Socks

Denali Relles

May 2, 2021

I've been doing a lot of laundry and I've had some thoughts regarding probabilistic algorithms. I think the way I want to do this is write about things in the order they occurred to me, not the way that seems more intuitive to me now.

1 The first try

My first thought was about a particular way to match up all your socks. It goes something like this:

- Start out with all the socks in a big pile.
- Pick up two socks at random. If they match, put them aside. Like, in a special "done" pile. Otherwise, put them in a new pile.
- Repeat that until the original pile is empty. There will hopefully be some pairs in the "done" pile and some in the mismatch pile you made.
- Go back to step 2, treating your "mismatch" pile as the original paile.
- Loop this until all the socks end up in the "done" pile.

I was thinking about how to figure out how good this algorithm is. I did a few calculations: for example, if you have a pile of socks, and you pick up two at random, what's the probability that you get a pair? I'm going to assume (for now) that you don't have, like, two copies of any pair. That is, every sock has exactly one match.

Actually that's how we'll calculate it first. If you think about picking up two socks as picking up one sock and then picking up another, I can figure out the probability. See, no matter which sock you pick up, it has exactly one match in the pile. So, the probability you get that match is exactly

$$\frac{1}{2n-1}$$

where n is the number of pairs of socks you start with (so, you started with 2n individual socks).

But there's another way of thinking about it. You could imagine reaching in and grabbing two socks simultaneously. Then the above analysis doesn't make as much sense. Instead,

you could think about the fact that you're taking one **pair** out of all possible pairs. The total number of pairs you would be happy to get is n, right, but the total number of pairs you could get (matched or non-matched) can be written as $\binom{2n}{2}$. That number is equal to $2n(2n-1)/2$, and we can see that the numbers will cancel out and we'll actually get the same probability as with the "individual socks" viewpoint. Cool.

I'm actually going to come back to this problem of individual socks vs pairs later on.

Anyway, it's hard to analyze the above algorithm, and I'll show you why. So suppose on your first try you grab a matching pair of socks. There's some chance that happens. Then what are your chances for the next one? Well, if you look carefully, you now have basically the same setup you did at the beginning but down one pair, two socks. So the odds for the next grab are $\frac{1}{2n-3}$.

But what if you got unlucky your first try? Well then, you put two socks that don't match in the mismatch pile, and in the original pile there are $n-2$ matching pairs, as well as two socks whose matches are now in the mismatch pile. Your odds are worse: only $\frac{n-2}{\binom{2n-2}{2}}$, compared to $\frac{n-1}{\binom{2n-2}{2}}$. Bummer.

Basically, what happens at every step affects what happens at every step, in a non-trivial way that's hard to deal with. So things (like the expected number of socks you successfully match each time you go through the pile) are hard to calculate exactly.

Zach was the one who had the idea to do upper bounds and lower bounds. Here's the way it works: at each step, the probability you get a match is between the best case. The best case is that you found a match at every step (very rare); the worst case is that you found no matches so far. The reason the probability is sandwiched like this is because: at the ith step, the will definitely be $2n-2i$ socks left in the original pile. If you got a best case, there will be $n-i$ matches among those socks. If you get a worst case, there may be only $n-2i$ matches to find. Note that if you're really unlucky, you may get halfway through the pile, and you've unwittingly put exactly one sock from each matching pair into the "mismatch" pile, and for the rest of your pile you definitely won't find any more matches.

Ok, so the upper bound first. I'm going to introduce some notation to simplify things. Let S be a random variable corresponding to the number of matching pairs you find going through the pile once. Let 1_i be the indicator that you find a match on the ith step.

$$\begin{aligned}
\mathbb{E}(S) &= \sum_{i=1}^{n} \mathbb{E}(1_i) \\
&\leq \sum_{i=1}^{n/2} \frac{n-i}{\binom{2(n-i)}{2}} \\
&\leq \sum_{i=1}^{n} \frac{1}{2(n-i)+1} \\
&= \sum_{i=0}^{n-1} \frac{1}{2i+1} \\
&\leq \frac{1}{2} \sum_{i=1}^{n-1} \frac{1}{i}
\end{aligned}$$

This is like the sum of the reciprocals of the first n odd numbers. This is half of the harmonic series, so the sum is like logarithmic. Ok, so our upper bound for the expected number of socks in one pass, if you have n to start with is like $\Theta(\log n)$.

Now let's do a lower bound.

$$\begin{aligned}
\mathbb{E}(S) &= \sum_{i=1}^{n} \mathbb{E}(1_i) \\
&\geq \sum_{i=1}^{n/2} \frac{n-2i}{\binom{2(n-i)}{2}} \\
&\geq \sum_{i=1}^{n/2} \frac{n-2i}{(n-i)(2n-2i)} \\
&= \sum_{i=1}^{n/2} \frac{n-i}{(n-i)(2n-2i)} - \frac{i}{(n-i)(2n-2i)} \\
&= \frac{1}{2} \sum_{i=1}^{n/2} \frac{1}{n-i} - \sum_{i=1}^{n/2} \frac{i}{(n-i)(2n-2i)} \\
&= \frac{1}{2} \sum_{i=n/2}^{n-1} \frac{1}{i} - \frac{1}{2} \sum_{i=1}^{n/2} \frac{i}{(n-i)^2}
\end{aligned}$$

Look at the first term. It's similar to the last one but it's just the tail of the harmonic sequence, so it's actually like $\log(n) - \log(n/2)$, and if we apply log rules we get a constant $\log(2)$. The second term is negative, so this lower bound can't tell us that the expectation is any better than constant. I'll come back to the negative term later, but I stopped here, because the lower bound didn't tell me that $\mathbb{E}(S) = \Theta(\log n)$, which is what I was hoping for.

But what exactly is the lower bound? Is it even constant? So the first term is like $\frac{1}{2}\log(2) = \log\sqrt{2} \approx 0.346$. If I can show that the second term is, like, definitely less than that, I'll have my constant lower bound. First, a naive approach:

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{n/2}\frac{i}{(n-i)(2n-2i)} &\leq \frac{n}{4}\left(\frac{n/2}{(n/2)^2}\right)\\ &= \frac{n}{4}\left(\frac{2}{n}\right)\\ &= \frac{1}{2}\end{aligned}$$

Now that's not good enough, because if we subtract that from $\log\sqrt{2}$ We get a negative number. We could try to do a better approimation, and prove that it's constant, but I'm not going to because I don't want to.

Instead I'm going to think about other stuff. I realized that this algorithm overlooked a much simpler algorithm.

- Start out with all the socks in a big pile.
- Pick out two at random. If they match, throw them in the "match" pile. Otherwise, just throw them back in the same pile.
- Repeat that until you're done.

This algorithm is more appealing, because it only uses one pile, and it's also easier to analyze. Here I go: the probability of picking out a match on your first try is $\frac{1}{2n-1}$ (using the individual socks viewpoint). If you fail, then it just loops back to the same probability. If you succeed, you end up in the inductively $n-1$ case. Using math, we see that the expected number of tries before your first match is $2n-1$, and once you get that the expected tries to get the next match is $2n-3$, and so on. Adding these all up, we get something $\Theta(n^2)$ total tries to match all the socks.

Comparing this to the first algorithm: suppose $\mathbb{E}(S) = c > 0$ is a constant. I'm actually a little shaky on what's going on here (because I'm switching between expected pairs in a given number of comparisons and expected comparisons before you get one pair), but it seems like that would imply that I would be able to find the first match in expected $\Theta(n)$. This would give the same expected time as the naive algorithm. Interesting.

There's another algorithm to talk about. It's the one you might use. Intuitively, it goes like this: pick up one sock, look through the pile to find it's match, set them aside, and do it again. At first glance, this algorithm seems much faster, and also it seems deterministic. But let me write it out:

- Start with all the socks in a pile.
- In your left hand, pick a sock at random.
- With your right hand, pick a sock at random. If it matches, set the pair aside. Otherwise, put the sock in your right hand into a mismatch pile and pick up a new sock.

- Repeat step 3 until you've matched the sock in your left hand (this must happen eventually). Then, put all the socks back in a single pile and go back to step 2.

Here we make a couple observations. One: this algorithm is in fact random, and we can analyze its expected runtime. In fact, you can expect to go through about half of the socks while trying to match the sock in your left hand. This means matching the first sock takes in expectation $\Theta(n)$ time, then you reset to $n-1$ pairs. Like the naive algorithm 2, the total time this takes $\Theta(n^2)$ time. Interesting: these algorithms are the same.

The thing is, as humans, we can look at 20 socks and pick out the one that matches the one in our hand in basically $\Theta(1)$ time (whatever that means). But these algorithms have a limited instruction set. They can only do the following:

- Pick up a sock,
- Put down a sock,
- Compare the two socks in your hands. If they match, put them both in the "match" pile.

And basic control flow (I guess just "goto", and you could have branching on the third command (compare the socks in your hands)).

You can edit the instructions to account for the fact that some of my algorithms involve putting socks into separate piles. But one important thing is that my imaginary sock robot has no memory. It can only make decisions about the two socks it is holding in its hands.

Seeing the two naive algorithms both have the same expected time $\Theta(n^2)$, I'd conjecture that you can't do better than that.

A couple more notes: I could rewrite the instructions to have "pick up a sock with your left hand" and "pick up a sock with your right hand" to be separate, and ditto with "put down", or I could just have "pick up two socks" and "put down two socks". It seems like giving fine grained control might allow faster mathing, but it didn't help with the naive algorithm.

Note: we could make the piles rigorous. I could say "you have k piles in addition to the 'match' pile. When you pick up or put down socks, you can choose which pile to take from or put into." It just doesn't really seem helpful to spread out your socks if you can't remember what you put where. Yeah, I think the naive algorithm(s) are the best. My favorite is "pick up a pair, if they match set them aside, otherwise throw them back" because it's so simple, but has (I think) the best runtime you could hope for anyway.