Q19)

In the following, ACK[N] means that all packets with sequence number *less* than N have been received.

1. The sender sends DATA[0], DATA[1], DATA[2]. All arrive.

2. The receiver sends ACK[3] in response, but this is slow. The receive window is now DATA[3]..DATA[5].

3. The sender times out and resends DATA[0], DATA[1], DATA[2]. For convenience, assume DATA[1] and DATA[2] are lost. The receiver accepts DATA[0] as DATA[5], because they have the same transmitted sequence number.

4. The sender finally receives ACK[3], and now sends DATA[3]-DATA[5]. The receiver, however, believes DATA[5] has already been received, when DATA[0] arrived, above, and throws DATA[5] away as a "duplicate". The protocol now continues to proceed normally, with one bad block in the received stream.

Q20)

We first note that data below the sending window (that is, <LAR) is never sent again, and hence – because out-of-order arrival is disallowed – if DATA[N] arrives at the receiver then nothing at or before DATA[N-3] can arrive later. Similarly, for ACKs, if ACK[N] arrives then (because ACKs are cumulative) no ACK before ACK[N] can arrive later. As before, we let ACK[N] denote the acknowledgment of all data packets less than N.

(a) If DATA[6] is in the receive window, then the earliest that window can be is DATA[4]-DATA[6]. This in turn implies ACK[4] was sent, and thus that DATA[1]-DATA[3] were received, and thus that DATA[0], by our initial remark, can no longer arrive.

(b) If ACK[6] may be sent, then the lowest the sending window can be is DATA[3]..DATA[5]. This means that ACK[3] must have been received. Once an ACK is received, no smaller ACK can ever be received later.

Q21)

(a) The smallest working value for MaxSeqNum is 8. It suffices to show that if DATA[8] is in the receive window, then DATA[0] can no longer arrive at the receiver. We have that DATA[8] in receive window
$\Rightarrow$ the earliest possible receive window is DATA[6]..DATA[8]
$\Rightarrow$ ACK[6] has been received
$\Rightarrow$ DATA[5] was delivered.
But because SWS=5, all DATA[0]'s sent were sent before DATA[5]
$\Rightarrow$ by the no-out-of-order arrival hypothesis, DATA[0] can no longer arrive.

(b) We show that if MaxSeqNum=7, then the receiver can be expecting DATA[7] and an old DATA[0] can still arrive. Because 7 and 0 are indistinguishable mod MaxSeqNum, the receiver cannot tell which actually arrived. We follow the strategy of Exercise 27.

1. Sender sends DATA[0]...DATA[4]. All arrive.
2. Receiver sends ACK[5] in response, but it is slow. The receive window is now DATA[5]..DATA[7].
3. Sender times out and retransmits DATA[0]. The receiver accepts it as DATA[7].

(c) MaxSeqNum $\geq$ SWS + RWS.

Q24)

T=0	A sends frames 1-4. Frame[1] starts across the R–B link. Frame[2] is in R's queue; *frames 3 & 4 are lost.*
T=1	Frame[1] arrives at B; ACK[1] starts back; Frame[2] leaves R.
T=2	ACK[1] arrives at R and then A; A sends Frame[5] to R. R immediately begins forwarding it to B. Frame[2] arrives at B; B sends ACK[2] to R.
T=3	ACK[2] arrives at R and then A; A sends Frame[6] to R. R immediately begins forwarding it to B. Frame[5] (not 3) arrives at B; B sends no ACK.
T=4	Frame[6] arrives at B; again, B sends no ACK.
T=5	A TIMES OUT, and retransmits frames 3 and 4. R begins forwarding Frame[3] immediately, and enqueues 4.

T=6	Frame[3] arrives at B and ACK[3] begins its way back. R begins forwarding Frame[4].
T=7	Frame[4] arrives at B and ACK[6] begins its way back. ACK[3] reaches A and A then sends Frame[7]. R begins forwarding Frame[7].