Exercise 2

Deadlines: Friday 2008.09.19 (copy) and Friday 2008.09.26 (corrected)

PROBLEM 1

Suppose you have a maximization algorithm, A, that has an approximation ratio of ¼. When run on some input π, $A(\pi) = 12$. What can you say about the true (correct) answer $OPT = OPT(\pi)$?

1. $OPT \geq 3$
2. $OPT \leq 3$
3. $OPT \geq 12$
4. $OPT \leq 12$
5. $OPT \geq 48$
6. $OPT \leq 48$

What if A is a minimization algorithm? (Hint: This is a trick question)

PROBLEM 2

a) Perform the ImprovedBreakpointReversalSort algorithm (below) with π = 3 4 6 5 8 1 7 2 (Remember to start with **0** 3 4 6 5 8 1 7 2 **9**).

b) The if-test in line 2 ensures that the algorithm never gets stuck in a situation were there is no reversal that decrease the number of breakpoints. Can you construct a permutation **σ** where this if-test is needed (i.e. with no decreasing strips and no reversal that reduces the number of breakpoints)?

c) Since this is an approximation algorithm, there might be a sequence of reversals that is shorter than the one found by ImprovedBreakpointReversalSort. Can you construct a permutation **σ** for which this is the case?

```
ImprovedBreakpointReversalSort(π)
1   while b(π) > 0
2       if π has a decreasing strip
3           Among all possible reversals, choose reversal ρ that minimizes b(π·ρ)
4       else
5           Choose a reversal ρ that flips an increasing strip in π
6       π ← π·ρ
7       output π
8   return
```

PROBLEM 3

The pseudo-code for ImprovedBreakpointReversalSort above leaves out a number of implementation details. Write subroutines that in detail perform the tasks of

a) line 2,

b) line 3,
c) line 5, and
d) line 6.

You may assume that the operation b(π) is available to you (i.e. you don't have to implement it). Also, note that you will implement the operation $\pi \cdot \rho$ in **d)**, so you don't have to implement it when solving **a)**, **b)** and **c)**.

PROBLEM 4

a) The GreedyMotifSearch algorithm below may miss a strong pattern because of its greedy nature. Design an input for GreedyMotifSearch that causes the algorithm to output an incorrect answer.

b) The popular CONSENSUS motif finding software is basically an implementation of GreedyMotifSearch. One important difference is that CONSENSUS can scan the sequences in a random order. Design a strategy that utilizes this feature to make it less likely that the algorithm outputs an incorrect answer. Would this strategy stop the algorithm from outputting the wrong answer for the input you designed in **a)**?

c) Another difference between GreedyMotifSearch and CONSENSUS is that the latter saves the 1000 best motifs from the two first sequences. Rewrite GreedyMotifSearch to include this feature. Does this make the algorithm correct (in the strict sense: for every possible input the algorithm will output the correct answer)?

```
GreedyMotifSearch(DNA, t, n, l)
1   s ← (1,1, …, 1)
2   bestMotif ← s
3   for s_1 ← 1 to n – l + 1
4       for s_2 ← 1 to n – l + 1
5           if Score(s, 2, DNA) > Score(bestMotif, 2, DNA)
6               bestMotif_1 ← s_1
7               bestMotif_2 ← s_2
8   s_1 ← bestMotif_1
9   s_2 ← bestMotif_2
10  for i ← 3 to t
11      for s_i ← 1 to n – l + 1
12          if Score(s, i, DNA) > Score(bestMotif, i, DNA)
13              bestMotif_i ← s_i
14      s_i ← bestMotif_i
15  return bestMotif
```