ORIE 5270: BIG DATA TECHNOLOGIES
HOMEWORK 2 – DUE: MONDAY, 03/22/2021

Instructions. The deadline to submit is **Monday, March 22nd** at **12pm (noon) US EST)**. Submit your answers on **Gradescope**. Please submit a `.zip` archive containing a **single file per problem**, with the exception of Problem 6.

Note: Problem 6 asks you to build your personal website, and contains separate submission instructions. It is due on the last lecture of the semester.

Problem 1 (Complexity classes)**.** In this problem, you are given a set of functions $f_i(n)$. Sort these functions **in ascending order** according to their complexity class, as indicated by $O(\cdot)$ notation; in particular, if f_i appears before f_j in your answer, it should satisfy $f_i(n) = O(f_j(n))$.

Note: there is *only one* correct ordering for the set of functions given below. For some of these comparisons, you might find Stirling's formula useful.

$$
\begin{array}{llll}
f_1(n) = n^2 + n + 7, & f_2(n) = \log(n!), & f_3(n) = n^{1/4}, & f_4(n) = \log^{100}(n), \\
f_5(n) = 2^{n/2}, & f_6(n) = \log(n^{100}), & f_7(n) = 2^{n - \log_2 n}, & f_8(n) = \frac{n!}{\log(n!)}, \\
f_9(n) = \frac{1}{n}, & f_{10}(n) = n^{11} + 2^{-n}, & f_{11}(n) = 10^{20}. &
\end{array}
$$

What to submit: submit a single `.txt` file that contains a comma-separated list of indices i corresponding to the f_i's placed in ascending order. For example, if the first 3 functions in your answer are f_7, f_9 and f_2, your file should start like this: `7, 9, 2, ...`

Problem 2 (Power)**.** A standard operation in programming languages is computing the power of an element, x^p. In this problem, you will implement an algorithm in Python to efficiently compute powers when p is an integer.

1. Write a Python function `pow(x, n)` that implements the operation x^n. Assuming multiplication between two numbers takes time $O(1)$, your algorithm should run in time $O(\log_2(n))$. *You should not use Python's built-in `**` operator for this.*

2. Generalize your power function so that it computes powers of *matrices*, A^n, whenever A is **symmetric**, by writing a Python function `matrix_pow(A, n)`.

   You will likely find the following fact useful: if A is symmetric, there exists a matrix $Q \in \mathbb{R}^{d \times d}$ as well as a diagonal matrix Λ such that

   $$A = Q \Lambda Q^\mathsf{T}, \text{ and } Q^\mathsf{T} Q = Q Q^\mathsf{T} = I_d.$$

   You can use the `numpy.linalg.eigh` function from NumPy to obtain the above decomposition.

Problem 3 (Merging sorted lists)**.** Consider the following setup: you are given K lists sorted in ascending order, such that the *total number of elements* (i.e. summed over all K lists) is n. You wish to merge these K lists into a single, n-element sorted list.

1. Write a function called `simple_merge` that accepts a collection of sorted lists (e.g. a list of lists) and returns a sorted list containing all their elements in time $O(K \cdot n)$.
2. Write another function called `merge` that accepts a collection of sorted lists and merges them in time $O(n \log K)$.

   Hint: you can either (i) use an appropriate data structure for choosing the next element to insert to the final list, or (ii) design a divide-and-conquer style algorithm.

Problem 4 (Iterative quicksort)**.** In this problem, you will implement an iterative version of quicksort and compare it with Python's built-in sorting algorithm.

1. Implement a **non-recursive** version of quicksort, called `quicksort_iter(A)`. In particular, you should use a stack to keep track of the start and end indices, p, r, of the subarray you are currently partitioning.

   Feel free to use the code for the `partition` subroutine available in the lecture slides.
2. Generate a few random integer arrays of size $10^3, 10^4, \ldots, 10^6$ and compare the time elapsed by the algorithm you wrote in Part 1 with Python's built-in `sorted()` function. Is there a clear winner? You can use the `timeit` module to measure the runtime, and `numpy.random.randint` to generate random integer arrays. Make sure to allow large enough integer values for the elements of the array (comparable to the size of the array).

   The code you submit for this part should include the function and `timeit` statements you wrote to test the performance. You can include the results you got for each array size as comments in the source code.

Problem 5 (Word Count)**.** Write a `bash` script that accepts a single argument containing the path to a file and generates a file containing two columns: the first column contains all the distinct words encountered in the original file, while the second column contains the number of appearances of each word.

You may assume the following:

- The file will only contain letters (uppercase and lowercase) and the following 4 punctuation symbols: `?`, `,` (comma), `.` (dot), and `!`. In addition, *every pair of words in the input file is guaranteed to be separated by a space and/or punctuation symbol.*
- You don't need to worry about words sharing a common stem, e.g. plurals. For example, the words `computer` and `computers` should be treated as distinct words.
- Your output should be case-insensitive. This means that if the input file contains the words `computer` and `Computer`, you should treat them as appearances of the same word (computer) and count them both.

Hint 1: you will likely need the `tr` command that translates characters.

Hint 2: make use of the pipe construct to compose operations.

Problem 6 (Personal website)**.** If you don't have a personal website, now might be the time to build one! Many repository hosting services, such as Github and Gitlab, allow you to build static websites (meaning: HTML + CSS + Javascript, but no backend components).

How it works: you create a repository named like `<username>.github.io` which contains the source for your website. Then, a *static site generator* specified by a configuration file that you include in your repository builds your website from the source files you included. The website is then made available under the same URL as the name of your project.

Because the process varies depending on the hosting service as well as the site generator of your choice, you can take your time to browse and familiarize yourself with the service and the static website generator of your choice. Here are some links to help you get started.

1. Static website generators:
    - Jekyll: https://jekyllrb.com/
    - Hugo: https://gohugo.io/
    - Hakyll: https://jaspervdj.be/hakyll/
2. Links to static website hosting services:
    - Github pages: https://pages.github.com/
    - Gitlab pages: https://docs.gitlab.com/ee/user/project/pages/

What to turn in: by the last lecture of the semester, you should build at least a minimal website (e.g. Home / About Me / Projects / Misc) with some meaningful content. For example, you should add your bio and academic interest and a showcase of projects you have done in the past or are currently undertaking. Then, you should send me an email with subject line `[ORIE 5270] - Personal Website` containing the following:

- if you make your website public, your email should contain a link to your website.
- if you would rather not make your website publicly available at the time, your email should contain a `.zip` file with your website's source code, as well as instructions on how to "build" it locally.