Question 1. (Solution and Grading by: Jayasi)

Counter example to prove that the algorithm will not work in 4 rounds:
7 processes P1, P2 .. P7

In round 1: P1 sends value \( v_1 \) to P2 and fails
**At the end of round 1:**
P1 has failed, P2 has \( v_1 \) and P3-P7 don’t have \( v_1 \)

In round 2: P2 sends value \( v_1 \) to P3 and fails
**At the end of round 2:**
P1 and P2 have failed, P3 has \( v_1 \) and P4-P7 don’t have \( v_1 \)

In round 3: P3 sends value \( v_1 \) to P4 and fails
**At the end of round 3:**
P1, P2, P3 have failed, P4 has \( v_1 \) and P5-P7 don’t have \( v_1 \)

In round 4: P4 sends value \( v_1 \) to P5 and fails
**At the end of round 4:**
P1, P2, P3, P4 have failed
P5: has \( v_1 \) but P6 and P7 don’t have \( v_1 \). Since the processes have different set of values, safety will be violated.
Question 2. (Solution and Grading by: Ashwini)
Question 3. (Solution and Grading by: Ashwini)

11/12/2017: Fixed the timestamp of Earth’s delivery of multicast M7 from NHX.
Question 4. (Solution and Grading by: Rui)

Solution: (Note: Broadly, answers that show an understanding of total ordering and causal ordering will be accepted.)

If considering the arrow as delivery, causal relation:

\[ 1 \rightarrow 2, \quad (1, 2, 5, 6) \rightarrow 7, \]
\[ (1, 2, 4) \rightarrow 6, \quad (1, 2) \rightarrow 3 \rightarrow 5 \]

All the orderings that obey above causal relations should be considered. Total permutation number: 12.

1->2->3->4->5->6->7, 1->2->3->4->6->5->7, 1->2->3->5->4->6->7
1->2->4->3->5->6->7, 1->2->4->3->6->5->7, 1->2->4->6->3->5->7
1->4->2->3->5->6->7, 1->4->2->3->6->5->7, 1->4->2->6->3->5->7
4->1->2->3->5->6->7, 4->1->2->3->6->5->7, 4->1->2->6->3->5->7

If take multicast causal buffer into consideration, causal relation:

\[ 1 \rightarrow 2 \rightarrow 7, \quad (1, 2, 4) \rightarrow 6, \quad (1, 2) \rightarrow 3 \rightarrow 5 \]

All the orderings that obey above causal relations should be considered. Total permutation number: 54.

Note: The solution assumes that the arrows are delivery points (the original intent of the question). That is, you will get full credit if you did this topological sorting.

Note 2: We will also give full credit if you argued that because of buffering, multicasts may not be causally dependent on each other, so the the sequencer may reorder messages in various orders perhaps even combinatorially many orders (you do not need to list all of those orders).
Question 5. (Solution and Grading by: Faria)

A. Yes. Eventual liveness may be reached. Remember that (eventual) liveness only needs that there be ONE state reachable from the current state in which processes set their decision/output variables. Here is one such scenario. Consider a partition with N/2 nodes each. The algorithm will run to completion in each partition (since it has > N/3+1 nodes), and decide (potentially different) values. Liveness does not require these values be the same (that’s Safety), hence it is fulfilled.

B. No. Remember - safety requires that ALL states reachable make only one decision (all 0s or all 1s). Here is a scenario that makes BOTH decisions. Two leaders get elected simultaneously by getting N/3+1 votes each. Then both these leaders can both get quorums for their proposed values and have two subsets of processes deciding different values.

C. It is faster as there are only N/3 processes that need to respond. However, it is still not correct (safe + eventually live).

The second argument that it is slower as consensus will take more rounds to achieve is also acceptable.
Question 6. (Solution and Grading by: Akshun)

Possible snapshots arise from the fact that we don't know when the marker messages from the Moon and NHX arrive on other processes. So we need to think about all possible arrivals of marker messages from the Moon and NHX keeping in mind that messaging channels are FIFO ordered.

State is captured at each process when the first marker message is received.
Marker Messages:
Moon to Earth: Any time the marker message arrives at earth, $C_{ME} = <b>$.

Moon to NHX: $C_{MN} = <>$ because marker message from moon to NHX will arrive before message e arrives (FIFO ordering)

NHX to Earth: $C_{NE} = <c>$ (FIFO ordering)

NHX to Moon: marker message arrives before f arrives, $C_{NM} = <>$

Since we know the initiator's marker message arrives before any other marker messages, the snapshot algorithm ends after all the above marker messages arrive at the processes.

$S_n = a$-receive
$S_e = a$-send
$S_m = b$-send

Earth possible snapshots

A
After c arrives
$S_e, C_{ME} = <b>$
$C_{NE} = <c>$

NHX possible snapshots

B
Before e arrives
$S_n, C_{MN} = <>, C_{EN} = <>$
Moon Possible snapshots

   C

Before f arrives

$S_m, C_{EM} = <>$

$C_{NM} = <>$

List of all possible global snapshots = [ABC]
Question 7. (Solution and Grading by: Faria) **Mandatory question - everyone must answer this question.**

**Assumptions:**
- Each process has a membership list of all other processes. Therefore, it knows which processes are in its own subnet.
- Network is synchronous.

The solution is for nodes to select one leader per subnet using the bully algorithm. The leader in the subnet is the node with the highest lexicographical IP address.

**Modified Bully Algorithm to elect one leader per subnet:**

1. When a process recognizes that its leader is dead,
   a. If it has the highest ID in a subnet, it elects itself as leader and sends Coordinator messages to all processes with lower IPs (in lexicographical order) in its own subnet. Election is complete.
   b. Otherwise,
      i. It initiates election by sending an Election message to only processes that have a higher IP than itself in its own subnet
      ii. If it receives no answer within timeout T1, the process calls itself leader and sends Coordinator message to all lower id processes within the subnet. Election is complete.
      iii. If an answer is received, then there exists some non-faulty process with a higher ID. Thus, the process waits for a Coordinator message. If a Coordinator message is not received after another timeout T2, the process starts a new election run.

2. A process that receives an Election message from a process from its own subnet message replies with OK message, and starts its own leader election protocol (unless it has already done so)

Safety: Safety is guaranteed within a subnet as the algorithm chooses a leader that has the highest ID in the subnet.
Liveness: As the network is synchronous, we can calculate and set appropriate timeouts. When failures stop in a subnet, a leader will eventually be elected.
Question 8. (Solution and Grading by: Mayank)

We replace <T_i, P_i> with <P_i, T_i> in this modified algorithm.

a. This approach **violates safety**. Consider a system with 3 processes, P1, P2 and P3 and a shared resource that no process holds. Assume each process’ Lamport timestamp is currently 0. Let P3 initiate a request <P3, 1> for the resource. When the request from P3 arrives, P1 and P2 will reply since neither of them hold the resource. Let the reply from P2 be delayed for now. Let P1 now make a request <P1, 2> (Lamport timestamp is incremented, but irrelevant to the example) for the resource. Thus, P2 will reply to this request immediately, while P3 will be in a ‘Wanted’ state and will reply to this request (since <P1,2> is less than <P3,1> in the lexicographic ordering). Now let the replies from P2 reach both processes (so both of them have received N-1 replies). Now, we have constructed a scenario where both P1 and P3 are in the ‘Held’ state, which violates mutual exclusion, and hence safety.

Rubric - worth 4 points
If argued that it’s safe : -4
If counterexample is incorrect but mentioned it was unsafe : -2
If concrete counterexample (demonstrating an actual execution) not provided : -2

b. Since any request will be replied to eventually (either a process replies immediately, or replies during exit(), or checks the lexicographic ordering if in ‘Wanted’ state), **liveness is maintained** (it is, however, worthwhile to ponder the value of liveness in a system that doesn’t actually provide mutual exclusion)

Rubric - worth 3 points
If argued it was not live : -3

c. **Causality doesn’t hold**, since we use unique PIDs to determine ordering (which are not causally ordered), not Lamport timestamps. Thus, even if a process Pi makes a request causally before process Pj, if Pj < Pi, Pj’s request might be served first.

Rubric - worth 3 points
1) P3 broadcasts request \(<P3, 1>\)  
2) P1 replies to P3

3) P1 broadcasts request \(<P1, 2>\)  
4) P3 replies to P1 (\(<P1,2>\) is greater than \(<P3, 1>\))
5) P2 replies to both, safety violated
Question 9. (Solution and Grading by: Rui)

Solution:

a) Incorrect. M32 was sent in V11 but delivered in V12.
b) Correct. Every non-faulty process received the same set of multicasts and everything in the view stayed in the view.
c) Correct. Every non-faulty process received the same set of multicasts and everything in the view stayed in the view.
d) Incorrect. P1 and P2 have different multicast set in V11.
e) Correct. Every non-faulty process received the same set of multicasts and everything in the view stayed in the view.
f) Correct. Every non-faulty process received the same set of multicasts and everything in the view stayed in the view.
g) Incorrect. M32 was sent in V11 but received in V12.
h) Incorrect. Two possible explanations (both acceptable): 1) V11 is inconsistent as a view--P1, P2, P3 deliver a different view than P4 (and M32 is delivered in this inconsistent view); 2) From P4’s view, the sender of M32 does not belong to View {P1,P2,P3,P4}
Question 10. (Solution and Grading by: Xiaoyao) **Mandatory question - everyone must answer this question.**

a.

Like the original Chandy-Lamport snapshot algorithm discussed in class, we assume all channels are point to point and FIFO. However, the system is synchronous. Choose a future instance of time $t$ for all processes to take snapshots (choose $t >> \text{max of one way transmission latency from initiator to other processes}$).

For each process $P_i$, set a timer so that when time=$t$ has come:

When time=$t$:

- Records its own state.
- For $j=1$ to $N$ except $i$
  - $P_i$ sends out a Marker message on outgoing channel $C_{ij}$.
- Starts recording incoming messages on each of the incoming channels at $P_i$: $C_{ji}$ (for $j=1$ to $N$ except $i$)

After time=$t$, when a process $P_i$ receives a Marker message on an incoming channel $C_{ki}$ (note that the marker message will only be received after $t$):

- Mark the state of channel $C_{ki}$ as all the messages that have arrived on it since recording was turned on for $C_{ki}$.

The algorithm terminates when all processes have received Marker messages from all other $N-1$ processes. Then, (if needed), a central server collects all these partial state pieces to obtain the full global snapshot.

b.

Proof that the algorithm captures a consistent cut:

The consistent cut here is the time frontier at time=$t$.

Let $e_i$ and $e_j$ be events occurring at $P_i$ and $P_j$, respectively such that

- $E_i \rightarrow e_j$ ($e_i$ happens before $e_j$)
- The snapshot algorithm ensures that
  - If $e_j$ is in the cut, then $e_i$ is also in the cut.
- That is: if $e_j \rightarrow <P_j \text{ records its state}>$, then
  - It must be true that $e_i \rightarrow <P_i \text{ records its state}>$. 
• Since this question assumes a synchronous system, and we know that <P1 records its state> and <Pj records its state> happens at exactly time=t given the algorithm described above.

• So we can convert the statement to prove to be: if ej -> t, then
  ○ It must be true that ei -> t.

• Proof:
  ○ Given ei -> ej, and ej -> t
  ○ We have ei -> ej -> t
  ○ Then ei -> t is always true.

c.
All answers are accepted.