Questions on Zerbino and Birney (2008) *Genome Research*.

Velvet: Algorithms for de novo short read assembly using de Bruijn graphs.

1. What where the practical challenges that prompted this paper?

2. What are de Bruijn graphs?

3. What are attractive features of de Bruijn graphs for sequence assembly?

4. In the context of an assembled genome, what is a contig?

5. The paper uses coverage and N50 length to measure the quality of assemblies. What is the N50 length? [You might want to refer the original papers describing sequencing of the human genome for a description of N50 measure of assembly quality].

6. The paper summarizes four main algorithmic steps involved in using a de Bruijn graph to build a genome assembly using Velvet. What are they?

7. In the context of an assembled genome, what is a super-contig? What is the difference between a sequenced connected super-contig (in a de Bruijn graph framework) and a gapped super-contig?

8. Consider Table 1 and Table 2. What are the most striking features of the tables, in your opinion? What is the ideal assembly?

9. How are read pairs used in the Velvet assembly algorithm?

10. What struck you most about the paper?

PSEUDO-CODE, adapted from Figure 3 in Li and Durbin (2009)

```
InexactRecursion(W, i, z, k, l)
     // Parameter W is the word being searched for
     // Parameter i is an index within the word
     // Parameter z is the maximum allowed number of differences
     // Parameters k and l denote the current suffix array interval

     if (z < D[i]) then return {};

     if (i < 0) then return {{k,l}};

     RESULT <- {}

     RESULT <- RESULT ∪ InexactRecursion(W, i-1, z-1, k, l)

     Foreach base ∈ {A,C,G,T} do

          k* <- C(base) + O(base,k-1) + 1
          l* <- C(base) + O(base,l)

          if (k <= l) then

               RESULT <- RESULT ∪ InexactRecursion(W, i, z-1, k, l)

               if (base == W[i])
                    RESULT <- RESULT ∪ InexactRecursion(W, i-1, z, k, l)
               else
                    RESULT <- RESULT ∪ InexactRecursion(W, i-1, z-1, k, l)

     return RESULT
```