Problem 1:

```python
def h1(x):
    return (x+1) % 32

def h2(x):
    return (2*x+x) % 32

def h3(x):
    return (3*x+x**2) % 32

def h4(x):
    return (4*x+x**3) % 32

l = [0]*32

c = 2017
h = 4
noFP = True

while(noFP):
    c += 1
    h = 4
    if (l[h1(c)] == 0):
        l[h1(c)] = 1
        h -= 1
    if (l[h2(c)] == 0):
        l[h2(c)] = 1
        h -= 1
    if (l[h3(c)] == 0):
        l[h3(c)] = 1
        h -= 1
    if (l[h4(c)] == 0):
        l[h4(c)] = 1
        h -= 1
    print(c)
    print(l)
    if (h == 4):
        noFP = False
        print(c)
```

Python script to the left outputs the stuff below. It appears that **2030** is the first false positive.

```
nathan@coeusx2:~/Dropbox/School/UIUC/CS425/Week5$ python hw2p1.py
2018
[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
2019
[0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
2020
[0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
2021
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
2022
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]
2023
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0]
2024
[1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0]
2025
[1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1]
2026
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1]
2027
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1]
2028
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]
2029
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]
2030
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]
```

Problem 2:

For a given Bloom filter with k = 1, the false positive rate would be: $\left(1 - e^{-n/m}\right)$
Using rules of probability, this would mean that for a series of L Bloom filters, the odds of them all returning a false positive would be: $\left(1 - e^{-n/m}\right)^L$
This is the rate of false positives for the proposed protocol. The proposed protocol will have a better false positive rate when:

$$\left(1 - e^{-n/m}\right)^L < \left(1 - e^{-kn/m}\right)^k$$

$$L \cdot log(1 - e^{-n/m}) < k \cdot log(1 - e^{-kn/m})$$

$$L > k \cdot \frac{log(1 - e^{-kn/m})}{log(1 - e^{-n/m})}$$

Note: inequality sign flips because both logs ***must*** be negative.

(1) when n = 10, m = 1024, k = 4: $L > 4 \cdot \frac{log(1-e^{-40/1024})}{log(1-e^{-10/1024})} = 2.82$
meaning that the proposed protocol would need 3 or more Bloom filters to have a better rate of false positives. However, in $L = 2^r$, if r is an integer then L will have to be 4 or bigger, where r = 2.

(2) when n = 80, m = 1024, k = 4: $L > 4 \cdot \frac{log(1-e^{-320/1024})}{log(1-e^{-80/1024})} = 2.03$
meaning that the proposed protocol once again needs 3 or more Bloom filters to have a better rate of false positives. However, in $L = 2^r$, if r is an integer then L will have to be 4 or bigger, where r = 2. (although L = 2, r = 1 is really close to satisfying it).

Interestingly, the requirement (exact value) has decreased as more elements are inserted…

Problem 4:
Hermione's approach suffers from local clock drift. While the clocks will be synchronized with each other within a margin of error, Cristian's algorithm does not guarantee that the local system will be in sync with all other external systems. The cluster as a whole will eventually fall out of sync with the rest of the world's systems. In order to maintain synchrony with all external systems, Hermione will need to connect to her cluster a time measuring device that is as accurate as the atomic clocks that the internet uses. In other words, Hermione, if she wants to prevent the local system clock drift, will need to construct/acquire and hook up her cluster to a local atomic clock (~$1500 cheapest).

Problem 5:
Error is at most:

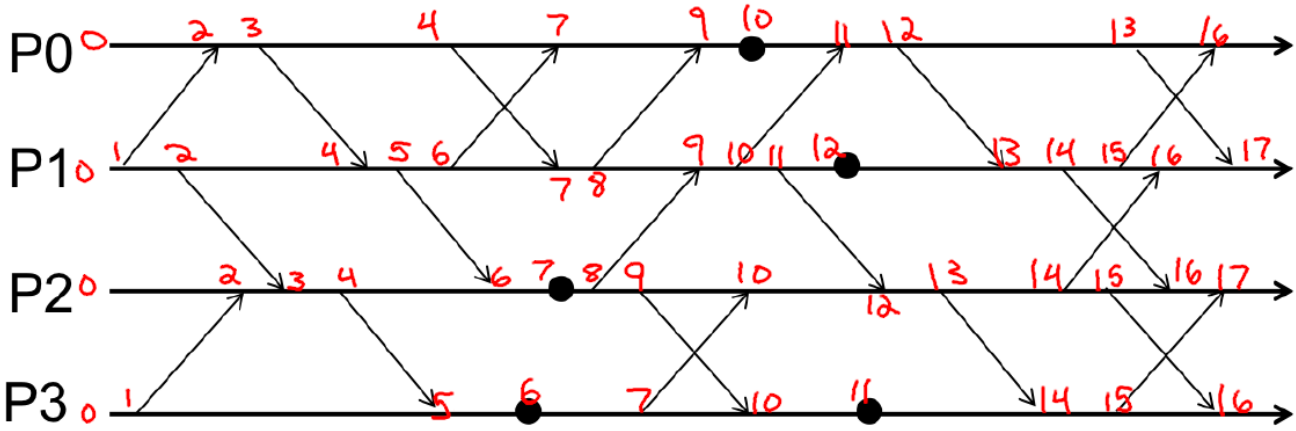
$$\frac{3990-(333.3+x)-(330+10)}{2} = \frac{3316.7-x}{2}$$

Since the NIC to Application buffer delay from the client side is unknown (x), the error could be a range of values. Since $x \geq 0$ we can at least get a upper bound of the error:

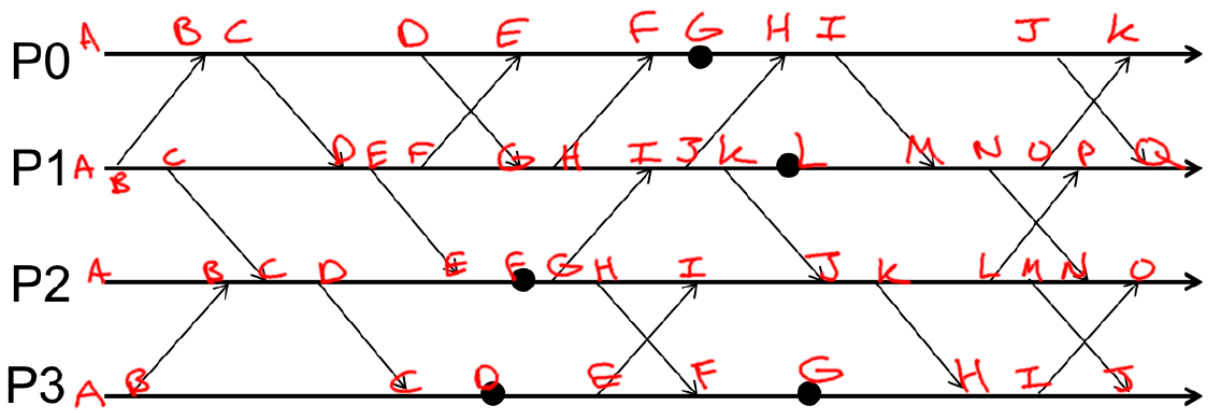
$$\frac{3316.7-0}{2} = \frac{3316.7}{2} = 1658.35 = 1.658ms$$

Thus, with the given data, the error of the system is at most **$0 \leq E \leq 1.658$ ms**.

Problem 6:



Problem 7:



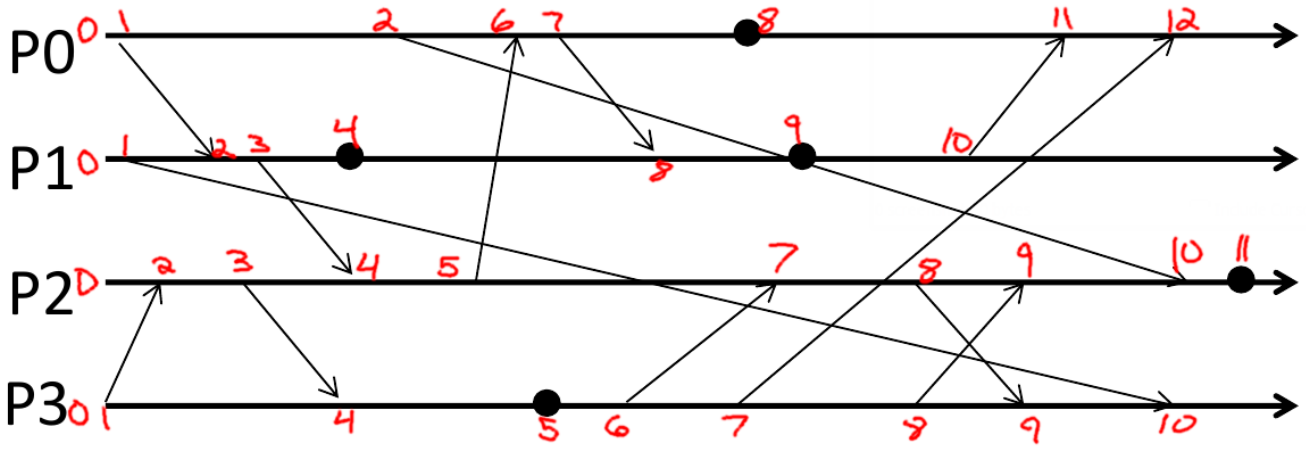
	P0	P1	P2	P3
A	(0,0,0,0)	(0,0,0,0)	(0,0,0,0)	(0,0,0,0)
B	(1,1,0,0)	(0,1,0,0)	(0,0,1,1)	(0,0,0,1)
C	(2,1,0,0)	(0,2,0,0)	(0,2,2,1)	(0,2,3,2)
D	(3,1,0,0)	(2,3,0,0)	(0,2,3,1)	(0,2,3,3)
E	(4,5,0,0)	(2,4,0,0)	(2,4,4,1)	(0,2,3,4)
F	(5,7,0,0)	(2,5,0,0)	(2,4,5,1)	(2,4,7,5)
G	(6,7,0,0)	(3,6,0,0)	(2,4,6,1)	(2,4,7,6)
H	(7,9,6,1)	(3,7,0,0)	(2,4,7,1)	(3,10,10,7)
I	(8,9,6,1)	(3,8,6,1)	(2,4,8,4)	(3,10,10,8)
J	(9,9,6,1)	(3,9,6,1)	(3,10,9,4)	(3,10,12,9)
K	(10,14,6,1)	(3,10,6,1)	(3,10,10,4)	
L		(3,11,6,1)	(3,10,11,4)	
M		(8,12,6,1)	(3,10,12,4)	
N		(8,13,6,1)	(8,13,13,4)	
O		(8,14,6,1)	(8,13,14,8)	
P		(8,15,11,4)		
Q		(9,16,11,4)		

Problem 9:
If one were to pick a quorum of size Q while prioritizing least common members first for three rounds, the three quorums will have a minimal intersection with cardinality = 3Q – 2N, assuming N/2 < Q < N. K is equal to this cardinality. (ie, K = 3Q – 2N, see below for more explanation)

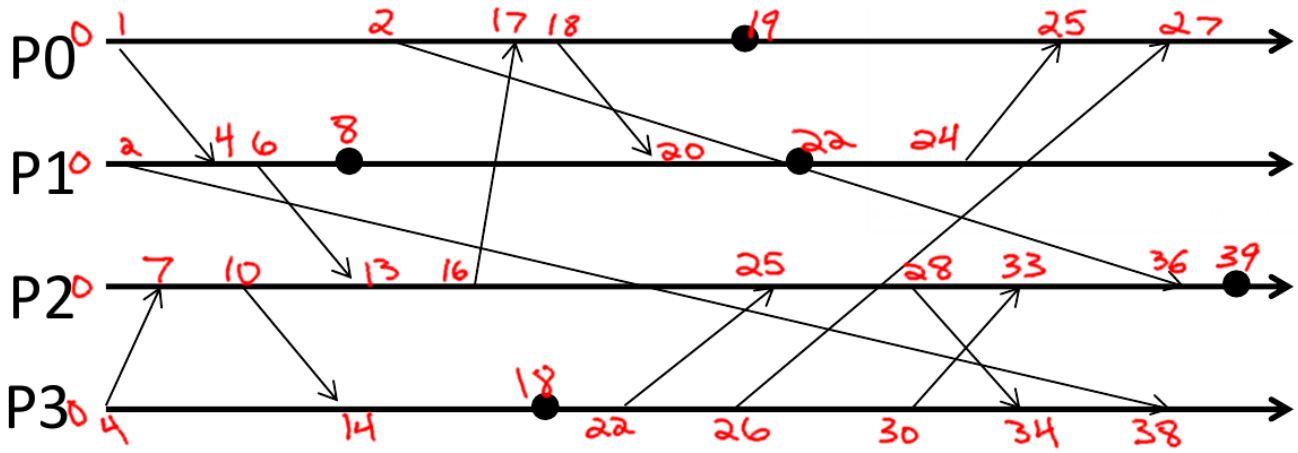
(1) for N= 50, K= 1; we have 3Q – 2(50) = 1, which implies that Q = 101/3 = 33.666 ≈ 34. (must take ceiling to satisfy conditions). Therefore, **Q = 34**.
(2) for K = 2 for all sufficiently large N, we have 3Q - 2N = 2, which implies that $Q = \lceil \frac{2N+2}{3} \rceil$ (note the ceiling again here and in the next two).
(3) for K = N/2 for all sufficiently large N, we have 3Q – 2N = N/2, which implies that $Q = \lceil \frac{5N}{6} \rceil$
(4) for K = N/4 for all sufficiently large N, we have 3Q – 2N = N/4, which implies that $Q = \lceil \frac{3N}{4} \rceil$

Reasoning for K = 3Q – 2N:
Starting with N members, pick Q members. Those left over will be N-Q. For the second quorum, first pick the N-Q that were not selected in the first. Then fill out the rest of the quorum by picking Q-(N-Q) = 2Q – N from the first quorum. This is the intersection of these two quorums so far. Note that there are two subsets in which members belong strictly to the first or second quorum, totaling 2 * (N-Q) exclusive members. For the third quorum, first pick those that exclusively belong to either the first or second quorum. Then pick the remaining members from the intersection of the first two quorums to fill out the third quorum of Q members. The amount to choose from the previous two's intersection will be Q – 2*(N-Q) = 3Q – 2N . This is the minimal intersection of all three quorums,
thus K = 3Q – 2N. □

Problem 10:
Here is what it would be with regular Lamport timestamps:



Here is what it would be with Madonna timestamps:



If you pick two arbitrary events, there's no guarantee that the event's ordering will be the same between the two methods. EG, 10 from P1 and 7 from P3 in the Lamport method (10 > 7) will switch to the 24 from P1 and 26 from P3 in the Madonna method (24 < 26), thus the ordering of these two events would be different. However, these are just concurrent events by the rules of causality, as they do not share a causal path. What's important here is whether causality is maintained between the two methods. It appears that causality is in fact maintained in the Madonna method. If you trace any causal path in the example above, the timestamps' ordering along that causal path will be the same in both methods. This works because in both methods the event timestamp is strictly increasing at each event along a causal path. While the amount it increases for each process is different, since the max is taken upon receipt of a message it will ensure that each event timestamp is strictly increasing. If the max were not taken at each message receipt, then the method would no longer be correct. But, as the method is now, it will be 100% correct.