1. Consider the following modification of the Bully algorithm: The initiating node (which we assume does not fail) sends an Election message only to the process with the highest id. If it does not get a response after a timeout, it then sends an Election message to the process with the second highest id. If after another timeout it gets no response, it tries the third highest id, and so on. If no higher numbered processes respond, it sends a Coordinator message to all lower-numbered processes.

```python
class ModifiedBully:
    def run_election(self, failed_leader = None):
        """
        'failed_leader': previous leader that has failed, to avoid
        sending a message to it
        """
        self.leader = None
        # process pid's in reverse order. self.group
        # is a list of all processes in the group
        for pid in sorted(self.group, reverse=True):
            if pid == failed_leader:
                continue  # skip previous leader
            elif pid == self.pid:
                # I am the new leader
                self.leader = self.pid
                for pid2 in self.group:
                    if pid2 < self.pid:
                        unicast(pid2, "Coordinator")
                        break
                unicast(pid, "Election")
                sleep(TIMEOUT)
                if self.leader is not None:
                    break

        def receive_message(self, message):
            if message.contents == "Coordinator":
                self.leader = message.sender
            elif message.contents == "Election":
                # fill in the rest
```

(a) (2 points) Complete the receive_message function (pseudocode OK).

For the following parts, consider a distributed system of 8 processes that uses the modified Bully algorithm for leader election (including your solution to part (a)). The processes are called \{P_1, \ldots, P_8\} with \(P_i\) having PID \(i\). Initially all 8 processes are alive and \(P_8\) is the leader. Then \(P_8\) fails, \(P_4\) detects this, and initiates the election. Assume one-way message transmission time is \(T\), and timeout is set using the knowledge of \(T\).

(b) (2 points) If no other node fails during the election run, how many total messages will be sent by all processes in this election run?

(c) (2 points) If no other node fails during the election run, how long will it take for the election to finish?
(d) (2 points) Now assume that immediately after $P_4$ detects $P_8$’s failure and initiates the election, $P_7$ fails. How many total messages will be sent by all processes in this election run?

(e) (2 points) For the above scenario (where $P_7$ fails right after $P_4$ initiates election upon detecting $P_8$’s failure), how long will it take for the election to finish?

(f) (4 points) What are the best- and worst-case turnaround times for this election algorithm?

2. Consider a system of $N$ processes, $\{P_1, \ldots, P_N\}$ arranged in a ring. Each process can only communicate to its ring successor; i.e., $P_i$ can only send a message to $P_{i+1}$, and $P_N$ can only send a message to $P_1$. Furthermore, each message is a pair of two signed 16-bit integers, i.e., they can each take values from $-32768$ to $32767$. Assume that $N \leq 1000$. Assume that that there are no failures, and the communication channel delivers all messages correctly and exactly once.

(a) (6 points) In this problem each process has a 16-bit signed integer input $x_k \ (-32768 \leq x_k \leq 32767)$, and an output variable $y_k$, initialized to None (meaning undecided). A consensus algorithm is designed to calculate the maximum of all the input variables. In other words, the safety condition is that, at any point $y_k$ is either None or $y_k = \max_{i \in \{1, \ldots, N\}} x_i$.

We will adapt the (first version of the) ring-based election algorithm for this problem.

- A process $P_i$ initiates the algorithm by sending $(0, x_i)$ to its ring successor. (The 0 indicates this is a proposal message).
- When a process $P_j$ receives $(0, x)$ from its ring predecessor:
  - if $x > x_j$, it forwards $(0, x)$ to its successor.
  - if $x < x_j$, it sends replaces $x$ with $x_j$ and sends $(0, x_j)$ to its successor.
  - if $x = x_j$, it concludes that $x = x_j$ is the minimum value, sets $y_j = x$ and sends $(1, x)$ to its successor, (1 indicating it is a decided message.)
- When a process $P_j$ receives $(1, x)$ and its $y_j$ is None, it sets $y_j = x$ and forwards $(1, x)$ to its successor. If $y_j$ is not None it ignores any received messages.

Note that multiple processes may initiate the algorithm simultaneously. Here is a Python implementation:

```python
PROPOSAL = 0
DECIDED = 1
X_K = # my input
Y_K = None

def start_consensus():
    # send forwards two integers to the successor in the ring
    send(PROPOSAL, X_K)

def receive_message(a,b):
    if Y_K is not None:
        continue # ignore messages after decided state
    if a == PROPOSAL:
        if b > X_K: # forward
            send(PROPOSAL, b)
        elif b < X_K: # replace b with X_K
            send(PROPOSAL, X_K)
        else: # b == X_K
            Y_K = b
            send(DECIDED, Y_K)
    elif a == DECIDED:
        # set variable, forward decision
        Y_K = b
        send(DECIDED, b)
```

Does the algorithm described above guarantee safety condition for the problem? If yes, prove how. If not, (i) describe a scenario where safety is violated, and (ii) suggest modifications to the algorithm
that would guarantee the safety condition. You do not need to submit code but you should have enough detail in your modified algorithm for the grader to be able to implement the modification.

(b) (4 points) In this problem, the inputs $x_k$ are all either 0 or 1. In this case, the output variable should be set to represent the majority value. In other words, if $y_k$ is not None, it must be 1 if a majority of processes have input 1, 0 if a majority of processes have input 0, and 2 if there is a tie. Describe an algorithm for solving this problem (using either Python or pseudocode). Again assume that multiple processes may initiate your algorithm simultaneously.

3. Consider the following algorithm:

- A process, say $P_i$ initiates consensus by multicasting a Query message to the group
- Each process $P_j$ unicasts a reply back to $P_i$ with a message $\text{Value}(x_j)$ that includes its input
- After receiving replies from everyone or a timeout, it computes the minimum of all received values (including its own) $y_i = \min x_j$ and multicasts $\text{Decision}(y_i)$ to the group.

Assume that we operate in a synchronous system with a maximum one-way delay of $T$ for a unicast message transmission, including any processing time. Assume that unicast channels are reliable.

(a) (3 points) For the multicast of the Query message, should R-multicast or B-multicast be used, and why?

(b) (2 points) What should the timeout value be?

(c) (3 points) For the multicast of the Decision message, should R-multicast or B-multicast be used, and why?

(d) (2 points) Assuming no failures, how long will the algorithm take to complete?

(e) (3 points) This algorithm can lead to a safety failure while trying to achieve consensus. Explain how it could happen. (For this part, you may need to consider the possibility of process failures.)
4. Consider a system implementing Paxos with two proposers, \( P_1 \) and \( P_2 \), and three acceptors, \( A_1, A_2, A_3 \). Assume that the input value for \( P_1 \) is 1 and the input value for \( P_2 \) is 2. Answer the following sub-questions, each of which is unrelated to the other two sub-questions.

Refer to the figure above. It shows the Prepare messages sent by \( P_1 \) and \( P_2 \). The responses (if any) are not shown. Assume no other proposals are initiated.

i. (1 point) Which processes will reply back to \( P_1 \)’s Prepare messages?

ii. (1 point) Which processes will reply back to \( P_2 \)’s Prepare messages?

iii. (2 points) Assuming each Promise message take exactly 2 time units, at what time will \( P_1 \) send Accept messages? At what time will \( P_2 \) send Accept messages?

iv. (2 points) Assuming each Accept message takes at least 1 time units and, as above, the Promise messages take exactly 2 time units, is it possible for for proposal \#1 to be accepted by a majority of acceptors? If no, explain why not. If yes, explain how.
Refer to the figure above. P1 sends Prepare messages at time 1, receives a Promise from all acceptors, and sends Accept messages at time 6. Meanwhile, P2 sends a prepare for proposal #2 at time 7.

i. (1 point) Which processes will reply to P2’s Prepare message?

ii. (1 point) Which processes will accept P1’s proposal?

iii. (1 point) Assuming each Promise response for P2’s Prepare message takes exactly 2 units, when will P2 send Accept messages?

iv. (1 point) Assuming P2 sends messages at the time stated in the previous parts, which processes will accept P2’s proposal?

v. (1 point) What will be the value in P2’s proposal?

vi. (3 points) Come up with a scenario where the consensus value changes by switching the arrival time of one message. Complete the diagram showing the timing of the promise messages from the acceptors to P2 and the accept messages from P2 back to the acceptors. (For this subpart any messages not shown on the original diagram must take at least one time unit, but have no other constraints.)
5. Consider a system of 5 processes, \( \{P1, P2, P3, P4, P5\} \) implementing Raft’s algorithm for leader election. The one-way delay between each process is specified in the figure below. Assume no processing time. Each of the parts below are independent scenarios.

(a) (1 point) Which server is the most likely to be elected leader in any term? No justification is necessary.

(b) (4 points) Suppose \( P1 \) is the leader in term 1. It sends its last heartbeat at time 0 and then fails. Suppose, upon receiving the heartbeat, \( P2, P3, P4, \) and \( P5 \) set their timeout values to 50, 75, 100, and 150 ms, respectively. Who will each process vote for as the leader in term 2?

(c) (2 points) Suppose now that \( P2, P3, P4, P5 \) set their timeout values to 150, 100, 75, and 50 ms, respectively. Who will each process vote for as leader in term 2?

(d) (2 points) Consider the logs of the processes, with each number \( n \) denoting an event logged in term \( n \). Order the servers from least up-to-date to most up-to-date:

\[
\begin{align*}
P_1 &: 1, 1, 1, 2, 2, 3, 3, 6 \\
P_2 &: 1, 1, 1, 2, 2, 3, 3, 4 \\
P_3 &: 1, 1, 2, 2, 3, 3, 4, 4 \\
P_4 &: 1, 1, 1, 2, 2, 3, 6, 6 \\
P_5 &: 1, 1, 1, 2, 2, 3, 3, 4, 5
\end{align*}
\]