Question 1:

a) Show an example in which running the causal ordering broadcast protocol with vector timestamps does not work as is for enforcing causal ordering with point-to-point messages. Explain your example.

**Answer:** Assume a system with 3 nodes S0, S1, S2 – all start with a local vector timestamp [0,0,0]. S0 sends a message m₀¹ only to S1 with a vector [1,0,0] (this message can be accepted by S2 according to the protocol’s code). Then S1 sends a second message m₀² only to S2. This message will be sent with a vector [2,0,0]. When S2 receives m₀² then S2 cannot accept m₀² since its local vector is still [0,0,0] and so V2[0]≠Vm₀²[0] + 1.

b) Consider the following two implementations of atomic broadcast: (a) repeatedly invoke an instance of Paxos as we saw in class for deciding on each message, i.e., instance 1 decides on the first message, instance 2 on the second, etc. (b) the same as (a) except that the leader of the first round of each Paxos instance is a proposer whose id is <instance_number> modulo <number_of_proposers>. When would you prefer (a) over (b) and vice versa? Explain.

**Answer:** Obviously, the rotating initial proposer/leader scheme provides better load balancing between the nodes, and also an inherent rate control guaranteeing that no node falls too far behind the others. However, it is more sensitive to failures since regardless of which server fails (as long as not removed from the membership), it will repeatedly become an initial proposer/leader causing a slow consensus instance again and again. Also, if some node has more capabilities or better communication links with the others, it is better to have it always as the initial proposer/leader.

Assuming the value sent by the proposer includes the message itself (rather than propagating the message using another mechanism and only voting for its header), then if all nodes have constantly new messages to send, a rotating approach is better whereas if only one node typically has messages to send then the fixed approach is better.

Question 2:

Does flooding messages over the overlay network of Bitcoin preserves causal ordering between these messages assuming messages are always delivered as soon as they arrive (without additional control information or buffering)? Explain.

**Answer:** The bitcoin overlay can dynamically change due to churn. Suppose node S0 is initially connected to node S1 over a very slow link and starts flooding a message m₀¹. Before S1 receives m₀¹, node S2 becomes a neighbor of S0 and S1 – both are fast links. Now S0 floods a message m₀², which is received by S2 that forwards it to its neighbor S1. S1 receives m₀² before m₀¹ even though m₀² is causally after m₀¹.
Question 3:

a) Why does Spanner use locks at the leader of each replication group? Explain.
   Answer: In Spanner, the leader of each replication group is the (only) server in the replication group that actually executes the transaction (or to be precise, the part of the transaction involving the local shard) and dictates its results to the other replicas in the same group. Yet, as computers are multi-core and multi-threaded, performance calls for concurrent execution. Hence, the leader employs locks to enforce correct concurrency control (serializability) on (the part of) the transactions it executes.

b) Why does Spanner need Paxos inside each replication group? Explain.
   Answer: Paxos is used to ensure that all replicas in the same replication group are synchronized on all state changes and actions the leader takes, so if the leader fails and another replica takes over, this will be done in a consistent manner.

c) How does Spanner ensure no blocking during the execution of its 2 Phase Commit protocol (executed among leaders of different replication groups when a transaction accesses objects from different groups/shard)? Explain.
   Answer: The role of 2PC is (only) to ensure that when a transaction accesses objects from multiple shards, all shards agree on the outcome of the transaction (commit or abort). Part of the state changes that go through Paxos are the 2PC protocol steps taken by the leader. Hence, when the leader fails, the replica that takes over can continue its role in the 2PC protocol correctly, and so the 2PC protocol does not block.

Question 4:

a) Explain why waiting for a timeout even after receiving more than 2/3 pre-votes that are not all the same is required for Tendermint's liveness, whereas if all these votes are the same, there is no need to wait for the timeout.
   Answer: If more than 2/3 of the votes already agree on the same value, then this is what the node will vote regardless of any other vote. On the other hand, if there is disagreement and we would not wait for the timeout, then the following liveness attack by Byzantine nodes would be possible: It is possible that the Byzantine nodes are the fastest and always pre-vote nil. In this case, the nil votes of the Byzantine nodes will always be counted among the first 2/3+ pre-votes, and so all nodes will continuously in each round vote nil, and a block will never be decided on. When waiting the timeout and assuming a partial synchrony model, eventually all correct nodes would be able to get their votes to every other correct node before the timeout, thereby ensuring termination.

Submission instructions:
You should solve this exercise alone – submissions are individual. Solutions must be submitted through the course web site – either printed or a high-resolution scan of handwriting. Solutions must be in Hebrew unless you get an authorization from Prof. Friedman to submit in English. Try to be brief. If your answer is very lengthy, it could be a sign that it is wrong.