Exercise 1

Deadlines: Tuesday 2008.09.16 (copy) and Tuesday 2008.09.23 (corrected)

PROBLEM 1

Write two algorithms that iterate over every index combination from (0, 0, ..., 0) to (n_1, n_2, ..., n_d). Make one algorithm iterative and the other recursive. What application do you see for this algorithm?

PROBLEM 2

Fibonacci's model of rabbit expansion: One pair of adult rabbits creates a new pair of rabbits in the same time that it takes bunnies to grow into adults (i.e. one year). Thus the number of rabbits at time n is $F_n = F_{n-1} + F_{n-2}$, where $F_1 = F_2 = 1$. The intuition behind this is that the number of adult rabbits at time n is the number of rabbits (adults and babies) at time $n-1$, i.e. F_{n-1}, while the number of baby rabbits at time n is the number of adult rabbits at time n-1, which is F_{n-2}.

Propose a more realistic model of the rabbit life (and death) that limits the life span of rabbits by $k = 2.999$ years. Then the corresponding sequence grows more slowly than the Fibonacci sequence. Write a recurrence relation and pseudo-code to compute the number of rabbits under this model. Will the number of rabbits ever exceed the number of atoms in the universe under these assumptions?

PROBLEM 3

Let x = n.

Is $\log n = O(x)$?
Is $\log n = \Omega(x)$?
Is $\log n = \Theta(x)$?

If the answer is "no" to any of the questions, restate the question by changing x so that the answer is "yes".

PROBLEM 4

A multiset $\measuredangle X$ is the set of all pairwise positive distances between elements in an order set X, e. g. the multiset of $X = \{0, 2, 4, 7, 10\}$ is $\measuredangle X = \{2,2,3,3,4,5,6,7,8,10\}$.

Write an algorithm that, given the set X, calculates the multiset $\measuredangle X$.

PROBLEM 5

Consider the partial digest

$$L = \{1,1,1,2,2,3,3,3,4,4,5,5,6,6,6,9,9,10,11,12,15\}$$

Use the PartialDigest algorithm below to solve the partial digest problem for L (i.e. find X such that $\Delta X = L$). Illustrate the recursive calls by drawing a tree.

```
PartialDigest(L)
1       width ← Maximum element in L
2       Remove width from L
3       X ← {0, width}
4       Place(L, X)

Place(L,X)
1       if L is empty
2               output X
3               return
4       y ← Maximum element in L
5       if Δ(y, X) ⊆ L
6               Remove lengths Δ(y, X) from L and add y to X
7               Place(L, X)
8               Remove y from X and add lengths Δ(y, X) to L
9       if Δ(width – y, X) ⊆ L
10              Remove lengths Δ(width – y, X) from L and add width – y to X and
11              Place(L, X)
12              Remove width – y from X and add lengths Δ(width – y, X) to L
13      return
```

PROBLEM 6

A *complete k-ary* tree is a tree where each vertex that is not a leaf has exactly *k* children. It is also *balanced* since the number of edges in the path from the root to any leaf is the same (often referred to as the *height* of the tree). Find a closed-form expression for the total number of vertices in a complete and balanced k-ary tree of height L.

PROBLEM 7

Derive a tighter bound for the branch-and-bound strategy for the median string problem (see the BrachAndBoundMedianSearch algorithm below).

Hint: Split the *l*-mer *w* into two parts, *u* and *v*. Use TotalDistance(*u*,***DNA***) + TotalDistance(*v*,***DNA***) to bound TotalDistance(*w*,***DNA***). Take advantage of the fact that you might already have computed the best distances for substrings of length |v|.

```
BranchAndBoundMedianStringSearch(DNA,t,n,l)
1       s ← (1,1,…,1)
2       bestDistance ← ∞
3       i ← 1
4       while i > 0
5               if i < l
6                       prefix ← Nucleotide string corresponding to (s1, s2, …, si)
7                       optimisticDistance ← TotalDistance(prefix,DNA)
8                       if optimisticDistance > bestDistance
9                               (s, i ) ← Bypass(s,i, l, 4)
10                      else
11                              (s, i ) ← NextVertex(s, i, l, 4)
12              else
13                      word ← Nucleotide string corresponding to (s1, s2, …, si)
14                      if  TotalDistance(s,DNA) < bestDistance
15                              bestDistance ← TotalDistance(word, DNA)
16                              bestWord ← word
17                      (s,i) ← NextVertex(s,i,l, 4)
18      return bestWord
```