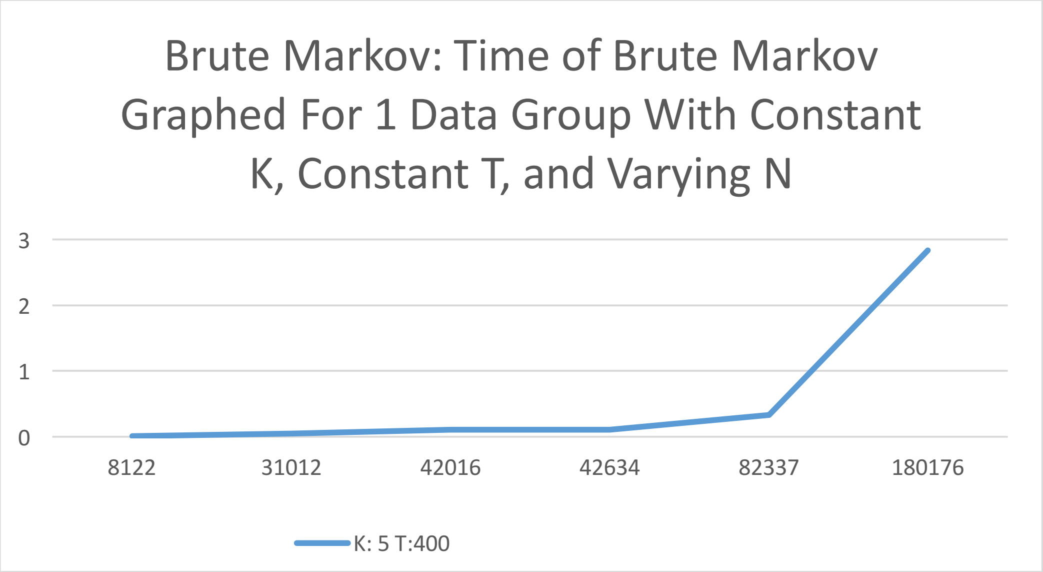
Bella Hutchins
Markov Analysis
October 7, 2016

Note: I used the wording from the google doc when retyping out these hypotheses.

**Brute Markov Hypothesis (BH):** the runtime for generating *T* random characters when trained on a text of size N is *O(NT)*. So, it takes time proportional to *NT*, and this is independent of *K,* where K is the order of the Markov process.

Part 1 of hypothesis: the runtime for generating T random characters is proportional to N
Part 2 of hypothesis: the runtime for generating T random characters is also proportional to T
Part 3 of hypothesis: the runtime for generating T random characters is independent of K

Looking at Part 1 of the BH: *Is the runtime proportional to N?*


Brute Markov: Time of Brute Markov Graphed For 1 Data Group With Constant K, Constant T, and Varying N

Key:
K = markov order, T = number of characters, N = length of text file
X axis: N (length of text file)
Y axis: Mean Time (seconds)

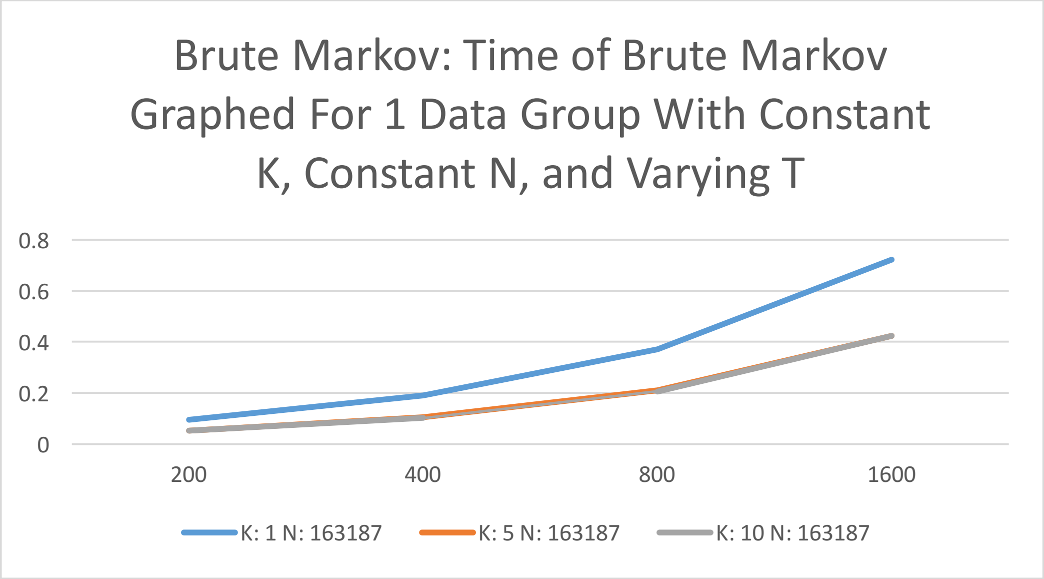
**Answer**: Yes, the runtime for generating T random characters is proportional to N. As depicted in the graph, as the value for N increases, the runtime increases. This becomes clearer and more consistent as the value of N increases greatly. The steeper the increase in the value of N, the steeper the increase in the runtime value, indicating proportionality between N and runtime.

```
while (pos < myText.length()){
                    int start = myText.indexOf(key,pos);
                    if (start == -1){
                           //System.out.println("didn't find "+key);
                           break;
                    }
                    if (start + key.length() >= myText.length()){
```

```
                              //System.out.println("found end with "+key);
                              follows.add(PSEUDO_EOS);
                              break;
                      }
                      // next line is string equivalent of myText.charAt(start+key.length())
                      String next = myText.substring(start+key.length(), start+key.length()+1);
                      follows.add(next);
                      pos = start+1;  // search continues after this occurrence
               }
```

This while loop (found in the getFollows method in BruteMarkov) causes the proportionality between N and the runtime. In order to generate characters, this method must scan through the text to find the characters that follow the key passed into the method. The while loop runs while the value of position is less than myText.length(), so this length, therefore, affects the runtime.

Looking at Part 2 of the BH: *Is the runtime proportional to T?*



Key:
K = markov order, T = number of characters, N = length of text
X axis: T (number of characters)
Y axis: Mean Time (seconds)
Note: There is an orange line behind the grey line!

Note: the focus should be on each individual line, not on the lines in respect to one another.

**Answer**: Yes, the runtime for generating T random characters is proportional to T. As depicted in the graph, as the value for T increases, the runtime increases. The steeper the increase of value T, the steeper the increase in the runtime value, indicating proportionality between T and runtime.

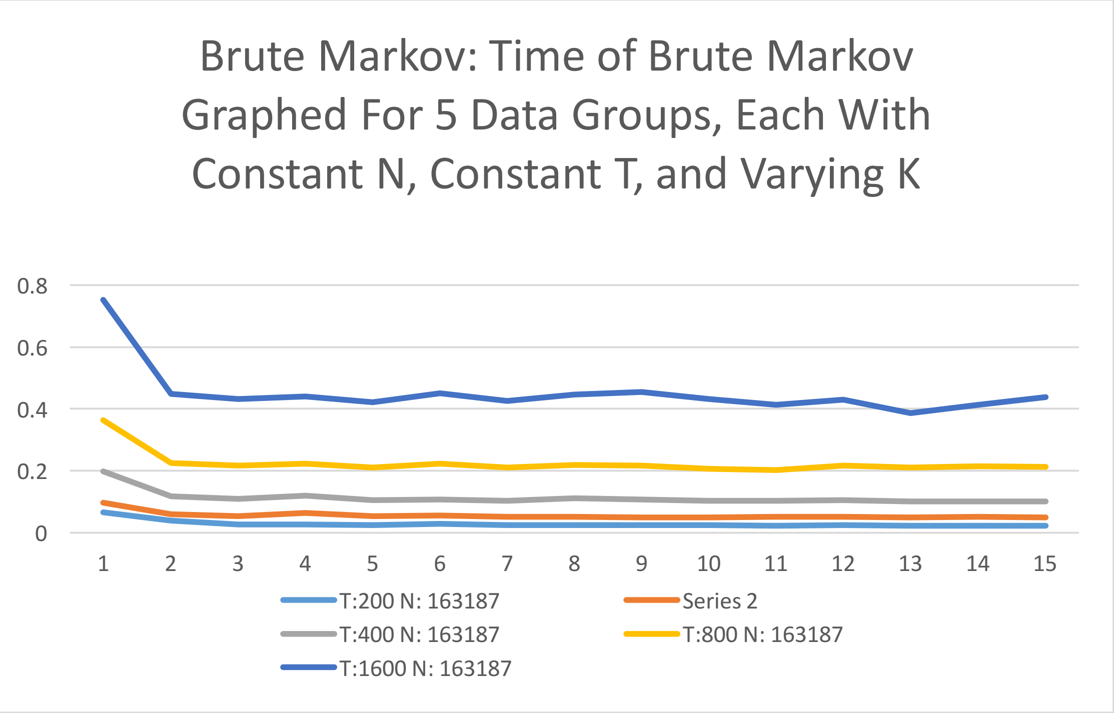
```
for(int k=0; k < length-myOrder; k++){
                      ArrayList<String> follows = getFollows(current);
```

```
                if (follows.size() == 0){
                        break;
                }
                index = myRandom.nextInt(follows.size());

                String nextItem = follows.get(index);
                if (nextItem.equals(PSEUDO_EOS)) {
                        //System.out.println("PSEUDO");
                        break;
                }
                sb.append(nextItem);
                current = current.substring(1)+ nextItem;
        }
```

This for loop (found in the getRandomText method in BruteMarkov) causes the dependency of runtime on the value of T. This for loop calls the getFollows method, which generates the random characters. The variable "length" in the number of characters that need to be generated to create the text, so, the loop will run length-myOrder times, making the runtime proportional to this length, which is the same thing as T.

Looking at Part 3 of the BH: *Is the runtime independent of K?*



Key:
K = markov order, T = number of characters, N = length of text
X axis: K (markov order)
Y axis: Mean Time (seconds)
Series 2: T: 100 N: 163187

Note: the focus should be on each individual line, not on the lines in respect to one another

**Answer:** The runtime for generating T random characters using BruteMarkov is mostly independent of K. As seen in the graph above, each data set, which has a constant N and T and a varying K shows that the time is mostly not dependent on K. Rather than rising or dropping with variations in K, the runtime remains somewhat constant for each data group. However, it seems that having a K value of 1 seemed to slow down the runtime, which we can also see when we look at the graph called "Looking at Part 2 of the BH." I think the effect that the K value had on the runtime makes sense, and I also think that having a big K value would affect runtime as well, however, we're looking at values up to 15, as having really big values would slightly defeat the purpose of Markov. But overall, I don't believe that the runtime should be/is totally independent of K, as explained below:

The reason I think that K values could affect run time is because the line "`int start = myText.indexOf(key,pos);`" in the getFollows method of BruteMarkov means that the program will move through myText based on how often "key" appears, and key is more likely to appear more often if the K value is smaller. So, perhaps it makes sense that having a really small K value would mean K appears very often, meaning the value of "start" takes a longer time to reach the value of myText.length-myOrder which is the value that causes the method's main loop to break. Using similar logic, it would therefore make sense for large K values to decrease the runtime, as the loop might execute less times, so I'm surprised the graphs remained so flat once K was larger than 1.

`for(int k=0; k < length-myOrder; k++){ and if (start + key.length() >= myText.length()){`
These are loops from both the getRandomText method and the getFollows method, which could potentially affect the runtime if the K value is big because it would mean the program might break out of the loop faster, because mylength-myOrder will be smaller if myOrder is big; however, I wouldn't expect this to make much of a noticeable difference unless the myOrder values were really big (which they wouldn't be in Markov).

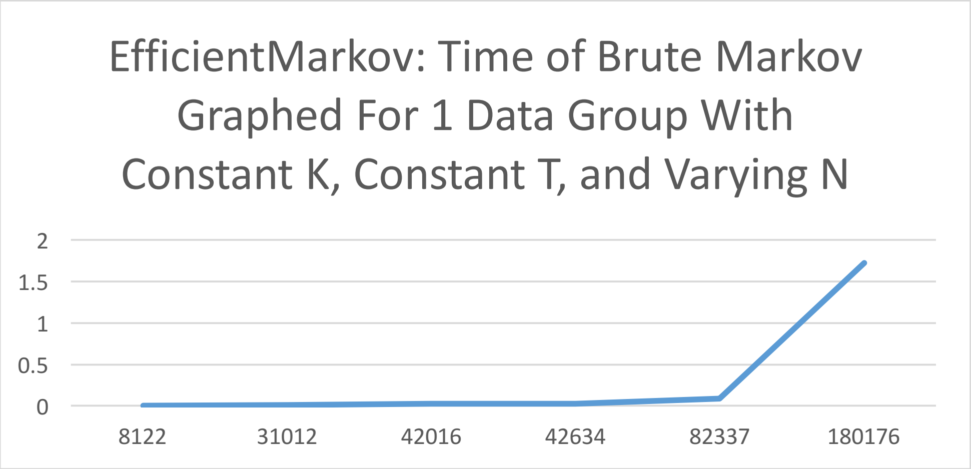
So, I think the value of K has a bigger impact on the getFollows method of BruteMarkov than on the getRandomText method because of the "int start" line pasted above. I think that the other 2 lines of code pasted above (the for loop and if statement) indicate that if K did affect them, it would only happen towards the end of the loops execution, depending on how big K is. So, I don't think K would have a huge impact in this way.

Conclusion: The Brute Markov Hypothesis proved mostly true. The runtime for generating *T* random characters when trained on a text of size N is *O(NT)*. So, it takes time proportional to *NT*, and this is mostly independent of *K,* where K is the order of the Markov process, but not completely, as I explained above. I believe that if there were higher discrepancies between the K values we benchmarked, that we would see that runtime decreases as K values increase. I was overall surprised by the hypothesis and the results of the relationship between K and the runtime. The reason that N and T are multiplied in the runtime O measurement is because the while loop causing the N time is nested within the for loop causing the T time, meaning the two time variables are multiplied.

**Efficient Markov Hypothesis (EH):** generating *T* random characters when trained on a text of size *N* (ignoring training time) is *O(T.)* So, it 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, (which we won't in this analysis), the time is *O(N + T)* since training time will be proportional to *N*, the size of the training text.

Part 1 of hypothesis: the runtime for generating T random characters is proportional to N
Part 2 of hypothesis: the runtime for generating T random characters is also proportional to T
Part 3 of hypothesis: the runtime for generating T random characters is independent of K

Looking at Part 1 of the EH: *Is the runtime proportional to N?*



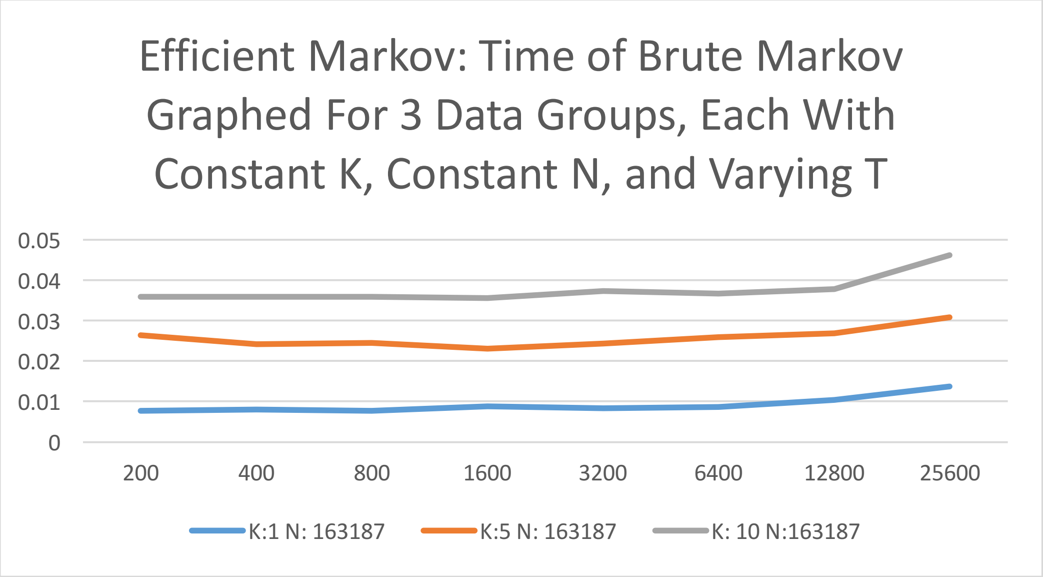
Key:
K = markov order, T = number of characters, N = length of text file
Here, K is 5 and T is 400
X axis: N (length of text file)
Y axis: Mean Time (seconds)

**Answer:** Yes, the runtime for generating T random characters is proportional to N. As depicted in the graph, as the value for N increases, the runtime increases. The steeper the increase of value N, the steeper the increase in the runtime value, indicating proportionality between N and runtime (you can see this most clearly between X values 82337 and 180176).

```
for (int i=0; i<=myText.length()-myOrder;i++){
                        sub = myText.substring(i, i+myOrder);
                        if (kgrams.containsKey(sub)){
                                if (i+myOrder>=myText.length()){
                                        kgrams.get(sub).add(PSEUDO_EOS);
                                        break;
                                }
                                kgrams.get(sub).add(String.valueOf(myText.charAt(i+myOrder)));
                        }else{
                                kgrams.put(sub, new ArrayList<String>());
                                if (i+myOrder>=myText.length()){
                                        kgrams.get(sub).add(PSEUDO_EOS);
                                        break;
                                }
                                kgrams.get(sub).add(String.valueOf(myText.charAt(i+myOrder)));
                        }
                }
```

This for loop (found in the setTraining method in EfficientMarkov) causes the proportionality between N and the runtime. In order to generate characters, this method must scan through (almost) the entire text to find the characters that follow the key passed into the method. Scanning through the text, as we can see from the for loop statement (for int i=0; i< myText.length-myOrder; i++) takes myText.length-myOrder time. So, this creates the proportionality between N and runtime.

Looking at Part 2 of the EH: *Is the runtime proportional to T?*



Key:
K = markov order, T = number of characters, N = length of text
X axis: T (number of characters)
Y axis: Mean Time (seconds)

Note: I changed line 112 in the Benchmark program to "for (int i=200; i<25600; i=2){" when I generated the data to create this graph.

Note: the focus should be on each individual line, not on the lines in respect to one another.

Yes, the runtime for generating T random characters is proportional to T, but this only becomes clear at T gets really big. When T is between 200-1600, as it was for the Markov project, T didn't have much of an effect on runtime. As depicted more on the right half of the graph, as the value for T increases, the runtime increases. The steeper the increase of value T, the steeper the increase in the runtime value, indicating proportionality between T and runtime (you can mostly only see this at the end of the graph, when the T value becomes really big). I think the reason a bigger T shows a clearer relationship between T and runtime is because the program is very efficient, so it's hard to break down the factors that contribute to the runtime until the time slows down a bit.

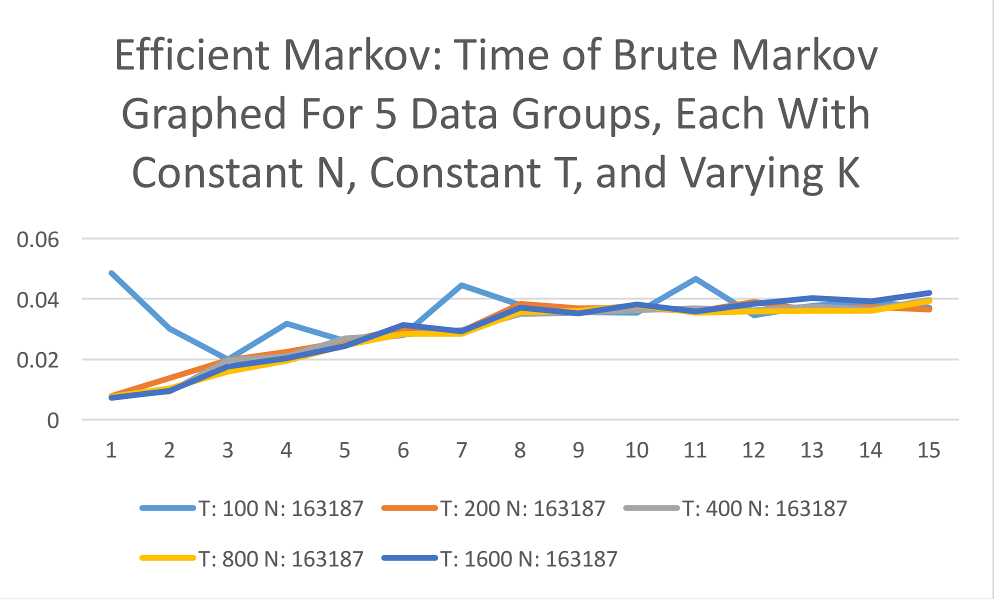
```
for(int k=0; k < length-myOrder; k++){
			ArrayList<String> follows = getFollows(current);
			if (follows.size() == 0){
				break;
```

```
                }
                index = myRandom.nextInt(follows.size());

                String nextItem = follows.get(index);
                if (nextItem.equals(PSEUDO_EOS)) {
                        //System.out.println("PSEUDO");
                        break;
                }
                sb.append(nextItem);
                current = current.substring(1)+ nextItem;
        }
```

This for loop (found in the getRandomText method in EfficientMarkov) causes the dependency of runtime on the value of T. This for loop calls the getFollows method, which returns the random characters. The variable "length" in the number of characters that need to be generated to create the text, so, the loop will run length-myOrder times, therefore making the runtime proportional to length, which is the same thing as T (the number of characters).

Looking at Part 3 of the EH: *Is the runtime independent of K?*



Key:
K = markov order, T = number of characters, N = length of text
X axis: K (markov order number)
Y axis: Mean Time (seconds)

Note: the focus should be on each individual line, not on the lines in respect to one another

**Answer:** From the graph, it seems as though the data group generating 100 characters of text was independent of the variable K, but that the other groups of data were somewhat proportional to K in that their general curves ascend as the K value ascends. Perhaps only generating 100 characters didn't allow enough time to show that K noticeably impacted the runtime. Because of the brute force algorithm in BruteMarkov, I would expect K to have affected that runtime more than the runtime of EfficientMarkov, for the reasons I explained above. However, I've offered some explanations as to how K might affect the EfficientMarkov process, though I am confused how K impacted EfficientMarkov more than it did BruteMarkov.

```
for(int k=0; k < length-myOrder; k++){
```

This is the for loop from the getRandomText method in EfficientMarkov. As I theorized in my explanation for the BruteMarkov K Value hypothesis, stopping the loop once the loop iterator is less than length-myOrder could impact the run time because it could mean that a bigger myOrder value would cause the loop to end earlier, but I think this would only be noticeable if the K value was really big, otherwise not that many for-loop executions are avoided.

```
if (i+myOrder>=myText.length()){
```

This is the if statement from the setTraining method in EfficientMarkov. If the statement == true, the big loop in the setTraining method is broken. Perhaps this effects runtime in the same way that the for loop above does, in that a really large myOrder value might cause the program to break out of the loop faster, however I wouldn't expect this to make much of a noticeable difference unless the myOrder value were really big.

I wouldn't expect K (also called myOrder) to impact either the for loop or if statement very much, and am therefore surprised that EfficientMarkov's runtime seems to have a more proportional relationship with K than BruteMarkov's runtime.

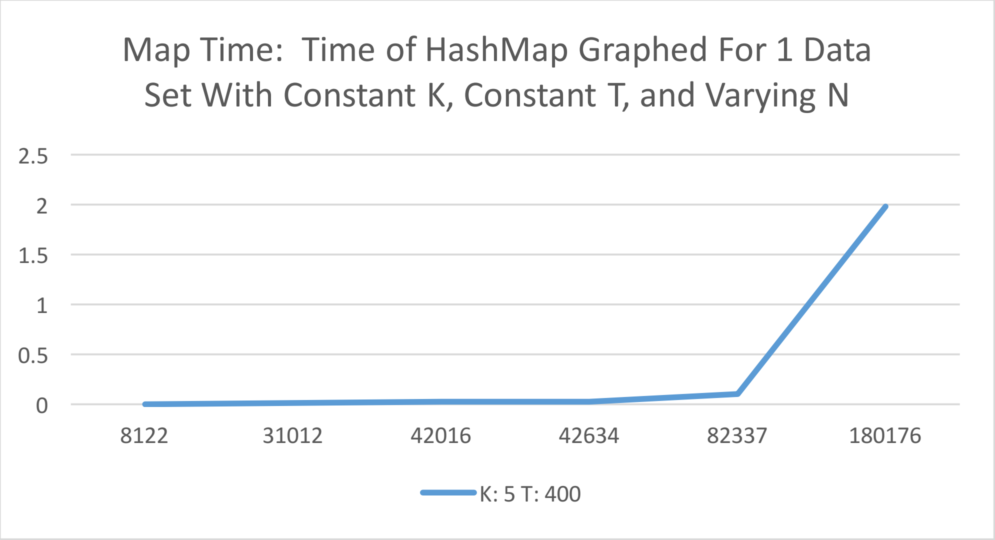
Conclusion: The Efficient Markov Hypothesis proved somewhat true. The runtime for generating *T* random characters when trained on a text of size N is *O(N+T)*, where the runtime is proportional to both N and T. However, it overall didn't seem as though the runtime was independent of *K,* where K is the order of the Markov process, and I described possibilities for this above. The reason that the runtime is proportional to N + T is because one for loop takes N time and is executed once and the other for loop takes T time and is also only executed once. The two loops are not nested, so N and T are added, not multiplied.

**Map Time Hypothesis (MH):** **Map Hypothesis** or **MH**: inserting *U* keys in a HashMap is *O(U)* and inserting *U* keys into a Treemap is *O(U log U)*. (We are under the assumption that if the size of the text is N, then there are roughly N K-grams, which is only really valid when K is small, which it is in this benchmark program).

To make these charts, I changed the Benchmark code slightly: I moved the line "times[i] = (System.nanoTime()-start)/1.0e9;" to right after line 40.

Part 1: The runtime to insert U keys into a HashMap is proportional to U, where U is the number of grams (so, approx. N).
Part 2: The runtime to insert U keys into a TreeMap is proportional to U, where U is the number of grams (so, approx. N). This runtime should be longer than that of HashMap, as indicated by the "U log U"

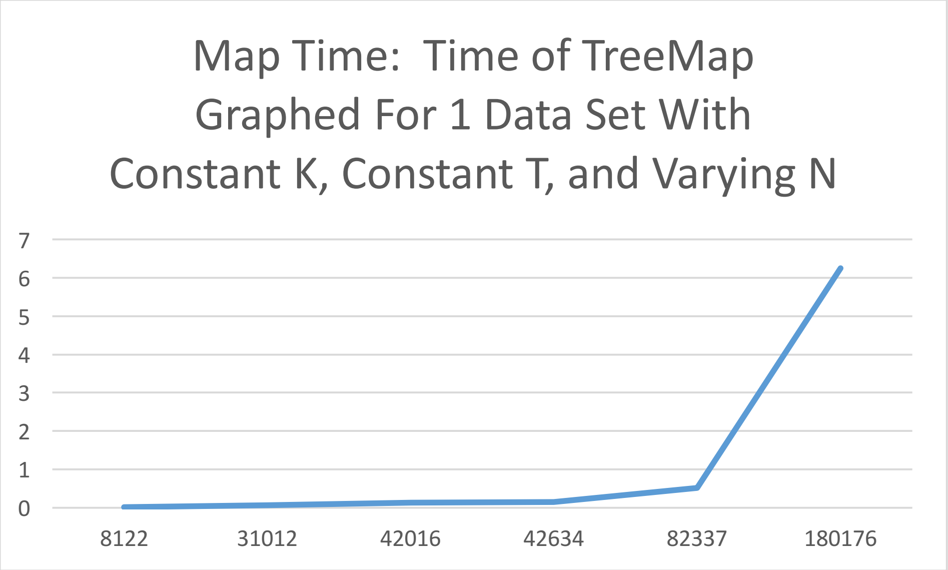
Part 1 of the MH: *Is the runtime proportional to U for a HashMap?*



Key:
K = markov order, T = number of characters, N = length of text
X axis: N (length of text)
Y axis: Mean Time (seconds)
N is the same thing as U

**Answer:** Yes, the runtime for inserting U keys into a HashMap is proportional to U. As depicted in the graph, as the value for U increases, the runtime increases. The steeper the increase of value U, the steeper the increase in the runtime value, indicating proportionality between U and runtime.

Part 2 of the MH: *Is the runtime proportional to U for a TreeMap? Does the insertion time take longer for a TreeMap than a HashMap?*



Key:
K = markov order, T = number of characters, N = length of text

X axis: N (length of text)
Y axis: Mean Time (seconds)
N is the same thing as U

**Answer:** Yes, the runtime for inserting U keys is proportional to U. As depicted in the graph, as the value for U increases, the runtime increases. The steeper the increase of value U, the steeper the increase in the runtime value, indicating proportionality between U and runtime. Also, the time for inserting U keys into a TreeMap is slower than the time for inserting U keys into a HashMap, as predicted by the hypothesis.

**Conclusion**: The Map Hypothesis is accurate in that the runtime to add U keys to a HashMap and to a TreeMap is proportional to U and the time to add U keys to a TreeMap is longer than the time to add U keys to a HashMap