6. This problem analyzes a hashing method called **uniform hashing**. We store the \( n \) keys in an array with \( m \) slots. We also have a sequence of hash functions, \( h_1, h_2, \ldots, h_m \). For all keys \( y \), the hash addresses \( h_1(y), h_2(y), \ldots, h_m(y) \) form a permutation of \( \{1, 2, \ldots, m\} \). To search for a key \( y \):

1. Set \( i \leftarrow 1 \).
2. Examine slot \( h_i(y) \). If nothing is there, then insert \( y \) there and \textbf{STOP [unsuccessful search]}.
3. Otherwise, if slot \( h_i(y) \) contains \( y \), \textbf{STOP [successful search]}. Else, increment \( i \) by 1 and return to Step 2.

The number of \textit{probes} in a search is the number of slots that are examined. This is the number of times Step 2 is executed.

The best known hashing method is called **linear probing**: it is a special case of uniform probing. In linear probing, we only have to compute the first hash function \( h_1(y) \). The search for \( y \) starts at slot \( h_1(y) \) and proceeds cyclically through the table. That is, the other hash functions are defined implicitly by:

\[
h_{i+1}(y) = \begin{cases} 
h_i(y) - 1 & \text{if } h_i(y) > 1; \\
\frac{m}{h_i(y)} & \text{if } h_i(y) = 1.
\end{cases}
\]
This method is hard to analyze, because the hash functions \( h_2, h_3, \ldots \), are not independent of \( h_1 \). Instead, we will analyze a simpler model: We assume that for each key \( y \), the order in which we probe the table \( h_1(y), h_2(y), \ldots, h_m(y) \) is a random permutation of \( \{1, 2, \ldots, m\} \).

Let \( Y_n \) be the random variable describing the number of probes per unsuccessful search (or insertion) when there are \( n \) keys already in the hash table. For any given configuration of the \( n \) keys, we have

\[
p_{nk} = \Pr\{Y_n = k\}
= \Pr\{\text{slots } h_1(y), \ldots, h_{k-1}(y) \text{ are occupied and slot } h_k(y) \text{ is not occupied}\}.
\]

Let \( S \) be the set of locations of the \( n \) inserted keys. Then

\[
p_{nk} = \frac{1}{m!} \left( \# \text{ permutations s.t. } \{h_1(y), \ldots, h_{k-1}(y)\} \subset S \text{ and } h_k(y) \notin S \right)
= \frac{1}{m!} (n(n-1) \cdots (n-k+2)(m-n)(m-k)!) \]
= \frac{1}{m!} n^{k-1} (m-n)(m-k)!
= \frac{n^{k-1}}{m^k} (m-n).
\]

(a) For \( n < m \), compute

\[
E(Y_n) = \sum_{1 \leq k \leq n+1} kp_{nk}
\]
using the techniques of Chapter 5. (As a check, your final answer should be \((m+1)/(m-n+1)\); don’t use induction.)

(b) Let \( X_n \) be the random variable for the number of probes per successful search when there are \( n \) inserted keys. Using the same argument that we used in class for the analysis of the separate chaining hashing scheme, a successful search for a key \( y \) takes the same number of probes as when it was inserted (after a prior unsuccessful search). Hence, the average number of probes in a successful search is the average of the expected number of probes for each of the \( n \) unsuccessful searches (insertion); that is,

\[
E(X_n) = \frac{1}{n} \sum_{1 \leq k \leq n} E(Y_{k-1}).
\]

Compute \( E(X_n) \) using part (a).