Problem 1 (Card Sharks)  Consider a standard deck of 52 cards.

(a) What is the probability of drawing a face card (joker, queen, king, ace) when drawing a single card from the deck?
(b) What is the probability of drawing a face card three times in a row if we draw three cards from the deck, replacing the drawn card every time?
(c) What is the probability of drawing a five card straight when drawing a hand of five cards from the deck?
(d) What is the probability of drawing a pair of 10s when drawing a hand of five cards from the deck? (The other three cards can be anything.)
(e) What is the probability of drawing a full house (a pair and a three-of-a-kind) when drawing five cards from the deck?

In all of these cases, do not report a final probability value but instead give a combinatorial formula (i.e., the unsimplified expression) denoting the desired probability.

Problem 2 (Expectation)  Most states in America run weekly lotteries to raise funds. The most common of these is Powerball (the following description comes from the Iowa Lottery website\textsuperscript{1}:

Powerball costs $2 per play; $3 with the Power Play\textsuperscript{®} option.
Choose 5 out of 69 numbers for the white balls; then choose 1 out of 26 numbers for the Powerball [\textit{ed}: which are red for dramatic effect].

\begin{itemize}
  \item The jackpot begins at $40 million and grows until it’s won.
  \item Win the jackpot by matching all 5 white ball numbers and the Powerball.
\end{itemize}

In addition to the jackpot, you can win smaller amounts of money by matching fewer balls. In particular, you can win for:

- Matching only the powerball (0+PB)
- Matching 1 white ball and the powerball (1+PB)
- Matching 2 white balls and the powerball (2+PB)
- Matching 3 white balls (3)
- Matching 3 white balls and the powerball (3+PB)
- Matching 4 white balls (4)
- Matching 4 white balls and the powerball (4+PB)

\textsuperscript{1}http://www.ialottery.com/Pages/Games-Online/Powerball.aspx
• Matching 5 white balls (5)
• Matching 5 white balls and the powerball (5+PB, the jackpot)

As required by law, the Iowa Lottery must publish the estimated likelihood of each of these outcomes which can be found on the Iowa Lottery website linked above.

(a) Let $X$ be the random variable denoting the payoff for playing a single game of Powerball without the Power Play® option. Based on the published data from the Iowa Lottery, compute $E[X]$, the expected payoff of a single game of Powerball. Give both an unsimplified, combinatorial formula for the expected value as well as a final, computed result. Assume that the jackpot is at its starting value of $40,000,000$, and that you would take the annuity-based reward for the jackpot instead of the cash-only option (which reduces the jackpot payoff by approximately a half). Furthermore, assume that you will be the only one winning the jackpot on this play, i.e., don’t worry about the case where you have to split the jackpot with other people.

(b) All of the payoff amounts for Powerball are fixed except the jackpot which starts at a minimum value $k = 40,000,000$ and grows every week if no one wins the jackpot amount. When someone wins the jackpot, the value $k$ is reset to the minimum. What value of $k$ is necessary to expect to break even from a single play of Powerball without the Power Play® option.

(c) With the Power Play® option, you may pay one more dollar (entry cost of $3$ overall) to multiply your non-jackpot payoffs by some random multiplier ($2$, $3$, $4$, $5$, or $10$). The multipliers are chosen at random during the drawing of the numbers, i.e., you do not know in advance which of your multipliers will apply. Assume that the random multipliers are equally likely to be drawn. Let $Y$ be the random variable denoting the payoff for playing a single game of Powerball with the Power Play® option. What is $E[Y]$? Based on this data, is it worthwhile to add the Power Play® option when you play Powerball?

**Problem 3 (Faulty)** Consider a distributed system where a set of four computers $A$, $B$, $C$, and $D$ contribute computations towards a shared task. The computers have different specs and thus different rates of output. Furthermore, the computers each have different rates of failure when performing a computation (due to programming errors, hardware failure, or other craziness). The following table summarizes the output and failure rates of the machines:

<table>
<thead>
<tr>
<th>Computer</th>
<th>Output Rate</th>
<th>Failure Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>30%</td>
<td>2.5%</td>
</tr>
<tr>
<td>$B$</td>
<td>20%</td>
<td>1.5%</td>
</tr>
<tr>
<td>$C$</td>
<td>40%</td>
<td>2.5%</td>
</tr>
<tr>
<td>$D$</td>
<td>10%</td>
<td>1.0%</td>
</tr>
</tbody>
</table>

(a) Suppose we choose a random output from the distributed system. What is the probability that it was produced by computer $B$?

(b) Suppose that we choose a random output from the distributed system and find that it is incorrect. What is the probability that it was produced by computer $B$?

**Problem 4 (Probabilistic Sampling)** Consider the problem of randomly choosing a set of $k$ elements from a collection of elements all with equal probability. If the size $n$ of the collection is known, then the problem amounts to choosing an (unordered) $k$-sized subset of the $n$ indices of the collection. However, if
the size of the collection is not known in advance, e.g., the collection is generated live and streamed over the network, then this technique no longer works.

One approach towards solving this problem is to create an array of size \( k \) and treat it like a kind of queue. You initially fill the queue with the first \( k \) elements from the collection and then each additional element you encounter has a random chance of bumping one of the elements from the array. After considering all of the elements from the collection, the array is your set of sampled elements. However, how can we ensure that our random sampling ensures that all of the elements of the collection have equal probability of ending up in the sample?

Consider the following algorithm presented in Java/C-like notation:

```java
T[] sample(int k, T[] collection) {
    T[] output = new T[k]; // output is an array of size k
    for (int i = 0; i < k; i++) {
        output[i] = collection[i];
    }
    for (int i = k; i < collection.length; i++) {
        int j = randint(i); // a random integer in the range [0, i]
        if (j < output.length) {
            output[j] = collection[i];
        }
    }
}
```

Roughly, this algorithm proceeds as follows:

1. First, populate the output array of size \( k \) with the first \( k \) elements.
2. Then, for each successive element (at index \( i \) of the collection), generate a random index in the range 0 to \( i \), call it \( j \).
3. If \( j \) is in bounds of the output array (i.e., \( 0 \leq j < k \)), then replace the \( j \)th element of the output array with the \( i \)th element of the collection.

Prove that the probability of any element considered by the algorithm to be in the sample is \( \frac{k}{n} \) where \( n \) is the number of elements in the collection. Do this by proving the following stronger claim by induction on the number of iterations of the algorithm (i.e., the second for-loop above):

Claim 1. After the \( i \)th iteration of the algorithm, the first \( i \) elements of the collection have probability \( \frac{k}{i} \) of being in the sample array.

(Hint: in your induction, focus on whether a particular element (a) is in the array and (b) it is replaced by the incoming element.)