

CSC / BIO 310

Bioinformatics

Instructor: Dr. Laurie J. Heyer

Review I

Due Thursday, Feb 14, at 11:30 a.m.

Instructions:

VERY IMPORTANT: Once you read beyond page 1 in this file, you may not get help with Perl from ANYONE. I will answer questions up until the time you read page 2, but not afterward. Read these instructions carefully before you continue.

You may use any openly available inanimate resources to help you with this review.

However, you may not use code from the internet, you may not request a copy of already written code from anyone, and you may not discuss any aspect of the review with anyone other than me.

Take special care to protect your work so others do not accidentally find it laying around in the lab, up on a computer screen somewhere, or on the whiteboard.

You have unlimited time to work on this review, in any number of sittings.

In addition to the accuracy of your solutions, you will be graded on the readability of your code and your output, and the use of good programming practices as discussed in the text and in class.

Page two will specify exactly what you need to turn in to me by email. Failure to follow those instructions will incur a penalty. Late penalty of 10 points per hour.

Email to me the following three files:

- (1) A Perl file called *name_r1_compare.pl*, where *name* is replaced with your name, containing your solutions to problems 1-4
- (2) A Word document containing your answers to questions 5 and 6
- (3) A Perl file called *name_r1_iupac.pl*, where *name* is replaced with your name, containing your solutions to problem 7

If you have trouble getting code to work, put lots of comments in to explain what you are trying to do, so I can give you some partial credit.

1. Write a subroutine that generates a random DNA sequence of a desired length and probability distribution. The subroutine should accept the sequence length and the probability of an “A” occurring in any position, and return a string containing the random DNA sequence.
2. Write a subroutine that compares two strings of the same length. The subroutine should accept the two strings as arguments and return the number of positions in the two strings at which the corresponding letters are identical. For example, if the two input strings are ACAGAT and CCTGAC, the subroutine would return the value 3, since the strings match at 3 positions (2, 4, and 5).
3. Write a program that makes 1000 comparisons of pairs of randomly generated DNA sequences, where each sequence has length 100. Use your subroutine from #1 to generate the sequences and your subroutine from #2 to make each comparison. Keep track of the smallest and largest comparison scores among the 1000 comparisons, and compute the average comparison score over the 1000 comparisons. Print these three values. Do not print the individual sequences or scores.
4. Devise an alternative method for comparing two strings as described in #2, using the `tr` command and subtraction. If your answer to #2 already uses the `tr` command, then devise an alternative method that does not use `tr`. Write a subroutine that implements this alternative comparison method, using the same specifications as the subroutine in #2.
5. Discuss the relative advantages and disadvantages of the two subroutines you wrote in #2 and #4 to compare sequences. Which one is better? Why?
6. For each of the following regular expressions, describe in words the pattern that is found by the regex. List the number of occurrences of the pattern in the *E. coli* genome, and of these occurrences, how many have length greater than 8.
 - a. `'ACGG(A|T)\1*`
 - b. `'ACGG[AT]{4,8}'`
 - c. `'(.{12,20})\1+'`

7. Perform an Internet search to find information on IUPAC degeneracy code for DNA sequences. Write a Perl subroutine that converts an IUPAC formatted string into a regular expression that can be used to find all matches to the IUPAC string. The subroutine should accept the IUPAC string as an argument, and return a string containing the regular expression. Write a Perl program that calls this subroutine with the IUPAC string BADCANDYCAT to create the equivalent regular expression, and then prints all matches to this IUPAC string found in the *E. coli* genome.