**Question 1**

What does Isis virtual synchrony depend on a failure detection service within the Group Management Service instead of just having individual processes detect failures via timeouts?

The problem with having each process manage its own concept of failure is that different processes may end up with different ideas of group membership:
- If process $A$ times out when trying to contact process $X$, it will remove process $X$ from its group.
- If process $B$ manages to reach process $X$, it will keep it in its group.

With a centralized group membership service, even inaccurate reports (e.g., process $X$ is dead) will be treated as global truth.

**Question 2**

What causes a view change in virtual synchrony?

When the membership of a group changes:
1. A new member joins the group.
2. A member leaves the group.
3. The group membership service decides that a member is dead and is taken out of the group.

**Question 3**

Assume that the coordinator in a two-phase commit protocol dies during the protocol and another node is elected to step in as a new coordinator. Transaction state is maintained in a persistent log.

Under what circumstance can the protocol not complete?

- If a participant crashes as well.
- The 2-phase commit protocol can be enhanced to support replacement coordinators. However, the new coordinator needs to know what the unanimous vote is among all participants.

**Question 4**

What problem does the three-phase commit protocol try to solve?

- If a participant crashes along with a coordinator.
- With a 2-phase commit protocol:
  - The system had no way of figuring out what the result of a transaction was – it needs to know how everyone voted.
- With a 3-phase commit protocol:
  - Vote results are propagated to every participant before any participant commits (this is phase 2).
  - If a recovery coordinator knows that any participant received the vote results, then it knows that every participant is ready to commit.
  - If any participant does not have vote results, the coordinator knows that no participant has committed yet.
Question 5
How does Paxos's acceptance of a proposal avoid the blocking behavior of a two-phase commit?

- In Paxos, only a majority of acceptors must agree to a proposal.
  - The protocol will block (or fail) only if there is no majority
  - A majority ensures that, if there are two concurrent proposals, at least one acceptor
    gets both of them and forces a consensus among them.
- In 2PC, all ~100% – participants (acceptors) must agree.

Question 6
When does a proposer have to change the value that it is proposing during the Paxos consensus protocol?

- A proposer sends a value to an acceptor (= prepare message)
  - Multiple proposers may do this concurrently
- Acceptors respond to a prepare request from a proposer with the highest numbered proposal that they accepted.
  - Usually, this will be the same value as in the prepare message.
  - However, if multiple requests came in concurrently, an acceptor may have seen a
    higher number. It responds to each proposer with that higher number.
- A proposer must ask for that value to be accepted even if it initially
  proposed a different value.
- This does not violate the requirement of consensus since the
  algorithm selects one of the proposed values.

The End