CSE 422 Homework 1

Hyunggil Woo

1. The power of two (or more) choices

(a)

```
def strategy_1(N : int):
    bins = np.zeros(N, dtype=int)
    for _ in range(N):
        n = np.random.randint(low=0, high=N, size=None)
        bins[n] += 1
    return np.argmax(bins) + 1

  def strategy_2(N):
    bins = np.zeros(N, dtype=int)
    for _ in range(N):
        b_1 = np.random.randint(low=0, high=N, size=None)
        b_2 = np.random.randint(low= 0, high=N, size=None)
        if bins[b_1] <= bins[b_2]:
            bins[b_1] += 1
        else:
            bins[b_2] += 1
    return np.argmax(bins) + 1

def strategy_3(N):
    bins = np.zeros(N, dtype=int)
    for _ in range(N):
        b_1, b_2, b_3 = np.random.randint(N, size=3)
        minimal_bin = min(bins[b_1], bins[b_2], bins[b_3])
        bins[minimal_bin] += 1
    return np.argmax(bins) + 1

def strategy_4 (N):
    bins = np.zeros(N, dtype=int)
    for _ in range(N):
        b_1 = np.random.randint(m.floor(N/2))
        b_2 = np.random.randint(m.ceil(N/2))
        minimal_bin = min(bins[b_1], bins[b_2])
        bins[minimal_bin] += 1
    return np.argmax(bins) + 1
```

(b) Let N= 200,000 and simulate each of the four strategies 50 times. For each strategy, plot a histogram of the 50 values of M (the maximum load). Discuss the pros and cons of each of the strategies based on the histograms.

I was unable to print the histogram because I did not have enough time to identify the error.

(c)

```
def simulate_with_cons_hashing(strategy, N):
    """
    Implemented an adaptation to the following.
    """
    bins = np.zeros(N, dtype=int)
    servers = sorted(np.random.sample(range(2**32), N))
    for i in range(N):
        x = np.random.randint(0, 2**32 - 1)
        bin_index = bisect_left(servers, x) % N
        bins[bin_index] += 1
    max_balls = strategy(bins)
    return max_balls
```

(d) Let N= 100,000 and simulate each of the three strategies 50 times. For each strategy, plot a histogram of the 50 values of M (the maximum load). Discuss the pros and cons of these variations of the strategies based on the histograms.

I was unable to fix my code to print the histogram. My code took a long time to print the items. I felt busy this week, and I wish I had more time to complete the assignment. But I found a partner to collaborate on the assignment.

(e) If there are **N** elements and **N** buckets, then it can be implemented, then the search time can take a linear time to identify the item.

(f) With strategy 4, the need for chaining may decrease because the buckets are chosen from two different regions of the array.

2. The Count-Min sketch

(a)
```
import numpy as np
import matplotlib.pyplot as plt
import math as m
import hashlib # import MD5

class CMS:
    """
    Representation of Count-Min Sketch.
    hash-table is present and number of buckets are needed to represent it.
    As an implementer, we modulate the episilon for the percentage that the
        computation is correct, and
    also the percentage of when the computation is also correct.

    """
    # TODO: hard coding l and b value for the count-min sketch.
    def __init__(self, s: int):
        """
        Takes s as an input.
        """
        self.s = s

        self.l = [hashlib.md5] * 5
        self.b = 256

        self.CMS = np.zeros((self.l, self.b), dtype = int)

    def inc(self, x:int, s:int):
        """
        Takes s as input.
        Where does s fit in incrementing the stuff?"""
        for i in range(self.b):
            # TODO: How should h(x)
            h_i = self.l[i](x) % s # hi(x) := MD5(x * i) mod b
            self.CMS[i][h_i] += 1 #


    def count(self, x:int, s:int) -> int:
        """
        Return the estimated count of an item in CMS
        """
        min_count = float('inf')

        for i in range(self.b):
            h_i = self.l[i](x) % s
            min_count = min(min_count, self.array[i][h_i])

        return min_count

        """
        The five hash functions h1,h2,h3,h4,h5: U  {0,1,   ,255}
        are computed as follows:

        For  x U , define hi(x) to be the i_th byte of the MD5 hash of
            str(x)  str  (s), where str(   )
        is the string representation of an integer and     corresponds to
            concatenation of strings.
```

```
    """

    # computing the five hash functions

def count_total_length():
    total_length = 0
    for i in range(1, 10):
        sum(i*(1000*(i+1) - 1000*i))

def heavy_hitter(array):
    """
    Implementing the heavy hitter algorithm that was computed earlier as
    a baseline.
    """
    counter = 0
    current = None
    length = len(array)
    for i in range(length):
        if counter == 0:
            current = array[i]
            counter += 1
        elif array[i] == current:
            counter += 1
        else:
            counter += -1
    return current
```

(b) Call an integer a *heavy hitter* if the number of times it appears in the data stream is at least 1% of the total length of the stream. How many heavy hitters are there in a stream with the above frequencies? **Answer = 24**

The two rules are as follows:

For each i=1,2,. . . ,9, each integer between 1000i+1 and 1000(i+1) appears i times in the stream.

Every integer i with 1i60, the number 2023+i appears an additional i2 times in the stream.

For the first rule, we have:

For i = 1, there are 1000 numbers from 1001 to 2000 that appears 1 time each.

For i = 2, there are 1000 numbers from 2001 to 3000 that appears 2 times each. And so on until

For i = 9, there are 1000 numbers from 9001 to 10000 that appear 9 times each. Thus, the total length contributed by the first rule is $1000 * \frac{9*(9+1)}{2} = 45000$.

The second rule provides us with the following: For i = 1, the number 2024 appears 1 time (1^2). For i = 2, the number 2025 appears 4 times (2^2). Until For i = 60, the number 2083 appears 3600 times (60^2). The sum of the second rule is $\frac{(60*(60+1)*(2*60+1))}{6} = 91080$.

45000 + 91080 = 136080. A heavy hitter appears at least 1% of 136080, which is 1361.

There is no number matching from first rule because the maximum appearance is 9, but according to the second rule, we can compute a number. We will find the number i that is $i^2 = 1361$. In this case, i = 36.86 or 37. According to the second rule, number that is higher than 37 should also be a heavy hitter. Thus, 60 - 37 + 1= 24.

(c)

(d)

(e) Among the five counters, only the smallest count is updated. This implementation appears already similar to how count finds the minimal value across the **5** hash functions. It would seem that if only a subset of counters are updated, then they have a chance in which some are not updated, which causes the count to stagnate, which causes some group of counts to become smaller and eventually smallest. If we have the additional rule saying that if few are tied for smallest, then we increment them all, then the count can eventually reach what they ought to have been.

(f)