COMP550 Markov Model Extra Credit
Michael Gu

Part A

1. Brute Force Runtimes on romeo.txt

# Characters	order-1	order-5	order-10
100	0.087	0.042	0.031
200	0.102	0.064	0.065
400	0.189	0.12	0.111
800	0.337	0.282	0.243
1600	0.7	0.478	0.495
3200	1.885	1.526	1.118

The order-5 time grows in a manner that is linear with respect to the number of characters, or O(N) where N = number of characters. This is also the case with order-1 and order-10 Markov models.

Order-1 has the longest runtime because under brute force, for each character generated we have to traverse entire text to find occurrences of the previous character. Because there are more occurrences of 1-grams than 5-grams or 10-grams, 1-gram traversals take more space and thus more time to build character sequences

For the same reason, order-10 is about 75% faster than order-5 markov model because there are fewer occurrences of 10-grams, making each character choice is highly deterministic.

2.

Because order-5 grows at O(N), where N is number of characters to build, I believe the growth of the training text will have a similar effect

To predict the runtime, we calculate the ratio of characters between Hawthorne and Romeo = 500,000/150,000 = 3.3333

Because we aren't storing history of our traversals in a map, more training text will have same effect as more characters to generate. Either way, we must traverse either more text or generate more character instances, which will grow in a linear fashion with respect to time.

Thus I expect each runtime to increase by approximately a factor of 3.

Brute Force Runtimes on Hawthorne, Prediction vs. Empirical

# Characters	Prediction	Empirical
400	0.12*3=0.36	0.341
800	0.282*3=0.846	0.679

1600	0.478*3=1.434	1.382

My predictions are slightly off from the empirical results, by ~5%.

King James Bible has 4.4 million characters. 4.4 million / 150,000 = 29.3
As before, we take how long it took to produce 1,600 on romeo.txt using order-5 markov model, and multiply by factor of 30, which equals 0.478 * 30 = 14.34 s

3.

Map/Smart Model, order-5 Markov Model

# Characters	Time
200	0.001
400	0.002
800	0.003
1600	0.006

Because we are no longer recreating the map for every character generated, the Map implementation of Markov chain operates at about 2-5% the time of the brute force method.

For each situation, the first traversal creates the map necessary while subsequent traversals perform get operations on the map. Thus the first traversal takes the most time, while subsequent ones will grow linearly with respect to how many characters need to be generated.

Part B

Key vs. Time for generating 32,000 characters using order-2 Markov Model & various lexicons

HashMap

Lexicon	# Keys	Time
alice.txt	17471	0.032
clinton08.txt	1978	0.051
hawthorne.txt	56451	0.04
mccain-legion08.txt	2560	0.033
melville.txt	11322	0.032
romeo.txt	20883	0.035
obama08.txt	3944	0.034
palin-accept.txt	1246	0.033
poe.txt	2061	0.037
unabomber.txt	22308	0.057
kingjames.txt	249986	0.05

TreeMap

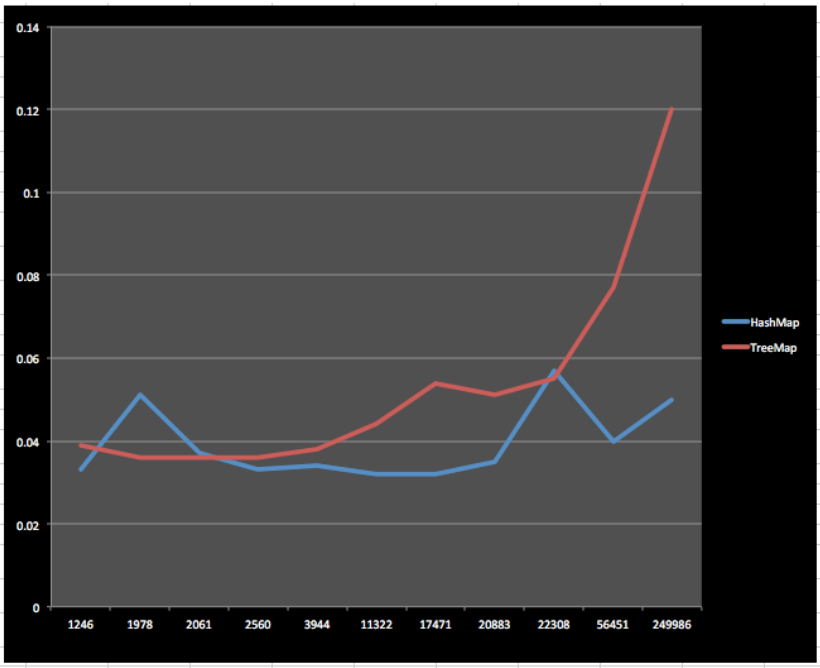
Lexicon	# Keys	Time
alice.txt	17471	0.054
clinton08.txt	1978	0.036
hawthorne.txt	56451	0.077
mccain-legion08.txt	2560	0.036
melville.txt	11322	0.044
romeo.txt	20883	0.051
obama08.txt	3944	0.038
palin-accept.txt	1246	0.039
poe.txt	2061	0.036
unabomber.txt	22308	0.055
kingjames.txt	249986	0.12

Sorting the results by key in ascending order, then graphing these results:

Lexicon	# Keys	Time
palin-accept.txt	1246	0.033
clinton08.txt	1978	0.051
poe.txt	2061	0.037
mccain-legion08.txt	2560	0.033
obama08.txt	3944	0.034
melville.txt	11322	0.032
alice.txt	17471	0.032
romeo.txt	20883	0.035
unabomber.txt	22308	0.057
hawthorne.txt	56451	0.04
kingjames.txt	249986	0.05

TreeMap

Lexicon	# Keys	Time
palin-accept.txt	1246	0.039
clinton08.txt	1978	0.036
poe.txt	2061	0.036
mccain-legion08.txt	2560	0.036
obama08.txt	3944	0.038
melville.txt	11322	0.044
alice.txt	17471	0.054
romeo.txt	20883	0.051
unabomber.txt	22308	0.055
hawthorne.txt	56451	0.077
kingjames.txt	249986	0.12



At small key values, the two implementations have approximately similar runtimes. Across all keys found in our lexicons, TreeMap can take 15-55% longer compared to the HashMap implementation.

As keys get large, the performances of the two implementations diverge and can be identified as significantly different from each other. Using kingjames.txt with 4.4 million characters and 249,986 keys generated, the runtime for TreeMap is nearly double that of HashMap.