CS313 Exercise 2 Cover Page

Name(s): _____

In the *TIME* column, please estimate the time you spent on the parts of this exercise. Please try to be as accurate as possible; this information will help us to design future exercises.

PART	TIME	SCORE
Exercise		

Task 1: Searching for TATA boxes

Many eukaryotic genes contain a DNA sequence, called a TATA box, in their promoter region, i.e., just upstream of their start codon. The TATA box is so named because it often consists of the following hexamer (i.e., 6 nucleotide) motif: "TATAAA". Consider the following DNA sequence:

`AATTTATATATACAAATG`

1) How many TATA boxes does the sequence contain?

2) How many TATA boxes does the sequence on the opposite strand of the genome (i.e., the reverse complement of the above sequence) contain?

Rather than a TATA box, some eukaryotic genes contain a degenerate version of the TATA box in their promoter region. A degenerate TATA box motif is similar to a non-degenerate TATA box motif, except that the fifth nucleotide is not an adenine, i.e., `TATACA`, `TATAGA`, and `TATATA` are degenerate TATA box motifs.

3) How many degenerate TATA boxes does the sequence contain?

4) How many degenerate TATA boxes does the sequence on the opposite strand of the genome contain?

Task 2: PAM and BLOSUM protein substitution matrices

Both **PAM and BLOSUM substitution matrices** are derived from empirical amino acid substitution data and contain scores that represent the likelihood that any particular amino acid will mutate to another specific amino acid in a protein. However, there are several important differences between PAM and BLOSUM models. For example, many more sequences were analyzed for BLOSUM matrices than for PAM matrices. Another important difference is that each BLOSUM matrix is calculated independently, while PAM matrices for analyzing divergent sequences are extrapolated by repeated self-multiplication of the original PAM1 matrix. Finally, BLOSUM matrix calculations use clustering methodology to limit the contributions of large sets of highly similar sequences, while PAM matrices do not.

One similarity between PAM and BLOSUM substitution matrices is that they are both **log odds matrices**; the substitution score is the logarithm of the specific mutation probability (the probability that amino acid *X* will mutate to amino acid *Y* in a non-random way) divided by the probability that amino acid *Y* is substituted at random (the mean frequency of amino acid *Y* in proteins).

1) Why is it important to consider the likelihood that a substitution occurred by chance when calculating substitution matrix scores using log odds methodology?

2) Log odds matrices contain integer values that are positive, negative, or zero. What does a larger PAM and BLOSUM matrix score signify relative to a lower score?

3) BLOSUM matrices were generated from alignments of diverse proteins, while PAM matrices were generated from alignments of closely related proteins. How does this impact the versatility of these scoring matrices?

4) The BLOSUM matrices were derived using many more proteins than the original PAM matrices. Why is the number of sequences used to generate these empirical models an important factor in substitution matrix accuracy / reliability?

5) In order to generate PAM matrices for use with more divergent sequences, the PAM1 matrix is multiplied by itself many times. Why is this a concern, particularly with matrices generated from relatively limited data sets?

6) When calculating BLOSUM substitution matrices, sequences that share similarity above a threshold are clustered together, and the contribution of the sequences in these clusters is limited by lightly weighting their contribution to the model. What is the reason that BLOSUM matrices use clustering methodologies?

Task 3: Jukes-Cantor correction

In this task, you will generate a random DNA sequence and then repeatedly (1000 times) mutate one of the nucleotides in the sequence. During the 1000 times that you mutate a nucleotide in the sequence, you may at times mutate a nucleotide that has been mutated previously and you may at times mutate a nucleotide that has not been mutated previously. After 1000 iterations, you will have generated a mutated sequence that may look quite different from the original sequence. The distance, *p*, between the two sequences (the original sequence and the mutated sequence) is the number of nucleotides that differ between the two sequences. Since you mutated 1000 nucleotides, but some of those 1000 mutations may have occurred on the same nucleotide, the distance *p* between the two sequences will likely be less than 1000.

Suppose the distance, *p*, between the original sequence and the mutated sequence is 600, i.e., 600 nucleotides differ between the two sequences. *If we did not know that the mutated sequence was generated from 1000 mutations to the original sequence*, how might we estimate the number of mutations that yielded a mutated sequence with a distance of *p*=600 from an original sequence? The *Jukes-Cantor correction* is a means for estimating the number of *actual* mutations that have occurred between two sequences when we only know the distance, i.e., the *observed* number of mutations, between the two sequences.

To begin, download the `/home/cs313/download/Mutagenesis` directory from the `CS` server. Study the `Mutagenesis.java` file in the `Mutagenesis` directory. There are four method skeletons in the file that you must fill in.

```
/* The "mutateOneNucleotide" method takes a genomic sequence as a parameter
 * and returns a new genomic sequence, the same as the input sequence,
 * except a single randomly chosen nucleotide from the sequence is changed
 * to a different nucleotide.
 */
public static String mutateOneNucleotide(String s);


/* Takes two sequences, s1 and s2, of the same length and
 * returns the number of nucleotides that differ between the two sequences.
 * For instance, if s1="ACCGTGCTA" and s2="GCCGAGCCA" then the method
 * should return the number 3 since s1 and s2 contain different
 * nucleotides at indices 0, 4, and 7.
 */
public static int distanceBetweenSequences(String s1, String s2);


/* Given the *observed* distance, p, between two sequences, and
 * the length, "length", of the sequences, this method
 * applies the Jukes-Cantor correction in order to estimate the *actual*
 * distance between the two sequences. The actual distance, K, can be
 * estimated using the Jukes-Cantor correction as:
 *               K = -3.0/4.0 * (length) * ln(1.0-(4.0/3.0)*p/length)
 * where "ln" refers to the natural logarithm.
 */
public static double JukesCantorCorrection(int p, int length);
```

```
/* Takes a genomic sequence, "seq", and mutates randomly chosen
 * nucleotides in the sequence 1000 times. After each of the
 * 1000 mutations, the method prints out two numbers per line:
 *   (1) p = the distance between the mutated sequence and the
 *       original sequence, i.e., the observed number of mutations
 *   (2) K = the Jukes-Cantor correction to p, i.e., the estimated
 *       number of actual mutations that occurred between the two sequences.
 * For example, the first 10 (of 1000) lines printed out might look as follows:
 * 1        1.00066725985
 * 2        2.00267141691
 * 3        3.00601604815
 * 4        4.01070474495
 * 5        5.0167411131
 * 6        6.02412877295
 * 7        7.03287135945
 * 7        7.03287135945
 * 8        8.04297252223
 * 9        9.0544359257
 */
public static void mutagenesis(String seq);
```

When you have correctly implemented the above four methods, the `Mutagenesis` application should print out 1000 lines, each containing 2 numbers. Now store these 1000 lines of output in a text file, e.g., `JukesCantor.txt`. Using a spreadsheet or graphing program, you should then generate two line graphs, one for each of these two columns of 1000 numbers. One line will represent the observed number of mutations between two sequences as a function of the actual number of mutations between two sequences. The other line will represent the estimated number of actual mutations between two sequences as calculated (using the Jukes-Cantor correction) from the observed number of mutations between two sequences.

1) Based on studying your line graphs, how well do you think the Jukes-Cantor correction works?

2) Do you think the Jukes-Cantor correction is more accurate at estimating the number of actual mutations for pairs of sequences that are closely related or for pairs of sequences that are highly divergent?

3) When submitting this exercise, please include a hardcopy of your modified `Mutagenesis.java` file as well as your line graphs.