III Raft and Lab 2

Ben Bitdiddle decides that, when the last log term of a Raft candidate and a follower are equal, it is safer if the follower only votes for the candidate if the candidate's log is *longer* than the follower's own, rather than if it is at least as long.

He therefore modifies his `RequestVote` RPC handler to respond with `voteGranted` set to `true` only if either *(a)* the candidate has a higher last log entry term, or *(b)* the last log entry's term is equal to the follower's and the candidate has a longer log.

5. **[8 points]:** Explain why Ben's modification will break Raft.

Answer: If all peers have identical logs, they will not be able to elect a leader.

Alyssa P. Hacker's Lab 2 Raft implementation has a long-running thread that runs periodic tasks:

```
func (rf *Raft) mainThread() {
    for {
        rf.mu.Lock()
        state := rf.state
        rf.mu.Unlock()

        if state == Leader {
            if a heartbeat is needed {
                rf.sendAll()
            }
        } else if state == Follower {
            ...
        } else {
            ...
        }
        // pause...
    }
}

func (rf *Raft) sendAll() {
    rf.mu.Lock()
    defer rf.mu.Unlock()

    for i := 0; i < len(rf.peers); i++ {
        if i == rf.me {
            continue
        }
        var args AppendEntriesArgs
        args.Term = rf.currentTerm
        // ...
        go func(i int) {
            var reply AppendEntriesReply
            ok := rf.sendAppendEntries(i, &args, &reply)
            rf.mu.Lock()
            // handle the response...
            rf.mu.Unlock()
        }(i)
    }
}
```

Assume that code that is not shown is correct.

6. [8 points]: Alyssa's code has a bug. Describe a sequence of events that demonstrates a violation of Figure 3's State Machine Safety property, caused by the code above.

Answer: Alyssa's code releases the lock between checking whether the peer is leader and constructing the RPC arguments. Suppose peer P1 is leader for term 11, and mainThread() calls sendAll(). While the lock isn't held, P1 receives an RPC with term 12 (because peer P2 is now the leader), so P1's RPC handler increases rf.currentTerm to 12 and becomes a follower. P1's sendAll() will proceed to send AppendEntries RPC, with Term set to 12. That's not correct, because P1 was leader for term 11, not term 12. If P1 includes uncommitted log entries in the RPCs, P1's log entries may replace different log entries sent by P2. If P2 then advances the commit index, some peers may execute P1's log entries, and others P2's log entries. That would violate State Machine Safety.