Student ID: _____ Student Name: _____

CompSci 369: Computational Biology
Midterm Test 2007

1 Short Answers

1. Homology and similarity

   (a) Two genes from the same genome are found to be homologous. Is the homology orthologous or paralogous? Explain.

   The homology must be paralogous. Orthology is homology by speciation and occurs between two genes in different genomes that arose from the same ancestral gene in a common ancestral genome. Paralogy is homology by gene duplication and can occur between two genes in the same genome or in different genomes

   (***3 marks***)

   (b) Explain how sequence alignment can be used to infer homology.

   In general terms if two sequences are similar over a significant length they are generally assumed to be homologous. Specifically, if two sequences are more closely related to each other (more similar) than can be explained by chance, then this is usually taken as evidence of common ancestry and therefore homology. To make such an inference formal, the crucial requirement is to compare the similarity with what would be expected by chance. This requires a models that takes into account what would be expected under the null model (similarity by chance) and what would be expected under the alternative model (that the two sequences are related by an shared evolutionary history). Markov models of nucleotide or amino-acid substitution are generally used to assess the latter.

   (***5 marks***)

2. Modeling genetic change

(a) Explain the Jukes-Cantor genetic distance:

The Jukes-Cantor genetic distance is based on a simple time-continuous discrete-state homogenous Markov model of genetic change that assumes all mutations are equally likely and all character states have equal probability at equilibrium. The Jukes-Cantor genetic distance is always greater than the p-distance because it includes the possibility of multiple changes at a single site and hidden changes.

(***3 marks***)

(b) What is the p-distance?

The p-distance is also called the Hamming distance and is simply the proportion of sites in the two aligned sequences that are different.

(***3 marks***)

(c) Can the Jukes-Cantor distance be calculated from the p-distance?

Yes

(***1 marks***)

3. Dot plots and sequence alignment

(a) What does a dot in a dot plot signify? If the window size is w and the two sequences are length $s \geq w$ and $t \geq w$, how many substring comparisons does the dot plot represent?

A dot at position (i, j) represents two substrings of length w centered around position i in the first sequence and position j in the second sequence that share at least k nucleotide matches (where $k \leq w$). Since every pairs of substrings of length w are compared, there is a total of $(s - w + 1)(t - w + 1)$ comparisons ($\Theta(st)$).

(***5 marks***)

(b) Explain the difference between local and global pairwise alignment and give an example of when you would use each:

Local sequence alignment is more relevant when only a small region of each of the sequences is expected to be homologous with the other such as when aligning two genomic fragments. Global pairwise sequence alignment is relevant when you expect the two sequences to be homologous along their entire length, such as in the case of full length cDNAs from two related organisms.

(***5 marks***)

(c) What are the main differences between the progressive alignment methods of PILEUP and CLUSTALW?

The main difference between PILEUP and CLUSTALW is that CLUSTALW does a profile alignment at each node in the guide tree, whereas PILEUP does a simple pairwise alignment at each node in the guide tree between the two closest sequences in the subtrees. Also CLUSTALW uses the Neighbour-joining algorithm to construct the guide tree, whereas PILEUP uses the UPGMA algorithm. In addition there are many specific heuristics in CLUSTALW such as (a) special penalties for gaps based on the types of amino acids in a region, (b) the use of different score matrices depending on how related the sequences are, and many more.

(***5 marks***)

2 Algorithms

1. We want to design a dynamic programming algorithm that, given a sequence $a_1, a_2, \ldots, a_n$ of real numbers, finds a *clean subsequence* $a_{i_1}, a_{i_2}, \ldots, a_{i_t}$ of elements so that the sum $\Sigma_{j=1}^{t} a_{i_j}$ is a maximum. The subsequence is *clean* if $i_j < i_{j+1} \leq i_j + 2$ for all indices $1 \leq j < t$. In other words, the sequence is almost consecutive—allows gaps of size at most 1. If a_i is negative for all i, we define the maximum sum to be zero (which is obtained by selecting the empty subsequence).

   (a) For the sequence $(3, -4, \pi, -3.5, 8, -3, -2.5, 2\sqrt{2}, -3.2, -10, 5.5)$ find a clean subsequence with the largest sum.

   Take $3, \pi, 8, -2.5, 2\sqrt{2}, -3.2, 5.5$

   (***3 marks***)

   (b) For a given sequence of length n, consider the number of differenent clean subsequences that it may contain. Is this count a *polynomial* or *exponential* function of n? Explain why. [Hint: observe that there are $\binom{n}{2} + n + 1 = O(n^2)$ possible different consecutive subsequences.]

   Consider the number $f(i)$ of clean sequences with final element at position $1 \leq i \leq n$. The length of these sequences is 1 or more. We can easily see $f(1) = 1$ and $f(2) = 2$. For $i > 2$, we have $f(i) = 1 + f(i-1) + f(i-2)$ by counting the number of length 1, sequences including element at position $i - 1$ and sequences *not* including elements at postition $i - 1$. Thus this function is *exponential* since there is a Fibonachi-like recursion that counts them.

   (***3 marks***)

   (c) Explain precisely what are the parameters and subproblems with respect to your dynamic program design.

   A couple possible answers. The better approach is to consider the subproblems of finding the maximum clean sequence sum where the last element is at position i. Define $S(i)$ for $1 \leq i \leq n$ to be the maximum sum of a clean sequence such that the last element is a_i. The original problem is then $\max_{i=1}^{n} S(i)$ or zero, whatever is larger.

   (***3 marks***)

(d) Give a high level description and informal justification of correctness of your algorithm, but not the implementation details.

The dynamic program follows the counting relation given earlier.

$S(1) = a_1$;
$S(2) = \max\{a_1 + a_2, a_2\}$;
For i=3 to n:
    $S(i) = \max\{a_i, a_i + S(i-1), a_i + S(i-2)\}$;
$best = 0$;
For i=1 to n:
    $best = \max\{best, S(i)\}$;
return $best$;

(***4 marks***)

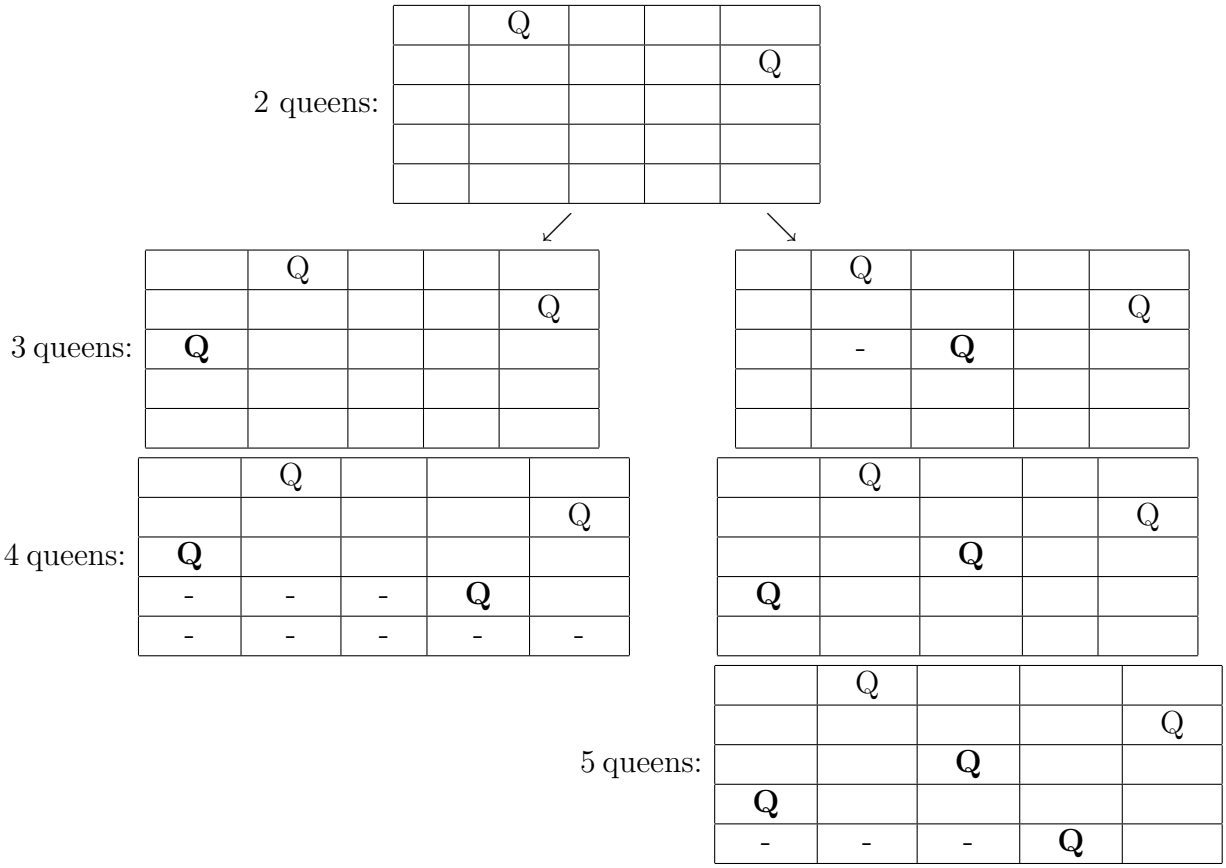
(e) Assuming each of the real numbers can be encoded with a constant number of bits, what is the running time of your algorithm? Express your answer in terms of $\Theta(f(n))$ notation, for some simple polynomial function f.

The above algorithm needs two linear loops so $\Theta(n)$

(***2 marks***)

2. Recall the n-Queens problem, where we want to place n non-attacking queens on a $n \times n$ chess board. A queen may attack horizontally, vertically and diagonally any number of squares.

   (a) For the $n = 5$ case illustrate (by placing "Q" in the cells) the next several moves of a backtracking algorithm starting at the following board configuration.



(***9 marks***)

   (b) Recall that we can use a permuation on $\{1, 2, \ldots, n\}$ to represent an embedding of n queens on a $n \times n$ chess board. That is, the i-th element corresponds to which column to put a queen on the i-th row. For $4 \leq n \leq 6$ list the first solution of the n queens problem by filling in the least lexicographic permutation in the following table:

n	**permutation on** $\{1, 2, \ldots, n\}$
4	2 4 1 3
5	1 3 5 2 4
6	2 4 6 1 3 5

(***6 marks***)