# MATH578A 2020: Assignment #1

Due: March 1, 2020

This assignment is worth 15% of your total grade in the course. The relative point values are indicated for each question (out of 100).

**General instructions:** *This is not group work.* This assignment includes "practical" questions and "theoretical" questions. For the practical questions, you may discuss the questions and issues related to coding with other students. You cannot copy code from other students, and this extends to reading solution code other students have written. For the theoretical questions (all taken from Gusfield's text), you may not discuss the questions or their solutions with other students. Doing so will be considered cheating. If you need clarification, or get stuck on a question, ask me. I will often be able to give you some pointers, and in many cases to ensure fairness I will also send the same information to the entire class (don't worry: if I share the answer to your question with the class, I will not identify who asked the question).

The assignment should be handed in by email, and must include source code files and a single PDF document containing additional information requested in the questions, along with solutions to the theoretical questions. These should be sent as an archive produced by tar with bzip2 compression. Make sure decompression generates a directory that contains your files, rather than a collection of files in the current directory. Do not include compiled code or data files. Pay close attention to how your answers are presented. As a Ph.D. student, you are expected to hand in professional quality work.

Code must be clean and readable, with comments when appropriate. Code must be in C or C++. Although you are free to use whatever development environment or tools you want, your code should not include any evidence of the environment you used for coding (e.g. no hidden files created by IDEs or your OS). Make sure to use pass-by-reference as appropriate; failure to do so can render your implementation incorrect. Note that 20% of the value of this assignment is reserved for how your work is presented.

**Note:** Until February 17, I will do my best to answer *any* of your questions related to coding, even if they are specifically related to the code you must produce for this assignment, and even if it requires that I give feedback on code you have written. Until February 24, I will answer questions related to the coding you must do for this assignment, but excluding questions about your code for the Boyer-Moore algorithm (e.g. I will still answer specific questions about how you are reading input or timing portions of your code). After February 24, I will only answer general coding questions, and I will not answer questions that require me to read code you have written. So by starting early and consulting me as needed, you can almost guarantee a perfect score on Question 1 and most of Question 6.

**Question 1 (40 Points)**

**(Implement the Boyer-Moore algorithm)** The Boyer-Moore algorithm (BM) is described in Gusfield's text, Chapter 2. I have also posted the original paper, but be warned that the approaches are slightly different. You must implement the BM algorithm, using what Gusfield calls the extended bad character rule and the "strong" good-suffix rule. You will do this twice (copy the first version and modify) – the second version includes extra lines of code that (a) count the number of character comparisons done while searching the text for the pattern, and (b) time the portion of your code that does the searching (not including pre-processing the pattern).

Important points:

- The following rules make it easier for me to read your code, which is important since I need to read code from all of you. They are also pretty standard. (1) Do not surpass 80 characters per line in your source code. (2) Break lines at sensible places for readability. (3) Be consistent in how you use of white-space around operators, near parentheses, etc. (4) Convert tabs to spaces before submitting your code. Most text editors (e.g. vi, emacs, nano, etc.) can do this automatically. (5) Do not finish any non-empty lines of code with white-space characters. (6) Remember that I need to read the code you submit. Sloppiness means I may not be able to read your code, which may indirectly cost you marks (and will directly cost you marks on Question 6).

- Gusfield shows how to obtain the preprocessing for the good suffix rule by first computing Z values. You may do this, or follow the original approach in the article I posted. It might be easier to use the Z values, and you are free to use the Z value code I posted on Blackboard as a function in your program. Attempting Question 5 first might help with this part.

- You are free to copy any parts any code I posted to help you implement the BM. If you have not previously coded in C or C++, this will allow you to avoid many potential problems. For example, I've already provided functions you would need to read the input and parse the command line arguments.

- Your program must accept 2 inputs: one is a FASTA format file containing the text, and the other is the pattern written directly on the command line. This is similar to one of the KMP implementations I posted. The way the FASTA file is read in my code concatenates all sequences in the file. I would rather have the text loaded quickly, and we will not need be concerned with the locations of matches within chromosomes.

- The first version must be named `boyer_moore.c` or `boyer_moore.cc` / `boyer_moore.cpp` depending on which language you selected. The second version will differ just by including `_count` before the filename extension (e.g. `boyer_moore_count.c`).

- The output of the first program should be the number of occurrences of the pattern in the text, and the output of the second program should be the number of occurrences, the number of character comparisons, and the time required, in seconds to the appropriate precision (excluding reading the input and pre-processing). Nothing extra, just those numbers each preceded by a very brief label.

- The source code you submit must be able to be compiled using either gcc or g++. If you require particular flags to compiler your programs, make sure to give me instructions.

- Identify a vertebrate species with a name (could be genus if needed, or informal name) that starts with the same letter as your name (either first or last). You will use the full genome of this species as

the "text" in your tests. Indicate in your report which species you selected, where you obtained the genome, how large is the genome, and how many sequences are in the FASTA file. Make sure you do not download a repeat-masked genome. Also, make sure the genome is all upper-case, which can be done easily using "tr" on the command line. You only need to consider one strand.

- Run your program using the patterns in I post on Blackboard with this assignment. In your report, for each pattern, give the total run time (in seconds), the number of occurrences, and the total character comparisons.
- Then using the human genome (hg38) as the text, run your program with a random 50bp portion of a human Alu element as the pattern. In your report, provide the 50bp you used, along with the run time, number of occurrences of this 50bp and total character comparisons.

**Question 2 (10 Points)**
Chapter 1, exercise 11. Pseudocode is required.

**Question 3 (5 Points)**
Chapter 6, exercise 1.

**Question 4 (10 Points)**
Chapter 7, exercise 1.

**Question 5 (15 Points)**
Chapter 7, exercise 2.

**Question 6 (20 Points)**
Quality of presentation: a portion of the overall grade on this assignment will be determined by the degree of professionalism in how your work is presented.