# 100 Prisoners and a Light Bulb

Gregory Schare

19 December 2021

## 1 A Riddle

100 prisoners are sentenced to life in prison in solitary confinement. Upon arrival at the prison, the warden proposes a deal to keep them entertained, certain that the prisoners are too dim-witted and impatient to accomplish it.

The warden has a large bowl containing the cell numbers of all the prisoners. Each day he randomly chooses one cell from the bowl, the corresponding prisoner is taken to the interrogation room, and the cell number is returned to the bowl.

While in the interrogation room, the prisoner will not be allowed to touch anything except the light switch, which the prisoner may choose to turn on or off.

The prisoner may make the assertion that all 100 prisoners have been in the room. If the prisoner's assertion is correct, all prisoners will be released. If the prisoner is incorrect, the game is over and their chance to be freed is gone forever.

The prisoners are given one meeting to discuss a strategy before their communication is completely severed. What strategy should they adopt in order to ensure, with 100% certainty, that one of them will guess correctly and all be freed?

The initial state of the light is OFF when the first prisoner enters the room.

## 2 Context

A friend of mine introduced me to this riddle recently. It is a simple concept, but unlike many similar riddles, it has multiple valid solutions each with their own merits.

I will present two different solutions, prove that they work, and compare them.

One of the key metrics by which we can compare the two solutions is how long we can expect them to take, i.e. the expected number of days to pass before a prisoner makes the assertion that all 100 prisoners have been in the room.

We can attempt to calculate these values using the techniques we learned during the course along with some good old simulation programming.

Since each prisoner has an equal chance every day of being picked, there is no way we can *certainly* conclude that every prisoner has been in the room simply based on how many days have passed. However, this may be a viable strategy if you are willing to accept some risk. Given the stakes, however, the two solutions I propose are not probabilistic.

Clearly, any deterministic solution will require making use of the single bit of information that the prisoners can use to communicate with each other: the light bulb. At its heart, this riddle is a problem of how to produce a Boolean value (all 100 prisoners have or have not been in the room) using another Boolean value. At first glance this is mathematically impossible. But, underneath the hood, we actually have access to more information: each prisoner can *individually* hold arbitrary data in his head, and must somehow communicate that across a single bit of information.

## 3 First Strategy

The prisoners get together in their strategy meeting, and everyone is stumped. Then, one prisoner gets up and presents a possible strategy. Here is what he proposes.

### 3.1 Algorithm

The first prisoner who is called to the room will be the one who makes the assertion and secures the prisoners' freedom. This prisoner, call him Prisoner 1, has two responsibilities. First, he must leave the light ON every time he leaves the room. Second, he must remember how many times, when he enters the room, the light has been OFF.

Every other prisoner has just one responsibility. The first time he enters the room and sees the light ON, he must turn OFF the light. Every subsequent time he enters the room, he should do nothing.

Once Prisoner 1 has seen the light left OFF 100 times, he can confidently say every prisoner has been in the room.

### 3.2 Proof

The other prisoners are impressed with the simplicity of the strategy, but they want to know for sure that it works. After all, the stakes of getting it wrong are dire. The proponent of this strategy patiently explains why we can trust it to give the right answer.

Since only Prisoner 1 is responsible for getting it right, we simply need to show that he will get it right; we do not need to worry about anyone else getting it wrong.

It is assumed that every prisoner will correctly fulfill his duties according to this strategy, i.e., no one will accidentally leave the light on when it should have been off or visa versa.

With each prisoner, we are using the single bit of information (the light bulb) to transmit the message "someone has been in here who we have not counted already." ON means this has not happened, OFF means this has happened.

If that message is always successfully transmitted without error, then we can certainly conclude that after Prisoner 1 encounters the light OFF 100 times (including his first entry into the room), that there have been 100 unique visitors to the room, and so Prisoner 1 can make the assertion and everyone can go free.

It remains to show that, with this strategy, we can never have data corruption. That is to say, Prisoner 1 will never find the light ON when there *has* been an uncounted visitor (false negative), and he will never find the light OFF when there *has not* (false positive).

Let us consider these two possible errors and see if they can occur. Remember that we assume Prisoner 1 dutifully leaves the light ON every time he leaves the room.

### 3.2.1 Discussion of Errors

Note that any data corruption will, in fact, ruin the strategy. A single false positive will likely result in Prisoner 1 announcing the assertion too early, and a single false negative will (usually) result in him never reaching 100. This is because the prisoners have no way of communicating that an error was made or how to correct it.

It is possible that a false positive will not ruin the strategy. This happens when a prisoner creates a false positive, but the prisoner who was overshadowed had already visited the room, just not on a day where the light was left on so he could document his visit. This highlights the way in which the strategy does not necessarily finish the moment each prisoner has entered the room once. It is entirely possible that between Prisoner 1's first and second visit, every single other prisoner comes into the room. Unfortunately, only the first of them will be counted.

A good question to ask is: is there any strategy which finishes as soon as every prisoner has been in the room? Is this equivalent to asking if the presence of a single false positive always results in wrong output?

I mentioned that false negatives "usually" ruin the strategy. In fact, it is possible to recover from this if the prisoner who leaves the light ON when it should be OFF remembers his mistake and leaves it OFF at the next chance to document his visit, rather than proceeding to leave it ON on future visits. We can see thus that the strategy is resilient to false negatives if the originator of the error is aware of the mistake.

### 3.2.2 False Negative: Light ON when it should be OFF

If Prisoner 1 finds the light ON, but there has been a new visitor between this visit and his last visit, that means that some prisoner did not turn off the light even though it was his first time visiting the room with the light ON, *or*, someone else before the new visitor created a false positive.

Every prisoner is assumed to correctly fulfill his duty and we are about to see that a false positive cannot occur, so a false negative cannot occur.

#### 3.2.3 False Positive: Light OFF when it should be ON

If Prisoner 1 finds the light OFF, but there has been no new visitors, that means that some prisoner between Prisoner 1's last visit to the room and this one turned off the light even though he had been there before. But we assume that every prisoner correctly fulfills his duty and remembers whether or not he has been in the room before, so this cannot occur.

### 3.3 Conclusion

As we can see, the proof of correctness is quite trivializing since it is really just an algorithm for counting the number of unique visitors to the room. On the assumption that every prisoner fulfills his duty according to the strategy, it is impossible to have an error. The heart of the correctness of the algorithm lies in the fact that we have chosen a good meaning to assign to the Boolean value of the light bulb being on or off. In the next strategy, we will see an alternate meaning and its implications.

## 4 Second Strategy

One of the other prisoners stands up and shouts in response that the proposed strategy places too much trust in a single individual: the first prisoner to enter the room. He begins to outline a strategy which more evenly distributes the burden.

### 4.1 Algorithm

Assign each prisoner a unique number 1-100.

Each prisoner gets his hands on a ledger and a pencil. On this ledger, we write down 100 entries, initially all blank, and we keep track of what day it is, in cycles of 100 days (so Day 101 corresponds to the same entry as Day 1). For simplicity, from now on when I refer to Day 1, it also refers to Day 101, Day 201, and so on.

Each prisoner, if he enters the room on the day corresponding to his assigned number, turns the light ON.

If the light was already ON when he enters the room on Day $n$, that tells him the previous prisoner has a number corresponding to Day $n-1$. Since this uniquely identifies the prisoners, the current prisoner can note in his ledger that prisoner $n-1$ has been in the room.

Now, every subsequent time the current prisoner enters the room on Day $n-1$, he will leave the light ON. This effectively passes on the message to further prisoners that prisoner $n-1$ has been in the room.

Otherwise, he leaves the light OFF.

Once any prisoner's ledger is entirely filled, he can announce that all 100 prisoners have been in the room, and they all go free.

## 4.2 Proof

Once again, the other prisoners demand proof that this strategy works.

In this strategy, the light being ON on Day $n$ corresponds to the message "Prisoner $n-1$ has been in the room." Since every prisoner notes this down in his ledger, we can be sure that this strategy works if we can show that the ledger is full only if every prisoner has been in the room. This is true given that the message is effectively passed along without corruption.

Assuming that everyone fulfills his duty correctly, it suffices to show that the message passing does not overwrite existing ledger data. But since each prisoner is assigned a unique number and the light is only left ON if the prisoner for that day has been there (up to a recursive chain of indirect witnesses of him being there through the light bulb message), we can be sure this works out.

### 4.2.1 Discussion of Errors

This strategy is resilient to false negatives in the same way as the first strategy. If a prisoner fails to leave the light on after visiting the room on his day, the next prisoner will simply not note it down in his ledger and the day remains open. If the prisoner realizes his mistake, he can fix it the next time he visits the room on his day. However, if he goes on thinking he did his job, the algorithm will never finish.

This strategy is susceptible to false positives just like the first strategy. If a prisoner leaves the light on when it is not his day, the next prisoner will erroneously note this down in his ledger and begin the chain of passing bad information to other prisoners, possibly resulting in an undercount if the prisoner does not happen to visit the room later.

There are more sources of error in this strategy. Any individual can enter data in the wrong ledger entry, leave the light ON on the wrong day, or forget to write it down. However, these can all be grouped into the same kinds of false positives and false negatives, so it is overall only riskier than the first strategy because there are more people allowed to make the final assertion, so any one of them can make a mistake on the last entry and doom the rest.

## 4.3 Conclusion

The proof of correctness is simpler for this algorithm, partly because I have glossed over the more trivial aspects since we already saw them in the proof of the first strategy, and partly because it is truly a simpler algorithm.

# 5 Comparison

Another prisoner stands up. He agrees that both strategies will work, but points out the difficulty of deciding which one to go with. The first strategy is simpler to organize and seems like it will take less time, but it places all of the trust on a single individual. The second strategy is expensive in terms of resources and time, and while it means any one person can make a dire mistake, it seems more fair to include everyone equally.

The other prisoners murmur in agreement, unsure of what to do, until another one of them stands up. He introduces himself as a programmer (in prison for consumer privacy violations) and says he can write a simulation of the strategies to figure out which one is optimal. He requests the warden for a computer on which to run the simulations. The warden begrudgingly fulfills his request, and the programmer gets to work.

## 5.1 Coding the First Strategy

It turns out that the first strategy works just as well when we elect the leader beforehand rather than just having the first person who goes to the room become the leader. Since the light starts as OFF, without modifying their strategy at all, the prisoners who go into the room before the leader gets there will not affect the strategy. Once the leader gets there, the strategy proceeds as described.

For our purposes, we will just have Prisoner 1 be the leader.

First, we need to work with random integers, so we import the `random` package.

```
import random
```

This being a simulation, it makes sense to create some simple classes defining the entities with which we are working.

A prisoner contains two pieces of data: a prisoner ID number, and whether or not he has visited the room (technically speaking this is whether he has visited the room AND left the light off).

We will also keep a variable that is unused in the simulation that keeps track of when a prisoner *really* first visits the room. This will help as a point of comparison for our strategy.

```
class Prisoner:
    def __init__(self, id):
        self.id = id
        self.visited = False
        self.actual_visited = False
```

The leader does not need to know if he has been in the room before. He just needs an ID number and a tally of how many times he has seen the light off when he enters the room.

We also need an actual visited variable for him.

```
class Leader:
    def __init__(self, id):
        self.id = id
        self.tally = 0
        self.actual_visited = False
```

The light bulb is just a wrapper for a Boolean value with some methods for turning on and off.

```
class Light:
    def __init__(self):
        self.state = False

    def close(self):
        self.state = False

    def open(self):
        self.state = True
```

Now we can implement the strategy itself. This class is parameterized by $n$, the number of prisoners in the game, but for our purposes this will always be 100.

To begin, we initialize our entities. We will store the prisoners in a dictionary keyed by prisoner ID. The day starts at 0 because we have not called any prisoners yet.

We keep a tally of how many prisoners have actually visited the room, for comparison. This will determine when we stop counting the "ideal day" they could have left had they had perfect information.

```
class Strategy:
    def __init__(self, n):
        self.n = n
        self.leader = Leader(1)
        self.prisoners = {k:Prisoner(k) for k in range(2, n+1)}
        self.light = Light()
        self.day = 0
        self.actual_num_visited = 0
        self.ideal_day = 0
```

The game is a simple loop that runs until the leader finds the tally to be 100. Then we return the number of days that elapsed in the simulation and the ideal day they could leave.

We can assert that the actual number who visited is 100, or else we got the answer wrong and our algorithm failed.

```
class Strategy:
    # ...
    def start(self):
        while self.leader.tally < self.n:
            self.next_day()
            if self.actual_num_visited < self.n:
                self.ideal_day += 1
        assert self.actual_num_visited == self.n
        assert self.ideal_day <= self.day
        return self.day, self.ideal_day
```

Each day, we pick a random prisoner ID and take that prisoner to the room. If it happens to be the leader, we add to the tally if the light was off and we make sure the light is on when we leave.

If it was an ordinary prisoner, we turn off the light and mark ourselves as visited if the light is on and we have not visited when the light is on before.

Also, we have some extra code to keep track of the actual first visits.

```python
class Strategy:
    # ...
    def next_day(self):
        id = random.randint(1,self.n)

        if id == self.leader.id:
            if not self.leader.actual_visited:
                self.leader.actual_visited = True
                self.actual_num_visited += 1
            if not self.light.state:
                self.leader.tally += 1
            self.light.open()

        else:
            if not self.prisoners[id].actual_visited:
                self.prisoners[id].actual_visited = True
                self.actual_num_visited += 1
            if self.prisoners[id].visited == False and self.light.state:
                self.light.close()
                self.prisoners[id].visited = True

        self.day += 1
```

Now it remains to run the simulation many, many times. It turns out that Python is a very slow language and my computer takes a long time when computing upwards of more than a few thousand iterations, but 10,000 should be well enough.

```python
def main():
    iterations = 10000
    total = 0
    ideal_total = 0
    for i in range(iterations):
        s = Strategy(100)
        days, ideal_days = s.start()
        total += days
        ideal_total += ideal_days
    avg_days = total / iterations
    avg_years = avg_days / 365
    ideal_avg_days = ideal_total / iterations
    ideal_avg_years = ideal_avg_days / 365
    print(f"average days: {avg_days} (ideal: {ideal_avg_days})")
    print(f"average year: {avg_years} (ideal: {ideal_avg_years})")
```

### 5.1.1 Simulation Results

Running this code gives the following output:

```
average days: 10452.08 (ideal: 509.68)
average year: 28.635835616438357 (ideal: 1.3963835616438356)
```

Over 28 years on average, when it takes under a year and a half for everyone to visit the room! That's not very good at all! Let's see if the alternate strategy works better.

## 5.2 Coding the Second Strategy

We still `import random` and the `Light` class does not change, nor does the code to actually run the simulation.

In the Prisoner class, we need a new variable, the ledger. This can be a simple list of prisoner IDs, starting with one's own.

We do not actually need the `visited` variable for the simulation; this is again just to assert that we have the correct answer.

```
class Prisoner:
    def __init__(self, id):
        self.id = id
        self.ledger = [self.id]
        self.visited = False
```

We shift the ID numbers and day back by one so that we can more easily perform modulus operations to compare prisoner IDs and days.

Unlike before where we had one person who we checked with every day to see if we were finished, now we have a variable `finished` which anyone can modify when they find that their ledger is complete.

```
class Strategy:
    def __init__(self, n):
        self.n = n
        self.prisoners = {k:Prisoner(k) for k in range(0, n)}
        self.light = Light()
        self.day = -1
        self.actual_num_visited = 0
        self.ideal_day = 0
        self.finished = False
```

Hence, the looping code is very simple:

```
class Strategy:
    # ...
    def start(self):
        while not self.finished:
            self.next_day()
            if self.actual_num_visited < self.n:
                self.ideal_day += 1
        assert self.actual_num_visited == self.n
        assert self.ideal_day != None
        return self.day+1, self.ideal_day
```

The code for running the simulation day by day is also simpler because we do not have to switch based on whether we picked the leader or an ordinary prisoner. Now, everyone performs the same actions.

If the light is on, note the previous day in your ledger if you have not already. If the current day is in your ledger, turn the light on. Otherwise, make sure it is off.

```
    def next_day(self):
        id = random.randint(0,self.n-1)

        if not self.prisoners[id].visited:
            self.prisoners[id].visited += 1
            self.actual_num_visited += 1

        if self.light:
            if (self.day-1) % self.n not in self.prisoners[id].ledger:
                self.prisoners[id].ledger.append((self.day-1) % self.n)
                if len(self.prisoners[id].ledger) == self.n:
                    self.finished = True

        if self.day % self.n in self.prisoners[id].ledger:
            self.light.open()
        else:
            self.light.close()

        self.day += 1
```

And that’s it! We can run the simulation now.

### 5.2.1 Simulation Results

The results for the second strategy are not looking good. They take so much longer that I had to bump the number of iterations down by an order of magnitude to get an answer in a reasonable amount of time.

```
average days: 30070.095 (ideal: 517.626)
average year: 82.38382191780822 (ideal: 1.4181534246575342)
```

This is even worse than before: 82 years compared with 28 years before. While the first strategy is hardly desirable, at least there's a chance everyone lives through it! 82 years to go free is not much different than life in prison, let alone the vagueness of the warden's rules when it comes to a prisoner dying of old age during the game.

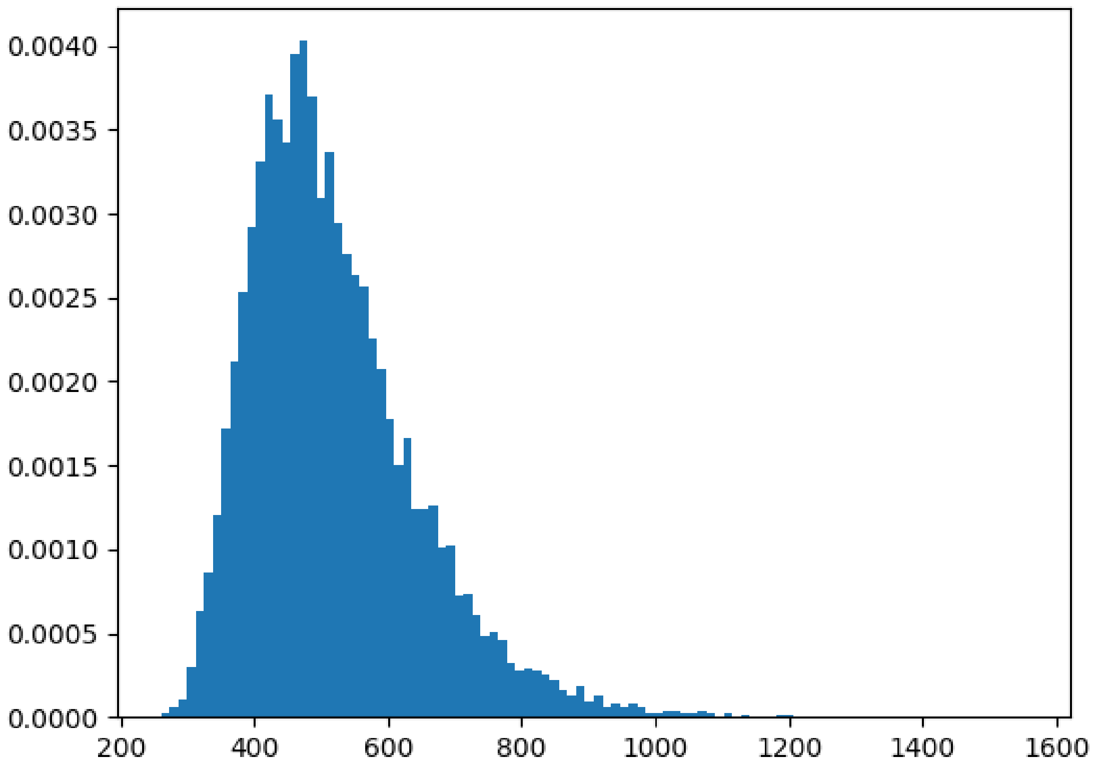
## 6 Conclusion

Having presented his results, the programmer gives everyone the news they did not want to hear: if they have any hope of getting out of here in a reasonable amount of time, they have to take a risk. He does a few more quick calculations:

```
import random
import numpy as np
import matplotlib.pyplot as plt

def game(n):
    prisoners = {k:False for k in range(1,n+1)}
    n_visited = 0
    days = 0
    while n_visited != n:
        id = random.randint(1,n)
        if not prisoners[id]:
            prisoners[id] = True
            n_visited += 1
        days += 1
    return days

def main():
    iterations = 10000
    total = 0
    days = []
    for i in range(iterations):
        days.append(game(100))

    print("average:", np.average(days))
    print("standard deviation:", np.std(days))

    plt.hist(days, density=True, bins=100)
    plt.show()

main()
```

This produces the following output:

```
average: 518.877
standard deviation: 127.40266508593923
```



He explains that the graph shows how many days it takes across a number of simulations for all the prisoners to be called to the room. On average this is less than a year and a half, but that still leaves a 50% probability of being wrong and being stuck in prison for life. He argues it is far more prudent to wait a few standard deviations above the mean.

If the data follows a relatively normal distribution, about 99.7% of all the simulations should lie within 3 standard deviations of the mean. This means that waiting `518 + 127*3 = 899` days, or just under 2.5 years, is very likely (almost 100% likely) to ensure all 100 prisoners have been in the room.

This is a far better deal than 28 years, let alone 82 years. While there is still a small risk that some prisoners have not yet been in the room after 2.5 years, the prisoners agree it is worth it rather than spending most of their lives in prison anyway just for the guarantee of leaving.

The strategy is finalized. The prisoners will wait.

## 7 References

1. Rana Urek, CC'24, who introduced me to this puzzle and worked out solutions with me.
2. 100 Prisoners and A light Bulb by Yisong Song
3. 100 Prisoners and a Light Bulb
4. 100 Prisoners and a Light Bulb Riddle & Solution by Brett Berry

And finally thank you to Yining Liu for talking about the riddle with me!