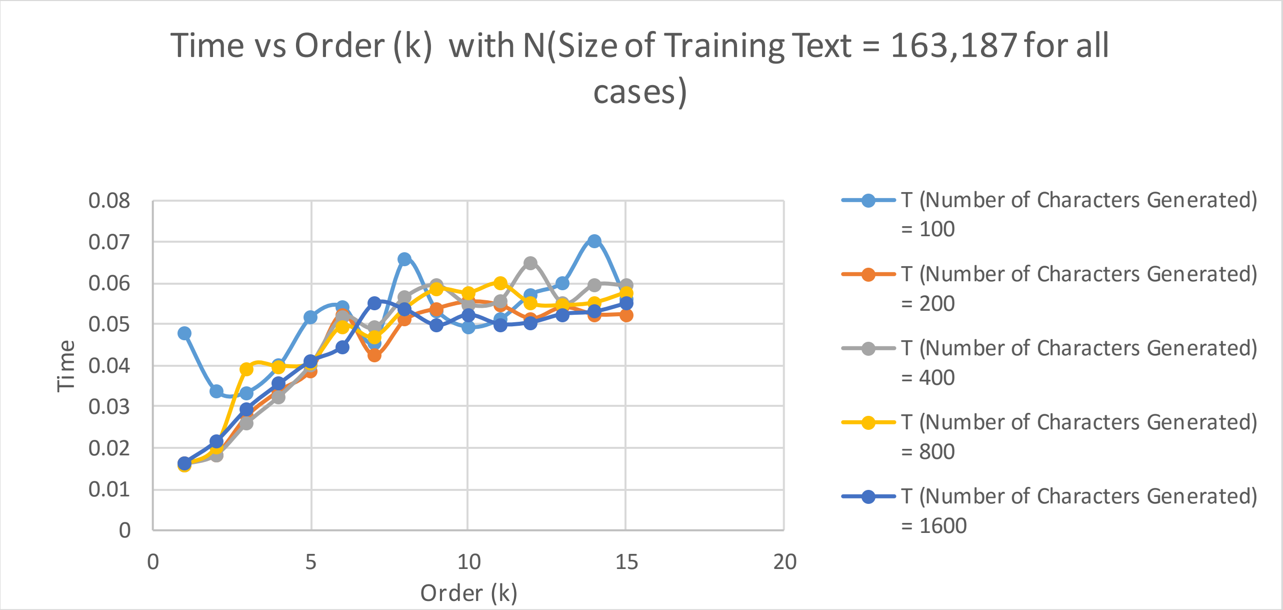

Figure #5: Time vs order (k)

As we can see with the plots above, the time seems to be unchanged as T increases, yet time seems to change as the order rises.

There are variety of reasons for this. It could be that the larger order k could lead to longer time for the hashcodes to be generated. It could be that a good majority of the time could be taken up by having the hashmap made, and thus once it is made, the time is no longer affected by the number of random text (T) that needs to be made, since the map is generated and it is easier to find the characters to make the random text.

**Map Time Hypothesis (MT):**

After looking at the data, it appears that the hypothesis is mostly true for the Hashmap structure (that the process is dependent on O(U), proportional to U), yet the hypothesis is not true for the TreeMap (that the process is proportional to UlogU). The following plots (figures 6 & 7) show that the hypothesis is mostly but not completely true for the Hashmap:

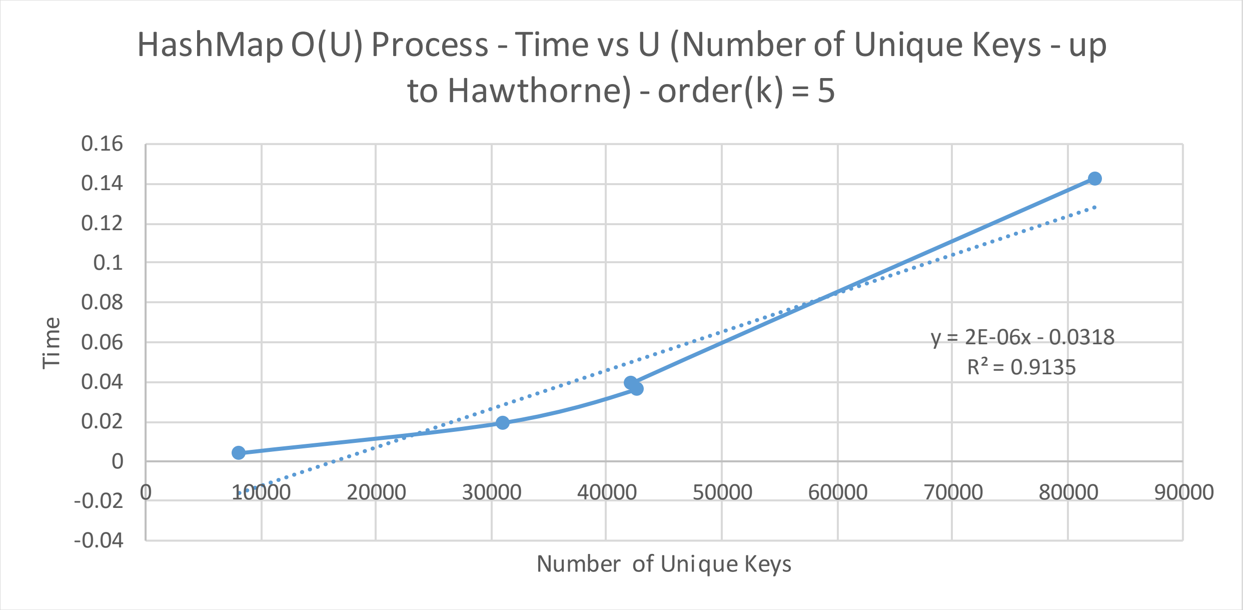

Figure 6: Plot of Time vs Unique Keys – not including the kjv10 text, which has the most keys.

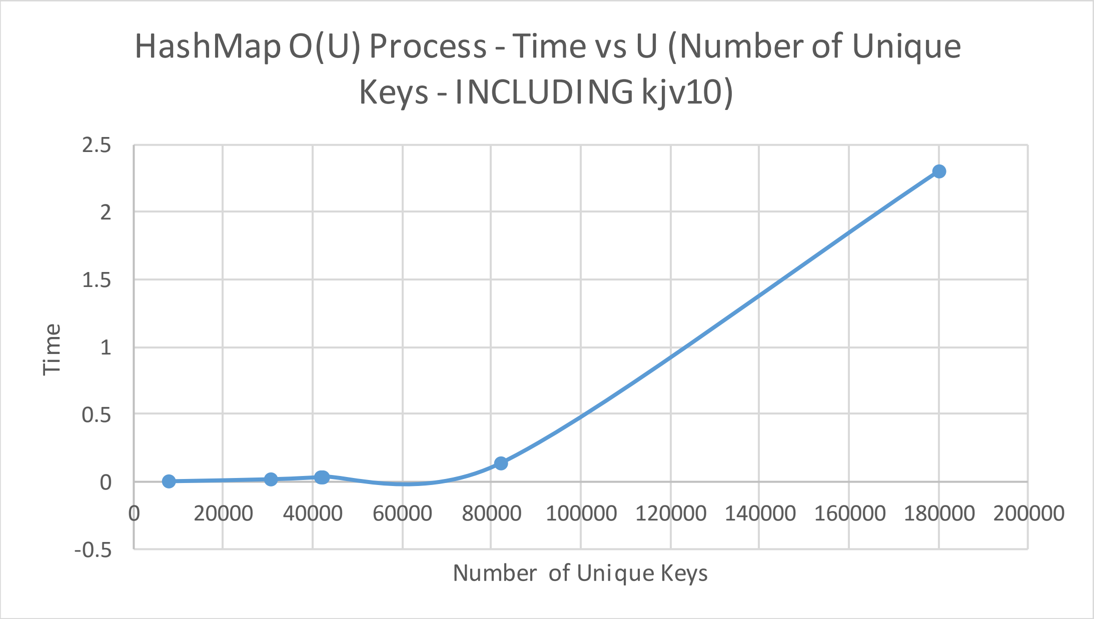

Figure #7: Plot of Time vs Unique Keys - including the kjv10 text.

From figure #6, we can see that as the unique keys increases the time seems to increase linearly – the linear fit and R squared value justifies this. However, in figure 7, this linear fit is gone, since the large number of unique keys from kjv10 seems to affect the fit. This could mean that the hypothesis for the HashMap is correct up to a certain number of unique keys, and past a threshold number of Unique Key values, the hypothesis no longer holds true for HashMaps.

The hypothesis for the TreeMap does not seem to be correct, as illustrated by the plot below:

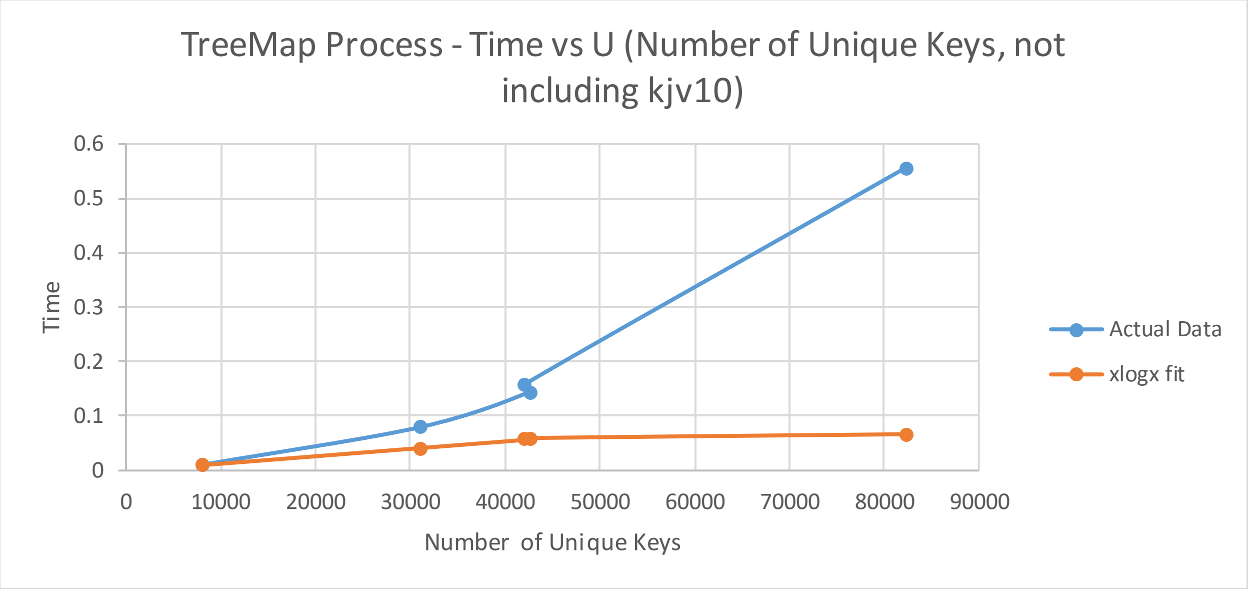

Figure #8: Actual Data vs xlogx fit (ULogU) fit.

As we can see, the data does not seem to match the fit of the xlogx fit. This could be because it takes longer to generate TreeMaps, and shorter time to generate HashMaps, because the keys are ordered in TreeMaps and not in HashMaps. Thus, the setTraining methods with TreeMaps could be taking more time than the setTraining methods with HashMaps.