# Computational Thinking 2019/20
# Bioinformatics Coursework

Rob Powell

Hand in by 2pm on 14th Feb 2020 via DUO.

**Coursework Description**

This coursework consists of three parts and all three parts require you to develop some Python code. The first two objectives each require you to implement an algorithm for aligning DNA sequences, and the third requires you to implement an algorithm that allows us to build phylogenetic trees. You should submit your three pieces of Python code via DUO.

**Objective One: Using recursion to align two DNA sequences**

Develop a Python program that uses recursion (as described in the lectures) to compute the optimal alignment of two sequences. It should meet the following conditions:

- The algorithm should read in two sequences from text files (the sequences may be of different lengths).
- Use a recursive algorithm to compute all possible alignments of the two sequences.
- Score each alignment using the following scores:
  - +3 for a matching pair of 'A' bases,
  - +2 for a matching pair of 'C' bases,
  - +1 for a matching pair of 'G' bases,
  - +2 for a matching pair of 'T' bases,
  - -3 for a mismatching pair of bases,
  - -4 for a gap.
- The program should print out the following:
  - An optimal alignment and its score.
  - The number of alignments calculated.
  - Execution time.

As an example consider the following alignment of two sequences of length 4. It would be given score of -7.

| A | A | T | G | – |
|---|---|---|---|---|
| A | C | – | G | T |
| +3 | -3 | -4 | +1 | -4 |

Note: A template is provided for you that covers the first and last bullet point. You shouldn't need any additional packages. You should test your program with the test sequences provided on DUO. These will allow you to check your program is working as expected. Your program should be able to handle two input sequences of different length e.g aligning a sequence of length 8 with a sequence of length 10, as well as two sequences of the same length.

**Objective Two: Using local alignment to find matching regions in DNA sequences**

Develop a Python program that uses local alignment (as described in the lectures) to compute the optimal local alignment of two sequences. It should meet the following conditions:

- The algorithm should read in two sequences from text files (the sequences may be of different lengths).
- Initialise the scoring and backtracking matrices.
- Score each alignment using the following scores:
  - +3 for a matching pair of 'A' bases,
  - +2 for a matching pair of 'C' bases,
  - +1 for a matching pair of 'G' bases,
  - +2 for a matching pair of 'T' bases,
  - -3 for a mismatching pair of bases,
  - -4 for a gap.
- Use the backtracking matrix to determine an optimal local alignment.
- The program should print out the following:
  - An optimal local alignment and its score.
  - Execution time.

For an example of local alignment you should refer to the lecture slides.

**Objective Three: Implementing the Neighbour Joining algorithm**

Develop a Python program that implements the Neighbour Joining algorithm (as described in the lectures). This should continually cluster species together until only two species remain (we are not going to build the actual tree). It should meet the following conditions:

- Your code should have a function called NJ which takes as its input the name of a text file.
- The algorithm should read in an inter-species distance matrix from this text file, and print this matrix to the screen, including the row sums.
- Calculate and print to the screen all of the Q scores
- Follow the Neighbour Joining algorithm, and print out to the screen every reduced distance matrix (including the row sums), and their associated Q score matrix.

Note: You may use a package such as numpy for matrices if you find it helpful. There are two test matrices stored in text files available on DUO. You should use these to understand the format of the input that I will test your code with, and they allow you to check your program is working as expected. Your program should be able to handle an input matrix of any size up to 25 by 25.

**Marking Criteria**

The breakdown of marks are as follows:

- Objective One - 25%
- Objective Two - 40%
- Objective Three - 35%

The majority of marks are given for correctness. I will test your code against a large set of input test cases, of which you will be given a subset to test your code. Partial marks will be awarded for aspects of the code that work as expected, even if the code is incomplete or fails to always return correct outputs.

5 of the 25 marks for Objective One, and 5 of the 40 marks for Objective Two will be given for how efficiently your code executes, provided your code correctly passes all of the test cases. These will be awarded based on a sliding scale against the execution time of a Python program of my own.

There are no marks for code quality, but I will read your code, and it may be helpful to include comments that aid understanding.