Determine (from running Benchmark.java) how long it takes for `MarkovModel` to generate 200, 400, 800, and 1600 random characters using (the default alice.txt) file and orders of 1, 5, and 10. Include these timings in your report. Do the same for the file hawthorne.txt. Include these empirical results in your REFLECT document. You can copy/paste from running the program or create a table.

alice.txt

```
Varying order, text length 100, source size 163187
order      #chars     source     mean       sigma
1          100        163187     0.085      0.002
5          100        163187     0.031      0.000
10         100        163187     0.027      0.000

Varying order, text length 200, source size 163187
order      #chars     source     mean       sigma
1          200        163187     0.137      0.004
5          200        163187     0.057      0.000
10         200        163187     0.053      0.000

Varying order, text length 400, source size 163187
order      #chars     source     mean       sigma
1          400        163187     0.246      0.004
5          400        163187     0.130      0.000
10         400        163187     0.114      0.000

Varying order, text length 800, source size 163187
order      #chars     source     mean       sigma
1          800        163187     0.394      0.000
5          800        163187     0.228      0.000
10         800        163187     0.220      0.000

Varying order, text length 1600, source size 163187
order      #chars     source     mean       sigma
1          1600       163187     0.807      0.001
5          1600       163187     0.471      0.001
10         1600       163187     0.512      0.002
```

hawthorne.txt

```
Varying order, text length 100, source size 496768
```

```
order      #chars     source     mean       sigma
1          100        496768     0.208      0.005
5          100        496768     0.098      0.000
10         100        496768     0.097      0.003

Varying order, text length 200, source size 496768
order      #chars     source     mean       sigma
1          200        496768     0.467      0.028
5          200        496768     0.194      0.000
10         200        496768     0.153      0.000

Varying order, text length 400, source size 496768
order      #chars     source     mean       sigma
1          400        496768     0.740      0.007
5          400        496768     0.398      0.000
10         400        496768     0.344      0.000

Varying order, text length 800, source size 496768
order      #chars     source     mean       sigma
1          800        496768     1.441      0.031
5          800        496768     0.747      0.001
10         800        496768     0.738      0.000

Varying order, text length 1600, source size 496768
order      #chars     source     mean       sigma
1          1600       496768     2.618      0.006
5          1600       496768     1.460      0.001
10         1600       496768     1.561      0.037
```

Do your results justify the claim that `MarkovModel` has an NT running time as described in this document? Explain and justify.

Yes, because if we let N be the number of characters in the file and T be the number of characters we need to generate, we find that N is constant for a given file. However, in our trials, we change T, from 100 to 200 to 400 to 800 to 1600. Each time, for the same order, we find that the mean time in the next trial up (for example, from 100 to 200 or from 400 to 800, et.) is approximately double that of the previous one. This relation holds true for any trial for the same file–by doubling the amount of characters needed to be generated, we essentially double the mean time (given a standard deviation of sigma). The relation holds for both alice.txt and hawthorne.txt.

Explain the relationship between the runtimes and the length of the training text and the order of the markov generation. Use a formula or words as you think best. Also include an explanation of how long you think it will take to generate 1600 random characters for an order-5 Markov model with a training text of 5.5 million characters. Justify your answer.

As the length of the training text increases, the run time increased by the same amount. In other words, runtime and length of training text are proportional. For example, the hawthorne text is about 3 times (and a little bit more) longer than the alice text, and each hawthorne trial takes about 3 times longer than the corresponding alice trial, hence supporting the relationship.

Furthermore, as the order of the markov generation increases, the runtime decreases, albeit more and more slowly (logarithmically decreasing relationship). From order 1 to 5 of hawthorne.txt text length 800, for example, the mean time decreases almost 50%, but from order 5 to 10, it doesn't decrease much, if at all.

For an order 5 markov model with a training text of 5.5 mil characters to generate 1600 random char, we find that we already have an order 5-markov model with training text (hawthorne) and approximately 0.5 mil characters that generates 1600 random characters. This gives us a time of approximately 1.46 seconds. Thus, since we proposed that runtime and length of training text are proportional, we can multiply the original time (1.46 seconds) by the quotient of 5.5mil by 0.5 mil, or a factor of 11.

Thus, 11*1.46 sec = 16.06 seconds, which is the proposed time it could take for an order 5 markov model with a training text of 5.5 million characters to generate 1600 random characters.

Change the Benchmark code to use your `EfficientMarkov`

implementation. Run the same tests that you ran for `MarkovModel` and explain whether the runtimes justify the claim that this has an N + T running time as explained above. Include the timing results and your explanation in your REFLECT document (this is a repeat of 1 and 2 above, but for your efficient/map implementation).

alice.txt

```
order      #chars     source     mean       sigma
1          200        163187     0.020      0.000
1          400        163187     0.021      0.000
1          800        163187     0.020      0.000
1          1600       163187     0.021      0.000
5          200        163187     0.048      0.000
5          400        163187     0.047      0.000
5          800        163187     0.051      0.000
5          1600       163187     0.055      0.000
10         200        163187     0.054      0.000
10         400        163187     0.070      0.001
10         800        163187     0.072      0.000
10         1600       163187     0.072      0.000
```

hawthorne.txt

```
order      #chars     source     mean  sigma
1          200        496768     0.069 0.000
1          400        496768     0.062 0.000
1          800        496768     0.058 0.000
1          1600       496768     0.060 0.000
5          200        496768     0.175 0.000
5          400        496768     0.179 0.000
5          800        496768     0.180 0.000
5          1600       496768     0.179 0.000
10         200        496768     0.212 0.000
10         400        496768     0.210 0.000
10         800        496768     0.211 0.000
10         1600       496768     0.216 0.000
```

As can be observed from the results derived from the trials, the EfficientMarkov is much faster. The runtimes do justify the claim that this has an N+T running time as opposed to a NT runtime. When the number of characters is doubled within each order, the mean barely changes at all; at most, it increases by 0.005 or so, a fraction of the avg time spent for each trial; this implies that as # of characters increases, time increases, but much less than the NT runtime of the MarkovModel.

Use testfile.txt as the training text and run `MarkovDriver` with an `EfficientMarkov` class to generate 5,000 characters (note that testfile.txt has only 329 characters). You likely will not generate 5,000 characters. How many characters will be generated, on average, if you run this 10 times for an order-5 Markov process. Explain how you arrived at your result.

I actually ran it 10 times with an order-5 markov process with different seeds just to see what it looked like, and my average value was approximately 308 characters. I predict that roughly 329 characters would be generated on average (the length of the testfile.txt file), because there’s an approximately 1/329 chance that the key chosen would have a value that was equal to PSEUDO_EOS (which allows for the passage to end)