**Exercise 1**(30 points)

Examine the contents of [hw2-patients.xml](https://yale.instructure.com/courses/70314/files/5401025?wrap=1)[Download hw2-patients.xml](https://yale.instructure.com/courses/70314/files/5401025/download?download_frd=1)in a text-editor to see its structure, but in brief, there are a handful of fields you can ignore for this exercise and then several <patient> entries, all contained inside <patients>. Each <patient> has several attributes that we will want, namely name, age, and gender. Some patients have other associated data (e.g. diagnoses), but we won't need that here.

Load the data. Plot a histogram showing the distribution of ages (2 points). Do any of the patients share the same exact age? (2 points) How do you know? (2 points).

(For an extra 2 points: explain how the answer to the question about multiple patients having the same age affects the solution to the rest of the problem.)

Plot the distribution of genders. (2 points). In particular, how did this provider encode gender? What categories did they use? (2 points)

Sort the patients by age and store the result in a list (use the "sorted" function with the appropriate key, or implement sort yourself by modifying one of the algorithms from the slides or in some other way). (2 points) Who is the oldest patient? (2 points).

Identifying the oldest person from a list sorted by age should be an O(1) task... but sorting is an O(n log n) process (assuming we're using an efficient algorithm), so the total time for the above is O(n log n). Describe how (you don't need to implement this, unless that's easier than writing it out) you could find the second oldest person's name in O(n) time. (2 points). Discuss when it might be advantageous to sort and when it is better to just use the O(n) solution. (2 points).

Recall from our discussion of the motivating problem for September 9th that we can search within a sorted list in O(log n) time via bisection. Use bisection on your sorted list (implement this yourself; don't trivialize the problem by using Python's bisect module) to identify the patient who is 41.5 years old. (2 points)

Once you have identified the above, use arithmetic to find the number of patients who are at least 41.5 years old. (2 points)

Generalizing the above, write a function that **in O(log n) time** returns the number of patients who are at least low\_age years old but are strictly less than high\_age years old. (2 points) Test this function (show your tests) and convince me that this function works. (2 points). (A suggestion: sometimes when you're writing high efficiency algorithms, it helps to make a slower, more obviously correct implementation to compare with for your tests. Be sure your function works both for ages that are and are not in the dataset.)

Modify the above, including possibly the data structure you're using, to provide a function that returns both the total number of patients in an age range AND the number of males in the age range, all in O(log n) time as measured after any initial data setup. (2 points). Test it (show your tests) and justify that your algorithm works. (2 points)

**Exercise 2** (25 points)

Download and uncompress the latest Human Reference Genome(GRCh38.p13) from [https://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA\_000001405.28\_GRCh38.p13/GCA\_000001405.28\_GRCh38.p13\_genomic.fna.gz (Links to an external site.)](https://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA_000001405.28_GRCh38.p13/GCA_000001405.28_GRCh38.p13_genomic.fna.gz)

The above should create the file GCA\_000001405.28\_GRCh38.p13\_genomic.fna. This is a FASTA file a little over 3 GB in size, representing the about 3 billion bases in the human genome.

(The Human Reference Genome is a product of the Genome Reference Consortium. It is a composite sequence representing no individual human but primarily derived from 11 individual humans. Other projects are trying to characterize the diversity possible across the species.)

**Write code to loop through all 15-mers, subsequences of 15 bases within chromosome 2 (CM000664.2)**.  (5 points)

Hint: Remember, it's generally best not to try to parse files yourself. If you have BioPython installed, you can do something like the following:

from Bio import SeqIO  
human\_genome = SeqIO.parse("GCA\_000001405.28\_GRCh38.p13\_genomic.fna", "fasta")  
for chromosome in human\_genome:  
    if chromosome.name == "CM000664.2":  
        sequence = str(chromosome.seq).lower().encode('utf8')  
        # DO STUFF HERE

(If you don't, consider installing BioPython.)

If you would like to learn more about BioPython's SeqIO module, its documentation is at [https://biopython.org/wiki/SeqIO (Links to an external site.)](https://biopython.org/wiki/SeqIO)

In the above, sequence is a byte string of lower-cased letters. I suggest doing this to simplify our call to the hashing function later, but you could leave it as a regular string instead.

Naively, you would expect 4^15 ≈ 1 billion potentially unique distinct subsequences, so by the pigeon hole principle, at least one subsequence must occur more than once in the genome. Things are a little more complicated because these FASTA files also include N for places where any nucleotides might occur. Ignore all subsequences containing more than 2 Ns.

(Recall that N can stand for any nucleotide, but for the purposes of this question consider two 15-mers the same if and only if they have the same sequence of A, C, T, G, and Ns... i.e. if two 15-mers are the same except the 3rd position is an N in one and an A in the other, then they are different.)

**How many total subsequences are there (counting duplicates) that do not contain more than 2 Ns?** (5 points)

**Using 100 hash functions from the family below and a single pass through the sequences, estimate the number of distinct 15-mers in the reference genome's chromosome 2 using the big data method for estimating distinct counts discussed in class**. (5 points) (Here I mean distinct in the sense that if we have the list 1, 1, 2, 3, 4, 5, 4, there are 5 distinct values that appear.) **How does your estimate change for different-sized subsets of these hash functions, e.g. the one with a=1 only, or a=1, 2, .., 10, or a=1, 2, ...100, etc?** (5 points) (I suggest combining the hashes by taking the median of the minimum values and then turning that into an estimate for distinct elements, but you may want to experiment with other strategies. I further suggest that to get a feel for this, you may want to vary the number of hash functions used more smoothly, e.g. 1, 2, 3, 4, ... 100.)

p = 2\_549\_536\_629\_329  
bits\_48 = 2 \*\* 48 - 1  
scale = 0x07ffffffff  
from hashlib import sha256  
def get\_ath\_hash(a):  
 def my\_hash(subseq):  
 return (((int(sha256(subseq).hexdigest(), 16) % bits\_48) \* a) % p) & scale  
 return my\_hash

For any byte string, this returns an integer between 0 and scale. For example,

>>> first\_hash = get\_ath\_hash(1)  
>>> second\_hash = get\_ath\_hash(2)  
>>> first\_hash(b"penicillin")  
21805768735  
>>> second\_hash(b"penicillin")  
9251799102

To get a number between 0 and 1 from this, you'll want to divide the returned value by scale. Note: you probably don't want to literally use this code as the sha256 return value is the same for all hashes in our family, so feel free to factor that out rather than calling it over and over again.

*(For those curious about the mathematics, this is an approximately universal hash family, and this result depends in part on the fact that this p is prime. This particular large prime p belongs to the family of Wieferich primes base 23.)*

This will take a while to run. Test your code on some medium sized fake data before trying it on chromosome 2. E**xplain your tests and why they convinced you that your code works** (5 points).

**Exercise 3**(20 points)

Your friend says to you, "you have to help me! I'm supposed to present in lab meeting in less than an hour, and I haven't been able to get the simplest thing to work!" After you help them calm down, they explain: through a collaboration with a fitness app, they have a 4GB file of high-precision weights of exactly 500 million people throughout the world. Even with 8GB of RAM, they get a MemoryError when trying to load the file in and find the average weight. They show you their code which worked fine when they tested it on a small number of weights:

with open('weights.txt') as f:  
 weights = []  
 for line in f:  
 weights.append(float(line))  
print("average =", sum(weights) / len(weights))

Aha! You exclaim.

**Explain what went wrong (6 points). Suggest a way of storing all the data in memory that would work (7 points), and suggest a strategy for calculating the average that would not require storing all the data in memory (7 points).**

*Remember, your friend has to present soon, so keep your answers concise but thorough.*

**Exercise 4**(25 points)

Identify a data set online (10 points) that you find interesting that could potentially be used for the final project; the main requirements is that there needs to be many (hundreds or more) data items with several identifiable variables, at least one of which could be viewed as an output variable that you could predict from the others.

Describe the dataset (10 points) Your answer should address (but not be limited to): how many variables? Are the key variables explicitly specified or are they things you would have to derive (e.g. by inferring from text)? Are any of the variables exactly derivable from other variables? (i.e. are any of them redundant?) Are there any variables that could in principle be statistically predicted from other variables? How many rows/data points are there? Is the data in a standard format? If not, how could you convert it to a standard format?

Describe the terms of use and identify any key restrictions (e.g. do you have to officially apply to get access to the data? Are there certain types of analyses you can't do?) (5 points)

Remember: if you can't find explicit permission to use a given dataset, assume that you cannot do so.

Note: You're not committing to use this dataset for the project, but this will give you one option.