(1) Exercise 11.2-1 on Page 261 [40 points]

11.2-1
Suppose we use a hash function \( h \) to hash \( n \) distinct keys into an array \( T \) of length \( m \). Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of \( \{ \{ k, l \} : k \neq l \text{ and } h(k) = h(l) \} \)?

For each pair of keys \( k, l \), where \( k \neq l \), define the indicator random variable \( X_{kl} = I \{ h(k) = h(l) \} \). Since we assume simple uniform hashing, \( \Pr\{X_{kl} = 1\} = \Pr\{h(k) = h(l)\} = 1/m \), and so \( E[X_{kl}] = 1/m \).

Now define the random variable \( Y \) to be the total number of collisions, so that

\[
Y = \sum_{k \neq l} X_{kl}
\]

The expected number of collisions is

\[
E[Y] = E\left[ \sum_{k \neq l} X_{kl} \right] = \sum_{k \neq l} \frac{1}{m} = \frac{n(n-1)}{2m}
\]

(2) Exercise 11.2-5 on Page 261 [30 points]

11.2-5
Suppose that we are storing a set of \( n \) keys into a hash table of size \( m \). Show that if the keys are drawn from a universe \( U \) with \( |U| > nm \), then \( U \) has a subset of size \( n \) consisting of keys that all hash to the same slot, so that the worst-case searching time for hashing with chaining is \( \Theta(n) \).

If \( |U| > nm \), then, by the Pigeonhole Principle, one slot must have more than \( nm/m = n \) keys.

So, searching that slot using linear search (since there’s no guarantee that the list of keys is ordered) takes \( \Theta(n) \). Thus, the worst-case searching time for hashing with chaining is \( \Theta(n) \).
Exercise 11.3-1 on Page 268 [30 points]

11.3-1
Suppose we wish to search a linked list of length $n$, where each element contains a key $k$ along with a hash value $h(k)$. Each key is a long character string. How might we take advantage of the hash values when searching the list for an element with a given key?

Each key is a long character thus to compare keys, at every node we need to perform a string comparison operation which is very time consuming.

Instead we generate a hash value for the key (i.e., generate a numeric value for each string) we are searching for and comparing hash values $h(k)$ along the length of the list, which turns out to be numeric values and the comparison is faster. If the hash value is the same, then we compare the key itself.