Please print your answers and place them in the CS1810 homework bin on the 2nd floor of the CIT. You can use \LaTeX or a word document to write up your answers, but we prefer you use \LaTeX. You may scan hand-written work or images for parts of solutions only if they are extremely clean and legible. Please put your Banner ID on the top of each page of your homework. Please ensure that your name does not appear anywhere on the paper you hand in.

**Problem 1: Global alignment table**

Create and fill out the global alignment dynamic programming table, with scores and back pointers (including any ties), for aligning TIEBREAK and TIMBERLAKE. For scoring, use the match bonus +1, mismatch penalty −1, and indel penalty −1. What is the score of the optimal alignment and to which alignment does this score correspond?

**Problem 2: Multiple alignment**

In class you have learned about the global alignment of two strings. A natural question to ask is how we might go about aligning three strings. Your task is to design an efficient algorithm that solves the following problem.

> **GLOBAL ALIGNMENT OF THREE STRINGS**

**Input:** Strings \(u, v,\) and \(w\) and a scoring function \(\delta\) that takes in three arguments.

**Output:** An alignment of \(u, v,\) and \(w\) for which the score is maximal (as defined by \(\delta\)) among all alignments of \(u, v,\) and \(w.\)

Whereas we had to work fairly hard to build up the original binary global alignment algorithm from scratch, it turns out that we can obtain an algorithm for the global alignment of three strings by making some careful modifications to the algorithm you’ve seen in class.
In your solution, you must

- give a new recurrence relation for finding the score of an optimal alignment of a given combination of prefixes of $u$, $v$, and $w$,
- describe how the structure of the dynamic programming table changes in the extension to three strings,
- specify the order in which the new dynamic programming table should be filled out.

If you wish, you may also include diagrams, pseudocode, mathematical expressions, and plain English text in your answer.

Once we know how to extend our global alignment algorithm to three strings, we might consider making the general extension to $k$ strings. Show that if we continue to insist on constructing dynamic programming tables, the runtime of our algorithm will be $O((2^n)^k)$ for $k$ sequences of length $n$. Since this approach is exponential in $k$, we will quickly hit a computational wall in attempting to align sequences of even modest length.

**BONUS**: Sketch in a few sentences a reasonable heuristic we might use to skirt this problem if we would still like to align multiple sequences.

**Problem 3: Counting global alignments**

A brute force (rather than dynamic programming) approach to global alignment would iterate through every possible alignment of the two input strings, computing the score and storing the max along the way. To understand the performance of such an approach, we need to know the number of possible alignments between two strings, one of length $m$ and the other of length $n$. Finding an exact expression for the number of possible alignments is actually fairly difficult. In this problem, we simply ask you to determine whether the number of alignments is

1. polynomial in $m$ and $n$
2. logarithmic in $m$ and $n$
3. exponential in $m$ and $n$

Indicate your answer choice by number and give a few lines of justification.

**Problem 4: The Black Plague**

The Black Plague was one of the most devastating pandemics in human history, resulting in the death of more than one third of Europe’s total population and peaking during the years 1346-1353. The current scientific consensus is that the Black Plague was due to a pathogenic bacterium known as *Yersinia pestis*. 
In your first programming project, you will work with sequence data to determine whether London’s Great Plague of 1665 was also due to *Y. pestis*. The goal of this problem is to acquire such sequence data.

The primary resource for biomedical data is the collection of databases housed by the National Center for Biotechnology Information (NCBI). Navigate to the NCBI homepage in your browser. Select the “Genome” option in the database dropdown menu next to the search box. Enter “*Yersinia pestis*” in the search box and hit enter. Click on the appropriate link from the list of results. The NCBI’s user interface is a little dated, but it is also rich in information.

Click on the link to the bacterium’s reference genome. A reference genome (or reference assembly) is a digital nucleic acid sequence designed to serve as a representative example of a species’ DNA content. Reference genomes are often assembled by sequencing the DNA from a number of samples, and thus do not necessarily represent the genome of any single organism. The Human Genome Project took 13 years to assemble the first complete human genome. Today, it takes just over 24 hours to fully sequence a person’s DNA. The speedup is partly due to computational and technological advances over the last two decades, but it is also because we have a reference genome to serve as a guide. Sequencing DNA using a reference genome is like putting together a puzzle using the picture on the front of the box.

Returning to our plague bacterium, scroll down to the section titled “Genome Region” and click on the link “*Yersinia pestis* CO92 chromosome, complete genome.” (If you want to get a sense of the scale of the whole genome, click instead on the “FASTA” link to the right.) We will be interested in the pla2 gene, which codes for an outer membrane protease that serves as a plasminogen activator that contributes to virulence during infection (don’t worry if you don’t understand what that means!). You can simply do a “ctrl-F” for “pla2” to find the relevant section of the page. There should be two results: a gene section and a CDS section. CDS stands for “coding sequence,” i.e. the part of the gene that is transcribed to mRNA. In bacteria, the CDS and the gene are identical (this is not true in eukaryotes, including humans), so click on either one and then click on the “FASTA” link in the bottom right. FASTA is a standardized file format for biological sequence data. Download the gene by clicking on the “send” link in the upper right and selecting “coding sequences” from the dropdown. Print the downloaded FASTA file and turn it in along with your homework.

That’s it, you’re done! You have successfully downloaded a real gene sequence from the internet. The fun part is to actually run algorithms on the data and interpret the biological significance of the results. You’ll be doing just that on your first project.