Exercise 2

Exercise 2.3

2.3 (a)

Yes, it implements Uniform Causal Order because it says these lines:

[Partial algorithm from slides]

```
p has prev. executed deliverCA(m’) ∀ m’ ∈ D
then
deliverCA(m)
prevDlvrs := prevDlvrs ∪ {m}
else
discard m
```

That tells if m' is delivered before m then all other process has to deliver m' before m and if it does not happen it discards m. We can say m' has caused m in the above FIFO atomic broadcast.

(b)

No, because Uniform agreement(V) says if a CORRECT or NOT process delivers a message m, every correct process eventually delivers m. So for the wrong process it does not accept total order(XIII) i.e a wrong process can deliver m2 before m1 but in actual m1 should be delivered before m2.

We can simplify it if we consider only causal ordering where m1 causes m2 or use consensus.

Exercise 2.4

The **Comparison** of different channels are as follow.

	Reliable Channel	Semi-Reliable Channel : **Deterministic**	Semi-Reliable Channel : **Random**
No Duplication	No message **delivered** more than once	A message is **received** at most once	A message is received at most once
No Creation	No message is **delivered** unless it was **broadcast**	No message is **received** unless some process did **send** it	No message is received unless some process did send it
Validity	For Correct P_i & P_j then for every message broadcast by P_i is **eventually delivered** by P_j	For Correct P_i & P_j , It provides **50% assurance** for every message to be received by P_j. Anyhow One of two m would be received.	For Correct P_i & P_j , It provides **25% assurance** for every message to be received by P_j. Anyhow One of two m would be received.
Agreement	If correct P_i delivers m then P_j eventually **delivers**	It cann't provide any delivery assurance as there is no agreement rule defined.	**No delivery assurance**

Both of them don't fulfill the criteria for a Reliable channel.

Statement
Reliable channel is stronger and stricter than Semi-Reliable Channel because

1. Semi-Reliable channel can be derived from Reliable channel.
2. Reliable channel is stronger than Semi-Reliable channel not vice-versa.

Proof

Algorithm

1. Process P sends every second message m (or drops one of the two messages) to every other process including itself.
2. Every process which receives m for the first two time (one of two) sends it to every other process (except the sender) and delivers it.

∀ message m € $\{m_1, m_2 \ldots\ldots\ldots\ldots m_n\}$

(m_1, m_2) Process P_i $(m_1 || m_2)$ -----------> P_j $(m_1 || m_2)$
P_i drops one of two message deterministically or random as given and similarly P_j

The above algorithm satisfies the assumption that **Semi-Reliable channel could be derived from** more stricter form of **Reliable channel**.
Reliable channel could not be derived from Semi-Reliable channel as nearly **half of the messages are being dropped** from being eventually received by P_j and hence preventing eventual delivery of every messages by correct processes P_i and P_j .

Exercise 2.5

(a)

```
p1: a,1; b,2; c,3; d,5; e,6; f,7;

p2: g,1; h,2; i,3; j,6; k,7; l,8;

p3: m,1; n,3; o,4; p,5; q,6; r,10;

p3: s,4; t,5; u,5; v,7; w,8; x,9;
```

(b)

```
p1[0,0,0,0]: a,[1,0,0,0]; b,[2,1,0,0]; c,[3,1,0,0]; d,[4,1,3,0]; e,[5,1,3,0]; f,[6,1,3,0];

p2[0,0,0,0]: g,[0,1,0,0]; h,[0,2,1,0]; i,[0,3,1,0]; j,[3,4,1,2]; k,[3,5,5,2]; l,[6,6,6,2];

p3[0,0,0,0]: m,[0,0,1,0]; n,[1,0,2,0]; o,[1,0,3,0]; p,[1,0,4,0]; q,[1,0,5,0]; r,[5,3,6,6];

p3[0,0,0,0]: s,[3,0,0,1]; t,[3,0,0,2]; u,[3,3,1,3]; v,[5,3,3,4]; w,[5,3,4,5]; x,[5,3,4,6];
```

(c)
No, there is no causal because as per vector time stamp, NONE of the message time stamp is less than local time vector.

(d)
C1 is inconsistent as v is in cut but e is not where as e has caused the v, so it voilates the consistent cut definition.

(e)
Ca and Cb are consistent because the cause of the event is in the cut.[refer d.png]

(f)

The algorithm is not fault tolerant because it has no termination detection. One can not d**istinguish** between.

- message dropped from network
- server is down
- timestamp s is not yet exceeded

With server the singleton initiator process oft the Mattern algorithm is ment.

(g)

You can do three things as described in: An introduction to snapshot algorithms in distributed computing (1995):

1. Let the initiator (server) be a virtual process instead of a physical one

2. Remove the unnecessary ACKs with incrementing s so that no „dummy-message“ is needed
3. Replace the time with a binary value. Because only a binary value is needed to determine wheter a report is needed.

All in all you also need a termination protection and a recovery routine to handle the case that the server is terminated.