1.
a.
.setTraining()
Runtime: O(T) — in terms of T, O(1) — in terms of N

Explanation:
Theory: .setTraining() iterates through the entire text file and splits every String at spaces (“ “) and line breaks and adds each String to an index of the array myWords. This is O(T) because the assignment of elements in an array is constant time and the .split() method is O(T). .setTraining has no contact with the generated string of random words so, its runtime in terms of N has no bearing on the method.

Experiment: For each text tested, the training times are all uniform to the size T of the training text. Thus O(T) makes sense. The change in N has no effect on the training times also corroborating my claim that N has no effect on runtime in .setTraining().

b.
.getRandomText()
Runtime: O(N*T) - in terms of N and T

Explanation:
Theory: Creating an array list is constant time. Creating an int using .nextInt is constant time. Creating a WordGram object is constant time. Adding a singular element to an array list is constant time. Looping over length of the generated word (N) is O(N) runtime. Creating a string using .getNextWord() is O(T) runtime because this corresponds to the getFollows method which runs through the length of the array myWords which corresponds to every word in the text file. The .equals method is constant time. The break operation is constant time. Adding to the array list randomWords is constant time because the array list only adds words that follow the WordGram and thus doesn’t add every element of T or N. The .shftAdd method is constant time because the order of the array list is set to 2 per the MarkovDriver class. The .join method is constant time.

Thus the internals of the loop have a runtime of O(T). Since O(T) is inside the loop that runs N times the total runtime of .getRandomText() is O(N*T)

Empirical: The empirical data reflects my hypothesis that .getRandomText() has a runtime in terms of N of O(N *T). This is because the generating times have an increasing curve. The O(T) runtime is evident because in the 3 text files run, the larger the T, the larger the generating times were.

Data File	T	N (TEXT_SIZE)	Training Time(s)	Generating Time(s)
alice.txt	28196	100	0.013s	0.109s
alice.txt	28196	100,000	0.014s	1.342s
alice.txt	28196	1,000,000	0.014s	1.335s
trump-sou17.txt	5190	100	0.005s	0.044s
trump-sou17.txt	5190	100,000	0.005s	0.188s
trump-sou17.txt	5190	1,000,000	0.005s	0.205s

shakespeare.txt	901325	10	0.117s	0.223s
shakespeare.txt	901325	100	0.118s	1.773s
shakespeare.txt	901325	1000	0.119s	14.962s

2.
a.
.setTraining()
Runtime: O(T) — in terms of T, O(1) — in terms of N

Explanation:
Theory: The creating of myWords using the .split method is O(T) runtime because the method runs through the entire length of the text file. The .clear method is constant time. The for loop runs through every element of myWords, starting at myOrder which is equal to the number of words in the text file so it has a runtime of O(T). The initializing of a WordGram is constant time. The initializing of a String and accessing of an array element is constant time. The .containsKey method of a Map is constant time. The .get and .add methods for Maps are constant time. The .put method for Maps is constant time. The .shiftAdd method is constant time because it has a preset length of 2 as seen in the MarkovDriver class.

Experiment: The training times reflect the size of T (if T is bigger, the training time is bigger). The change in N has no bearing on Training Time.

b.
.getRandomText()
Runtime: O(N)

Explanation:
Theory: Similar to BaseMarkov, inside the body of a for loop of runtime O(N), the method .getNextWord is called. However, in HashMarkov, the .getNextWord method and subsequent .getFollows() method use HashMap operations which are constant time. Thus the total runtime of the method is just O(N).

Every other operation within .getRandText() is the same as with BaseMarkov.

Experiment: The empirical evidence displays a massive increase in efficiency and thus decrease in runtime compared to BaseMarkov. The generating times show no effect by the increase in T. The increase in N indicates a slight increase in the generating time. This matches a total runtime of O(N).

Data File	T	N	Training Time(s)	Generating Time(s)
alice.txt	28196	100	0.052s	0.001s
alice.txt	28196	100,000	0.054s	0.006s
alice.txt	28196	1,000,0000	0.052s	0.007s

trump-sou17.txt	5190	100	0.022s	0.000s
trump-sou17.txt	5190	100,000	0.024s	0.008s
trump-sou17.txt	5190	1,000,000	0.025s	0.006s
shakespeare.txt	901325	10	0.608s	0.005s
shakespeare.txt	901325	100	0.585s	0.003s
shakespeare.txt	901325	1000	0.569s	0.008s

3.

While OpenAI's mission statement portrays a dystopian tone, I think that their motives and the end goal behind AI is ultimately a positive thing for humanity. Our society has progressed through history because of our curiosity and innate intent to innovate. Ai represents the next frontier of human discovery that needs to be explored. As with all innovation, not every person is able to adapt to new tools and thus they lose some of their relevance in society, but if we were to stop all innovation for the sake of the slow adopters our society would be at a far greater risk of extinction. An emphasis on the future will always be more in important for our survival than looking to the past.

I think that OpenAi is entitled to a certain degree of corporate secrecy and thus in these developmental stages has the right to keep their source code protected. However, in the event that Ai were to gain sentient awareness or the ML algorithms were to progress to the level of replacing human critical thinking, an argument needs to be made for open sourcing AI programs. When new innovation begins to threaten the livelihood a majority of society, it dissolves a company's right to trade secrets and becomes an issue of human survival. In conclusion, OpenAI has the right to hide their source code within reason but should not be hesitant to share their code when Ai reaches a certain level of awareness.