1. Power of two choices (25 points)

Goal: The goal of this part is to gain an appreciation for the unreasonable effectiveness of simple load balancing, and measure the benefits of some lightweight optimizations.

Description: We consider random processes of the following type: there are $N$ bins, and we throw $N$ balls into them, one by one. We’ll compare four different strategies for choosing the bin in which to place a given ball.

1. Select one of the $N$ bins uniformly at random, and place the current ball in it.

2. Select two of the $N$ bins uniformly at random (either with or without replacement), and look at how many balls are already in each. If one bin has strictly fewer balls than the other, place the current ball in that bin. If both bins have the same number of balls, pick one of the two at random and place the current ball in it.

3. Same as the previous strategy, except choosing three bins at random rather than two.

4. Select two bins as follows: the first bin is selected uniformly from the first $N/2$ bins, and the second uniformly from the last $N/2$ bins. (You can assume that $N$ is even.) If one bin has strictly fewer balls than the other, place the current ball in that bin. If both bins have the same number of balls, place the current ball (deterministically) in the first of the two bins.
(a) (5 points) Write code to simulate strategies 1–4. For each strategy, there should be a function that takes
the number \( N \) of balls and bins as input, simulates a run of the corresponding random process, and outputs
the number of balls in the most populated bin (denoted by \( X \) below).

(b) (10 points) Let \( N = 200,000 \) and simulate each of the four strategies 30 times. For each strategy, plot
the histogram of the 30 values of \( X \). \(^1\) Briefly discuss the pros and cons of the different strategies. Does
one of them stand out as a "sweet spot"?

(c) (5 points) Propose an analogy between the first of the random processes above and the standard imple-
mentation of hashing \( N \) elements into a hash table with \( N \) buckets, and resolving collisions via chaining
(i.e., one linked list per bucket). Discuss in particular any relationships between \( X \) and search times in the
hash table.

(d) (5 points) Do the other random processes suggest alternative implementations of hash tables with chaining?
Discuss the trade-offs between the different hash table implementations that you propose (e.g., in terms
of insertion time vs. search time).

Deliverables: Your code for part (a); your histograms for part (b); your written answers for parts (c) and (d).

2. Heavy Hitters – Count-Min Sketch (30 points)

Goal: The goal of this part is to understand the count-min sketch from the lecture on January 24 via an imple-
mentation, and to explore the benefits of a conservative updates optimization.

Description: You will use a count-min sketch with \( \ell = 4 \) independent hash tables, where each hash table
has \( b = 256 \) counters (256 buckets). Your sketch should take a "trial" as input. The hash value of an element
\( x \) during trial \( i \) (\( i = 1, 2, \ldots, 10 \)) for table \( j \) (\( j = 1, 2, 3, 4 \)) will be computed as described below.

Hash Functions: First fix a prime number \( p \) that is greater than the largest element in the data stream. In this
assignment, use \( p = 10,007 \). The hash function for table \( j \) (for \( j = 1, 2, 3, 4 \)) is chosen as follows:

- Choose an integer \( e_j \) such that \( 1 \leq e_j \leq p - 1 \) uniformly at random. Then choose another integer \( g_j \) such
  that \( 0 \leq g_j \leq p - 1 \) uniformly at random. The integers \( e_j \) and \( g_j \) are chosen independently and randomly
  for each \( j \) (and each trial).
- The hash function for table \( j \) is then defined as \( h_j(x) = (e_j x + g_j) \mod p \mod b \).

(a) (5 points) Implement the count-min sketch using hash functions as described above.

The streams you will use will consist of the integers 1 to 9050 (inclusive) with the following frequencies:

- Integers \( 1000 \cdot (i - 1) + 1 \) to \( 1000 \cdot i \), for \( 1 \leq i \leq 9 \), appear \( i \) times in the stream. That is, the
  integers 1 to 1000 appear once in the stream; 1001 to 2000 appear twice; and so on.
- An integer \( 9000 + i \), for \( 1 \leq i \leq 50 \), appears \( i^2 \) times in the stream. For example, the integer 9050
  appears 2500 times.

(Each time an integer appears in the stream, it has a count of 1 associated with it.)

(b) (5 points) Call an integer in the stream a **heavy hitter** if the number of times it occurs (its frequency) is
at least \( 1\% \) of the total number of stream elements. How many heavy hitters are there in a stream with
the above frequencies?

\(^1\) For example, in Python, the numpy and matplotlib.pyplot libraries are useful for creating histograms. See the course website for
some starter code you can use and modify as necessary.
Next, you will consider 3 different permutations of this data stream:

- **Forward**: the elements appear in non-decreasing order.
- **Reverse**: the elements appear in non-increasing order.
- **Random**: the elements appear in a random permutation.

(c) (9 points) For each of the three data streams, feed it into a count-min sketch (i.e., successively insert is elements), and compute the values of the following quantities, averaged over 10 trials for each order of the stream:

- The sketch’s estimate for the frequency of element 9050.
- The sketch’s estimate for the number of heavy hitters (elements with estimated frequency at least 1% of the stream length).

Does the order of the stream affect the estimated counts? Explain your answer.

(d) (5 points) Implement the conservative updates optimization as follows. When updating the counters during an insert, instead of incrementing all 4 counters, only increment the subset of these 4 counters that have the lowest current count (if two or more of them are tied for the minimum current count, then we increment each of these).

(e) (2 points) Explain why, even with conservative updates, the count-min sketch never underestimates the count of a value.

(f) (9 points) Repeat part (c) with conservative updates.

(g) (3 points) We used $\ell = 4$ hash tables and the size of each table was $b = 256$. Suppose that element $x$ has frequency $0.003n$. Use Markov’s inequality to give a bound on the probability that $x$ is included in the output above as a heavy hitter.

(h) (2 points) Briefly describe 2 real-world applications of the Count-Min Sketch.

**Deliverables**: Your code for parts (a), (c), (d) ; your written answers to parts (b), (e), (f) – (h).