You are to work on the following questions *alone*. Typeset your answers and submit as a `PDF` on Canvas. **Try to keep your answers short.**

1. In this set of questions, we will examine how to provide strong ordering guarantees on message delivery. Let → represent the happens-before relationship, and let *send(m)* and *recv(m)* represent the events of sending and receiving the message *m*. The ordering property that we would like to provide is the following causal delivery property: Let messages m_1 and m_2 be two messages sent to the same node. If $send(m_1) \rightarrow send(m_2)$, then we would like to ensure that the receiver receives m_1 before m_2. (Note that m_1 and m_2 might be sent by different nodes in the system. In the case of them being sent by the same node, then FIFO delivery using some sort of sequence number would suffice.)

   - Consider designing a scheme that has the following properties. Every node in the system maintains Lamport clocks, and each node tags messages with the Lamport clock value of the node sending the message. Further, assume that receivers don't immediately "deliver" or process a message but rather wait for other events to happen before processing messages. (In particular, a receiver could buffer a set of messages before deciding which of them to deliver or process next.) Design a scheme that uses no additional messages or metadata information to provide the desired ordering property.
   - What is the worst-case delay in processing a message with the scheme outlined above? Discuss the implications of introducing periodic pings, i.e., each node in the system periodically sends ping messages to other nodes.
   - Now consider the special case of broadcast messages, which are messages sent from a node to every other node in the system. Assume that you have a distributed system that uses both unicast and broadcast messages. Suppose you want to provide a strong ordering guarantee on message delivery only for broadcast messages. (In other words, if m_1 and m_2 are unicast messages, then there is no need to order their delivery at a receiver. However, if they are broadcast messages and if there is a causal order between their send events, then the receive events would be consistent with that order.) How would you use a scheme that uses vector clocks to provide this strong ordering guarantee? What property does this scheme have over the scheme that you developed for the first part of this question?

2. We now consider how we can generalize the distributed snapshot algorithm discussed in class.

   - Assume that any node in the distributed system can trigger the distributed snapshot algorithm by sending a "take a snapshot" message to its neighbors. Consider a setting that a node might make an independent and autonomous decision in initiating the snapshot. (For example, a node could initiate a snapshot when it suspects that some liveness or safety condition is getting violated based on local knowledge.) Assume, however, that a node would initiate a snapshot at most once. What changes would you make to the distributed snapshot algorithm (Section 13 of the chapter on Consistent Global States of Distributed Systems) to accommodate this new requirement while keeping the number of protocol messages to a minimum?

   - Suppose that you want to perform consistent snapshots repeatedly. Again assume that any node in the distributed system can initiate a snapshot, and it can do so multiple times during the course of executing a distributed program. What changes would you make to the distributed snapshot algorithm to support the repeated snapshotting of a distributed system?
   - The distributed snapshot algorithm assumes that all nodes are operating. Discuss how you would perform distributed snapshots if some of the nodes are allowed to fail. You can assume that the nodes are fail-stop, i.e., a node doesn't recover after it fails. State any assumptions that you would need in order to develop your algorithm.

3. An acceptor in Paxos maintains the highest accepted proposal (i.e., the proposal number and its associated value) or $\perp$, which indicates that the acceptor has not accepted any proposal. We use the notation A_i: $(20,x)$ to indicate that the highest proposal A_i has accepted is proposal numbered 20 and value x. Which of the following states are valid states that you might encounter in a single-instance Paxos with five acceptors? Justify your answer. (Note that each of the following parts relates to an independent execution sequence.)
   - A_1: $(8,x)$, A_2: $(9,y)$, A_3: $\perp$, A_4: $\perp$, A_5: $\perp$
   - A_1: $(8,x)$, A_2: $(9,y)$, A_3: $(10,z)$, A_4: $\perp$, A_5: $\perp$
   - A_1: $(8,x)$, A_2: $(9,y)$, A_3: $(10,z)$, A_4: $(11,w)$, A_5: $\perp$
   - A_1: $(8,x)$, A_2: $(11,y)$, A_3: $(14,z)$, A_4: $(13,y)$, A_5: $\perp$
   - A_1: $(8,x)$, A_2: $(11,y)$, A_3: $(14,z)$, A_4: $(13,y)$, A_5: $(15,x)$

4. Assume that you want to support the fail-stop-restart model of fault-tolerance wherein a node can fail and subsequently reboot to join the distributed system using data stored in its stable storage (i.e., disks, SSDs, etc.). We want the recovering node to be considered as one of the non-failed nodes in the system when it is back online. The use of stable storage would allow, for instance, the entire Paxos group to fail simultaneously (say during a power failure) and then recover to operate as a replicated state machine without losing state on what has been agreed upon in the past. What state should a Multi-Paxos acceptor (i.e., a multiple instance Paxos node) maintain in stable storage to enable this form of fault tolerance?

5. Suppose you want to perform reconfiguration of a Paxos group, i.e., change the set of nodes that are part of the consensus group. For example, you might want to change the membership of a Paxos group from $\{A,B,C\}$ to $\{A,B,D\}$. In particular, you could use reconfiguration to swap out failed nodes with new nodes to achieve a higher degree of fault tolerance. Describe how reconfiguration can be performed in the context of Multi-Paxos under the constraint that your protocol should involve just the old or the new members of a Paxos group, i.e., you cannot use an additional separate service for performing the reconfiguration.

6. Consider the Chain Replication protocol. Discuss how you would extend the scheme to handle the simultaneous failure of two adjacent nodes in the chain. Describe what state is being maintained by the nodes and how the master could coordinate a transition to a new chain while preserving the correctness guarantees.