Question 1:

Consider the quorum replication protocol shown in class.

a) Suppose in the write protocol, we eliminate lines 1 and 2 and instead use as timestamp the output of an atomic clock (plus process id to break symmetry). Would the protocol work correctly? If your answer is yes, explain why. Otherwise, give a scenario in which some properties are violated.

Answer: Yes. The use of atomic clocks ensures that the timestamp obtained by any write operation W1 is larger than the timestamp of any other write operation that terminated before the beginning of W1.

b) Repeat (a) when the timestamp is the output of the network time protocol (NTP).

Answer: No. Due to network latencies (at all level of the network stack), the timestamps obtained by NTP might not reflect real time. Hence, a write W1 that started after another write W2 might get a smaller timestamp and therefore its value will not be returned by following reads.

c) What properties would be violated if we skip Lines 3&4 of the read protocol? Give an example.

Answer: Suppose we have a write operation W(X,1) that takes a long time to complete due to network latencies. The write updates one server S1. A read operation R1 to X starts concurrently and the read quorum that returns first includes S1. Hence, R1 returns 1. After R1 completes but before W1 completes, another read R2 to X starts. The read quorum that returns first does not include S1, and thus R2 returns an older value. The property violated is that a read returns an older value than a previous read.

Question 2:

Suppose we have access to a consensus service. What is the difference between its usage in implementing a replicated state machine vs. its usage in implementing a hot-standby solution.

Answer: In active replication, consensus is used to decide on which clients' requests/operations should be executed and in what order (i.e., to ensure total ordering). In hot-standby it is used to agree on which operations (state updates) have been performed. There is no need to agree on the a-priori order since only a single node executes operations.
Question 3:

a) Explain how the properties of Ω failure detectors help the Paxos protocol solve the consensus problem?
   Answer: The FD ensures that eventually all processes believe in the same single live leader. Hence, no process will be stuck forever waiting for messages from a crashed process and eventually, if consensus was not reached before, there will be a single leader trusted by everyone. Once this happens, this leader will be able to complete a successful round of the protocol and lead to everyone deciding. Notice that without this, multiple concurrent leaders can continuously sabotage each other's attempts to reach consensus.

b) Why is the optimization in which each node sends its accepted message to all other nodes allowed?
   Answer: When a coordinator receives an accepted message from a quorum, it means that there is a quorum of acceptors that adopted its value. Hence, this value is already "locked" in the system. Any possible following attempt to reach consensus will find one of these nodes, due to quorum intersection, and thus no other value will be sent in a following accept message by a coordinator. The commit message simply informs all other nodes about this fact. Hence, when the optimization is applied, each node can independently discover if there is such a quorum and will not decide otherwise. Hence, the safety of the protocol is preserved and assuming a quorum never crashes, then also its liveness.
**Question 4:**

Provide a pseudo-code for a simple ZooKeeper based decentralized compare&swap (CAS) register service. Describe what assumptions you make and what are the membership properties provided by your service.

*Note:* A `CAS(obj, old, new)` operation accepts an object `obj`, an expected value `old` and atomically stores the value `new` in `obj` if its value just prior to invoking the CAS was `old`.

**Answer:** Solving this problem requires running a mutual exclusion algorithm. This is obtained with essentially the same mechanism as in the leader election protocol shown in class. That is:

```plaintext
cas(obj, old, new) {
    myseq:=Create(OBJECTS/obj/n_, SEQUENCE | EPHEMERAL; (
    return tryCAS(obj, old, new);
}
```

Upon receiving a notification of znode deletion:

```plaintext
tryCAS(obj, old, new);
```

```plaintext
tryCAS(obj, old, new) {
    C:=getChildren(OBJECTS/obj);
    if myseq is the smallest znode in C, then
        oldval:=GetData(obj);
        if (oldval == old) then
            SetData(obj, new);
            return oldval;
        endif
        let j be the lowest index of a znode in C; set watcher on ELECTION/n_j;
    endif
    else
        let j be the lowest index of a znode in C; set watcher on ELECTION/n_j;
    endif
}
```