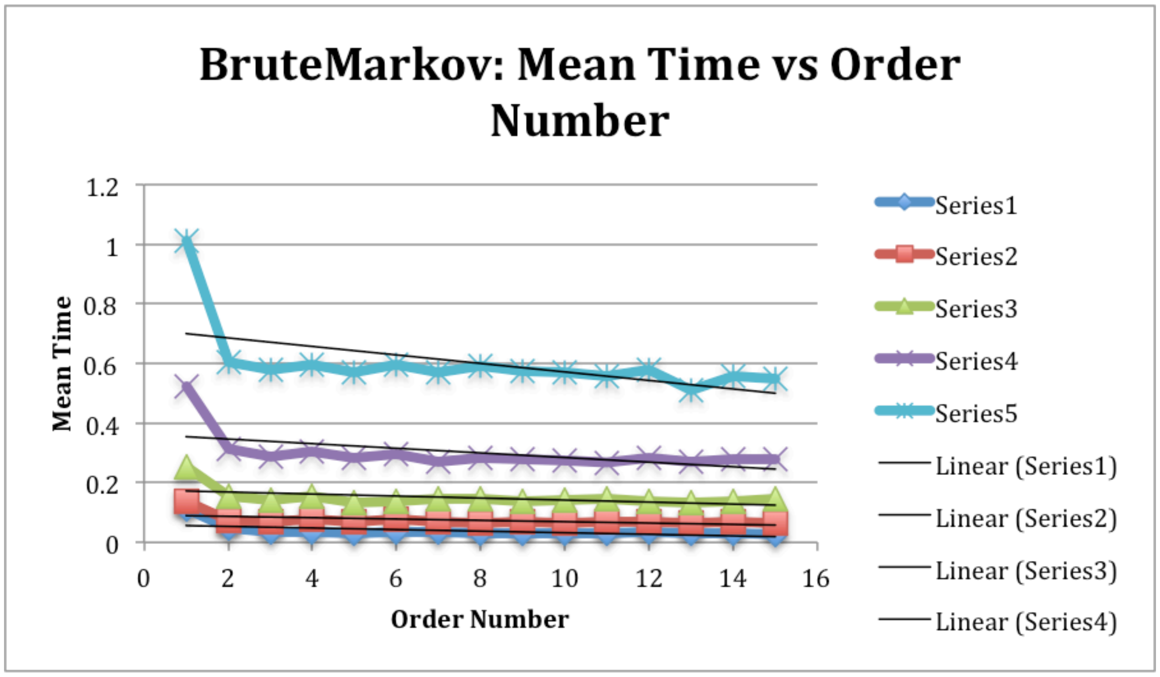
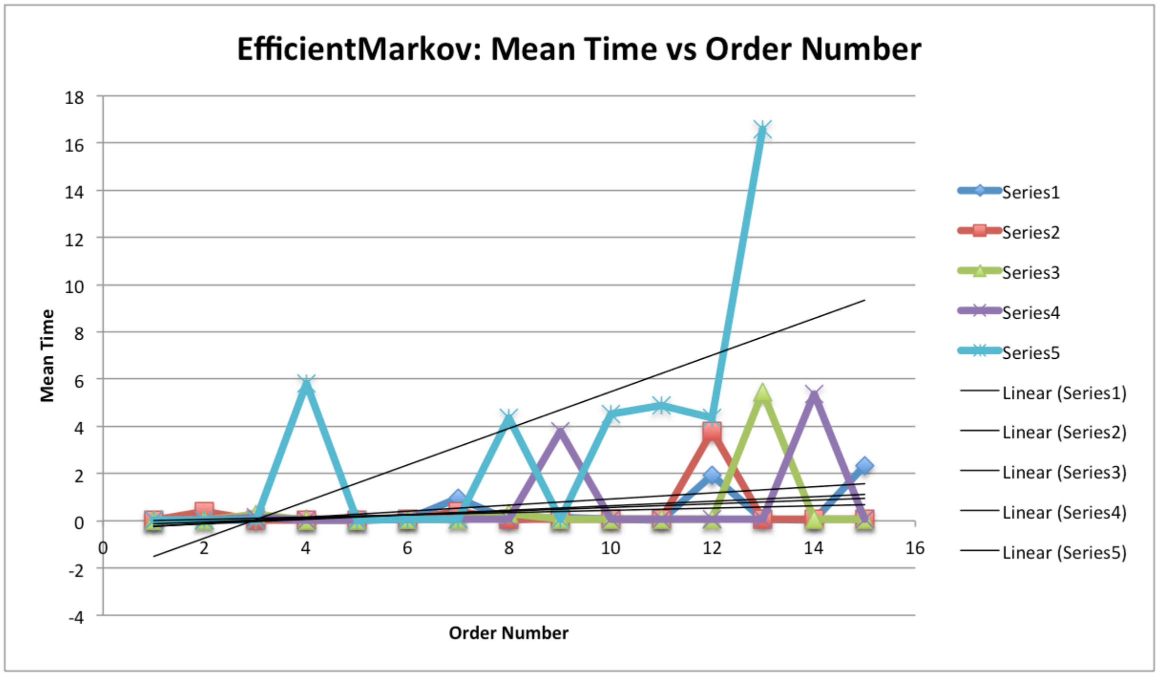
Jeffrey Blair
Markov Analysis



<u>BruteMarkov Hypothesis:</u> generating *T* random characters when trained on a text of size N is *O(NT)* that is takes time proportional to *NT*, and this is independent of *k,* where k is the order of the markov process.

In the above graph, series 1 corresponds to 100 random characters, series 2 to 200, series 3 to 400, series 4 to 800, and series 5 to 1600. (Do not know how to change the line names in Excel)

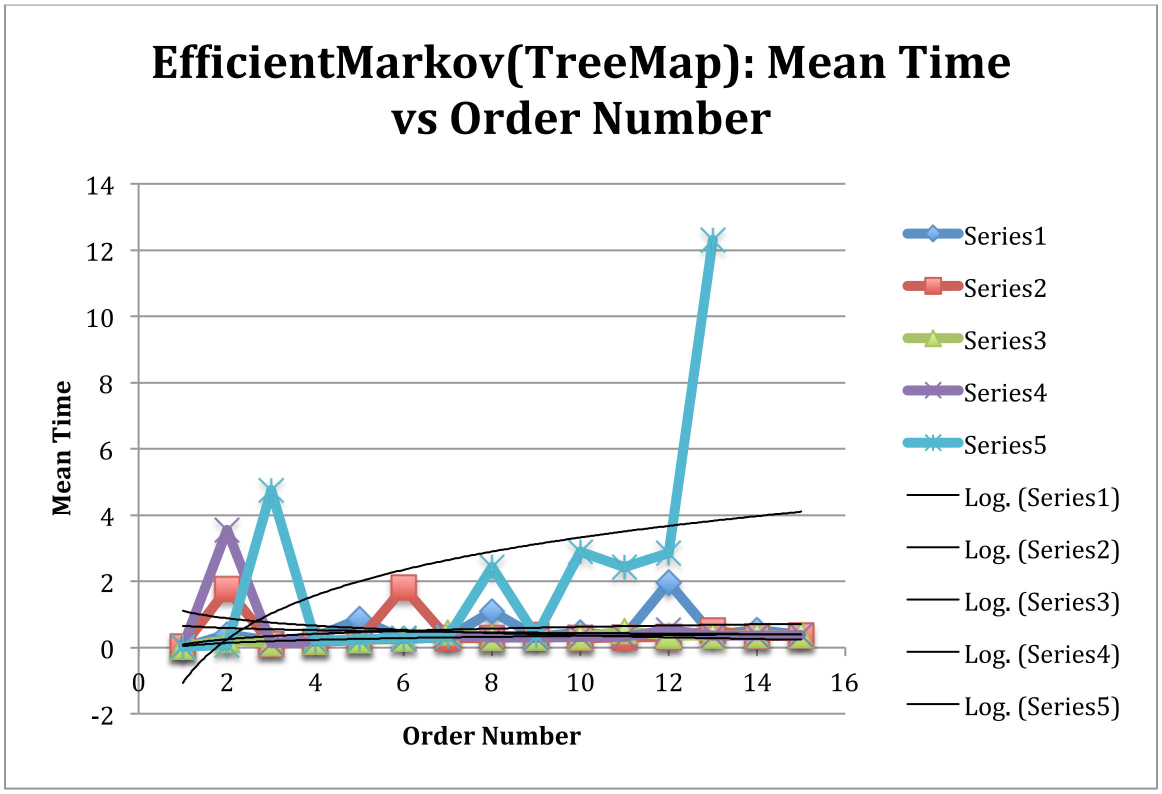
The above hypothesis is proven by the above data, run with BruteMarkov. The five lines on the above graph each correspond to an increasing T value, however the overall trend of each of these is linear, in that the time is independent of the order (N). When order increases, the overall trend at varying T values is a constant time. An exception to this is the change from order = 1 to order = 2. For each of the series in this graph, the graph time decreases going to order = 2, but then stays linear throughout. The value of T does effect the time, in that an increase in T will increase the overall time, proving that time only depends on NT, not order (k). Analytically, this hypothesis is true. In order to inter through every possible phrase and its following letter, the program will loop through N times and then an inner loop of T times. This will create a time of O(NT), as hypothesized.

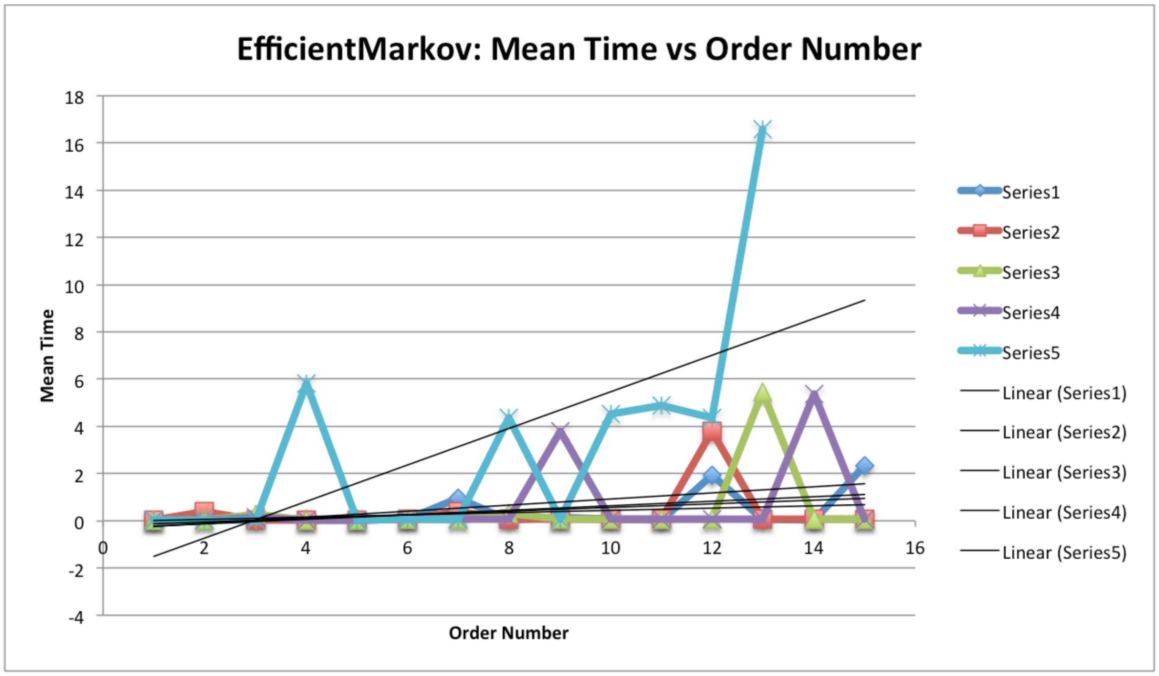


<u>Efficient Hypothesis</u>: generating *T* random characters when trained on a text of size *N* (ignoring training time) is *O(T)* that is takes time proportional to *T*, and this is independent of *k*, where *k* is the order of the markov process. If we don't ignore training time, the time is *O(N + T)* since training time will be proportional to *N*, the size of the training text.

In the above graph, series 1 corresponds to 100 random characters, series 2 to 200, series 3 to 400, series 4 to 800, and series 5 to 1600. (Do not know how to change the line names in Excel)

This hypothesis is proven by this program, as seen by the above graph. With the exception of 1600 characters, the linear trend lines for each of the varying character amounts has nearly a horizontal slope. This implies that as order increases fro each series, the time does not increase, as hypothesized. There may be error in this graph, since my computer was unable to process the data to run 15 tests properly without risking over heating the machine. Following the TA's direction, this graph only represents 1 test for each order and character combination (it was too risky using anything higher). With only 1 test, it is hard to statistically prove the hypothesis, however the trend can still be seen. Analytically this hypothesis is true, since the use of a HashMap in the Efficient Markov code will cut dependency on looping through N times again, therefore making the time O(T) instead of O(NT), proving the hypothesis.


EfficientMarkov(TreeMap): Mean Time vs Order Number
Mean Time
Order Number
Series1
Series2
Series3
Series4
Series5
Log. (Series1)
Log. (Series2)
Log. (Series3)
Log. (Series4)
Log. (Series5)


EfficientMarkov: Mean Time vs Order Number
Mean Time
Order Number
Series1
Series2
Series3
Series4
Series5
Linear (Series1)
Linear (Series2)
Linear (Series3)
Linear (Series4)
Linear (Series5)

Map Hypothesis: inserting *U* keys in a HashMap is *O(U)* and inserting *U* keys into a Treemap is *O(U log U)*.

The top graph above shows EfficientMarkov using a TreeMap. The graph contains logarithmic trend lines.
The bottom graph shows the original EfficientMarkov using a HashMap. The graph contains linear trend lines.

The linear trend lines of the HashMap seen in the bottom graph prove that inserting U keys in a HashMap will be O(U). With the exception of seriers 5, the overall trend for each series is linear and independent of the k value, therefor, an amount of keys (U) will follow a linear pattern in a HashMap. Analytically this hypothesis would be true, since the HashMap will only have to go through the U number of keys in order to find every following letter in the values.

The top graph does not prove the second part of the hypothesis, in that U keys will have O(U log U) in a TreeMap. The logarithmic trend lines poorly represent a logarithmic pattern, implying that the hypothesis does not hold under this program. Analytically, this hypothesis is true, since as the program runs through the keys of the TreeMap to sort them (creating the time U). It then has to run through the sorted keys in binary tree structure in order to find the proper key, adding the time logU. These two times are multiplied together in order to run through all U keys in the TreeMap, therefore creating the time of O(U logU).

These two conclusions may be statistically insignificant, since my computer was unable to process the data to run 15 tests properly without risking over heating. Following the TA's direction, this graph only represents 1 test for each order and character combination (it was too risky using anything higher), instead of an average. With only 1 test, it is hard to statistically prove the hypothesis, however the trends can still be seen.