1. Explain how you approached creating tests for each of your functions. For each function, this explanation should include what features you decided to test and how you created tests for those features. Discuss edge cases, common cases, and similar such concepts if relevant.

For creating all four unit testing functions for each DNA processing function, we've approached it by structuring the test flow to go through the unit test based on common cases and then the unit test based on edge cases.

For the readingFrames(dna, frameNum) function, we decided to test the feature of all three reading frames that return a list of length-3 strings representing all the codons of the chosen reading frame in the order that the codons appear. We approached it by structuring the common cases based on the DNA string "aatcgtagctt" by testing three different reading frames. The edge cases were based on the case when an empty DNA string was passed and the cases when there were too short lengths of DNA string were passed for a specific reading frame. For example, passing "aa" as a DNA string and frameNum as 0 should return an empty list because at least there should be three nucleotides for this specific reading frame but the given DNA string has only two nucleotides.

For the nextORF(dna) function, we decided to test the feature for returning two integers, where the first one represents the index of the "a" in the earliest start codon, and the second one is the index of the nucleotide just after the stop codon. We approached it by structuring the common cases based on the DNA string "atgaatcgtagctt", "cccatggcccggtgagatga", "acccatggcccggtgagatga", and "aacccatggcccggtgagactgattt". The edge cases were based on the case when an empty DNA string was passed, the case when only one nucleotide was passed, and the cases when only the start codon or one of the stop codons was passed.

For the potentialGenes(dna, size) function, we decided to test the feature for returning an integer counting the number of ORFs of at least that size contained in the DNA, where size is counted by the number of codons. We approached it by structuring the common cases based on the following pairs.

1. "cccatggcccggtgagactgattt", 2
2. "cccatggcccggtgagatga", 2
3. "atgaatcgtagctt", 2
4. "atgtaacccatgtaaccc", 2
5. "cccatgtaacccatgtaacccatgccctga", 2

The edge cases were based on the case when an empty DNA string was passed, the case when only one nucleotide was passed, the case when the minimum number of codons to consider in an

ORF is bigger than the number of codons in the given DNA string, and the case when there are existing ORFs but they don't match the minimum number of codons.

For the longestMatch(dna1, dna2) function, we decided to test the feature for returning a string representing the longest DNA sequence that is common to both arguments, ensuring that if more than one shared sequence is of the longest length, then the function returns the one that occurs earliest in the argument dna1. We approached it by structuring the common cases based on the following DNA string pairs.

1. "attattagccgc", "attattccgccgcc"
2. "attattccgccgcc", "attattagccgc"
3. "attatt", "att"
4. "attattgccgcc", "gccgccattatt"
5. "gctataa", "tagcaa"
6. "tagcaa", "gctataa"
7. "gctaaa", "tagcaa"

Especially, common cases 4, 5, 6, and 7 were specifically designed to test the feature that if more than one shared sequence is of the longest length, then the function returns the one that occurs earliest in the argument dna1.

The edge cases were based on the cases where one or both DNA strings were empty, the cases where only one nucleotide was passed to either dna1 or dna2 or both dna1 and dna2 and the cases where there was no common match.

2. Explain how you ended up using your unit tests. For example, did you you write your tests before, alongside, or after writing your functions? Did your tests help you uncover any bugs?

We ended up using our unit tests to apply additional code for possible edge cases. We constructed our unit tests while writing all four functions, and this led us to uncover bugs that were being thrown, which was helpful in improving the performance of our functions.

3. Explain how you worked together as a group. For example: When did you work together? How did you share code? Who took the lead (if anyone)? What worked well in your group dynamics and what did not?

We shared our code files via the Google Drive folder and worked together with sharing the status by sending DMs. Daniel (Chanhui) has taken the lead on this project. Overall process for this project worked well except that during the work for this project there was a moment when our

team misunderstood the existence of the precondition for this project in Section 1.2 of the writeup.