Problem 1: (Chernoff Bounds via Negative Dependence - from MU Ex 5.15)

While deriving lower bounds on the load of the maximum loaded bin when \( n \) balls are thrown in \( n \) bins, we saw the use of negative dependence. We now consider another example, where this technique can be used to derive Chernoff-style bounds for the number of empty bins.

Suppose \( n \) balls are thrown in \( n \) bins, and let \( \{X_i\}_{i \in [n]} \) be a collection of indicator r.v.s indicating whether bin \( i \) is empty (i.e., \( X_i = 1 \) iff bin \( i \) has 0 balls). On the other hand, let \( \{Y_i\}_{i \in [n]} \) be a set of i.i.d. Bernoulli r.v.s which are 1 with probability \((1 - 1/n)^n\).

Part (a)

For any \( k \geq 1 \), show that \( E[X_1 X_2 \ldots X_k] \leq E[Y_1 Y_2 \ldots Y_k] \).

Part (b)

Let \( X = \sum_{i=1}^{n} X_i \) and \( Y = \sum_{i=1}^{n} Y_i \). Using the above result, prove that for any \( \theta \geq 0 \), we have:
\[
E[e^{\theta X}] \leq E[e^{\theta Y}]
\]

Hint: Think of the Taylor series of the exponential function.

Part (c)

Finally, using this result, state a Chernoff bound for \( P[X \geq (1 + \epsilon)E[X]] \).

(You can use bounds you know from before without re-deriving them).

Problem 2: (Bucket Sort)

Suppose we are given \( n = 2^m \) elements, each of which are \( k \) bit sequences drawn uniformly at random from \( U = \{0, 1\}^k \) (where \( k \geq m \)). We’ll now consider a simple deterministic algorithm for sorting these, that takes \( O(n) \) time on average. First, we place place each element in one of \( m \) buckets, where the \( j \)th bucket \((j \in \{0, 1, \ldots, 2^m - 1\}) \) is used to place all elements whose first \( m \) bits correspond to the number \( j \). Next, we use any sorting algorithm with quadratic running time (for example, a simple bubble sort or insertion sort) to sort the elements in each bucket, and then merge the buckets. Prove that the expected running time of this algorithm is \( O(n) \).

Hint: Recall the analysis of the FKS hashing scheme.

Problem 3: (Open Addressing)

In class, we saw the chaining technique for designing hash tables for answering exact set-membership (i.e., without allowing for false-positives). Another common approach is that of open-addressing, where given a set \( S \) of \( m \) items, we hash the elements in a single array of length > \( m \). Each entry in the array either contains an element from \( S \), or is empty. The hash function defines for each element \( x \in U \), a probe sequence \( \{h(x, 1), h(x, 2), \ldots\} \). To insert an element \( x \) in the array, we first check position \( h(x, 1) \) – if this is occupied, we try to insert it in \( h(x, 2) \), and so on till we find an open cell in the array.
Part (a)

Suppose we use an array of length $2m$ to store $m$ items, and suppose each hash-function $h(x, i)$ is independent and uniform over $\{0, 1, \ldots, 2m - 1\}$. Show that for any of the first $m$ elements to be inserted, the insertion required more than $k$ probes with probability $\leq 2^{-k}$ – hence show that the probability that the $i^{th}$ insertion (for $i \leq m$) took more than $2 \log m$ probes is less than $1/m^2$.

Part (b)

Next, let $X$ be the maximum number of probes required by an item during insertion of the first $m$ items. Show that $X$ is less than $2 \log m$ with probability at least $1 - 1/m$. Using this, also show that the $E[X]$ is $O(\log m)$.

Problem 4: (Extensions of Bloom Filters)

In class we saw the basic Bloom filter, where we used $k$ independent random hash-functions $\{h_1, h_2, \ldots, h_k\}$ to hash a set $S$ of $m$ elements into an array $A$ of $n$ bits. Recall that in order to get a false-positive rate of $\delta = O(1)$, we chose $n = cm$, for some constant $c$, and $k c \ln 2$ (in particular, for false-positive rate of 2%, we used $c = 8$ and $k = 6$). We now see how this basic structure can be modified in various ways.

Part (a)

In order to support item deletions in addition to insertions and look-ups, we can replace each bit $A[i]$ in $A$ with a counter – when an element is hashed to bucket $i$, we increment $A[i]$, and to delete an element $x$, we decrement the counter for each $A[i]$ corresponding to $\{h_1(x), h_2(x), \ldots, h_k(x)\}$. As before, if we use $n = O(m)$ and fixed-size counters of $b$-bits. What is the probability that counter $A[i]$ overflows after inserting $m$ elements? Also argue that $O(\log \log m)$-bit counters are necessary and sufficient to prevent overflow in any counter (with high probability).

Part (b)

Suppose we use the same hash functions $\{h_1, h_2, \ldots, h_k\}$ to hash two separate sets $S_1$ and $S_2$ (both of size $m$) – let the resulting Bloom filters (each of $n$ bits) be $A_1$ and $A_2$ respectively. Suppose we create a new Bloom filter $A_{OR}$ by taking the bit-wise OR of the bits of $A_1$ and $A_2$. Is this the same as the Bloom filter constructed by adding the elements of $S_1 \cup S_2$ one at a time?

Part (c)

Suppose we create another new Bloom filter $A_{AND}$ by taking the bit-wise AND of the bits of $A_1$ and $A_2$. Argue that this is not the same as the Bloom filter constructed by adding the elements of $S_1 \cap S_2$ one at a time. However, also argue that $A_{AND}$ can be used to check if $x \in S_1 \cap S_2$ with one-sided error (i.e., give an algorithm that always returns TRUE if $x \in S_1 \cap S_2$), and explain how we can get false-positives.
Problem 5: (Similarity functions with no linear-LSH family)

In class we discussed locality sensitive hashing for the Hamming and Jaccard similarity functions. Recall that for a ground set $U$ and subsets $A, B \subseteq U$, these two distances corresponded to:

\[ s_{\text{Hamming}}(A, B) = 1 - \frac{|A \Delta B|}{|U|}, \quad s_{\text{Jaccard}}(A, B) = \frac{|A \cap B|}{|A \cup B|}, \]

where $A \Delta B$ is the symmetric difference between sets $A$ and $B$ (i.e., $A \Delta B = (A \cup B) \setminus (A \cap B)$). Moreover, in both cases, we obtained families of hash-functions $H$ satisfying:

\[ P[h(x) = h(y)] = s(x, y) \]

A natural question to ask is if such linear-LSH families exists for other similarity functions, in particular, for two other natural subset-similarity measures – the Overlap and Dice similarities:

\[ s_{\text{Overlap}}(A, B) = \frac{|A \cap B|}{\min\{|A|, |B|\}}, \quad d_{\text{dice}}(A, B) = \frac{2|A \cap B|}{|A| + |B|} \]

Part (a)

As in class, suppose we define a distance function $d : U \times U \to [0, 1]$ corresponding to a similarity function as $d(x, y) = 1 - s(x, y)$. Show that for a given similarity function $s$, if we have a linear-LSH family $H$, i.e., whose hash functions satisfy $P[h(x) = h(y)] = s(x, y)$, then the distance functions must obey the triangle inequality, i.e., for any $x, y, z \in U$, we must have:

\[ d(x, y) + d(y, z) \geq d(x, z) \]

Part (b)

Using the above result, prove that the Overlap and Dice similarity functions can not have a linear-LSH family.