Callie Mao

MARKOV ANALYSIS

PART A

1. Using the brute force code to generate text with the Romeo and Juliet training text, the following data was collected:

	Time to Generate		
# of Random Characters	Order-5 Text	Order-1 Text	Order-10 Text
100	0.038	0.03	0.023
200	0.059	0.063	0.055
400	0.096	0.126	0.072
800	0.194	0.274	0.19
1600	0.369	0.503	0.372

Doubling the size of the number of characters approximately doubles the amount of time it takes to generate text. Based on the above data, the brute-force model appears to be O(N).

For different order text, the times taken to generate differing numbers of random characters varies but are still relatively similar. An order-10 model takes approximately the same amount of time as an order-5 model. However, an order-1 model takes a greater amount of time likely because there are more 1-grams in this text than 5- and 10-grams.

2. I expect the brute-force code to take approximately 3 times the time it took for each group of characters (100, 200, 400, etc.) since the Scarlet Letter has approximately 3 times as many characters and the time of generation is based largely upon the length of the text. The following empirical results were collected:

# of Random Characters	**Predicted Time**	**Elapsed Time (Empirical)**
100	0.038*3=0.114	0.077
200	0.059*3=0.177	0.166
400	0.096*3=0.288	0.286
800	0.194*3=0.582	0.555
1600	0.369*3=1.107	1.091

This data matches up with the predicted and still reflects the O(N) runtime. Based on this same logic, the King James Bible should take about 4,400,000/153,000=28 times as long as Romeo and Juliet took for 5-order text, or about 0.369*28=10.33 seconds.

3.

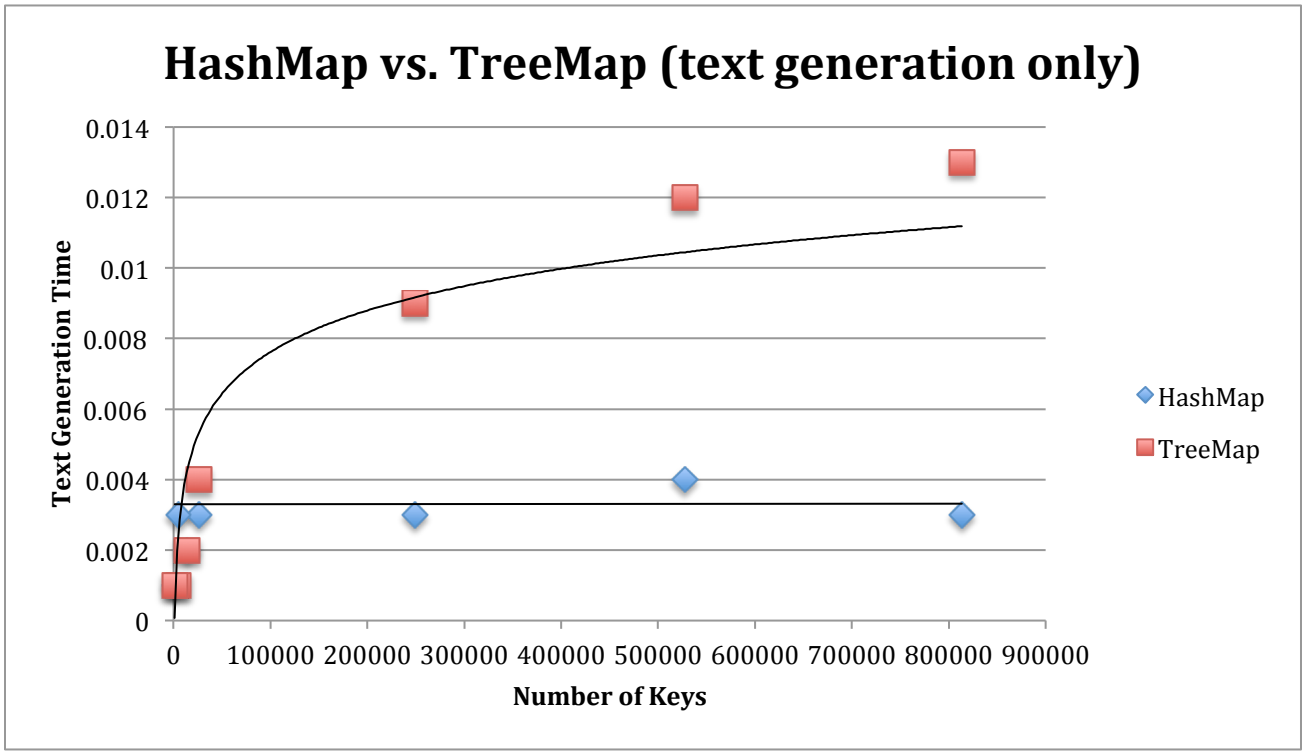
	Time to Generate	
# of Random Characters	Order-5 Text (not including map creation time)	Creating a Map
100	0	0.103
200	0.0010	0.1
400	0.0010	0.11
800	0.0020	0.103
1600	0.0010	0.112

The time needed to generate order-5 text with the new MapMarkovModel method is significantly faster than that of the brute-force code. The amount of time taken to generate order-5 text is fairly constant and seems to take around 0.0010 seconds consistently. Generating order-5 text is around O(1) with MapMarkovModel. Additionally, the amount of time taken to generate the map does not depend on the number of characters to generate, is O(1), and is only contingent upon the size and variability of the training text (which is held constant with romeo.txt in this case).

PART B

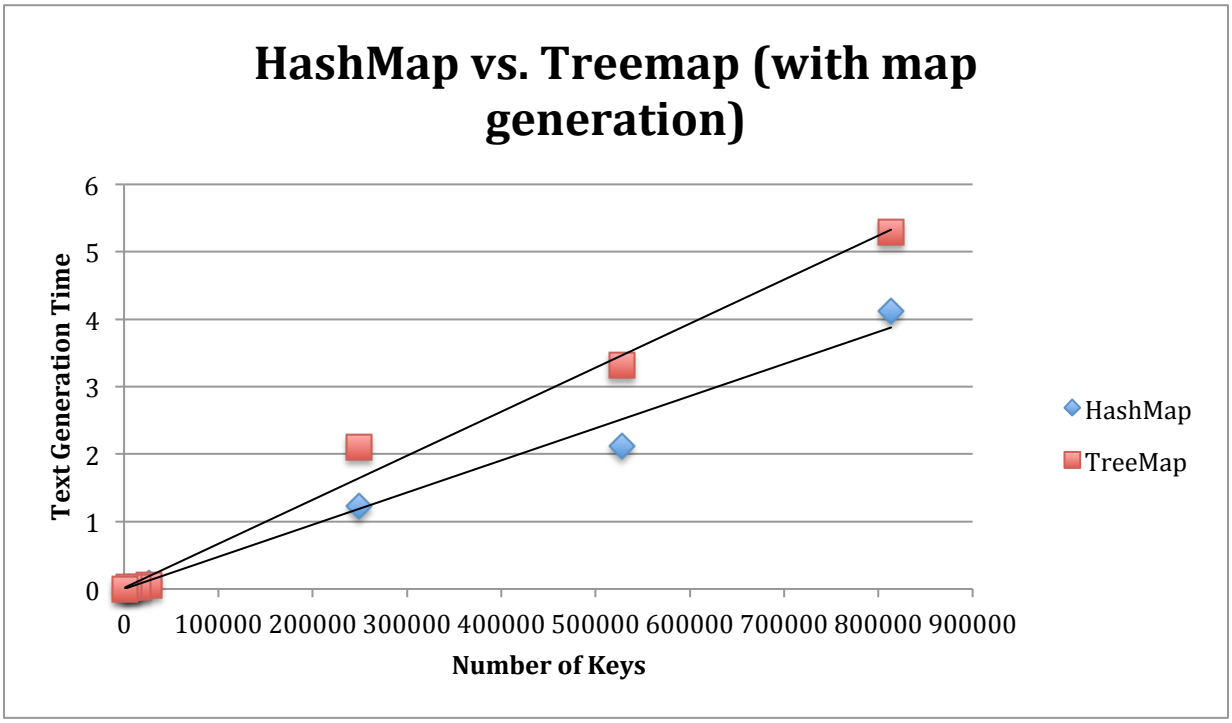
The following data was collected for HashMap and TreeMap text generation times *not including* the time taken to generate the initial map:

K (n-gram length)	File	# Keys	HashMap Text Generation Time	TreeMap Text Generation Time
10	Kjv10	813486	0.003	0.013
3	Kjv10	527456	0.004	0.012
2	Kjv10	249662	0.003	0.009
5	Alice	26222	0.003	0.004
5	Melville	14316	0.002	0.002
1	Alice	5313	0.003	0.001
5	Decl-independence	1195	0.001	0.001



The following data was collected for the time taken to generate the text *including* the time taken to generate the original map:

K-value	File	# Keys	HashMap Text Generation Time	TreeMap Text Generation Time
10	Kjv10	813486	4.125	5.29
3	Kjv10	527456	2.119	3.31
2	Kjv10	249662	1.225	2.111
5	Alice	26222	0.088	0.054
5	Melville	14316	0.044	0.022
1	Alice	5313	0.019	0.019
5	Decl-independence	1195	0.005	0.002



I added the Declaration of Independence text (obtained from http://www.archives.gov/exhibits/charters/declaration_transcript.html) and tested it in addition to the provided training texts. For TreeMaps, there appears to be a logarithmic increase in text generation time as the number of keys increases. As the number of n-Grams and keys in the TreeMap increases, the text generation time continues to increase but at a slower rate than previous values. Because of this relationship, the time to perform operations for TreeMaps from this data appears to be O(logN).

For the Hashmap, the text generation time for all the number of keys hits a ceiling around 0.003 seconds. The graph suggests that HashMap's runtime has a relatively constant, linear relationship and supports the notion that HashMaps have a runtime of O(1).

A HashMap simply needs to use hashCode() for individual elements while TreeMaps may need to use compareTo() to compare an element to many other different elements. Since WordNgrams do not start out sorted (they are in arbitrary order), it will take a longer time to sort through *and* create a map of the keys and values than a HashMap will take, especially when the number of keys is large.