1. Permute by sorting, again.
Do problem 5.3-5.

2. No collisions.
Find the largest value of $n$ that you can, such that if $n$ items are stored independently in random locations in a hash table of size $m$, \[ \Pr[ \text{there are no collisions} ] \geq 1/2 \]

3. Duplicate messages.
A series of $n$ messages are broadcast on a communications channel. Your job is to determine if any of them are duplicates. With a very small probability $- 1/n^{10}$ - you may incorrectly report that two different messages are duplicates, but you should never miss a duplicate if it occurs.

The messages are far to long to store, but you have a flexible-size hash function $h(x, b)$ which takes a message $x$ and number of bits $b$ as input, and produces an integer in the range $0 \ldots 2^b - 1$. We’ll assume (unrealistically) that the integers produced by $h(x, b)$ are uniformly random and all independent of each other.

How few bits of memory can you use to solve this problem? Note that making a hash table of size $m$, with $k$-bit entries, counts as $mk$ bits, even if most of the entries are empty.

4. Exponential backoff.
This problem is related to the idea of exponential backoff, which is part of the ethernet protocol.

We have a large number of computers sharing a communications channel, and $n$ of them have a packet of data to send. Time is divided into intervals $t_0, t_1, \ldots$. Any one packet can be sent in a single time interval, but if two or more packets are sent during the same interval $t_i$ the packets collide, and none of them is successfully transmitted. The computers can only detect whether their packet is successfully transmitted or not; they do not otherwise communicate with any of the other computers, and they do not know $n$ or the total number of computers on the channel.

The computers use the following protocol. They all try to transmit at $t_1$; assuming $n > 1$ there will be a collision. We call this round zero. In round 1, each computer then chooses randomly whether to transmit during either $t_2$ or $t_3$. If it fails again to transmit, it continues to round 2. In general, if a computer fails at time $t_i$, $2^k \leq i < 2^{k+1}$ for some integer $k$, it picks a new time $t_j$ at random, from the range $2^{k+1} \leq j < 2^{k+2}$, and tries again at $t_j$. The process continues until all of the $n$ computers succeed in transmitting their packets.

Give the best upper bound you can on the number of rounds which will be required for every computer to transmit its message.