Problem 1

Consider a group of distributed processors P1, P2, P3, and P4 that use the Ricarti-Agrawala algorithm for ensuring mutual exclusion. Assume that P4 is currently in the critical section and there is no other node in the WANTED state. Now consider requests from P1 and P2 (in that order) to enter the Critical Section.

1. Show the state and queue entries at each processor.

2. Now, P4 exits the CS and informs all relevant nodes that CS is released. Show the state and queue entries at each processor, at this stage.

Answer:

1)
P1: wanted, Queue1 = {P2}
P2: wanted, Queue2 = {}
P3: released, Queue3 = {}
P4: held, Queue4 = {P1, P2}

2)
P1: held, Queue1 = {P2}
P2: wanted, Queue2 = {}
P3: released, Queue3 = {}
P4: released, Queue4 = {}

Problem 2

In a certain system, each process typically uses a critical section many times before another process requires it. Explain why Ricart and Agrawala's multicast-based mutual exclusion algorithm is inefficient for this case, and describe how to improve its performance. Does your adaptation satisfy liveness condition in $ME2$? (Problem 15.7 in the $5th$ edition)

Answer: Ricart and Agrawala's algorithm multicast-based algorithm multicasts requests, and requires reply from all other processes before entering a critical section, which is expensive when one process needs several access to the critical section before any other process requests for it.

One possible solution is to change the state from *HELD* to *TEMP* instead of *RELEASE* when the process is done with the critical section. If it needs access again, it can change the state from TEMP to HELD without sending multicast message.

To satisfy ME2 the process has to change state TEMP to RELEASE if there is any request for the critical section, i.e., its queue is not empty.

Problem 3

Is leader election possible in a synchronouse ring in which all but one processor have the same identifier? Either give an algorithm or prove an impossibility result.

Answer: Yes, it is possible, since one of the processors has an id which is different from every other processors id. We can propose different algorithms to choose the process with different id as the leader. One possible algorithm is algorithm 1. We assume no failures happen.

1 Leader election
```
To start election
    send (election, my ID) to the left and right processors in the ring
When receiving message (election.id) from both left and right neighbors
if my ID is not equal to any of my two neighbors id then
    I am selected as the leader
    send (elected.my ID) to my right neighbor
end if
(elected.id) is forwarded clockwise in the ring until it comes back to leader (to inform every processor
    which processor is the leader)
```

Problem 4

Suggest how to adapt the bully algorithm to deal with temporary network partitions (slow communication) and slow processes. (Problem 15.9 in the 5*th* edition)

Answer: In case of network partitions, subgroups will be formed. Each subgroup can run Bully algorithm and elect a coordinator with the highest ID in the subgroup. When the network heals, the subgroups should merge and elect one coordinator with the highest ID.

Problem 5

Modify the basic ring-based leader election algorithm to elect 2 leaders (two processes with the highest IDs).

Answer: To start an election, a process sends a message $< election >$ with its ID appended. Each node that receives this message appends its ID to it. Once the election message reaches the initiator after going through the circle, the top two nodes (nodes with the highest and second highest IDs) are selected, and a message $< choose : highest1, highest2 >$ is sent with the initiator's ID appended to it. Each node that receives this message appends its ID again. When this message reaches the initiator, if the appended ID list contains the top two nodes, then the election is successful and ends. Otherwise, a new election is initiated.

Problem 6

Consider a synchronous system in which processors fail only by crash failure, with the additional constraint that the crash is always clean, that is, in a round, a processor either sends all its messages or none.

1. For this system, design an algorithm that solves the consensus problem in the least possible number of rounds.
2. Explain why your algorithm is correct.

2 Consensus
```
Initially $Values = \{v_i\}$
for one round do
    multicast $v_i$ (processor's value)
    for each $v_j$ received do
        $Values = Values \cup v_j$
    end for
end for
$y_p = \min(Values)$
```

Answer:

Problem 6 continued on next page...

1) Algorithm 2 solves the consensus problem in one round.

2) Since the crash is clean, after the first round, if a process i gets a value from process j, every other process k ($k \neq i$) gets value of process j as well. Furthermore, if a process i does not get anything from process j, no other process k ($k \neq i$) has value of process j. So, at the end of one round, all fault-free processes have received the same set of values. (There is no need for more rounds to exchange values among fault-free processes.)