You're given a class **`Benchmark`** that runs several tests that allow you to compare the performance of the default, brute-force **`BaseMarkov`** and the more efficient map-based **`EfficientMarkov`** code. The code you start with uses data/hawthorne.txt, which is the text of ***A Scarlet Letter*** a text of 487,614 characters (as you'll see in the output when running the benchmark tests). The class uses **`BaseMarkov`**, but can be easily changed to use **`EfficientMarkov`** by changing the appropriate line in the main method. You're free to alter this class.

Answer the following questions in your analysis.txt file. You'll submit your analysis as a separate PDF as a separate assignment ***to Gradescope***.

1. Determine (from running **`Benchmark.java`**) how long it takes for **`BaseMarkov`** to generate 2,000, 4,000, 8,000, 16,000, and 32,000 random characters using the default file and an order 5 Markov Model. Include these timings in your report. The program also generates 4,096 characters using texts that increase in size from 487,614 characters to 4,876,140 characters (10 times the number). In your analysis.txt file include an explanation as to whether the timings support the O(NT) analysis. Use the fact that for some runs N is fixed and T varies whereas in the other runs T is fixed and N varies.

time	source	#chars
0.109	487614	1000
0.174	487614	2000
0.304	487614	4000
0.753	487614	8000
1.380	487614	16000
2.691	487614	32000
5.561	487614	64000
0.362	487614	4096
0.714	975228	4096
1.040	1462842	4096
1.399	1950456	4096
1.725	2438070	4096
2.083	2925684	4096
2.439	3413298	4096
2.827	3900912	4096
3.145	4388526	4096
3.529	4876140	4096

Because BaseMarkov uses an ArrayList, the loop has to iterate through the entire list to check for each T, resulting in a runtime of O(NT). The runtime O(NT) can be seen in the first set of trials where the source text is held constant and the number of characters double each time. The runtime itself also doubles from trial to trial, showing O(NT) runtime. This run time order can

also be seen when the number of characters generated are fixed and the source text increases by a factor of ten. The runtime also roughly increases by a factor of 10, demonstrating the O(NT) runtime.

2. Determine (from running **`Benchmark.java`**) how long it takes for **`EfficientMarkov`** to generate 2,000, 4,000, 8,000, 16,000, and 32,000 random characters using the default file and an order 5 Markov Model. Include these timings in your report. The program also generates 4,096 characters using texts that increase in size from 487,614 characters to 4,876,140 characters (10 times the number). In your analysis.txt file include an explanation as to whether the timings support the O(N+T) analysis.

time	source #chars	
0.144	487614	1000
0.118	487614	2000
0.069	487614	4000
0.085	487614	8000
0.063	487614	16000
0.072	487614	32000
0.075	487614	64000
0.067	487614	4096
0.157	975228	4096
0.222	1462842	4096
0.310	1950456	4096
0.436	2438070	4096
0.494	2925684	4096
0.607	3413298	4096
0.794	3900912	4096
1.234	4388526	4096

By using a HashMap, the runtime for searching through has a runtime of only O(1), so after creating the map with runtime O(N), the only extra added time is O(1) going through T times. The first set of trials is where N is held constant and the T is increased exponentially. The time differences between each trial are so small they are pretty negligible, and they don't follow the runtime of adding T to N exactly. The runtime of the second set of trials where T was held constant does have an increase as N increases, and they are much less dramatic than the increase in runtime with O(NT).