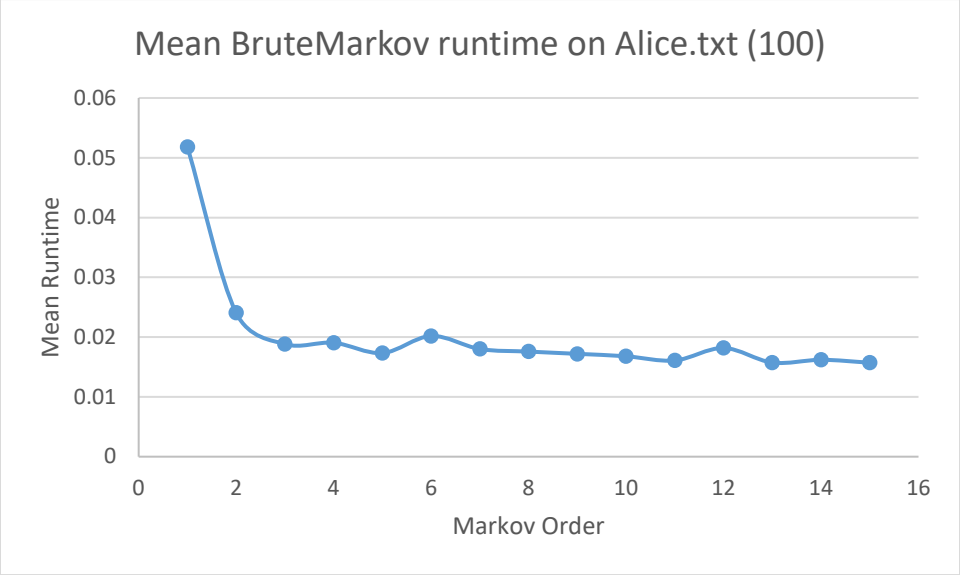
Harry Wang

CS201

Markov Analysis

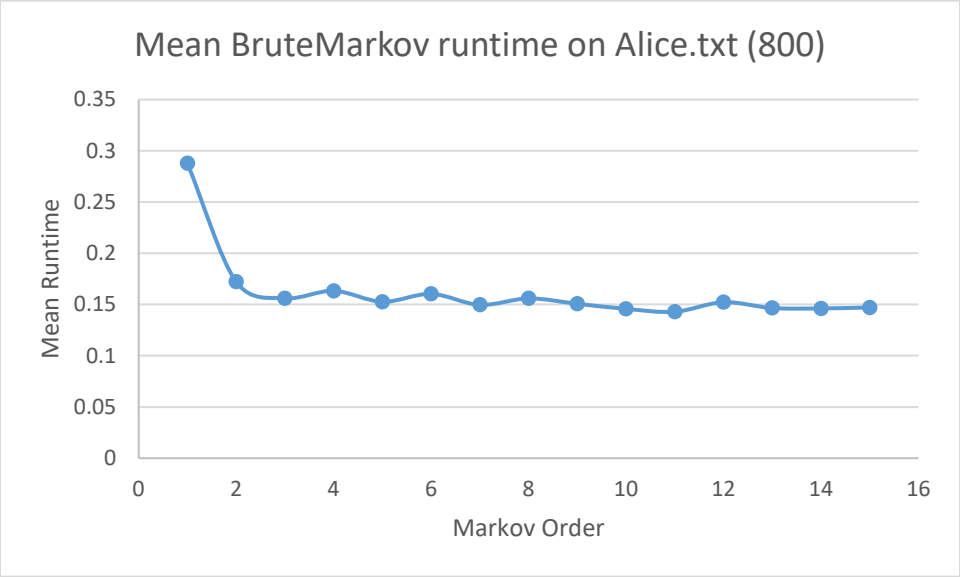
BruteMarkov Hypotheses (BH): generating ***T*** random characters when trained on a text of size ***N*** is ***O(NT)*** that it takes time proportional to ***NT***, and this is independent of ***k***, where ***k*** is the order of the Markov process.

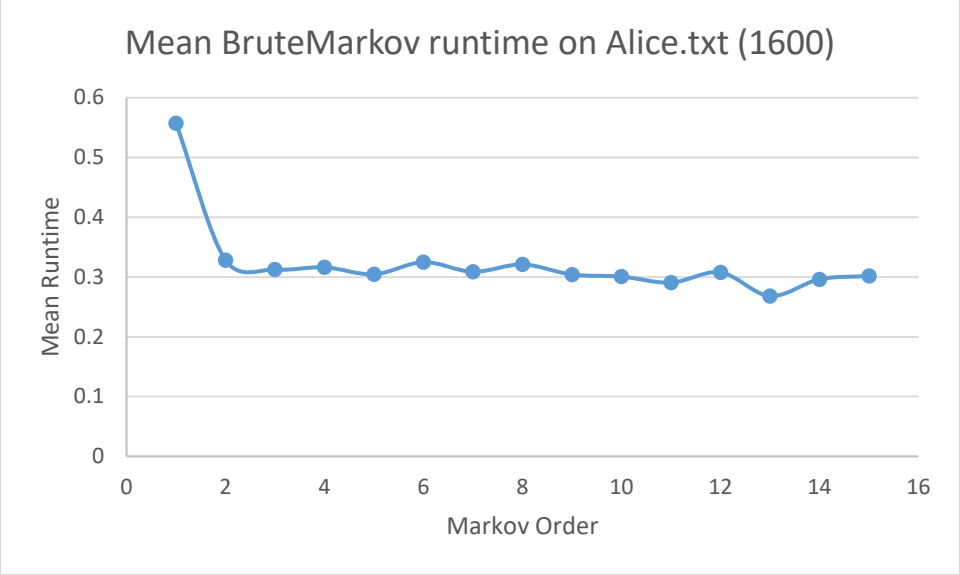
Using the Benchmark file to test BruteMarkov on the *Alice.txt* gives results that almost satisfies hypothesis ***NT***. Apart from the first two orders (1, 2), the mean runtime is shown to be linear and independent of ***k*** as ***k*** gets larger. Plotting the results of the mean BruteMarkov runtime on *Alice.txt* where 100 characters are generated shows:



Taking the linear correlation coefficient of the data, yields -0.58545. This is heavily influenced by the two outliers orders (1, 2), and if the two outliers are removed from the equation, the correlation coefficient yields -0.77792, which is exceedingly strong negative linear relationship.

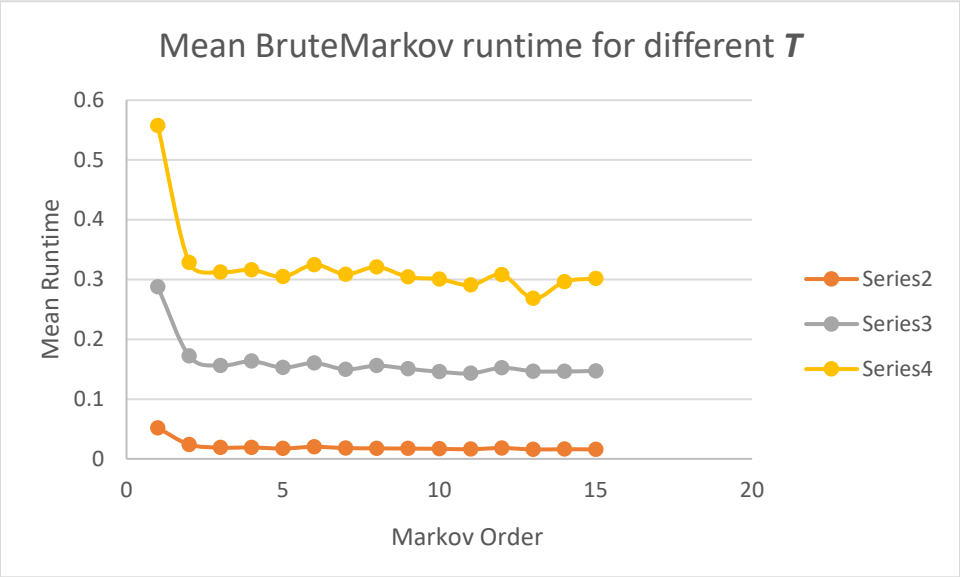
Plotting the mean runtime results with larger random values (800, 1600) gives:





These all show very similar results. All three benchmark tests have the first two points, order 1 and 2, as a significant outlier, but when the outliers are taken out, both show somewhat linear runtimes independent of ***k***. Using the linear correlation coefficient, the results for random number 800 and 1600 are -0.57836 and -0.55988 respectively. Also, when the first two orders are taken out (1, 2), the correlation coefficient yields -0.74793 and -0.61681 respectively.

As number of generated texts increases ***T***, the runtime also increases. This is significantly emphasized when all three graphs are combined below:

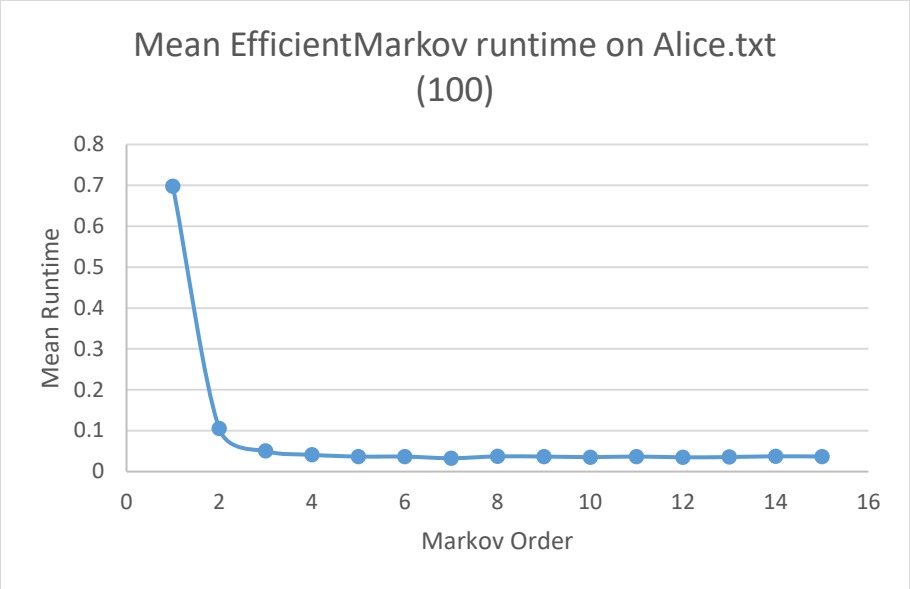


Thus, the empirical evidence present supports the hypothesis stated on BruteMarkov. Since all the data sets show a similar linear correlation coefficient (both with and without the outliers), and show that as the number of generated texts increases, ***T***, runtime also increases, the above BH is true (Series 2 is where ***T*** is 100, Series 3 is where ***T*** is equal to 800, and Series 4 is where ***T*** is equal to 1600). Taking the median point in each of the three Series and solving for the linear correlation coefficient between varying ***T*** and runtime, gives 0.99987, an extremely strong linear correlation that supports the hypothesis.

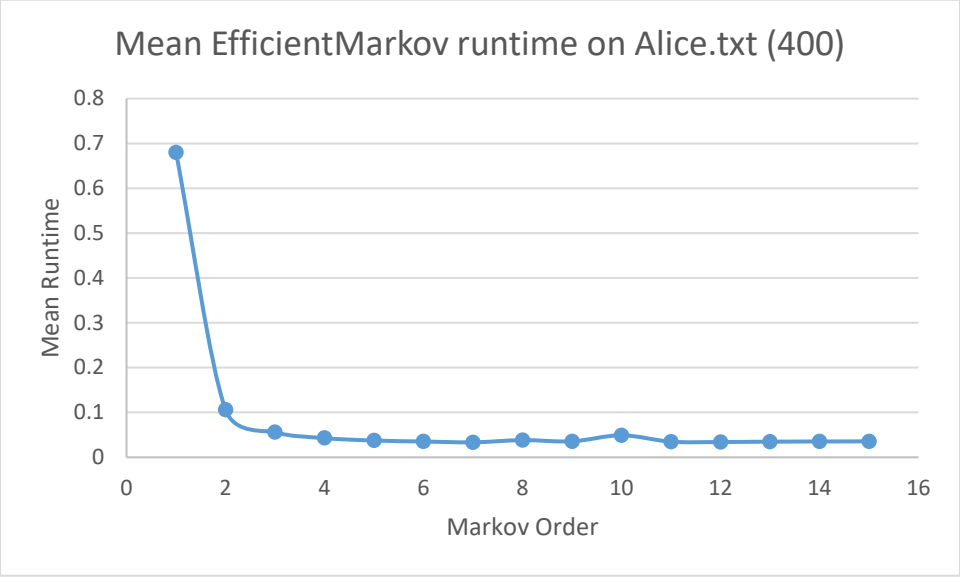
EfficientMarkov Hypotheses(EH): generating ***T*** random characters when trained on a text of size ***N*** (ignoring training time) is ***O(T)*** that is takes time proportional to ***T***, and this is independent of ***k***, where ***k*** is the order of the Markov process. If we don't ignore training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

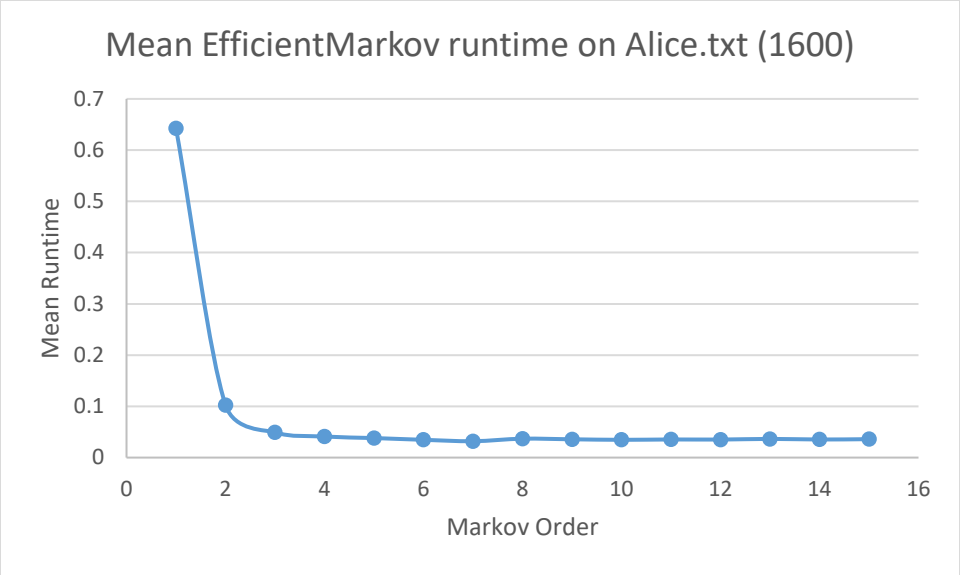
To minimize runtime in EfficientMarkov, I used HashMaps. I also used a StringBuilder to create the key of the HashMap and one ArrayList to log the character that followed. I would then check to see if my HashMap contained the key through the myMap.containsKey() method, and if it existed, I would add the ArrayList stored into my current ArrayList (this way I wouldn't have to create a new ArrayList every time I meet a duplicate).

If we don't ignore training time, the runtime should is ***O(N+T)***. Using the benchmark to test EfficientMarkov on *Alice.txt*, yields results that are very similar throughout. Plotting the mean runtime where 100 random values are generated is shown as:

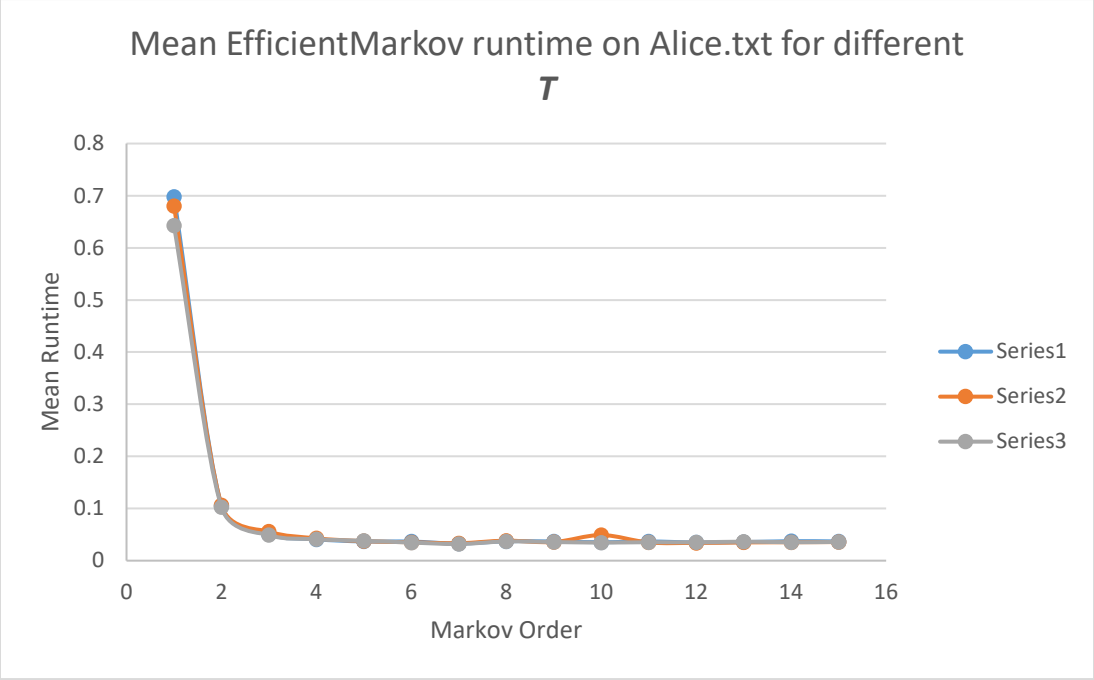


Where the correlation is -0.48147. However, if we take out the big outliers as shown in the graph, orders 1 and 2, the correlation coefficient yields -0.51312. The hypothesis states that running time should be roughly the same as it is proportional to the size of the training text and number of random characters. Thus, plotting the mean runtime for random characters generated at 400 and 1600, shows:



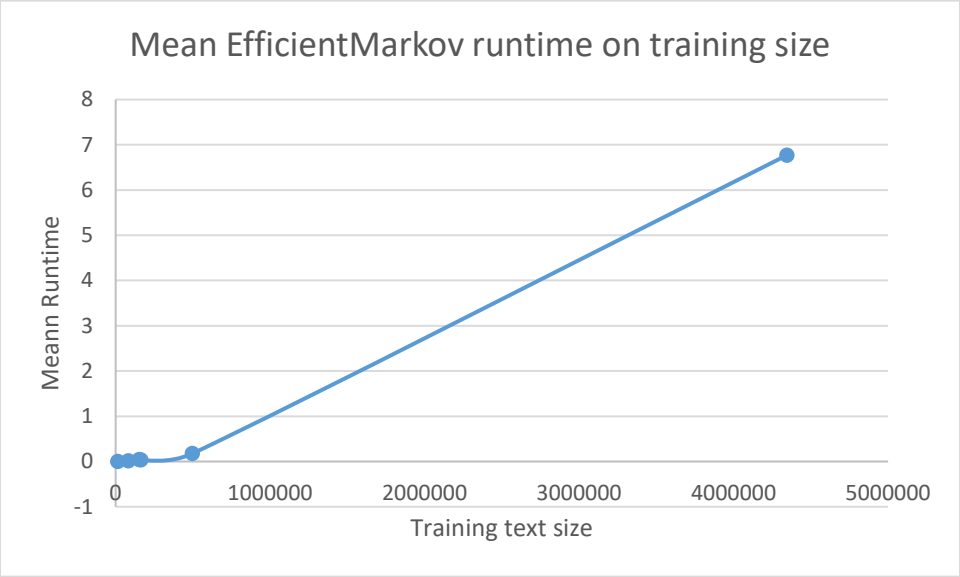


Both graphs, and also the first, show that the runtimes to be very similar. The hypothesis above (EH) states that if training time is taken into consideration, runtime will be proportional to ***N***, the size of the training text. This is shown to be true through the above graphs. If plot together, they give very similar and almost identical values.



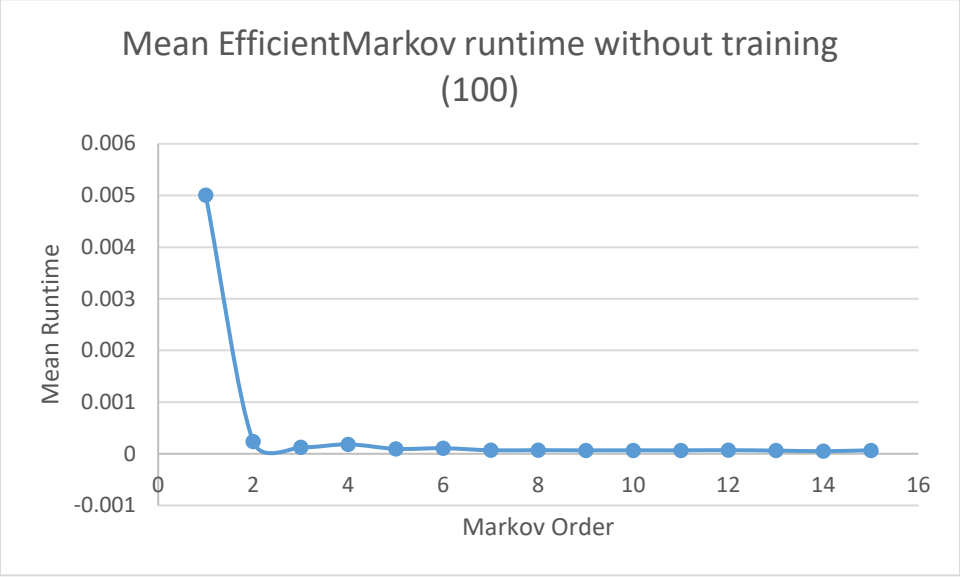
Thus, this proves that the hypothesis above (EH) for when training time is taken into consideration, is true. The runtimes above are shown to be independent of ***k*** (especially when the outliers are taken out of consideration). The graph below shows the different runtimes for different training text sizes. Using the median points from all 3 Series and solving for the linear correlation coefficient between varying ***T*** and mean runtime gives -0.23239. A very weak correlation which supports the hypothesis by showing that the runtime is in fact independent of ***T***. This is because in the code, whenever getFollows is called,

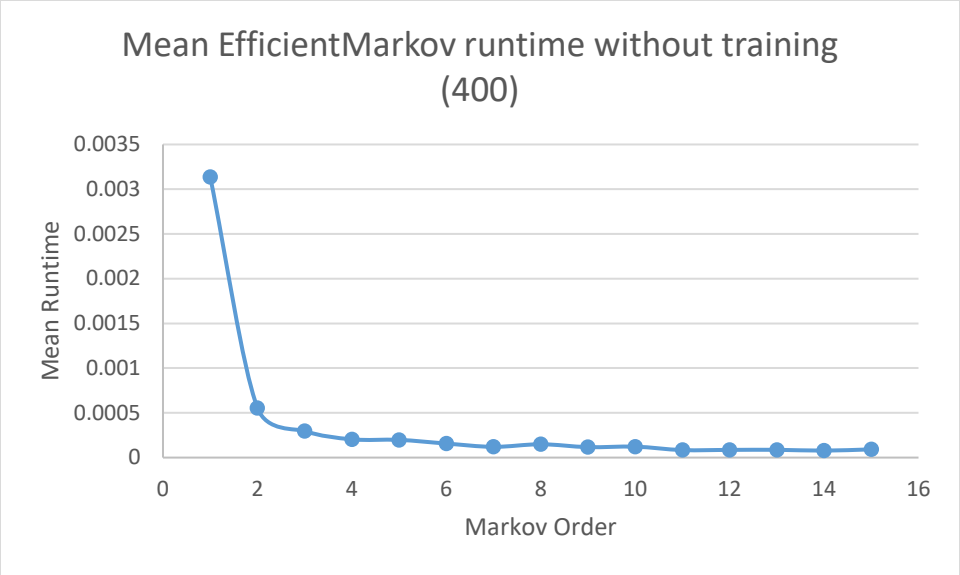
the runtime is instant O(1) as it only retrieves the ArrayList from the HashMap.

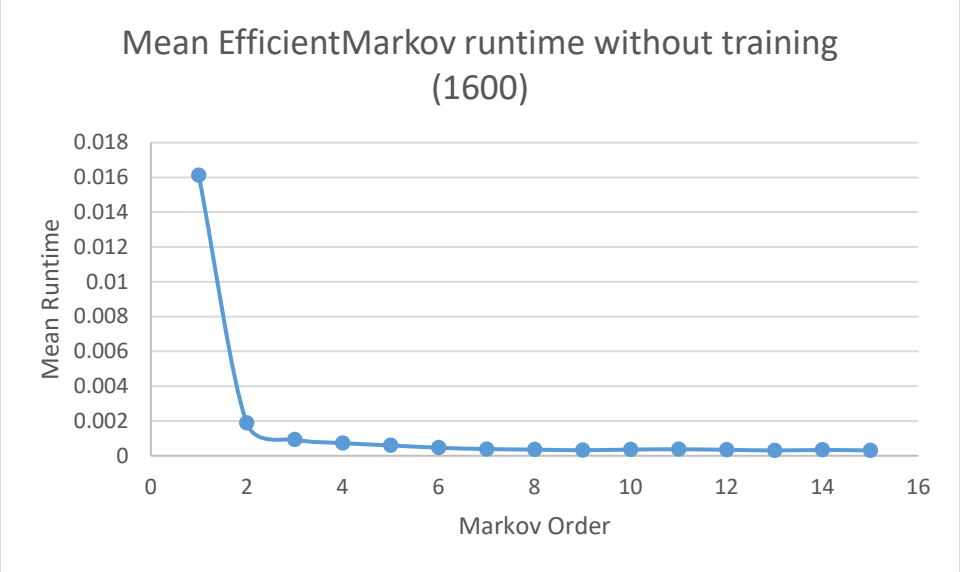


As seen in the graph above, as the training text size increases, there is a linear relationship proportional to ***N***. The correlation coefficient of which, yields 0.997171, an extremely strong linear correlation. In conclusion, the above results show that the hypothesis for runtime including training text is very true.

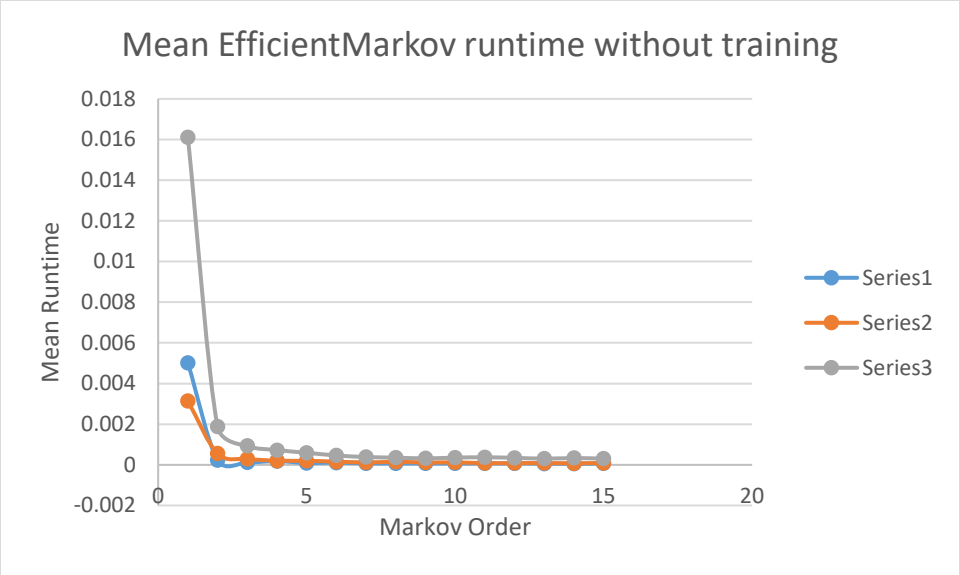
To show when training time is not taken into account, I modified benchmark and moved double start=System.nanoTime() after the line model.setTraining(source). When the training time is not taken into account for EfficientMarkov, the graphs for varying ***k*** show:







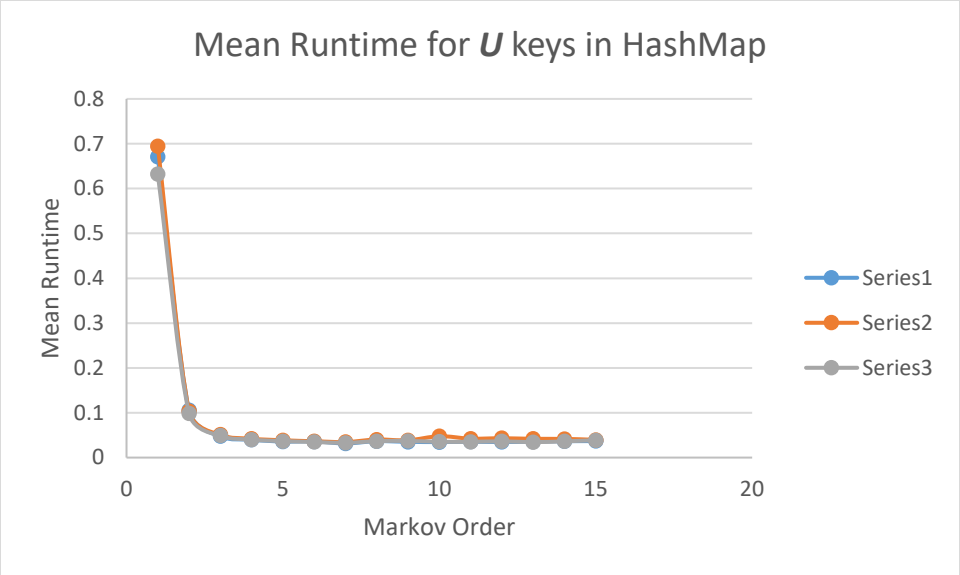
This follows the hypothesis stated above, which is that the runtime is proportional to the ***T***, the number of random characters generated. As seen in the graphs above, as ***T*** increases from 100 to 1600, the mean runtime increases. This is also seen in the graph below, which combines all 3.



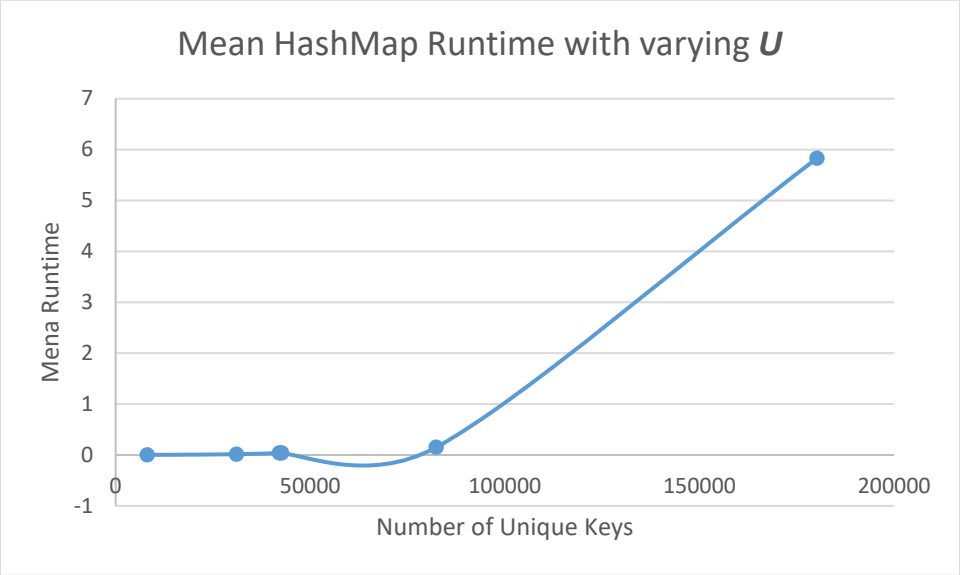
As seen in the graph, Series 3, where ***T*** is 1600 has the largest runtime in comparison to Series 1, where ***T*** is 100. Taking the median point of the three samples and finding the linear correlation coefficient between mean runtime and ***T***, gives 0.996668. This is an extremely strong linear correlation which proves that the above hypothesis is true.

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

I will evaluate HashMap first. To do this, I moved the line times[i] =(System.nanoTime()-start)/1.0e9 to after model.setTraining(source). This way the timer would only start before the adding of keys and end after the map has been created. If we keep the same size text and vary only random characters ***T***, the results show that the mean runtime is in fact independent. All three series below (100, 400 and 1600), show a very similar mean runtime. If we solve for the correlation coefficient of runtime and varying ***T*** through the median point of each series, the result gives -0.07545, which is an extremely weak relationship and shows that the runtime is independent of ***T***.

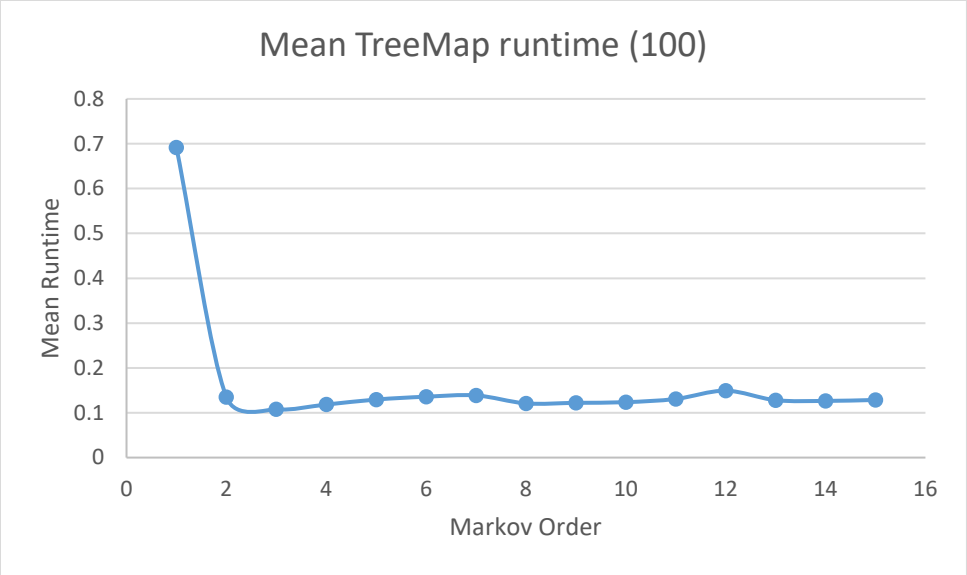


Since Inserting ***U*** keys in a HashMap is ***O(U)***, the runtime is proportional to ***U***. Thus, if we graph the results of varying unique keys added into a HashMap, the result gives:

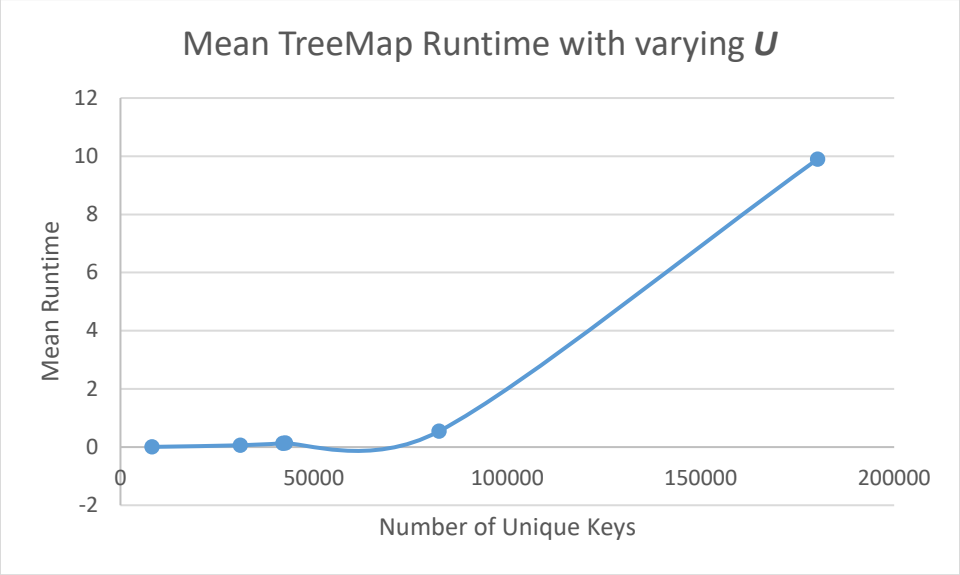


As we can see, as ***U*** increases, there is an increase in the mean runtime. If we solve for the linear correlation coefficient of the plotted values, we get 0.928739, which is an extremely strong positive relationship that supports the hypothesis.

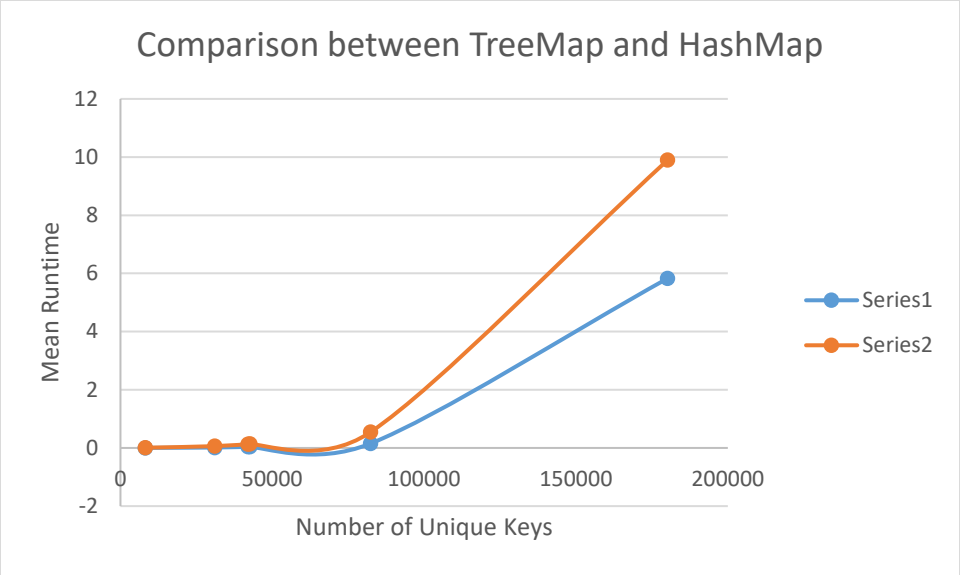
To evaluate a TreeMap, I had to change where I declared a HashMap in my EfficientMarkov, to a TreeMap. The graph below shows the mean runtime as the ***k*** increases. A part from the outliers in order 1 and 2, the graph shows to be relatively similar which shows that the runtime is independent of ***k***.



Graphing the results below where the number of unique keys are varied gives:



The graph shows to be somewhat similar to the HashMap, except runtime is significantly longer. The hypothesis states that the runtime for a TreeMap is ***O(U log U)***, which is similar to a linear relationship whe ***U*** is small. However, as ***U*** increases, ***O(U log U)*** diverges heavily from a linear relationship, but is still significantly smaller than ***O(U^2)***. This is seen in the graph below.



Series 2 is the TreeMap mean runtime, and Series 1 is the HashMap mean runtime. As the number of unique keys increase, the rate of increase from the TreeMap increases more drastically. This is because as the unique keys are added into the map, the map needs to be maintained and sorted, thus, contributing to the increased runtime. Thus, the graph supports the stated hypothesis.