Fall 2013 **CMSC 423: MidTerm2** H. Corrada Bravo
Time: 1 Hour, 15 Minutes

WAIT FOR INSTRUCTIONS BEFORE BEGINNING

HONOR PLEDGE: "I pledge on my honor that I have not given or received any unauthorized assistance on this examination."

Signature and UID: ______________________________

Print name: ______________________________

- ***Write your answers with enough detail about your approach and concepts used, so that the grader will be able to understand it easily.***
- ***The sum of the grades is 85, but your grades would be out of 80 (thus you get 5 bonus points by solving all the problems).***
- ***Select the best choice for the first 5 problems and mark it by X in the table below.***

Problem	1	2	3	4	5
A					
B			X		
C				X	
D		X			X
E	X				

DO NOT WRITE BELOW THIS LINE

Problems 1-5:		Problem 10:	
Problem 6-8:			
Problem 9:		**Total:**	

Multiple-choice Problems (Answer THE BEST CHOICE in the Table of the First Page and NOT HERE):

Problem 1. **(2 points)** Which of these best represents the relationship between genotype and phenotype?

a) there is **no** relationship between genotype and phenotype
b) an individual's phenotype **completely** determines their genotype
c) an individual's genotype **completely** determines their phenotype
d) an individual's phenotype **partially** determines their genotype
e) none of the above

Problem 2. **(3 points)** What is an *open reading frame* (ORF)?

a) any translatable sequence of nucleotides

b) any sequence of codons

c) a long enough sequence of aminoacids

d) a long enough sequence of codons without an intervening stop codon

e) None of the above

Problem 3. (5 points) Which of the following statements best represents complexity of using suffix arrays for exact string matching of query string Q to target string T

a) preprocessing: $O(|Q|)$, space: $O(|Q|)$, search: $O(|T|)$

b) preprocessing: $O(|T|)$, space: $O(|T|)$, search: $O(|Q| * \log |T|)$

c) preprocessing: $O(|T| * \log |T|)$, space $O(|T|)$, search $O(|Q| * \log |T|)$

d) preprocessing: $O(|T| + |Q|)$, space $O(|T|)$, search $O(|Q| * |T|)$

e) None of the above

Problem 4. (5 points) Assume I want to use a Bayesian formulation to compute the *posterior* probability P(M1|x,y) that sequences x (query) and y (target in a large database D) are related by, say, evolution (model M1), rather than chance (model M2). It requires ratio P(M1)/P(M2) is given. How should this ratio be set:

a) proportional to |D|

b) independent of |D|

c) proportional to 1/|D|

d) proportional to |x| * |y|

e) none of the above

Problem 5. (5 points) Consider the recurrence relations for global alignment with affine gap penalties below. How many mistakes are there? (Assume the cost of a gap of length g is open + g * extend).

$$\boldsymbol{M(i,j) = cost(x_i, y_j) + min \begin{cases} M(i-1, j-1) \\ I_x(i, j-1) \\ I_y(i-1, j) \end{cases}}$$

$$\boldsymbol{I_x(i,j) = min \begin{cases} open + extend + M(i, j-1) \\ extend + I_x(i, j-1) \\ extend + I_y(i, j-1) \end{cases}}$$

$$\boldsymbol{I_y(i,j) = min \begin{cases} open + extend + M(i-1, j) \\ extend + I_x(i-1, j) \\ extend + I_y(i-1, j) \end{cases}}$$

a) None b) One c) Two d) Four e)Nine

Short Questions (show all derivations as appropriate for full credit):

Problem 6. (10 points) You are trying to find the placement of sequence queries (reads) of length 100 along the human genome (3 Gbp), allowing for at most 4 mis-matches, however you do not want to use Smith-Waterman directly. Instead you will use an exact matching algorithm to find good candidate matches then extend these with Smith-Waterman. What is the length of the exact match seeds (substrings of the query) you will use in order to guarantee you that no correct alignments are missed? Explain.

You can use this scheme by splitting reads into 100/5=20 bp seeds and perform exact matching on each seed. If the 100bp sequence matches a location with at most 4 mismatches then at least one of the five seeds is an exact match in that location.

Problem 7. (10 points) Construct the suffix array for string RATATATRA and trace how binary search is used to find all matches of query TAT.

RATATATRA$

1. $	1. $	1. $
2. A$	2. A$	2. A$
3. ATATATRA$	3. ATATATRA$	3. ATATATRA$
4. ATATRA$	4. ATATRA$	4. ATATRA$
5. ATRA$	5. ATRA$	5. ATRA$
→ 6. RA$	6. RA$	6. RA$
7. RATATATRA$	7. RATATATRA$	7. RATATATRA$
8. TATATRA$	→ 8. TATATRA$	→ 8. TATATRA$
9. TATRA$	9. TATRA$	→ 9. TATRA$
→ 10. TRA$	→ 10. TRA$	10. TRA$

Problem 8. (10 points) The table below shows the optimal pairwise alignments between strings ATCTTAG, ATTAG, AGCTG and ATG. Show how to construct a multiple sequence alignment (MSA) using the STAR approximation. Make sure to state 1) which sequence is the "center", and why?, 2) the order in which sequences are added to the MSA, and 3) each partial MSA as sequences are added (i.e., you should have 3 MSAs in all). The pairwise alignments are calculated using global alignment with linear gap penalty with the following parameters: match: -2, mismatch: 1, gap: 5

`ATCTTAG`
 	
`A--TTAG cost: 0`		

`ATCTTAG`
 	
`AGC-T-G cost: 3`
	`ATTAG`
	
	`AGCTG cost: -1`	

`ATCTTAG`
 	
`A---T-G cost: 14`
	`ATTAG`
	
	`A-T-G cost: 4`
		`AGCTG`
		
		`A--TG cost: 4`

The "star" approximation chooses as the core the sequence that minimizes the sum of costs when aligned (pairwise) to the other sequences. The sum of costs are:

ATCTTAG: 0+3+14=17

ATTAG: 0-1+4=3

AGCTG: 3-1+4=6

ATG: 14+4+4=22

So, sequence ATTAG is chosen as the core. The progressive MSAs are then

```
1)  ATCTTAG

    A--TTAG

2)  ATCTTAG

    A--GCTG

    A--TTAG

3)  ATCTTAG

    A--GCTG

    A---T-G

    A--TTAG
```

Long Questions (you should always PROVE THE CORRECTNESS of your solutions)

Problem 9. (20 points) In class we discussed two approaches to sequence alignment – global and local alignment. A global alignment requires the two sequences to be aligned end-to-end while a local alignment allows one to ignore any mismatches occurring at the end of the sequences. There is, however, a middle ground - the **semiglobal** alignment. In a semi-global alignment all characters in the two sequences must be aligned, however only gaps internal to the alignment are counted, while gaps at either end of the alignment are "free".
For example, the following sequences aligned optimally using global alignment:

```
CAGCACTTGGATTCTCCGG
CAGC-----G-T-----GG
```

can be aligned optimally in a semiglobal fashion as follows:

```
CAGCA-CTTGGATTCTCGG
---CAGCGTGG--------
```

Describe a dynamic programming algorithm that computes the semiglobal alignment of two strings in time O(mn). I.e., describe recurrence relations, starting conditions and how to use table(s) to implement the DP algorithm.

You can achieve this by modifying the DP algorithm for global alignment with affine gaps. The important idea here is that alignments that have gaps in the beginning or end are 0-cost. For example, an alignment of \eps (empty string) to y[1..j] (i.e., starts with j gaps on the x string) has 0 cost. Therefore Ix[0,j]=0 for all j>0, and similarly Iy[i,0]=0 for all i>0.

To get free gaps at the end of the alignment, we modify the recurrence relations for the gap matrices (Ix and Iy). For Ix we would have:

$$I_x(i,j) = \begin{cases} M(i,j-1) + g(i)(o+e) \\ I_x(i,j-1) + g(i)e \\ I_y(i,j-1) + g(i)(o+e) \end{cases}$$

where $g(i) = \begin{cases} 0 \; if \; i = |x| \\ 1 \; o.w. \end{cases}$ Thus, moves along the corresponding border incur no additional cost. You can define a similar relation for Iy. Trace back starts from with same rule as global alignment: max(M(n,m),Ix(n,m),Iy(n,m)).

Problem 10. (15 points) Suppose you are given a *dictionary D* of strings $s_1, \dots, s_k$. Define language L as the set of strings that can be generated by concatenating two or more strings in D. We are interested in determining if D is a *minimal*, that is, no subset of strings $s_1, \dots, s_k$completely generate language L. You can use keyword trees to address this question

(1). Write an algorithm using a keyword tree to determine if dictionary *D* is minimal. *Hint: use a keyword tree to determine if a string in D can be generated by concatenating two or more other strings in D.*

(2). Write an algorithm using keyword trees to find a *minimal* subset of *D.*

(3). Prove that you can do this in linear time.

See midterm 1