**Question 1**

The asymptotic (big O) runtime complexity of the setTraining() method for the BaseMarkov implementation is expected to be O(T). This is because the only algorithm used in the method is split(), which splits the training text into separate words and adds each one to the array myWords. As T is the number of words in the training text, this addition would happen T times, making the asymptotic runtime complexity O(T).

The asymptotic (big O) runtime complexity of the getRandomText() method for the BaseMarkov implementation is expected to be O(NT). As we are given that nextInt() is a constant time operation, the first four rows of getRandomText() is constant time as they are generations of new array lists, integers, etc. The for loop, then, iterates for (N – order) times. This means the big O of the loop is O(N – order), which is equivalent to O(N) times because we can drop the order constant. Inside the loop, the runtime for getFollows() is O(T) since the for loop used for this method in WordGram runs for T times, with other codes running in constant time. Hence, with O(T) being iterated for N times, they are multiplied, so the final asymptotic runtime complexity is O(NT).

Experiment:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data file** | **T** | **N** | **Training Time (s)** | **Generating time (s)** |
| biden-2021.txt | 6,129 | 100 | 0.010 | 0.053 |
| biden-2021.txt | 6,129 | 1,000 | 0.014 | 0.248 |
| biden-2021.txt | 6,129 | 10,000 | 0.013 | 1.592 |
| alice.txt | 28,196 | 100 | 0.024 | 0.135 |
| alice.txt | 28,196 | 1,000 | 0.029 | 0.840 |
| alice.txt | 28,196 | 10,000 | 0.026 | 6.712 |
| kjv10.txt | 823,135 | 100 | 0.169 | 2.168 |
| kjv10.txt | 823,135 | 1,000 | 0.158 | 20.165 |
| kjv10.txt | 823,135 | 10,000 | 0.148 | 205.645 |
| shakespeare.txt | 901,325 | 100 | 0.175 | 2.236 |
| shakespeare.txt | 901,325 | 1,000 | 0.165 | 23.086 |
| shakespeare.txt | 901,325 | 10,000 | 0.173 | 214.276 |

Experimenting with different values of T and N, it can be found that the empirical data conforms to my expectations for the runtime complexity of getRandomText(). When N is held constant, there is a linear relationship between T and generating time. For example, when N = 10,000, as T increases by a factor 4.6 (6,129 🡪 28,196), generating time increases by a factor of 4.2 (1.592 🡪 6.712), and as T increases by a factor of 29 (28,196 🡪 823,135), generating time increases by a factor of 30 (6.712 🡪 205.645). When T is held constant, there is also a linear relationship between N and generating time. For example, when T = 901,325, as N increases by a factor of 10 (100 🡪 1,000), generating time increases by a factor of 10 (2.23 🡪 23.09), and as N increases by a factor of 100 (1,000 🡪 10,000), generating time increases by a factor of 100 (23.09 🡪 214.28). Combining these effects, when T increases by a factor of t and N increases by a factor of n, the generating time is shown to increase by a factor of nt, conforming to our expectations.

**Question 2**

The asymptotic (big O) runtime complexity of the setTraining() method for the HashMarkov implementation is expected to be O(T). First, the split() method which splits the training text into separate words and adds each one to the array myWords. As T is the number of words in the training text, this addition happens T times, so it has a runtime complexity of O(T). Then, the for loop used iterates for a total of (T – order) times, so its big O will be O(T), dropping the order since it is a constant. Inside the for loop, the put(), get(), and add() methods are all constant time operations. Hence, the final big O would be O(T+T) = O(2T), which can be simplified as O(T) for runtime complexity.

The asymptotic (big O) runtime complexity of the getRandomText() method for the BaseMarkov implementation is expected to be O(N). As we are given that nextInt() is a constant time operation, the first four rows of getRandomText() is constant time as they are generations of new array lists, integers, etc. The for loop, then, iterates for (N – order) times, so its runtime complexity would be O(N) dropping the constant order. The getFollows() method in the loop, this time, is more efficient and has constant time. Hence, the final big O is O(N).

Experiment:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data file** | **T** | **N** | **Training Time (s)** | **Generating time (s)** |
| biden-2021.txt | 6,129 | 100 | 0.047 | 0.001 |
| biden-2021.txt | 6,129 | 1,000 | 0.037 | 0.005 |
| biden-2021.txt | 6,129 | 10,000 | 0.036 | 0.016 |
| alice.txt | 28,196 | 100 | 0.091 | 0.001 |
| alice.txt | 28,196 | 1,000 | 0.098 | 0.014 |
| alice.txt | 28,196 | 10,000 | 0.093 | 0.033 |
| kjv10.txt | 823,135 | 100 | 0.598 | 0.003 |
| kjv10.txt | 823,135 | 1,000 | 0.563 | 0.022 |
| kjv10.txt | 823,135 | 10,000 | 0.598 | 0.016 |
| shakespeare.txt | 901,325 | 100 | 0.773 | 0.003 |
| shakespeare.txt | 901,325 | 1,000 | 0.775 | 0.008 |
| shakespeare.txt | 901,325 | 10,000 | 0.772 | 0.028 |

Experimenting with different values of T and N, it can be found that the empirical data conforms to our expectations as the generating time is significantly more efficient here than in BaseMarkov, all taking less than 0.1 seconds even when T and N reach large values. However, the trend doesn’t exactly conform to our expected linear trend in the runtime complexity of getRandomText(). While an increasing trend is shown in generating time as N increases, the relationship is not exactly linear. For instance, when T is held constant as 901,325, as N increases by a factor of 10 (100 🡪 1,000), generating time increases by a factor of 2.67 (0.003 🡪 0.008), and when N increases by a factor of 100 (1,000 🡪 10,000), generating time increases by a factor of 3.5 (0.008 🡪 0.028). This discrepancy could be due to our small sample size or the lack of data points to display a more accurate trend.

**Question 3**

As OpenAI’s mission states, there would clearly be advantages in the development of AI/ML as it enhances efficiency and can bring humanity to a state of greater convenience. One of the most commonly brought up problems in this idea of “AI taking over” is the gradual decrease in the job market with technology performing economically valuable work over humans. I believe the greater problem in the domination of artificial intelligence and especially with language, however, lies in bias that is inherently baked into the system. The algorithms personalized for users, as reflections of their search histories, seem to enhance our knowledge on areas of our interest. However, as users navigating through this algorithm, we access and absorb knowledge in a very passive manner filtered by the “highly autonomous systems” that pick out what information to present. I think this has a risk of inflicting the unification of thought upon users of technology — a great majority of the world’s population — and affecting people’s views of the world.

This is where the problem with sources used comes in; what AI/ML generates is based on the sources they are given. This means that a bias in the selection of sources will lead to a result that shadows this bias, whether it be political, social, ethical, etc. With GPT-3 not being open-source, users do not know what kind of information is being inputted by the programmers, meaning that the developers of this software hold ultimate power in choosing what sources to use based on their discretion. I think this is one of the greatest risks non-open sources hold and believe that more open-source codes are necessary in ensuring that bias is reduced by being reviewed by more diverse groups of people.