1. For this question, consider a distributed system where each pair of processes is connected by a first-in, first-out (FIFO) reliable channel. Assume that there are no process failures.

(a) (4 points) In this system, if the basic multicast scheme (B-multicast) is used, what ordering guarantees would be satisfied? FIFO, causal, and/or total? Justify your answer: either argue why a guarantee would be satisfied, or present a counterexample.

Assume that a process immediately delivers a message to itself before sending it out to any other processes.

**Solution:** Messages would be FIFO: if \( p_i \) multicasts \( m \) and then \( m' \), then at every \( p_j \), \( m \) will arrive on the channel from \( p_i \) to \( p_j \) before \( m' \).

Neither causal nor total ordering would not be satisfied, see counterexample below. \( A \rightarrow B \) but \( P2 \) delivers \( B \) before \( A \).

(b) (4 points) What if the reliable multicast scheme (R-multicast) were used instead? Assume again that the process immediately delivers a message to itself before sending it out.

**Solution:** Causal ordering would be satisfied. When \( p_i \) sends message \( X \) to \( p_j \) (either because \( p_i \) initiated the message or during a rebroadcast), any message already delivered by \( p_i \) before \( X \) will have already been sent from \( p_i \) to \( p_j \).

Total ordering would not be satisfied, see counterexample below. The dashed lines represent the rebroadcast of messages \( A \) and \( B \) by the receiver. (The implementation of R-multicast on the slides re-multicasts the received message to all other, including the sender, though a real implementation could optimize this out.)

(c) (2 points) Do your answers for the above two parts change in a system with only two processes?
Solution: For B-multicast, with 2 processes the ordering is causal. Consider two events A and B with A → B. Assume without loss of generality that A was sent by P1. If B was also sent by P1 then P2 must deliver A before B due to FIFO ordering. If B was sent by P2, then B must follow the delivery of A by P2. (If A is delivered after B is sent, then any messages sent by P1 after A are also delivered after B is sent, which would make A and B concurrent.) In both cases, A is delivered before B by both P1 and P2.

B-multicast is not total following a counterexample similar to the previous part.

R-multicast is causal but not total, since the arguments and counterexample in part (b) apply in a system with only 2 processes.

2. (a) (3 points) Consider a synchronous system with N = 100 processes running the synchronous consensus protocol. When sending a message, each value v_i = z is encoded using two bytes. Assuming there are no failures, how many total bytes are sent by the system over five rounds of the protocol, starting from round 1?

Solution: Each process in the 1st round multicasts only its own value, therefore each process will send 2(N−1) bytes in the first round. Assuming a synchronous system with no failure, every process will receive and communicate all the values in the next rounds which means 2N(N−1) bytes will be communicated by each process. Therefore, in 5 rounds there will be 2(N−1) + 4 × 2N(N−1) communicated bytes by each process in total. Since N = 100, total number of bytes will be N[2(N−1) + 4 × 2N(N−1)] = 7,939,800.

Assuming that we are also interested in counting the messages that each process sends to itself the result changes to N[2N + 4 × 2N2] = 8,020,000.

(b) (3 points) Consider a synchronous system where processes can only fail in pairs, so that in any round, an even number of processes fail (e.g., 0, 2, 4, etc.) How many rounds would you need to ensure consensus among N = 100 processes if there is no bound on the number of failures?

Solution: After a failure-free round, the synchronous consensus algorithm ensures that all processes have the same set of values and can therefore terminate. If we run the protocol for 50 rounds, then either there has been at least one failure-free round, in which case we can terminate the protocol, or all processes have failed, in which case the protocol has terminated anyway. So therefore 50 rounds are sufficient. (You can verify that 49 rounds are not sufficient since it is possible that two processes with remain with different values.)

3. (3 points) Consider the following sequence of events, using the model from the Fisher-Lynch-Patterson proof:

\[\begin{align*}
&c_1 \rightarrow p_1 \quad \text{receive}(m_1), \text{send}(p_2, m_2), \text{send}(p_3, m_3) \\
&c_2 \rightarrow p_2 \quad \text{receive}(m_2), \text{send}(p_1, m_4) \\
&c_3 \rightarrow p_3 \quad \text{receive}(m_3), \text{send}(p_2, m_5), \text{send}(p_1, m_6) \\
&c_4 \rightarrow p_1 \quad \text{receive}(m_4), \text{send}(p_3, m_7) \\
&c_5 \rightarrow p_1 \quad \text{receive}(m_5), \text{send}(p_2, m_8) \\
&c_6 \rightarrow p_2 \quad \text{receive}(m_8)
\end{align*}\]

Which events can commute with c_2?
4. For this question, consider the diagram below. It shows four messages, $A, B, C, D$, being multicast by processes $P1$ and $P3$.

(a) (6 points) Consider a FIFO multicast implementation using sequence numbers, as discussed in class. Write down the sequence number vector for each send and receive event at each process, and the sequence number sent with each message. Also indicate when each message will be delivered at each process. (Do not worry about processes delivering their own message, e.g., $P1$ delivering $A$.)

(b) (5 points) Repeat the exercise with a causal multicast implementation. (Instead of a sequence number for each message you will include its sequence vector).
Solution: