Problem 1:
The first assumption is that if the attacker do anything other than playing by the rules, people will leave the Bitcoin community and value of Bitcoion will collapse, which means the Bitcion stolen by the attacker is worthless. This is really bad considering the attacker needs to amass a lot of expensive hardware to have the ability to attack the network. It is not in his or her interest to allow Bitcoin to lose value.

Problem 2:
From 340 blocks behind, probability that the attacker finds the next block is 45%, and the probability that an attacker catches up with the honest chain is less than 0.1%.

Problem 3:

q	z
0.1	3
0.15	3
0.2	5
0.25	6
0.3	10
0.35	17
0.4	36
0.45	137

```
package main

import (
  "fmt"
  "math"
  )

func main() {
  fmt.Println(AttackerSuccessProbs(0.45, 137))
}

func AttackerSuccessProbs(q float64, z float64) float64{
  var p float64
  p = 1.0 - q
  var lambda float64
  lambda = z * (q / p)
  var sum float64
  sum = 1.0
  var i float64
  var k float64
  for k = 0; k <= z; k ++ {
    var poisson float64
    poisson = math.Exp(-lambda)
```

```
   for i = 1; i <= k; i ++ {
     poisson = poisson * lambda / i
   }
   sum = sum - poisson * (1 - math.Pow(q/p, z - k))
  }
  return sum
}
```

Problem 4:
If we only sign all data and check if the data is signed by me when retrieving it, then the server could just always serve me one data entry, which is signed by me, but only saved one data entry and not others, so I actually can't verify if they saved the specific data that I sent, only that the data was sent by me.

Problem 5:

a) Write: first retrieve all existing element, concatenate them, hash the concatenated version, save the hash, write to the database
   Read: retrieve all data from the database
   Verify: concatenate all data retrieved, hash them, compare the hash with the local hash
b) Writing to the database and reading from the database will scale badly since you have to read every data point in the database first, get the hash with the new data, and then write the new data to the database or read it from database. It will always be O(n).

Problem 6:

a) When writing, need to get the sibling in the tree for a block, hash both to get the parent, then get the sibling of the parent, then hash them to get another parent, repeat this process until you update the whole path from the updated node to the Merkel Tree root node. When reading and verifying, you need to retrieve the block, get the sbiling block, hash them, and then get the sibling of the hashed string, then get the sibling of this hashed string, then hash them together, repeat this until you get to the root of the Merkel Tree, if the root is the same as the one you stored, then the block you want to read and verify is correct.
b) Reading and writing will get to a worst case of O(lgn) since we are in a tree structure.

Problem 7:

a) 21 blocks
b) answer = α^* (t/10)

Problem 8:
Every day: 144 blocks
Answer = α^* ((l/60)/10) * (60*60*24/l)

Problem 9:
Answer = α* ((l/60)/10) * (60*60*24/l) + (1440 / (α^2 * (2 + (α) / (1 - 2α)) * 10 * 1 / α)) * (α^2 * (2 + (α) / (1 - 2α))

Problem 10b:
Question for the law professor:
Today we have many big companies that have control over what is said on the Internet. For example, Facebook could decide to ban any discussion on a certain topic and choose not to show users these stories. Is it illegal for Facebook to do this and if it is, how wuld government deal with this type of behavior?
For the FBI agent:
What are some command misconceptions that criminals have regarding Bitcoin?