1

Even though all bound inequalities are useful under different situations, I will stick with Chernoff's inequality. The reason is that it provides an upper bound on the probability that the sum of independent random variables deviates from its expected value by more than a certain amount, which is particular useful when you try to analyze streaming randomized algorithms where the sum of independent random variables is a common methodology to describe events of streams.

2

We start with some definitions. First, the random variable X_i is defined as:

$$X_i = \begin{cases} 1 & \text{if countMin sketch algorithm provide a good i-th estimate} \\ 0 & \text{if countMin sketch algorithm provide a bad i-th estimate} \end{cases}$$

Also, we define as p the probability of having a good estimate = outside of desired range. We run the countMin sketch n independent times, so we have $x_1, ..., x_n$ indepdendent variables, whose sum is defined as $X = x_1 + ... + x_n$ and it has an expected value of $E[X] = E[x_1 + ... + x_n] = n \cdot p$.

Using the hint provided, I will take the median instead of the minimum to get a good estimate. In particular, we denote the error probability that at least $\frac{n+1}{2}$ estimates are outside of the desired range as:

$$Pr(\text{At least } \frac{n+1}{2} \text{ estimates are outside of desire range}) = Pr(\sum_{i=1}^{n} x_i \geq \frac{n+1}{2}) \tag{1}$$

Using the Hoeffding inequality for sum of binomial random variables we get that (1):

$$\begin{aligned} Pr(\sum_{i=1}^{n} x_i \geq \frac{n+1}{2}) &= Pr(\sum_{i=1}^{n} x_i - E[X] \geq \frac{n+1}{2} - E[X]) \xrightarrow{E[X]=n\cdot p} \\ &= Pr(\sum_{i=1}^{n} x_i - E[X] \geq \frac{n+1}{2} - n \cdot p) \xrightarrow{\text{Hoeffding inequality}} \\ &\leq e^{\frac{-2 \cdot (\frac{n+1}{2} - n \cdot p)^2}{n}} \implies \\ &\leq e^{-2 \cdot n \cdot p^2} \leq e^{-n \cdot p^2} \end{aligned}$$

Therefore, if we bound the error probability with δ:

$$\begin{aligned} Pr(\text{At least } \frac{n+1}{2} \text{ estimates are outside of desire range}) \leq e^{-n\cdot p^2} < \delta \implies \\ e^{-n\cdot p^2} < \delta \implies \\ n > -\frac{ln(\delta)}{p^2} \end{aligned}$$

In conclusion, if we run countMin sketch with $n = -\frac{ln(\delta)}{p^2}$ independent times, and take the median of those estiamtes, then with probability at least $1 - \delta$ the estimate will be within the desire range.

3

a)

We are going to answer the question with same analysis as we did in countmin sketch algorithm. We define the ground truth counter of an item i as $\tilde{c_i}$ and the estimated one from countmin sketch as c_i. Also, we define error Err as the sum of deviations of all elements from their ground truth number, such as $Err = \sum_{i=1}^{n}(|c_i - \tilde{c_i}|)$. The probability that the error deviates from a small value a for a countmin sketch of one row is bounded by a small number called δ, such that $Pr(Err \geq a) = Pr(|\sum_{i=1}^{n}(c_i - \tilde{c_i})| \geq a) \leq \delta$. By markov inequality, this probability is bounded as follows:

$$Pr(|\sum_{i=1}^{n}(c_i - \tilde{c_i})| \geq a) \leq \delta \Longrightarrow$$
$$Pr(Err \geq a) \leq \delta \xrightarrow{\text{Markov inequality}}$$
$$Pr(Err \geq a) \leq \frac{E[Err]}{a} \leq \delta$$

The $E[Err]$ is the estimate value of overcounting buckets in a countmin sketch table with one row of k buckets is given as follows:

$$E[Err] = \sum_{y \neq \tilde{y}} C_{y,\tilde{y}} \cdot f_y = \frac{\sum_{y \neq \tilde{y}} f_y}{k} = \frac{S}{k}$$

where $C_{i,j}$ is a binomial random variable that is 1 when there is collision between items i and j and 0 otherwise, $S = \sum_{y \neq \tilde{y}} f_y$ is the sum of frequencies of all numbers in the stream.

Having said all that, based on the analysis we did in lectures, if we set $a = \epsilon \cdot S$ then the first equation takes the format:

$$Pr(Err \geq a) \leq \frac{E[Err]}{a} \leq \delta \Longrightarrow$$
$$Pr(Err \geq \epsilon \cdot S) \leq \frac{E[Err]}{\epsilon \cdot S} \leq \delta \xrightarrow{E[Err] = \frac{S}{k}}$$
$$\leq \frac{\frac{S}{k}}{\epsilon \cdot S} \leq \delta \Longrightarrow$$
$$Pr(Err \geq \epsilon \cdot S) \leq \frac{1}{\epsilon \cdot k} \leq \delta$$

Therefore, the total number of buckets k in one row can be expressed as $k \geq \frac{1}{\epsilon \cdot \delta}$. So, in conclusion the total number of buckets having a sketch table with one row having the same guarantees as the original countMin sketch algorithm is at least $\frac{1}{\epsilon \cdot \delta}$.

b)

We will follow the same idea for two rows. The countmin sketch table will have 2 rows and k buckets. Each row has a universal hash function. For each element e, algorithm will return the minimum value of the 2 hash functions as the estimation of the frequency of this element. So the probability of error Err will be the minimum of errors in row 1 and 2 respectively. The error for row 1 is defined as $E_1 = \sum_{i=1}^{n}(c_i - \tilde{c_i}$ and

for row 2 is given as $E_2 = \sum_{j=1}^{n}(c_j - \tilde{c_j}$, where c_i, c_j are the counters for elements i, j and $, \tilde{c_j}$ are the ground truth values of counters for elements i, j. The error probability is bounded as follows:

$$Pr(Err) = Pr(min(E_1, E_2)) = Pr(min[\sum_{i=1}^{n}(c_i - \tilde{c_i}), \sum_{j=1}^{n}(c_j - \tilde{c_j})] \geq a) \leq \delta \xrightarrow{a=\epsilon \cdot S}$$

$$Pr(min[\sum_{i=1}^{n}(c_i - \tilde{c_i}), \sum_{j=1}^{n}(c_j - \tilde{c_j})] \geq \epsilon \cdot S) \leq \delta \xrightarrow{\text{Independent universal hash functions}}$$

$$Pr(min(E_1, E_2)) = Pr(E_1) \cdot Pr(E_2) \leq (\frac{1}{\epsilon \cdot k})^2 \leq \delta \Longrightarrow$$

$$(\frac{1}{\epsilon \cdot k})^2 \leq \delta \Longrightarrow$$

$$\frac{1}{\epsilon \cdot k} \leq \sqrt{\delta} \Longrightarrow$$

$$k \geq \frac{1}{\epsilon \cdot \sqrt{\delta}}$$

Therefore, the total number of buckets k for 2 rows can be expressed as $k \geq \frac{1}{\epsilon \cdot \sqrt{\delta}}$. So, in conclusion the total number of buckets having a sketch table with two rows having the same guarantees as the original countMin sketch algorithm is at least $\frac{1}{\epsilon \cdot \sqrt{\delta}}$.

4

a)

The pseudo code is provided in python language and the data structures that were utilized are a dictionary and two lists.

```
import random
def heavy_hitters(stream, k):
    count_item_dict = {}
    candidates = []

    for item in stream:
        # Check if item exists increase counter,
        # otherwise create (key,value) pair with value = 1.
        if item in count_item_dict:
            count_item_dict[item] += 1
        else:
            count_item_dict[item] = 1

        # Check if number of items exceed parameter k to decrease counter
        # and if counter reaches zero remove item from data structure.
        if len(count_item_dict) > k:
            remove_list = []
            for key in count_item_dict:
                count_item_dict[key] -= 1
                if count_item_dict[key] == 0:
                    remove_list.append(key)
            for key in remove_list:
```

```
                del count_item_dict[key]

        # Remove an arbitrary item from data structure
        if len(count_item_dict) > k:
            count_item_dict.pop( random.choice(count_item_dict.keys()))

    # After checking all items, keep as candidates
    # those whose counter exceeds the below threshold.
    threshold = len(stream) / k
    for item in count_item_dict:
        if count_item_dict[item] > threshold:
            candidates.append(item)

    return candidates
```

b)

We define as linear sketch, a data structure that uses a fixed amount of memory to summarize a potentially large amount of data. In our case, the heavy hitters algorithm uses a dictionary to keep track of the counts of each element in the stream serving as the linear sketch in this case. To elaborate, the dictionary has a fixed size of at most k, and for each element in the stream, its count is incremented in the dictionary. When the dictionary size exceeds k, the algorithm decrements the count of each element in the dictionary by 1, and removes any elements with a count of 0. This effectively compresses the information about the stream into a fixed-size summary. The final step of the algorithm checks the count of each remaining element in the dictionary to see if it is greater than a threshold value, and adds it to the list of heavy hitters if so. This threshold value is calculated as n/k, where n is the total number of elements in the stream. This step effectively extracts the relevant information from the summary to identify the heavy hitters. For the above reasons, the heavy hitters algorithm is a linear sketch.

c)

We define an algorithm deterministic when it produces the same output for the same input every time it is run. In our case, a malicious user who knows the algorithm could deliberately choose a stream that causes the algorithm to fail. For example, if the malicious user defines his stream of 1000 items that each one has a frequency of 1. The dictionary size is defined as $size = 10$. This setup will give as a different number with a simple shuffling of the items in the input stream giving different heavy hitters for the same stream of number for 2 runs of the algorithm. an adversary could craft a stream that only contains two elements with very similar counts, and arrange the order of the stream such that the algorithm always keeps the wrong element as a heavy hitter. Therefore, since the heavy hitters algorithm is not randomized and its behavior is entirely deterministic, it is vulnerable to attacks by malicious users who can manipulate the input stream to cause it to fail.

5

a)

The space usage of the algorithm provided is $O(1)$, as it only needs to store a single variable y that represents a uniformly random element in $[n]$. Also, the variable c, which keeps track of the count of y in the stream, can be updated in constant space, so it does not contribute to the space complexity of the algorithm.

b)

In order to show that the estimator is unbiased we need to show that the output of the estimator is equal to the expected value of the estimator, so $E[X] = F_p$, where $X = n \cdot c^p$. The random variable is variable c which is the counter of the estimator and could be described per each element i as the value in the i-th position of the frequency vector $c_i = f_i$. The expected value of the output is given as follows:

$$\begin{aligned} E[X] &= \sum_{i=i}^{n} Pr(X = x_i) \cdot x_i \xrightarrow{x_i = n \cdot f_i^p} \\ &= \sum_{i=i}^{n} Pr(X = x_i) \cdot n \cdot f_i^p \xrightarrow{Pr(X = x_i) = \frac{1}{n}} \\ &= \sum_{i=i}^{n} \frac{1}{n} \cdot n \cdot f_i^p = \sum_{i=i}^{n} f_i^p = F_p \end{aligned}$$

At step 1, we rewrite the output x_i as a function of the frequency vector f at position i. At step 2, we take advantage of the uniformly random distribution of $[n]$. In conclusion, the estimator is unbiased.

c)

The algorithm is useful for the following reasons. Starting with the fact that it provides a space-efficient way to estimate frequency moments $O(1)$, which is a useful statistic for understanding the distribution of values in a stream and can be applied in a variety of domains such as machine learning, in which researchers might want to retrieve frequency moments of high-dimensional data vectors. This can be useful in large-scale data processing tasks, where the high dimentionality of the data makes it impractical to store the data vectors in memory.

6

a)

We define the probability distribution of $h'(x)$ as follows:

$$Pr(h'(x) = i) = Pr(\{g(x) \text{ is divisible by } 2^i\} \& \{g(x) \text{ is not divisible by } 2^{i+1}\})$$

In order to assume if function h' is pairwise independent we need to show that the join probability is equal to the multiplication of probabilities for the respected events. In particular,

$$Pr(h'(x) = i, h'(y) = j) = Pr(h'(x) = i) \cdot Pr(h'(y) = j)$$

Therefore, the joint probability distribution can be written based on the first equation as follows:

$$Pr(h'(x) = i, h'(y) = j) = Pr(\{g(x) \text{ is divisible by } 2^i\}\&\{g(x) \text{ is not divisible by } 2^{i+1}\},$$
$$\{g(y) \text{ is divisible by } 2^j\}\&\{g(y) \text{ is not divisible by } 2^{j+1}\})$$

Thanks to pairwise independence of function $g(\cdot)$ the previous equation can split as follows:

$$Pr(h'(x) = i, h'(y) = j) = Pr(\{g(x) \text{ is divisible by } 2^i\}\&\{g(x) \text{ is not divisible by } 2^{i+1}\})$$
$$*Pr(\{g(y) \text{ is divisible by } 2^j\}\&\{g(y) \text{ is not divisible by } 2^{j+1}\})$$

which can be rewritten thanks to the probability distribution of $h'(x)$ that we give as:

$$Pr(h'(x) = i, h'(y) = j) = Pr(h'(x) = i) \cdot Pr(h'(y) = j)$$

Therefore, $h'(x)$ is a pairwise independent function since it follows the rule that its join probability distribution function is a multiplication of the respected probabilities.

b)

Based on the fact that $g(x)$ is uniformly distributed on $\{1, ..., 2^k\}$, we know that $\forall x \in X$ and $\forall i \in N$ the probability p of $g(x)$ is divided by 2^i is $p = \frac{1}{2^i}$. Hence:
For $i < k$:

$$Pr(h'(x) = i) = Pr(\text{g(x) is divisible by } 2^i \text{ but not } 2^{i+1}) \Longrightarrow$$
$$= Pr(\text{g(x) is divisible by } 2^i) - Pr(\text{g(x) is not divisible by } 2^{i+1}) \Longrightarrow$$
$$= \frac{1}{2^i} - \frac{1}{2^{i+1}} = \frac{1}{2^{i+1}} = 2^{-(i+1)}$$

For $i = k$:

$$Pr(h'(x) = i) = Pr(\text{g(x) is divisible by } 2^i \text{ but not } 2^{i+1}) \Longrightarrow$$
$$= Pr(\text{g(x) is divisible by } 2^i) - Pr(\text{g(x) is not divisible by } 2^{i+1}) \Longrightarrow$$
$$= \frac{1}{2^k} - 0 = 2^{-k}$$

For $i > k$:

$$Pr(h'(x) = i) = Pr(\text{g(x) is divisible by } 2^i \text{ but not } 2^{i+1}) \Longrightarrow$$
$$= Pr(\text{g(x) is divisible by } 2^i) - Pr(\text{g(x) is not divisible by } 2^{i+1}) \Longrightarrow$$
$$= 0$$

So the probability distribution of $h'(x)$ is given as:

$$Pr(h'(x) = i) = \begin{cases} 2^{-(i+1)} & i < k \\ 2^k & i = k \\ 0 & i > k \end{cases}$$

Related to the comparison between the two probability distributions of h' and h, we can see that $Pr(h'(x) = i) = Pr(h(x) = i) = 2^{-(i+1)}$ when $i < k$. However, there is a difference in cases where i is greater equal to k where the value of probability distribution drops to zero.

c)

We compare the functions $h'(x)$ and $h(x)$ which are defined as:

$$Pr(h'(x) = i) = \begin{cases} 2^{-(i+1)} & i < k \\ 2^k & i = k \\ 0 & i > k \end{cases}$$

and $Pr(h(x) = i) = 2^{-(i+1)}$. It is obvious that the probability distribution of $h(x)$ follows a decreasing geometrical distribution which takes values close to zero as we increase the number i, whereas the the probability distribution of $h'(x)$ is zero for large i numbers. Moreover, we can see that for values $i < k$ the probabilities are the same as well, so in general h' and h share similar probability distribution functions.

7

a)

The algorithm is quite simple. We increase the counter no matter what is the element. Since the values are from Z space then increasing the counter with positive values is additions while decreasing the counter with negative values is deletions. At the end, we are promised that one element will be left in S, so the counter will have exactly this number. In particular, algorithm's code in Python is the following:

```
    def algorithm_7a(stream):
    # Increase the counter with the updates of the stream
    # Additions will be positive values increasing the counter
    # Deletions will be negative values decreasing the counter
    # At the end, only one item will be left as it is quaranteed from the algorithm
    definition.
    counter = 0
    for item in stream:
        counter += item
    output_element = counter
    return output_element
```

The space complexity calculation is simple. Only a counter is utilized so the space complexity is constant $O(1)$.

b)

The algorithm is simple. We will calculate 2 sums, one is called "expected sum" and the other "calculated sum". In particular, the first is the expected sum of all items from 1 to n, which is given as $n\frac{(n+1)}{2}$, while the second is the sum we will calculate as we pass across the elements. At the end, we subtract the "calculated sum" with the "expected sum" and the difference will be the extra element that is double in our stream. More specifically, algorithm's code in Python is the following:

```
def algorithm_7b(stream):
    # n defined the total unique elements of the stream
    n = len(stream) - 1
    expected_sum = n * (n-1) / 2
    calculated_sum = 0
```

```
    for item in stream:
        calculated_sum += item
    repeated_element =  calculated_sum - expected_sum
    return repeated_element
```

The space complexity calculation is simple. No data structure was created rather than 3 variables to hold the sums and the output value, which require a constant space complexity Therefore, the total space complexity is O(1).

8

a)

I read the section titled "Collisions (the Birthday Paradox)" in the handout on probabilistic inequalities as requested.

b)

Reference link for my data is here. I utilized euro to dollar exchange. It took 12 days to find a pair of days with exchange rate that has the 2 least significant digits same. At this point I have to mention that those data are not distributed uniformly since there is a bias with the holiday days, where the stock market is closed. I noticed that especially on Christmas days.

Day	EUR-USD
Monday 20 February 2023	**€1 EUR = $1.0683**
Sunday 19 February 2023	**€1 EUR = $1.0685**
Saturday 18 February 2023	**€1 EUR = $1.0716**
Friday 17 February 2023	**€1 EUR = $1.0721**
Thursday 16 February 2023	**€1 EUR = $1.0665**
Wednesday 15 February 2023	**€1 EUR = $1.0692**
Tuesday 14 February 2023	**€1 EUR = $1.0735**
Monday 13 February 2023	**€1 EUR = $1.0731**
Sunday 12 February 2023	**€1 EUR = $1.0678**
Saturday 11 February 2023	**€1 EUR = $1.0688**
Friday 10 February 2023	**€1 EUR = $1.0688**

c)

We will rephrase the Birthday paradox based on this example as follows: Our k independent samples $x_1, ..., x_k$ from the uniform distribution on $\{1, ..., n\}$ are the days. As n we define the total different digits we can get for the 2 least significant bits, therefore n = 99 so the uniform distribution get samples from a distribution with 100 elements. Given all that: the Birthday Paradox theorem defines that if $k \geq 2 \cdot \lceil \sqrt{n} \rceil = 20$, in other words if we check at least 20 days then the probability of collision (find two days with an exchange rate with the same least 2 significant bits) in this set of samples is at least $\frac{1}{2}$. I repeat the experiment for 4 currency exchange more. (EUR-GBP, EUR-CNY, EUR-INR and EUR-CHF). Even though the least two

significant bits for my experimental data, do not follow uniform distribution due to holiday same exchange value, the total number of days to find two days with the same 2 least significant bits are 23, 17, 18 and 27 respectively. It looks that if I run the experiment over multiple different currencies the expected days of finding at least one collision falls over a range close to 20 days, which is matching the theory of Birthday paradox.

9

We define the probability of an event E at an experiment as p, therefore the probability of failure is $1-p$. If we run the experiment k independent times then the probability of failure to get this event E is defined as $(1-p)^k$, while the probability of successfully find this event E is $1-(1-p)^k$.

Now, we will bound the success probability of getting the event E at least one time by a constant value z and solve for the inequality for k, which defines the total number of independent times we need to run the experiment to get the event E. In particular:

$$
\begin{aligned}
1-(1-p)^k \geq q \geq 0 \implies \\
1-q \geq (1-p)^k \xrightarrow{ln() \nearrow} \\
ln(1-q) \geq k \cdot ln(1-p) \xrightarrow{0 \leq p \leq 1} \\
k \geq \frac{ln(1-q)}{ln(1-p)} \xrightarrow{(1-x) \leq e^{-x}} \\
k \geq -\frac{ln(1-q)}{p}
\end{aligned}
$$

Therefore, the total number of independent times we need to run is at least $-\frac{ln(1-q)}{p}$ and since q is a constant probability then the whole value of $ln(1-q)$ is constant. So overall, the independent times needed to run the experiment just to see E occur at least once with constant probability q is $\Omega(\frac{1}{p})$.

10

I spent 3-4 days in total to answer the questions and write my solutions. The homework was neither easy nor hard. It had both easy questions and hard questions to answer.