Big-Oh Analysis

Morse Code converted encode and decode methods

Encode() method:

The encode method has a bigOh of o(n2), so it’s relatively efficient from a bigOh standpoint. The method contains 3 nested for loops, one of which will only ever run a maximum of 26 times. So that loop run o(26) times. Then I have two embedded loops, one which loops through the entire English sentence and then one that loops through each word in the English sentence. Both of the loops run o(n) times, which means that the whole method will run o(26) \* o(n) \* o(n) times. This will reduce to o(26n2), which is then further reduced to just o(n2). This whole method could easily be cut down to have a bigOh of o(n), but it wouldn’t actually reduce the amount of characters I need to loop through, but would just order them differently. On top of these two loops however, the encode method will also use the visit method from the CodeVisitor class. The visit method contains no loops, however it will call itself recursively. Because of this, it will still have a bigOh of o(n). That means we now have a method that’s o(n2) \* o(n), which will give us o(n3). Again though, this can be simply reduced to o(n2) by dropping the loop through a single word and replace it with some simple logic.

Decode() method

The decode method will also have an o(n2) runtime efficiency. While it also has a third embedded loop in there, the worst case scenario for that loop is that it has to run 4 times, since it’s looping through each symbol making up the Morse code character. Similar to the encode method, the decode method first will loop through the sentence, hitting each word individually. This loop will run o(n) times, where n is the number of words in the sentence. The next loop will look through each letter in the Morse code word. It will also run o(n) times, where n is the number of consecutive characters without having a double space between them, double space being the delimiter for each word. I will then loop through each symbol making up the Morse code character, which will be at max 4, so this loop will run o(4) times. This means the whole algorithm will run o(n) \* o(n) \* o(4) times, making it o(4n2). This will then be reduced to o(n2) since we can treat the 4 as being negligible. Similarly, to the encode method, this method could relatively easily be reduced to o(n) simply by removing the loop through each word and replacing it with a little bit of logic to check if the next two characters will be spaces, delimiting a word. While this method could be reduced, again like the encode method, it wouldn’t really change anything since the method is simply looping through a sentence, and whether you do that character by character or word by word, doesn’t really matter in the end since it’s the same amount of input into the method.