Algorithms for Bioinformatics (autumn 2010)

Exercise 1 (Mon 13.9, 10-12, C222)

1. **Simple things with Python I.**

   a) Write a Python procedure that computes the reverse complement of a given DNA sequence.

   b) Find out what are *BioPython* and *FASTA* file format. How would you solve a) with BioPython?

2. **Simple things with Python II.**

   a) The course book (page 29) describes the naive and simple sorting algorithm called *selection sort*. Implement it with Python.

   b) What build-on procedures does Python offer for sorting? Are they expected to perform better than your implementation of selection sort?

3. **Exhaustive enumeration I.**

   Complete the `BRUTEFORCECHANGE(`M, c, d`)` Python program given at the lectures (see below) with the procedure `nextTuple(i,M,c,d)`. Try the program with some inputs. What do you observe? Do you find any ways to speed up the program?

```
from Numeric import zeros, alltrue

def BRUTEFORCECHANGE(M,c,d):
   smallestNumberOfCoins = M
   i = zeros(d)
   bestChange = zeros(d)
   while True:
      valueOfCoins = 0
      for k in range(d):
         valueOfCoins = valueOfCoins + i[k]*c[k]
      if valueOfCoins == M:
         numberOfCoins = 0
         for k in range(d):
            numberOfCoins = numberOfCoins + i[k]
         if numberOfCoins < smallestNumberOfCoins:
            smallestNumberOfCoins = numberOfCoins
            bestChange = i.copy()
      i = nextTuple(i,M,c,d)
      if alltrue(i==zeros(d)):
         break
   return bestChange

print BRUTEFORCECHANGE(55,(20,10,5,2,1),5)
```

4. **Exhaustive enumeration II.**

   Write a Python program that generates all possible DNA sequences of length ℓ.

5. **Exhaustive enumeration III.** Below is the codon to amino acid translation table an its reverse coded in Python:

```
code = {  'ttt': 'F', 'tct': 'S', 'tat': 'Y', 'tgt': 'C',
          'ttc': 'F', 'tcc': 'S', 'tac': 'Y', 'tgc': 'C',
          'tta': 'L', 'tca': 'S', 'taa': '*', 'tga': '*',
          'ttg': 'L', 'tcg': 'S', 'tag': '*', 'tgg': 'W',
          'ctt': 'L', 'cct': 'P', 'cat': 'H', 'cgt': 'R',
          'ctc': 'L', 'ccc': 'P', 'cac': 'H', 'cgc': 'R',
          'cta': 'L', 'cca': 'P', 'caa': 'Q', 'cga': 'R',
          'ctg': 'L', 'ccg': 'P', 'cag': 'Q', 'cgg': 'R',
          'att': 'I', 'act': 'T', 'aat': 'N', 'agt': 'S',
          'atc': 'I', 'acc': 'T', 'aac': 'N', 'agc': 'S',
          'ata': 'I', 'aca': 'T', 'aaa': 'K', 'aga': 'R',
          'atg': 'M', 'acg': 'T', 'aag': 'K', 'agg': 'R',
          'gtt': 'V', 'gct': 'A', 'gat': 'D', 'ggt': 'G',
          'gtc': 'V', 'gcc': 'A', 'gac': 'D', 'ggc': 'G',
          'gta': 'V', 'gca': 'A', 'gaa': 'E', 'gga': 'G',
          'gtg': 'V', 'gcg': 'A', 'gag': 'E', 'ggg': 'G'
       }
codons = dict()
for c in code:
   if code[c] not in codons:
      codons[code[c]]=list()
   codons[code[c]].append(c)
for a in codons:
   print a
   print codons[a]
```

   Continue the above Python program so that it generates all those DNA sequences that translate into a given amino acid sequence. For example, given `IQ`, the program should output

```
attcaa
attcag
atccaa
atccag
atacaa
atacag
```