1. (14 pt.) [Another way to sketch sparse vectors.] Suppose that $A$ is an list of length $n$, containing elements from a large universe $U$. Our goal is to estimate the frequencies of each element in $U$: that is, for $x \in U$, how often does $x$ appear in $A$?

The catch is that $A$ is too big to look at all at once. Instead, we see the elements of $A$ one at a time: $A[0], A[1], A[2], \ldots$. Unfortunately, $U$ is also really big, so we can’t just keep a count of how often we see each element.

In this problem, we’ll see a construction of a randomized data structure that will keep a “sketch” of the list $A$, use small space, and will be able to efficiently answer queries of the form “approximately how often did $x$ occur in $A$?”

Specifically, our goal is the following: we would like a (small-space) data structure, which supports operations $\text{update}(x)$ and $\text{count}(x)$. The $\text{update}$ function inserts an item $x \in U$ into the data structure. The $\text{count}$ function should have the following guarantee, for some $\delta, \epsilon > 0$. After calling $\text{update}$ $n$ times, $\text{count}(x)$ should satisfy

$$C_x \leq \text{count}(x) \leq C_x + \epsilon n$$  \hspace{1cm} (1)

with probability at least $1 - \delta$, where $C_x$ is the true count of $x$ in $A$.

(a) (3 pt.) Your friend suggests the following strategy (this will not be our final strategy). We start with an array $R$ of length $b$ initialized to 0, and a random hash function $h : U \rightarrow \{0, 1, \ldots, b - 1\}$. You can assume that $h$ is drawn from some universal hash family, i.e $P(h(x) = h(y)) = 1/b$ for any $x \neq y$. Then the operations are:

- $\text{update}(x)$: Increment $R[h(x)]$ by 1.
- $\text{count}(x)$: return $R[h(x)]$.

For every entry $A[i]$ in the list it encounters, the scheme calls $\text{update}(A[i])$.

After sequentially processing all $n$ items in the list, what is the expected value of $\text{count}(x)$?

(b) (2 pt.) Show that there is a choice of $b$ that is $O(1/\epsilon)$ so that, for any fixed $x \in U$, we have

$$\Pr[\text{count}(x) < C_x] = 0$$

and

$$\Pr[\text{count}(x) \geq C_x + \epsilon n] \leq \frac{1}{e}.$$  

[HINT: The first of the requirements is true no matter what $b$ is.]

(c) (2 pt.) Explain how you would use $T$ copies of the construction in part (a) to define a data structure that, for any fixed $x \in U$, satisfies (1) with high probability. How big do you need to take $T$ so that the (1) is satisfied with probability at least $1 - \delta$? How much space does your modified construction use? (It should be sublinear in $|U|$ and $n$).
Give a complete description and analysis of the data structure, and explain how much space it uses. You may assume that it takes $O(\log |\mathcal{U}|)$ bits to store the hash function $h$ and $O(\log n)$ to store each element in the array $R$.

(d) Explain how to use your algorithm to solve the following problem:

i. (4 pt.) Given a $k$-sparse vector $a \in \mathbb{Z}_0^N$ ($\mathbb{Z}_0$ is the set of non-negative integers), design a randomized matrix $\Phi \in \mathbb{R}^{m \times N}$ for $m = O\left(\frac{k \log N}{\epsilon^2}\right)$ so that the following happens. With probability at least 0.99 over the choice of $\Phi$, you can recover $\tilde{a}$ given $\Phi a$, so that simultaneously for all $i \in 1, \ldots, N$, we have

$$|\tilde{a}[i] - a[i]| \leq \frac{\epsilon \|a\|_1}{2k}.$$  

[HINT: Think of the $k$-sparse vector $a$ as being the histogram of the items in the list $A$ from the previous parts.]

[HINT: How can you represent a hash function as a matrix multiplication?]

[HINT: Note that we want a tighter bound, and we want the bound to hold simultaneously for all $i$. How can we change $b$ and $T$ to achieve this?]}

ii. (3 pt.) Now, assuming the above holds for all $i$, use the $k$-sparseness of $a$ to construct $\hat{a}$ from $\tilde{a}$ such that

$$\|\hat{a} - a\|_1 \leq \epsilon \|a\|_1.$$ 

iii. (0 pt.) [This question is zero points, but worth thinking about.] How does the guarantee in the previous part compare to the RIP matrices (and the compressed sensing guarantee that we can get from them, Theorem 1 in the Lecture 9 lecture notes) that we saw in class? (i.e., is this guarantee weaker? Stronger? Incomparable? The same?)

2. (10 pt.) [The probabilistic method for coding bounds.]

(a) (3 pt.) Suppose that we have a finite collection $C$ of finite strings from an alphabet $\Sigma$ of size $a$, such that no string in $C$ is a prefix of another one. Let $m$ be the maximum length of any string in $C$. If $N_i$ is the number of strings of length $i$ in $C$, prove that

$$\sum_{i=1}^{m} \frac{N_i}{a^i} \leq 1.$$  

[HINT: Use the probabilistic method. Consider a random string $x_1 x_2 x_3 \cdots$ where each $x_i \in \Sigma$ is chosen uniformly at random. What is the expected number of prefixes of this string (that is, strings that look like $x_1 x_2 \ldots x_k$ for some $k$) that are contained in $C$?]

[Comment: This gives a bound on the number/composition of length-$i$ strings that can be in any prefix-free code, which you may have seen in CS161.]

(b) (7 pt.) Now suppose that we have a finite collection $C$ of finite strings from an alphabet $\Sigma$ of size $a$ such that no two distinct concatenations of two finite sequences of strings from $C$ are the same. For example, $C$ could not contain all of the strings $abc, de, fg, abcd, efg$, because the concatenation of the first three is equal to the concatenation of
the last two: \(abc \circ de \circ fg = abcd \circ efgh\). Let \(m\) be the maximum length of any string in \(C\).

If \(N_i\) is the number of strings of length \(i\) in \(C\), we will prove in the parts below that

\[
\sum_{i=1}^{m} \frac{N_i}{a^i} \leq 1
\]

**Comment:** These types of sets are useful for encoding words. For example, if we want to include English words in binary, and we map each character ‘a’, ‘b’, ‘c’... to a different string in \(C\) (where \(\Sigma = \{0, 1\}\)), this ensures that no two words will encode to the same string. This result gives a bound on the number/composition of length-\(i\) strings in such a code.

i. (1 pt.) Let \(p_{k,j}\) be the probability a uniformly randomly generated string of length \(j\) is the concatenation of a sequence of \(k\) strings from \(C\). Prove that

\[
\sum_{j=1}^{m} p_{1,j} = \sum_{i=1}^{m} \frac{N_i}{a^i}
\]

ii. (3 pt.) Now prove that

\[
\sum_{j=1}^{2m} p_{2,j} = \left( \sum_{i=1}^{m} \frac{N_i}{a^i} \right)^2
\]

[HINT: How does each term \(\frac{N_i N_j}{a^{i+j}}\) from the RHS contribute to \(p_{2,i_1+i_2}\)?]

iii. (0 pt.) [Optional: This part will not be graded. You can use the results of this part in future parts.]

Convince yourself that this generalizes, i.e.

\[
\sum_{j=1}^{km} p_{k,j} = \left( \sum_{i=1}^{m} \frac{N_i}{a^i} \right)^k
\]

for any positive integer \(k\).

iv. (1 pt.) Explain why \(\sum_{j=1}^{km} p_{k,j} \leq mk\).

[HINT: This is not a trick question, don’t over-think it.]

v. (2 pt.) Combine the bounds in parts (iii) and (iv) to arrive at the conclusion that

\[
\sum_{i=1}^{m} \frac{N_i}{a^i} \leq 1
\]