Agreement form for CS take-home midterm

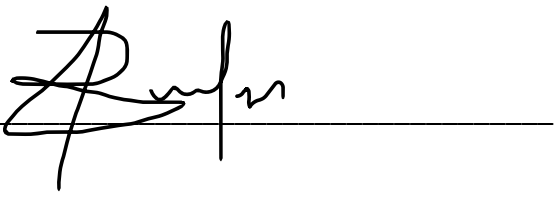
I understand that the take-home midterm in Bioinformatics is intended to test my own personal knowledge of and ability to apply concepts, methods, and factual information from the course. I therefore agree to do **ALL** of the following:

1. Complete all problems **on my own**, without discussing them with any other person (other than Dr. Weisstein).

2. Use **only the following resources:**
   - course PowerPoint slides and your own lecture notes.
   - Codecademy lessons assigned as part of this course.
   - any code and documentation that you or a member of your own team produced for an earlier assignment in this course.
   - any other resource that has been approved *in writing* by Dr. Weisstein.

3. **Report any evidence** that other students are violating or seeking to violate these terms.

I understand that failure to uphold these terms constitutes a serious violation of Truman's Academic Integrity policy and will be treated accordingly.

Signature:  Date: 10/18/25

2. In class, we discussed the difference between ***transitions*** and ***transversions*** as two common types of mutation. Bioinformatics researchers have found that transitions typically occur more often than transversions, and calculate the ***transversion ratio*** to quantify this disparity for a particular data set (e.g., a portion of a specific gene).

   Purine: A, G pyrimidine: C, T

   Transversion: purine->pyrimidine, pyrimidine-> purine

   Transition: purine -> purine, pyrimidine -> pyrimidine

a. The file *CS Midterm Sequences.docx* includes five short DNA sequences. Assume that all five of these sequences are descended from a single ancestral sequence. Begin by considering only the nucleotides observed at the **first** position in each sequence. What is the smallest possible number v_1 of transversions that could have occurred at this position?. **Briefly but clearly explain your reasoning.**

   Nucleotides in 1st position for:

   Alpha: C

   Bravo: A

   Charlie: C

   Delta: C

   Echo: T

   In the first position, there is 1 purine and 4 pyrimidine. If we assume that there were less number of mutation that a nucleotide went through too, then the ancestral nucleotide at the first position should be the most common one which is C.

   Then, C-> A mutation is transversion.

   So, there is at least one transversion. So, the smallest possible number for v1 of transversions is 1.

b. Given your answer to part (a), what is the smallest possible number s_1 of transitions that could have occurred at the first position? (See the sample calculations on the next page.) Again, **briefly explain your reasoning.**

   Nucleotides in 1st position for:
   Alpha: C
   Bravo: A
   Charlie: C
   Delta: C
   Echo: T
   In the first position, there is 1 purine and 4 pyrimidine. If we assume that there were less number of mutation that a nucleotide went through too, then the ancestral nucleotide at the first position should be the most common one which is C.
   Then, C -> T mutation is transition.

So, there occurred at least one transition. So, the smallest possible number for s1 of transition is 1.

c. Apply the two steps above to each of the remaining positions in turn. Use these results to estimate the **total** number *V* of transversions, and the total number *S* of transitions, that have occurred in these sequences ($V = v_1 + v_2 + \ldots + v_k$, $S = s_1 + s_2 + \ldots + s_k$, where *k* is the length of each sequence). Then calculate the transversion ratio as *V* / *S*. Include your answers to parts (a) – (c) in your Word document.

	Position									
Sequence	1	2	3	4	5	6	7	8	9	10
Alpha	C	G	T	A	G	T	A	A	G	T
Bravo	A	G	A	A	T	T	G	A	A	T
Charlie	C	G	T	A	T	T	T	A	A	C
Delta	C	A	T	A	C	T	G	A	A	T
Echo	T	G	A	A	C	T	C	A	G	A

For first position, v1=s1=1

For second position, the most common nucleotide is G, so we can assume that ancestral DNA also had G in this position. G->A mutation is transition, and there are no transversion in this position, so v2 = 0, s2 = 1.

For third position, the common nucleotide is T, so we can assume that ancestral DNA also had T in this position. T->A mutation is transversion, and there are no transition in this position, so v3 = 1, s3 = 0.

For forth position, the common and only nucleotide is A, so we can assume that ancestral DNA also had A in this position. There are no other nucleotide in this position so we can assume that there were no mutations, so v4 = s4 = 0.

For fifth position, there are 4 pyrimidine and 1 purine present. So, we can assume that there was at least 1 transition and at least one transversion in this position, so the smallest number of transversion and transition is 1, so v5 = 1, s5 = 1.

For sixth position, the most common and only nucleotide is T, so we can assume that ancestral DNA also had T in this position. There is no other nucleotide in this position so we can assume that there were no mutations, so v6 = s6 = 0.

For seventh position, there are 3 pyrimidine and 2 purine present. So, we can assume that there was at least 1 transition and at least 1 transversion in this position, so the smallest number of transversion and transition is 1, so v7 = 1, s7 = 1.

For eighth position, the common and only nucleotide is A, so we can assume that ancestral DNA also had A in this position. There are no other nucleotide in this position so we can assume that there were no mutations, so v8 = s8 = 0.

For ninth position, there are only purine bases present, so we can assume that ancestral DNA also had purine in this position for smallest number. G->A or A->G mutation is transition, and there are no transversion in this position, so v9 = 0, s9 = 1.

For tenth position, there are 4 pyrimidine and 1 purine present. So, we can assume that there was at least 1 transition and at least 1 transversion in this position, so the smallest number of transversion and transition is 1, so v10 = 1, s10 = 1.

V = v1+v2+v3+v4+v5+v6+v7+v8+v9+v10 = 1 + 0 + 1 + 0 + 1 + 0 + 1 + 0 + 0 + 1 = 5

S = s1+s2+s3+s4+s5+s6+s7+s8+s9+s10 = 1 + 1 + 0 + 0 + 1 + 0 + 1 + 0 + 1 + 1 = 6

So, V/S = 5/6

2. In class, we have discussed several common bioinformatics tasks that involve comparing short sequences to find areas of similarity. Many other bioinformatic tasks involve comparing much longer sequences (often >100 kb), and/or comparing a reference sequence against **all** sequences in a database that may contain millions of individual records.

Explain why the nature of biological data complicates standard methods for optimizing code (e.g., for runtime and/or memory allocation) for such tasks. Relate your answer to specific features of cases we have discussed in class (e.g., locating splice site consensus sequences at the beginning and end of introns, searching for common promoter elements, etc.).

DNA and protein sequences are long, irregular, and inconsistent in structure. In prokaryotes, there is presence of two consensus sequence known as -35 cs and -10 cs. These help to identify the promoter, however each actual sequence may differ by 1-2 nucleotides from the consensus sequence. The number of base pairs between consensus sequences and the transcription start site may vary, or consensus sequence may be absent as well. This difference makes it difficult for algorithms to rely on exact matches or fixed distances.

For eukaryotes, the promotor contains different set of consensus sequence. The most common ones are TATA box, GC box, CAAT box, and Octamer box, however there are even more than those set of consensus sequence, even though they are in lower frequency. The consensus sequences occur 20-100 base pairs upstream of the transcription start site and in the non-template strand. Their occurrence, position, and conservation vary a lot so it further complicates standard method for optimizing code. It is because optimized code is only going to look for similar or same patterns, however the code will need to have, let's say, 5 different functions just for it to locate consensus sequence, and promoter and terminator after that.

We will also be scanning through large sequence, even containing a larger number of introns for short(6-8 base pairs) motifs. In the large sequence, the motifs may even appear in different forms than defined which makes it harder for algorithm to algorithms to identify or even sometimes disregard the mismatches. So, as a result, optimizing runtime and memory is difficult because the code needs to be flexible for finding out exact match across large and variable dataset.